University of Kent

MASTER OF SCIENCE THESIS

A Copula Augmentation of the Mean-Field Approach to Automatic Differentiation Variational Inference

Author:
Daniel William SCOTT

Supervisors:
Dr. Eduard CAMPILLO-FUNOLLET
Dr. Bruno SANTOS

A thesis submitted in fulfillment of the requirements for the degree of Master of Science (MSc) in Statistics

in the

Statistics Research Group
School of Mathematics, Statistics and Actuarial Science

September 14, 2023

Declaration of Authorship

I, Daniel William SCOTT, declare that this thesis titled, "A Copula Augmentation of the Mean-Field Approach to Automatic Differentiation Variational Inference" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: DWScott

Date: September 14, 2023

UNIVERSITY OF KENT

Abstract

Division of Computing, Engineering and Mathematical Sciences

School of Mathematics, Statistics and Actuarial Science

Master of Science (MSc) in Statistics

A Copula Augmentation of the Mean-Field Approach to Automatic Differentiation Variational Inference

by Daniel William SCOTT

A common problem in Bayesian statistics is in evaluating a posterior density when we have access to the corresponding prior and likelihood, but have no practical means of evaluating the corresponding marginalisation. The Markov chain Monte Carlo (MCMC) algorithm is the algorithm that has traditionally been used to find approximations to such difficult-to-compute posterior densities. However, in recent years, variational inference (VI), an alternative approximation algorithm with the same function and objectives as MCMC, has been shown to generally be significantly faster and less computationally expensive than MCMC (and many other such alternative approximation algorithms). Yet VI is generally lacking the development and academic rigor that MCMC and the likes have been subjected to over the years.

In this thesis, an original contribution to a branch of research into VI will be made. Two recently developed variants of VI that attempt to address specific limitations concerning the general computational complexity and accuracy of VI, are automatic differentiation variational inference (ADVI) and copula VI, respectively. In this thesis, a new variant of VI, that will combine elements of the two aforementioned variants, and be referred to in this thesis as "copula ADVI", will be developed, empirically tested and evaluated. The empirical testing and evaluation of copula ADVI will seek to unveil whether combining automatic differentiation and copulas helps to effectively address the specific limitations of VI that have previously been identified, and whether such a combination of developments could offer a promising future potential for VI in general.

Acknowledgements

I would primarily like to pay tribute to my supervisors: Dr Eduard Campillo-Funollet and Dr. Bruno Santos. These individuals, throughout the duration of the Master of Science course to which this thesis is attributed, have demonstrated unwavering and outstanding support and guidance in my pursuit of achieving the aims and objectives of this course. I am very grateful for the hard work and effort of my supervisors and my academic guide.

I would also like thank other academics and supporting staff at the University of Kent, together with the authors of various third-party online learning resources, for their guidance and help in: understanding background material; supporting my general well being; and progressing my research.

To name but a few of these academics, supporting staff & authors:

- Dr. Diana Cole: Supervisory Chair of my Master of Science Course
- Dr. Steffen Krusch: Senior Lecturer in Applied Mathematics & Postgraduate Leader at the University of Kent
- Ms. Siobhan McGhee: Disability Adviser at the University of Kent
- Mr. David Refaeli: Founder of, and Commentator at, Meerkat Statistics
- Dr. Bo Chang: Independent Blogger
- Mr. Mark Saroufim: Independent Blogger

Contents

List of Abbreviations

$\mathcal{N}$	Gaussian/Normal Density Function
Φ	Gaussian/Normal Cumulative Distribution Function
$\mathbb{Z}^+$	$\{0\} \cup \mathbb{N}$
$\mathbb{R}^+$	$[0, \infty)$
$\mathbb{R}^{++}$	$(0, \infty)$
VI	**V**ariational **I**nference
KL divergence	**K**ullback-**L**eibler divergence
MCMC	**M**arkov **C**hain **M**onte **C**arlo
ELBO	**E**vidence **L**ower **BO**und
MFVI	**M**ean-**F**ield **V**ariational **I**nference
CAVI	**C**oordinate **A**scent **V**ariational **I**nference
GMM	**G**aussian **M**ixture **M**odel
SVI	**S**tochastic **V**ariational **I**nference
ADVI	**A**utomatic **D**ifferentiation **V**ariational **I**nference
MC integration	**M**onte **C**arlo integration
CDF	**C**umulative **D**istribution **F**unction
BBVI	**B**lack **B**ox **V**ariational **I**nference
IDE	**I**ntegrated **D**evelopment **E**nvironment
MNIST	**M**odified **N**ational **I**nstitute of **S**tandards and **T**echnology
PCA	**P**rinciple **C**omponent **A**nalysis
vector	column vector

1 Background Subjects & Material

1.1 An Introduction to Variational Inference

This section introduces the core ideas and methodology of the variational inference method: A method from machine learning and statistics that is used to approximate complex and/or difficult-to-compute probability densities. This method underpins the research conducted in this course and is an omnipresent topic throughout this thesis. Variational inference is commonly abbreviated to **VI**.

The term "Inference" is defined simply as a conclusion reached on the basis of evidence and reasoning. In a statistical framework, this amounts simply to extracting information from data. In the context of **VI**, the term "Variational" refers to "Calculus of Variations" (Jordan et al., 1999) – a branch of calculus that pertains to the optimisation (finding the maxima or minima) of functionals.

The precise definition of a functional can vary slightly depending on the context. When considered in terms of **VI**: A functional is defined as a real valued function, on a space of functions with a domain in $\mathbb{R}^m$ (for any $m \in \mathbb{N}$) and a codomain in $\mathbb{R}$. Let V denote the space of all functions such that $v : \mathbb{R}^m \to \mathbb{R}$, where v is a generic member of V. A function U defined as $U : V^n \to \mathbb{R}$ (for any $n \in \mathbb{N}$), within the scope of **VI**, would then be a functional (functionals are usually denoted by capital letters). Henceforth, the notational convention of specifying the arguments of a functional within square brackets (as opposed to parentheses, for regular functions) will be adopted - i.e. if v_1 is some member of V, such that $v_1(x) \in \mathbb{R}$ for any $x \in \mathbb{R}$, then $U[v_1] \in \mathbb{R}$.

Now consider a function $f : \mathbb{R} \to \mathbb{R}$ and variables $z, y \in \mathbb{R}$, such that $f(z) = y$. Inference in this context amounts simply to establishing the value of y, for any given precisely defined value of z. Let's say, though, that f does not have a tractable closed-form expression, yet we require one for a particular application. Let Q be the space of all closed-form expressions within V (so $Q \subset V$) that would be considered tractable for our application. We thus need to find a member of Q that could be used to approximate f up to an accuracy that would still be considered appropriate for our application - much like how a Taylor series expansion, up to a suitable finite degree, can be used to approximate a non-polynomial function, such as $sin(x)$, in the polynomial space (consider the finite Taylor series expansion, about zero, of $sin(x)$ in equation 1.1 for reference).

$$\sin(x) \approx x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots + \frac{(-1)^n x^{2n+1}}{(2n+1)!} = \sum_{i=0}^{n} \frac{(-1)^i x^{2i+1}}{(2i+1)!}, \text{ for some } n \in \mathbb{Z}^+ \tag{1.1}$$

To find the member of Q that best approximates and represents f, denoted by, say, q^*, a measure of proximity is required between q - a generic member of Q - and f.

Let such a measure of proximity, denoted by, say, s, be in $\mathbb{R}^+$ and take 2 parameters from the space V. The proximity measure s shall be defined, in an intuitive sense, like a distance, such that: The smaller the value of $s(v_1, v_2)$ (where $v_1, v_2 \in V$), the more "similar" v_1 and v_2 are; where $s(v_1, v_2) = 0$ if, and only if, $v_1 = v_2$. The function q^* is then given as the member of Q with the smallest value of s to f. Mathematically, this can be written as:

$$q^* = \arg\min_{q \in Q} s[f, q] \tag{1.2}$$

As $s : V^n \in \mathbb{R}$, where $n = 2$, then s is a functional in the context of **VI**. We thus solve the variational optimisation problem of equation 1.2 in order to make "Inferences" about f using q^* - the best tractable closed-form representative that we have for f. In a broad sense, this is what **VI** is, considering each of its constituent terms ("Variational" and "Inference").

There are a number of axioms that constitute the definition of **VI**, that are not implied by its constituent terms (“Variational” or “Inference”) but have rather attached themselves to **VI** as the method has evolved within the fields of statistics and machine learning. Firstly, **VI** is a method for approximating, not just arbitrary functions, but specifically "probability densities" (Blei, Kucukelbir, and McAuliffe, 2017; Jordan et al., 1999; Wainwright, Jordan, et al., 2008). Note that the total (hyper) volume under all probability density (hyper) surfaces must integrate to one, and such functions must have an image set contained within $\mathbb{R}^+$. Furthermore, **VI** is used specifically and purposefully as an "approximation" method (Blei, Kucukelbir, and McAuliffe, 2017; Jordan et al., 1999) – **VI** may provide an exact solution in some circumstances (a “perfect” approximation) but obtaining an approximation should always be the objective.

Given the two axioms outline above, let the spaces V and Q now be restricted such that they each only consider probability density functions: V now denotes the space of all probability density functions such that $v : \mathbb{R}^m \rightarrow \mathbb{R}$, where v is a generic member of V; and Q now denotes the space of all closed-form probability density functions within V ($Q \subset V$) that would be considered tractable for our application. Furthermore, let f now be a probability density function.

In theory, if Q were to encompass all closed-form probability density functions, q^* would be equal to f. This theoretical principle is analogous to the equality of the $sin(x)$ function – a non-polynomial function – to its infinite Taylor series expansion – a polynomial equation of infinite degree. For reference, the infinite Taylor series expansion, about zero, of $sin(x)$ is given in the following equation:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots = \sum_{i=0}^{\infty} \frac{(-1)^i x^{2i+1}}{(2i+1)!} \tag{1.3}$$

However, Q encompassing all closed-form probability density functions would never be the case in practice - otherwise, we would not be “approximating” the probability density f (Blei, Kucukelbir, and McAuliffe, 2017; Jordan et al., 1999; Wainwright, Jordan, et al., 2008). In fact, in practice, for the majority of applications, Q only takes one probability density family, parameterised by free variational parameters. For example, the family all univariate Gaussians, parameterised by the mean μ and standard deviation σ of a univariate Gaussian. This is done in order to considerably reduce the computational expense of solving the optimisation problem of equation 1.2 - optimising over vectors/scalars is considerably easier and less computationally expensive than optimising over functions (not technically "Calculus of variations" anymore, but the name has stuck). The reach of the family Q manages the complexity of this optimisation (Blei, Kucukelbir, and McAuliffe, 2017). In general, the variational family Q is chosen, based on information related to the application in question.

Although it is not strictly necessary for Q to only take one variational family, this common practice will be adopted throughout the rest of this thesis.

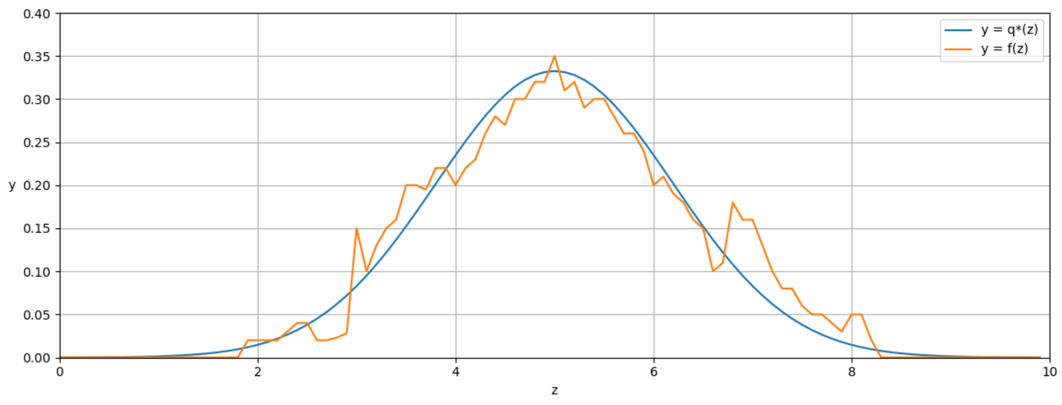


FIGURE 1.1: Closely Matching Probability Density Functions f and q^*

Figure 1.1 provides a visual illustration of **VI** in practice. As outlined above, we require a tractable closed-form expression for our particular application, but this is not provided by the current function that we have – f. The probability density f either has a closed-form expression that is too complicated to be useful or lacks one completely. Given the shape of the function f in figure 1.1, f could be adequately approximated by a far simpler and more tractable univariate Gaussian distribution (a judgement of what is adequate depends on the application in question).

The problem that now remains is to establish exactly which univariate Gaussian best represents and approximates f. To achieve this, we first set Q to be the family of all possible univariate Gaussians, parameterised by μ and σ (the mean and standard deviation, respectively, of a generic univariate Gaussian). It is then just a question of solving the optimisation of equation 1.2, over the free variational parameters μ and σ. In this specific application, the optimisation can be expressed as:

$$q^* = \underset{\mu \in \mathbb{R}, \sigma \in \mathbb{R}^{++}}{\arg\min} \; s[f(z), q(z)], \text{where } q(z) = \mathcal{N}(z; \mu, \sigma) \tag{1.4}$$

We can then use q^* to make approximate inferences that we would have otherwise used f for, were it not intractable. This is the basis of what the method of **VI** is.

1.2 The Kullback-Leibler Divergence

This section provides an outline of what the Kullback-Leibler divergence is and how it relates to **VI**. The term "Kullback-Leibler divergence" is commonly abbreviated to

"**KL** divergence".

The **KL** divergence is an information-based measure of disparity among two probability densities (Kullback and Leibler, 1951; Joyce, 2011) - in layman's terms, the **KL** divergence is a numerical measure of the information gained about one probability density from another. The smaller the numerical value of the measure, the more information is gained. The domains of the probability densities, for which the **KL** divergence is measured, must be equal.

Let p and q be two probability density functions on $\mathbb{R}$. The **KL** divergence from p to q, expressed in one of its simplest and most general forms, is defined as:

$$\mathcal{KL}(p||q) = \int p(x) \log \left(\frac{p(x)}{q(x)} \right) dx \tag{1.5}$$

If p and q are discrete probability density functions, each on a discrete set X, then the **KL** divergence from p to q can be equivalently expressed as:

$$\mathcal{KL}(p||q) = \sum_{x \in X} p(x) \log \left(\frac{p(x)}{q(x)} \right) \tag{1.6}$$

Let's say that p and q now both have a support of $\{(x, y) \in \mathbb{R}^2 : x \geq -1, 2 \leq y < 7\}$. A full expression and definition of the KL divergence from p to q could then be given by:

$$\mathcal{KL}(p||q) = \int_{-1}^{\infty} \int_{2}^{7} \left(p(x,y) \log \left(\frac{p(x,y)}{q(x,y)} \right) dy \right) dx \tag{1.7}$$

Given more elaborate and complex domains of the arguments of the **KL** divergence, a full expression of the corresponding **KL** divergence could be established in much the same way as in the examples above (these examples could be used as templates).

As outlined in section 1.1, **VI** requires a measure of proximity between two probability densities, that we denoted by s, to be specified. The **KL** divergence is an example of one such proximity measure that could be used. For reasons that will be elaborated on in proceeding sections, the **KL** divergence is a very useful and influential proximity measure for **VI**. Key characteristics of the **KL** divergence have allowed for revolutionary developments within **VI**, in addressing core problems related to speed and computational expense in modern machine learning.

Two highly noteworthy properties of the **KL** divergence, that provide an intuitive sense of how to interpret the numerical value of $\mathcal{KL}(p||q)$ as a divergence from p to q, are:

i) $\mathcal{KL}(p||q) \geq 0$, with $\mathcal{KL}(p||q) = 0$ if, and only if, $p = q$ (Joyce, 2011).

ii) $\mathcal{KL}(p||q) \neq \mathcal{KL}(q||p)$. That is, in general, it is not symmetric (Joyce, 2011).

In terms of information gain, the fact that $\mathcal{KL}(p||q) = 0$ implies $p = q$, means that when $\mathcal{KL}(p||q) = 0$, the information gained about q from p is enough to establish that $q = p$, which is the most information that can be gained about q from p alone. Likewise, when we have $p = q$, then having p means we know q exactly, which, again, is the most information that can be gained about q from p alone. Thus $p = q$

implies $\mathcal{KL}(p||q) = 0$, as $\mathcal{KL}(p||q) \geq 0$ and "the smaller the numerical value of the measure, the more information is gained" - Maximum information is gained when $p = q$, so $\mathcal{KL}(p||q)$ takes its minimum value of zero.

The **KL** divergence is not a "distance metric", as it is non-symmetrical. It is thus important to refer to the **KL** divergence as a divergence, proximity, or difference (or similar), but not as a "distance". However, for the benefit of a visual intuition, the KL divergence can be considered, in some sense, like a distance (in 2D space): $\mathcal{KL}(p||q) \geq 0$; p and q are more "similar", the smaller the value of $\mathcal{KL}(p||q)$ or $\mathcal{KL}(q||p)$; and $\mathcal{KL}(p||q) = \mathcal{KL}(q||p) = 0$ if, and only if, $p = q$.

Given the definition of the expectation, an equivalent expression and definition of $\mathcal{KL}(p||q)$, that will aid calculations and computations in proceeding work, is:

$$\mathcal{KL}(p||q) = \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\left[\log\left(\frac{p(\mathbf{z})}{q(\mathbf{z})}\right)\right] \tag{1.8}$$

Most recent and practical applications of **VI** use the **KL** divergence as the required proximity measure. The use of the **KL** divergence as the proximity measure in **VI** is so common, that many statisticians assume that the **KL** divergence is a necessary component of **VI**. However, it is not necessary for a given **VI** method to adopt the **KL** divergence as its proximity measure in order to be a valid **VI** method.

