Summative Assignments

COMP0085

Candidate number : GKLN9

Contents

Introduction **2**

1 A biochemical pathway **3**
1.1 3
1.2 4
1.3 8
1.4 9
1.5 10

2 Bayesian linear and Gaussian process regression **10**
2.1 10
2.2 13
2.3 15
2.4 15
2.5 17
2.6 19
2.7 28

3 Mean-Field learning **28**
3.1 28
3.2 34
3.3 35
3.4 36
3.5 37
3.6 38
3.7 41

4 Variational Bayes for binary factors **45**
4.1 45
4.2 51

5 EP for the binary factor model **69**

5.1 69

5.2 71

5.3 79

5.4 82

A Appendix: Code to plot features Mean-Field **85**

B Appendix: Code to plot features VB ARD **85**

1 A biochemical pathway

1.1

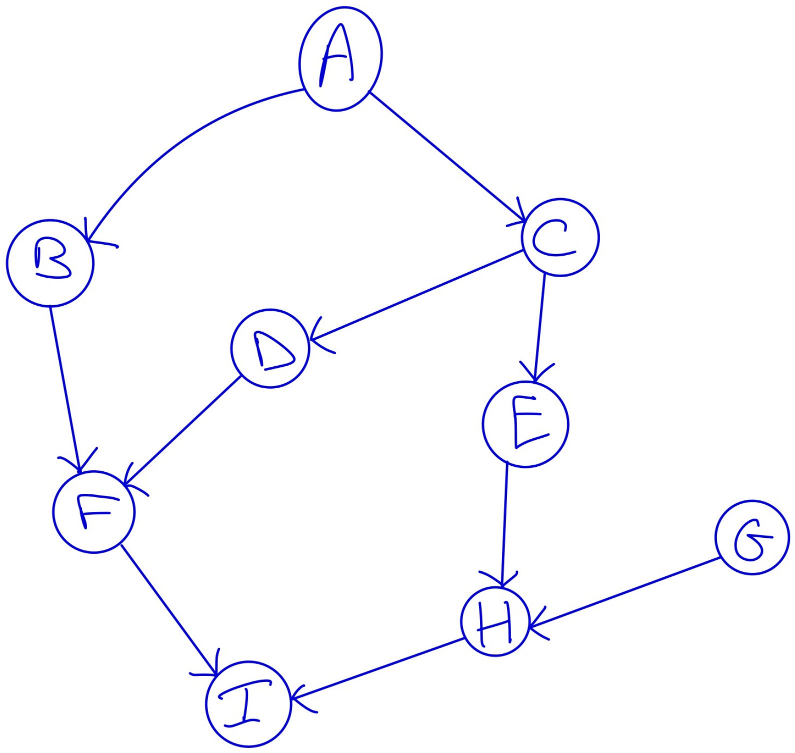


Figure 1: DAG

1.2

Moralised graph: Green edges are the added edges

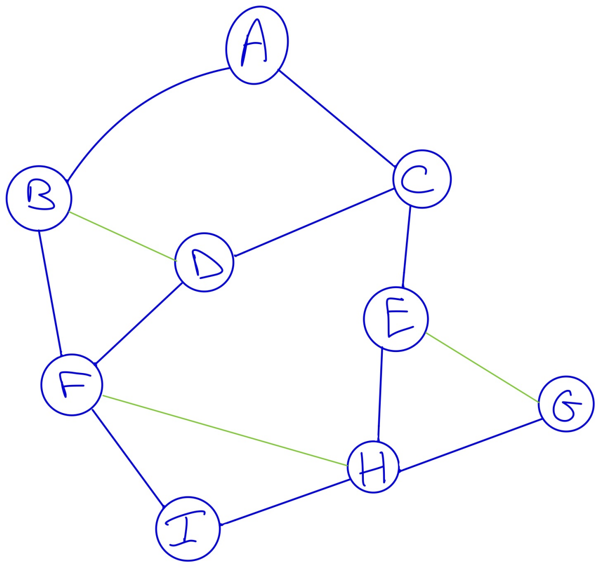


Figure 2: moralised graph

An efficient triangulation: Red edges are added to triangulate the moralized graph thus deleting any cycle subgraphs of more than 3 vertices that do not contain any triangles. ABCD formed a C_4 so add edd edge BC. CDFHE forms a C_5 so add CF and then add CH as well otherwise CFHE would form C_4.

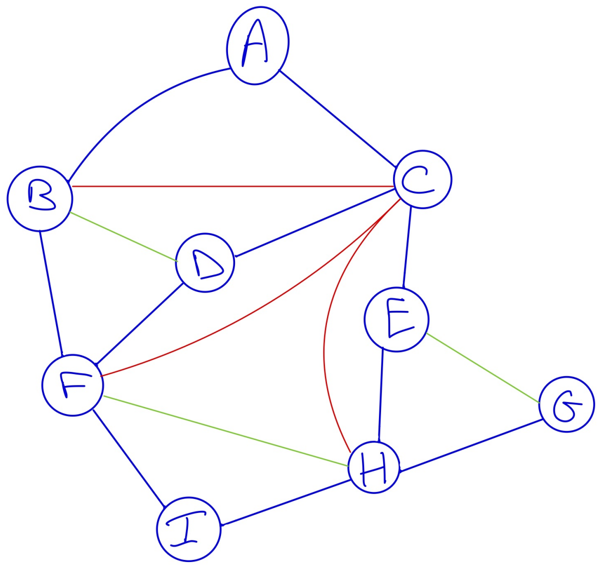


Figure 3: Efficient triangulation

Junction tree:

First, we construct a weighted graph with nodes labelled by the maximal cliques and edges connecting each pair of cliques that share variables giving

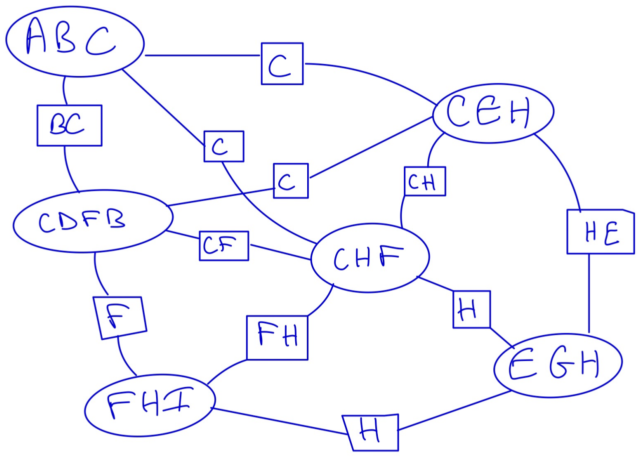


Figure 4: Weighted graph

Then we found the maximal-weight spanning tree of this weighted graph (weight of an edge is given by the number of nodes in the corresponding separator) using Prim's algorithm, giving Junction tree :

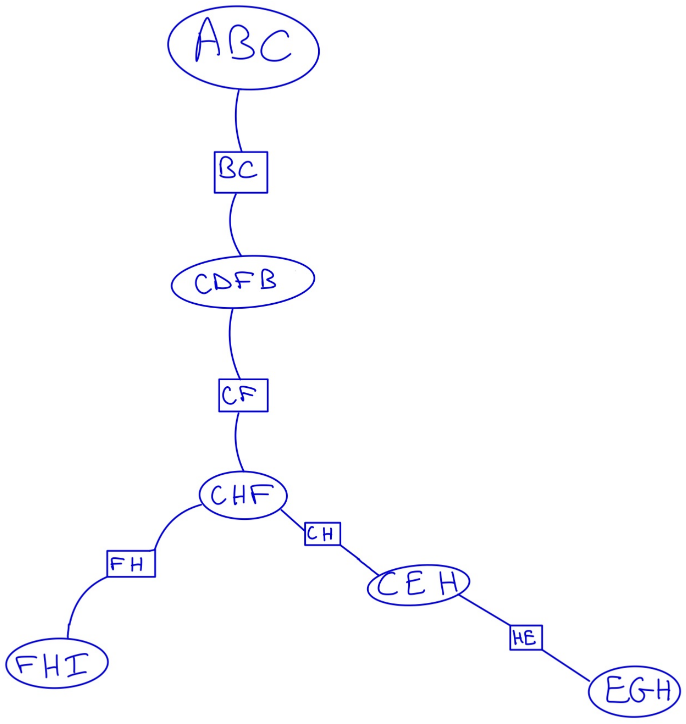


Figure 5: resulting Junction tree

Junction tree redrawn as a factor graph :

The joint distribution over the junction tree nodes is given by

$$P(A, B, C, D, E, F, G, H, I) \;=\; f_{ABC}\, f_{CDFB}\, f_{CHF}\, f_{FHI}\, f_{CEH}\, f_{EGH}$$

With corresponding factor graph

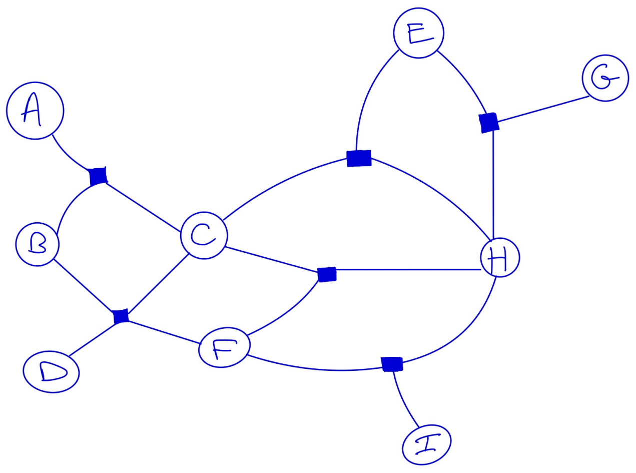


Figure 6: Junction tree redrawn as a factor graph

1.3

Using the technique given in the slide on D-separation we get $\{A, F, D, E, H\}$ as a (non-unique) smallest set of set of molecules, such that if the concentrations of the species within the set are known, the concentrations of the others, i.e $\{B, C, I, G\}$, would all be independent.

1.4

In our setting, Λ is a 9×9 loading matrix with columns A, B, C, D, E, F, G, H and I being respectively the first, second, ..., ninth column and similarly for rows So, e.g $\Lambda_{AB} = (\Lambda)_{12}$. Now the direct dependencies are given by the directed edges in the corresponding DAG, i.e Λ has non-zero elements :

Λ_{AB} (i.e corresponding to the direct dependency of concentration of B from A, i.e directed edge AB), Λ_{AC}, Λ_{CD}, Λ_{BF}, Λ_{DF}, Λ_{CE}, Λ_{EH}, Λ_{GH}, Λ_{GH}, Λ_{HI}, Λ_{FI} and the rest of Λ elements are zero-valued.

Now given observations $\delta[B]$, $\delta[D]$, $\delta[E]$, $\delta[G]$, and model of form :

$$\delta[\mathbf{x}] = \Lambda\delta[\mathbf{z}] + \epsilon_{\mathbf{x}}$$

Where $\delta[\mathbf{z}]$ is $\mathbf{x}$'s parent concentration perturbation and $\epsilon_{\mathbf{x}}$ is nodes reaction-specific gaussian noise variable, we get :

$$\delta[B] = \Lambda_{AB} \times \delta[A] + \epsilon_B$$

$$\delta[D] = \Lambda_{CD} \times \delta[C] + \epsilon_D$$

$$\delta[E] = \Lambda_{CE} \times \delta[C] + \epsilon_E$$

$$\delta[G] = 0 \times 0 + \epsilon_G$$

$$\Leftrightarrow \begin{bmatrix} \delta[B] \\ \delta[D] \\ \delta[E] \end{bmatrix} = \begin{bmatrix} \Lambda_{AB} & 0 \\ 0 & \Lambda_{CD} \\ 0 & \Lambda_{CE} \end{bmatrix} \begin{bmatrix} \delta[A] \\ \delta[C] \end{bmatrix} + \begin{bmatrix} \epsilon_B \\ \epsilon_D \\ \epsilon_E \end{bmatrix}$$

Since G has no parent and is observed, it doesn't contribute to a common factor in this model. Its variance is explained by its own unique noise and does not share any variance with other variables that would be explained by a common factor. Thus, we expect to recover the factors of A and C, i.e $\delta[A]$, $\delta[C]$.

1.5

In the context of the given biochemical cascade, using factor analysis poses specific limitations and possibilities regarding the inference about the concentration perturbations of the species and the linear weights in the graph. Factor analysis in this scenario can recover $\delta[A]$, and $\delta[C]$, because they are direct parents factors influencing the measured perturbations $\delta[B]$, $\delta[D]$, $\delta[E]$. However, for species that are children and children of children (i.e down the cascade) of the measured ones, like F, H, and I, their concentration perturbations cannot be directly inferred. This is because factor analysis here relies on the linear influences of parent factors on the measured variables, and without direct measurements or parent factor information for F, H, and I, their perturbations remain unidentified.
Regarding the identifiability of linear weights, factor analysis can identify the linear weights that link the recovered factors ($\delta[A]$, and $\delta[C]$,) to the measured perturbations ($\delta[B]$, $\delta[D]$, $\delta[E]$). These weights would be Λ_{AB} , Λ_{CD} , and Λ_{CE} . The weight Λ_{AC} can be inferred to some extent through factor analysis. Although $\delta[C]$, is recovered from the factor analysis and is influenced by $\delta[A]$, the model only captures the pattern of this relationship. Factor analysis focuses on estimating the relationships between latent factors (like $\delta[C]$,) and observed variables, not directly between latent factors themselves as the model assumes that latent factors are uncorrelated, which is why it only indirectly captures the influence of $\delta[A]$, on $\delta[C]$, as a pattern, rather than as a direct, quantifiable relationship. This is because A is a direct parent of C, and thus even though factor analysis assumes linear independence between these two latents, it doesn't change the fact that A and C are dependent variables. On the other hand, for the direct and indirect children weights (those influencing F, H, and I), identifiability is not feasible through this factor analysis, as it requires direct measurement of these nodes or information about their direct children (i.e I being observed in our case), which are not present in the current dataset.

2 Bayesian linear and Gaussian process regression

2.1

$$P(a, b \,|\, \mathcal{D}) \propto P(\mathcal{D} \,|\, a, b)\, P(a)\, P(b)$$

Let $\mathbf{w} = (a, b)^T$, $y_t = f(t)$ so $y_t = \mathbf{w}^T \mathbf{x}_t + \epsilon_t$ with $\mathbf{x}_t = (t, 1)^T$

So $\mathbf{w} \sim \mathcal{N}(\,\mu_w\,,\,\Sigma_w\,)$ where $\mu_w = (0, 360)^T$

$$\Sigma_w = \begin{pmatrix} 10^2 & 0 \\ 0 & 100^2 \end{pmatrix} \quad \Rightarrow \quad \Sigma_w^{-1} = \begin{pmatrix} \frac{1}{10^2} & 0 \\ 0 & \frac{1}{100^2} \end{pmatrix}$$

$$\Rightarrow \log P(\mathbf{w} \,|\mathcal{D}) \propto \sum_t \log \mathcal{N}(y_t | \mathbf{w}^T \mathbf{x}_t, 1) + \log \mathcal{N}(\mathbf{w} \,|\, \mu_w \,, \Sigma_w)$$

$$\propto -1/2 \left[(\mathbf{w} - \mu_w)^T \Sigma_w^{-1} (\mathbf{w} - \mu_w) + (\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w}) \right]$$

where $\mathbf{X}$ is a 500×2 matrix where each row is an $\mathbf{x}_t$ vector and $\mathbf{y}$ is a 500×1 column vector containing all 500 averages, i.e all y_t values. Keeping all the values w.r.t $\mathbf{w}$ we get

$$\propto -1/2 \left[\mathbf{w}^T (\Sigma_w^{-1} + \mathbf{X}^T\mathbf{X}) \mathbf{w} - 2\mathbf{w}^T \Sigma_w^{-1} \mu_w + \mu_w^T \Sigma_w^{-1} \mu_w - 2\mathbf{w}^T \mathbf{X}^T \mathbf{y} \right]$$

Rearranging the term above, we get

$$\log P(\mathbf{w} \,|\mathcal{D}) \propto -1/2 \left[\mathbf{w}^T (\Sigma_w^{-1} + \mathbf{X}^T\mathbf{X}) \mathbf{w} - 2\mathbf{w}^T (\Sigma_w^{-1} \mu_w + \mathbf{X}^T \mathbf{y}) + \mu_w^T \Sigma_w^{-1} \mu_w \right]$$

$$\propto -1/2 \left[\mathbf{w}^T (\Sigma_w^{-1} + \mathbf{X}^T\mathbf{X}) \mathbf{w} - 2\mathbf{w}^T (\Sigma_w^{-1} \mu_w + \mathbf{X}^T \mathbf{y}) \right]$$

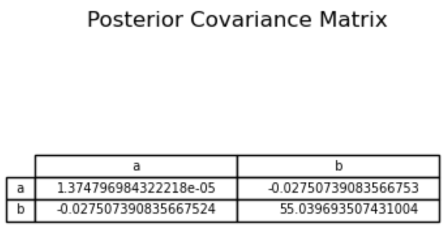
Now posterior is equal to a product of Gaussians so it is a Gaussian itself and thus the quadratic term in its pdf has form $\mathbf{w}^T \Sigma_{post}^{-1} \mathbf{w} - 2\mathbf{w}^T \Sigma_{post}^{-1} \mu_{post} + \mu_{post}^T \Sigma_{post}^{-1} \mu_{post}$. Thus, by completing the square in our expression above we get

$$\mathbf{w} \,|\mathcal{D} \sim \mathcal{N}(\mu_{post} \,, \Sigma_{post})$$

with,

$$\Sigma_{post} = \left(\Sigma_w^{-1} + \mathbf{X}^T\mathbf{X} \right)^{-1}$$

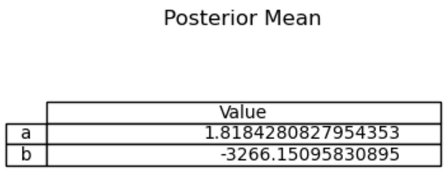
$$\mu_{post} = \Sigma_{post} \left(\Sigma_w^{-1} \mu_w + \mathbf{X}^T \mathbf{y} \right)$$



Posterior Covariance Matrix

	a	b
a	1.374796984322218e-05	-0.02750739083566753
b	-0.027507390835667524	55.039693507431004

Figure 7:



Posterior Mean

	Value
a	1.8184280827954353
b	-3266.15095830895

Figure 8:

Code for question a) :

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# Define the file path
file_path = 'C:\\Users\\chdor\\OneDrive\\Desktop\\co2.txt'

# Load the data into a pandas DataFrame
# Skipping the initial descriptive rows and focusing on the tabular
    data
# The columns are: year, month, decimal, average, trend
co2_data = pd.read_csv(file_path,
                        delim_whitespace=True,
                        skiprows=58,  # Skipping the initial
    descriptive text
                        names=['year', 'month', 'decimal', 'average'
    , 'trend'])

co2_data.head()

```

```python
# Extract the 'decimal' column
decimals = co2_data['decimal'].values

# Create a column of ones
ones = np.ones_like(decimals)

# Concatenate these two columns to form the matrix X
# Note: we reshape decimals and ones to be column vectors
X = np.column_stack((decimals, ones))

prior_precision_w= np.diag([1/100, 1/10000])
X_T_X = X.T @ X
precision = X_T_X + prior_precision_w
posterior_cov = np.linalg.inv(precision)
posterior_cov

y = co2_data['average'].values
X_T_y = X.T @ y
prior_mean = np.array([0, 360])
Sigma_w_mean_w = prior_precision_w @ prior_mean
M = X_T_y.reshape(-1, 1) + Sigma_w_mean_w.reshape(-1, 1)
posterior_mean = posterior_cov @ M
posterior_mean
```

2.2

Code for question b) :

```python
# Extract a_MAP and b_MAP from the posterior mean
a_MAP = posterior_mean[0, 0]
b_MAP = posterior_mean[1, 0]

# Compute the residuals
g_obs = y - (a_MAP * decimals + b_MAP)

plt.figure(figsize=(10, 6))
plt.plot(decimals, g_obs, label='Residuals')
plt.xlabel('Decimal Year')
plt.ylabel('Residuals')
plt.title('Residuals over Time')
plt.legend()
plt.show()

# Further analysis: histogram and statistics
plt.figure(figsize=(10, 6))
plt.hist(g_obs, bins=50, edgecolor='k', alpha=0.7)
plt.title('Histogram of Residuals')
plt.xlabel('Residual Value')
plt.ylabel('Frequency')
plt.show()

print("Mean of residuals:", np.mean(g_obs))
print("Standard deviation of residuals:", np.std(g_obs))
```

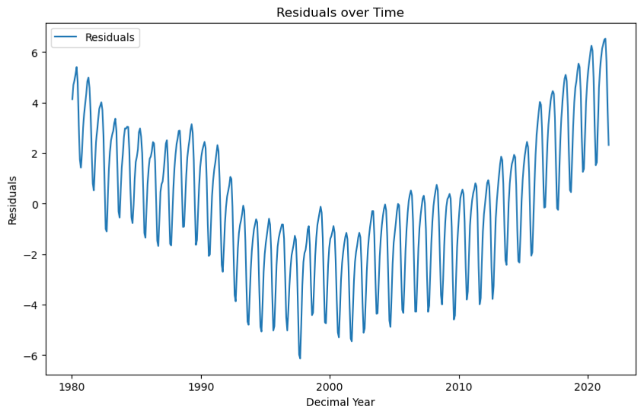


Figure 9: Enter Caption

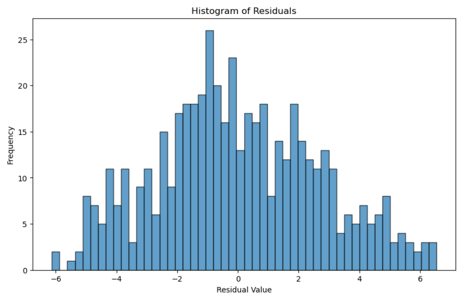


Figure 10: Enter Caption

Mean of residuals : -0.000725229868844508

Standard deviation of residuals : 2.6810899030171584

The residuals from the linear regression do not conform to our prior belief that the noise term $\epsilon(t)$ is i.i.d. with a distribution N(0,1) (i.e prior belief of i.i.d time-invariant, homoscedastic standard Gaussian noise). The residual plot reveals a periodic pattern (probably due to the seasonal cycles of CO2 emissions), indicating dependence on time, which contradicts the assumption of independence. While the histogram may suggest a normal distribution around zero, the assumption of a standard deviation of one is not supported due to the changing amplitude in the residuals (and also by looking at the standard deviation of residuals above). These observations, combined with the non-linear trend in the data, demonstrate that the simple linear model and the associated prior on $\epsilon(t)$ are inadequate. A more complex model that can capture the periodic and non-linear nature of the data, such as a Gaussian process with an appropriate kernel, is required.

2.3

