# Free energy after convergence ults

# Assignment 6

## Probabilistic and Unsupervised Learning

## Yuan Zhang

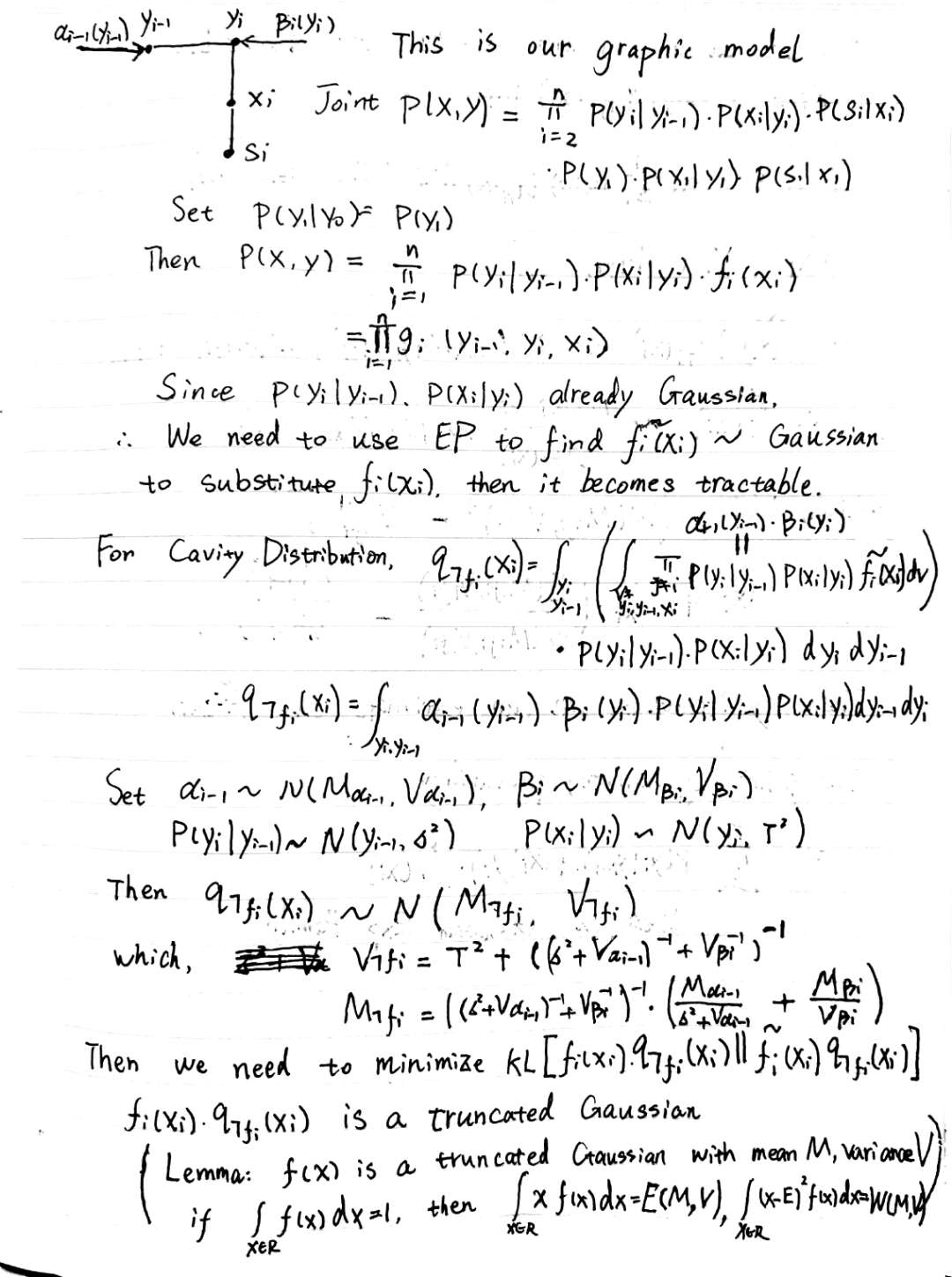
## SN: 17044633

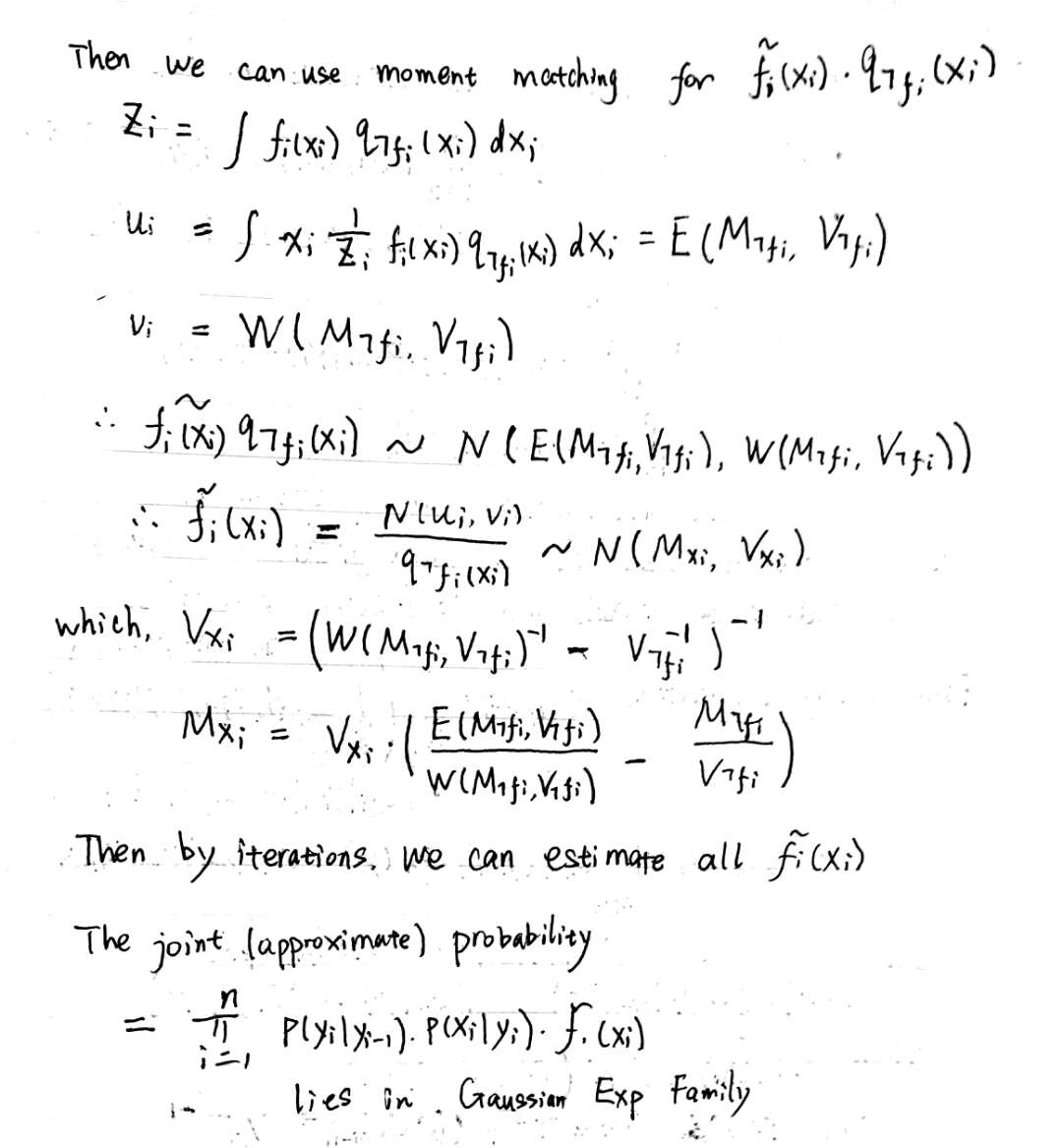
## [ucaby02@ucl.ac.uk](mailto:ucaby02@ucl.ac.uk)