Henceforth in this thesis, the **KL** divergence will be used exclusively as the proximity measure for **VI** – The variants of **VI** that underpin the research topic in this thesis necessarily require the **KL** divergence.

1.3 Bayesian Variational Inference

This section discusses the broad definition of "Bayesian Variational Inference" (or "Bayesian **VI**"), as well as the precise Bayesian framework for **VI** that is generally considered the paradigm of Bayesian **VI**.

The form of Bayes' theorem that will henceforth be used is:

$$p(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{p(\mathbf{x})} \tag{1.9}$$

where:

i) $\mathbf{z}$ is a vector of the latent variables

ii) $\mathbf{x}$ is a vector of the observations

iii) $p(\mathbf{z}|\mathbf{x})$ is the posterior probability density

iv) $p(\mathbf{x}|\mathbf{z})$ is the likelihood probability density

v) $p(\mathbf{z})$ is the prior probability density

vi) $p(\mathbf{x})$ is the evidence probability density (also known as the: marginalisation, marginal and marginal likelihood)

The latent variables and observations, denoted by $\mathbf{z}$ and $\mathbf{x}$, are members of the sets $\mathbb{R}^m$ and $\mathbb{R}^n$, respectively ($\mathbf{z} \in \mathbb{R}^m$ & $\mathbf{x} \in \mathbb{R}^n$), for some $n, m \in \mathbb{N}$. That is, $\mathbf{z}$ and $\mathbf{x}$ are

m-dimensional and n-dimensional numerical vectors, respectively. All information to which Bayes' theorem is applied must be encoded numerically via the numerical vectors $\mathbf{z}$ and $\mathbf{x}$, as appropriate.

A very common problem in modern Bayesian statistics is to infer the posterior probability density $p(\mathbf{z}|\mathbf{x})$ when the evidence $p(\mathbf{x})$ is intractable. We can't calculate $p(\mathbf{z}|\mathbf{x})$ using Bayes' theorem if we don't have an (practically) accessible expression for $p(\mathbf{x})$: Given Bayes' theorem alone, if $p(\mathbf{x})$ is intractable, then so is $p(\mathbf{z}|\mathbf{x})$. It is this very problem that methods such as **VI** and **MCMC** – Markov Chain Monte Carlo (Hastings, 1970; Metropolis et al., 1953) – are commonly employed to resolve. How the problem of an intractable evidence $p(\mathbf{x})$ arises in practice, and why this problem is so common, will be outlined in detail in section 1.4.

Bayesian **VI** predominantly refers to a Bayesian based **VI** framework defined in such a way as to seek a resolution to this 'very common problem in modern Bayesian statistics': It is within this broad framework that Bayesian **VI** will be defined in the following discussion. However, the term "Bayesian **VI**" is technically a very general and broad term: Any **VI** method that has been applied in such as way that it somehow relates to Bayesian statistics or Bayes' theorem can be considered Bayesian **VI**.

Bayesian **VI** amounts to posing a family of probability densities Q, that have an adequately/relatively simple closed-form expression and that we suspect take the general form of $p(\mathbf{z}|\mathbf{x})$, and then finding the member of this family $q^*(\mathbf{z})$ that minimises the **KL** divergence to the exact posterior. That is, we solve the optimisation problem:

$$q^*(\mathbf{z}) = \operatorname*{arg\,min}_{q(\mathbf{z})\in Q} \mathcal{KL}\left(q(\mathbf{z})||p(\mathbf{z}|\mathbf{x})\right) \tag{1.10}$$

For any given Bayesian **VI** model, the observations $\mathbf{x}$ are fixed – we can't change what we have observed. The vector $\mathbf{x}$ is thus effectively a constant: $p(\mathbf{z}|\mathbf{x})$ is only a function of $\mathbf{z}$. It is for this reason that the members of Q are only parameterised by $\mathbf{z}$ - $q(\mathbf{z}) \in \mathbb{R}$ for $q \in Q$.

Consider, once again, the example of an application of **VI** depicted in figure 1.1. Let's now assume that f is an intractable posterior, and the horizontal axis represents a one-dimensional latent vector (scalar) z. It was established in section 1.1 that a suitable family of probability densities, from which a suitable approximation of f could be found, is the one-dimensional Gaussian family: $Q = \mathcal{N}(\mu, \sigma)$, parameterised by $\mu \in \mathbb{R}$ and $\sigma \in \mathbb{R}^{++}$. Taking the **KL** divergence as the proximity measure, the associated **VI** optimisation can be expressed as:

$$q^*(z) = \operatorname*{arg\,min}_{\mu\in\mathbb{R},\sigma\in\mathbb{R}^{++}} \mathcal{KL}\left(q(z)||p(z|\mathbf{x})\right), \text{where } q(z;\mu,\sigma) = \mathcal{N}(z;\mu,\sigma) \tag{1.11}$$

Within the field of approximate Bayesian inference, **VI** falls into the class of optimization-based approaches (Bishop and Nasrabadi, 2006; Zhang et al., 2018; Hennig, 2011). Precisely how the resulting optimisation problems in equations 1.10 and 1.11 are approached and solved, for varying applications, will be outlined in proceeding sections.

Employing variational techniques for Bayesian inference arguably first came from the paper “A Mean Field Theory Learning Algorithm for Neural Networks” (Blei, Kucukelbir, and McAuliffe, 2017; Anderson and Peterson, 1987). The scope of variational techniques in Bayesian statistics has since grown in prominence and substance, given ongoing research into their intriguing potential in addressing ongoing limitations in increasingly complex and large-scale Bayesian frameworks. Bayesian **VI** is one such variational technique that is becoming increasingly distinguished, particularly over other algorithms that have the same Bayesian inference objective, such as **MCMC**, as we shall see in section 1.4.

Just as for the case of the **KL** divergence being used as the proximity measure for **VI**, **VI** applied via the Bayesian framework outlined above (attempting to approximate $p(\mathbf{z}|\mathbf{x})$ using a member of Q, given an intractable $p(\mathbf{x})$) is so common, that many statisticians assume that this Bayesian framework is an invariable feature of **VI**. As an indication of just how common this framework is, consider the paper "Advances in Variational Inference" (Zhang et al., 2018), where they state “Classical Variational Inference aims at determining a variational distribution $q(\mathbf{z};\boldsymbol{\lambda})$ that is as close as possible to the posterior $p(\mathbf{z}|\mathbf{x})$, measured in terms of the Kullback-Leibler divergence. Minimizing the Kullback-Leibler divergence to zero would guarantee that the variational distribution matches the exact posterior.”. Let it be emphasised once again that **VI** amounts only to what was outlined in section 1.1: This Bayesian framework is not a necessary feature of **VI**.

Henceforth, as a means of more effectively focusing on the specific research topic of this thesis, only **VI** in the context of the Bayesian framework and setting outlined above will be considered, unless stated otherwise.

1.4 Why is Variational Inference Used?

So far the most general outline and definition of **VI** has been provided (section 1.1), together with what would generally be considered two of its most common and important optional conventions: Using the **KL** divergence as the proximity measure (section 1.2); and considering a Bayesian ‘posterior approximation’ objective, given the ’intractable evidence and posterior’ framework (section 1.3). It is through this branch of **VI** - that will be considered implicitly in this thesis (i.e., unless stated otherwise) - that an appreciation and understanding of precisely why **VI** is used can be obtained.

VI is an approximation algorithm – it uses optimisation to approximate a probability density. What is implied by the fact that **VI** is an approximation algorithm, is precisely what **VI** is (primarily) used for. When we have a probability density that is difficult, or even impossible, to find a tractable closed-form expression for, **VI** can be used to find an alternative approximate (close enough) probability density that can be used instead: One that is far simpler and easier to work with. For many applications, the benefit of working with a far simpler and less computationally expensive approximate probability density outweighs the cost in the loss of accuracy that stems from the approximation itself. In fact, in some circumstances, an approximation based fitting of the target density can lead to more accurate predictions of what the target density is ultimately modelling, due to a reduction in an often very large variance associated with the target density. For example, consider a case where the underlying data that resulted in probability density f in figure 1.1 actually have an underlying Gaussian relationship.

To outline why Bayesian approximation methods, such as Bayesian **VI**, are primarily used as an alternative to Bayes' Theorem directly, first consider the following (simple & contrived) "one-dimensional Exponential-Normal" Bayesian model:

$$Z \sim Exp(\lambda = 1) \quad \text{and} \quad X \sim \mathcal{N}(\mu = Z, \sigma = 1) \tag{1.12}$$

where Z is a random latent variable and X is a random observed variable, both of which are scalars (one-dimensional vectors) in this one-dimensional case.

The model outlined in 1.12 has 2 deterministic hyperparameters: $\lambda = 1$ for the associated Exponential density, and $\sigma = 1$ for the associated Gaussian density. We are assuming that, given a "training dataset" of multiple observations $\{X_i\}$ and their corresponding latent target labels $\{Z_i\}$, the Bayesian prior $p(Z) = Exp(Z; \lambda = 1)$ and the Bayesian likelihood $p(X|Z) = \mathcal{N}(X; \mu = Z, \sigma = 1)$ were established as being the "most befitting" for the overall model (the model is $p(X, Z) = p(Z) * p(X|Z)$) of the training data.

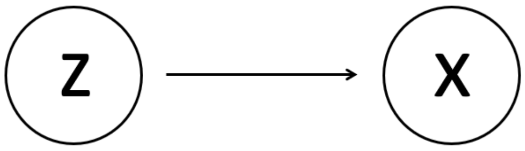


FIGURE 1.2: A Directed Graphical Model from a random latent variable Z to an random observed variable X in the one-dimensional case

The establishment of model 1.12, via the given prior and likelihood, is depicted by the directed graphical model in figure 1.2. The ultimate objective of this model is to make inferences 'in the other direction' (i.e., to test or make predictions): Infer a Z given an (previously unseen) X. To achieve this objective, we require the posterior.

As a substantial example of what the probability densities in 1.12 could be modelling, and how they are established, in practice, consider an experiment where we measure the temperature of metal samples that had been left in a furnace for a long time (such that the temperatures of the metal samples when in the furnace are the same temperature as the environment in the furnace itself).

i) For any given metal sample, let X be the measured temperature of the sample, and Z be the time from when the sample was removed from the furnace to when its temperature was measured.

ii) The aim of the experiment is to devise a means of predicting when a metal sample had been removed from the furnace (Z), based on its temperature (X). This is equivalent to establishing/deriving the posterior $p(Z|X)$.

iii) In the experiment, we remove each metal sample from the furnace for a selected period of time (Z), and then measure its temperature (X). A very large number of metal samples were used in the experiment.

iv) The observation set $\{X_i\}$ contains the measured temperatures of the metal samples. The latent set $\{Z_i\}$ contains the corresponding times from when metal samples were taken out of the furnace to when the experimenter measured their temperatures.

v) As we understand that metal samples will tend to cool quicker when initially removed from the furnace, we measure the temperature of more metal samples that have been very recently removed from the furnace, with fewer samples having more of such a delay.

vi) For the large number of metal samples in the experiment, the times from "removal from the furnace" to "measurement of temperature" ($\{Z_i\}$) were chosen such that their distribution closely approximates the continuous exponential distribution $Exp(\lambda = 1)$. That is: $Z \sim Exp(\lambda = 1), \forall Z \in \{Z_i\}$.

vii) In the experiment, it was found that the measured temperatures of the metal samples, grouped by any given time $Z \in \{Z_i\}$, were found to approximately follow the continuous Gaussian distribution $\mathcal{N}(\mu = Z, \sigma = 1)$. That is: $X \sim \mathcal{N}(\mu = Z, \sigma = 1), \forall X \in \{X_i\}$ (in this context, Z and X are in the same corresponding positions in their respective sets $\{Z_i\}$ and $\{X_i\}$, respectively).

All that remains to calculate the required target density $p(Z|X)$ for the model in 1.12, using Bayes' theorem, is an expression for $p(X)$. When the observations X are taken from a population of an indefinite size, such as for the 'metal samples from the furnace' example above – we can always generate more independent observations by repeating the experiment - $p(X)$ can not be calculated directly. However, $p(X)$ can be calculated from $p(Z)$ and $p(X|Z)$ as follows - derived in appendix section A.1:

$$p(X) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}X^2} \int_0^\infty e^{XZ} e^{-\frac{1}{2}Z^2} e^{-Z} dZ \tag{1.13}$$

Even for this relatively simple model, the computational expense of calculating $p(X)$ is already significantly greater (in relative terms) than for $p(Z)$ or $p(X|Z)$. Given Bayes' theorem (equation 1.9), this computational expense would then also carry over for $p(Z|X)$. Furthermore, as the dimension $m \in \mathbb{N}$ of the latent vector $\mathbf{z}$ increases (where $\mathbf{z} \in \mathbb{R}^m$), a calculation of the evidence $p(\mathbf{x})$ would then require marginalising out each of the latent variables $z_i \in \mathbf{z}$, for $i \in \{1, 2, \ldots, m\}$, from the joint distribution $p(\mathbf{x}, \mathbf{z})$ $(= p(\mathbf{x}|\mathbf{z}) * p(\mathbf{z}))$ – i.e.,

$$\begin{aligned} p(\mathbf{x}) &= \int_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}) d\mathbf{z} \\ &= \int_{z_m} \left(\int_{z_{m-1}} \cdots \left(\int_{z_1} p(\mathbf{x}, \mathbf{z}) dz_1 \right) \ldots dz_{m-1} \right) dz_m \end{aligned} \tag{1.14}$$

Computing the exact Bayesian posterior requires summing or integrating over all latent variables, which can be in the millions or billions for complex models and large-scale applications. Coupled with the often 'difficult to compute' integral of the joint model, exact inference is therefore typically intractable in these models, and approximations are needed. The central idea of **VI** is to approximate the model posterior by a simpler distribution (Zhang et al., 2018).

As the complexity of Bayesian models $p(\mathbf{x}, \mathbf{z})$ increases, and particularly as the dimension m of the latent vector $\mathbf{z}$ increases (where $\mathbf{z} \in \mathbb{R}^m$), the complexity and computational expense of calculating $p(\mathbf{x})$ – and thus, by extension, $p(\mathbf{z}|\mathbf{x})$ – increases in an exponential-like fashion. This is a substantial example of why we often don't have an (practically) accessible expression for $p(\mathbf{z}|\mathbf{x})$. Algorithms such as **VI** and **MCMC** are predominantly used to address this very Bayesian problem, allowing us to obtain a suitable approximation for $p(\mathbf{z}|\mathbf{x})$.

From the 1990s until about 2010, **MCMC** was the prevailing algorithm used to address this issue (Gelfand and Smith, 1990). Over these 2 decades and since, **MCMC** in general has been widely studied, extended, and applied (Blei, Kucukelbir, and McAuliffe, 2017; Robert, Casella, and Casella, 1999). The paper "Sampling Based Approaches to Calculating Marginal Densities" (Gelfand and Smith, 1990) also touches on the difficulties of calculating Bayesian Posterior densities, but employs **MCMC** to solve the problem (Blei, Kucukelbir, and McAuliffe, 2017). Landmark **MCMC** algorithms include the Metropolis–Hastings algorithm (Hastings, 1970; Metropolis et al., 1953) and the Gibbs sampler (Geman and Geman, 1984). Let it be emphasized that **MCMC** and **VI** are different approaches to solving the same problem (Blei, Kucukelbir, and McAuliffe, 2017). **MCMC** has the added benefit of providing asymptotically accurate approximations of the target posterior density, following successive iterations of the algorithm.

Although **VI** does not tend to be as accurate as **MCMC**, **VI** has shown to be significantly faster than **MCMC**, with certain variants of **VI** scaling far more efficiently to massive data (Blei, Kucukelbir, and McAuliffe, 2017; Wainwright, Jordan, et al., 2008). Given the large influx of data and massive data requirements of widespread (Bayesian-based) applications over the last 10 years (or so), the fact that **VI** has shown through recent research to scales so effectively to massive data has highlighted **VI** as a very promising alternative to **MCMC** (Blei, Kucukelbir, and McAuliffe, 2017). Yet, **VI** is lacking the academic rigor and scrutiny that **MCMC** has previously been subject to (Blei, Kucukelbir, and McAuliffe, 2017).

Although there are also numerous other marginal applications to which **VI** is applied, **VI** is currently used predominantly due to the following (each outlined above):

i) Obtaining an approximation of the posterior in the omnipresent 'intractable evidence' Bayesian problem in modern machine learning

ii) The empirically demonstrated effectiveness of **VI** to scale so effectively to massive data and complex models

iii) The intriguing potential of **VI**, over algorithms such as **MCMC**, given its effectiveness in meeting key objectives, demonstrated with relatively little research

It is this outstanding 'intriguing' potential of **VI** and the open-ended research, at the time of writing, that has provided me with the opportunity to pursue the topic and research outlined in this thesis. Proceeding chapters will outline my contribution to the ongoing research into **VI**.

1.5 The Evidence Lower Bound (ELBO)

This section will define, outline, and elaborate on an essential component of most approaches in solving the optimisation problem of equation 1.10 in Bayesian **VI** – The evidence lower bound. The evidence lower bound is commonly abbreviated to **ELBO** (**E**vidence **L**ower **BO**und), a very common acronym in **VI**. The acronym **ELBO** is pronounced in the same way as the word "elbow".

The main outstanding problem that we have with the optimisation equation 1.10 is that we don't have an accessible expression for $p(\mathbf{z}|\mathbf{x})$ – If we did, we would have no need to appeal to algorithms such as **VI** or **MCMC** in the first place. Without $p(\mathbf{z}|\mathbf{x})$, we have no means of calculating $\mathcal{KL}\left(q(\mathbf{z})||p(\mathbf{z}|\mathbf{x})\right)$, and thus no means of 'directly' solving optimisation equation 1.10.