```python
def samples_from_GP(kernel_function, x_points, num_samples=1):

    num_points = len(x_points)

    # Initializing the Square matrix, i.e covariance kernel
    covariance_matrix = np.zeros((num_points, num_points))
    for i in range(num_points):
        for j in range(num_points):
            covariance_matrix[i, j] = kernel_function(x_points[i], x_points[j])

    # Generating samples form joint gaussian
    samples = np.random.multivariate_normal(mean=np.zeros(num_points), cov=covariance_matrix, size=num_samples)

    return samples
```

2.4

```python
def kernel_function(s, t):

   # Periodic part of the kernel function
    First_term = np.exp(-2 * (np.sin( ( np.pi * (s - t) ) / tau) / sigma)**2)

    # Squared exponential part
    Second_term = (phi**2) * np.exp(-0.5 * ((s - t) / eta)**2)

    # LHS term
    LHS = (theta**2) * (First_term + Second_term)

    # Noise term, have delta so that noise is only applied on diagonal elements of covariance kernel as noise is i.i.d
    Noise = zeta**2 if s == t else 0


    sample_function = LHS + Noise


    return sample_function
```

The functions drawn from this GP are characterized by their oscillatory nature (due to the periodic component), local variability and adaptability (due to the squared exponential component), and an element of randomness or noise. The interplay of these components, controlled by the hyperparameters, allows the GP to model complex, real-world phenomena that exhibit both structured patterns (like seasonal cycles), non-linear trends (non-linear increase), and random variations.

Periodic Component (Influenced by τ and σ):

Oscillatory Nature: Functions exhibit a cyclic, oscillatory pattern due to the periodic component. This is especially noticeable in phenomena with regular, repeating patterns, like seasonal effects.

Frequency of Oscillations (τ): Determines the frequency of these oscillations. A smaller τ leads to more frequent oscillations (shorter cycles), while a larger τ results in fewer, more spaced-out oscillations.

Smoothness of Oscillations (σ): Influences the sharpness or smoothness of oscillations. A larger σ smoothens the oscillations, making them less distinct, whereas a smaller σ produces sharper, more pronounced oscillations. It inversely scales the sine squared term in the periodic component of the kernel, directly influencing the sharpness and prominence of the oscillations in the functions generated by the GP. Smaller values of σ lead to more acute responses to changes in the input space, while larger values smooth out these responses, leading to more gentle oscillations.

Squared Exponential Component (Influenced by ϕ and η):

Local Variability and Flexibility: Introduces smooth variations and allows functions to deviate from the strict periodic pattern. This component adds a level of adaptability and complexity to the functions.

Amplitude of Variability (ϕ): Scales this part of the kernel, affecting the vertical variation. A larger ϕ allows for more dramatic deviations, leading to functions that are more variable and can exhibit larger vertical movements. It acts as a scaling factor for the squared exponential component of the kernel. It directly influences the vertical variability or amplitude of the function values attributable to this component. This parameter is crucial in determining how much the squared exponential component contributes to the overall variability and complexity of the functions. A smaller ϕ emphasizes the periodic nature, while a larger ϕ enhances the local variability and vertical fluctuations, leading to functions that are more complex and less strictly periodic.

Length Scale (η): The length scale η determines how sensitive the squared exponential kernel is to the distance between input points, i.e determines the rate at which function values change. A smaller η means that even small changes in the input space will lead to large differences in the covariance (and hence in the function values), i.e it allows for rapid variations over short distances With a large η, the functions drawn from the GP will tend to exhibit clear and smooth periodic behavior, as the influence of the squared exponential component is spread over a larger range, reducing rapid, local fluctuations and emphasizing the underlying periodic pattern. This results in functions that appear more regularly periodic, with smoother transitions between the peaks and troughs of the oscillations.

Scaling factor: θ serves as a scaling factor for the magnitude of both the periodic and squared exponential components in the kernel. It allows you to adjust how prominently these components feature in the functions sampled from the GP, affecting the overall vertical scale of the function variations.

Noise Term (ζ): The term ζ represents inherent uncertainty or potential measurement error in each data point, rather than adding random noise. This term contributes to the diagonal of the covariance matrix, reflecting the model's allowance for variability or inaccuracy in individual observations. The magnitude of ζ influences how much the model accounts for this inherent uncertainty: a larger ζ implies greater expected variability or error in the observations. While it does not make the functions more irregular in a random sense, a higher ζ can lead to broader confidence intervals around the model's predictions, indicating less certainty in the data points. This effect is particularly noticeable when examining smaller sections of the function, where the model may exhibit less smoothness due to the increased emphasis on potential measurement error.

2.5

Upon examining the residual plot, it is evident that the oscillations possess a considerable amplitude. In light of this, selecting a value of 4 for θ is intentional to capture this pronounced amplitude. This choice necessitates a corresponding adjustment of the squared exponential parameters to accommodate the increased θ, ensuring that the large amplitude is preserved without disproportionately amplifying the squared exponential term. The frequency of oscillations, as observed in the residuals relative to the decimal year, is not particularly high, prompting the selection of τ as 1 to align with what appears to be an annual cycle (behavior of global CO2 emissions). A σ value of 1.5 strikes a balance, chosen to replicate the smooth yet distinct oscillatory behavior noted in the residuals. For the squared exponential term, a larger η of 5 is chosen to reflect the residuals' apparent periodic trend, promoting a gradual shift in function values without abrupt local fluctuations. While θ influences the squared exponential term's scale, a slight increase to ϕ at 1.1 introduces an element of flexibility, allowing the model to emulate the quadratic nature of the residual plot. Finally, a modest ζ value of 0.1 is incorporated to account for a minimal level of noise, mirroring the subtle irregularities present in the residual plot.

Code :

```python

x_points = np.linspace(0, 40, 500)

# Hyperparameters
theta = 4
tau = 1
sigma = 1.5
phi = 1.1
eta = 5
zeta = 0.1


# Generate samples
samples = samples_from_GP(kernel_function, x_points, num_samples=3)

y_min, y_max = -7, 7  # Adjust these values as needed for your data

# Plotting samples
plt.figure(figsize=(10, 6))
for i in range(samples.shape[0]):
    plt.plot(x_points, samples[i], label=f'Sample {i+1}')


plt.xlabel('Input points (x)')
plt.ylabel('Function values')
plt.ylim(y_min, y_max)
plt.legend()
plt.show()
```

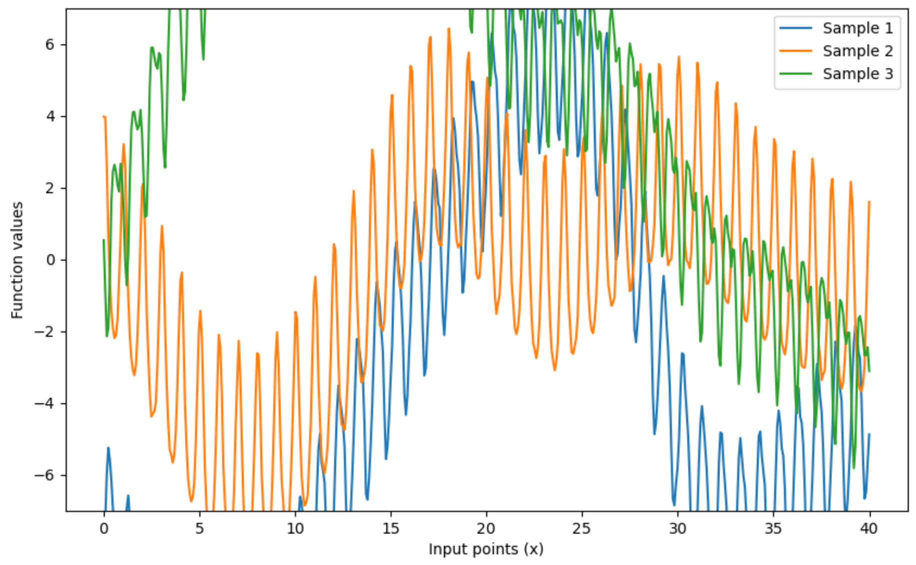


Figure 11: Random samples given hyperparameters above

2.6

From regression with Gaussian Processes slides we know that a finite set of predictions follows joint multivariate Gaussian:

$$y \,|\, X \,,\, Y \,,\, \mathbf{x} \sim \mathcal{N}\left(\, K_{\mathbf{x}X}\, \tilde{K}_{XX}^{-1}\, Y,\;\; K_{\mathbf{x}\mathbf{x}} \,-\, K_{\mathbf{x}X}\, \tilde{K}_{XX}^{-1}\, K_{X\mathbf{x}}\right)$$

where, $\tilde{K}_{XX}^{-1} = (K_{XX} + \sigma^2 I)^{-1}$ and in our case $\sigma^2 = 0$ and the covariance Kernel is the one defined in the previous question. Now, here we will have $\mathbf{x}$ be X pred, Y be residuals and X be decimals (all of them are defined in the code below).

Code:

```

# Hyperparameters
theta = 4
tau = 1
sigma = 1.5
phi = 1.1
eta = 5
zeta = 0.1


# First future point
First_future_decimal = 2021.708

# Creating future decimals (X_test) from last point to December
    2035
future_decimals = np.arange(First_future_decimal, 2036 + 1/12,
    1/12)

future_decimals = [round(d + 0.0002, 3) for d in future_decimals]
future_decimals = future_decimals[:-1]

# Input points
X_pred = np.array(future_decimals)

# Number of prediction input points and training points
num_pred_points = len(X_pred)
num_train_points = len(decimals)

# Computing the inverse covariance matrix K(X_train, X_train)
Kernel_matrix = np.zeros((num_train_points, num_train_points))
for i in range(num_train_points):
    for j in range(num_train_points):
        Kernel_matrix[i, j] = kernel_function(decimals[i], decimals
    [j])

inv_Kernel_matrix = np.linalg.inv(Kernel_matrix)

# Computing the Kernel matrix K(X_pred, decimals)
```

```
Kernel_matrix_pred_train = np.zeros((num_pred_points,
    num_train_points))

for i in range(num_pred_points):
    for j in range(num_train_points):
        Kernel_matrix_pred_train[i, j] = kernel_function(X_pred[i],
     decimals[j])


# Computing the Kernel matrix K(decimals, X_pred)
Kernel_matrix_train_pred = np.zeros((num_train_points,
    num_pred_points))

for i in range(num_train_points):
    for j in range(num_pred_points):
        Kernel_matrix_train_pred[i, j] = kernel_function(decimals[i
    ], X_pred[j])


# Computing the Kernel matrix K(X_pred, X_pred)
Kernel_matrix_pred_pred = np.zeros((num_pred_points,
    num_pred_points))

for i in range(num_pred_points):
    for j in range(num_pred_points):
        Kernel_matrix_pred_pred[i, j] = kernel_function(X_pred[i],
    X_pred[j])


# Computing Predictive mean
mean_Kernel_matrix_term = Kernel_matrix_pred_train @
    inv_Kernel_matrix
Predicitive_mean = np.dot(mean_Kernel_matrix_term, g_obs)


# Computing Predictive Covariance matrix
Predictive_cov = Kernel_matrix_pred_pred - (
    mean_Kernel_matrix_term @ Kernel_matrix_train_pred )
```

Plotting code :

```

f_t = a_MAP * X_pred + b_MAP + Predicitive_mean

# Calculating the standard deviation for the error bars
std_dev = np.sqrt(np.diag(Predictive_cov))

plt.figure(figsize=(10, 6))

# mean predictions as a continuous line
plt.plot(X_pred, f_t, label='GP Extrapolation', color='orange',
    linewidth=1)

# one standard deviation error bars as a shaded area
plt.fill_between(X_pred, f_t - std_dev, f_t + std_dev, color='grey'
    , alpha=0.2, label='1 SD')
```

```

# plot the observed CO2 levels
plt.plot(decimals, y, label='Observed CO2 Levels', color='blue',
    linewidth=1)

plt.xlabel('Decimal Year')
plt.ylabel('CO2 Concentration')
plt.title('Extrapolated CO2 Concentrations to 2035')
plt.legend()
plt.show()
```

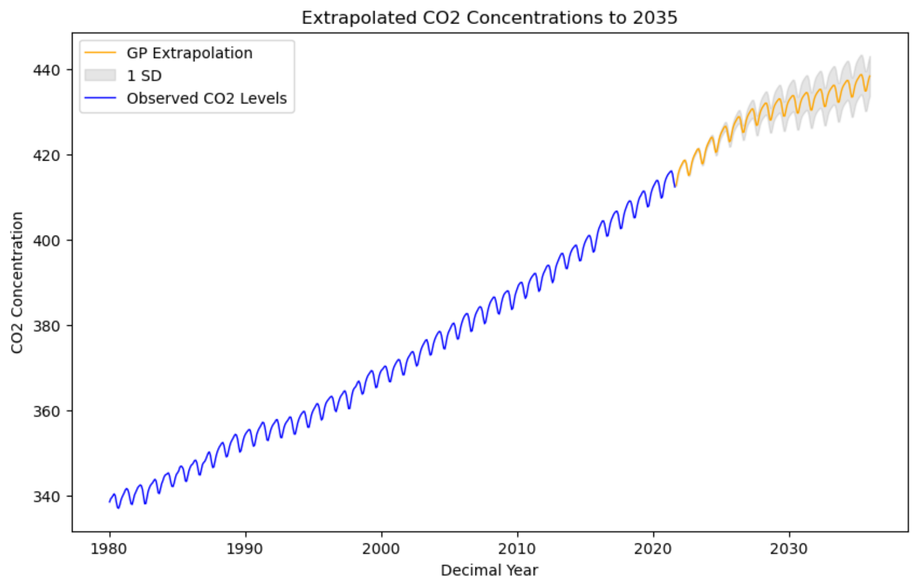


Figure 12:

The behavior of the model's extrapolation of CO2 concentration levels aligns with my expectations in key aspects. It accurately captures the seasonal periodicity observed in past emissions data, reflecting a realistic continuation of these patterns in terms of both period and amplitude. The model also aligns with my expectation of a steady increase in emissions, which is consistent with the observed data trend. However, the rate of this projected increase is somewhat conservative compared to the empirical observation that CO2 emissions have been increasing at a slightly faster, non-linear rate, indicative of the accelerating nature of climate change. This discrepancy in the rate of increase could be attributed to the inclusion of the COVID-19 period in the training data, which is marked by an atypical and temporary reduction in global emissions. Such anomalies in the training data can challenge the model's ability to accurately predict future trends, especially in scenarios like climate change where the progression is often more severe than linear projections would suggest.

To assess the sensitivity of our conclusions to the settings of the kernel hyperparameters, we will conduct a series of experiments. Starting with the hyperparameter values established in question e) as our baseline, we will vary one hyperparameter at a time, assigning it two distinct values while keeping the other hyperparameters constant. By observing the changes in the resulting plots for each of these variations, we can evaluate how each specific hyperparameter influences the model's performance and predictions.

τ : τ is set to 1 to match the annual seasonal cycle evident in the CO2 data, a pattern that has remained consistent since 1980. Since seasonal trends are unlikely to change significantly, altering τ would misalign the model's frequency with the known yearly periodicity, leading to less accurate predictions (see below).

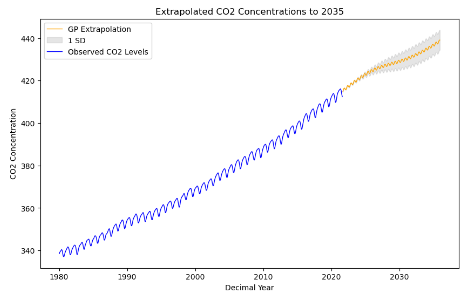


Figure 13: $\tau = 0.1$

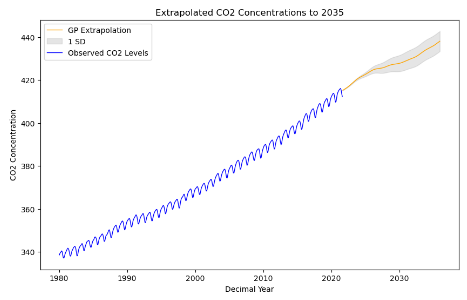


Figure 14: $\tau = 10$

σ : Despite theoretical expectations, varying σ in the Gaussian Process model between 0.1 and 10 shows little impact on the predictions. This suggests that the influence of σ on the smoothness of oscillations is overshadowed by other kernel components or by the nature of the CO2 data itself, which may not be sensitive to changes in the sharpness of seasonal patterns. Consequently, σ appears to play a minor role in this specific modeling context.

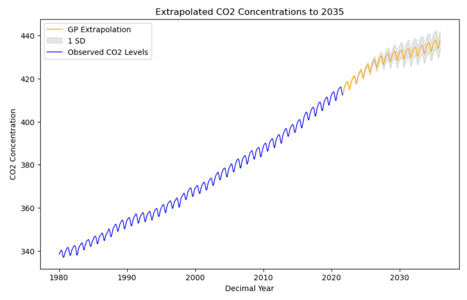


Figure 15: $\sigma = 0.1$

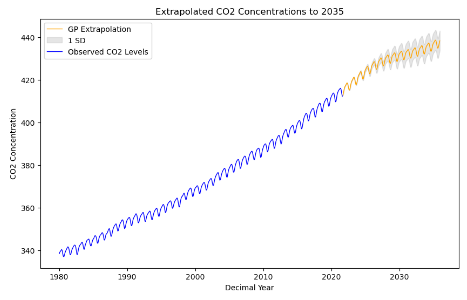


Figure 16: $\sigma = 10$

ϕ : Varying ϕ in the model shows a significant impact on predictions. With $\phi = 0.1$, the model exhibits minimal vertical variability, emphasizing a linear increase and consistent periodicity. As ϕ increases to 3 and 8, there's a notable rise in vertical fluctuations, reflecting an increase and then a decrease, possibly due to COVID-19, followed by another increase. Higher values of ϕ amplify this effect, making the model more sensitive to recent data fluctuations, including the pronounced impact of the pandemic. Additionally, the widening of the one standard deviation error bars at higher ϕ values indicates growing uncertainty in future predictions which is preferable.

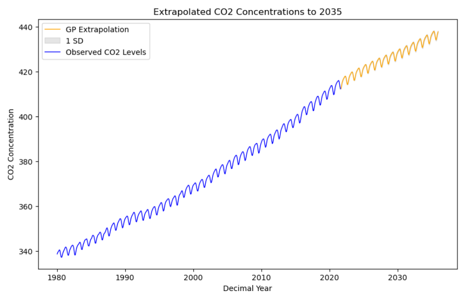


Figure 17: $\phi = 0.1$

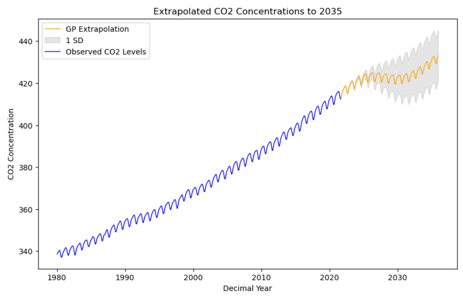


Figure 18: $\phi = 3$

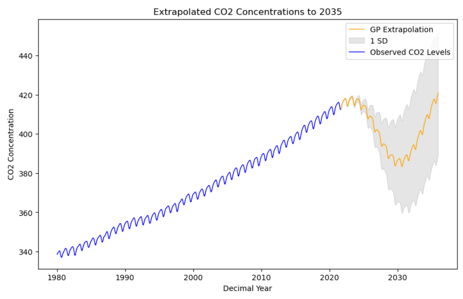


Figure 19: $\phi = 8$

η : With η set to different values in the model, the impact on predictions is notable. At $\eta = 0.1$, the model shows a uniform linear increase with consistent oscillations and standard deviation, but an abrupt decrease from the last point of observation to the first point in predictions, indicating a high sensitivity to local variations. For $\eta = 10$, the predictions resemble those with the original parameters but have narrower standard deviation error bars, suggesting a moderate sensitivity to input space changes. Most interestingly, at $\eta = 20$, the model significantly smoothens its responses, leading to an almost quadratic increase in emissions and minimal standard deviation error bars. This larger value of η causes the model to emphasize a more regular long-term trend, essentially overlooking short-term fluctuations like the temporary reduction in CO2 emissions during the COVID-19 pandemic. The negligible impact of COVID-19 in this setting indicates that the model, with a high η, prioritizes longer-term trends, projecting a trajectory for CO2 emissions as if the pandemic and its temporary effects had not occurred. This highlights the role of η in balancing the model's focus between short-term data fluctuations and the overarching, smoother long-term trends in the data.

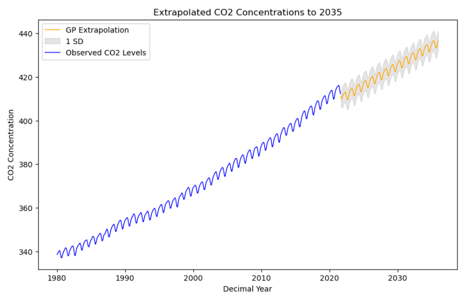


Figure 20: $\eta = 0.1$

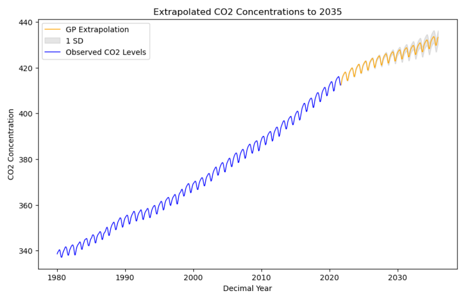


Figure 21: $\eta = 10$

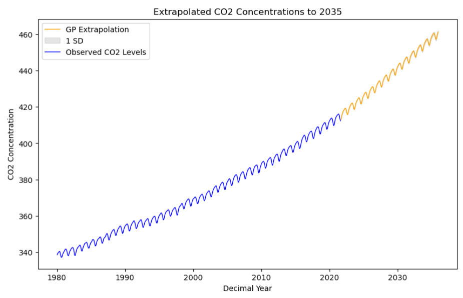


Figure 22: $\eta = 20$

ζ : At $\zeta = 0.01$, the model assumes very low noise, leading to an over-sensitivity to data structure and resulting in exaggerated, stretched periodic patterns in the predictions, akin to a stretched spring. This effect indicates potential overfitting, as the model closely fits even minor variations, potentially misinterpreting them as significant. In contrast, with $\zeta = 1$, the model acknowledges higher inherent variability or uncertainty in the observations, resulting in smoother, more cautious predictions that focus on broader trends. The increase in standard deviation error bars at this higher ζ value reflects greater uncertainty and a reduced tendency to overfit, with the model prioritizing the general trend (global increase) over detailed data fluctuations. This comparison highlights how ζ influences the model's balance between fitting the data closely and capturing overarching patterns, with low ζ leading to overfitting and high ζ to more generalized predictions.

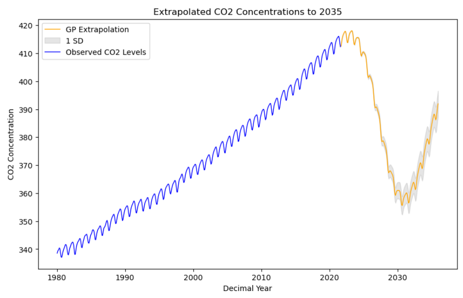


Figure 23: $\zeta = 0.01$

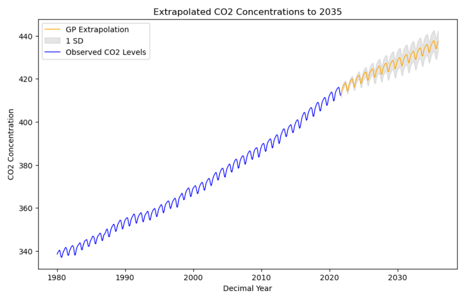


Figure 24: $\zeta = 1$

θ : With $\theta = 1$, the Gaussian Process model exhibits no additional scaling to the kernel components, leading to predictions that show a steady increasing trend with minimal standard deviation error bars, indicating a focus on the broader trend and lower sensitivity to short-term fluctuations like the COVID-19-induced decrease. Conversely, at $\theta = 10$, the amplification of both the periodic and squared exponential components results in more variable predictions, evidenced by larger standard deviation error bars and a more pronounced response to recent data changes, such as the temporary dip during the pandemic period. This suggests that a higher θ makes the model more attuned to local fluctuations and rapid changes in the data, reflecting increased uncertainty and responsiveness to short-term variations.

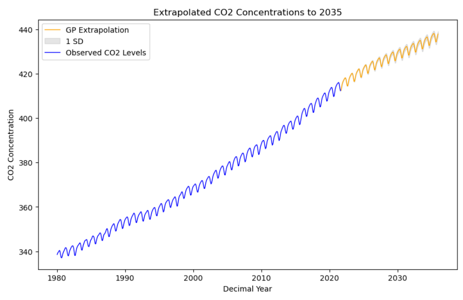


Figure 25: $\theta = 1$

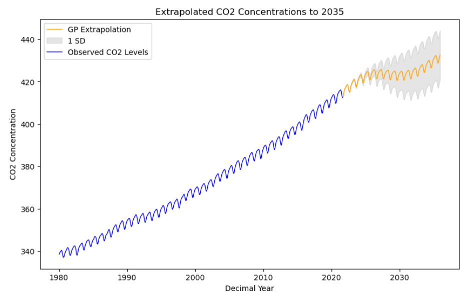


Figure 26: $\theta = 10$

2.7

The procedure described isn't fully Bayesian as it primarily utilizes MAP estimates for the model parameters, focusing on the most probable values rather than the entire range of possible values. This approach, while Bayesian in its use of priors and posterior maximization, doesn't encompass the full Bayesian paradigm of uncertainty quantification.

To model $f(t)$ in a Bayesian framework, you would directly treat $f(t)$ as a Gaussian Process defined by a suitable kernel function. This involves selecting a kernel that reflects the characteristics of the CO2 data, such as a combination of periodic and squared exponential kernels to capture both seasonal and long-term trends. Prior distributions are assigned to the kernel hyperparameters based on initial beliefs or assumptions. Gaussian Process inference is then used to compute the GP posterior distribution, considering the observed data, which provides a range of possible functions $f(t)$. For predictions, this posterior GP is used to generate predictive distributions for new points, offering both mean predictions and measures of uncertainty. This inherently accounts for uncertainty in the data and the model, and as new data becomes available, the GP model can be updated, refining its predictions over time.

3 Mean-Field learning

3.1

From lectures, we know that the factored variational E-step has general form (i.e taking the variational derivative of the lagrangian, enforcing normalization of q_i):

$$q_i(\mathcal{Z}_i) \propto \exp \left\langle \log P(\mathcal{X}, \mathcal{Z} | \theta^{k-1}) \right\rangle_{\prod_{j \neq i} q_j(\mathcal{Z}_j)}$$

Thus, in our case we have

$$q_i(s_i^n) \propto \exp \langle \log P(\mathbf{s}^n, \mathbf{x}^n|\theta)\rangle_{\prod_{j\neq i} q_j(s_j)}$$

$$\Rightarrow\; q_i(s_i^n) \;\propto\; \exp \left[\langle \log P(\mathbf{x}^n|\mathbf{s}^n, \theta\rangle_{\prod_{j\neq i} q_j(s_j)} + \langle \log P(\mathbf{s}^n|\theta\rangle_{\prod_{j\neq i} q_j(s_j)}\right]$$

Now,

$$\langle \log P(\mathbf{s}^n|\theta\rangle_{\prod_{j\neq i} q_j(s_j)} = \left\langle \sum_{j=1}^{K} s_j^n \log \pi_j + (1 - s_j^n) \log 1 - \pi_j \right\rangle_{\prod_{j\neq i} q_j(s_j)}$$

By linearity of expectation, and noting that $\left\langle s_j^n \right\rangle_{\prod_{j\neq i} q_j(s_j)} \;=\; \left\langle s_j^n \right\rangle_{q_j(s_j)} \;=\; \lambda_{jn}$
we get

$$\langle \log P(\mathbf{s}^n|\theta\rangle_{\prod_{j\neq i} q_j(s_j)} = s_i^n \log \pi_i + (1-s_i^n) \log 1-\pi_i + \sum_{j\neq i} \lambda_{jn} \log \pi_j + (1 - \lambda_{jn})\; \log 1 - \pi_j$$

Note that $s_i^n \log \pi_i \;+\; (1 - s_i^n) \log 1 - \pi_i \;=\; s_i^n \;\log \frac{\pi_i}{1 \,-\, \pi_i} \;+\; \log 1 \;-\; \pi_i$

The terms in $\langle \log P(\mathbf{s}^n|\theta\rangle_{\prod_{j\neq i} q_j(s_j)}$ that do not contain s_i^n will get folded in the normalizing constant after taking expectation so we will only keep the relevant terms, to get

$$\Rightarrow\; \langle \log P(\mathbf{s}^n|\theta\rangle_{\prod_{j\neq i} q_j(s_j)} \;\propto\; s_i^n \;\log \frac{\pi_i}{1 \,-\, \pi_i}$$

Now, (changed $\mathbf{x}^n$ to $\mathbf{x}_n$ for notation convenience)

$$\langle \log P\left(\mathbf{x}_n|\mathbf{s}^n, \theta)\right\rangle_{\prod_{j\neq i} q_j(s_j)} \propto -\frac{1}{2\sigma^2} \left\langle \left(\mathbf{x}_n - \sum_{j=1}^{K} s_j^n \boldsymbol{\mu}_j\right)^T \left(\mathbf{x}_n - \sum_{j=1}^{K} s_j^n \boldsymbol{\mu}_j\right) \right\rangle_{\prod_{j\neq i} q_j(s_j)}$$

$$= \; -\frac{1}{2\sigma^2} \left\langle \mathbf{x}_n^T \mathbf{x}_n - 2 \sum_{j=1}^{K} \boldsymbol{\mu}_j^T \; s_j^n \; \mathbf{x}_n \; + \; \sum_{k=1}^{K} \sum_{j=1}^{K} s_k^n s_j^n \boldsymbol{\mu}_k^T \boldsymbol{\mu}_j \right\rangle_{\prod_{j\neq i} q_j(s_j)}$$

By linearity of expectation

$$= -\frac{1}{2\sigma^2}\left(\mathbf{x}_n^T\mathbf{x}_n - 2\sum_{j=1}^{K}\boldsymbol{\mu}_j^T \left\langle s_j^n\right\rangle_{q_j(s_j)} \mathbf{x}_n + \sum_{k=1}^{K}\sum_{j=1}^{K}\left\langle s_k^n s_j^n\right\rangle_{q_k(s_k)q_j(s_j)} \boldsymbol{\mu}_k^T\boldsymbol{\mu}_j\right)$$

again, we will drop all terms that are constant with respect to s_i^n as they are folded into the normalizing constant :

$$-2\sum_{j=1}^{K}\boldsymbol{\mu}_j^T \left\langle s_j^n\right\rangle_{q_j(s_j)} \mathbf{x}_n = -2\sum_{j=1}^{K}\boldsymbol{\mu}_j^T \lambda_{jn}\,\mathbf{x}_n \propto -2\boldsymbol{\mu}_i^T s_i^n\mathbf{x}_n$$

The third term in in (1) is equal to

$$\sum_{k\neq i}^{K}\sum_{k\neq i\, j\neq k}^{K}\left\langle s_k^n s_j^n\right\rangle_{q_k(s_k)q_j(s_j)} \boldsymbol{\mu}_k^T\boldsymbol{\mu}_j + 2\,s_i^n\sum_{j\neq i}^{K}\left\langle s_j^n\right\rangle_{q_j(s_j)} \boldsymbol{\mu}_i^T\boldsymbol{\mu}_j + \sum_{j\neq i}^{K}\left\langle (s_j^n)^2\right\rangle_{q_j(s_j)} \boldsymbol{\mu}_j^T\boldsymbol{\mu}_j + (s_i^n)^2\boldsymbol{\mu}_i^T\boldsymbol{\mu}_i$$

Dropping all the irrelevant terms we get,

$$\propto 2\,s_i^n\sum_{j\neq i}^{K}\left\langle s_j^n\right\rangle_{q_j(s_j)} \boldsymbol{\mu}_i^T\boldsymbol{\mu}_j + (s_i^n)^2\boldsymbol{\mu}_i^T\boldsymbol{\mu}_i$$

Now, note that the latents are binary thus $s_i^2 = s_i \;\forall\, i$,it is also noted in 5)a). Thus, the term above is equal to

$$2\,s_i^n\sum_{j\neq i}^{K}\lambda_{jn}\,\boldsymbol{\mu}_i^T\boldsymbol{\mu}_j + s_i^n\,\boldsymbol{\mu}_i^T\boldsymbol{\mu}_i$$

$$\Rightarrow \left\langle \log P\left(\mathbf{x}_n|\mathbf{s}^n,\theta\right)\right\rangle_{\prod_{j\neq i} q_j(s_j)} \propto -\frac{s_i^n}{2\sigma^2}\left(-2\boldsymbol{\mu}_i^T\mathbf{x}_n + 2\sum_{j\neq i}^{K}\lambda_{jn}\,\boldsymbol{\mu}_i^T\boldsymbol{\mu}_j + \boldsymbol{\mu}_i^T\boldsymbol{\mu}_i\right)$$

Now regrouping the expectation of log-likelihood and log-prior we get

$$\log q_i(s_i^n) \propto s_i^n\left[\log\frac{\pi_i}{1-\pi_i} - \frac{1}{2\sigma^2}\left(-2\boldsymbol{\mu}_i^T\mathbf{x}_n + 2\sum_{j\neq i}^{K}\lambda_{jn}\,\boldsymbol{\mu}_i^T\boldsymbol{\mu}_j + \boldsymbol{\mu}_i^T\boldsymbol{\mu}_i\right)\right] \quad (1)$$

We're approximating the posterior distribution over the hidden variables by distribution :

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

$$\Rightarrow \log q_i(s_i^n) = \log\left(\lambda_{in}^{s_i^n} (1 - \lambda_{in})^{(1-s_i^n)}\right) = s_i^n \log\frac{\lambda_{in}}{1-\lambda_{in}} + \log 1 - \lambda_{in} \propto s_i^n \log\frac{\lambda_{in}}{1-\lambda_{in}} \quad (2)$$

Matching the terms in (1) (i.e the terms we kept as they are factored with s_i^n) and (2) we get

$$\log\frac{\lambda_{in}}{1-\lambda_{in}} = \left[\log\frac{\pi_i}{1-\pi_i} - \frac{1}{2\sigma^2}\left(-2\boldsymbol{\mu}_i^T\mathbf{x}_n + 2\sum_{j\neq i}^{K}\lambda_{jn}\,\boldsymbol{\mu}_i^T\boldsymbol{\mu}_j + \boldsymbol{\mu}_i^T\boldsymbol{\mu}_i\right)\right]$$

Now, $\log\frac{y}{1-y} = x \Leftrightarrow y = \sigma(x)$ where $\sigma(x) = \frac{1}{1+\exp -x}$ i.e sigmoid function.

$$\Rightarrow \lambda_{in} = \sigma\left(\log\frac{\pi_i}{1-\pi_i} + \frac{1}{\sigma^2}\boldsymbol{\mu}_i^T(\mathbf{x}_n - \sum_{j\neq i}^{K}\lambda_{jn}\,\boldsymbol{\mu}_j) - \frac{1}{2\sigma^2}\boldsymbol{\mu}_i^T\boldsymbol{\mu}_i\right)$$

Let's derive the Free energy. As data is i.i.d we will compute the free for each n points add aggregate them during VE-step, by definition

$$\mathcal{F}_n(\lambda^n, \theta) = \langle \log P(\mathbf{s}^n, \mathbf{x}^n|\theta)\rangle_{q(\mathbf{s}^n)} - \langle \log q(\mathbf{s}^n)\rangle_{q(\mathbf{s}^n)}$$

The expected log-joint (LHS) is given by

$$\left\langle \sum_{i=1}^{K} s_i^n \log\frac{\pi_i}{1-\pi_i} + \log 1 - \pi_i - \frac{D}{2}\log 2\pi\sigma^2 - \frac{1}{2\sigma^2}\left(\mathbf{x}_n^T\mathbf{x}_n - 2\sum_{j=1}^{K}\boldsymbol{\mu}_j^T s_j^n\,\mathbf{x}_n + \sum_{k=1}^{K}\sum_{j=1}^{K} s_k^n s_j^n \boldsymbol{\mu}_k^T\boldsymbol{\mu}_j\right)\right\rangle$$

Where the expectation above is w.r.t $q(\mathbf{s}^n)$. Now by linearity of expectation, the fact that $\langle s_i^n\rangle_{q(\mathbf{s}^n)} = \lambda_{in}$, and $\langle s_i^n s_j^n\rangle_{q(\mathbf{s}^n)} = \langle s_i^n\rangle_{q_i(s_i^n)}\langle s_j^n\rangle_{q_j(s_j^n)} =$

$\lambda_{in}\,\lambda_{jn}$ (due to fully factored Mean-field assumption) and that $\langle\, s_i^n\, s_i^n \,\rangle_{q(\mathbf{s}^n)} = \langle\, s_i^n \,\rangle_{q_i(s_i^n)} = \lambda_{in}$, we get expected log-joint to be equal to

$$\sum_{i=1}^{K} \lambda_{in}\, \log \frac{\pi_i}{1-\pi_i} + \log(1-\pi_i) - \frac{D}{2}\log(2\pi\sigma^2)$$
$$-\frac{1}{2\sigma^2}\left(\mathbf{x}_n^T\mathbf{x}_n - 2\sum_{i=1}^{K}\boldsymbol{\mu}_i^T\lambda_{in}\mathbf{x}_n + \sum_{i=1}^{K}\sum_{j\neq i}^{K}\lambda_{in}\lambda_{jn}\boldsymbol{\mu}_i^T\boldsymbol{\mu}_j + \sum_{i=1}^{K}\lambda_{in}\,\boldsymbol{\mu}_i^T\boldsymbol{\mu}_j\right)$$

$$\Leftrightarrow \sum_{i=1}^{K} \lambda_{in}\, \log \frac{\pi_i}{1-\pi_i} + \log(1-\pi_i) - \frac{D}{2}\log(2\pi\sigma^2)$$
$$-\frac{1}{2\sigma^2}\left[(\mathbf{x}_n - \boldsymbol{\mu}\boldsymbol{\lambda}_n)^T(\mathbf{x}_n - \boldsymbol{\mu}\boldsymbol{\lambda}_n) + \sum_{i=1}^{K}(\,\lambda_{in} - \lambda_{in}^2\,)\,\boldsymbol{\mu}_i^T\boldsymbol{\mu}_i\right]$$

With $\boldsymbol{\mu} \in \mathbb{R}^{D\times K}$ and $\boldsymbol{\lambda}_n$ is a column vector of K dimension, i.e transpose of the nth row of matrix lambda. We chose to represent the right-hand side of the term above in this form for computational efficiency, as computing the vectorized quadratic term and then correcting the squared lambda's is more attractive computationally speaking.

Now the entropy term in free energy is

$$-\,\langle\,\log\, q(\mathbf{s}^n)\,\rangle_{q(\mathbf{s}^n)} = -\sum_{i=1}^{K}\lambda_{in}\,\log\,\lambda_{in} + (\,1-\lambda_{in}\,)\,\log\,1-\lambda_{in}$$

Finally, by adding both terms and factorizing we get

$$\mathcal{F}_n(\,\lambda^n\,,\,\theta\,) = \sum_{i=1}^{K}\lambda_{in}\,\log\,\frac{\pi_i}{\lambda_{in}} + (1-\lambda_{in})\,\log\,\frac{1-\pi_i}{1-\lambda_{in}} - \frac{D}{2}\log(2\pi\sigma^2)$$
$$-\frac{1}{2\sigma^2}\left[(\mathbf{x}_n - \boldsymbol{\mu}\boldsymbol{\lambda}_n)^T(\mathbf{x}_n - \boldsymbol{\mu}\boldsymbol{\lambda}_n) + \sum_{i=1}^{K}(\,\lambda_{in} - \lambda_{in}^2\,)\,\boldsymbol{\mu}_i^T\boldsymbol{\mu}_i\right]$$

Code for question a)

```python
import numpy as np
from numpy.random import shuffle
import matplotlib.pyplot as plt
from scipy.special import expit as stable_sigmoid

def compute_free_energy_n(x_n, mu, sigma, pie, lambda_n):
    D, K = mu.shape

    pie = pie.ravel()
    cst = 1e-10

    # First Term
    # Added small cst to avoid nan values and computational stability
    term1 = np.sum(lambda_n * (np.log(pie + cst ) - np.log(lambda_n + cst)) + (1 - lambda_n) * (np.log(1 - pie + cst) - np.log(1 - lambda_n + cst)))

    # Second Term
    term2 = -(D / 2) * np.log(2 * np.pi * sigma**2)

    # Third Term
    mu_lambda = mu @ lambda_n  # Efficient computation of sum of mu_i * lambda_in
    term3_a = np.dot((x_n - mu_lambda) , (x_n - mu_lambda))
    term3_b = np.sum((lambda_n - lambda_n**2) * np.sum(mu**2, axis=0))
    term3 = -(1 / (2 * sigma**2)) * (term3_a + term3_b)

    return term1 + term2 + term3

def MeanField(X, mu, sigma, pie, lambda0, maxsteps):
    N, D = X.shape
    _, K = mu.shape
    Lambda = np.copy(lambda0)
    F = 0

    pie = pie.ravel()

    for n in range(N):
        lambda_n = Lambda[n, :]
        F_n_old = -np.inf

        for step in range(maxsteps):
            # Generate a shuffled list of indices from 0 to K-1
            indices = list(range(K))
            shuffle(indices)

            for i in indices: # any schedyle of VE_i updates will increase free energy
                sum_except_i = sum(lambda_n[j] * mu[:, j] for j in range(K) if j != i)
                lambda_n[i] = stable_sigmoid(
                    np.log(pie[i] / (1 - pie[i])) +
                    (1 / sigma**2) * mu[:, i].T @ (X[n, :] - sum_except_i) -
                    (1 / (2 * sigma**2)) * np.linalg.norm(mu[:, i])
**2
                )

            F_n = compute_free_energy_n(X[n, :], mu, sigma, pie,
lambda_n)

            # Check that F_n is not smaller than F_n_old
            if F_n < F_n_old:
                print(f"Free energy decreased for data point {n} at
 iteration {step}.")
            F_n_old = F_n

            # Check for convergence for this data point
            if np.abs(F_n - F_n_old) < 1e-7:
                break
            F_n_old = F_n

        Lambda[n, :] = lambda_n
        F += F_n

    return Lambda, F
```

3.2

In M-step, the latent variables are treated as fixed (i.e observed) inputs, thus the In the derived M-step for our model, the update for the mean parameter $\boldsymbol{\mu}$ has a form reminiscent of the Ordinary Least Squares (OLS) estimation in linear regression. Specifically, the computation of $\boldsymbol{\mu}$ as $(\text{inv}(ESS) \cdot ES^\top \cdot X)^\top$ mirrors the process of estimating regression coefficients, where the latent variables expected values ES and their covariances ESS play roles analogous to the design matrix and its transpose in regression. The update for σ^2 , resembling the calculation of residual variance in linear regression, further aligns the M-step with regression analysis. It incorporates the difference between observed data and its reconstruction from the model, similar to how linear regression quantifies the spread of residuals. For the parameter π , updated as the mean of ES, there isn't a direct linear regression counterpart, but it essentially reflects the average activation of each latent factor across all data points, indicating their relative importance in the model. These parallels arise because, in the M-step, latent variables are treated as fixed (based on their expected values, akin to observed inputs in regression), under the framework of a linear relationship between the means $\boldsymbol{\mu}_i$ (akin to weights) and the latent variables, and the assumption of homoscedastic Gaussian noise, analogous to linear regression's error structure.

3.3

We have $\mathrm{ESS} \in \mathbb{R}^{K\times K}$, $\mathrm{ES} \in \mathbb{R}^{N\times K}$, $X \in \mathbb{R}^{N\times D}$, $\mu \in \mathbb{R}^{D\times K}$

For $\boldsymbol{\mu}$:

$\mathrm{inv}(\mathrm{ESS}) = K \times K$ matrix inversion $\Rightarrow O(K^3)$
$(\mathrm{ES}^\top)X = (K \times N) \times (N \times D) \Rightarrow O(KND)$
$\mathrm{inv}(\mathrm{ESS})(\mathrm{ES}^\top X) = (K \times K) \times (K \times D) \Rightarrow O(K^2D)$

$\Rightarrow \boldsymbol{\mu} = O(K^3) + O(KND) + O(K^2D)$

(chose this way of expanding for computational efficiency as we're going to reuse $\mathrm{ES}^\top X$ in later calculations)

For σ^2:

$\mathrm{trace}(X^\top X) =$ sum of squares of all elements in X, but assume computer doesn't know that $\mathrm{trace}(X^\top X) = O(D^2N)$ as $X^\top X$ is $= (D \times N) \times (N \times D)$.

$\boldsymbol{\mu}^\top\boldsymbol{\mu} = (K\times D)\times(D\times K) = O(K^2D) \Rightarrow \mathrm{trace}(\boldsymbol{\mu}^\top\boldsymbol{\mu}\ \mathrm{ESS}) = \sum_i \sum_j (\boldsymbol{\mu}^\top\boldsymbol{\mu})_{ij}(\mathrm{ESS})_{ij} = O(K^2D)$ (noting that here $\boldsymbol{\mu}^\top\boldsymbol{\mu}$ has already been calculated)

$\Rightarrow \mathrm{trace}(\boldsymbol{\mu}^\top\boldsymbol{\mu}\ \mathrm{ESS}) = O(K^3) + O(K^2D) = O(K^3)$

Assuming we've already calculated $\mathrm{ES}^\top X$ for $\boldsymbol{\mu}$ calculations then :

$(\mathrm{ES}^\top X)\boldsymbol{\mu} = (K \times D) \times (D \times K) = O(K^2D) \Rightarrow \mathrm{trace}((\mathrm{ES}^\top X)\boldsymbol{\mu}) = O(K^2D)$ as trace is just a sum over diag i.e., $O(K)$ so gets overwritten by $O(K^2D)$.

$\Rightarrow \sigma^2 = O(D^2N) + O(K^3) + O(K^2D)$

Lastly, π is $O(NK)$ as it takes the mean over $N \times K$ elements.

$\Rightarrow$ M-step $= O(K^3{+}KND{+}K^2D{+}D^2N{+}NK) = O(K^3{+}KND{+}K^2D{+}D^2N)$

3.4

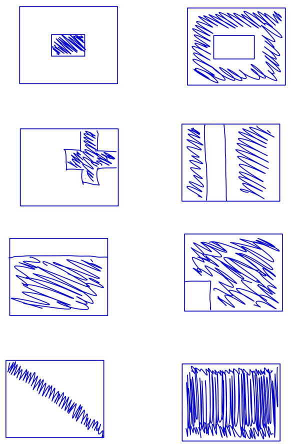


Figure 27: Identified features, for cross and diagonals the colours should be inverted. After looking at genimage I got one feature wrong (should have right column instead of all black square)

After looking at data, I identified at least 8 different distinct features (see appendix), each appearing in a binary state, by binary I mean either fully present of completely absent. These features are clearly recognizable when present in an image and maintain a consistent appearance, meaning their shape and the pixels they comprise do not vary. When features overlap, the overlapping pixels display a specific intermediate intensity. Another observation is that having more than two features (at least the ones I have identified) present in one image is rare, on average, visually speaking, there are about two features present in an image.

Factor analysis might not be the most effective model for this particular dataset. FA assumes that the observed data are linear combinations of Gaussian latent factors, with added Gaussian noise. However, the dataset exhibits features that are binary in nature (they are either present or not) and interact deterministically. This deterministic and binary nature of feature presence and combination does not align well with the Gaussian assumptions inherent in FA. FA is more suited to scenarios where the data exhibits continuous variation and the overlapping of features results in a probabilistic blending, which seems not to be the case here.

ICA, on the other hand, appears to be a more suitable choice for this dataset. ICA is designed to separate statistically independent, non-Gaussian components within the data. The fact that the features in the images are distinct and maintain their integrity when combined aligns well with the capabilities of ICA. The method is particularly effective when components are non-Gaussian and independent, as seems to be the case with the features in these images. ICA's ability to identify and separate these independent components makes it a strong candidate for modeling this data, especially given the deterministic way the features combine.

As for the MoG, it may not be the optimal choice for this dataset unless the images form distinct subgroups or clusters, each characterized by different combinations of features. MoG is adept at modeling such subpopulations within the data. However, the focus here is on identifying individual features and their specific interactions, so MoG might not capture the essence of the data as effectively as ICA, especially considering the deterministic interactions and the binary nature of the feature presence.

3.5

Code :

```
def LearnBinFactors(X, K, iterations):
    N, D = X.shape
    # Initialize parameters
    mu =  np.random.rand(D, K)
    sigma = np.random.rand() * 0.1
    pie = np.random.rand(1, K)

    # Initialize lambda0 randomly or with zeros
    lambda0 = np.random.rand(N, K)

    F_old = -np.inf
    F_values = []

    for iter in range(iterations):
        # E-step: Update Lambda and compute free energy
        Lambda, F = MeanField(X, mu, sigma, pie, lambda0, 200)
        F_values.append(F)

        # Convergence check
        if iter > 0 and np.abs(F - F_old) < 1e-6:
            print(f"Convergence reached at iteration {iter}")
            break
        F_old = F

        # M-step: Update mu, sigma, and pie
        ES = Lambda

        ESS = np.einsum('ni,nj->ij', Lambda, Lambda)
        diagonal_correction = np.sum(Lambda - Lambda**2, axis=0)
        np.fill_diagonal(ESS, np.diag(ESS) + diagonal_correction)

        mu, sigma, pie = m_step(X, ES, ESS)

        # Update lambda0 for the next iteration
        lambda0 = Lambda

    return mu, sigma, pie
```

3.6

These are the features we get when running LearnBinFactors with random initializations:

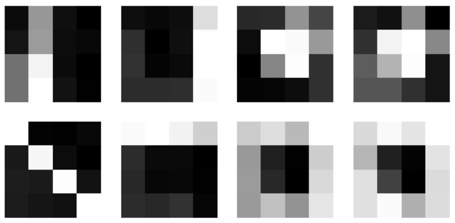


Figure 28:

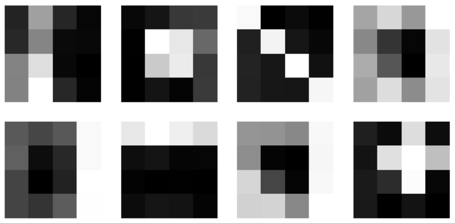


Figure 29:

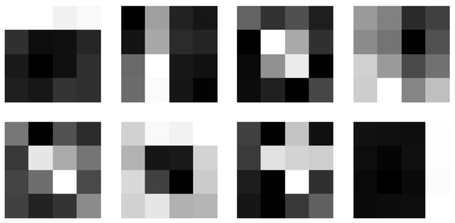


Figure 30:

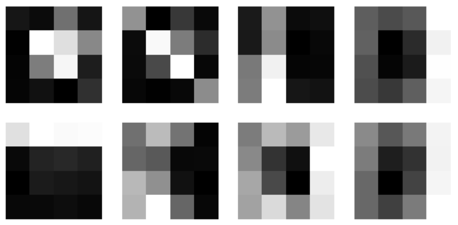


Figure 31:

In general, the convergence is pretty fast (40 to 70 iterations) and occasionally goes up 100-150 but the majority of the time the features learned are pretty decent, even though there is always at least 1 feature that combines two sperate features. Albeit, there is a lack of consistency in the results, with some redundancy present in all of our converged results which is likely due to the random initializations and the mean-field approximation.

I've determined that setting K to 8 is appropriate for our model, as there appear to be 8 distinct features present in the dataset. With a lower K, I tend to get the same 8 solutions, but only 7 are chosen each time, indicating a loss of feature representation. With a higher K, I observe too much redundancy among the features. Therefore, K=8 seems to be the right balance, giving a minimal amount of redundancy and ensuring that the most amount of distinct features are captured (usually 7 out of 8).

For the initialization of μ, I've opted for a completely random approach, assigning values between 0 and 1. This decision is based on the observation that most features occupy a small portion of the image and have complex patterns, like the cross shape, which are not easily captured with a uniform or targeted initialization. Setting values randomly across the entire range from 0 to 1 offers an efficient exploration of the feature space. This approach avoids biases towards lower or higher pixel values, which is essential since most features are defined by higher pixel values forming the shape against a background of zeros. Hence, random initialization across the full range provides a balanced start.

Regarding σ, I've decided to go with random values between 0 and 0.1. Given that the dataset comprises greyscale images with pixel values ranging from 0 to 1, a standard deviation greater than 0.1 would result in notable changes in pixel intensities, which is not consistent with the observed data. Thus, initializing σ within this range seems to be a reasonable estimate of the noise level in the dataset.

Regarding π, my initial observation suggested that some features might appear slightly more frequently than others. However, upon further experimentation, I found that initializing π with random values between 0.2 and 0.3, based on the assumption of average feature presence, did not yield consistently good results. It appears that this approach potentially constrains the exploration space too much, leading to less effective learning. Therefore, I have opted for a fully random initialization for π, which allows for a broader and more flexible exploration of feature activation probabilities. This random approach aligns better with the diverse and variable nature of our dataset, ensuring that the model doesn't prematurely converge to suboptimal solutions due to restrictive initial assumptions. Here are some examples of learned features with π between 0.2 and 0.3

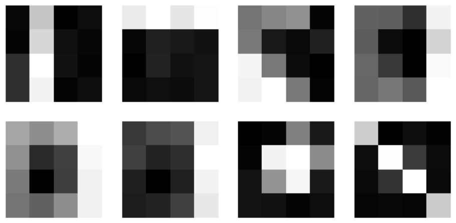


Figure 32:

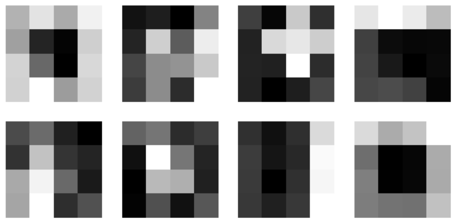


Figure 33:

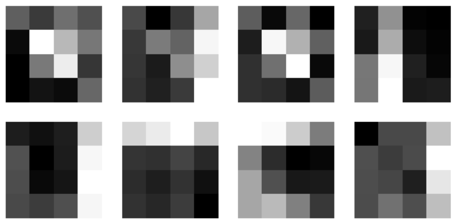


Figure 34:

For λ, the variational approximation to the posterior probability, I've chosen a different approach. Since λ is updated for each data point and is specific to each image, it's more effective to allow for a broader exploration of feature presence. Thus, I've initialized λ with random values between 0 and 1 for each element. While on average, there are about two features per image, setting λ uniformly across all images to reflect this average (as is done with π) is not ideal because λ is image-specific. A random initialization between 0 and 1 for λ allows the EM algorithm to explore feature associations more freely for each specific image, leading to better model performance as evidenced by the results

After reviewing our model's initialization strategies, it's clear that random initialization works surprisingly well. I have tried to initialize with different initializations for μ, λ and σ, even if it should theoretically give worse results, and it did. This really highlights the strength of random initialization in handling complex datasets like ours.

3.7

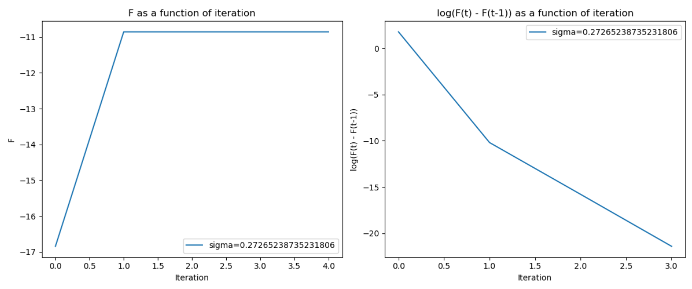


Figure 35: Convergence for sigma found after running LearnBinFactors

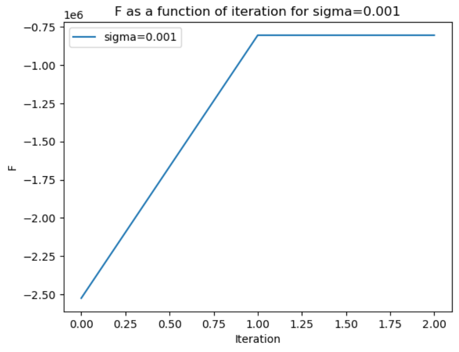


Figure 36: Free energy sigma 0.001

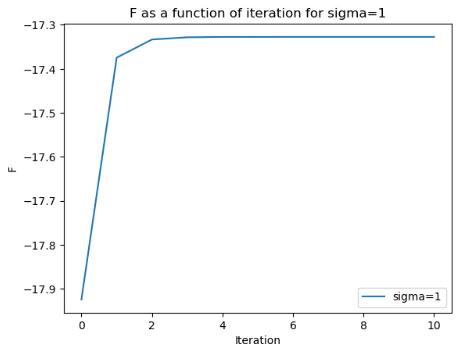


Figure 37: Free energy sigma 1

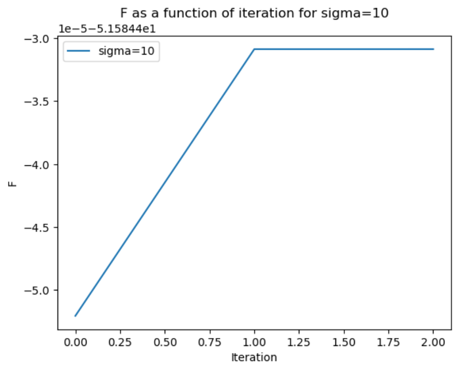


Figure 38: Free energy sigma 10

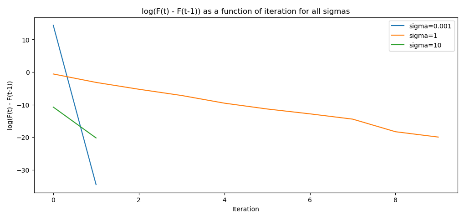


Figure 39: $\log F(t) - F(t-1)$

The convergence rate of the variational approximation is deeply influenced by the choice of sigma, the parameter determining the breadth of the Gaussian distribution in our latent factor model. The free energy, which serves as a lower bound to the log-likelihood, exhibits a marked dependency on sigma which can be observed in the previous plots.

For sigma set to 0.001, convergence is notably swift, occurring in merely two iterations. This rapidity is a consequence of the excessively peaked nature of the Gaussian distribution at such a low sigma, creating a highly deterministic system. With limited flexibility, the optimization rapidly descends towards a local maximum, given the steep gradients in the free energy landscape. The resultant free energy is profoundly negative, reflecting the improbability of data points under such a constrained distribution, hence widening the gap between the free energy and the log-likelihood.

As sigma increases to 1, aligning more closely with the true variance of the data obtained after running LearnBinFactors, the convergence moderates, unfolding across eight steps. The free energy function presents a complex optimization problem characterized by multiple local maximums. The presence of these local extrema is a result of the variational distribution's increased capacity to capture the probabilistic structure of the data without being overly constrained or excessively diffuse. The optimization path is therefore not direct and requires multiple iterations to ascertain the global optimum that maximizes the free energy. The resultant free energy, less negative in this instance, intimates a more accurate approximation of the posterior distribution. The optimization process balances the model's precision against its generalizability, tightening the bound on the log-likelihood.

Further elevation of sigma to 10 once again precipitates a precipitous convergence. Such an expansive sigma induces a flat Gaussian distribution, lacking in discriminative power and suggesting uniform plausibility across a vast range of the data. This creates a shallow free energy landscape with a diffuse optimum, which the Variational E-step can locate with haste. However, the near-zero free energy attained signifies a distribution that is so unselective that it scarcely penalizes deviations from the mean, thus failing to offer a meaningful approximation of the data distribution or to snugly bound the log-likelihood.

The observed convergence behavior underscores the sensitivity of the variational E-step to sigma. At extremes, whether diminutive or excessive, sigma induces an optimization landscape that is less challenging to navigate, fostering rapid convergence. However, such rapid convergence does not equate to the accuracy of the model; rather, it is indicative of the model's underfitting or overfitting tendencies.

Code :

```python
sigmas = [0.001, 1, 10]

def MeanField_single_point(X, mu, pie, maxsteps, sigmas):
    _, K = mu.shape
    Lambda_ini = np.random.rand(1, K)
    pie = pie.ravel()

    all_delta_F_log = []

    for sigma in sigmas:
        lambda_n = Lambda_ini[0, :]
        F_values = []
        delta_F_log = []
        F_n_old = -np.inf

        for step in range(maxsteps):
            indices = list(range(K))
            np.random.shuffle(indices)

            for i in indices:
                sum_except_i = sum(lambda_n[j] * mu[:, j] for j in range(K) if j != i)
                lambda_n[i] = stable_sigmoid(
                    np.log(pie[i] / (1 - pie[i])) +
                    (1 / sigma**2) * mu[:, i].T @ (X[0, :] - sum_except_i) -
                    (1 / (2 * sigma**2)) * np.linalg.norm(mu[:, i])**2
                )

            F_n = compute_free_energy_n(X[0, :], mu, sigma, pie, lambda_n)
            F_values.append(F_n)

            if step > 0:
                delta_F = np.abs(F_n - F_n_old)
                delta_F_log.append(np.log(delta_F + 1e-15))
                if delta_F < 1e-8:
                    print(f"Convergence reached at iteration {step} for sigma={sigma}")
                    break

            F_n_old = F_n

        # Plot for Free Energy F(t) for each sigma
        plt.figure()
        plt.plot(F_values, label=f'sigma={sigma}')
        plt.title(f'F as a function of iteration for sigma={sigma}')
        plt.xlabel('Iteration')
        plt.ylabel('F')
        plt.legend()
        plt.show()

        all_delta_F_log.append((sigma, delta_F_log))

    # Plot for Logarithmic Change in Free Energy log(F(t) - F(t-1)) for all sigmas together
    plt.figure(figsize=(12, 5))
    for sigma, delta_F_log in all_delta_F_log:
        plt.plot(delta_F_log, label=f'sigma={sigma}')
    plt.title('log(F(t) - F(t-1)) as a function of iteration for
    all sigmas')
    plt.xlabel('Iteration')
    plt.ylabel('log(F(t) - F(t-1))')
    plt.legend()
    plt.show()
```

4 Variational Bayes for binary factors

4.1

In this question, we will apply ARD for unsupervised learning using VB to determine K. We will follow a similar approach as discussed in the lectures, by first setting a column-wise prior to μ, where $\mu \in \mathbb{R}^{D\times K}$, $\mu_i \in \mathbb{R}^{D}$, i.e

$$\mu_i \sim \mathcal{N}(0, \alpha_i^{-1} I) = \mathcal{N}(0, \Lambda_i^{-1}), \forall i$$

First, we will derive a VB-"EM". As we saw, and derived in the lectures, coordinates maximization of the VB free energy lower bound leads to EM-like updates where

VB E-like step minimizes KL$[\, q(\mathbf{s}) \,|\, \text{everything else}]$ by setting

$$q(\mathbf{s}) \propto \exp\left\langle \log P(\mathbf{s}, \mathbf{x}, \mu \,|\, \pi, \sigma^2, \boldsymbol{\alpha}) \right\rangle_{q(\mu)}$$

Where, $\boldsymbol{\alpha} = \{\alpha_i\}_{i=1}^{K}$ and here we only treat parameter μ as a latent now and keep π and σ^2 to be learned point-wise and not in a bayesian way. Were only approximating the posterior distribution of μ. Now, only keeping the relevant terms we have

$$q(\mathbf{s}) \propto \exp\left\langle \log P(\mathbf{s}, \mathbf{x} \,|\, \mu, \pi, \sigma^2, \boldsymbol{\alpha}) \right\rangle_{q(\mu)}$$

Now, the approximate posterior has a fully-factored form of bernouilli's as we know from question 3. Thus, we will derive the approximate posterior for each $q_i(s_i)$, i.e

$$q_i(s_i) \propto \exp\left\langle \log P(s_i, \mathbf{x} \,|\, \mu, \pi, \sigma^2, \boldsymbol{\alpha}) \right\rangle_{q(\mu)}$$

As observed in question 5, the log-joint distribution belongs to the joint exponential family, characteristic of a Boltzmann machine. Given this, and considering that the approximate posterior distribution is fully factorized, the E-like step in VB analysis mirrors the regular E-step found in standard EM. In essence, the VB posterior is equivalent to the posterior derived in question 3)a), but adjusted to account for the expected natural parameters of the prior distribution of μ. This adjustment aligns with the discussions and derivations presented in our lectures on VB. Therefore, we can apply the results from question 3)a) to our VB analysis, with the modification of incorporating expectations relative to μ's prior :

$$\Rightarrow \lambda_i = \left\langle \sigma\left(\log\frac{\pi_i}{1-\pi_i} + \frac{1}{\sigma^2}\mu_i^T(\mathbf{x} - \sum_{j\neq i}^{K}\lambda_j\,\mu_j) - \frac{1}{2\sigma^2}\mu_i^T\mu_i \right)\right\rangle_{q(\mu)}$$

Now, as each column-wise priors are independent, we get, by linearity of expectation

$$\Rightarrow \lambda_i = \sigma\left(\log\frac{\pi_i}{1-\pi_i} + \frac{1}{\sigma^2}\,\langle\mu_i\rangle^T_{q(\mu_i)}\,(\mathbf{x} - \sum_{j\neq i}^{K}\lambda_j\,\langle\mu_j\rangle_{q(\mu_j)}) - \frac{1}{2\sigma^2}\,\langle\mu_i^T\,\mu_i\rangle_{q(\mu_i)} \right) \tag{1}$$

The log-joint is indeed jointly exponential but it is not in the Conjugate-Exponential family. Boltzmann machines do not possess simple conjugacy, the prior over columns is a conjugate prior to the likelihood but not to the entire joint distribution. Thus, Partial VB-M step sets,

$$q_i(\mu_i) \propto \mathcal{N}(0, \Lambda_i^{-1}) \times \exp\sum_{n=1}^{N}\left\langle \log P(\mathbf{s}_n, \mathbf{x}_n \,|\, \mu, \pi, \sigma^2, \boldsymbol{\alpha})\right\rangle_{q(\mathbf{s}_n)\prod_{j\neq i}^{K} q_j(\mu_j)}$$

(data is i.i.d)

(The update of $q(\mu_i)$ necessitates accounting for the interdependencies among all model parameters and latent variables. Therefore, the update is derived by considering expectations with respect to both $q(\mathbf{s})$ and all other $q(\mu_i)$ for $j \neq i$.)

$$\propto \exp\ \mathcal{N}(0, \Lambda_i^{-1}) \times \exp\sum_{n=1}^{N}\left\langle \log P(\mathbf{x}_n \mid \mathbf{s}_n, \mu, \pi, \sigma^2, \boldsymbol{\alpha})\right\rangle_{q(\mathbf{s}_n)\prod_{j\neq i}^{K} q_j(\mu_j)}$$

Threw away the prior on latents as it becomes constant with respect μ_i after taking the expectation (gets folded into the normalizing constant).

$$\Rightarrow\ q_i(\mu_i) \propto \exp(-\frac{1}{2}\mu_i^T \Lambda_i \mu_i) \exp \sum_{n=1}^{N} \left\langle -\frac{1}{2\sigma^2} (\mathbf{x}_n - \mu \mathbf{s}_n)^T (\mathbf{x}_n - \mu \mathbf{s}_n) \right\rangle_{q(\mathbf{s}_n) \prod_{j \neq i}^{K} q_j(\mu_j)}$$

$$= \exp(-\frac{1}{2}\mu_i^T \Lambda_i \mu_i) \exp - \frac{1}{2\sigma^2} \sum_{n=1}^{N} \langle (\mathbf{x}_n^T \mathbf{x}_n - 2\mathbf{s}_n^T \mu^T \mathbf{x}_n + \mathbf{s}_n^T \mu^T \mu \mathbf{s}_n) \rangle_{q(\mathbf{s}_n) \prod_{j \neq i}^{K} q_j(\mu_j)}$$

Define $\langle \mu_j \rangle_{q_j(\mu_j)} := m_{\mu_j}$. Now using the fact that $s_i^2 = s_i$, $\langle s_i \rangle_{q(s_i)} = \lambda_i \ \forall i$, and throwing away terms that are constant with respect to μ_i we get

$$\Rightarrow\ q_i(\mu_i) \propto \exp(-\frac{1}{2}\mu_i^T \Lambda_i \mu_i - \frac{1}{2\sigma^2} \left(\sum_{n=1}^{N} [-2\lambda_{in}\mu_i^T \mathbf{x}_n + 2 \sum_{j \neq i} \lambda_{in} \lambda_{jn} \mu_i^T m_{\mu_j} + \lambda_{in} \mu_i^T \mu_i] \right)$$

Note that $(\sum_{n=1}^{N} \lambda_{in})\mu_i^T \mu_i = \mu_i^T ((\sum_{n=1}^{N} \lambda_{in}) \times I) \mu_i$ (where I is the D by D identity matrix).

$$= \exp(-\frac{1}{2} \left[\mu_i^T \Lambda_i \mu_i + \mu_i^T (\frac{\sum_{n=1}^{N} \lambda_{in}}{\sigma^2} \times I) \mu_i + \frac{1}{\sigma^2} \left(\sum_{n=1}^{N} [-2\lambda_{in}\mu_i^T \mathbf{x}_n + 2 \sum_{j \neq i} \lambda_{in} \lambda_{jn} \mu_i^T m_{\mu_j}] \right) \right]$$

Rearranging, we get

$$q_i(\mu_i) \propto \exp - \frac{1}{2} \left(\mu_i^T \left(\Lambda_i + (\frac{\sum_{n=1}^{N} \lambda_{in}}{\sigma^2} \times I) \right) \mu_i - 2\mu_i^T \left(\frac{1}{\sigma^2} \sum_{n=1}^{N} \lambda_{in} (\mathbf{x}_n - \sum_{j \neq i} \lambda_{jn} m_{\mu_j}) \right) \right)$$

$$\Rightarrow q_i(\mu_i) = \mathcal{N}(m_{\mu_i}, \Sigma_{\mu_i})$$

Where,

$$\Sigma_{\mu_i} = \left(\Lambda_i + \left(\frac{\sum_{n=1}^{N} \lambda_{in}}{\sigma^2} \times I \right) \right)^{-1} = \operatorname{diag} \left(\frac{\sigma^2}{\sigma^2 \alpha_i + \sum_{n=1}^{N} \lambda_{in}} \right)$$

$$m_{\mu_i} = \Sigma_{\mu_i} \left(\frac{1}{\sigma^2} \sum_{n=1}^{N} \lambda_{in} \left(\mathbf{x}_n - \sum_{j \neq i} \lambda_{jn} \, m_{\mu_j} \right) \right)$$

$$= \operatorname{diag} \left(\frac{\sigma^2}{\sigma^2 \alpha_i + \sum_{n=1}^{N} \lambda_{in}} \right) \frac{1}{\sigma^2} \left(\sum_{n=1}^{N} \lambda_{in} \left(\mathbf{x}_n - \sum_{j \neq i} \lambda_{jn} \, m_{\mu_j} \right) \right)$$

$$\Rightarrow m_{\mu_i} = \left(\frac{1}{\sigma^2 \alpha_i + \sum_{n=1}^{N} \lambda_{in}} \right) \left(\sum_{n=1}^{N} \lambda_{in} \left(\mathbf{x}_n - \sum_{j \neq i} \lambda_{jn} \, m_{\mu_j} \right) \right)$$

Let's now implement ARD in the context of unsupervised learning. This involves developing the hyperparameter M-step focused on determining the optimal values for each α_i, considering every i. The priors are independent across different dimensions, necessitating a separate optimization for each α_i.

$$\left\langle \log \mathcal{N}(0, \Lambda_i^{-1}) \right\rangle_{q_i(\mu_i)} = \left\langle \frac{D}{2} \log \alpha_i - \frac{1}{2} \mu_i^T \Lambda_i \, \mu_i \right\rangle_{q_i(\mu_i)} = \frac{D}{2} \log \alpha_i - \frac{\alpha_i}{2} \left\langle \mu_i^T \mu_i \right\rangle_{q_i(\mu_i)}$$

By properties of the multivariate gaussian distribution $\left\langle \mu_i^T \mu_i \right\rangle_{q_i(\mu_i)}$ is equal to

$$\operatorname{Tr}(\Sigma_{\mu_i}) + m_{\mu_i}^T m_{\mu_i}$$

Thus,

$$\left\langle \log \mathcal{N}(0, \Lambda_i^{-1}) \right\rangle_{q_i(\mu_i)} = \frac{D}{2} \log \alpha_i - \frac{\alpha_i}{2} \left(\operatorname{Tr}(\Sigma_{\mu_i}) + m_{\mu_i}^T m_{\mu_i} \right)$$

Now the dependency of Σ_{μ_i} and m_{μ_i} on α_i is implicit through the precision matrix $\Lambda_i = \operatorname{diag}(\alpha_i)$. However, during the optimization of α_i, we treat the current estimates of Σ_{μ_i} and m_{μ_i} as fixed. This simplification is the approach of ARD in Variational Bayesian to make the optimization tractable. It relies on the iterative nature of the algorithm to gradually converge to a consistent set of parameter estimates. Thus, differentiating with respect to α_i and setting to 0 we get:

$$\frac{\partial}{\partial_{\alpha_i}} \left\langle \log \mathcal{N}(\, 0,\, \Lambda_i^{-1}\,) \right\rangle_{q_i(\,\mu_i\,)} \;=\; 0$$

$$\Leftrightarrow\; \frac{D}{2\alpha_i} \;-\; \frac{1}{2}\left(\,\mathrm{Tr}(\Sigma_{\mu_i}) \;+\; m_{\mu_i}^T m_{\mu_i}\,\right) = 0$$

$$\Leftrightarrow\; \frac{D}{2\alpha_i} \;-\; \frac{1}{2}\left(\,\mathrm{Tr}(\Sigma_{\mu_i}) \;+\; m_{\mu_i}^T m_{\mu_i}\,\right) = 0$$

$$\Rightarrow\; \alpha_i \;=\; \frac{D}{\mathrm{Tr}(\Sigma_{\mu_i}) \;+\; m_{\mu_i}^T\, m_{\mu_i}}$$

which can be solved for α_i iteratively.

Lets derive the VB free energy to monitor convergence in our implementation. The free energy is given by :

$$\mathcal{F}(\, q(\,\mu\,)\,,\, q(\mathbf{s}\,)\,) \;=\; \left\langle \log\, P(\mathbf{x}\,,\, \mathbf{s}\,,\, \mu \,|\, \pi\,,\, \sigma^2\,,\, \alpha\,) \right\rangle_{q(\,\mu\,)\, q(\,\mathbf{s}\,)} + \mathbf{H}[q(\,\mathbf{s}\,)] + \mathbf{H}[q(\,\mu\,)] \qquad (2)$$

Note that

$$\left\langle \log\, P(\mathbf{x}\,,\, \mathbf{s}\,,\, \mu \,|\, \pi\,,\, \sigma^2\,,\, \alpha\,) \right\rangle_{q(\,\mu\,)\, q(\,\mathbf{s}\,)} = \left\langle \log\, P(\mathbf{x} \,|\, \mathbf{s}\,,\, \mu\;\pi\,,\, \sigma^2\;) \right\rangle_{q(\,\mu\,)\, q(\,\mathbf{s}\,)} \;\; + \langle \log\, P(\,\mathbf{s} \,|\, \pi\,) \rangle_{q(\,\mathbf{s}\,)}$$
$$+ \langle \log\, P(\,\mu \,|\, \alpha\,) \rangle_{q(\,\mu\,)}$$

First two terms are the same as in Mean-Field, with the only difference that the log-likelihood term takes the expectation of mu under approximate posterior as an input. The third term is given by

$$\langle \log\, P(\,\mu \,|\, \alpha\,) \rangle_{q(\,\mu\,)} \;=\; \sum_{i=1}^{K} \left\langle -\frac{D}{2} \log 2\pi \;-\; \frac{D}{2} \log \alpha_i \;-\; \frac{\alpha_i}{2}\, \mu_i^T\, \mu_i \right\rangle_{q_i(\,\mu_i\,)}$$

$$= \sum_{i=1}^{K} -\frac{D}{2} \log 2\pi \;-\; \frac{D}{2} \log \alpha_i \;-\; \frac{\alpha_i}{2}\, \langle\, \mu_i^T\, \mu_i \,\rangle_{q_i(\,\mu_i\,)}$$

$$= \; - \frac{DK}{2} \log 2\pi \; - \; \sum_{i=1}^{K} \left[\frac{D}{2} \log \alpha_i \; + \; \frac{\alpha_i}{2} \left(\; \mathrm{Tr}(\Sigma_{\mu_i}) + m_{\mu_i}^T m_{\mu_i} \; \right) \right]$$

Coming back to free energy expression (2). The entropy $\mathbf{H}[q(\,\mathbf{s}\,)]$ is already calculated in Mean-Field. The only term left is $\mathbf{H}[q(\,\mu\,)]$ which is the entropy of independent multivariate gaussians, with known formula.

$$\mathbf{H}[q(\,\mu\,)] \; = \; \sum_{i=1}^{K} \left[\frac{D}{2} (\, 1 \; + \; \log 2\pi \,) \; + \; \frac{1}{2} \log |\, \Sigma_{\mu_i} \,| \right]$$

$$= \; \frac{DK}{2} \; + \; \frac{DK}{2} \log 2\pi \; + \; \frac{1}{2} \sum_{i=1}^{K} \log |\, \Sigma_{\mu_i} \,|$$

Knowing that $\Sigma_{\mu_i} \; = \; \mathrm{diag}(\, \frac{\sigma^2}{\sigma^2 \, \alpha_i \; + \; \sum_{n=1}^{N} \, \lambda_{in}} \,)$

$$\Rightarrow \; \mathbf{H}[q(\,\mu\,)] \; = \; \frac{DK}{2} \; + \; \frac{DK}{2} \log 2\pi \; + \; \frac{D}{2} \sum_{i=1}^{K} \log \; \frac{\sigma^2}{\sigma^2 \, \alpha_i \; + \; \sum_{n=1}^{N} \lambda_{in}}$$

Coming back to E-step (1) :

$$\lambda_{in} \; = \sigma \left(\log \frac{\pi_i}{1 \, - \, \pi_i} \; + \; \frac{1}{\sigma^2} \, \langle \, \mu_i \, \rangle_{q(\,\mu_i\,)}^T \, (\mathbf{x}_n \; - \; \sum_{j \neq i}^{K} \lambda_{jn} \, \langle \, \mu_j \, \rangle_{q(\,\mu_j\,)} \,) \; - \; \frac{1}{2\sigma^2} \, \langle \, \mu_i^T \, \mu_i \, \rangle_{q(\,\mu_i\,)} \right)$$

$$\Rightarrow \; \lambda_{in} \; = \sigma \left(\log \frac{\pi_i}{1 \, - \, \pi_i} \; + \; \frac{1}{\sigma^2} \, m_{\mu_i}^T \, (\mathbf{x}_n \; - \; \sum_{j \neq i}^{K} \lambda_{jn} \; m_{\mu_j} \;) \; - \; \frac{1}{2\sigma^2} \, (\; \mathrm{Tr}(\Sigma_{\mu_i}) + m_{\mu_i}^T m_{\mu_i} \;) \right)$$

4.2

Found features, ordered by their alpha value:

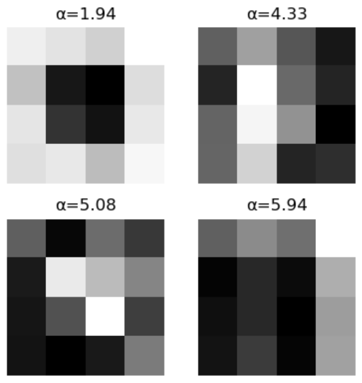


Figure 40: K=4

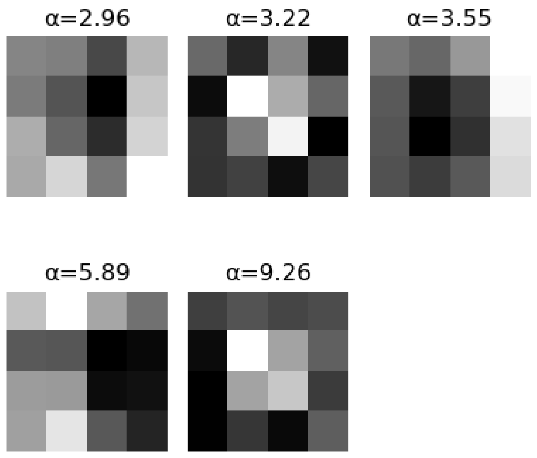


Figure 41: K=5

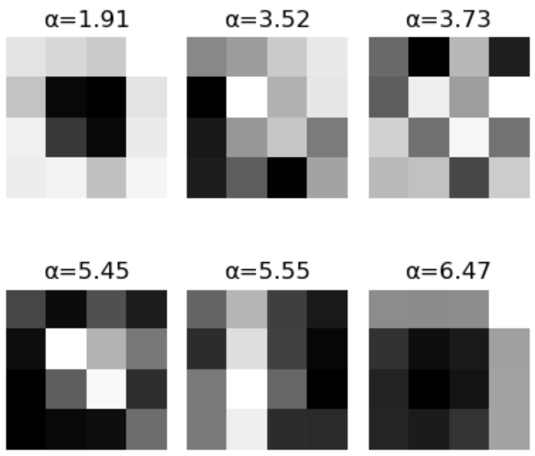


Figure 42: K=6

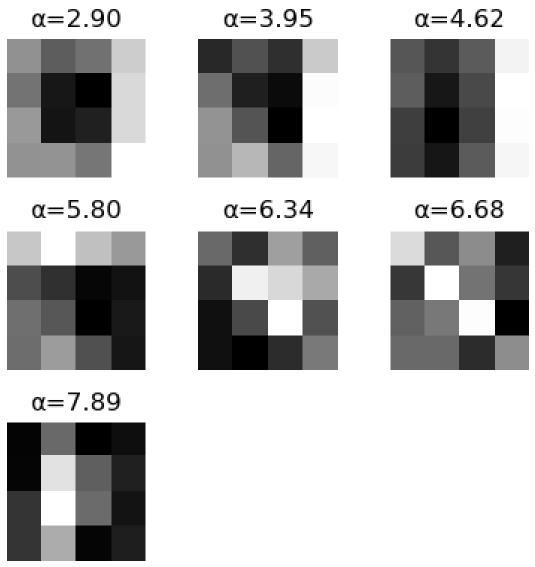


Figure 43: K=7

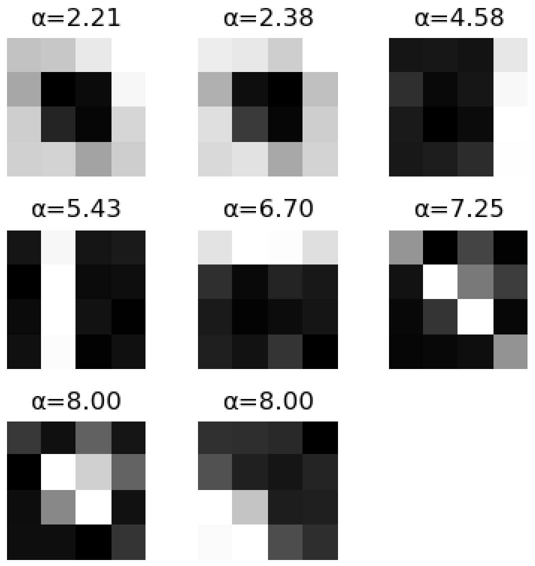


Figure 44: K=8

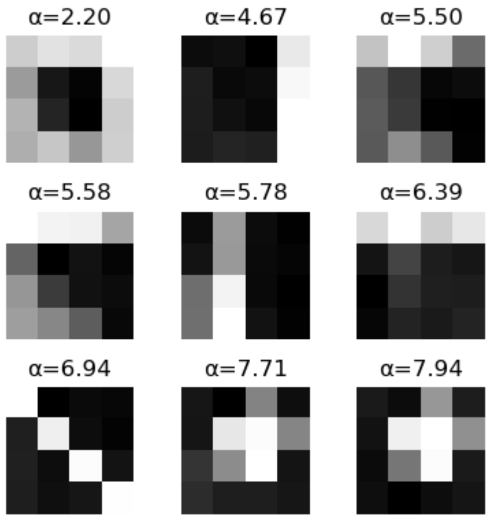


Figure 45: K=9

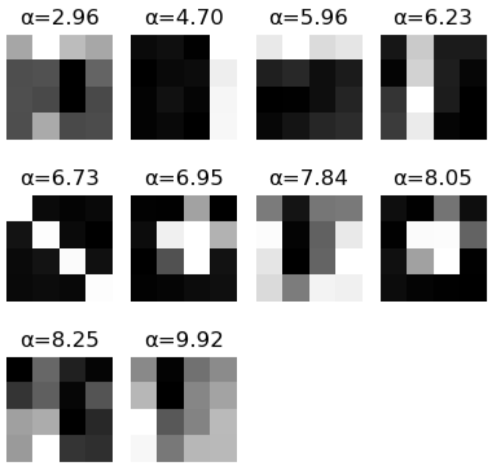


Figure 46: K=10

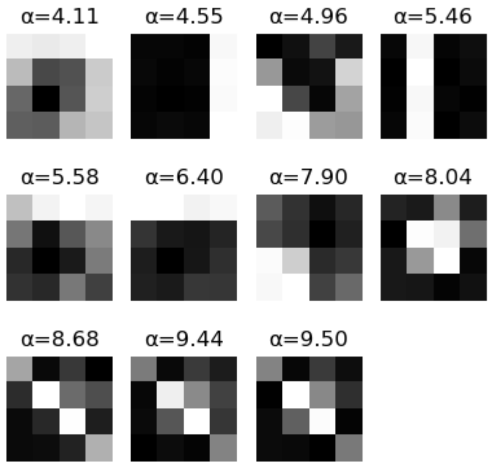


Figure 47: K=11

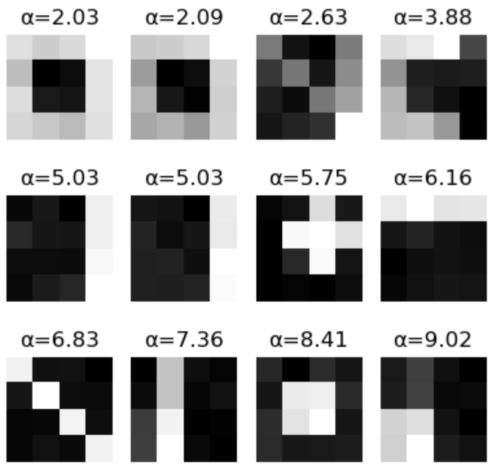


Figure 48: K=12

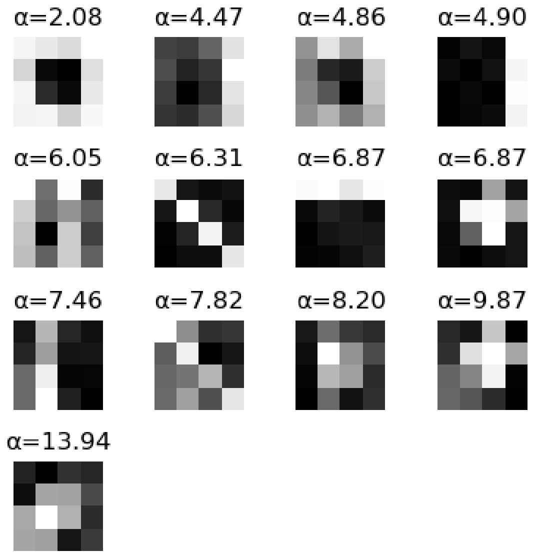


Figure 49: K=13

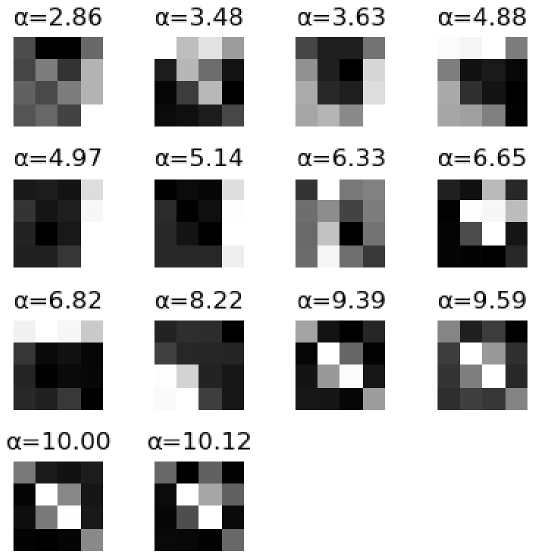


Figure 50: K=14

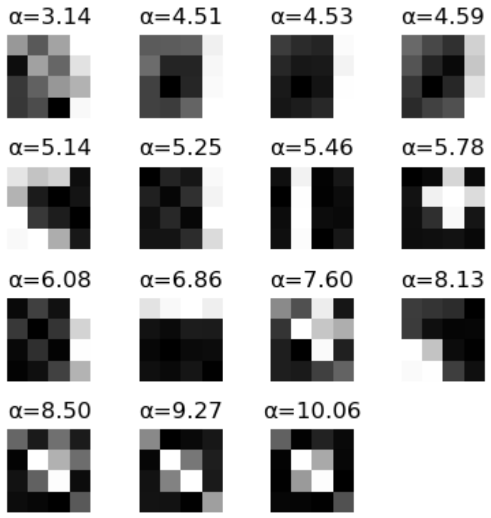


Figure 51: K=15

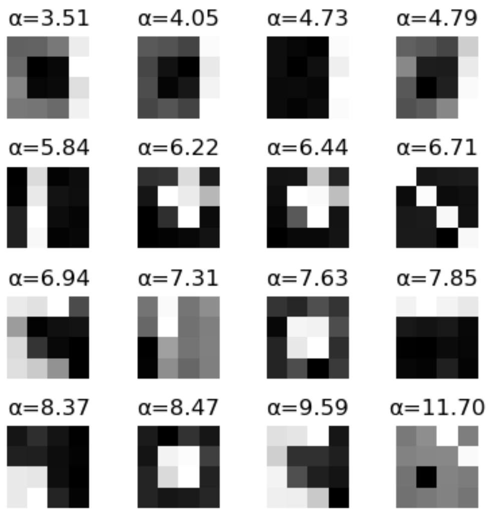


Figure 52: K=16

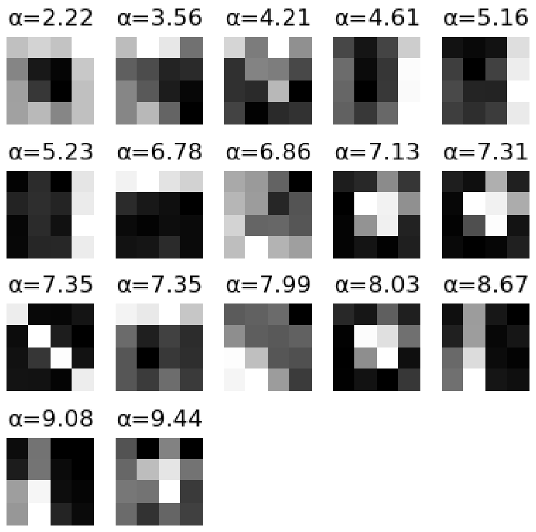


Figure 53: K=17

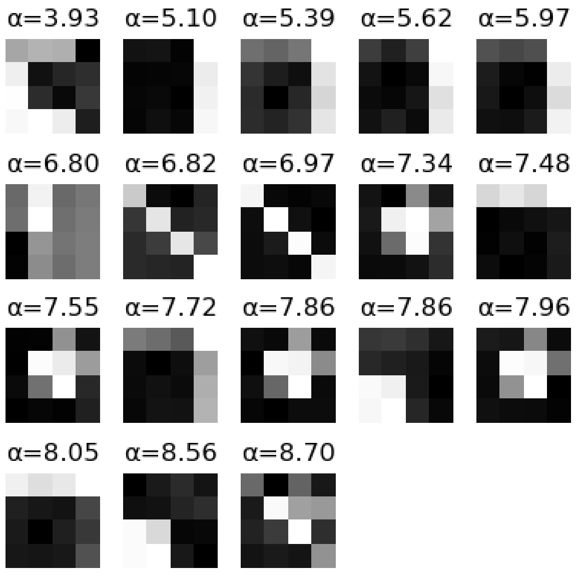


Figure 54: K=18

Variational Bayes Free energies for all K's :

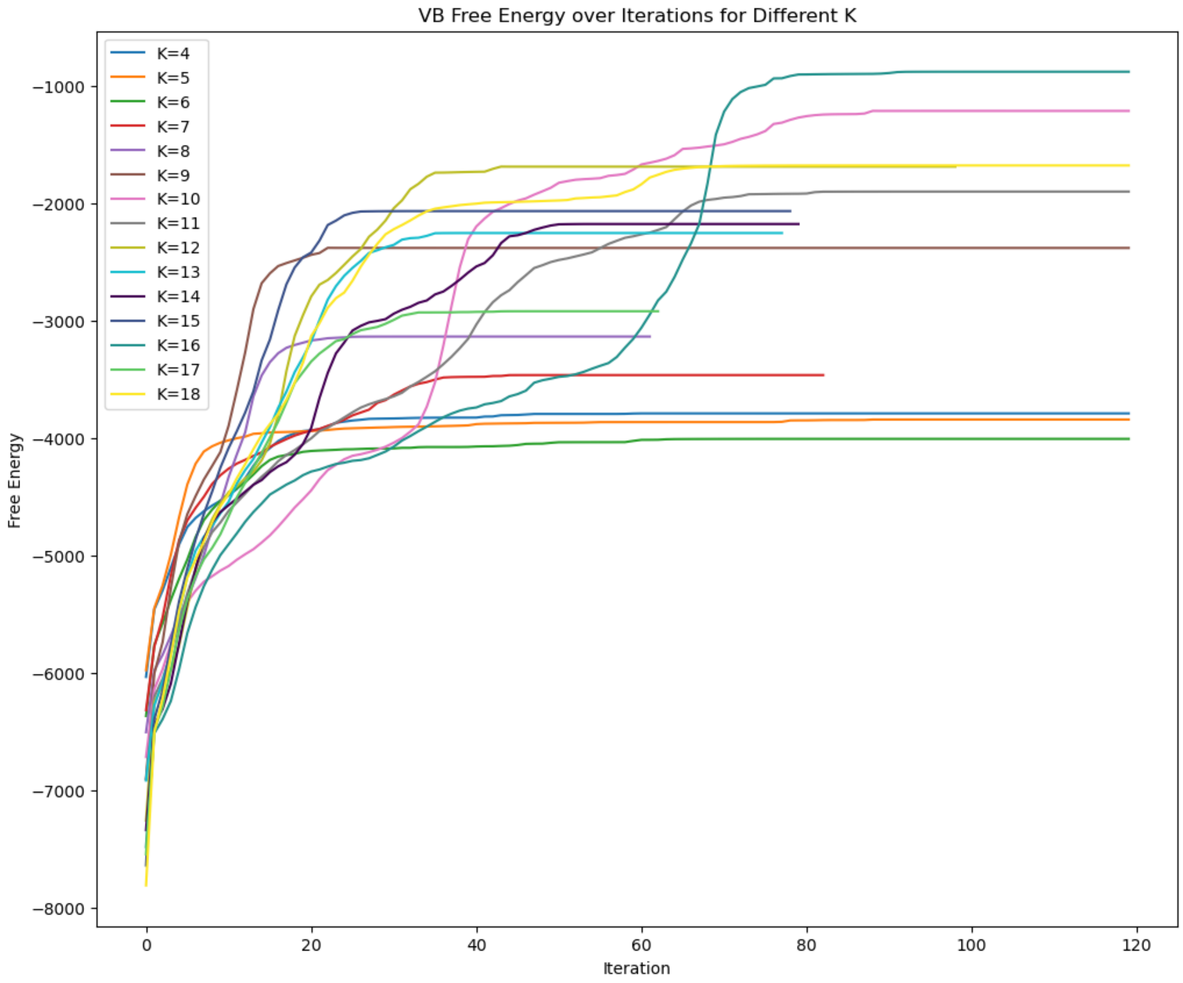


Figure 55: VB free energy for K=4 to K=18

Measure of the effective number of factors at each iteration for each K :

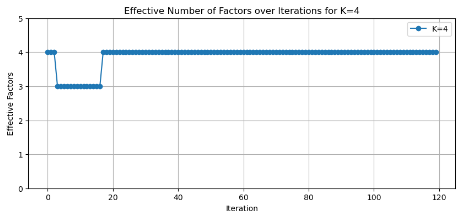


Figure 56: K=4

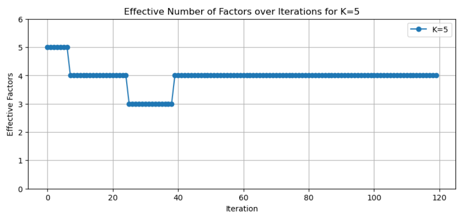


Figure 57: K=5

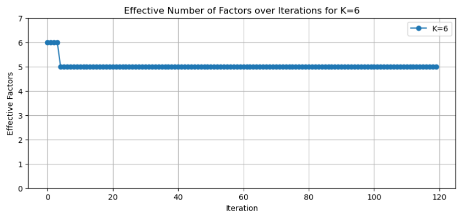


Figure 58: K=6

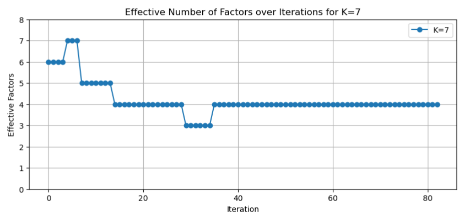


Figure 59: K=7

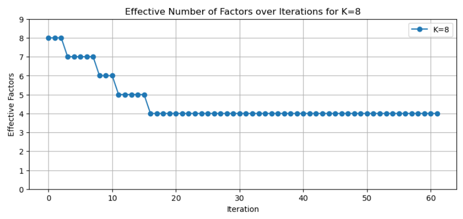


Figure 60: K=8

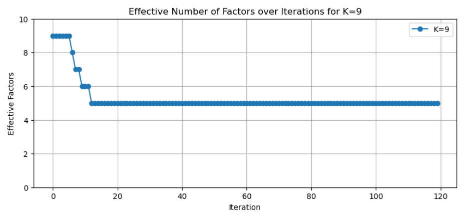


Figure 61: K=9

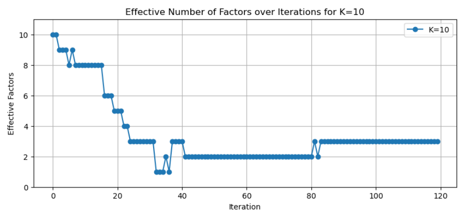


Figure 62: K=10

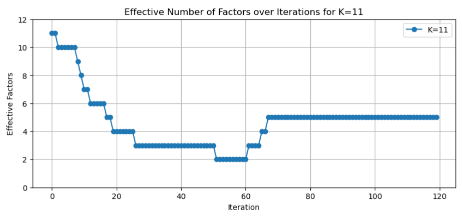


Figure 63: K=11

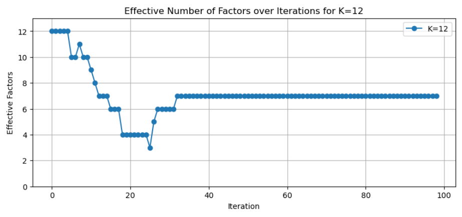


Figure 64: K=12

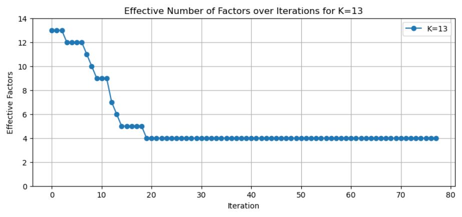


Figure 65: K=13

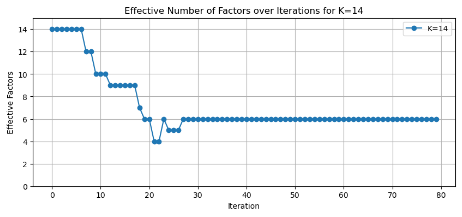


Figure 66: K=14

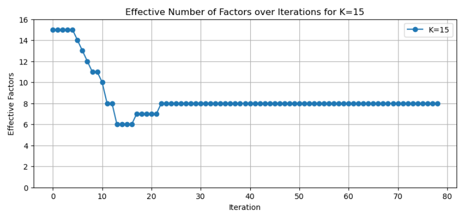


Figure 67: K=15

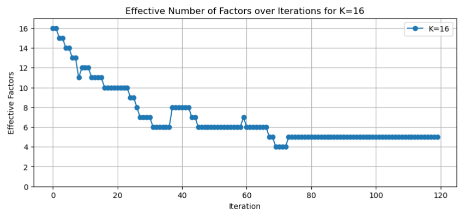


Figure 68: K=16

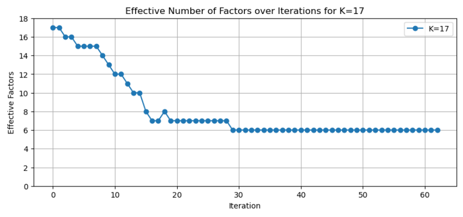


Figure 69: K=17

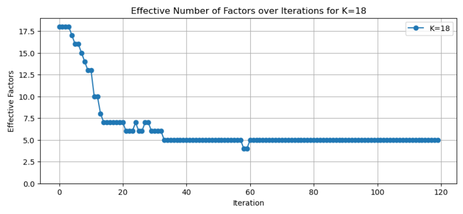


Figure 70: K=18

Results analysis :

Firstly, there is a notable correlation between the convergence of free energy and the number of effective factors. As the algorithm iterates, both these metrics tend to converge proportionally. This simultaneous convergence suggests a strong link between the model's fit to the data and the optimal number of factors needed to capture the data's underlying structure.

Regarding the free energy values, a general increase is observed with higher K values. This trend aligns with expectations, as increasing the number of latent dimensions typically grants the model more degrees of freedom, potentially leading to overfitting. However, this pattern is not entirely consistent across different runs of the algorithm, likely due to the randomness in initializations for each K value. On rerunning the algorithm, it's usually seen that higher K values correlate with increased free energy, reinforcing the notion that more complex models (higher number of latent dimensions) are prone to overfitting.

When examining the learned features, the corresponding values of the factors (alphas) do not always accurately reflect the true importance of these features. This inconsistency is evident upon visual inspection, where some actual features might receive higher factor values compared to noisy ones. Despite this, the algorithm generally performs well in identifying the underlying features of the dataset.

Comparatively, when VB is stacked against the Mean-field approach, the results are slightly inferior, possibly due to the inconsistencies in the alpha values. A potential improvement to enhance the model's robustness and reduce randomness could be the introduction of a prior on the alpha values, such as an exponential distribution. Ideally, this prior would be scaled with K, becoming less concentrated with an increase in K. This adjustment could address the current shortcomings of initializing alphas randomly, which appears to be a suboptimal strategy.

In summary, while the algorithm effectively identifies key data features and shows a consistent pattern in the convergence of free energy and the number of effective factors, there's room for improvement. Adjusting the approach to initializing and handling alpha values could lead to more reliable and robust model performance, especially in terms of accurately reflecting the importance of learned features.

VB ARD code :

```
import numpy as np
from scipy.special import expit as stable_sigmoid
from scipy.special import logit as logit
import matplotlib.pyplot as plt
from random import shuffle
import math

# VB E-step

def update_lambda(X, pie, sigma, mean_mu, Lambda,
    tr_sigma_plus_m_mu_squared_list):
    N, D = X.shape
    _, K = mean_mu.shape

    pie = pie.ravel()

    for n in range(N):
        lambda_n = Lambda[n, :]

        # Shuffle indices for updating in random order
        indices = list(range(K))
        shuffle(indices)

        for i in indices:
            sum_except_i = sum(lambda_n[j] * mean_mu[:, j] for j in
     range(K) if j != i)
            lambda_n[i] = stable_sigmoid(
                logit(pie[i]) +
                (1 / sigma**2) * mean_mu[:, i].T @ (X[n, :] -
    sum_except_i) -
                (1 / (2 * sigma**2)) *
    tr_sigma_plus_m_mu_squared_list[i]
            )

        Lambda[n, :] = lambda_n

    return Lambda


# VB M-step

def update_q_mu(X, sigma, alpha, Lambda, mean_mu):
    N, D = X.shape
    K = len(alpha)

    diag_cov_mu = np.zeros((K, D))

    # Shuffle indices for updating in random order
    indices = list(range(K))
    shuffle(indices)

    for i in indices:
        sum_m_mu_i = np.zeros(D)

        # Compute the scalar for the diagonal elements of
```

```python
Sigma_mu_i
        sigma_mu_i_scalar = sigma**2 / ((sigma**2 * alpha[i]) + np.sum(Lambda[:, i]))
        diag_cov_mu[i] = np.full(D, sigma_mu_i_scalar)

        # Loop through all data points
        for n in range(N):
            # Calculate the sum for j != i
            sum_except_i = sum(Lambda[n, j] * mean_mu[:, j] for j in range(K) if j != i)

            # Update the sum for m_mu_i, which is a sum of vectors
            sum_m_mu_i += Lambda[n, i] * (X[n, :] - sum_except_i)

        m_mu_i = (sigma_mu_i_scalar / sigma**2) * sum_m_mu_i
        mean_mu[:, i] = m_mu_i

    return diag_cov_mu, mean_mu


def update_sigma_pie(X, Lambda, mean_mu):

    # Same update as in Mean-field but with mean_mu instead of mu

    N, D = X.shape
    K = Lambda.shape[1]

    # ESS and ES calculations as in Mean Field
    ES = Lambda
    ESS = np.einsum('ni,nj->ij', Lambda, Lambda)
    diagonal_correction = np.sum(Lambda - Lambda**2, axis=0)
    np.fill_diagonal(ESS, np.diag(ESS) + diagonal_correction)

    # Update sigma
    sigma = np.sqrt((np.trace(np.dot(X.T, X)) + np.trace(np.dot(np.dot(mean_mu.T, mean_mu), ESS))
                     - 2 * np.trace(np.dot(np.dot(ES.T, X), mean_mu))) / (N * D))

    # Update pie
    pie = np.mean(Lambda, axis=0)

    return sigma, pie


# Free energy computation

def compute_free_energy(X, mean_mu, sigma, pie, Lambda, diag_cov_mu, alpha, tr_sigma_plus_m_mu_squared_list):
    D, K = mean_mu.shape
    N = X.shape[0]
    pie = pie.ravel()
    cst = 1e-10
    free_energy = 0


    # Precompute mean_mu^T mean_mu for efficiency
    mean_mu_T_mean_mu = mean_mu.T @ mean_mu

    # < log P(x,s | mu, pie, sigma, alpha) >_q(mu)q(s), same as in Mean Field,
    for n in range(N):
        x_n = X[n, :]
        lambda_n = Lambda[n, :]

        term1 = np.sum(lambda_n * (np.log(pie + cst) - np.log(lambda_n + cst)) +
                       (1 - lambda_n) * (np.log(1 - pie + cst) - np.log(1 - lambda_n + cst)))

        term2 = -(D / 2) * np.log(2 * np.pi * sigma**2)

        mu_lambda = mean_mu @ lambda_n
        term3_a = np.dot((x_n - mu_lambda), (x_n - mu_lambda))
        term3_b = np.sum((lambda_n - lambda_n**2) * np.sum(mean_mu**2, axis=0))
        term3 = -(1 / (2 * sigma**2)) * (term3_a + term3_b)

        free_energy += term1 + term2 + term3

    # <log P(mu | alpha)>_q(mu)
    for i in range(K):
        free_energy += -0.5 * D * np.log(2 * np.pi) - 0.5 * D * np.log(alpha[i]) - 0.5 * alpha[i] * tr_sigma_plus_m_mu_squared_list[i]

    # H[q(mu)]
    for i in range(K):
        diag_sigma_mu_i = diag_cov_mu[i]
        free_energy += 0.5 * D * (1 + np.log(2 * np.pi)) + 0.5 * np.sum(np.log(diag_sigma_mu_i))

    return free_energy


# Hyper M-step

def update_alpha(D, diag_cov_mu, mean_mu):
    K = diag_cov_mu.shape[0]
    alpha = np.zeros(K)
    tr_sigma_plus_m_mu_squared_list = []

    mean_mu_T_mean_mu = mean_mu.T @ mean_mu

    for i in range(K):
        diag_sigma_mu_i = diag_cov_mu[i]
        tr_sigma_mu_i = np.sum(diag_sigma_mu_i)
        m_mu_i_T_m_mu_i = mean_mu_T_mean_mu[i, i]
        tr_sigma_plus_m_mu_squared = tr_sigma_mu_i + m_mu_i_T_m_mu_i
        tr_sigma_plus_m_mu_squared_list.append(tr_sigma_plus_m_mu_squared)

        alpha[i] = D / tr_sigma_plus_m_mu_squared


    return alpha, tr_sigma_plus_m_mu_squared_list


def VB_ARD(X, K, max_iter, epsilon=1e-7):
    N, D = X.shape

    # Initialize parameters
    Lambda = np.random.rand(N, K)
    mean_mu = np.random.rand(D, K)
    sigma = np.random.rand() * 0.1
    pie = np.random.rand(1, K)
    alpha = np.random.rand(K) * 0.1

    # Initialise tr_sigma_plus_m_mu_squared_list that is consistent with alpha initialization
    tr_sigma_plus_m_mu_squared_list = [(D * alpha[i]) + np.dot(mean_mu[:, i], mean_mu[:, i]) for i in range(K)]

    effective_factors_counts = []
    free_energies = []
    F_old = -np.inf

    for iter in range(max_iter):
        # E-step
        Lambda = update_lambda(X, pie, sigma, mean_mu, Lambda, tr_sigma_plus_m_mu_squared_list)

        # M-step
        diag_cov_mu, mean_mu = update_q_mu(X, sigma, alpha, Lambda, mean_mu)
        sigma, pie = update_sigma_pie(X, Lambda, mean_mu)

        # Hyper M-step
        alpha, tr_sigma_plus_m_mu_squared_list = update_alpha(D, diag_cov_mu, mean_mu)

        # Count effective factors
        effective_factors_count = np.sum(alpha < 6) # set to 6 through empirical obs
        effective_factors_counts.append(effective_factors_count)

        # Compute free energy
        F = compute_free_energy(X, mean_mu, sigma, pie, Lambda, diag_cov_mu, alpha, tr_sigma_plus_m_mu_squared_list)
        free_energies.append(F)

        # Check for convergence
        if iter > 0 and np.abs(F - F_old) < epsilon:
            print(f"Convergence reached at iteration {iter}")
            break
        F_old = F

    return free_energies, mean_mu, alpha, effective_factors_counts


```

```

K_values = range(4, 19)
max_iter = 120
free_energies_all_K = []
final_mean_mu_all_K = []
final_alphas_all_K = []
effective_factors_all_K = []

for K in K_values:
    free_energies, mean_mu, alpha, effective_factors_counts =
    VB_ARD(Y, K, max_iter)

    free_energies_all_K.append(free_energies)
    final_mean_mu_all_K.append(mean_mu)
    final_alphas_all_K.append(alpha)
    effective_factors_all_K.append(effective_factors_counts)
```

5 EP for the binary factor model

5.1

$$\log P(\mathbf{x}, \mathbf{s}) = \log P(\mathbf{x} | \mathbf{s}) + \log P(\mathbf{s})$$

$$= -\frac{D}{2} \log 2\pi\sigma^2 - \frac{1}{2\sigma^2} (\mathbf{x} - \sum_{i=1}^{K} s_i \mu_i)^T (\mathbf{x} - \sum_{i=1}^{K} s_i \mu_i) + \sum_{i=1}^{K} s_i \log \pi_i + (1 - s_i) \log 1 - \pi_i$$

$$= -\frac{D}{2} \log 2\pi\sigma^2 - \frac{\mathbf{x}^T\mathbf{x}}{2\sigma^2} + \frac{1}{\sigma^2} \sum_{i=1}^{K} \mu_i^T \mathbf{x} s_i - \frac{1}{2\sigma^2} \sum_{i=1}^{K}\sum_{j=1}^{K} s_i s_j \mu_i^T \mu_j + \sum_{i=1}^{K} s_i \log \frac{\pi_i}{1 - \pi_i} + \sum_{i=1}^{K} \log 1 - \pi_i$$

Let $C = -\frac{D}{2} \log 2\pi\sigma^2 - \frac{\mathbf{x}^T\mathbf{x}}{2\sigma^2} + \sum_{i=1}^{K} \log 1 - \pi_i$ then,

$$\Rightarrow \log P(\mathbf{x}, \mathbf{s}) = C + \frac{1}{\sigma^2} \sum_{i=1}^{K} \mu_i^T \mathbf{x} s_i - \frac{1}{2\sigma^2} \sum_{i=1}^{K}\sum_{j=1}^{K} s_i s_j \mu_i^T \mu_j + \sum_{i=1}^{K} s_i \log \frac{\pi_i}{1 - \pi_i}$$

$$= C + \sum_{i=1}^{K} (\frac{\mu_i^T \mathbf{x}}{\sigma^2} + \log \frac{\pi_i}{1 - \pi_i}) s_i - \sum_{i=1}^{K}\sum_{j=1}^{K} (\frac{\mu_i^T \mu_j}{2\sigma^2}) s_i s_j$$

As we know (from previous questions), $\sum_{i=1}^{K}\sum_{j=1}^{K} \mu_i^T \mu_j \, s_i \, s_j = \sum_{i\neq j} \mu_i^T \mu_j \, s_i \, s_j + \sum_{i=1}^{K} \mu_i^T \mu_j \, s_i$ where $\sum_{i\neq j} = \sum_{i=1}^{K}\sum_{j\neq i}^{K}$

$$\Rightarrow \log P(\mathbf{x}, \mathbf{s}) = C + \sum_{i=1}^{K} \left[\frac{\mu_i^T}{\sigma^2}(\mathbf{x} - \frac{\mu_i}{2}) + \log \frac{\pi_i}{1 - \pi_i} \right] s_i - \sum_{i\neq j} (\frac{\mu_i^T \mu_j}{2\sigma^2}) \, s_i \, s_j$$

Now, $\forall \, i, j \;\; \mu_i^T \mu_j = \mu_j^T \mu_i$

$$\Rightarrow \sum_{i\neq j} (\frac{\mu_i^T \mu_j}{2\sigma^2}) \, s_i \, s_j = \sum_{i=1}^{K} \sum_{j=i+1}^{K} (\frac{2\mu_i^T \mu_j}{2\sigma^2}) \, s_i \, s_j = \sum_{i<j} (\frac{\mu_i^T \mu_j}{\sigma^2}) \, s_i \, s_j$$

$$\Rightarrow \log P(\mathbf{x}, \mathbf{s}) = C + \sum_{i<j} (\frac{-\mu_i^T \mu_j}{\sigma^2}) \, s_i \, s_j + \sum_{i=1}^{K} \left(\frac{\mu_i^T}{\sigma^2}(\mathbf{x} - \frac{\mu_i}{2}) + \log \frac{\pi_i}{1 - \pi_i} \right) s_i$$

Let $w_{ij} = \frac{-\mu_i^T \mu_j}{\sigma^2}$ and $b_i = \frac{\mu_i^T}{\sigma^2}(\mathbf{x} - \frac{\mu_i}{2}) + \log \frac{\pi_i}{1-\pi_i}$

$$\Rightarrow \log P(\mathbf{x}, \mathbf{s}) = C + \sum_{i<j} w_{ij} \, s_i \, s_j + \sum_{i} b_i$$

$$\Rightarrow \log f_i(s_i) = b_i \, s_i, \;\; \log g_{ij}(s_i, s_j) = w_{ij} \, s_i \, s_j$$

Exponentiating this we get the same form as a Boltzmann machine

$$P(\mathbf{x}, \mathbf{s}) = \frac{1}{Z} \exp \left(\sum_{i<j} w_{ij} \, s_i \, s_j + \sum_{i} b_i \, s_i \right)$$

In this formula:

- w_{ij} represents the weight of the interaction between binary latent variables s_i and s_j. For simplicity and symmetry (symmetry of dot product), we set $w_{ij} = w_{ji}$ for all i, j, and $w_{ij} = 0$ when $i = j$, changing the summation index to $i < j$.
- b_i is the single-variable contribution or the bias term for each latent variable s_i.

- Z, the normalizing constant, is given by $\exp(C)$, where C is a constant term derived from the distribution.
- The binary nature of the latent variables $\mathbf{s}$ and the undirected pairwise interactions result in a model with a Markov Random Field (MRF) structure.
- The exponential form of the distribution is indicative of a jointly exponential family.
- The normalizer Z, which has 2^K states, highlights the computational intractability of the model. This intractability arises from the exponential number of states the latent variables can take.

5.2

First, we can see that the joint can be rewritten as

$$P(\mathbf{x}, \mathbf{s}) \;=\; \frac{1}{Z} \prod_i \exp b_i\, s_i \prod_{ij} \exp w_{ij}\, s_i\, s_j \;=\; \frac{1}{Z} \prod_i f_i(s_i) \prod_{ij} g_{ij}(s_i, s_j)$$

In the expression above, we introduced the notation ij to replace the condition i<j for convenience. The pairwise factor product index ij represents a product over all possible pairs of distinct random variables ($i \neq j$) where the order of indices does not matter. In other words, pairs ij and ji are considered the same pair. This equivalence is captured by the relationship $g_{ij}(s_i, s_j) = g_{ji}(s_j, s_i)$. Moreover, we are only counting each pair once giving the equivalent with i<j notation. This choice of notation simplifies the expression for clarity and convenience. (3)

Thus, approximate posterior distribution is of the form :

$$q(\mathbf{s}) \;=\; \prod_i \tilde{f}_i(s_i) \prod_{ij} \tilde{g}_{ij}(s_i, s_j)$$

Approximation of $\tilde{f}_i$:

As we know each EP update involves a KL minimization, i.e

$$\tilde{f}_i^{new}(s_i) \;=\; \underset{\tilde{f}_i}{\operatorname{argmin}} \; \mathbf{KL}[\; f_i(s_i)\, q_{\neg \tilde{f}_i}(s_i) \mid \tilde{f}_i(s_i)\, q_{\neg \tilde{f}_i}(s_i) \;]$$

Now, we know that $f_i(s_i) \propto \exp b_i\, s_i$ where s_i is binary random variables taking values in $\{0, 1\}$, i.e singleton factor follows a bernouilli distribution where b_i is the natural parameter. This means that we do not need to make an EP approximation for $f_i(s_i)$ as we can just choose $\tilde{f}_i(s_i)$ to be a Bernoulli distribution of the form

$$\tilde{f}_i(s_i) \;=\; p_i^{s_i}\,(1 - p_i)^{1-s_i} \;=\; (1 \,-\, p_i)\,\exp\left(\log \frac{p_i}{1-p_i}\, s_i\right)$$

Let $\theta_i \;=\; \log \frac{p_i}{1-p_i} \;\Rightarrow\; \tilde{f}_i(s_i) \;=\; \exp\left(\,\theta_i\, s_i \;-\; \log\left(\,1 + \exp\,\theta_i\,\right)\right) \;\propto\; \exp\,\theta_i\, s_i$

Thus, as true and approximate singleton factors are in the same exponential family, minimizing the **KL** divergence above amounts to just setting the approximate natural parameter θ_i equal to the true natural parameter b_i. Moreover, we do not need to approximate the singleton factors, we just need to set

$$\theta_i \;=\; b_i$$

Therefore, singleton sites are equal to $f_i(s_i) \;=\; \exp\,[\,b_i\, s_i \,-\, \log\,(\,1 + \exp\, b_i\,)]\;\; \forall\, i$, (normalized exponential form of Bernouilli distribution with natural parameter b_i). (4)

Approximation of $\tilde{g}_{ij}$:

We chose to approximate the pairwise factors by a product of two Bernoulli so that both the cavity and the approximating pairwise site are in the same exponential family (Bernouilli) letting us use moment matching to minimize the KL divergence. Thus,

$$\tilde{g}_{ij}\,(s_i, s_j) \;=\; \tilde{g}_{ij}\,(s_i) \;\times\; \tilde{g}_{ji}\,(s_j)$$

$$=\; \exp\,[\,\theta_{ij}\, s_i \,-\, \log\,(\,1 + \exp\,\theta_{ij}\,)] \;\times\; \exp\,[\,\theta_{ji}\, s_j \,-\, \log\,(\,1 + \exp\,\theta_{ji}\,)]$$

Where θ_{ij} and θ_{ji} are the natural parameters of factored approximating sites $\tilde{g}_{ij}\,(s_i)$, $\tilde{g}_{ji}\,(s_j)$ respectively. Note that we used $\tilde{g}_{ji}\,(s_j)$ instead of $\tilde{g}_{ij}\,(s_j)$ for notation consistency with the order of indices of the natural parameters as θ_{ij} is not necessarily equal to θ_{ji} so we need to distinguish the two (but as I said in (3) this is just a notation convenience as we defined $g_{ij}(s_i, s_j) \;=\; g_{ji}(s_j, s_i)$ earlier.

Lets derive the expression for the cavity distribution $q_{\neg \tilde{g}_{ij}}\,(s_i, s_j)$:

$$q_{\neg\tilde{g}_{ij}}(s_i, s_j) \;=\; \sum_{\neg(s_i,s_j)} \frac{q(\mathbf{s})}{\tilde{g}_{ij}(s_i, s_j)} \;=\; \sum_{\neg(s_i,s_j)} \prod_{t=1}^{K} \tilde{f}_t(s_t) \prod_{\substack{kl \\ kl\neq(ij)}} \tilde{g}_{kl}(s_k, s_l)$$

Where the last product is just the product of all approximating sites $\tilde{g}_{kl}(s_k, s_l)$ where $(kl) \neq (ij)$, i.e the product over all possible pairs except pair ij. Additionally, $\sum_{\neg(s_i,s_j)}$ is the marginalisation over all binary latent variables except s_i and s_j.

$$\Rightarrow\; q_{\neg\tilde{g}_{ij}}(s_i, s_j) \;=\; \sum_{\neg(s_i,s_j)} \left(\prod_{t=1}^{K} \tilde{f}_t(s_t) \prod_{\substack{kl \\ kl\neq(ij)}} \tilde{g}_{kl}(s_k, s_l) \right)$$

Now,

$$\prod_{t=1}^{K} \tilde{f}_t(s_t) \;=\; \tilde{f}_i(s_i)\,\tilde{f}_j(s_j) \prod_{t\neq\{i,j\}} \tilde{f}_t(s_t)$$

and

$$\prod_{\substack{kl \\ kl\neq(ij)}} \tilde{g}_{kl}(s_k, s_l) \;=\; \prod_{m\neq\{i,j\}} \tilde{g}_{im}(s_i, s_m) \prod_{m\neq\{i,j\}} \tilde{g}_{jm}(s_j, s_m) \prod_{\substack{kl \\ k\neq\{i,j\} \\ l\neq\{i,j\}}} \tilde{g}_{kl}(s_k, s_l) \qquad (5)$$

where $m \neq \{i,j\}$ for $\tilde{g}_{im}(s_i, s_m)$ is equivalent all of the possible pairs with i excluding pair ij, and analogously for $\tilde{g}_{jm}(s_j, s_m)$ with $m \neq \{i,j\}$. Now as we defined earlier,

$$\tilde{g}_{ij}(s_i, s_j) \;=\; \tilde{g}_{ij}(s_i) \;\times\; \tilde{g}_{ji}(s_j),\; \forall\, ij$$

Therefore,

$$(5) \;=\; \prod_{m\neq\{i,j\}} \tilde{g}_{im}(s_i)\,\tilde{g}_{mi}(s_m) \prod_{m\neq\{i,j\}} \tilde{g}_{jm}(s_j)\,\tilde{g}_{mj}(s_m) \prod_{\substack{kl \\ k\neq\{i,j\} \\ l\neq\{i,j\}}} \tilde{g}_{kl}(s_k)\,\tilde{g}_{lk}(s_l)$$

$$= \prod_{m \neq \{i,j\}} \tilde{g}_{im}\left(s_i\right) \prod_{m \neq \{i,j\}} \tilde{g}_{jm}\left(s_j\right) \prod_{m \neq \{i,j\}} \tilde{g}_{mi}\left(s_m\right) \prod_{m \neq \{i,j\}} \tilde{g}_{mj}\left(s_m\right) \prod_{\substack{kl \\ k \neq \{i,j\} \\ l \neq \{i,j\}}} \tilde{g}_{kl}\left(s_k\right) \tilde{g}_{lk}\left(s_l\right)$$

$$\Rightarrow \; q_{\neg \tilde{g}_{ij}}\left(s_i, s_j\right) = \sum_{\neg(s_i,s_j)} \left(\tilde{f}_i(s_i)\, \tilde{f}_j(s_j) \prod_{t \neq \{i,j\}} \tilde{f}_t(s_t) \prod_{m \neq \{i,j\}} \tilde{g}_{im}\left(s_i\right) \prod_{m \neq \{i,j\}} \tilde{g}_{jm}\left(s_j\right) \right.$$
$$\left. \prod_{m \neq \{i,j\}} \tilde{g}_{mi}\left(s_m\right) \prod_{m \neq \{i,j\}} \tilde{g}_{mj}\left(s_m\right) \prod_{\substack{kl \\ k \neq \{i,j\} \\ l \neq \{i,j\}}} \tilde{g}_{kl}\left(s_k\right) \tilde{g}_{lk}\left(s_l\right) \right)$$

$$= \tilde{f}_i(s_i)\, \tilde{f}_j(s_j) \prod_{m \neq \{i,j\}} \tilde{g}_{im}\left(s_i\right) \prod_{m \neq \{i,j\}} \tilde{g}_{jm}\left(s_j\right)$$
$$\sum_{\neg(s_i,s_j)} \left(\prod_{t \neq \{i,j\}} \tilde{f}_t(s_t) \prod_{m \neq \{i,j\}} \tilde{g}_{mi}\left(s_m\right) \prod_{m \neq \{i,j\}} \tilde{g}_{mj}\left(s_m\right) \prod_{\substack{kl \\ k \neq \{i,j\} \\ l \neq \{i,j\}}} \tilde{g}_{kl}\left(s_k\right) \tilde{g}_{lk}\left(s_l\right) \right)$$

Each term inside the RHS term's sum is, by definition, a normalized (valid) Bernoulli distribution in its exponential form defined on a latent variable different than s_i and s_j. Thus, using the sum-product rule and the fact that each site is a valid Bernoulli distribution, we get

$$\sum_{\neg(s_i,s_j)} \left(\prod_{t \neq \{i,j\}} \tilde{f}_t(s_t) \prod_{m \neq \{i,j\}} \tilde{g}_{mi}\left(s_m\right) \prod_{m \neq \{i,j\}} \tilde{g}_{mj}\left(s_m\right) \prod_{\substack{kl \\ k \neq \{i,j\} \\ l \neq \{i,j\}}} \tilde{g}_{kl}\left(s_k\right) \tilde{g}_{lk}\left(s_l\right) \right)$$

$$= \left(\prod_{t \neq \{i,j\}} \sum_{s_t} \tilde{f}_t(s_t) \right) \left(\prod_{m \neq \{i,j\}} \sum_{s_m} \tilde{g}_{mi}\,(s_m) \right) \left(\prod_{m \neq \{i,j\}} \sum_{s_m} \tilde{g}_{mj}\,(s_m) \right)$$

$$\left(\prod_{\substack{kl \\ k \neq \{i,j\} \\ l \neq \{i,j\}}} \left(\sum_{s_k} \tilde{g}_{kl}\,(s_k) \right) \ \left(\sum_{s_l} \tilde{g}_{lk}\,(s_l) \right) \right)$$

$$= \left(\prod_{t \neq \{i,j\}} 1 \right) \times \left(\prod_{m \neq \{i,j\}} 1 \right) \times \left(\prod_{m \neq \{i,j\}} 1 \right) \times \left(\prod_{\substack{kl \\ k \neq \{i,j\} \\ l \neq \{i,j\}}} (\,1 \,\times\, 1\,) \right) \;=\; 1$$

Therefore,

$$q_{\neg \tilde{g}_{ij}}\,(s_i, s_j) \;=\; \tilde{f}_i(s_i)\,\tilde{f}_j(s_j) \prod_{m \neq \{i,j\}} \tilde{g}_{im}\,(s_i) \prod_{m \neq \{i,j\}} \tilde{g}_{jm}\,(s_j)$$

$$=\; f_i(s_i)\,f_j(s_j) \prod_{m \neq \{i,j\}} \tilde{g}_{im}\,(s_i) \prod_{m \neq \{i,j\}} \tilde{g}_{jm}\,(s_j)$$

as we are not approximating the singleton factors. Dropping normalizing constants we get

$$q_{\neg \tilde{g}_{ij}}\,(s_i, s_j) \;\propto\; \exp\, b_i\, s_i \;\; \exp\, b_j\, s_j \prod_{m \neq \{i,j\}} \exp\, \theta_{im}\, s_i \prod_{m \neq \{i,j\}} \exp\, \theta_{jm}\, s_j$$

$$\Rightarrow\; q_{\neg \tilde{g}_{ij}}\,(s_i, s_j) \;\propto\; , \exp\,[\,(\, b_i + \sum_{m \neq \{i,j\}} \theta_{im}\,)\; s_i\,] \;\times\; \exp\,[\,(\, b_j + \sum_{m \neq \{i,j\}} \theta_{jm}\,)\; s_j\,]$$

Now, from lectures, we know that

$$\tilde{g}_{ij}^{new}(s_i, s_j) \;=\; \underset{\tilde{g}_{ij}}{\operatorname{argmin}}\; \mathbf{KL}[\; g_{ij}(s_i, s_j)\, q_{\neg \tilde{g}_{ij}}\,(s_i, s_j) \mid \tilde{g}_{ij}(s_i, s_j)\, q_{\neg \tilde{g}_{ij}}\,(s_i, s_j)\;]$$

Observe

$$\tilde{g}_{ij}(s_i, s_j)\, q_{\neg \tilde{g}_{ij}}\,(s_i, s_j) \;\propto\; \exp\,[\,(\; b_i + \theta_{ij} + \sum_{m \neq \{i,j\}} \theta_{im}\;)\; s_i\,] \;\times\; \exp\,[\,(\; b_j + \theta_{ji} + \sum_{m \neq \{i,j\}} \theta_{jm}\;)\; s_j\,]$$

As $\tilde{g}_{ij}\,(s_i, s_j) \;=\; \tilde{g}_{ij}\,(s_i) \;\times\; \tilde{g}_{ji}\,(s_j)$

This gives a product of two Bernoulli's which in turn is an exponential family so we will approximate $\tilde{g}_{ij}\,(s_i, s_j)$ using moment matching.

The expectation of RHS term in KL is given by

$$\Big\langle\, \tilde{g}_{ij}(s_i, s_j)\, q_{\neg \tilde{g}_{ij}}\,(s_i, s_j)\,\Big\rangle_{s_i\, s_j} \;=\; \left\langle \exp\,[\,(\; b_i \;+\; \theta_{ij} \;+ \sum_{m \neq \{i,j\}} \theta_{im}\;)\; s_i\,] \right\rangle_{s_i} \times \left\langle \exp\,[\,(\; b_j \;+\; \theta_{ji} \;+ \sum_{m \neq \{i,j\}} \theta_{jm}\;)\; s_j\,] \right\rangle_{s_j} \qquad (6)$$

Now, the first term on the RHS of (6) is equal to

$$\left\langle \exp\,[\,(\; b_i \;+\; \theta_{ij} \;+ \sum_{m \neq \{i,j\}} \theta_{im}\;)\; s_i\,] \right\rangle_{s_i} \;=\; \frac{1}{Z_{\tilde{g}_{ij}(s_i)}} \sum_{s_i} s_i\,[\exp\,(\; b_i + \theta_{ij} + \sum_{m \neq \{i,j\}} \theta_{im}\;)\; s_i\,]$$

$$=\; \frac{\exp\;(\,b_i \;+\; \theta_{ij} \;+ \sum_{m \neq \{i,j\}} \theta_{im}\,)}{Z_{\tilde{g}_{ij}(s_i)}}$$

And by properties of Bernouilli exponential form,

$$Z_{\tilde{g}_{ij}(s_i)} \;=\; 1 \;+\; \exp\;(\,b_i \;+\; \theta_{ij} \;+ \sum_{m \neq \{i,j\}} \theta_{im}\,)$$

$$\Rightarrow\; \left\langle \exp\,[\,(\; b_i \;+\; \theta_{ij} \;+ \sum_{m \neq \{i,j\}} \theta_{im}\;)\; s_i\,] \right\rangle_{s_i} \;=\; \sigma\,\left(b_i \;+\; \theta_{ij} \;+ \sum_{m \neq \{i,j\}} \theta_{im} \right)$$

where $\sigma(x) = \frac{1}{1+\exp -x} = \frac{\exp x}{1 + \exp x}$ i.e sigmoid function.

Analogously,

$$\left\langle \exp\left[\left(b_j + \theta_{ji} + \sum_{m \neq \{i,j\}} \theta_{jm} \right) s_j \right] \right\rangle_{s_j} = \sigma\left(b_j + \theta_{ji} + \sum_{m \neq \{i,j\}} \theta_{jm} \right)$$

$$\Rightarrow \left\langle \tilde{g}_{ij}(s_i, s_j)\, q_{\neg \tilde{g}_{ij}}(s_i, s_j) \right\rangle_{s_i\, s_j} = \sigma\left(b_i + \theta_{ij} + \sum_{m \neq \{i,j\}} \theta_{im} \right) \sigma\left(b_j + \theta_{ji} + \sum_{m \neq \{i,j\}} \theta_{jm} \right)$$

Let's focus on the LHS term of the KL divergence. RHS of the form

$$g_{ij}(s_i, s_j)\, q_{\neg \tilde{g}_{ij}}(s_i, s_j) \propto \exp\left[w_{ij}\, s_i\, s_j + \left(b_i + \sum_{m \neq \{i,j\}} \theta_{im} \right) s_i + \left(b_j + \sum_{m \neq \{i,j\}} \theta_{jm} \right) s_j \right]$$

But as we saw above with $\left\langle \tilde{g}_{ij}(s_i, s_j)\, q_{\neg \tilde{g}_{ij}}(s_i, s_j) \right\rangle_{s_i\, s_j}$, the first moment wrt s_i, s_j can be split into first moment wrt s_i and first moment wrt s_j, calculated separately. Thus, we will proceed with moment matching but wrt to s_i and then wrt s_j, in a separate process.

To derive the first moment of $g_{ij}(s_i, s_j)\, q_{\neg \tilde{g}_{ij}}(s_i, s_j)$ wrt s_i we need marginalize out s_j and then compute first moment wrt s_i. Marginalizing out s_j we get

$$\exp\left[\left(b_i + \sum_{m \neq \{i,j\}} \theta_{im} \right) s_i \right] + \exp\left[\left(w_{ij} + b_i + \sum_{m \neq \{i,j\}} \theta_{im} \right) s_i + b_j + \sum_{m \neq \{i,j\}} \theta_{jm} \right]$$

We want this expression to be normalized, i.e have a valid probability distribution otherwise we can't compute the first moment. The normalizing constant of this expression, which we will define as $Z(s_i)$ is equal to

$$\begin{aligned} Z(s_i) = {} & \exp\left(b_i + \sum_{m \neq \{i,j\}} \theta_{im} \right) + \exp\left(w_{ij} + b_i + \sum_{m \neq \{i,j\}} \theta_{im} + b_j + \sum_{m \neq \{i,j\}} \theta_{jm} \right) \\ & + \exp\left(b_j + \sum_{m \neq \{i,j\}} \theta_{jm} \right) + 1 \end{aligned}$$

Thus, first moment is equal to

$$\frac{\sum_{s_i} \Big(s_i \ \exp[\,(b_i + \sum_{m\neq\{i,j\}} \theta_{im}\,)\, s_i\,] \ + \ s_i \ \exp[\,(\,w_{ij} + b_i + \sum_{m\neq\{i,j\}} \theta_{im}\,)\, s_i \ + \ b_j \ + \ \sum_{m\neq\{i,j\}} \theta_{jm}\,]\Big)}{Z(s_i)}$$

$$= \ \frac{\exp(\,b_i + \sum_{m\neq\{i,j\}} \theta_{im}\,) \ + \ \exp(\,w_{ij} + b_i + \sum_{m\neq\{i,j\}} \theta_{im} \ + \ b_j \ + \ \sum_{m\neq\{i,j\}} \theta_{jm}\,)}{Z(s_i)}$$

$$= \ \frac{\Big(\exp(\,b_i + \sum_{m\neq\{i,j\}} \theta_{im}\,)\Big)\Big(\,1 \ + \ \exp(\,w_{ij} + b_j + \sum_{m\neq\{i,j\}} \theta_{jm}\,)\Big)}{Z(s_i)}$$

Analogously for s_j we get first moment

$$\frac{\Big(\exp(\,b_j + \sum_{m\neq\{i,j\}} \theta_{jm}\,)\Big)\Big(\,1 \ + \ \exp(\,w_{ij} + b_i + \sum_{m\neq\{i,j\}} \theta_{im}\,)\Big)}{Z(s_j)}$$

Let $\alpha_i \ = \ b_i + \sum_{m\neq\{i,j\}} \theta_{im}$ and $\alpha_j \ = \ b_j + \sum_{m\neq\{i,j\}} \theta_{jm}$ then matching moments for s_i is equivalent to :

$$\frac{\exp(\alpha_i)\,(\,1 \ + \ \exp(\,w_{ij} \ + \ \alpha_j\,)\,)}{Z(s_i)} \ = \ \sigma\,(\,\alpha_i \ + \ \theta_{ij}\,)$$

$$\Leftrightarrow \ (\,\exp(\alpha_i) \ + \ \exp(\,w_{ij} \ + \ \alpha_i \ + \ \alpha_j\,)\,)\,(\,1 \ + \ \exp -(\,\alpha_i \ + \ \theta_{ij}\,)\,) \ = \ Z(s_i)$$

where $Z(s_i) \ = \ \exp(\alpha_i) \ + \ \exp(\,w_{ij} \ + \ \alpha_i \ + \ \alpha_j\,) \ + \ \exp(\alpha_j) \ + \ 1$

$$\Leftrightarrow \ (\,1 \ + \ \exp -(\,\alpha_i \ + \ \theta_{ij}\,)\,) \ = \ 1 \ + \ \frac{\exp\ \alpha_i \ + \ 1}{\exp(\alpha_i) \ + \ \exp(\,w_{ij} \ + \ \alpha_i \ + \ \alpha_j\,)}$$

$$\Leftrightarrow \ \frac{1}{\exp(\,\alpha_i \ + \ \theta_{ij}\,)} \ = \ \frac{\exp\ \alpha_i \ + \ 1}{\exp(\alpha_i) \ + \ \exp(\,w_{ij} \ + \ \alpha_i \ + \ \alpha_j\,)}$$

$$\Rightarrow \exp \theta_{ij} \; = \; \frac{\exp(\alpha_i) \; + \; \exp(\, w_{ij} \; + \; \alpha_i \; + \; \alpha_j\,)}{(\,\exp \alpha_i\,)(\,\exp(\,\alpha_j\,) \; + \; 1\,)}$$

$$\Rightarrow \; \exp \theta_{ij} \; = \; \frac{1 \; + \; \exp(\, w_{ij} \; + \; \alpha_j\,)}{1 \; + \; \exp(\,\alpha_j\,)}$$

Analogously for s_j we get,

$$\exp \theta_{ji} \; = \; \frac{1 \; + \; \exp(\, w_{ij} \; + \; \alpha_i\,)}{1 \; + \; \exp(\,\alpha_i\,)}$$

Where $\alpha_i \; = \; b_i \, + \, \sum_{m \neq \{i,j\}} \theta_{im}$ and $\alpha_j \; = \; b_j \, + \, \sum_{m \neq \{i,j\}} \theta_{jm}$. Finally, the updates for $\tilde{g}_{ij}(s_i, s_j)$ are :

$$\theta_{ij} \; = \; \log\left(\frac{1 \; + \; \exp(\, w_{ij} \; + \; \alpha_j\,)}{1 \; + \; \exp(\,\alpha_j\,)} \right)$$

$$\theta_{ji} \; = \; \log\left(\frac{1 \; + \; \exp(\, w_{ij} \; + \; \alpha_i\,)}{1 \; + \; \exp(\,\alpha_i\,)} \right)$$

5.3

As we are not approximating $f_i\,(s_i)$, we only work with the pairwise factors $g_{ij}\,(s_i, s_j)$. In the last question, we chose to approximate this untractable pairwise factor using the product of two Bernoulli distributions, i.e $g_{ij}\,(s_i, s_j) \; \approx \; \tilde{g}_{ij}(s_i, s_j) \; = \; \tilde{g}_{ij}\,(s_i)\; \tilde{g}_{ji}\,(s_j)$. These Bernouilli distributions are independent, thus this is the same as using factored message approximation, i.e :

$$\tilde{g}_{ij}\,(s_i) \; = \; \mathcal{M}_{j \to i}\,(s_i)$$

$$\tilde{g}_{ji}\,(s_j) \; = \; \mathcal{M}_{i \to j}\,(s_i)$$

Thus, the approximate posterior can be rewritten in the form

$$q(\,\mathbf{s}\,) \; = \; \frac{1}{Z} \prod_i f_i\,(s_i) \prod_{ij} \tilde{g}_{ij}\,(s_i, s_j) \; = \; \frac{1}{Z} \prod_i \left(f_i\,(s_i) \prod_{j \in \, ne(i)} \mathcal{M}_{j \to i}\,(s_i) \right)$$

where $ne(i)$ denotes the neighbors of s_i . Having $j \in ne(i)$ as index for pairwise products given i, is equivalent to our previous notation as Boltzmann machine represents an MRF where latents s_i are all connected (i.e form a complete subgraph).

And so, as we derived in the previous question,

$$q_{\neg \tilde{g}_{ij}}(s_i, s_j) = f_i(s_i)\, f_j(s_j) \prod_{m \neq \{i,j\}} \tilde{g}_{im}(s_i) \prod_{m \neq \{i,j\}} \tilde{g}_{jm}(s_j)$$

can be rewritten as

$$q_{\neg \tilde{g}_{ij}}(s_i, s_j) = \left(f_i(s_i) \prod_{m \in ne(i) \backslash j} \mathcal{M}_{m \to i}(s_i) \right) \left(f_j(s_j) \prod_{m \in ne(j) \backslash i} \mathcal{M}_{m \to j}(s_j) \right)$$

Thus cavity factors, and so, as seen in the Message-based EP lecture

$\mathcal{M}^{new}_{j \to i}(s_i)$ should be equal to $\sum_{s_j} \left(g_{ij}(s_i, s_j)\ f_j(s_j)\ \prod_{m \in ne(j) \backslash i} \mathcal{M}_{m \to j}(s_j) \right)$

$$\propto \sum_{s_j} \exp[\, w_{ij}\, s_i\, s_j + b_j\, s_j + (\sum_{m \in ne(j) \backslash i} \theta_{jm})\, s_j]$$

$$= \sum_{s_j} \exp[\, w_{ij}\, s_i\, s_j + (b_j + \sum_{m \in ne(j) \backslash i} \theta_{jm})\, s_j]$$

$$= \exp[\, w_{ij}\, s_i + b_j + \sum_{m \in ne(j) \backslash i} \theta_{jm}] + 1$$

which has normalizing constant $Z = \exp[\, w_{ij} + b_j + \sum_{m \in ne(j) \backslash i} \theta_{jm}] + \exp[b_j + \sum_{m \in ne(j) \backslash i} \theta_{jm}] + 2$

Now as we defined $\mathcal{M}_{j \to i}(s_i) = \tilde{g}_{ij}(s_i) = \exp[\theta_{ij}\, s_i - \log(1 + \exp \theta_{ij})]$ taking the expectation with respect to s_i of $\frac{\exp[\, w_{ij}\, s_i + b_j + \sum_{m \in ne(j) \backslash i} \theta_{jm}] + 1}{Z}$ is equal to sigmoid of natural parameter θ_{ij} (property of bernouilli exponential form)

Moreover,

$$\sigma(\theta_{ij}) = \frac{\exp[\, w_{ij} + \alpha_j \,] + 1}{\exp[\, w_{ij} + \alpha_j \,] + \exp[\alpha_j \,] + 2}$$

Analogously for $\mathcal{M}^{new}_{i\to j}(s_j)$ we get

$$\sigma(\theta_{ji}) = \frac{\exp[\, w_{ij} + \alpha_i \,] + 1}{\exp[\, w_{ij} + \alpha_i \,] + \exp[\alpha_i \,] + 2}$$

This gets us the same updates for pairwise approximating sites in terms of their natural parameters after applying the logit function (inverse of the sigmoid function) to the two expressions above. As you can see this leads to a messaged-based EP where were only approximate the messages and no two separate approximate like in the normal EP.

Thus, after running this message passing algorithm "sufficiently", the full joint is approximated by the fully factorized distribution

$$q(\mathbf{s}) = \frac{1}{Z}\prod_{i=1}^{K} \exp b_i\, s_i \prod_{ij} \exp \theta_{ij}\, s_i \;\exp \theta_{ij}\, s_i$$

$$\Rightarrow q(s_i) \propto \exp b_i\, s_i \prod_{m\neq\{i,j\}} \exp \theta_{im}\, s_i = \exp[\,(b_i + \sum_{m\neq\{i,j\}} \theta_{im})\, s_i\,]$$

As the singleton factors are Bernoulli distributions and we defined each pairwise site to be the product of two independent Bernoulli distributions, then the approximate posterior distribution $q(\mathbf{s})$ (just like in the Mean-field question) is equal to the product of K independent Bernouilli distributions :

$$q(\mathbf{s}) = \prod_{i=1}^{K} \lambda_i^{s_i}\,(1 - \lambda_i)^{1 - s_i}$$

$$\Rightarrow q(s_i) = \lambda_i^{s_i}\,(1 - \lambda_i)^{1 - s_i}$$

$$\Rightarrow \lambda_i^{s_i}\,(1 - \lambda_i)^{1 - s_i} \propto \exp[\,(b_i + \sum_{m\neq\{i,j\}} \theta_{im})\, s_i\,]$$

$$\Rightarrow \eta_i = b_i + \sum_{m\neq\{i,j\}} \theta_{im} \qquad \forall\, i = 1, \ldots, K$$

Where η_i is the natural parameter of $q(s_i)$ i.e $\eta_i = \log \frac{\lambda_i}{1-\lambda_i}$

The introduction of an approximate pairwise factor as a product of two independent Bernoulli distributions in the Boltzmann Machine neglects the original dependencies between nodes encoded in the pairwise factor. This approximation replaces the pairwise factors with pairs of singleton factors, and the subsequent application of EP adjusts these singleton factors to approximate the original pairwise factors based on incoming messages. In a fully connected graph without a tree-like structure, such as the Boltzmann Machine, the messages resulting from these approximations are not truly disjoint. This lack of disjointness leads to a loopy BP algorithm. Unlike a tree structure where messages can be separated within disjoint areas, the fully connected graph allows messages to circulate through cycles, introducing loops in the BP algorithm.

In other words, approximating pairwise factor by a product of messages :

$$g_{ij}(s_i, s_j) \approx \mathcal{M}_{j\to i}(s_i)\, \mathcal{M}_{i\to j}(s_j)$$

Where

$$\mathcal{M}_{j\to i}(s_i) = \sum_{s_j} \left(g_{ij}(s_i, s_j)\, f_j(s_j) \prod_{m \in ne(j)\setminus i} \mathcal{M}_{m\to j}(s_j) \right)$$

$$\mathcal{M}_{i\to j}(s_j) = \sum_{s_i} \left(g_{ij}(s_i, s_j)\, f_i(s_i) \prod_{m \in ne(i)\setminus j} \mathcal{M}_{m\to i}(s_i) \right)$$

Which is exactly the expression for belief propagation messages, leads to a loopy BP as, the Boltzmann machine (joint distribution) represents an undirected non-tree factor graph. Consequently, $ne(j) \setminus i \cap ne(i) \setminus j \neq \varnothing$ and in our case (Boltzmann machine) $ne(j) \setminus i = ne(i) \setminus j \;\; \forall\, ij$ as latents form a complete graph of K vertices. This presence of loops implies that messages will pass through cycles making them revisit nodes multiple times leading to a loopy BP, as the graph doesn't have a tree-like structure (i.e it possesses collider nodes so one bottom up top down pass isn't enough), and so the incoming messages described above aren't disjoint.

5.4

To select the number K in a Bayesian manner, we employ Automatic Relevance Determination (cf. Q4). In this approach, Gaussian priors are independently placed on each column of the matrix μ (i.e. μ_i), as explained in question 4. The expectation step for approximating $q(\mathbf{s})$ involves utilizing the previously described loopy BP algorithm. Subsequently, the VB M-like step, derived in question 4a, is applied to obtain the posterior distribution of μ_i 's and point estimates for parameters π and σ given in the Mean-Field M-step appendix. The ARD method is then employed with a hyper M-step to determine optimal α_i 's is using fixed-point equations at each iteration (cf. 4). The estimation of latent space dimensionality is therefore achieved by counting the number of α_i 's that remain finite through the iterations.

The primary computational challenges in this context stem from the loopy nature of the Boltzmann machine and the associated difficulties in achieving convergence during the "EM"-like algorithm iterations. The loopy BP, which replaces the E-step, faces insurmountable obstacles in reaching convergence due to the interconnectedness of all latent variables in the Boltzmann machine. The complex dependencies created by this fully connected graph make it infeasible for the E-step to converge (Boltzmann machine graphical representation is very loopy). As a consequence, the lack of convergence in the E-step cascades through subsequent stages of the algorithm, affecting both the M-step and the hyper M-step. This persistent fluctuation observed in the E-step hinders the ARD algorithm's convergence, creating difficulties in ensuring that the hyperparameters α_i consistently either diverge or remain finite. This variability poses challenges in decisively identifying which latent dimensions should undergo pruning. The intricacies of the loopy BP algorithm amplify the continuous variability, particularly given the exponential form of the approximate messages.

To address the convergence challenges associated with the loopy BP, employing Power EP emerges as a potential solution. Power EP introduces a power parameter β that influences the blending of messages with previous site approximations during local contextual minimization. Utilizing Power EP helps stabilize the E-step loopy BP, as small values of β i.e ($\beta < 1$) lead to more stable updates and improved convergence behavior. While convergence is not guaranteed even with Power EP, the updates become less volatile, settling within a range of values. To determine an appropriate stopping criterion, one could monitor the free energy given by the VB and observe its convergence within a certain range of values. If the free energy remains within this range for a predefined number of iterations, the algorithm could be halted, and the latent dimensions corresponding to the highest value within the range could be selected. Alternatively, multiple runs of the algorithm with different initializations could be conducted, and the best results could be averaged to enhance reliability, although this approach might pose computational challenges.

Another computational challenge tied to the Boltzmann machine arises from its demanding computational requirements, particularly when dealing with large values of K. The posterior distribution, encompassing 2^K possible states, introduces an exponential escalation in the number of latent dimensions, rendering the computation intractable. To tackle this challenge, a potential remedy involves the exploration of grouping latent variables into super nodes. By adopting this strategy, the total number of latent variables is reduced, alleviating the computational burden associated with the exponential growth in the latent space. Grouping latent variables into supernodes could offer a pragmatic approach to enhancing computational efficiency while maintaining the modeling capabilities of the Boltzmann machine. Furthermore, an additional approach could be to set a subset of pairwise interactions to zero. While this may result in less accurate approximations (like for the supernodes strategy), strategically pruning certain interactions could potentially provide an attractive trade-off between computational efficiency and model precision. This selective reduction of pairwise interactions could allow a reduction in computational complexity without sacrificing the overall effectiveness of the Boltzmann machine.

A Appendix: Code to plot features Mean-Field

```
K = 8
iterations = 500

# Run the EM algorithm
mu, sigma, pie = LearnBinFactors(Y, K, iterations)

# Visualize the learned features mu
plt.figure(figsize=(8, 8))
for k in range(K):
    plt.subplot(4, 4, k + 1)
    plt.imshow(np.reshape(mu[:, k], (4, 4)), cmap='gray',
    interpolation='none')
    plt.axis('off')
plt.show() #note default imshow displays black as values close to 0
     and white to values close to 1
```

B Appendix: Code to plot features VB ARD

```
# Plot for VB free energy
cmap = plt.get_cmap('viridis', 5)

# Free energies plot
plt.figure(figsize=(12, 10))
for idx, K in enumerate(K_values):
    if K < 14:
        # Use default colors for K=4 to K=13
        plt.plot(free_energies_all_K[idx], label=f'K={K}')
    else:
        # Use viridis colormap for K=14 to K=18
        color = cmap(K-14)  # Offset by 14 as the colormap starts
    from K=14
        plt.plot(free_energies_all_K[idx], color=color, label=f'K={
    K}')

plt.title('VB Free Energy over Iterations for Different K')
plt.xlabel('Iteration')
plt.ylabel('Free Energy')
plt.legend()
plt.savefig('free_energies_plot.png')
plt.show()


# Visualize Learned Features mean_mu for each K, sorted by alpha
    values
for idx, K in enumerate(K_values):
    mean_mu = final_mean_mu_all_K[idx]
    alphas = final_alphas_all_K[idx]

```

```python
    # Sort indices by alpha values
    sorted_indices = np.argsort(alphas)

    grid_cols = int(math.ceil(math.sqrt(K)))
    grid_rows = int(math.ceil(K / grid_cols))

    plt.figure(figsize=(4, 4))
    for k in sorted_indices:
        plt.subplot(grid_rows, grid_cols, np.where(sorted_indices == k)[0][0] + 1)
        plt.imshow(np.reshape(mean_mu[:, k], (4, 4)), cmap='gray', interpolation='none')
        plt.title(f' ={alphas[k]:.2f}')
        plt.axis('off')
    plt.tight_layout()
    plt.show()


# Plot Effective Number of Factors for each K
for idx, K in enumerate(K_values):
    effective_factors = effective_factors_all_K[idx]

    plt.figure(figsize=(10, 4))
    plt.plot(effective_factors, label=f'K={K}', marker='o')
    plt.title(f'Effective Number of Factors over Iterations for K={K}')
    plt.xlabel('Iteration')
    plt.ylabel('Effective Factors')
    plt.ylim([0, max(effective_factors) + 1])  # Adjust the y-axis limits
    plt.legend()
    plt.grid(True)
    plt.show()
```