## Jan 10th, 2018

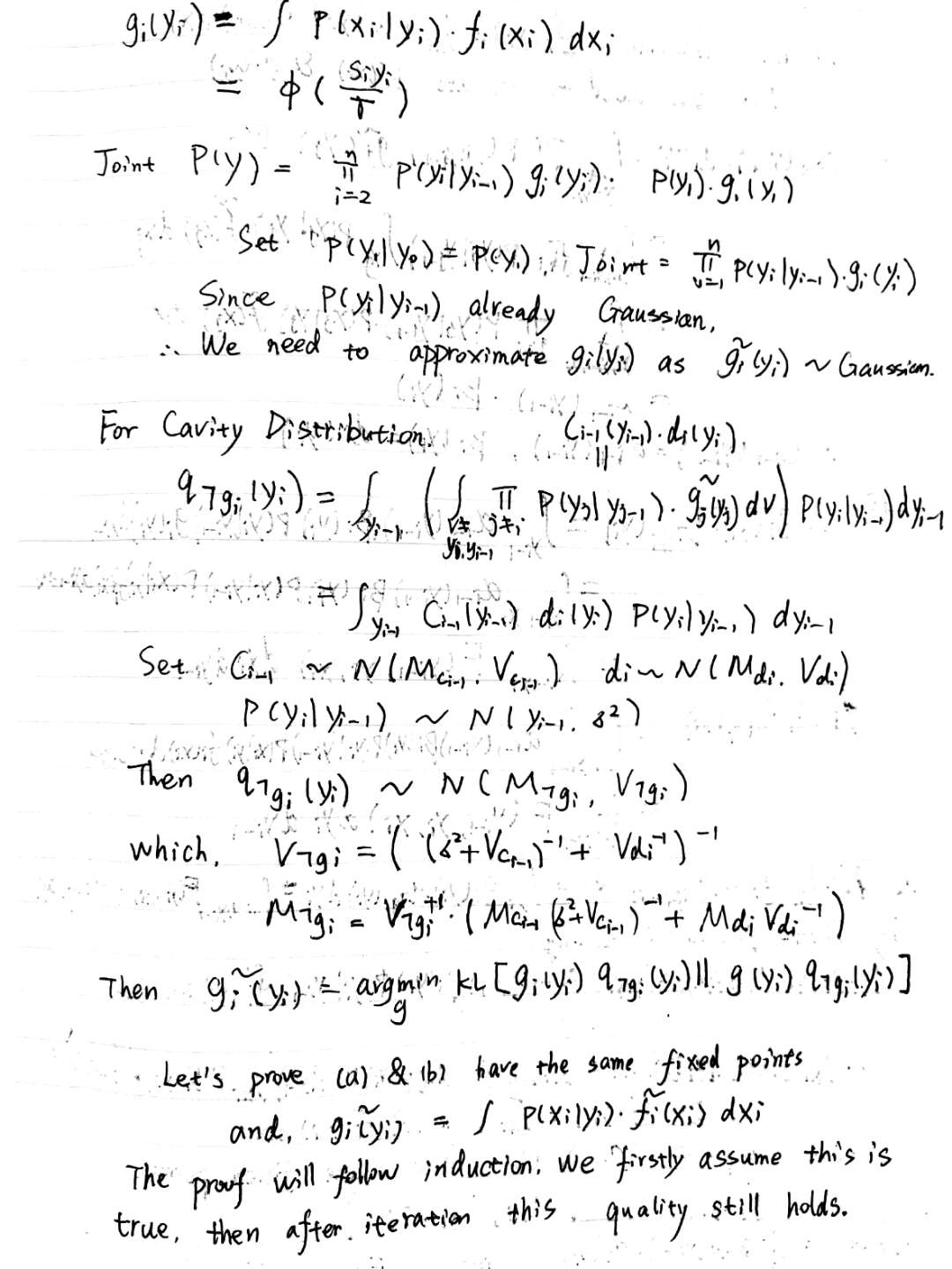
## EP for sign constraints

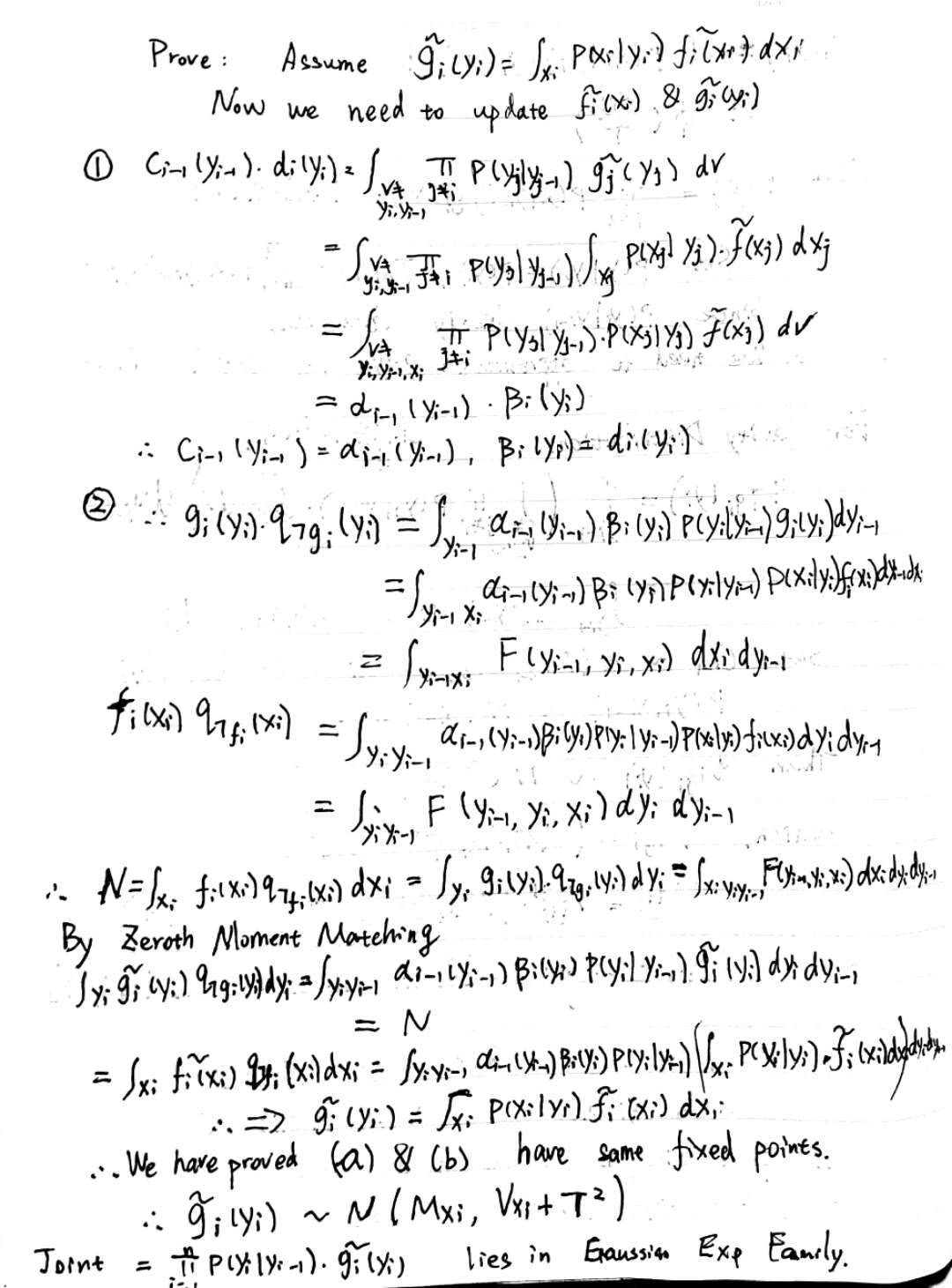
#### Add x\_i as latent





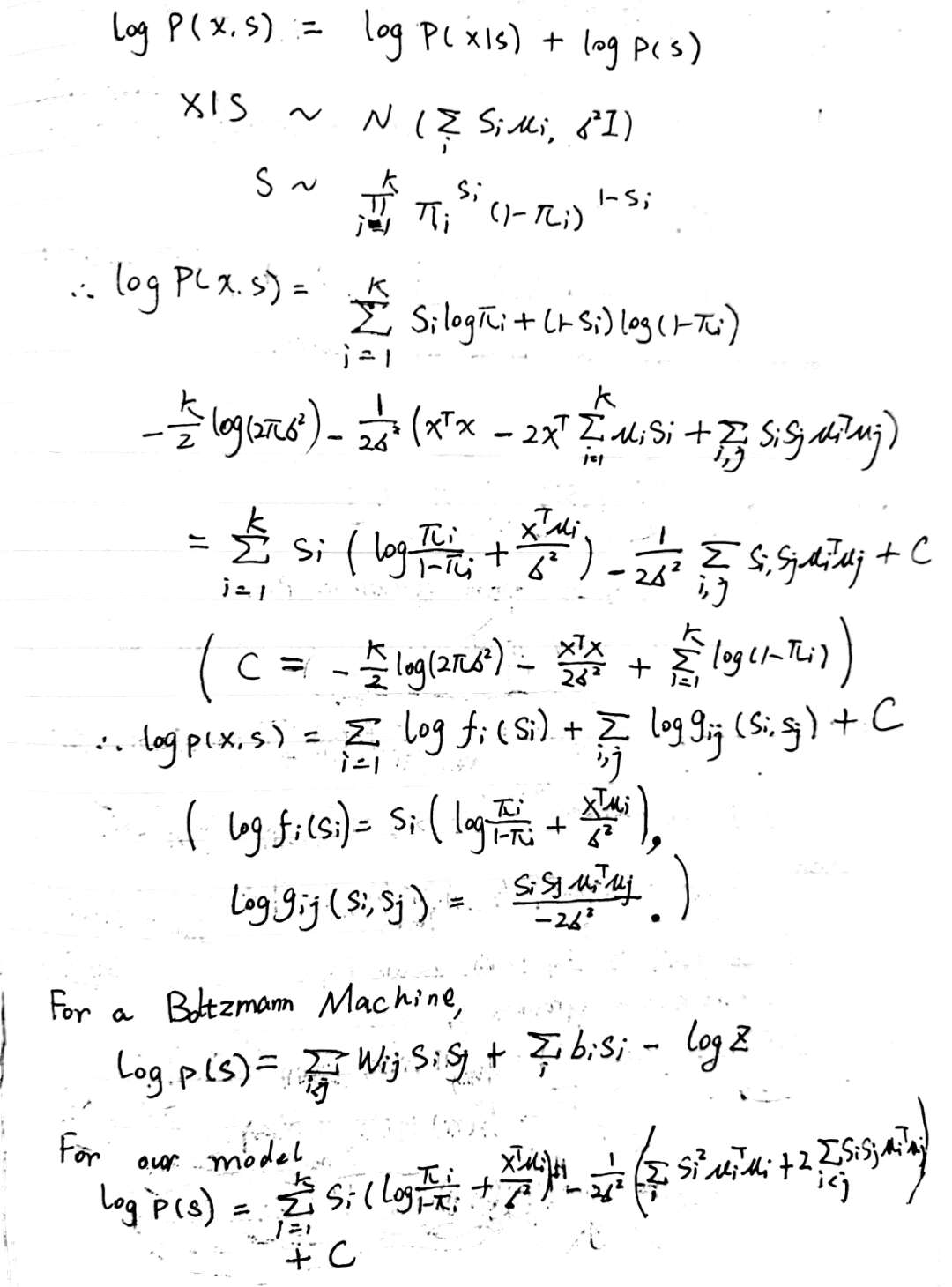
#### Integral x\_i





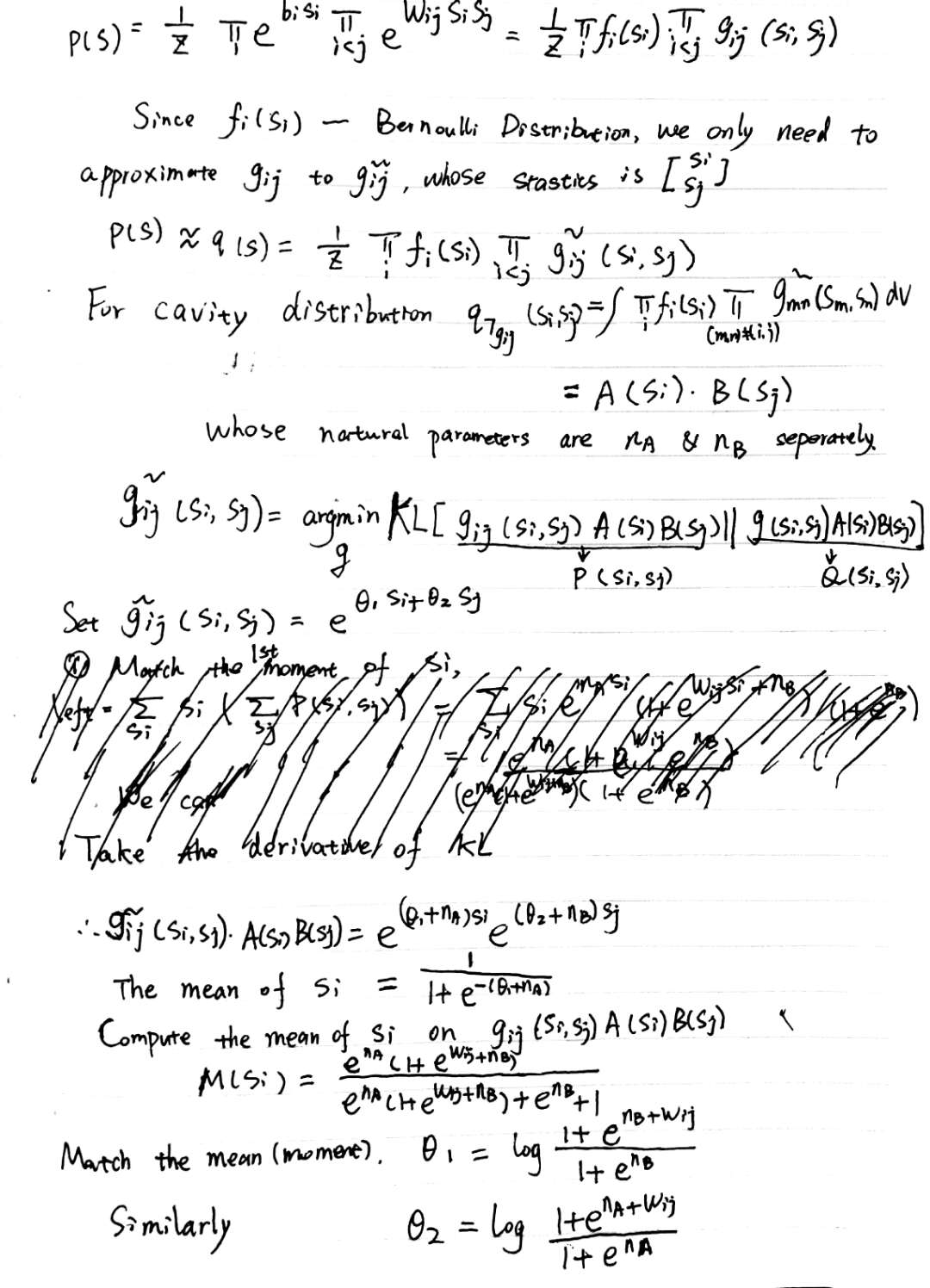
## EP for the binary factor model

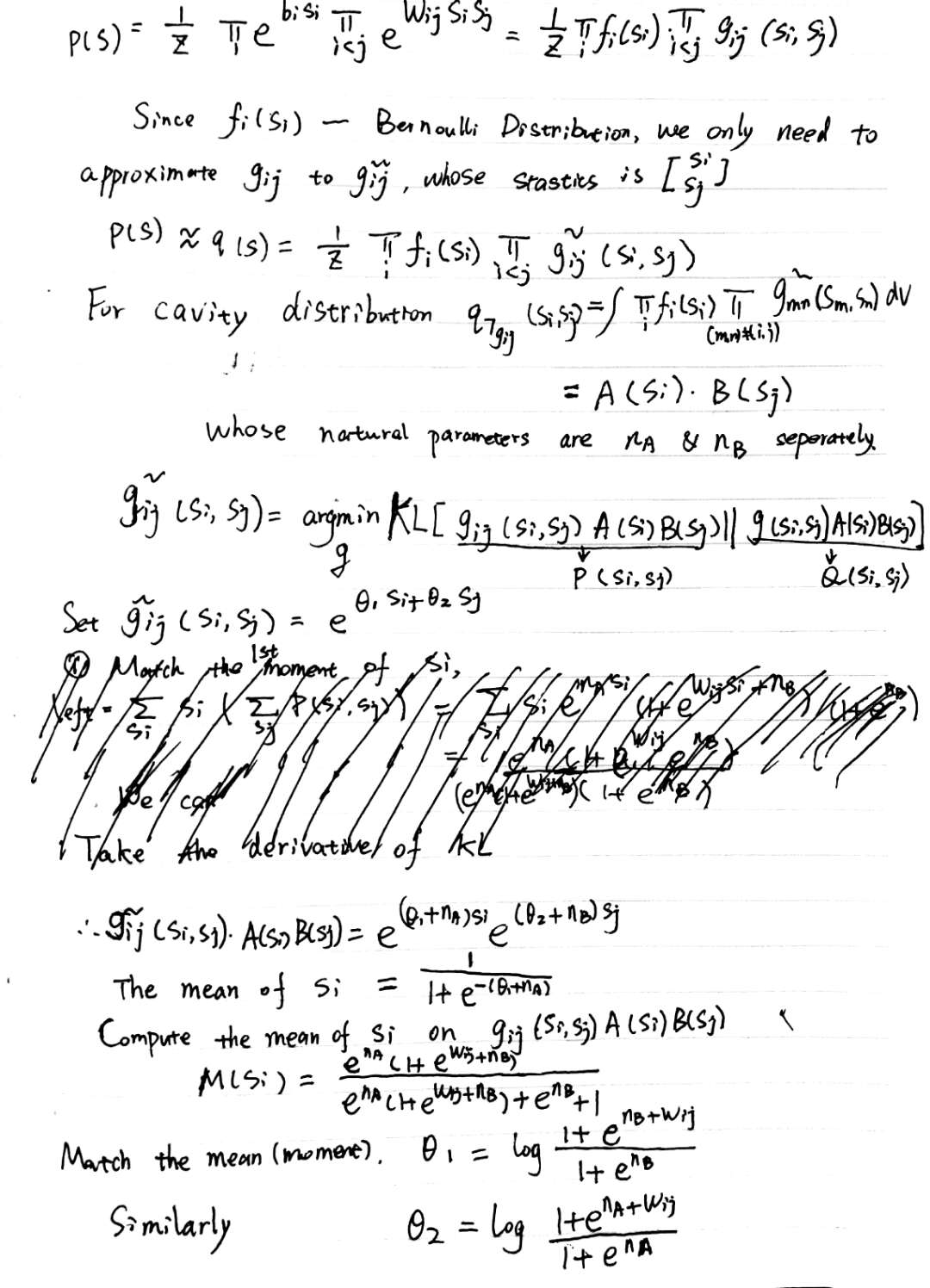
#### (a) Derive the log-joint



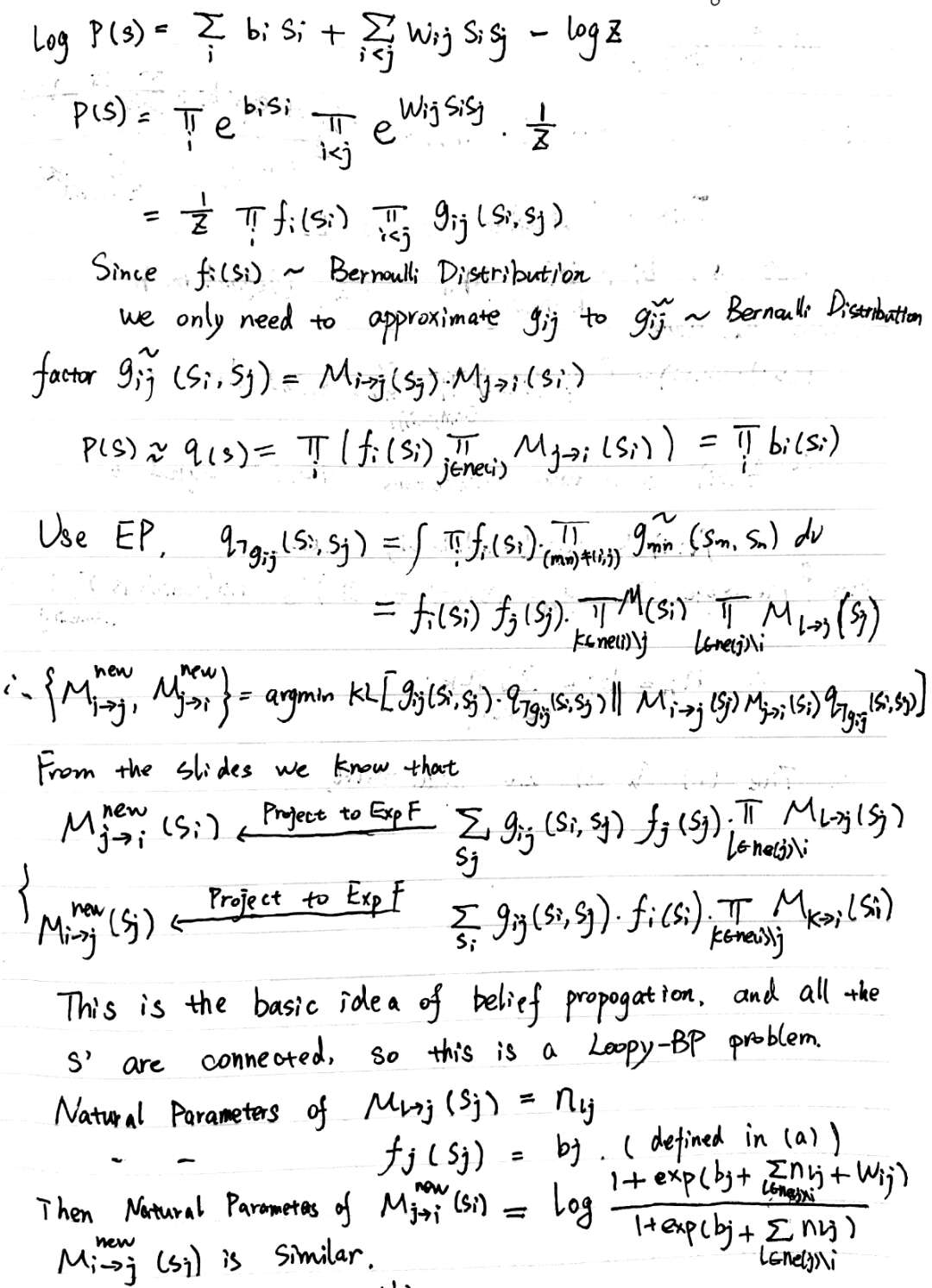
#### Macintosh HD:Users:yuanzhang:Desktop:未命名文件夹:WechatIMG237.jpeg

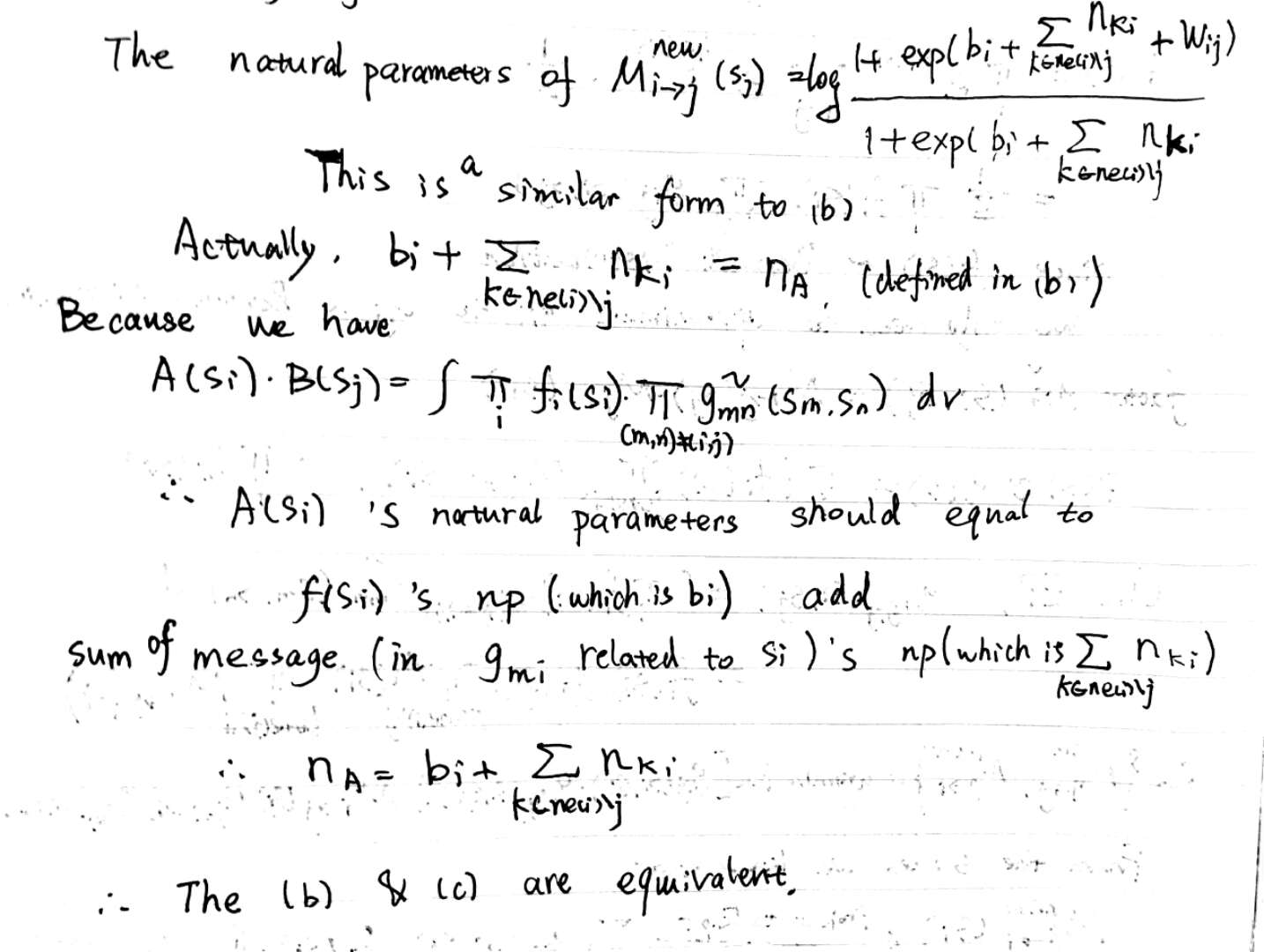
#### (b) EP message passing



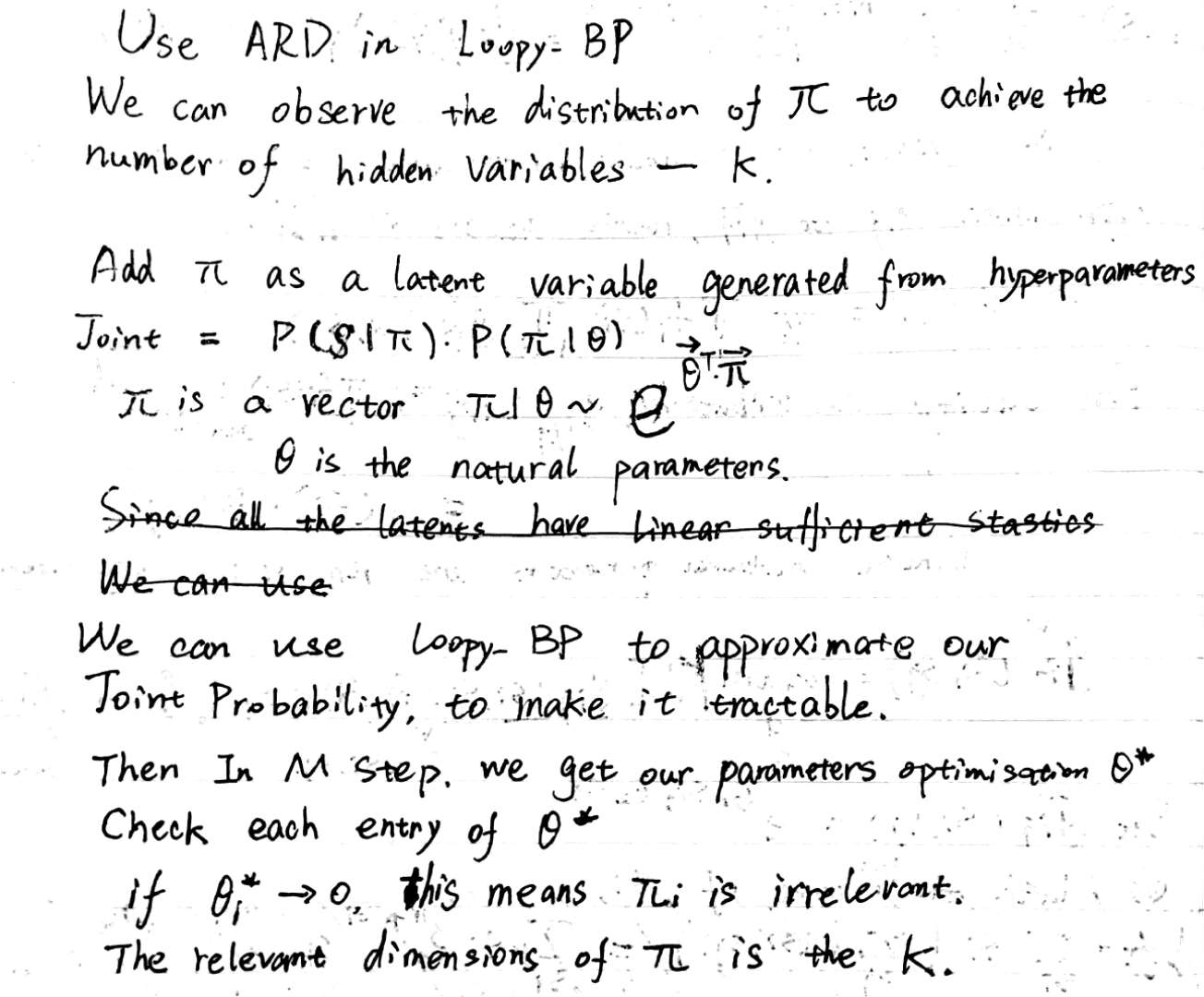


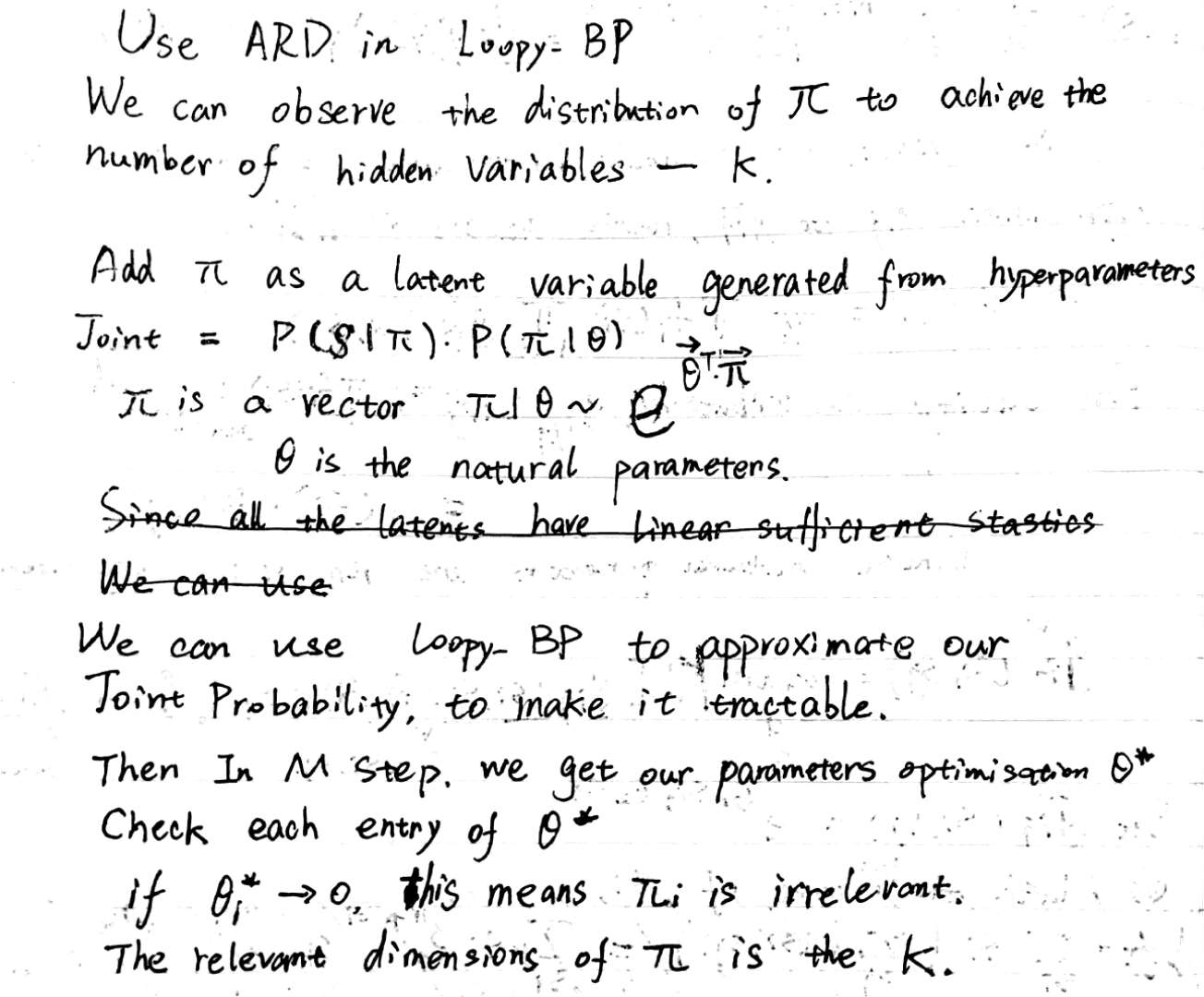
#### (c) Loopy-BP

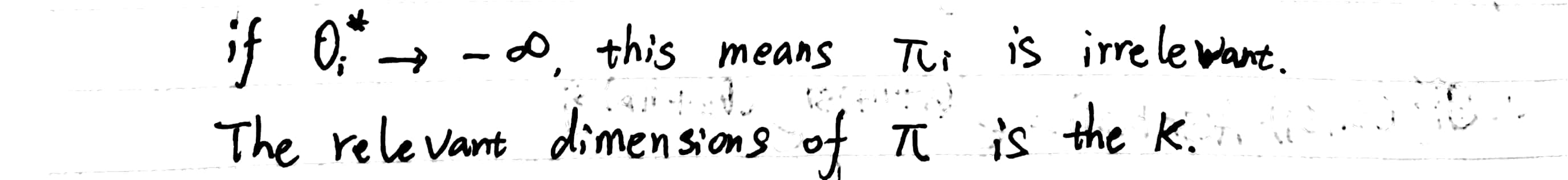




#### (d) Bayes Method for selecting K







## 3. Implement loopy-BP

*Illustration: We can use loopy-BP as an E step to substitute mean field algorithm. Keep the M step unchanged.*

*Code Parts: Q3.m LearnBinFactors.m* *LBP.m initial\_para.m*

%% Q3.m

K = 8;

iterations = 100;

[mu, sigma, pie,lambda,F\_his] = LearnBinFactors(Y,K,iterations);

% plot Free Energy

figure,plot(F\_his);

% plot mu

mu = mu';

figure,

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

% plot mu after image enhancement

mu\_eh = enhance(mu);

figure,

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

colormap gray;

for k=1:K

subplot(2,4,k);

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

axis off;

axis equal;

end

% LearnBinFactors.m

function [mu sigma, pie,lambda,F\_his] = LearnBinFactors(X,K,iterations)

[N,D] = size(X);

[Message0,mu,sigma,pie] = initial\_para(X,N,D,K);

F\_last = -inf;

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

F\_his = zeros(iterations,1);

for j=1:iterations

% Implement E step

[lambda,Message,~] = LBP(X,mu,sigma,pie,Message0,50);

% 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);

F = FreeEnergy(X,mu,sigma,pie,lambda);

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

F\_his(j) = F;

% check convergence

if abs(F-F\_last) < thres

break;

end

F\_last = F;

Message0 = Message;

end

end

% LBP.m

function [lambda,Message,F\_last] = LBP(X,mu,sigma,pie,Message0,maxsteps)

% Variational Estep for our models.

%

% Inputs:

% X: N ¡ÁD data matrix

% mu: D ¡Á K matrix of means

% pie: 1 ¡Á K vector of priors on s

% Message0: N\*K\*K initial natural parameters of Message passing.

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

%

% Outputs:

% lambda: N¡ÁK distributions on latent variables

% F: lower bound on the likelihood

% Message: N\*K\*K update natural parameters of Message passing.

[N,D] = size(X);

[~,K,~] = size(Message0);

% set new Mnatural parametes.

Message = zeros(size(Message0));

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

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

% set natural parameters of fi(si)

f = zeros(N,K);

for n=1:N

f(n,:) = log(pie./(1-pie)) + X(n,:)\*mu/(sigma^2)-self\_m/(2\*sigma^2);

end

F\_last = -inf;

thres = 10^-6; % convergence criterion to stop iteration

for run= 1:maxsteps

%% Update Message

for n=1:N

message\_n = Message0(:,:,n);

for i=1:K

for j=i+1:K

a=0.5;

% update Mj->i

np\_old = f(n,j)+sum(message\_n(:,j))-message\_n(i,j);

np\_true = - mu(:,i)'\*mu(:,j)/(sigma^2);

np\_new = (1+exp(np\_old+np\_true))/(1+exp(np\_old));

% message\_n(j,i) = log(np\_new);

message\_n(j,i) = a\*message\_n(j,i) + (1-a)\*log(np\_new);

% update Mi->j

np\_old = f(n,i)+sum(message\_n(:,i))-message\_n(j,i);

np\_true = - mu(:,j)'\*mu(:,i)/(sigma^2);

np\_new = (1+exp(np\_old+np\_true))/(1+exp(np\_old));

% message\_n(i,j) = log(np\_new);

message\_n(i,j) = a\*message\_n(i,j) + (1-a)\*log(np\_new);

end

end

Message(:,:,n) = message\_n;

end

lambda = zeros(N,K);

for n=1:N

np = f(n,:)+sum(Message(:,:,n));

lambda(n,:) = 1./(1+exp(-np));

end

%% Compute Free Energy

index = find(lambda>0 & lambda<1);

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

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

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

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

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

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

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

-sum(sum(lambda(index).\*log(lambda(index))))-sum(sum((1-lambda(index)).\*log(1-lambda(index))));

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

%% check if F increases after update

% if abs(F-F\_last) < thres;

% break;

% end

diff\_max = max(max(max((abs(Message-Message0)))));

if diff\_max < thres;

break;

end

F\_last = F;

Message0 = Message;

end

end

% initial\_para.m

function [Message0,mu,sigma,pie] = initial\_para(X,N,D,K)

% random generate natural parameters of lambda

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);

% random generate Message matrix

Message0 = rand(K,K,N);

for n=1:N

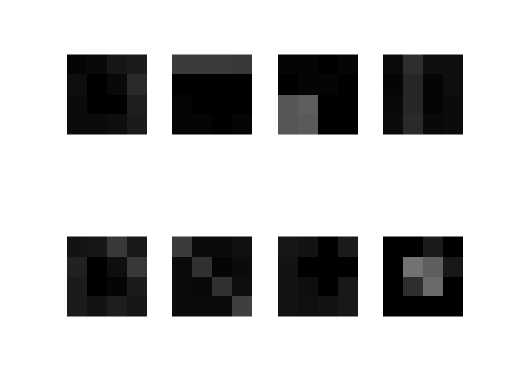
b = Message0(:,:,n);

b = b-diag(diag(b));

Message0(:,:,n) = b;

end

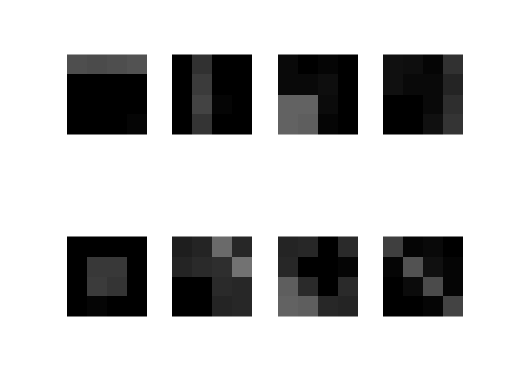
*Results: Plot the features we learn below.*



*Figure 1 Feature means for loopy-BP*

If we see clearly, our algorithm has already learned total 8 features. But some features are not very obvious.

Compare to the result we run on Mean-Field algorithm as below.



*Figure 2 Feature means for Mean-Field*

These results were all run on 100 iterations. I think the result of mean-field is better than loopy-BP. The hard part of loopy-BP is to check the convergence. It seems that free energy can’t totally reflect the update of message. Sometimes, free energy even fluctuates as below.



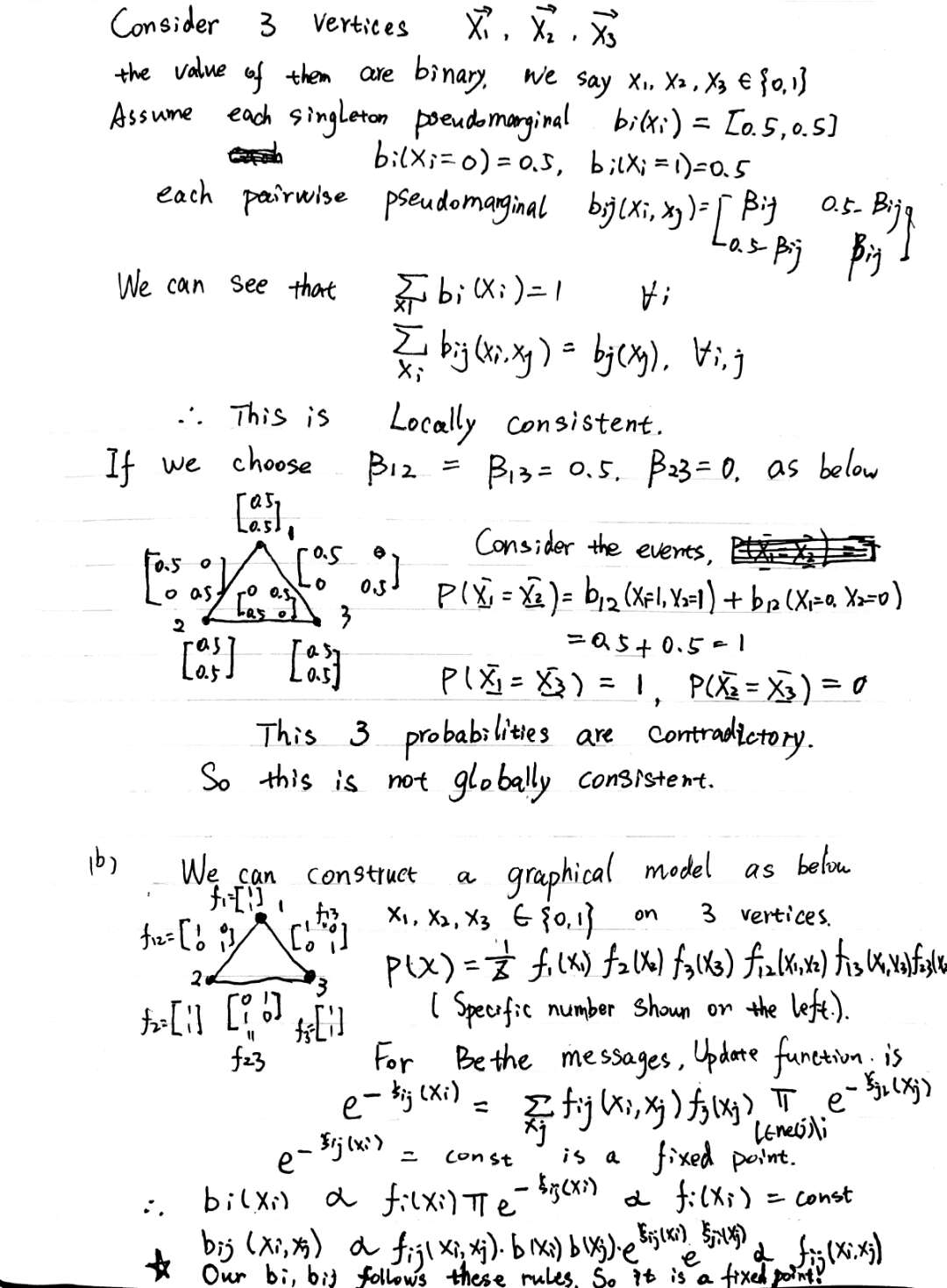
*Figure 3 Free Energy during iterations for loopy-BP*

Loopy-BP is not converged under this situation. To solve this problem, we can use power EP to update the message.

message = 0.5\*message\_old + 0.5\* message\_new

to make loopy-BP easier to converge.

#### 4. Inconsistency of Local Marginals (a) Locally consistent not globally consistent



#### (b) Construct a graphical model