The most common resolution to this problem, that invokes the **ELBO**, stems primarily from the fact that $\mathbf{x}$ is fixed/constant for any given model (as outlined in section 1.3). Combined with common properties of expectations and logarithms, we have:

$$\mathcal{KL}\left(q(\mathbf{z})||p(\mathbf{z}|\mathbf{x})\right) = -\mathbb{E}_{q(\mathbf{z})\in Q}\left[\log\left(\frac{p(\mathbf{x},\mathbf{z})}{q(\mathbf{z})}\right)\right] + \log\left(p(\mathbf{x})\right) \tag{1.15}$$

A full derivation of equation 1.15 is given in appendix section A.2.

The term $\mathbb{E}_{q(\mathbf{z})\in Q}\left[\log\left(\frac{p(\mathbf{x},\mathbf{z})}{q(\mathbf{z})}\right)\right]$ in equation 1.15 is the definition of the aforementioned **ELBO**, with respect to the surrogate posterior $q(\mathbf{z})$. That is:

$$\text{ELBO}\left[q(\mathbf{z})\right] = \mathbb{E}_{q(\mathbf{z})\in Q}\left[\log\left(\frac{p(\mathbf{x},\mathbf{z})}{q(\mathbf{z})}\right)\right] \tag{1.16}$$

Combining equations 1.15 and 1.16, we thus have:

$$\mathcal{KL}\left(q(\mathbf{z})||p(\mathbf{z}|\mathbf{x})\right) = -\text{ELBO}\left[q(\mathbf{z})\right] + \log\left(p(\mathbf{x})\right) \tag{1.17}$$

As the distribution $p(\mathbf{x},\mathbf{z})$ (note that the joint distribution $p(\mathbf{x},\mathbf{z})$ is not necessarily a density, unlike $p(\mathbf{z}|\mathbf{x})$, $p(\mathbf{x}|\mathbf{z})$, $p(\mathbf{x})$, $p(\mathbf{z})$ and $q(\mathbf{z})$) and the density $q(z)$ are known and tractable – recall that $p(\mathbf{x},\mathbf{z}) = p(\mathbf{x}|\mathbf{z}) * p(\mathbf{z})$ is a known tractable expression, given the implied setting and framework outlined in section 1.4 - then we are able to obtain a tractable (and often relatively simple) expression for the **ELBO** using equation 1.16.

One of the most important revelations and discoveries for **VI**, that first brought it into the realm of applicable machine learning, was a recognition of the equivalence of the following optimisations, provided that $\mathbf{x}$ is fixed:

$$q^*(\mathbf{z}) = \underset{q(\mathbf{z})\in Q}{\arg\min}\,\mathcal{KL}\left(q(\mathbf{z})||p(\mathbf{z}|\mathbf{x})\right) \quad \text{and} \quad q^*(\mathbf{z}) = \underset{q(\mathbf{z})\in Q}{\arg\max}\,\text{ELBO}\left[q(\mathbf{z})\right] \tag{1.18}$$

(equation 1.18 refers to the latter of the two equations stated in the line above).

As $\mathbf{x}$ is fixed, then so is $\log\left(p(\mathbf{x})\right)$. Thus, by equation 1.17, maximising $\text{ELBO}\left[q(\mathbf{z})\right]$ is equivalent to minimising $\mathcal{KL}\left(q(\mathbf{z})||p(\mathbf{z}|\mathbf{x})\right)$, and vice versa. That is, maximizing the **ELBO** is equivalent to minimizing the **KL** divergence from q to p (and vice versa) (Zhang et al., 2018).

The **ELBO** – Evidence lower bound – is so named, as its numerical value, for any $q(\mathbf{z}) \in Q$, lower bounds the evidence $p(\mathbf{x})$. Following a rearrangement of equation 1.17, it can be established that $\log\left(p(\mathbf{x})\right) = \text{ELBO}\left[q(\mathbf{z})\right] + \mathcal{KL}\left(q(\mathbf{z})||p(\mathbf{z}|\mathbf{x})\right)$. This result, combined with the fact that "$\mathcal{KL}\left(q(\mathbf{z})||p(\mathbf{z}|\mathbf{x})\right) \geq 0$" (this property was given in section 1.2), means we can infer that $\log\left(p(\mathbf{x})\right) \geq \text{ELBO}\left[q(\mathbf{z})\right]$. The functional

$\text{ELBO}\left[q(\mathbf{z})\right]$ is thus a lower bound of $\log\left(p(\mathbf{x})\right)$ for all $q(\mathbf{z}) \in Q$. As the logarithmic function is a strictly increasing monotonic function, then $\text{ELBO}\left[q(\mathbf{z})\right]$ also lower bounds $p(\mathbf{x})$ – although not in a direct way, as for $\log\left(p(\mathbf{x})\right)$: $p(\mathbf{x}) \geq e^{\text{ELBO}[q(\mathbf{z})]}$.

The definition of the **KL** divergence and its use as a proximity measure in **VI** has allowed us to derive the **ELBO** – a lower bounding of the fixed evidence $p(\mathbf{x})$ – and ultimately derive a measure of proximity between $q(\mathbf{z})$ and $p(\mathbf{z}|\mathbf{x})$ without knowing $p(\mathbf{z}|\mathbf{x})$. Furthermore, the **KL** divergence is well known for providing the best such lower bound (at least, at the time of writing) of the evidence $p(\mathbf{x})$ (Jordan et al., 1999).

As $p(\mathbf{x})$ is a probability density, then $0 \leq p(\mathbf{x}) \leq 1$, $\forall \mathbf{x}$. Therefore, $\log\left(p(\mathbf{x})\right) \leq 0$. It was established above that $\log\left(p(\mathbf{x})\right) \geq \text{ELBO}\left[q(\mathbf{z})\right]$. Therefore, combining these two results, we have:

$$\text{ELBO}\left[q(\mathbf{z})\right] \leq 0,\ \forall q(\mathbf{z}) \in Q \tag{1.19}$$

In one of the original literatures on **VI**, the **ELBO** was derived through Jensen's inequality (Blei, Kucukelbir, and McAuliffe, 2017; Jordan et al., 1999), via the inequality $\log\left(p(\mathbf{x})\right) \geq \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}\left[\log\left(p(\mathbf{x},\mathbf{z})\right)\right] + \mathbb{H}(\mathbf{z})$ $(= \text{ELBO}\left[q(\mathbf{z})\right])$ – a full derivation of which is given in appendix section A.3 – where $\mathbb{H}(\mathbf{z}) = -\mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}\left[\log\left(q(\mathbf{z})\right)\right]$ is the Shannon entropy (Yang, 2017).

As well as arguably being one of the first major revelation of applicable **VI**, this **ELBO** initiative also aided the distinguishing and cementing of the use of the **KL** divergence as the proximity measure in **VI**. **ELBO**-based approaches are by far the most common approaches of solving the **VI** optimisation problem of equation 1.10 (where we're not able to directly calculate $\mathcal{KL}\left(q(\mathbf{z})||p(\mathbf{z}|\mathbf{x})\right)$.

All **VI** variants that will henceforth be considered in this thesis have been developed in such a way that an **ELBO**-based approach can be, and will be, used to solve the underlying **VI** optimisation.

1.6 Mean-Field Variational Inference

This section outlines what a mean-field is, and how it is applicable to **VI**, ultimately constituting the generally accepted definition of the 'mean-field variational inference' class of **VI** variants. '**M**ean-**F**ield **V**ariational **I**nference' is commonly abbreviated to **MFVI**.

Just as for functionals (see section 1.1), the precise definition of a mean-field varies slightly depending on the context to which it is applied. In the context of (Bayesian) **VI**, a mean-field predominantly refers to an assumption applied to the surrogate posterior $q(\mathbf{z})$, when the dimension m of the latent vector $\mathbf{z}$ is greater than or equal to 2, for $\mathbf{z} \in \mathbb{R}^m$ and $m \in N$. The 'mean-field approximation' applied to **VI** generally infers that each of the latent variables are mutually independent and each governed by a distinct factor in the variational density Q (Blei, Kucukelbir, and McAuliffe, 2017). That is, if $\mathbf{z} = (z_1 \quad z_2 \quad \ldots \quad z_m)^T$ - i.e., $z_i \in \mathbb{R}$ for $i \in \{1, 2, \ldots, m\}$ - then:

$$q(\mathbf{z}) = \prod_{i=1}^{m} q_i(z_i) \tag{1.20}$$

The variational family Q, where $q_i \in Q$ for each $i \in \{1, 2, \ldots, m\}$ in equation 1.20, under this mean-field assumption generally only considers one dimensional probability density families. The surrogate posterior q, of $\mathbf{z}$, is then defined as a product of m densities from Q – one corresponding to each of the scalar component of $\mathbf{z}$ – as outlined in equation 1.20. As each of the scalar components of $\mathbf{z}$ are assumed to be mutually independent, then the product of probability densities q_i in equation 1.20 is also a probability density – that is, $q(\mathbf{z})$ is necessarily a probability density.

The 'mean-field approximation' for **VI** outlined above is generally what is referred to by **MFVI** and is known as a "full mean-field" or a "fully-factorised family" (Blei, Kucukelbir, and McAuliffe, 2017). There are variations to this full mean-field approach, such as structured **MFVI** (Saul and Jordan, 1995; Barber and Wiegerinck, 1998) – that generally includes factors that depend on more than one of the marginal components, thus introducing an element of dependence between the components, as a means of improving predictive accuracy - and mixture-based **VI** (Bishop et al., 1997) – that generally introduces additional latent components to $\mathbf{z}$, that often depend on combinations of existing latent components in $\mathbf{z}$, as a means of, again, introducing an element of dependence. However, variations such as structured **MFVI** and mixture-based **VI** are difficult to implement in practice as their underlying structures have to be tailored to individual models and model types. That is, these variations don't generalise very well, unlike the full mean-field approach, and are thus used less commonly in practice. **MFVI** and its variants, as outlined above, have their origins in the mean-field theory of physics (Zhang et al., 2018; Opper and Saad, 2001).

These mean-field approaches in **VI** are generally employed in order to make considerable savings on the computational expense of solving the underlying **VI** optimisation problem of equation 1.10 , whilst still allowing the **VI** method to make reasonable and relatively accurate approximations in many practical applications. This computational expense saving benefit is particularly important for **VI**, as this is one of the most important features that validates **VI** as a viable machine learning method – see section 1.4 (recall that **VI** is already generally far less computationally expensive than **MCMC**, its main rival).

As an example of the significance of the computational expense saving achieved by the mean-field simplification/approximation outlined above, consider the case where the surrogate posterior $q(\mathbf{z})$ takes the form of a multivariate Gaussian (assuming $\mathbf{z}$ is not one-dimensional – i.e., not a scalar). Any multivariate Gaussian density can be uniquely defined and identified by its mean, the variances of its marginal components, and the covariances between each non-ordered pair of its marginal components. Let's say that there are m marginal components (i.e., $\mathbf{z} \in \mathbb{R}^m$) – We would then have to consider the following: 1 mean, m variances, and $\frac{1}{2}(m^2 - m)$ covariances. As the number of marginal components m increases, so would the proportion of the number covariances over the number of means and variances, due to the dominant $\frac{1}{2}m^2$ term. In the practical realm, where the number of marginal components m can be in the millions or even billions (and is still increasing, given ongoing demand in industry and academia), the number of covariances to consider would overwhelmingly dominate the total number of statistics (mean, variances & covariances) to consider. The mean-field approximation means that we can effectively disregard consideration of all covariances, given the mutual independence assumption (i.e., the covariances would all be equal to zero), so that we are left with just the mean and m variances, which do not increase in an exponential fashion with

m, unlike the number of covariances (due to the $\frac{1}{2}m^2$ term). The computational expense saving in practical cases such as this can be of several orders of magnitude.

The general limitations of **MFVI** tend to be found consistently across a broad spectrum of models, as they result from characteristic features of **MFVI**. For example, mean-field approximated surrogate posteriors in **VI** tend to have associated variances that underestimate the corresponding variances of the exact target posterior (Blei, Kucukelbir, and McAuliffe, 2017). Consider figure 1.3, adapted from figure 1 in the paper "Variational Inference: A Review for Statisticians" (Blei, Kucukelbir, and McAuliffe, 2017), that provides a visual illustration of this limitation – In this case: $\mathbf{z} \in \mathbb{R}^2$, where $\mathbf{z} = (z_1 \quad z_2)^T$ for $z_1, z_2 \in \mathbb{R}$; the surrogate $q(\mathbf{z})$ (uncorrelated) and target $p(\mathbf{z}|\mathbf{x})$ (correlated) posteriors are two-dimensional Gaussians; and the coloured regions show where significant density has been placed. It is evident by inspection of figure 1.3 that the variances of both z_1 and z_2 in the exact posterior have been underestimated in the mean-field approximation. These variance underestimations ultimately stem from the way in which the **KL** divergence is calculated in the optimisation of **VI**, as summarised in equation 1.10. Due to the non-symmetric nature of the **KL** divergence and the order of the arguments $q(\mathbf{z})$ and $p(\mathbf{z}|\mathbf{x})$ in this divergence measure, the optimisation in equation 1.10 penalizes placing mass in $q(\mathbf{z})$ on areas where $p(\mathbf{z}|\mathbf{x})$ has little mass but penalizes less the reverse (Blei, Kucukelbir, and McAuliffe, 2017). Variations of **MFVI** such as 'Structured **MFVI**' (Saul and Jordan, 1995; Barber and Wiegerinck, 1998) and 'Mixture-Based **VI**' (Bishop et al., 1997) somewhat help to address these sorts of limitations but have not generally proven to be viable solutions in the practical realm, due to the lack of generalisation of these variants. There are models to which these variants can't even be applied.

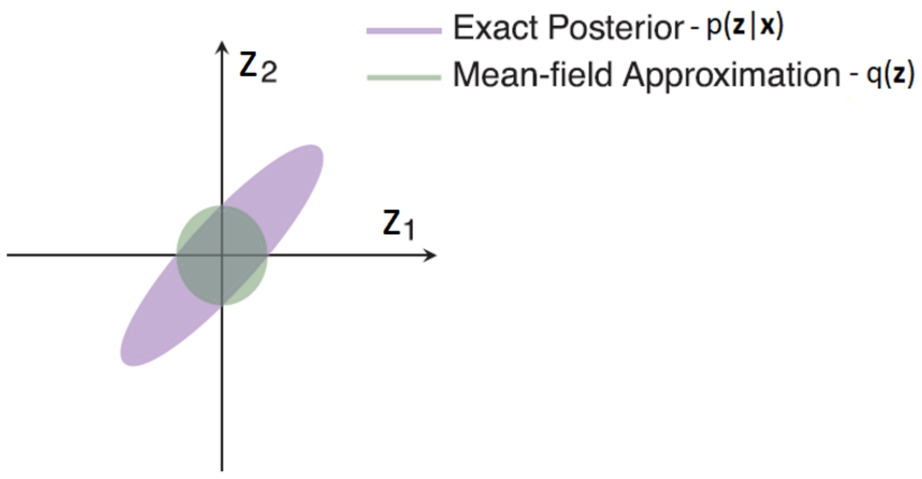


FIGURE 1.3: Mean-field approximation of a strongly correlated two-dimensional Gaussian posterior, adapted from the paper "Variational Inference: A Review for Statisticians" (Blei, Kucukelbir, and McAuliffe, 2017)

Although having been empirically shown to be a relatively effective practical assumption, ongoing concerns about the strong independence assumption of **MFVI** have given rise to substantial and ongoing research in an attempt to address possible inaccuracies. The popularity and common use of this assumption in the practical realm has highlighted such research as a likely future necessity for **VI** in general. It is paramount that the possible inaccuracies of concern are addressed whilst retaining at least most of the computational savings that the mean-field approach generally brings, as well as retaining the generalisation of the full mean-field **VI** (**MFVI**) over the likes of structured and mixture-based **MFVI** variants.

The underlying research objectives in this thesis aim to propose, summarise and support one such approach of addressing these possible inaccuracies, as will be summarised in subsequent chapters.

1.7 Coordinate Ascent Variational Inference

The **MFVI** class of **VI** variants outlined in the previous section (section 1.6) lays the foundation of all the main **VI** approaches used to obtain the given surrogate posterior in the Bayesian **VI** framework outlined in section 1.3. The simplest such approach is 'a mean-field approximation augmented with a coordinate-ascent optimization' (Blei, Kucukelbir, and McAuliffe, 2017). A 'coordinate-ascent optimization' approach to **MFVI** (as summarised in section 1.6) is commonly referred to as **CAVI** - Coordinate Ascent Variational Inference. This section outlines how coordinate-ascent optimization is used to solve an **MFVI** based method and ultimately obtain the desired surrogate posterior $q(\mathbf{z})$.

Coordinate ascent optimisation is a common approach used in solving optimisation problems in mathematics and statistics generally. Within **CAVI**, the required **VI** optimisation is carried out over each of the latent components of $\mathbf{z}$ in turn (Blei, Kucukelbir, and McAuliffe, 2017; Bishop and Nasrabadi, 2006) – That is, the latent components of $\mathbf{z}$ are the coordinates in coordinate ascent optimisation for **CAVI**. We hold all but one of the coordinates fixed, and then optimise only over this remaining variable coordinate, repeating this process in turn for each of the coordinates. This process, as a whole, is then iteratively repeated until the variational objective (i.e., $\text{ELBO}[q]$, for the **VI** variant classes that we have adopted) has converged – that is, in a practical sense, until $|\text{ELBO}_{i+1}[q] - \text{ELBO}_i[q]| < \epsilon$, where i corresponds to the iteration ($i = 1$: 1^{st} iteration, $i = 2$: 2^{nd} iteration, etc) of the **VI** method; $\text{ELBO}_i[q]$ corresponds to the numerical value of the **ELBO** at iteration i; and ϵ corresponds to a small positive numerical value, the magnitude of which can be considered small enough for convergence to have practically taken place.

As a means of introducing a general derivation of the underlying mathematics and logic behind the **CAVI** algorithm, let a 3-dimensional case first be considered for the latent vector variable $\mathbf{z}$, where $q(\mathbf{z}) = q_1(z_1)q_2(z_2)q_3(z_3)$. Corresponding equations of the factors of the surrogate posterior $q(\mathbf{z})$ - $q_1(z_1)$, $q_2(z_2)$ & $q_3(z_3)$ – under a 'coordinate-ascent optimization' approach are then given by equations 1.21, 1.22 and 1.23, where a full derivation of these equations is given in appendix section A.4.

$$q_1(z_1) = \alpha_1 e^{\mathbb{E}_{-1}[\log p(\mathbf{x},\mathbf{z})]} \tag{1.21}$$

$$q_2(z_2) = \alpha_2 e^{\mathbb{E}_{-2}[\log p(\mathbf{x},\mathbf{z})]} \tag{1.22}$$

$$q_1(z_3) = \alpha_3 e^{\mathbb{E}_{-3}[\log p(\mathbf{x},\mathbf{z})]} \tag{1.23}$$

Here, for $i \in \{1, 2, 3\}$: α_i are normalising constants (in general, α_1, α_2 and α_3 are not equal); and $\mathbb{E}_{-i}$ is an expectation with respect to all but the i^{th} scalar component of $\mathbf{z}$, distributed as q, excluding the i^{th} factor - i.e., $\mathbb{E}_{-1} = \mathbb{E}_{(z_2 \quad z_3)^T \sim q_2(z_2)q_3(z_3)}$, $\mathbb{E}_{-2} = \mathbb{E}_{(z_1 \quad z_3)^T \sim q_1(z_1)q_3(z_3)}$ & $\mathbb{E}_{-3} = \mathbb{E}_{(z_1 \quad z_2)^T \sim q_1(z_1)q_2(z_2)}$ in the example above.

The form of the mean-field factor expressions outlined above generalises for any dimension m of the latent vector $\mathbf{z}$, where $m \in \mathbb{N}$ and $m \geq 2$, given a full mean-field form for the surrogate posterior $q(\mathbf{z})$: This is evident via an extrapolation of the logical process and derivation set out in appendix section A.4. That is, when $q(\mathbf{z})$ takes the form of equation 1.20, we have:

$$q_j(z_j) = \alpha_j e^{\mathbb{E}_{-j}[\log p(\mathbf{x},\mathbf{z})]} \tag{1.24}$$

for each of the mean-field factors j, where: $1 \leq j \leq m$ for $j \in \mathbb{N}$; α_j is a normalising constant for $q_j(z_j)$; and $\mathbb{E}_{-j}$ is the expectation with respect to $\mathbf{z}$, excluding the component z_j, distributed as $\prod_{i=1,\, i\neq j}^{m} q_i(z_i) = \frac{\prod_{i=1}^{m} q_i(z_i)}{q_j(z_j)}$.

A complete and concise algorithmic process of the **CAVI** algorithm is given as follows (Blei, Kucukelbir, and McAuliffe, 2017; Bishop and Nasrabadi, 2006):

```
Coordinate Ascent Variational Inference (**CAVI**) Algorithm

**Input:** A model $p(\mathbf{x}, \mathbf{z})$ and a dataset $\mathbf{x}$.

**Output:** The fitted & optimised Variational Density $q(\mathbf{z})$.

**Initialisation:** Randomly initialise the variational parameters of each of the variational factors $q_j(z_j)$. Choose an $\epsilon$ that would be considered small enough for convergence to have practically taken place.

**Start:** $i = 1$

**For** $j \in \{1, \dots, m\}$ (where $\mathbf{z} \in \mathbb{R}^m$):
    Set $q_j(z_j) = \alpha_j e^{\mathbb{E}_{-j}[\log p(\mathbf{x},\mathbf{z})]}$, where $\alpha_j$ is a constant chosen such that $q_j(z_j)$ integrates
    to one over its domain.
    Each of the other $m - 1$ latent variable coordinates not currently being considered
    are treated as constants (held fixed), and are effectively absorbed into the
    normalising constant $\alpha_j$.
**Compute:** ELBO$_i$ $[q(\mathbf{z})]$ using equation 1.16
**If:** $|\text{ELBO}_{i+1}[q] - \text{ELBO}_i[q]| \geq \epsilon$ or $i = 1$:
    **Set:** $i = i + 1$
    **Jump:** To the above **For** loop, and repeat the execution

**End**
```

The sequence of calculated **ELBO** values in the algorithm above, as the iteration i increases, must be convergent; and the **ELBO** function itself must also have at least one (local) maxima. Otherwise, we will not have a viable means of mathematically optimising the **ELBO**, as required for the adopted **VI** class of methods (see equation 1.18).

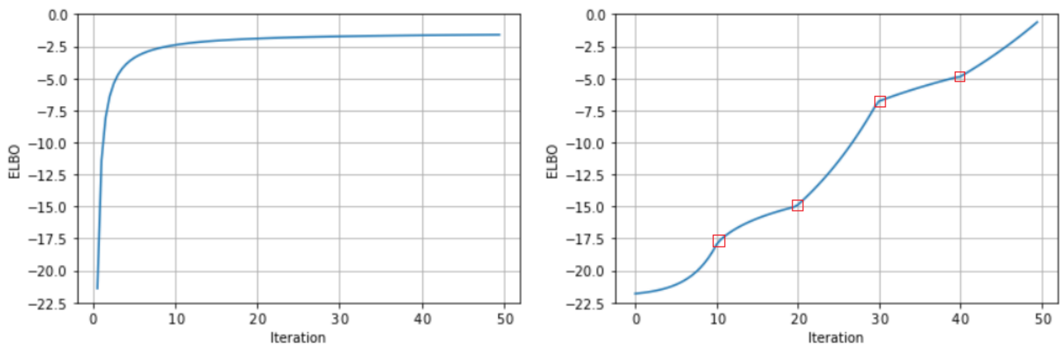


FIGURE 1.4: The **ELBO** value vs. the number of iterations i in the **CAVI** algorithm - two different scenarios. The red squares reveal abrupt stages in the optimisation process, in the context of 'more real world datasets', at which the it becomes significantly more or less efficient.

Depending on the model and dataset that the **CAVI** algorithm (above) is being applied to, a graph of the **ELBO** values plotted against the number of iterations i will take, in a generic sense, one of the two forms illustrated in figure 1.4. Recall that, as outlined in equation 1.19, the value of the **ELBO** for any given argument is less than or equal to zero.

The trajectory of the **ELBO** value as the number of iterations i increases will generally follow the left-hand plot in figure 1.4 if the dataset has "cleaner and more distinct patterns", and generally follow the right-hand plot in figure 1.4 if the dataset has "more obscure and/or randomised underlying patterns" (as tends to be the case for real-world datasets). The choice of model $p(\mathbf{x}, \mathbf{z})$ for a given dataset $\mathbf{x}$ can also influence which of the two general trajectories in figure 1.4 the **ELBO** values will follow in the **CAVI** algorithm: Models $p(\mathbf{x}, \mathbf{z})$ that more effectively and more amenably reveal the underlying patterns in the dataset $\mathbf{x}$ will tend to result in **ELBO** values that generally follow the left-hand plot in figure 1.4; otherwise, the **ELBO** will tend to follow a trajectory that takes a form more like the right-hand plot in figure 1.4.

The red squares in the right-hand plot of figure 1.4 reveal phases of the **ELBO** maximization process where the variational approximation suddenly changes its shape (Blei, Kucukelbir, and McAuliffe, 2017), resulting in the staggered plot. The occurrence of these sudden variational approximation shape changes, when a plot of the **ELBO** value against the number of iterations i is considered, is colloquially referred to as "the **ELBO** developing elbows" (Blei, Kucukelbir, and McAuliffe, 2017). These "elbows of the **ELBO**" are not generally ascribed a formal definition or a set of axioms, but are rather "informally & manually identified" from plots of the type depicted in figure 1.4.

The **CAVI** algorithm is generally seen as a "message passing" algorithm, iteratively updating each random coordinate variable's variational parameters based on the variational parameters of the variables among the fixed coordinates (in its Markov blanket) (Blei, Kucukelbir, and McAuliffe, 2017; Winn, Bishop, and Jaakkola, 2005). This perspective of **CAVI** as a variational "message passing" algorithm connects **VI** to the classical theories of graphical models and probabilistic inference (Blei, Kucukelbir, and McAuliffe, 2017; Pearl, 1988; Lauritzen and Spiegelhalter, 1988). Variants of the **CAVI** algorithm have been extended to non-conjugate models (Knowles and Minka, 2011) and generalised via factor graphs (Blei, Kucukelbir, and McAuliffe, 2017; Minka et al., 2005).

The **CAVI** algorithm is closely related to Gibbs sampling (Geman and Geman, 1984; Gelfand and Smith, 1990) - a sampling method that is generally considered the classical workhorse of approximate inference (Blei, Kucukelbir, and McAuliffe, 2017) - in that each of these algorithms use the same fundamental approach to obtain their required approximations: Across a coordinate space of variables to optimise over, all but one of the coordinates are held fixed while optimisation is performed over the remaining variable coordinate, with this process then repeated for all of the coordinates, often looping through this process as a whole, until the required approximations for the algorithms are obtained.

The **CAVI** algorithm is the simplest, and a relatively effective, algorithm for solving the **VI** optimisation problem of equation 1.18 (recall that an execution of the **VI** method can be achieved by optimising equation 1.18). In fact, the **CAVI** algorithm has been empirically shown to be far faster - even several orders of magnitude faster - than many of its alternatives, such as the Hamiltonian Monte Carlo-based sampler (Blei, Kucukelbir, and McAuliffe, 2017; Hoffman, Gelman, et al., 2014).

Despite this significant improvement in computational speed over algorithms such as the Hamiltonian Monte Carlo-based sampler, the **CAVI** algorithm is still considered too slow and laborious to be used in practice - that is, for the massive data requirements to which **VI** is typically applied. For example: For each iteration of the CAVI algorithm, the updates are often made across the whole dataset $\mathbf{x}$, which can be very large; furthermore, the optimal step direction for maximising the **ELBO** may only be predominantly in the direction of a handful of the coordinates, yet we always iterate through all of the coordinates.

Section 1.8 will provide a full derivation and implementation of this simplest approach to **VI**, to fully demonstrate and illustrate **VI** in action. Then section 1.9 will introduce a faster alternative to coordinate-ascent: Stochastic optimisation, the approach that is predominantly used in practice.

1.8 Coordinate Ascent Variational Inference for a Bayesian Mixture of Gaussians

The prior sections leading up to this section (section 1.8) have now outlined enough of the concepts of the simplest and main approach to **VI**, for a full implementation of **VI** to be demonstrated. This is what will be covered in this section. The dataset $\mathbf{x}$ applicable to the full implementation in question will be a contrived dataset based on a common "Bayesian mixture of Gaussians" model, or a "Gaussian mixture model", commonly abbreviated to **GMM** (Blei, Kucukelbir, and McAuliffe, 2017).

Each of the following optional concepts applicable to **VI**, that have since been explicitly adopted in general, are necessities for the derivations that will follow in this section:

i) The **KL** divergence is to be used as the proximity measure for **VI**.

ii) **VI** is to be cast in the Bayesian setting outlined in section 1.3.

iii) The surrogate posterior $q(\mathbf{z})$ of the **VI** method is to have the mean-field approximation (see section 1.6) applied to it.

iv) A fitting and optimisation of the surrogate posterior $q(\mathbf{z})$ is to be achieved via a coordinate-ascent optimisation approach.

The **GMM** that will be used as a demonstration of a full **CAVI**-based implementation of **VI** in this section will be based on the **GMM** from the paper "Variational Inference: A Review for Statisticians" (Blei, Kucukelbir, and McAuliffe, 2017).

The **GMM** in question is encompassed and summarised by the following distributions:

$$\mu_k \sim \mathcal{N}(0, \sigma^2), \quad k = 1, \dots, K \tag{1.25}$$

$$c_i \sim Cat\left(\frac{1}{K}, \dots, \frac{1}{K}\right), \quad i = 1, \dots, n \tag{1.26}$$

$$x_i|_{\boldsymbol{\mu}, c_i} \sim \mathcal{N}(c_i^T \boldsymbol{\mu}, 1), \quad i = 1, \dots, n \tag{1.27}$$

i) The variable σ in distribution 1.25 is a hyperparameter in this **GMM**.

ii) There are K Gaussian clusters in this model and a total of n data points - that is, $K, n \in \mathbb{N}$ and $\mathbf{x} \in \mathbb{R}^n$.

iii) Each of the K Gaussian clusters have an associated cluster centre μ_k (for $k = 1, \dots, K$).

iv) The distribution $Cat\left(\frac{1}{K}, \dots, \frac{1}{K}\right)$ in distribution 1.26 is a categorical distribution with K categories, each of which has a probability of occurrence of $\frac{1}{K}$.

v) The variables x_i, for $i = 1, \dots, n$, are each numerical values in $\mathbb{R}$, corresponding to the i^{th} data point in the dataset $\mathbf{x}$.

vi) Each c_i (for $i = 1, \dots, n$) corresponds to the Gaussian cluster assignment of point x_i, and is encoded as a one-hot vector of length K. That is, each c_i is equal to one of $(1 \quad 0 \dots 0)^T$, $(0 \quad 1 \dots 0)^T, \dots (0 \quad 0 \dots 1)^T$. The 1 in the one-hot vector is in the position of the category/cluster that has been selected as a result of the sampling in distribution 1.26.

vii) The vector $\boldsymbol{\mu}$ is an ordered vector, of length K, containing each of the cluster centres μ_k as its elements. That is: $\boldsymbol{\mu} = (\mu_1 \quad \mu_2 \dots \mu_K)^T$.

To generate a contrived dataset $\mathbf{x}$ from this **GMM**, of size n, we simply carry out the following procedure:

1. Set positive integer values for K and i (which can be chosen arbitrarily if this model is not applied in any specific setting or framework). In general, it is assumed that $i > K$ (often $i >> K$).

2. Set a value for the hyperparameter σ in distribution 1.25.

3. Independently sample K values from the distribution $\mathcal{N}(0, \sigma^2)$, each corresponding to μ_k for $k = 1, \dots, K$ - distribution 1.25.

4. Independently sample n values from the distribution $Cat\left(\frac{1}{K}, \ldots, \frac{1}{K}\right)$, each corresponding to c_i for $i = 1, \ldots, n$ - distribution 1.26.
5. For each value of i: Calculate the vector product $c_i^T \boldsymbol{\mu}$ - each such product will be equal to one of the previously sampled μ_k values; then sample each x_i for $\mathbf{x}$ from the corresponding distributions $\mathcal{N}(c_i^T \boldsymbol{\mu}, 1)$ - distribution 1.27.

Note that this is similar to regular **GMM**, but in regular **GMM** we do not have a distribution for the cluster centres, as is given by distribution 1.25 (Roberts et al., 1998).

When considering this **GMM** in the context of **VI**, the latent vector is given by $\mathbf{z} = \{\boldsymbol{\mu}, \mathbf{c}\} = \{\mu_1, \mu_2, \ldots, \mu_K, c_1, c_2, \ldots, c_n\}$ and the observations $\mathbf{x}$ are generated via the procedure outlined above. After the observations $\mathbf{x}$ have been generated, we disregard all of the **GMM** settings that resulted in the generation of this dataset, except for the number of Gaussian clusters K.

The objective of our **VI** method, given the **GMM** and generated observations $\mathbf{x}$, is to approximate the posterior cluster assignments $p(c_i|x_i)$ and the posterior mixture components $p(\boldsymbol{\mu}|\mathbf{x})$. These two posterior types, which collectively represent the general posterior $p(\mathbf{z}|\mathbf{x})$ in Bayesian statistics within this framework (given that $\mathbf{z} = \{\boldsymbol{\mu}, \mathbf{c}\}$), can be interpreted as follows:

i) $p(c_i|x_i)$ - The Gaussian cluster c_i that the data point x_i belongs to.

ii) $p(\boldsymbol{\mu}|\mathbf{x})$ - The means of the K Gaussian clusters (the cluster centres), given a presented dataset $\mathbf{x}$.

As the **KL** divergence is to be used as the proximity measure for **VI**, that we wish to minimise, then we are ultimately seeking to maximise the **ELBO** (see section 1.5). Given equation 1.16, we require expressions for the model $p(\mathbf{x}, \mathbf{z})$ and the surrogate posterior $q(\mathbf{z})$ before an expression for the **ELBO** can be obtained.

For this **GMM**, expressions for the Bayesian joint model and surrogate posterior are given by equations 1.28 and 1.29, respectively, derivations for which are given in appendix section A.5.

$$p(\boldsymbol{\mu}, \mathbf{c}, \mathbf{x}) = p(\boldsymbol{\mu}) \prod_{i=1}^{n} p(c_i) p(x_i|\boldsymbol{\mu}, c_i) = \prod_{k=1}^{K} p(\mu_k) \prod_{i=1}^{n} p(c_i) p(x_i|\boldsymbol{\mu}, c_i) \tag{1.28}$$

$$q(\boldsymbol{\mu}, \mathbf{c}) = \prod_{k=1}^{K} q_{\mu,k}(\mu_k) \prod_{i=1}^{n} q_{\mathbf{c},i}(c_i) \tag{1.29}$$

Given the **CAVI** algorithm outlined in section 1.7, we require an expression for the log of the joint model - $\log\left(p(\boldsymbol{\mu}, \mathbf{c}, \mathbf{x})\right)$. Such an expression is given in equation 1.30:

$$\begin{aligned}
\log\left(p(\boldsymbol{\mu},\mathbf{c},\mathbf{x})\right) &= \log\left(\prod_{k=1}^{K} p(\mu_k)\prod_{i=1}^{n} p(c_i)p(x_i|\boldsymbol{\mu},c_i)\right) \quad \text{by equation 1.28} \\
&= \log\left(\prod_{k=1}^{K} p(\mu_k)\right) + \log\left(\prod_{i=1}^{n} p(c_i)p(x_i|\boldsymbol{\mu},c_i)\right) \\
&= \sum_{k=1}^{K}\left(\log p(\mu_k)\right) + \sum_{i=1}^{n}\left(\log p(c_i)p(x_i|\boldsymbol{\mu},c_i)\right) \\
&= \sum_{k=1}^{K}\left(\log p(\mu_k)\right) + \sum_{i=1}^{n}\left(\log p(c_i)\right) + \sum_{j=1}^{n}\left(\log p(x_j|\boldsymbol{\mu},c_j)\right)
\end{aligned} \tag{1.30}$$

Now let the form of each of the one-dimensional surrogate posterior factor types $q_{\boldsymbol{\mu},k}(\mu_k)$ and $q_{\mathbf{c},i}(c_i)$ be considered, starting with $q_{\mathbf{c},i}(c_i)$.

As each of c_i, for $i = 1,\ldots,n$, can only be one of the K discrete one-hot vectors $(1 \quad 0\ldots0)^T, (0 \quad 1\ldots0)^T, \ldots (0 \quad 0\ldots1)^T$, then let:

$$q_{\mathbf{c},i}(c_i) = Cat(c_i; \phi_{i,1}, \phi_{i,2}, \ldots, \phi_{i,K}) \tag{1.31}$$

where $Cat(\ldots)$ in equation 1.31 is a categorical distribution (a categorical distribution is chosen, as c_i can only be equal to members from a discrete set), and the variables $\phi_{i,k}$, for $k = 1,\ldots,K$, correspond to the probabilities, according to the surrogate posterior $q_{\mathbf{c},i}(c_i)$, that the latent variable c_i is equal to the one-hot vector corresponding to the k^{th} position (the 1 is in the k^{th} position).

Considering a **CAVI** solution of the surrogate posterior factor type $q_{\mathbf{c},i}(c_i)$, using equation 1.24, we have:

$$\phi_{i,k} = Softmax\left(x_i\mathbb{E}_{-i}\left[\mu_k\right] - \frac{1}{2}\mathbb{E}_{-i}\left[\mu_k^2\right]\right) = \frac{\exp\left(x_i\mathbb{E}_{-i}\left[\mu_k\right] - \frac{1}{2}\mathbb{E}_{-i}\left[\mu_k^2\right]\right)}{\sum_{r=1}^{K}\exp\left(x_i\mathbb{E}_{-i}\left[\mu_r\right] - \frac{1}{2}\mathbb{E}_{-i}\left[\mu_r^2\right]\right)} \tag{1.32}$$

A full derivation of equation 1.32 is given in appendix section A.6. Consider the following important points, settings and assumptions that are relevant to the **CAVI** solution of $q_{\mathbf{c},i}(c_i)$, as summarised by equation 1.32.

i) The expectation $\mathbb{E}_{-i}$ is with respect to all of the latent variables in the **GMM** except i - that is, $\{\mu_1, \mu_2, \ldots, \mu_K, c_1, c_2, \ldots, c_{i-1}, c_{i+1}, \ldots, c_n\}$ - distributed as $\frac{q(\boldsymbol{\mu},\mathbf{c})}{q_{\mathbf{c},i}(c_i)}$.

ii) The softmax function is fully defined in appendix section A.6 (Bridle, 1989), although an intuitive sense of its definition is still provided in equation 1.32.

iii) The variable indices i and k in equation 1.32 take specific values, chosen from the sets $\{1, 2, \ldots, n\}$ and $\{1, 2, \ldots, K\}$, respectively.

iv) One complete update of the surrogate posterior $q_{\mathbf{c},i}(c_i)$ in the **CAVI** algorithm, for a given value of i, is achieved by setting each of its variational parameters - $\phi_{i,k}$ for each $k \in \{1, 2, \ldots, K\}$ - equal to the corresponding results of equation 1.32.

v) As an initialisation for the **CAVI** algorithm in section 1.7, all the ϕ values ($\phi_{i,k}$, for each $i \in \{1, 2, \ldots, n\}$ and each $k \in \{1, 2, \ldots, K\}$) must equal $\frac{1}{K}$.

Now let the form of the one-dimensional surrogate posterior factor type $q_{\mu,k}(\mu_k)$ be considered.

Given the normally sampled fashion by which the μ_k values, for $k = 1, \ldots, K$, and the x_i values, for $i = 1, \ldots, n$, are generated - as outlined in distributions 1.25 and 1.27, respectively - it can thus be deduced that an appropriate maximum likelihood model of the μ_k values, given an observed dataset $\mathbf{x}$, is a Gaussian distribution. That is:

$$q_{\mu,k}(\mu_k) = \mathcal{N}(\mu_k; m_k, s_k^2) \tag{1.33}$$

where m_k and s_k^2 are the mean and variance, respectively, of the Gaussian model for $q_{\mu,k}(\mu_k)$.

Considering a **CAVI** solution of the surrogate posterior factor type $q_{\mu,k}(\mu_k)$, using equation 1.24, we have:

$$m_k = \frac{\sum_{i=1}^{n} \phi_{i,k} x_i}{\frac{1}{\sigma^2} + \sum_{i=1}^{n} \phi_{i,k}} \quad \text{and} \quad s_k = \left(\frac{1}{\sigma^2} + \sum_{i=1}^{n} \phi_{i,k} \right)^{-1/2} \tag{1.34}$$

A full derivation of equations 1.34 is given in appendix section A.7. Consider the following important points, settings and assumptions that are relevant to the **CAVI** solution of $q_{\mu,k}(\mu_k)$, as summarised by equations 1.34.

i) The variables m_k and s_k are the mean and standard deviation of the univariate Gaussian density for $q_{\mu,k}(\mu_k)$ (see equation 1.33). Each of the other variables $\phi_{i,k}$, x_i and σ are similarly defined above.

ii) The variable indices k in equations 1.34 will take a single specific value, as appropriate, from the set $\{1, 2, \ldots, K\}$.

iii) One complete update of the surrogate posterior $q_{\mu,k}(\mu_k)$ in the **CAVI** algorithm, for a given value of k, is achieved by setting each of its variational parameters - m_k and s_k - equal to the corresponding results of equations 1.34.

iv) As an initialisation for the **CAVI** algorithm in section 1.7, the values m_k and s_k in the **GMM**, for $k \in \{1, 2, \ldots, K\}$, can technically be set to any values from the sets $\mathbb{R}$ and $\mathbb{R}^+$, respectively (any values for m_k and s_k in their respective domains). However, for this **GMM**, given the knowledge that we have at our disposal, particularly from distribution 1.25, distribution 1.27 and equation 1.33: The most effective initialisation for the m_k values would be 0; and the most effective initialisation for the s_k values would be some value between 1 and σ (effectiveness refers to the most efficient convergence in the **CAVI** algorithm).

In appendix section A.7, it was deduced that $\mathbb{E}_{-t}[c_{jt}] = \phi_{j,t}$, where t is a particular but arbitrary value of k, and c_{jt} denotes the value of the t^{th} element in the one-hot vector c_j (which will be either a 0 or a 1): Considering all of the latent variables of the **GMM** excluding t, we would expect the t^{th} element of the one-hot vector c_j to take, as an average, the probability of setting that element as the 1 in the categorical distribution for $q_{\mathbf{c},j}(c_j)$ (as we don't have the true posterior, see equation 1.31) - i.e., $\phi_{j,t}$.

Likewise, and considering the more general case (not just for $k = t$), it can be deduced that $\mathbb{E}_{-i}[\mu_k] = m_k$ and $\mathbb{E}_{-i}[\mu_k^2] = s_k^2 + m_k^2$, where $i \in \{1,2,\dots,n\}$, $k \in \{1,2,\dots,K\}$ and $\mathbb{E}_{-i}$ is defined via the same convention outlined previously. A full derivation of these two equations is provided in appendix section A.8. Substituting these results into equation 1.32, we have:

$$\phi_{i,k} = Softmax\left(x_i m_k - \frac{1}{2}\left(s_k^2 + m_k^2\right)\right) = \frac{\exp\left(x_i m_k - \frac{1}{2}\left(s_k^2 + m_k^2\right)\right)}{\sum_{r=1}^{K}\exp\left(x_i m_r - \frac{1}{2}\left(s_r^2 + m_r^2\right)\right)} \tag{1.35}$$

Appendix section A.9 provides a derivation of the **ELBO** for this **GMM**, as summarised in equations A.41 and A.42, based on information that has already been established and derived above (including in previous appendix sections).

The specific derivations applicable for the **GMM**, in the context of the generic **CAVI** algorithm provided in section 1.7, have now been outlined. The following algorithm is the **CAVI** algorithm from 1.7 with these specific derivations substituted as appropriate.

CAVI Algorithm for the **GMM**

Input: The model $p(\boldsymbol{\mu}, \mathbf{c}, \mathbf{x}) = \prod_{k=1}^{K} p(\mu_k) \prod_{i=1}^{n} p(c_i) p(x_i|\boldsymbol{\mu}, c_i)$ (equation 1.28); values for the hyperparameters σ & K; and a dataset $\mathbf{x}$ that has been generated in an unknown way from the **GMM** distributions 1.25, 1.26 and 1.27.

Output: The fitted & optimised Variational Density $q(\boldsymbol{\mu}, \mathbf{c}) = \prod_{k=1}^{K} q_{\boldsymbol{\mu},k}(\mu_k) \prod_{i=1}^{n} q_{\mathbf{c},i}(c_i)$ (see equation 1.29), parameterised by the variational parameters m_k, s_k and $\phi_{i,k}$, for $i \in \{1,2,\dots,n\}$ and $k \in \{1,2,\dots,K\}$ (see equations 1.31 and 1.33).

Initialisation: Set each of the variational parameters, $\forall i \in \{1,2,\dots,n\}$ & $\forall k \in \{1,2,\dots,K\}$: $\phi_{i,k}$ equal to $\frac{1}{K}$; m_k equal to 0; and s_k equal to $\frac{\sigma+1}{2}$ (avergae of σ and 1). Choose an ϵ that would be considered small enough for convergence to have practically taken place.

```
Start: $iter = 1$

For $i \in \{1,\dots,n\}$ (where $\mathbf{x} \in \mathbb{R}^n$):
  For $k \in \{1,\dots,K\}$:
    Set $\phi_{i,k} = Softmax\left(x_i m_k - \frac{1}{2}\left(s_k^2 + m_k^2\right)\right)$ (equation 1.35)

For $k \in \{1,\dots,K\}$:
  Set $m_k = \frac{\sum_{i=1}^{n}\phi_{i,k}x_i}{\frac{1}{\sigma^2}+\sum_{i=1}^{n}\phi_{i,k}}$ and $s_k = \left(\frac{1}{\sigma^2} + \sum_{i=1}^{n}\phi_{i,k}\right)^{-1/2}$ (equation 1.34)

Compute: $\text{ELBO}_{iter}[q]$ using equations A.41 and A.42.

If: $|\text{ELBO}_{iter}[q] - \text{ELBO}_{iter-1}[q]| \geq \epsilon$ or $iter = 1$:
  Set: $iter = iter + 1$
```

Jump: To the first **For** loop above, and repeat the execution

End

A full programmatic demonstration of the execution of this algorithm is provided in appendix section A.10, with $\epsilon = 0.001$, $\sigma = 0.5$, $K = 10$ and $n = 100$. This program is in Python, with the inclusion of the common third-party module NumPy (Numerical Python).

In this program, a dataset **x** is first generated using distributions 1.25, 1.26 and 1.27, and then the **CAVI** algorithm for the **GMM** outlined above is executed using this dataset, retaining only the hyperparameters σ and K used in the dataset's generation. A plot of the **ELBO** for subsequent iterations would look like the left-hand plot in figure 1.4, as the contrived dataset **x** is not particularly noisy.

1.9 Stochastic Variational Inference

This section introduces the optimisation approach to **VI**, built on the **VI** conventions and framework that have so far been adopted, that is predominantly used in practice (as was briefly mentioned in section 1.7) - stochastic variational inference, commonly abbreviated to **SVI**.

The **SVI** optimisation approach is a method that combines natural gradients (Amari, 1998) and stochastic optimization (Blei, Kucukelbir, and McAuliffe, 2017; Robbins and Monro, 1951). This approach is a direct alternative to the **CAVI** optimisation approach, in which each optimisation step is taken in a more efficient direction, as opposed to being strictly in the directions of the individual coordinates, as is the case for **CAVI**.

Recall from equation 1.18 that the required optimisation for the **VI** method seeks to find a maximum of the associated **ELBO** function. In **SVI**, we take the gradient of the **ELBO** with respect to all of its coordinates - its variational parameters - at the same time, in order to determine direction of steepest ascent, which is in general some combination of the coordinate parameters. A step is then taken in this direction, the **ELBO** gradient recalculated, and this process repeated. At each step, all the coordinates of the **ELBO** are thus in general subject to variation, as opposed to only one of them, as was the case for **CAVI**.

Consider the following 3D graphs in figure 1.5 (adapted from the source: `https://datascience.stackexchange.com/questions/97555/how-to-model-a-3d-graph-into-a-vector-so-that-i-can-feed-it-into-a-classificatio`) that visually illustrates the general trajectories of the **ELBO** optimisation (maximisation) steps for the **CAVI** and **SVI** optimisation approaches, respectively.

Each of the graphs in figure 1.5 are of **ELBO**s with only 2 variational parameters (the most complex example that can be illustrated visually), one for each of the 2 horizontal axes. The value of the **ELBO** is depicted by the vertical axis. Each of the steps, for either the **CAVI** or **SVI** cases, are depicted by the series of red lines connected by blue dots (each red line segment is a step in the respective algorithm). In each algorithm, we start from a random point (depicted by the most left-hand blue dot in each graph), and then converge to a local maximum of the **ELBO**. Note, however,

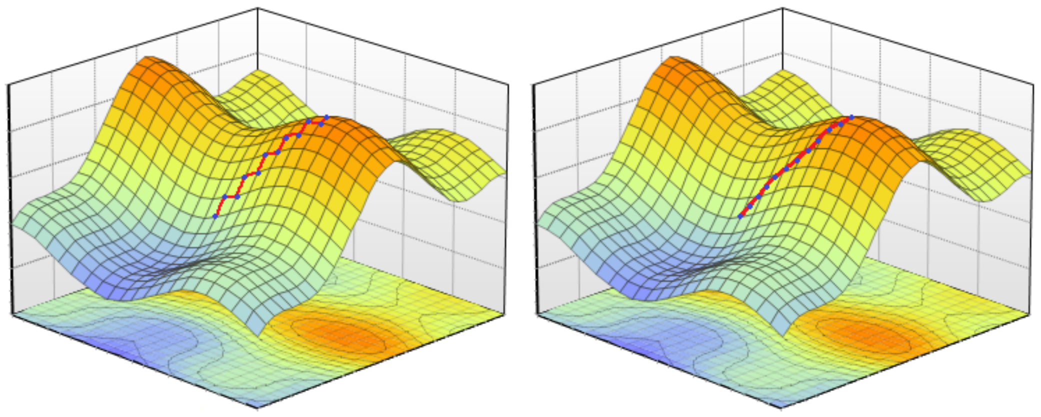


FIGURE 1.5: A Visual Illustration of the **ELBO** Optimisation (Maximisation) Trajectories of the **CAVI** (left) and **SVI** (right) Optimisation Approaches

that the steps for the **SVI** algorithm are far more direct and less staggered than for the **CAVI** algorithm, that can only step in the directions of the variational parameters (the coordinates). The generally shorter (as the trajectory path is not/less staggered) and more direct optimisation/maximisation trajectory of the **SVI** approach is what makes it considerably faster and more effective than **CAVI** in practice.

Unlike the step size in the **CAVI** approach, which is an inherent part of the **CAVI** solution of equation 1.24, the step size is manually chosen in the **SVI** optimisation approach - this step size, or learning rate, is commonly denoted by ρ_t, where t denotes which step number (or iteration) is of concern (the step size can be a function of the step number). It is important to choose an appropriate step size ρ_t for the **SVI** algorithm that will ensure efficient convergence without the steps "overshooting" the targeted local optimum/maximum of the **ELBO**. It has been demonstrated that the **SVI** algorithm is certain to converge to a local optimum of the **ELBO**, so long as the step size ρ_t abides by a Robbins-Monro schedule (Robbins and Monro, 1951):

$$\sum_{t=1}^{\infty} \rho_t = \infty \quad \text{and} \quad \sum_{t=1}^{\infty} \rho_t^2 < \infty \tag{1.36}$$

The "Stochastic" component in the name of the **SVI** algorithm stems from the fact that, unlike for the **CAVI** algorithm, we need not iterate thought the entire input dataset **x** at each optimisation step. The direct optimisation direction provided by the gradient of the **ELBO** means that we are able to only consider a random/stochastic sample of the input dataset **x** at each optimisation step, without significantly compromising the ideal optimisation direction at each step: Only considering a random sample means that an element of variation is introduced, that will thus induce a small amount of staggering in the trajectory, as is the case for the **CAVI** algorithm, although not to the same extent. However, the fact that only a sample of **x** needs to be considered at each optimisation step, with an element of staggering introduced from the variations of the samplings of **x** that is still less than than the staggering for **CAVI**, all means that the **SVI** algorithm is considerably more efficient than the **CAVI** algorithm in a practical sense.

A complete and concise algorithmic process of the **SVI** algorithm is given as follows (Blei, Kucukelbir, and McAuliffe, 2017; Hoffman et al., 2013):

```
Stochastic Variational Inference (SVI) Algorithm

Input: A model $p(\mathbf{x}, \mathbf{z})$, a dataset $\mathbf{x}$, and a step size sequence $\rho_i$.

Output: The fitted & optimised Variational Density $q(\mathbf{z})$.

Initialisation: Randomly initialise, as appropriate, the variational parameters of the VI model. Choose an $\epsilon$ that would be considered small enough for convergence to have practically taken place.

Start: $i = 1$

Let $\boldsymbol{\lambda}$ denote the vector set of variational parameters that govern the variational density $q(\mathbf{z})$, where $\boldsymbol{\lambda}_i$ denotes the settings of these variational parameters that govern $q(\mathbf{z})$ at iteration $i$.

Sample: A sufficiently large number of data points from the dataset $\mathbf{x}$ that is to act as a surrogate for subsequent computations involving $\mathbf{x}$ in this iteration.
Compute: $\nabla_{\boldsymbol{\lambda}} \text{ELBO}_i \left[q(\mathbf{z})\right]$ with the aid of equation 1.16
Set: $\boldsymbol{\lambda}_{i+1} = \boldsymbol{\lambda}_i + \rho_i \nabla_{\boldsymbol{\lambda}} \text{ELBO}_i$
Compute: $\text{ELBO}_{i+1} \left[q(\mathbf{z})\right]$ using the new settings of the variational parameters $\boldsymbol{\lambda}_{i+1}$ and equation 1.16
If: $|\text{ELBO}_{i+1}[q] - \text{ELBO}_i[q]| \geq \epsilon$ or $i = 1$:
    Set: $i = i + 1$
    Jump: To the first Compute statement, and repeat the execution

End
```

For the **SVI** algorithm, as outlined above, certain variants (such as the **ADVI** algorithm, that will be outlined in section 1.11) ultimately sample from the dataset $\mathbf{x}$ using the variational posteriors - $q(\mathbf{z}) \approx p(\mathbf{z}|\mathbf{x})$ - that are formulated at the end of each iteration (Ranganath, Gerrish, and Blei, 2014).

In initial research, prior to the discovery of the revelations that have been outlined in this section, neither **MCMC** nor **VI** scaled easily or sufficiently effectively to the kinds of massive data analysis requirements presented by modern industrial and academic demands. Researchers initially attempted to propose appropriate speed-ups of both approaches, but these were usually tailored to specific models or compromised the correctness of the algorithms (or both). However, the development of the general **SVI** method outlined above fundamentally changed the underlying scaling issue that plagued these previous approaches (Hoffman et al., 2013).

In the their seminal article "A stochastic approximation method" (Robbins and Monro, 1951), Robbins and Monro proved results implying that optimization algorithms in general can successfully use noisy, unbiased gradients, as long as the step size sequence satisfies the conditions outlined by equations 1.36 (Blei, Kucukelbir, and McAuliffe, 2017). This idea has since gained momentum (Blei, Kucukelbir, and McAuliffe, 2017; Kushner and Yin, 1997; Spall, 2005), enabling modern machine learning to scale to massive data via stochastic optimisation approaches in general (Blei, Kucukelbir, and McAuliffe, 2017; Bottou and Cun, 2003).

The foundations, conventions and frameworks, outlined and adopted in the sections prior to section 1.7, that were required for **CAVI**, are still required for the general **SVI** variant class that has been outlined in this section (above). **SVI** can be considered as "running in parallel" to **CAVI**, in that they are independent **VI** variant classes, but they are built on the same basic foundations, conventions and frameworks. **CAVI** was introduced as a first demonstration and illustration of a full **VI** method in action, as it is generally considered the simplest full **VI** variant class. However, the **SVI** variant class is what will henceforth be adopted and considered in the chapters and sections that follow in this thesis. Full implementations and demonstrations of the general **SVI** algorithm outlined above, as for the **CAVI** in section 1.8, will be outlined in the given research in subsequent chapters.

1.10 Automatic Differentiation

The particular **SVI** variant, from the general class of **SVI** variants outlined in section 1.9, that will underpin the research in this thesis is automatic differentiation variational inference. Automatic differentiation variational inference is commonly abbreviated to **ADVI**. However, prior to the introduction of **ADVI**, this section first introduces, considers and outlines automatic differentiation.

In general terms, there are 3 broad means by which derivatives can be algorithmically calculated: symbolic differentiation, numeric differentiation & automatic differentiation. Within the broader mathematics, statistics and computer science academic communities, symbolic differentiation and numeric differentiation are well understood and implemented, but the same can not be said for the latter mentioned main algorithmic means of calculating derivatives: automatic differentiation.

Symbolic differentiation is an analytical method used to calculate the derivatives of expressions that are given in a closed-form. Symbolic differentiation works by expanding out the more complicated expression that is to ultimately be derived into some linear combination of simpler expressions to be derived, that can then be derived using a series of well established rules for derivatives (e.g., sum rule, constant rule, chin rule, derivative of powers rule, etc). For example, consider the following symbolic means of calculating the derivative, $\frac{dy}{dx}$, of the expression $y = \sin^2(3x) + 2$:

$$\begin{aligned}
\frac{dy}{dx} &= \frac{d}{dx}\left(\sin^2(3x) + 2\right) \\
&= \frac{d}{dx}(\sin^2(3x)) + \frac{d}{dx}(2) \quad \text{(by the Sum rule)} \\
&= \frac{d}{dx}(\sin^2(3x)) + 0 \quad \text{(by the Constant rule)} \\
&= \frac{d}{dx}(\sin^2(3x)) \\
&= 2\sin(3x) * 3\cos(3x) \quad \text{(by the Chain rule)} \\
&= 6\cos(3x)\sin(3x)
\end{aligned} \tag{1.37}$$

Numeric differentiation, as its name suggests, is a numerical method used to calculate the derivatives of expressions, that may not necessarily be given in closed-form. Numeric differentiation works by numerically evaluating the derivative of the expression in question by first principles, to a given decimal accuracy δx (δx is a chosen small positive value, the magnitude of which determines the numerical accuracy). For example, consider the following numerical means of calculating the derivative, $\frac{dy}{dx}$, of the expression $y = \sin^2(3x) + 2$ ($= y(x)$):

$$\begin{aligned}\frac{dy}{dx} &= \frac{y(x+\delta x) - y(x)}{\delta x} \\ &= \frac{(\sin^2(3(x+\delta x)) + 2) - (\sin^2(3x) + 2)}{\delta x} \\ &= \frac{\sin^2(3x + 3\delta x) + 2 - \sin^2(3x) - 2}{\delta x} \\ &= \frac{\sin^2(3x + 3\delta x) - \sin^2(3x)}{\delta x}\end{aligned} \tag{1.38}$$

The crucial drawback of symbolic differentiation, when considering its application in modern practical machine learning settings, is that the expression that requires differentiating often need to be expanded into a linear combination with an exponentially large number of terms, as the complexity of such starting expressions increases - $\mathcal{O}(2^n)$. This exponential increase is the cause of symbolic differentiation becoming prohibitively slow in practice.

As for numeric differentiation, the crucial drawback is related to the fact that numeric differentiation is inherently an approximation method, the accuracy of which is governed by the quantity δx. If δx is too small, then the numeric differentiation of a given expression will require too large a computational time to execute. Also, if the numeric differentiation is done via computer, floating point errors may be encountered which will compromise the accuracy of the results. If δx is too large, then the approximate result may not have the required accuracy, or even "skip over" crucial points of inflection. It also suffers from the same prohibitively slow expression expansion issue as for symbolic differentiation, although not to the same extent - $\mathcal{O}(n)$.

Automatic differentiation, on the other hand, does not suffer from the exponential/-linear time complexity or approximation-based drawbacks that symbolic and numerical differentiation do: It provides exact derivatives in constant time. This capability of automatic differentiation stems from the relatively unknown concept of dual numbers.

A dual number is a number of the form $a + b\epsilon$, where $a, b \in \mathbb{R}$, $\epsilon \notin \mathbb{R}$ and $\epsilon^2 = 0$. Parallels can be considered when going from the mathematical concept of imaginary numbers to the mathematical concept of dual numbers, with i replaced by ϵ and $i^2 = -1$ replaced by $\epsilon^2 = 0$. The arithmetic for dual numbers is even defined in much the same was as for imaginary numbers, as briefly and broadly illustrated by the following equations, where $a, b, c, d \in \mathbb{R}$:

$$\begin{aligned}(a+b\epsilon)+(c+d\epsilon)&=(a+c)+(b+d)\epsilon\\(a+b\epsilon)*(c+d\epsilon)&=ac+ad\epsilon+bc\epsilon+bd\epsilon^2=ac+(ad+bc)\epsilon\quad(\text{As }\epsilon^2=0)\end{aligned}\tag{1.39}$$

However, despite these parallels, dual numbers and imaginary numbers remain independent concepts in mathematics generally.

The power of dual numbers in calculating derivatives in real time comes to fruition when considering their inclusion in Taylor series expansions. For example, the Taylor series expansion of a generic one-dimensional function $f(x)$, about a, with x substituted by the dual number $a+b\epsilon$, for $a,b\in\mathbb{R}$, is given as follows:

$$\begin{aligned}f(x)&=\sum_{n=0}^{\infty}\frac{f^{(n)}(a)}{n!}(x-a)^n\\f(a+\epsilon)&=\sum_{n=0}^{\infty}\frac{f^{(n)}(a)}{n!}((a+\epsilon)-a)^n\quad(\text{As }x=a+\epsilon)\\&=\sum_{n=0}^{\infty}\frac{f^{(n)}(a)}{n!}\epsilon^n\\&=f(a)+\frac{f'(a)}{1!}\epsilon+\frac{f''(a)}{2!}\epsilon^2+\ldots\\f(a+\epsilon)&=f(a)+\epsilon f'(a)\quad(\text{As }\epsilon^2=\epsilon*\epsilon^2=\epsilon^3=\cdots=0)\end{aligned}\tag{1.40}$$

As outlined by the last line in equation 1.40, the result of an appropriate inclusion of dual numbers in a Taylor series expansion is an expression of a function and its first derivative, each calculated in constant time (not suffering from the linear or exponential time complexity increases as for symbolic/numeric differentiation). The solution obtained is also an exact, as the terms in the Taylor series expansion that include derivatives with an order higher than one are not ignored, but are rather eliminated, as $\epsilon^n=\epsilon^{n-2}*\epsilon^2=\epsilon^{n-2}*0=0$ for any $n\in\mathbb{N}$ where $n\geq 2$.

There are two main automatic differentiation algorithms, both of which leverage this ingenious "differentiation via dual numbers" approach, accompanied by the desirable "constant time complexity of calculating a function's derivative" feature: Forward Mode Automatic Differentiation and Reverse Mode Automatic Differentiation. Simple examples of the application of each of these two automatic differentiation algorithms will be based on the article "Automatic Differentiation in Machine Learning: a Survey" (Baydin et al., 2018), for the function $f(x_1,x_2)=\log(x_1)+x_1x_2-\sin(x_2)$.

Firstly, the function $f(x_1,x_2)$ defined above needs to be broken apart into its constituent elementary functions and linear operations, as can be initially depicted by the following computational graph, which is also known as a Wengert list (Baydin et al., 2018; Wengert, 1964):

In figure 1.6, the intermediary nodes v_m, for $m\in\{1,2,3,4,5,6,7\}$, represent the layers of constituent functions that ultimately formulate the function $f(x_1,x_2)$. The directed vertices in figure 1.6 represent the elementary arithmetic operations that link these constituent functions together. A structured list of the precise functions and

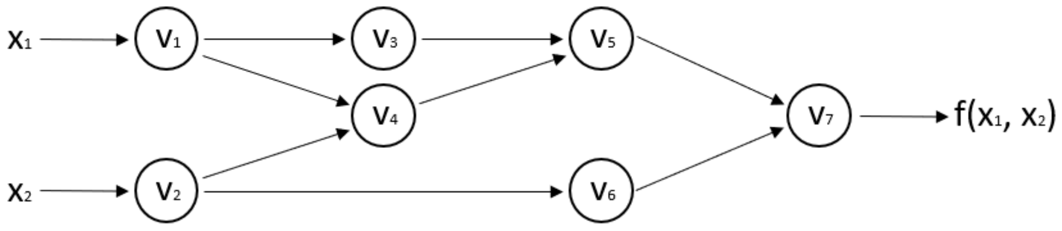


FIGURE 1.6: The computational graph, or Wengert list (Wengert, 1964), of the function $f(x_1, x_2) = \log(x_1) + x_1x_2 - \sin(x_2)$

arithmetic operations, via the series of layered functions v_m, for $m \in \{1,2,3,4,5,6,7\}$, that lead to an evaluation of the function $f(x_1, x_2)$, given specific values of the arguments x_1 & x_2, would be known as a primal trace of $f(x_1, x_2)$. The primal trace is the second essential component of the forward mode and reverse mode automatic differentiation algorithms. Consider the following example of a primal trace of $f(x_1, x_2)$ when $x_1 = 2$ and $x_2 = 3$:

1.10.1 Primal Trace: Of $f(x_1, x_2)$ for $x_1 = 2$ and $x_2 = 3$

$$\begin{aligned}
v_1 &= x_1 = 2 \\
v_2 &= x_2 = 3 \\
v_3 &= \log(v_1) = \log(2) \approx 0.693 \\
v_4 &= v_1 * v_2 = 2 * 3 = 6 \\
v_5 &= v_3 + v_4 \approx 0.693 + 6 = 6.693 \\
v_6 &= \sin(v_2) = \sin(3) \approx 0.141 \\
v_7 &= v_5 - v_6 \approx 6.693 - 0.141 = 6.552
\end{aligned} \tag{1.41}$$

It is at this stage that the forward mode and reverse mode automatic differentiation algorithms deviate from one another. The next stage for each of these algorithms is a calculation of the dual trace, which can be considered as a set of elementary derivatives that are structured and based on the corresponding primal trace. However, it is the precise derivatives and the means by which these derivatives are calculated that differentiates these two algorithms. Let the simpler of these two automatic differentiation algorithms - forward mode automatic differentiation - be considered first.

The dual trace of $f(x_1, x_2)$ with respect to x_2, based on the primal trace example outlined above, via the forward mode algorithm, is given as follows:

1.10.2 Dual Trace: Forward Mode, with respect to x_2

$$\begin{aligned}
\dot{v_1} &= \frac{\partial x_1}{\partial x_2} = 0 \\
\dot{v_2} &= \frac{\partial x_2}{\partial x_2} = 1 \\
\dot{v_3} &= \frac{\partial \log(v_1)}{\partial x_2} = \frac{\partial \log(x_1)}{\partial x_2} = 0 \\
\dot{v_4} &= \dot{v_1} v_2 + v_1 \dot{v_2} = (0 * 1) + (2 * 1) = 0 + 2 = 2 \\
\dot{v_5} &= \dot{v_3} + \dot{v_4} = 0 + 2 = 2 \\
\dot{v_6} &= \frac{\partial \sin(v_2)}{\partial x_2} = \frac{\partial \sin(x_2)}{\partial x_2} = \cos(x_2) = \cos(3) \approx -0.990 \\
\dot{v_7} &= \dot{v_5} - \dot{v_6} \approx 2 - (-0.990) = 2.990
\end{aligned} \tag{1.42}$$

Each of the $\dot{v_m}$ derivatives in dual trace 1.44, for $m \in \{1,2,3,4,5,6,7\}$, are the derivatives of v_m with respect to x_2: That is, $\dot{v_m} = \frac{\partial v_m}{\partial x_2}$. By inspection of the above primal and dual traces, it can be observed that v_m and $\dot{v_m}$ are effectively calculated in parallel, at the same time, adopting the dual evaluation feature of a function and its derivative as for a Taylor series expansion about a dual number (see equation 1.40). It is this parallel "function and derivative" analysis approach that ultimately allows automatic differentiation, in general, to compute derivatives of functions with a constant time complexity (as opposed to symbolic and numeric differentiation, with exponential and linear time complexities, respectively).

The main drawback of forward mode automatic differentiation is the need for the dual trace outlined above to be repeated for each input variable of the function to be derived. While this would not be an issue for the simple example outlined above (which only has two input variables, x_1 and x_2), this does pose a substantial problem for many machine learning and most deep learning applications, for which corresponding models tend to have a large number of input variables (in image recognition, for example, there are typically 3 input variables - the green, red & blue components - for each of the pixels that constitute the input images).

As a result, most machine learning and deep learning frameworks that require at least some element of automatic differentiation veer towards the "reverse mode automatic differentiation" algorithm, which does not require a recalculation of the dual trace for each of the input variables. Let reverse mode automatic differentiation now be considered.

We first need to consider two additional concepts prior to a full elaboration of the dual trace for the reverse mode automatic differentiation algorithm: The parent and adjoint of an intermediate function v_m, for some $m \in \{1,2,3,4,5,6,7\}$. A intermediate function v_m would be considered a parent of one or more of the other intermediate functions if v_m depends directly on those other intermediate functions, or equivalently, if v_m is at the endpoint of vertices in the Wengert list 1.6 that are connected to the nodes that denote those intermediate functions: For example, v_4 is a parent of both v_1 and v_2 in the example outlined above. Given a target output variable y_j, the adjoint of v_i, denoted by $\overline{v_i}$, is:

$$\overline{v_i} = \frac{\partial y_j}{\partial v_i} = \sum_{k=1}^{m} \frac{\partial y_j}{\partial v_{p_k}} \frac{\partial v_{p_k}}{\partial v_i} \tag{1.43}$$

The variable v_i in equation 1.43 has m parents, denoted by: $v_{p_1}, v_{p_2}, \dots, v_{p_m}$. Note that the adjoint of v_i in equation 1.43 - $\overline{v_i}$ - requires a specific target output variable - y_j - for that adjoint to be given with respect to.

The dual trace of $f(x_1, x_2)$ with respect to the one output variable $y = f(x_1, x_2)$, based on the primal trace example outlined above, via the reverse mode algorithm, is thus given as follows:

1.10.3 Dual Trace (Adjoint Trace): Reverse Mode

$$\begin{aligned}
\overline{v_7} &= \frac{\partial y}{\partial v_7} = \frac{\partial y}{\partial y} = 1 \\
\overline{v_6} &= \frac{\partial y}{\partial v_6} = \frac{\partial y}{\partial v_7}\frac{\partial v_7}{\partial v_6} = 1 * (-1) = -1 \\
\overline{v_5} &= \frac{\partial y}{\partial v_5} = \frac{\partial y}{\partial v_7}\frac{\partial v_7}{\partial v_5} = 1 * 1 = 1 \\
\overline{v_4} &= \frac{\partial y}{\partial v_4} = \frac{\partial y}{\partial v_5}\frac{\partial v_5}{\partial v_4} = 1 * 1 = 1 \\
\overline{v_3} &= \frac{\partial y}{\partial v_3} = \frac{\partial y}{\partial v_5}\frac{\partial v_5}{\partial v_3} = 1 * 1 = 1 \\
\overline{v_2} &= \frac{\partial y}{\partial v_2} = \frac{\partial y}{\partial v_4}\frac{\partial v_4}{\partial v_2} + \frac{\partial y}{\partial v_6}\frac{\partial v_6}{\partial v_2} = (1 * v_1) + ((-1) * \cos(v_2)) = v_1 - \cos(v_2) \\
&= 2 - \cos(3) \approx 2 - (-0.990) = 2 + 0.990 = 2.990 \\
\overline{v_1} &= \frac{\partial y}{\partial v_1} = \frac{\partial y}{\partial v_3}\frac{\partial v_3}{\partial v_1} + \frac{\partial y}{\partial v_4}\frac{\partial v_4}{\partial v_1} = \left(1 * \frac{1}{v_1}\right) + (1 * v_2) = \left(1 * \frac{1}{2}\right) + (1 * 3) \\
&= \frac{1}{2} + 3 = 3.5
\end{aligned} \tag{1.44}$$

As the dual trace for the reverse mode algorithm constitutes a set of equations of the adjoints of each of the intermediate functions from the associated Primal Trace / Wengert list, the dual trace is also known as the "Adjoint Trace" for the reverse mode automatic differentiation algorithm. The fundamental derivative concept that underpins the reverse mode algorithm is the chain rule.

Likewise for the forward mode version of the automatic differentiation for the input variables, the dual trace outlined immediately above for the reverse mode version of automatic differentiation needs to be repeated for each output variable of the function to be derived.

The forward/reverse mode automatic differentiation algorithms are so named due to, and depending on, whether the given algorithms scale with respect to the number of input/output variables - due to the need to recalculate the associated dual trace for each input or output variable, as appropriate. That is, the time complexity of forward and reverse mode automatic differentiation only scale with respect to the number of input and output variables, respectively, of the model/function to be derived.

Let n denote the number of input variables, and m denote the number of output variables, of a function to be derived. If $n < m$, then the forward mode version of automatic differentiation should be used. If $n > m$ - in practice, for many machine learning and most deep learning applications, we have $n >> m$ - then the reverse mode version of automatic differentiation should be used.

The outline and example in this section provide the core concepts and intuitions behind automatic differentiation. Automatic differentiation is an essential part of **ADVI** - Automatic Differentiation Variational Inference - so essential, that **ADVI** has been named after automatic differentiation. **ADVI** is covered in the next section - section 1.11.

1.11 Automatic Differentiation Variational Inference

Automatic differentiation variational inference - **ADVI** - is a specific variant of **SVI** (see section 1.9) that fundamentally leverages the computational power of automatic differentiation (see section 1.10) for the benefit of modern computational statistics, machine learning and deep learning, in the context of **VI**. It is the specific **VI** variant that underpins the theme and research of this thesis, as will be outlined in the subsequent chapters, and that will be introduced in this section.

The first step of the **ADVI** algorithm is to transform the latent variables of the input model, if required, to being unconstrained. In keeping with the notational consistency of the academic article "Automatic Differentiation Variational Inference" (Kucukelbir et al., 2017), which is the primary seed article for this section, let $\boldsymbol{\theta}$ denote the latent variables of the input model and $\boldsymbol{\zeta}$ denote the corresponding transformed latent variables that are unconstrained: That is, $\boldsymbol{\zeta} \in \mathbb{R}^m$, where the dimension of each of $\boldsymbol{\theta}$ and $\boldsymbol{\zeta}$ is some $m \in \mathbb{N}$. Note that in order for the **ADVI** algorithm to be applicable, we require the variables in $\boldsymbol{\theta}$ to be continuous and the gradient $\nabla_{\boldsymbol{\theta}} \log p(\mathbf{x}, \boldsymbol{\theta})$ to be defined (Kucukelbir et al., 2017). Consider the one-dimensional latent vector transformation example provided in the paper "Automatic Differentiation Variational Inference" (Kucukelbir et al., 2017), where $x|\theta \sim \text{Poisson}(\theta)$ (x still denotes the observation) and $\theta \sim \text{Weibull}(1.5, 1)$. That is, where:

$$\begin{aligned} p(x|\theta) &= \text{Poisson}(x;\theta) = \frac{\theta^x}{x!}e^{-\theta} \\ p(\theta) &= \text{Weibull}\left(\theta; \frac{3}{2}, 1\right) = \frac{3}{2}\theta^{1/2}e^{-\theta^{3/2}} \end{aligned} \tag{1.45}$$

The equations of 1.45 define an input model ($p(x,\theta) = p(x|\theta)p(\theta)$) for which the latent variable θ must live in the space $\mathbb{R}^{++}$ - i.e., $\theta \in \mathbb{R}^{++}$. Therefore, a transformation τ, defined by $\zeta = \tau(\theta) = \log(\theta)$, can be used to map θ in the constrained space $\mathbb{R}^{++}$ to ζ in the unconstrained real space $\mathbb{R}$, as required for **ADVI**.

This "real & unconstrained"-based transformation of the initial latent variable θ allows an unconstrained-based optimisation to be carried out, which is more computationally efficient and convenient in an automatic differentiation context. The constraints applicable to θ are absorbed into an appropriate expansion of its latent variable space, via the transformation τ, to the unconstrained space. These principles apply in general.

In the article "Automatic Differentiation Variational Inference" (Kucukelbir et al., 2017), the surrogate posterior q only takes the form of a Gaussian density. This is done as the Gaussian density encapsulates amenable mathematical features, that will be elaborated on, that allow much of the mathematics and process in the **ADVI** algorithm to be established, in a way that would not otherwise be possible. As the

Gaussian density has a domain over $\mathbb{R}$ for each of its input components, then the parameter of q for the example above is already unconstrained (in $\mathbb{R}$). We can therefore denote the surrogate posterior directly as $q(\xi)$: No such initial transformation is required for the parameter of q. The **ELBO** of the **VI** method, given model 1.45, can thus be expressed and manipulated as follows:

$$\begin{aligned}
\text{ELBO}[q(\xi)] &= \mathbb{E}_{\xi \sim q(\xi)}\left[\log\left(\frac{p(x,\theta)}{q(\xi)}\right)\right] \\
&= \mathbb{E}_{\xi \sim q(\xi)}\left[\log\left(p(x,\theta)\right) - \log\left(q(\xi)\right)\right] \\
&= \mathbb{E}_{\xi \sim q(\xi)}\left[\log\left(p(x|\theta)p(\theta)\right) - \log\left(q(\xi)\right)\right] \\
&= \mathbb{E}_{\xi \sim q(\xi)}\left[\log\left[p(x|\tau^{-1}(\xi))p(\tau^{-1}(\xi)) \cdot |\det \mathcal{J}_{\tau^{-1}}(\xi)|\right] - \log\left(q(\xi)\right)\right] \\
&= \mathbb{E}_{\xi \sim q(\xi)}\left[\log\left(p(x|\tau^{-1}(\xi))p(\tau^{-1}(\xi))\right) + \log|\det \mathcal{J}_{\tau^{-1}}(\xi)| - \log\left(q(\xi)\right)\right] \\
&= \mathbb{E}_{\xi \sim q(\xi)}\left[\log\left(p(x|\tau^{-1}(\xi))p(\tau^{-1}(\xi))\right) + \log|\det \mathcal{J}_{\tau^{-1}}(\xi)|\right] \\
&\qquad + \mathbb{E}_{\xi \sim q(\xi)}[-\log\left(q(\xi)\right)] \\
&= \mathbb{E}_{\xi \sim q(\xi)}\left[\log\left(p(x|\tau^{-1}(\xi))p(\tau^{-1}(\xi))\right) + \log|\det \mathcal{J}_{\tau^{-1}}(\xi)|\right] \\
&\qquad + \mathbb{H}_{\xi \sim q(\xi)}[q(\xi)]
\end{aligned} \tag{1.46}$$

The first equality in 1.46 follows from an application of equation 1.16 in the context of model 1.45 and the established form of q (outlined above). The last equality in 1.46 follows from the definition of entropy: That is, the entropy of the variational density $q(\xi)$, which is defined as $\mathbb{E}_{\xi \sim q(\xi)}[-\log(q(\xi))] = \mathbb{H}_{\xi \sim q(\xi)}[q(\xi)]$.

The transformation τ defined and outlined above is a bijection (as the log function is a bijection): In general for **ADVI**, all such transformations, used to transform the initially constrained latent variables to being unconstrained, must be bijections. Therefore, the inverse function τ^{-1} exists: As $\xi = \tau(\theta) = \log(\theta)$, then $\theta = \tau^{-1}(\xi) = e^{\xi}$. Transformations of continuous probability densities, such as $p(x|\theta)$ and $p(\theta)$ in 1.45, require an appropriate Jacobian; it accounts for how the transformation warps unit volumes and ensures that the transformed density integrates to one. A transformation of the constrained latent variable θ to its unconstrained counterpart ξ, in model 1.45, leads to the following equality (Olive, 2014; Kucukelbir et al., 2017):

$$p(x|\theta)p(\theta) = p(x|\tau^{-1}(\xi))p(\tau^{-1}(\xi)) \cdot |\det \mathcal{J}_{\tau^{-1}}(\xi)| \tag{1.47}$$

The fourth equality in 1.46 follows from a substitution of equation 1.47. Equations 1.46 and 1.47 hold in general: They are not just applicable to the particular example outlined by model 1.45.

An initial step of the **ADVI** algorithm is to consider which form of Gaussian density the surrogate posterior q is to take: Either a "Mean-Field Gaussian" or a "Full-Rank Gaussian". In general, we have $q(\boldsymbol{\xi}) = \mathcal{N}(\boldsymbol{\xi}; \boldsymbol{\mu}, \Sigma)$, where $\boldsymbol{\xi}$ and $\boldsymbol{\mu}$ are m-dimensional vectors, and Σ is an $m \times m$ matrix, for some $m \in \mathbb{N}$. As before $\boldsymbol{\xi}$ is a vector of the unconstrained versions of the latent variables $\boldsymbol{\theta}$ of the **VI** model. The vector $\boldsymbol{\mu}$ and the matrix Σ are the mean and covariance, respectively, of the Gaussian density that is taken for q.

For a "Mean-Field Gaussian" based surrogate posterior, the associated covariance matrix Σ is a diagonal matrix: The (general) multivariate Gaussian density that is taken for q is just a product of univariate Gaussian densities (see section 1.6). For a "Full-Rank Gaussian" based surrogate posterior, the associated covariance matrix Σ can take the form of any valid covariance matrix.

Just as for the latent vectors $\boldsymbol{\theta}$ of an input model $p(\mathbf{x}, \boldsymbol{\theta})$, and for the same reasons, the variational parameters of the surrogate posterior q must be unconstrained. The mean vector of $q(\boldsymbol{\xi})$, regardless of whether q takes the form of a "Mean-Field Gaussian" or a "Full-Rank Gaussian", is already unconstrained, so no transformation is required here. However, the same cannot be said for the associated covariance matrix Σ.

When q takes the form of a "Mean-Field Gaussian", then $q(\boldsymbol{\xi}; \boldsymbol{\phi}) = \prod_{i=1}^{m} \mathcal{N}(\xi_i; \mu_i, \sigma_i^2)$, where $\boldsymbol{\xi} = (\xi_1 \quad \xi_2 \quad \dots \quad \xi_m)^T$ and $\boldsymbol{\phi} = \{\mu_1, \mu_2, \dots, \mu_m, \sigma_1, \sigma_2, \dots, \sigma_m\}$, for some $m \in \mathbb{N}$. The variances of each of the variational factors - σ_i^2 for $i \in \{1, 2, \dots, m\}$ - must be positive, but a reparameterisation of $\omega_i = \log(\sigma_i)$ can be applied to remove these constraints. The final set of unconstrained variational parameters for q, over which the required **VI** optimisation is to be carried out, is $\boldsymbol{\phi}' = \{\mu_1, \mu_2, \dots, \mu_m, \omega_1, \omega_2, \dots, \omega_m\}$.

When q takes the form of a "Full-Rank Gaussian", then $q(\boldsymbol{\xi}; \boldsymbol{\phi}) = \mathcal{N}(\boldsymbol{\xi}; \boldsymbol{\mu}, \Sigma)$ in general, where, as before, $\boldsymbol{\xi} = (\xi_1 \quad \xi_2 \quad \dots \quad \xi_m)^T$. As the covariance matrix Σ for this form of q can take the form of any generic covariance matrix of the same dimension, then it can be appropriately re-parameterised via a Cholesky factorisation (Cholesky, 1924; Kucukelbir et al., 2017): $\Sigma = LL^T$. Σ can denote a valid covariance matrix so long as it is a symmetric and positive definite matrix: Any symmetric and positive definite matrix can be decomposed via the product LL^T, where L is an unconstrained real lower-triangular matrix. The final set of unconstrained variational parameters for q, over which the required **VI** optimisation is to be carried out, is thus $\boldsymbol{\phi}' = \{\mu_1, \mu_2, \dots, \mu_m, l_1, l_2, \dots, l_{\frac{1}{2}m(m+1)}\}$ (the unconstrained real lower-triangular matrix L has $\frac{1}{2}m(m+1)$ free variational parameters l), for some $m \in \mathbb{N}$.

Whether the surrogate posterior q is chosen to take the form of a "Mean-Field Gaussian" or a "Full-Rank Gaussian" depends on the situation and context for which the **ADVI** algorithm is being employed. The mean-field version is computationally cheaper (considerably so when the dimension of the latent vector is large), while still reasonably approximating the true posterior in many situations (see section 1.6), but the full-rank version guarantees a more accurate approximation of the true posterior.

Recall that **ADVI** is a variant of **SVI**, meaning that each iteration of the **ADVI** algorithm requires a stochastic gradient ascent of the **ELBO** (see section 1.9). In keeping with the notation outlined and adopted above, in this section, the most generalised form of the gradient ascent step is summarised by the following equation (see section 1.9):

$$\boldsymbol{\phi}_{i+1} = \boldsymbol{\phi}_i + \rho_i \cdot \nabla_{\boldsymbol{\phi}} ELBO[q_{\boldsymbol{\phi}}] \tag{1.48}$$

Here: i is a numerical index of the number of iterations leading up to, and including, the current iteration; $\boldsymbol{\phi}_i$ denotes the specific values of the unconstrained versions of the variational parameters at iteration i; ρ_i denotes the step size at iteration i, which is in general adaptive, and depends on the iteration index i; $\nabla_{\boldsymbol{\phi}}$ denotes the gradient with respect to each of the unconstrained versions of the variational parameters, in the set $\boldsymbol{\phi}$; and $ELBO[q_{\boldsymbol{\phi}}]$ denotes the **ELBO** of $q_{\boldsymbol{\phi}} = q_{\boldsymbol{\phi}}(\boldsymbol{\zeta}) = q(\boldsymbol{\zeta};\boldsymbol{\phi})$, given the current values of the (unconstrained) variational parameters $\boldsymbol{\phi}$.

In order for equation 1.48 to be evaluated at each iterative step of the **ADVI** algorithm, a means of calculating $\nabla_{\boldsymbol{\phi}} ELBO[q_{\boldsymbol{\phi}}]$ is required for each of the possible unconstrained variational parameter types: μ_i, ω_i and l_i, as appropriate, and as outlined above.

Given the equality of the first and last expressions in 1.46, and considering an extension of the example considered in those equations, where the dataset and unconstrained latent vector may now be of any dimension, an expression of the **ELBO** as two additive terms can be established as follows:

$$\begin{aligned} ELBO[q(\boldsymbol{\zeta})] = \mathbb{E}_{\zeta\sim q(\zeta)}\left[\log\left(p(\mathbf{x}|\tau^{-1}(\boldsymbol{\zeta}))p(\tau^{-1}(\boldsymbol{\zeta}))\right) + \log|\det \mathcal{J}_{\tau^{-1}}(\boldsymbol{\zeta})|\right] \\ + \mathbb{H}_{\zeta\sim q(\zeta)}[q(\boldsymbol{\zeta})] \end{aligned} \tag{1.49}$$

Let the latter entropy term on the right hand side of equation 1.49, and its respective gradients, as appropriate, first be considered. As q takes the form of a Gaussian density, where $q(\boldsymbol{\zeta}) = \mathcal{N}(\boldsymbol{\zeta};\boldsymbol{\mu},\Sigma)$, then:

$$\mathbb{H}_{\zeta\sim q(\zeta)}[q(\boldsymbol{\zeta})] = \frac{m}{2}(1+\log(2\pi)) + \frac{1}{2}\log|\Sigma| \tag{1.50}$$

(as stated previously, $\boldsymbol{\zeta}$ is an m-dimensional vector and Σ is an $m \times m$ matrix).

By inspection of equation 1.50, it is thus evident that:

$$\nabla_{\boldsymbol{\mu}}\mathbb{H}_{\zeta\sim q(\zeta)} = \mathbf{0} \tag{1.51}$$

The $\mathbf{0}$ in equation 1.51 is an m-dimensional zero vector. This derivative result follows from the fact that the entropy of the variational density, outlined in equation 1.50, does not depend on the mean vector $\boldsymbol{\mu}$.

When q takes the form of a "Mean-Field Gaussian", where $q(\boldsymbol{\zeta};\boldsymbol{\phi}) = \mathcal{N}(\boldsymbol{\zeta};\boldsymbol{\mu},\Sigma) = \prod_{i=1}^{m}\mathcal{N}(\zeta_i;\mu_i,\sigma_i^2)$ and $\omega_i = \log(\sigma_i)$, then:

$$
\begin{aligned}
\mathbb{H}_{\xi\sim q(\xi)}[q(\boldsymbol{\xi})] &= \frac{m}{2}(1+\log(2\pi)) + \frac{1}{2}\log|\Sigma| \\
&= \frac{m}{2}(1+\log(2\pi)) + \frac{1}{2}\log\left(\prod_{i=1}^{m}\sigma_i^2\right) \\
&= \frac{m}{2}(1+\log(2\pi)) + \frac{1}{2}\log\left(\prod_{i=1}^{m}e^{\log(\sigma_i^2)}\right) \\
&= \frac{m}{2}(1+\log(2\pi)) + \frac{1}{2}\log\left(\prod_{i=1}^{m}e^{2\log(\sigma_i)}\right) \\
&= \frac{m}{2}(1+\log(2\pi)) + \frac{1}{2}\log\left(\prod_{i=1}^{m}e^{2\omega_i}\right) \\
&= \frac{m}{2}(1+\log(2\pi)) + \frac{1}{2}\sum_{i=1}^{m}\log(e^{2\omega_i}) \\
&= \frac{m}{2}(1+\log(2\pi)) + \frac{1}{2}\sum_{i=1}^{m}(2\omega_i) \\
&= \frac{m}{2}(1+\log(2\pi)) + \sum_{i=1}^{m}\omega_i
\end{aligned}
\tag{1.52}
$$

By inspection of the first and last expressions in equations 1.52, it is thus evident that:

$$\nabla_{\boldsymbol{\omega}}\mathbb{H}_{\xi\sim q(\xi)} = \mathbf{1} \tag{1.53}$$

The $\mathbf{1}$ in equation 1.53 is an m-dimensional vector of 1s. This derivative result follows from the linear means by which the "Mean-Field Gaussian" based entropy $\mathbb{H}_{\xi\sim q(\xi)}[q(\boldsymbol{\xi})]$ depends on ω_i, for each $i \in \{1, 2, \ldots, m\}$, where $m \in \mathbb{N}$. That is, by equating the first and last expressions in 1.52, it can be deduced that $\nabla_{\boldsymbol{\omega}}\mathbb{H}_{\xi\sim q(\xi)} = (\nabla_{\omega_1}\mathbb{H}_{\xi\sim q(\xi)} \quad \nabla_{\omega_2}\mathbb{H}_{\xi\sim q(\xi)} \quad \ldots \quad \nabla_{\omega_m}\mathbb{H}_{\xi\sim q(\xi)})^T = (1 \quad 1 \quad \ldots \quad 1)^T = \mathbf{1}$.

When q takes the form of a "Full-Rank Gaussian", where $q(\boldsymbol{\xi};\boldsymbol{\phi}) = \mathcal{N}(\boldsymbol{\xi};\boldsymbol{\mu},\Sigma)$, $\Sigma = LL^T$, and L is an unconstrained real lower-triangular matrix, then:

$$
\begin{aligned}
\mathbb{H}_{\xi\sim q(\xi)}[q(\boldsymbol{\xi})] &= \frac{m}{2}(1+\log(2\pi)) + \frac{1}{2}\log|\Sigma| \\
&= \frac{m}{2}(1+\log(2\pi)) + \frac{1}{2}\log|LL^T|
\end{aligned}
\tag{1.54}
$$

A derivation of the covariance-based gradient $\nabla_L\mathbb{H}_{\xi\sim q(\xi)}$ in this full-rank case is more complex than for its mean-field counterpart - $\nabla_{\boldsymbol{\omega}}\mathbb{H}_{\xi\sim q(\xi)}$ - but the result of the derivation is still a closed-form expression, as is summarised by the following equation (Kucukelbir et al., 2017):

$$\nabla_L\mathbb{H}_{\xi\sim q(\xi)} = (L^{-1})^T \tag{1.55}$$

A derivation of the gradient $\nabla_L \mathbb{H}_{\xi \sim q(\xi)}$, in broad terms, can be considered as a matrix-based extrapolation of the derivation of $\nabla_\omega \mathbb{H}_{\xi \sim q(\xi)}$, as outlined by equations 1.52 and 1.53 (Kucukelbir et al., 2017).

Now let the prior of the two terms on the right hand side of equation 1.49, and its respective gradients, as appropriate, be considered. That is, we now seek a means of evaluating the following expressions: $\mathbb{E}_{\xi \sim q(\xi)}[f(\boldsymbol{\xi})]$, $\nabla_\mu \mathbb{E}_{\xi \sim q(\xi)}[f(\boldsymbol{\xi})]$, $\nabla_\omega \mathbb{E}_{\xi \sim q(\xi)}[f(\boldsymbol{\xi})]$ and $\nabla_L \mathbb{E}_{\xi \sim q(\xi)}[f(\boldsymbol{\xi})]$, where $f(\boldsymbol{\xi}) = \log(p(\mathbf{x}|\tau^{-1}(\boldsymbol{\xi}))p(\tau^{-1}(\boldsymbol{\xi}))) + \log|\det \mathcal{J}_{\tau^{-1}}(\boldsymbol{\xi})|$ (see equation 1.49).

The first step in evaluating the three gradient-based expressions above is to "push" the respective gradients inside their expectation functions: I.e., somehow transforming $\nabla_\alpha \mathbb{E}[f]$ to $\mathbb{E}[\nabla_\alpha f]$, where ∇_α is the gradient in question. This is achieved via a final transformation, for Gaussian densities, known as "Elliptical Standardization" (Kucukelbir et al., 2017).

The elliptical standardization transformation is denoted by $S_{\boldsymbol{\phi}}$ in the paper "Automatic Differentiation Variational Inference" (Kucukelbir et al., 2017), and transforms the unconstrained latent variables $\boldsymbol{\xi}$ to a standardised mapping of those latent variables, denoted by $\boldsymbol{\eta}$: That is, we have $\boldsymbol{\eta} = S_{\boldsymbol{\phi}}(\boldsymbol{\xi})$. As the surrogate posterior $q(\boldsymbol{\xi})$ is only assumed to take the form of a Gaussian density, then the values of the scalar variables in $\boldsymbol{\xi}$ are normally distributed. $\boldsymbol{\eta}$ is then the set of these variables that are transformed (standardised) such that they are modelled by a Gaussian density with a mean of $\mathbf{0}$ and a covariance of I. That is, we have $q(\boldsymbol{\eta}) = \mathcal{N}(\boldsymbol{\eta}; \mathbf{0}, I) = \prod_{i=1}^{m} \mathcal{N}(\eta_i; 0, 1)$, where $\boldsymbol{\eta}$ is an m-dimensional vector, $\mathbf{0}$ is an m-dimensional zero-vector, and I is an $m \times m$ identity matrix.

Given the case outlined above where q takes the form of a mean-field Gaussian density - $q(\boldsymbol{\xi}; \boldsymbol{\phi}) = \prod_{i=1}^{m} \mathcal{N}(\xi_i; \mu_i, \sigma_i^2)$ - and where $\omega_i = \log(\sigma_i)$, for $i \in \{1, 2, \ldots, m\}$ and $\boldsymbol{\phi} = \{\mu_1, \mu_2, \ldots, \mu_m, \omega_1, \omega_2, \ldots, \omega_m\}$, it can thus be deduced that:

$$
\begin{aligned}
\eta_i = S_{\boldsymbol{\phi}}(\xi_i) = \frac{\xi_i - \mu_i}{\sigma_i} = \frac{\xi_i - \mu_i}{e^{\omega_i}} \\
\Rightarrow \boldsymbol{\eta} = S_{\boldsymbol{\phi}}(\boldsymbol{\xi}) = diag(\exp(\boldsymbol{\omega}))^{-1}(\boldsymbol{\xi} - \boldsymbol{\mu})
\end{aligned}
\tag{1.56}
$$

Given the case outlined above where q takes the form of a full-rank Gaussian density - $q(\boldsymbol{\xi}; \boldsymbol{\phi}) = \mathcal{N}(\boldsymbol{\xi}; \boldsymbol{\mu}, \Sigma)$ - and where $\Sigma = LL^T$, for some unconstrained real lower-triangular matrix L, meaning that $\boldsymbol{\phi} = \{\boldsymbol{\mu}, L\} = \{\mu_1, \mu_2, \ldots, \mu_m, l_1, l_2, \ldots, l_{\frac{1}{2}m(m+1)}\}$, it can thus be deduced that:

$$
\boldsymbol{\eta} = S_{\boldsymbol{\phi}}(\boldsymbol{\xi}) = L^{-1}(\boldsymbol{\xi} - \boldsymbol{\mu}) \tag{1.57}
$$

All standardisation transformations of Gaussian densities are bijections. Therefore, the function $S_{\boldsymbol{\phi}}^{-1}$ exists. Given the definitions of $S_{\boldsymbol{\phi}}$ outlined in equations 1.56 and 1.57, it can thus be deduced that:

$$\begin{aligned}
\xi_i = S_{\boldsymbol{\phi}}^{-1}(\eta_i) = \eta_i e^{\omega_i} + \mu_i \quad & \text{(Mean-Field case, where } i \in \{1, 2, \ldots, m\}) \\
\boldsymbol{\xi} = S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta}) = diag(\exp(\boldsymbol{\omega}))\boldsymbol{\eta} + \boldsymbol{\mu} \quad & \text{(Mean-Field case)} \\
\boldsymbol{\xi} = S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta}) = L\boldsymbol{\eta} + \boldsymbol{\mu} \quad & \text{(Full-Rank case)}
\end{aligned} \tag{1.58}$$

Substituting the inverse elliptical standardisation transformation $S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta})$ in place of $\boldsymbol{\xi}$ (as $\boldsymbol{\xi} = S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta})$), for $\mathbb{E}_{\boldsymbol{\xi} \sim q(\boldsymbol{\xi})}[f(\boldsymbol{\xi})]$, we have:

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{\xi} \sim q(\boldsymbol{\xi})}[f(\boldsymbol{\xi})] &= \mathbb{E}_{\boldsymbol{\xi} \sim q(\boldsymbol{\xi})}[\log(p(\mathbf{x}|\tau^{-1}(\boldsymbol{\xi}))p(\tau^{-1}(\boldsymbol{\xi}))) + \log|\det \mathcal{J}_{\tau^{-1}}(\boldsymbol{\xi})|] \\
&= \mathbb{E}_{\boldsymbol{\eta} \sim \mathcal{N}(\mathbf{0}, I)}\left[\log\left(p\left(\mathbf{x}|\tau^{-1}\left(S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta})\right)\right)p\left(\tau^{-1}\left(S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta})\right)\right)\right)\right] \\
&\qquad + \mathbb{E}_{\boldsymbol{\eta} \sim \mathcal{N}(\mathbf{0}, I)}\left[\log\left|\det \mathcal{J}_{\tau^{-1}}\left(S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta})\right)\right|\right] \\
&= \mathbb{E}_{\boldsymbol{\eta} \sim \mathcal{N}(\mathbf{0}, I)}\left[f\left(S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta})\right)\right]
\end{aligned} \tag{1.59}$$

(the third expression in derivation 1.59 has been expressed as two expectation terms, instead of just one, so that it fits on the page)

In the specific case of an elliptical standardisation transformation, a multiplication by the Jacobian-based term $|\det \mathcal{J}_{S_{\boldsymbol{\phi}}^{-1}}(\boldsymbol{\eta})|$ (that generally accounts for how a given transformation warps unit volumes, as has already been briefly outlined above) is not required (Kucukelbir et al., 2017). As a means of gaining an intuitive sense as to why this is the case, as well as why the expectation is adapted from being with respect to "$\boldsymbol{\xi} \sim q(\boldsymbol{\xi})$" to being with respect to "$\boldsymbol{\eta} \sim \mathcal{N}(\mathbf{0}, I)$", please refer to appendix section A.11.

Since the resulting expectation in derivation 1.59 is with respect to $\boldsymbol{\eta}$, and is thus not in any way dependent on $\boldsymbol{\mu}$, $\boldsymbol{\omega}$ or L, we can calculate the gradients $\nabla_{\mu/\omega/L}\mathbb{E}_{\eta \sim \mathcal{N}(\mathbf{0}, I)}\left[f\left(S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta})\right)\right]$ by "pushing" the gradient component inside the expectation. That is:

$$\begin{aligned}
\nabla_{\mu}\mathbb{E}_{\eta \sim \mathcal{N}(\mathbf{0}, I)}\left[f\left(S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta})\right)\right] &= \mathbb{E}_{\eta \sim \mathcal{N}(\mathbf{0}, I)}\left[\nabla_{\mu} f\left(S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta})\right)\right] \\
\nabla_{\omega}\mathbb{E}_{\eta \sim \mathcal{N}(\mathbf{0}, I)}\left[f\left(S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta})\right)\right] &= \mathbb{E}_{\eta \sim \mathcal{N}(\mathbf{0}, I)}\left[\nabla_{\omega} f\left(S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta})\right)\right] \\
\nabla_{L}\mathbb{E}_{\eta \sim \mathcal{N}(\mathbf{0}, I)}\left[f\left(S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta})\right)\right] &= \mathbb{E}_{\eta \sim \mathcal{N}(\mathbf{0}, I)}\left[\nabla_{L} f\left(S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta})\right)\right]
\end{aligned} \tag{1.60}$$

The resulting expression inside the expectation - $\nabla_{\alpha} f\left(S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta})\right)$ - is then calculated using "Automatic Differentiation" (see section 1.10). This is the unique and revolutionary step in the **ADVI** algorithm that has led to the **ADVI** algorithm being named after automatic differentiation.

However, recall from section 1.10 that calculations of derivatives using automatic differentiation must be done about specific points. Such specific points are obtained using **MC** integration (Monte Carlo integration) - a sampling-based estimation approach - as a means of calculating the outer expectations (Robert, Casella, and Casella, 1999).

For a given argument $A(\boldsymbol{\eta})$ of the expectation $\mathbb{E}_{\eta \sim \mathcal{N}(\mathbf{0},I)}$, an **MC** integration-based approximation of the expectation $\mathbb{E}_{\eta \sim \mathcal{N}(\mathbf{0},I)}[A(\boldsymbol{\eta})]$ is given by the following equation:

$$\mathbb{E}_{\eta \sim \mathcal{N}(\mathbf{0},I)}[A(\boldsymbol{\eta})] \approx \frac{1}{S} \sum_{s=1}^{S} A(\boldsymbol{\eta}_s) \tag{1.61}$$

In equation 1.61, $S \in \mathbb{N}$ denotes a finite number of samples - $\{\boldsymbol{\eta}_1, \boldsymbol{\eta}_2, \ldots, \boldsymbol{\eta}_S\}$ - that are drawn from the distribution $\mathcal{N}(\mathbf{0}, I)$, as representatives of $\boldsymbol{\eta}$. The larger the chosen value of S, the more accurate the **MC** integration approximation in equation 1.61 will be, but a larger value of S will also incur a greater computational expense (a larger number of samples will need to be drawn).

Each of the individual samples - $\{\boldsymbol{\eta}_1, \boldsymbol{\eta}_2, \ldots, \boldsymbol{\eta}_S\}$ - are used in automatic differentiation based calculations of the arguments in the expectations $\mathbb{E}_{\eta \sim \mathcal{N}(\mathbf{0},I)}\left[\nabla_{\boldsymbol{\alpha}} f\left(S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta})\right)\right]$. This **MC** integration approach is also used to calculate the remaining non-gradient based expectation term of interest - $\mathbb{E}_{\eta \sim \mathcal{N}(\mathbf{0},I)}\left[f\left(S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta})\right)\right]$.

Elliptical standardisation is not only employed as a means of pushing respective gradients inside their expectations, for ease of their derivations, but is also done as sampling from standard Gaussians $\mathcal{N}(\mathbf{0}, I)$ - the S samples of $\boldsymbol{\eta}$ for equation 1.61 are drawn from $\mathcal{N}(\mathbf{0}, I)$ - is relatively easy and efficient in a computational setting.

Each of the elements required for the **ADVI** algorithm have now been outlined and explained in detail. Tying each of these elements together, as an appropriate algorithmic process, the **ADVI** algorithm can be summarised as follows (Hoffman et al., 2013; Kucukelbir et al., 2017):

Automatic Differentiation Variational Inference (**ADVI**) Algorithm

Input: A model $p(\mathbf{x}, \boldsymbol{\theta})$, a dataset $\mathbf{x}$, and a step size sequence ρ_i. Let the dimension of the latent vector $\boldsymbol{\theta}$ be denoted by m (where $m \in \mathbb{N}$).

Output: The fitted & optimised Variational Density $q(\boldsymbol{\xi}; \boldsymbol{\phi}) = \mathcal{N}(\boldsymbol{\xi}; \boldsymbol{\mu}, \Sigma)$.

Initialisation:

i) Decide whether the variational density $q(\boldsymbol{\xi}; \boldsymbol{\phi})$ is to take the form of a "Mean-Field" Gaussian or a "Full-Rank Gaussian". