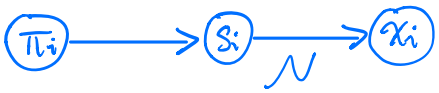


Mean-field learning

Consider a binary latent factor model. This is a model with a vector $\mathbf{s}$ of K binary latent variables, $\mathbf{s} = (s_1, \ldots, s_K)$, a real-valued observed vector $\mathbf{x}$ and parameters $\boldsymbol{\theta} = \{\{\boldsymbol{\mu}_i, \pi_i\}_{i=1}^K, \sigma^2\}$. The model is described by:

$$p(\mathbf{s}|\boldsymbol{\pi}) = p(s_1, \ldots, s_K|\boldsymbol{\pi}) = \prod_{i=1}^K p(s_i|\pi_i) = \prod_{i=1}^K \pi_i^{s_i}(1-\pi_i)^{(1-s_i)}$$

$$p(\mathbf{x}|s_1, \ldots, s_K, \boldsymbol{\mu}, \sigma^2) = \mathcal{N}\left(\sum_i s_i \boldsymbol{\mu}_i, \sigma^2 I\right)$$

where $\mathbf{x}$ is a D-dimensional vector and I is the $D \times D$ identity matrix. Assume you have a data set of N i.i.d. observations of $\mathbf{x}$, i.e. $\mathcal{X} = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}$.

We will implement generalized EM learning using the fully factored (a.k.a. mean-field) **variational approximation** for the model above. That is, for each data point $\mathbf{x}^{(n)}$, we will approximate the posterior distribution over the hidden variables by a distribution:

$$q_n(\mathbf{s}^{(n)}) = \prod_{i=1}^K \lambda_{in}^{s_i^{(n)}}(1-\lambda_{in})^{(1-s_i^{(n)})}$$

and find the $\boldsymbol{\lambda}^{(n)}$'s that maximize $\mathcal{F}_n$ holding $\boldsymbol{\theta}$ fixed.

(a) Write a Matlab function:
`[lambda,F] = MeanField(X,mu,sigma,pie,lambda0,maxsteps)`
where `lambda` is $N \times K$, `F` is the lower bound on the likelihood, `X` is the $N \times D$ data matrix ($\mathcal{X}$), `mu` is the $D \times K$ matrix of means, `pie` is the $1 \times K$ vector of priors on $\mathbf{s}$, `lambda0` are initial values for `lambda` and `maxsteps` are maximum number of steps of the fixed point equations. You might also want to set a convergence criterion so that if `F` changes by less than some very small number ϵ the iterations halt.

(b) We have derived the M step for this model in terms of the quantities: `X`, `ES` $= E_q[\mathbf{s}]$, which is an $N \times K$ matrix of expected values, and `ESS`, which is an $N \times K \times K$ array of expected values $E_q[\mathbf{s}\mathbf{s}^\top]$ for each n. The full derivation is provided in Appendix B. Write two or three sentences discussing how the solution relates to linear regression and why.

(c) Using the above, we have implemented a function:

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

This can be implemented either taking in `ESS` = a $K \times K$ matrix summing over N the `ESS` array as defined above, or taking in the full $N \times K \times K$ array. This code can be found in Appendix C and can also be found on the web site. Study this code and figure out what the computational complexity of the code is in terms of N, K and D for the case where `ESS` is $K \times K$. Write out and justify the computational complexity; don't assume that any of N, K, or D is large compared to the others.

(d) Examine the data `images.jpg` shown on the web site (Do **not** look at `genimages.m` yet!). This shows 100 greyscale 4×4 images generated by randomly combining several features and adding a little noise. Try to guess what these features are by staring at the images. How many are there? Would you expect factor analysis to do a good job modelling this data? How about ICA? mixture of Gaussians? Explain your reasoning.

(e) Put the E step and M step code together into a function:

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

where K is the number of binary factors, and `iterations` is the maximum number of iterations of EM. Include a check that `F` increases at every iteration (this is a good debugging tool).

(f) Run your algorithm for learning the binary latent factor model on the data set generated by `genimages.m`. What features `mu` does the algorithm learn (rearrange them into 4×4 images)? How could you improve the algorithm and the features it finds? Explain any choices you make along the way and the rationale behind them (e.g. what to set K, how to initialize parameters, hidden states, and `lambda`s).

(g) For the setting of the parameters learned in the previous step, run the variational approximation for just the first data point (i.e. to find $q_1(\mathbf{s}^{(1)})$) (i.e. set $N = 1$). Convergence of a variational approximation results when the value of λ's as well as F stops changing. Plot `F` and `log(F(t)-F(t-1))` as a function of iteration number `t` for `MeanField`. How rapidly does it converge? Plot `F` for three widely varying `sigma`s. How is this affected by increases and decreases of `sigma`? Why? Support your arguments.

(h) Describe a (variational) Bayesian method for selecting K, the number of hidden binary variables in this model. Does your method pose any computational difficulties and if so how would you tackle them?

[Continued . . .]

(a) The Free Energy

$$\mathcal{F} = \sum_{n=1}^{N} \left\langle \log \mathcal{P}(X^{(n)}, S^{(n)} \mid \theta, \pi) \right\rangle_{q(S^{(n)})} - \left\langle \log q(S^{(n)}) \right\rangle_{q(S^{(n)})}$$

$$= \sum_{n=1}^{N} \underbrace{\left\langle \log \mathcal{P}(X^{(n)} \mid S^{(n)}, \theta) \right\rangle_{q(S^{(n)})}}_{①} + \underbrace{\left\langle \log \mathcal{P}(S^{(n)} \mid \pi) \right\rangle_{q(S^{(n)})}}_{②} - \underbrace{\left\langle \log q(S^{(n)}) \right\rangle_{q(S^{(n)})}}_{③}$$

Maximising $\mathcal{F}$ to get expression of λ_{in} by taking derivative w.r.t. λ_i and set it to 0 to get fixed point.

$$② \equiv \sum_{n=1}^{N} \left\langle \log \prod_{k=1}^{K} \pi_k^{S_k^{(n)}} (1-\pi_k)^{(1-S_k^{(n)})} \right\rangle_{q(S^{(n)})} = \left\langle \sum_{k=1}^{K} S_k^{(n)} \log \pi_k + (1-S_k^{(n)}) \log (1-\pi_k) \right\rangle_{q(S^{(n)})}$$

$$= \sum_{n=1}^{N} \sum_{k=1}^{K} \langle S_k^{(n)} \rangle_{q(S^{(n)})} \log \pi_k + \langle 1 - S_k^{(n)} \rangle_{q(S^{(n)})} \log (1-\pi_k)$$

$$= \sum_{n=1}^{N} \sum_{k=1}^{K} \lambda_k^{(n)} \log \pi_k + (1-\lambda_k^{(n)}) \log (1-\pi_k).$$

$$③ \equiv \sum_{n=1}^{N} \left\langle \log \prod_{k=1}^{K} \lambda_k^{(n) S_k^{(n)}} (1-\lambda_k^{(n)})^{(1-S_k^{(n)})} \right\rangle_{q(S^{(n)})} = \sum_{n=1}^{N} \sum_{k=1}^{K} \langle S_k^{(n)} \rangle_{q(S^{(n)})} \log \lambda_k^{(n)} + \langle 1 - S_k^{(n)} \rangle_{q(S^{(n)})} \log (1-\lambda_k^{(n)})$$

$$= \sum_{n=1}^{N} \sum_{k=1}^{K} \lambda_k^{(n)} \log \lambda_k^{(n)} + (1-\lambda_k^{(n)}) \log (1-\lambda_k^{(n)})$$

$$① \equiv \sum_{n=1}^{N} \left\langle \log \mathcal{P}(\underset{\sim}{x}^{(n)} \mid \underset{\sim}{\mu}, \Sigma, \theta) \right\rangle_{q(S)} = \sum_{n=1}^{N} \left\langle \log (2\pi\sigma^2)^{-D/2} \exp\left\{ \left(\underset{\sim}{x} - \sum_{k=1}^{K} S_k^{(n)} \underset{\sim}{\mu}_k \right)^{T} \left(\underset{\sim}{x} - \sum_{k=1}^{K} S_k^{(n)} \underset{\sim}{\mu}_k \right) \right\} \right\rangle_{q(S)}.$$

$$= -\frac{ND}{2} \log 2\pi - ND \log \sigma - \frac{1}{2\sigma^2} \sum_{n=1}^{N} \left\{ \underset{\sim}{x}^{(n)T} \underset{\sim}{x}^{(n)} + \sum_{m,k}^{K} \underset{\sim}{\mu}_k^{T} \underset{\sim}{\mu}_m \langle S_k^{(n)} S_m^{(n)} \rangle_{q(S^{(n)})} - 2 \sum_{k=1}^{K} \underset{\sim}{\mu}_k^{T} \underset{\sim}{x}^{(n)} \langle S_k^{(n)} \rangle_{q(S^{(n)})} \right\}$$

where $\underset{\sim}{x}^{(n)} \in \mathbb{R}^{D\times 1}$, $\underset{\sim}{\mu}_k \in \mathbb{R}^{D\times 1}$

Note $\langle S_k S_m \rangle_{q(S)} = \begin{cases} \lambda_k \lambda_m & \text{if } m \neq k \\ \lambda_k & \text{if } m = k. \end{cases}$

$$= -\frac{ND}{2} \log 2\pi - ND \log \sigma - \frac{1}{2\sigma^2} \sum_{n=1}^{N} \left\{ \underset{\sim}{x}^{(n)T} \underset{\sim}{x}^{(n)} + \sum_{m\neq k}^{K} \sum_{k=1}^{K} \lambda_k^{(n)} \lambda_m^{(n)} \underset{\sim}{\mu}_k^{T} \underset{\sim}{\mu}_m + \sum_{k=1}^{K} \lambda_k^{(n)} \underset{\sim}{\mu}_k^{T} \underset{\sim}{\mu}_k - 2 \sum_{k=1}^{K} \underset{\sim}{\mu}_k^{T} \underset{\sim}{x}^{(n)} \lambda_k^{(n)} \right\}$$

$$\frac{\partial \mathcal{F}}{\partial \lambda_i^{(n)}} = 0, \quad \text{where } i \in \{1, \cdots, K\}, \quad n \in \{1, \cdots, N\}.$$

$$\frac{\partial \mathcal{F}}{\partial \lambda_i^{(n)}} = \log\frac{\pi_i}{1-\pi_i} - \log\frac{\lambda_i^{(n)}}{1-\lambda_i^{(n)}} - \frac{1}{2\sigma^2}\left\{\left(2\sum_{\substack{j=1\\ j\neq i}}^{K}\lambda_j^{(n)}\mu_i^T\mu_j\right) + \mu_i^T\mu_i - 2\mu_i^T x^{(n)}\right\} = 0$$

$$\Leftrightarrow \log\frac{\lambda_i^{(n)}}{1-\lambda_i^{(n)}} = \log\frac{\pi_i}{1-\pi_i} - \frac{1}{\sigma^2}\left\{\left(\sum_{\substack{j=1\\ j\neq i}}^{K}\lambda_j^{(n)}\mu_i^T\mu_j\right) + \frac{1}{2}\mu_i^T\mu_i - \mu_i^T x^{(n)}\right\}$$

$$= \log\frac{\pi_i}{1-\pi_i} - \frac{1}{\sigma^2}\left\{\left(\sum_{j=1}^{K}\lambda_j^{(n)}\mu_i^T\mu_j\right) - \lambda_i^{(n)}\mu_i^T\mu_i + \frac{1}{2}\mu_i^T\mu_i - \mu_i^T x^{(n)}\right\}$$

$$= \log\frac{\pi_i}{1-\pi_i} - \frac{\mu_i^T}{\sigma^2}\left\{\left(\sum_{j=1}^{K}\lambda_j^{(n)}\mu_j\right) + \left(\frac{1}{2} - \lambda_i^{(n)}\right)\mu_i - x^{(n)}\right\}.$$

Denote the right hand side by α

then $\lambda_i^{(n)} = \text{sigmoid}(\alpha) = \frac{e^{\alpha}}{1+e^{\alpha}} = \frac{1}{e^{-\alpha}+1}$

$$= \text{sigmoid}\left[\log\frac{\pi_i}{1-\pi_i} - \frac{\mu_i^T}{\sigma^2}\left\{\left(\sum_{j=1}^{K}\lambda_j^{(n)}\mu_j\right) + \left(\frac{1}{2} - \lambda_i^{(n)}\right)\mu_i - x^{(n)}\right\}\right]$$

$\mu_j \in \mathbb{R}^{D\times 1}$

Rewrite λ and $\mathcal{F}$ into matrix form:

$$\lambda_i^{(n)} = \text{sigmoid}\left[\log\frac{\pi_i}{1-\pi_i} - \frac{\mu_i^T}{\sigma^2}\left\{\left(\sum_{j=1}^{K}\lambda_j^{(n)}\mu_j\right) + \left(\frac{1}{2} - \lambda_i^{(n)}\right)\mu_i - x^{(n)}\right\}\right]$$

$\mu_j \in \mathbb{R}^{D\times 1}$

$$\underset{N\times 1}{\lambda_i} = \text{sigmoid}\left[\log\frac{\underset{N\times 1}{\pi_i}}{\underset{N\times 1}{1-\pi_i}} - \frac{1}{\sigma^2}\left\{\underset{N\times D}{(\lambda\mu^T)} + \underset{N\times 1}{\left(\frac{1}{2} - \lambda_i\right)}\underset{1\times D}{\mu_i^T} - \underset{N\times D}{X}\right\}\underset{D\times 1}{\mu_i}\right]$$

$$\mathcal{F} = \left\langle \log p(S, X|\theta)\right\rangle_{q(S)} - \left\langle \log q(S)\right\rangle_{q(S)} = \sum_{n=1}^{N}\left\langle \log p(s^{(n)}, x^{(n)}|\theta)\right\rangle_{q(s^{(n)})} - \sum_{n=1}^{N}\left\langle \log q(s^{(n)})\right\rangle_{q(s^{(n)})}$$

$$= -\frac{ND}{2}\log 2\pi - ND\log\sigma - \frac{1}{2\sigma^2}\left[\sum_{n=1}^{N}x^{(n)T}x^{(n)} + \sum_{i,j=1}^{K}\mu_i^T\mu_j\sum_{n=1}^{N}\left\langle s_i^{(n)}s_j^{(n)}\right\rangle_{q(s^{(n)})} - 2\sum_{i=1}^{K}\mu_i^T\sum_{n=1}^{N}\left\langle s_i^{(n)}\right\rangle_{q(s^{(n)})}x^{(n)}\right]$$

$$+ \sum_{i=1}^{K}\left[\log\pi_i\sum_{n=1}^{N}\left\langle s_i^{(n)}\right\rangle_{q(s^{(n)})} + \log(1-\pi_i)\left(N - \sum_{n=1}^{N}\left\langle s_i^{(n)}\right\rangle_{q(s^{(n)})}\right)\right] - \sum_{i=1}^{K}\sum_{n=1}^{N}\left[\lambda_i^{(n)}\log\lambda_i^{(n)} + (1-\lambda_i^{(n)})\log(1-\lambda_i^{(n)})\right]$$

$$= -\frac{ND}{2}\log 2\pi - ND\log\sigma - \frac{1}{2\sigma^2}\left[\sum_{n=1}^{N}x^{(n)T}x^{(n)} + \sum_{i,j=1}^{K}\mu_i^T\mu_j\sum_{n=1}^{N}\left\langle s_i^{(n)}s_j^{(n)}\right\rangle_{q(s^{(n)})} - 2\sum_{i=1}^{K}\mu_i^T\sum_{n=1}^{N}\left\langle s_i^{(n)}\right\rangle_{q(s^{(n)})}x^{(n)}\right]$$

$$+ \sum_{k=1}^{K}\sum_{n=1}^{N}\left[\lambda_i^{(n)}\log\pi_i + (1-\lambda_i^{(n)})\log(1-\pi_i)\right] - \sum_{i=1}^{K}\sum_{n=1}^{N}\left[\lambda_i^{(n)}\log\lambda_i^{(n)} + (1-\lambda_i^{(n)})\log(1-\lambda_i^{(n)})\right]$$

(b) Consider the Multivariate Linear Regression in Lecture 1:

$P(\underset{\sim}{y}|\underset{\sim}{x}, W, \Sigma_y) = \mathcal{N}(W\underset{\sim}{x}, \Sigma_y)$ where $\underset{\sim}{y}$ is a function of $\underset{\sim}{x}$, with Gaussian noise, then the ML estimate for W is:

$$W^{ML} = \hat{W}^T = \left(\sum_i \underset{\sim}{x}_i \underset{\sim}{x}_i^T\right)^{-1} \sum_i y_i \underset{\sim}{x}_i$$

In our case, $P(\underset{\sim}{x}|s_1, \dots s_k, \mu, \sigma^2) = \mathcal{N}(\sum_i s_i \underset{\sim}{\mu}_i, \sigma^2 I)$, where $\underset{\sim}{x}$ is a linear function of $\underset{\sim}{\mu}_i$, with Gaussian noise. And in the M-step, update of $\underset{\sim}{\mu}_j$ has the same form of ML estimate.

i.e. $\frac{\partial}{\partial \underset{\sim}{\mu}_j}$ Free Energy $= \frac{\partial}{\partial \underset{\sim}{\mu}_j}$ Log likelihood

And we get the same form as linear regression:

$$\underset{\sim}{\mu}_j = \sum_i \left(\sum_n \langle s^{(n)} s^{(n)T} \rangle_{q(s^{(n)})}\right)^{-1}_{ji} \sum_n \langle s_i^{(n)} \rangle_{q(s^{(n)})} \underset{\sim}{x}^{(n)}.$$

(c) $O(K^3)+O(K^2N)+O(KND)$.

```
mu = (inv(ESS)*ES'*X)';
sigma = sqrt((trace(X'*X)+trace(mu'*mu*ESS)-2*trace(ES'*X*mu))/(N*D));
pie = mean(ES,1);
```

(KxK) (KxN) (NxD) (DxK) (KxN) (NxD) (DxK)

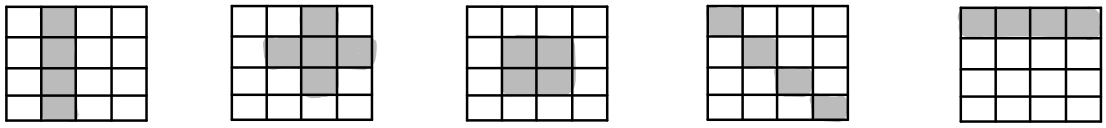
$O(D^2N)$ $O(K^2D)+O(K^3)$ $O(KND)+O(K^2D)$

* Inverse of Matrix: $O(n^3)$, where n is the dimension.
* Matrix Multiplication: $A \in \mathbb{R}^{x \times y}$, $B \in \mathbb{R}^{y \times z}$.
  AB: $O(xyz)$.

Here we omit the small terms.

The total computational cost is about $O(K^3) \approx O(KND) \approx O(K^2D)$
$\approx O(D^2N) \approx \cdots$

(d) Features from staring at the graphs: 5 factors



* Mixture of Gaussian: NO.
  This requires that each cluster follows a circle or ellipsis
  But the features doesn't follow this assumptions
  Also the features are not continuous but discrete.
* Factor Analysis: NO
  FA requires $p(x|s)$ and $p(s)$ to be both Gaussian,
  but in this case, $p(s)$ is Bernoulli. So we can't use FA.
* ICA: YES.
  ICA requires that $x|y$ is Gaussian and y is non Gaussian,
  which fits the model assumption here.

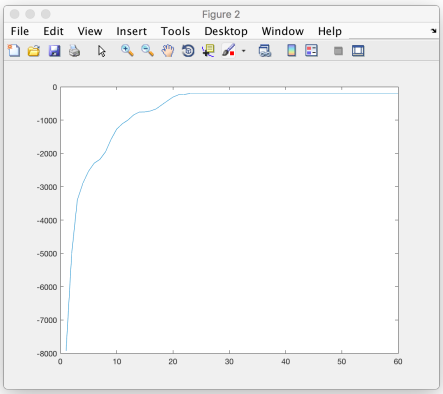
(e) LearnBinFactors :

```
function [mu, sigma, pie, FF] = LearnBinFactors(X,K,iterations)
%% INPUT:
% K: number of latent factors
% X: observed data
%% Initialise mu, sigma, pie, ESS (lambda)
    [N,D] = size(X);
    mu = rand(D,K);
    lambda = ones(N,K);
    FF = zeros(iterations,1);
    sigma = 1;
    pie = ones(1,K) * (1/K);
    maxsteps = 500; % max 500 steps in Esteps, but usually finishes much early than 100
    for iter = 1:iterations
        [lambda,F] = MeanField(X,mu,sigma,pie,lambda,maxsteps);
        FF(iter) = F;
        % check whether Free Energy increases every time
%         if (iter > 1) && (F - FF(iter-1)<0)
%             disp('Free Energy decreases! Something is wrong!!!!')
%             disp(F)
%         else
%             disp(F)
%         end
        ES = lambda;
        [ESS] = CalculateESS(ES);
        [mu, sigma, pie] = MStep(X,ES,ESS);
    end
end
```

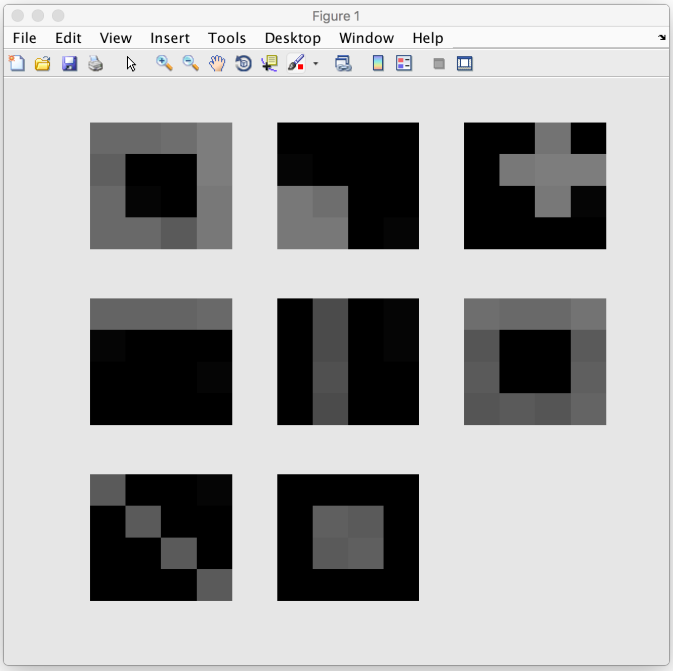
==========

```
function [ESS] = CalculateESS(ES)
    [N,K] = size(ES);
    ESS = zeros(K,K,N);
    for n = 1:N
        ESS(:,:,n) = ES(n,:)' * ES(n,:);
        for k = 1:K
            ESS(k,k,n) = ES(n,k);
        end
    end
    ESS = sum(ESS,3);
end
```

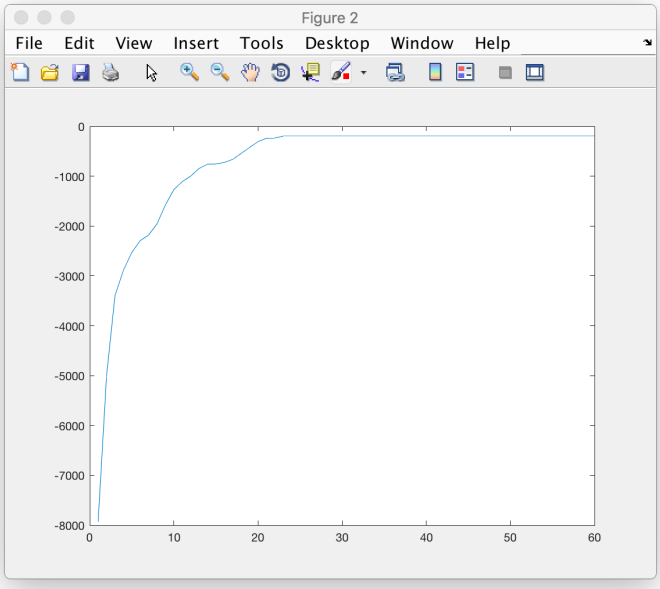
Also do a plot to check that ↓ free energy always increase.



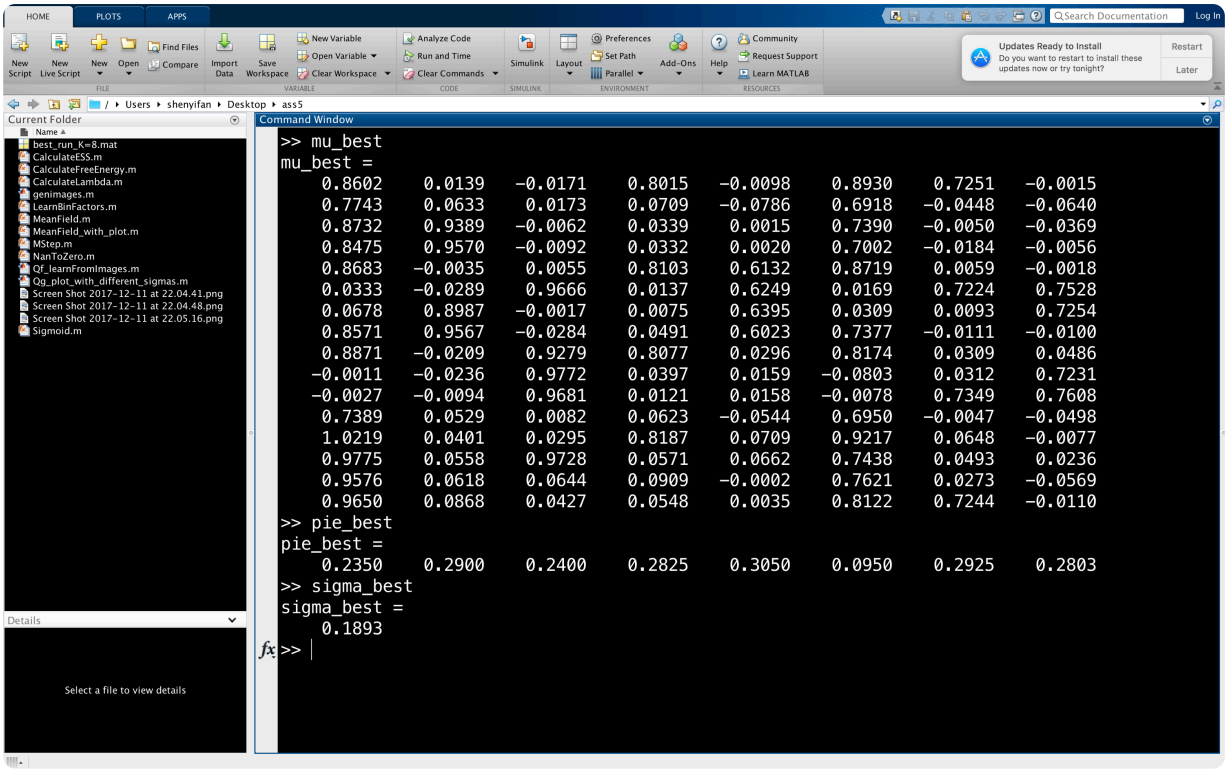
(f) Learnt mu: Successfully learnt 7 out of 8 features.



A check that free energy always increase.



Parameters learnt for (f)



* How to improve the features it find ?

Run multiple times with different random initialisation.

As it is likely to stuck at a local optimum.

Then choose the run with the largest free energy.

* About initialisations :

K : Test K with different numbers, from 5 to 10, then look at the features it find and whether the features are clear.

I start with K=5, which is the number of features I spotted (or look at genimages.m, know that we have 8 features ...)

π: Set $\pi = \text{ones}(1,k) * (1/k)$

$= (\frac{1}{k}, \frac{1}{k}, \cdots, \frac{1}{k})$

to give each feature an "equal" chance to be discovered.

μ: Set $mu = \text{rand}(D,k)$, take uniform values from $[0,1]$ at each element.

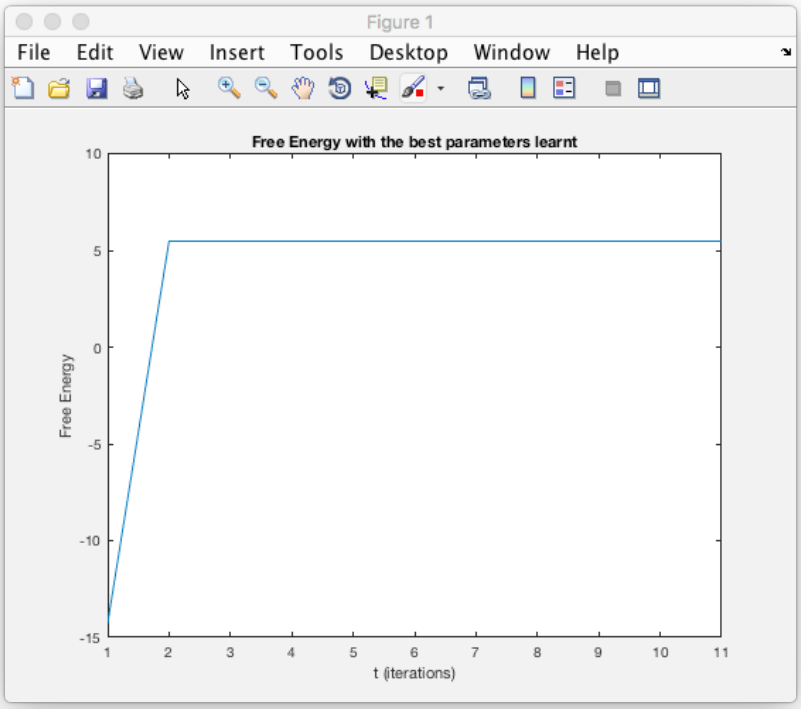
Because each "pixel" takes value between 0, 1, this is a good initialisation

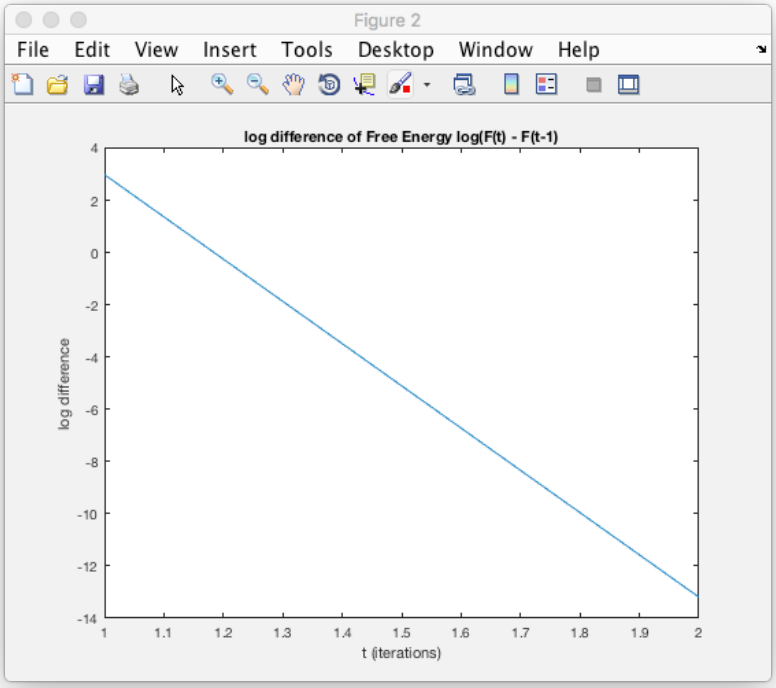
lambda: I tried rand(N,k), ① $\begin{pmatrix} \frac{1}{k}, \frac{1}{k}, \cdots \frac{1}{k} \\ \frac{1}{k} \\ \vdots \\ \frac{1}{k} \quad \cdots \quad \frac{1}{k} \end{pmatrix}$ ② N, ones(N,k) ③.

repmat$(\text{ones}(1,k)*(1/k), N, 1)$.

I think the second one is the best, because it gives each feature the equal chance to be discovered.

(g) (The parameters learnt are printed in the previous page ...)

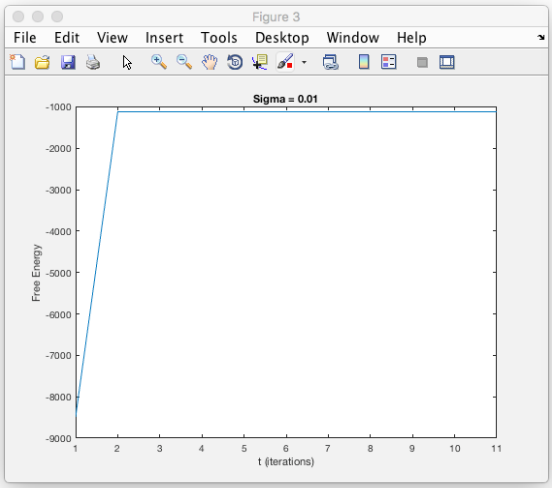


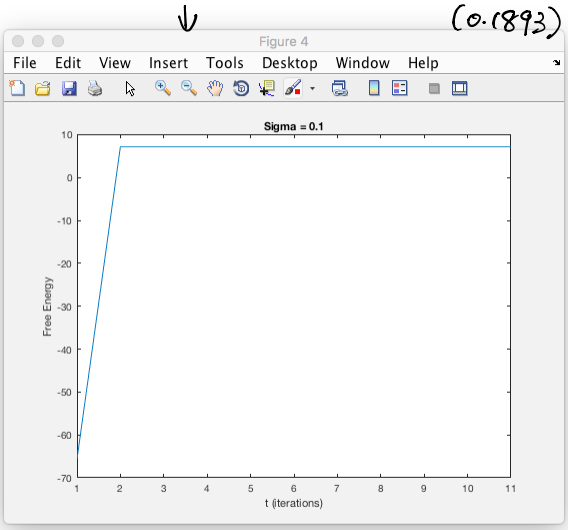


The convergence is RAPID, just at the second step
Because we use the parameters learnt from previous question, also we just use a single data, it is not surprising that the Variational E step converges rapidly.

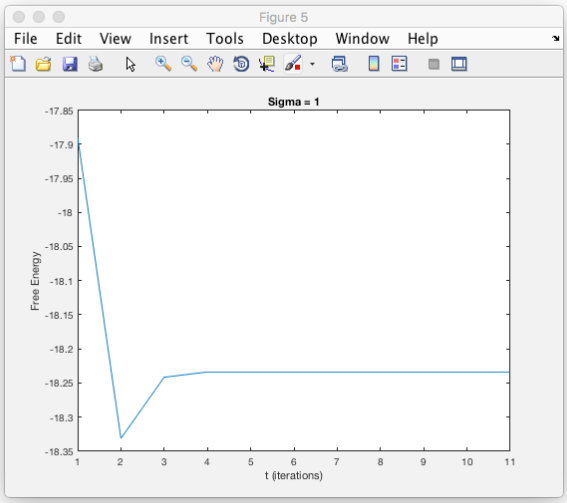
With different sigmas: 0.01, 0.1 and 1

Close to the learnt sigma (0.1893)





Free Energy is smaller with small sigma, but convergence is still rapid.



For a large sigma, it takes longer (4 steps here) to converge.
The free energy decreases to around −18 compared with the previous sigma

(h): By the Variational Bayesian Model Selection described in Lecture

We factor $Q(S, \mu) = q(S)\, q(\mu)$ for variational Bayes.

The VB free Energy in this case:

$$\mathcal{F}(q_S(S), q_\mu(\mu), \theta, \pi) = \left\langle \log P(X, S \mid \mu, \theta) + \log P(\mu \mid \pi) + \log(\theta) \right\rangle_{q_S, q_\mu} + \cdots$$

Then we optimise with respect to $q(S)$ and $q(\mu)$ to obtain a lower bound

Then we do a "Hyper" M-step to optimise hyperparameter λ

$$\pi \leftarrow \operatorname{argmax} \left\langle \log P(\mu \mid \pi) \right\rangle_{q_\mu}.$$

In this step, the dimension of S is selected, effectively learning the number of K.

Computational Difficulties:

Working with $P(\underset{\sim}{x} \mid \Sigma, \underset{\sim}{\mu}, \theta^2) = \mathcal{N}\left(\sum_i S_i \underset{\sim}{\mu}_i, \theta^2 I\right)$.

the computational complexity of inference scales cubically with the number of data points we have. (i.e $O(n^3)$) because we have to do covariance matrix inversion

Solution:

Introduce additionally latent variables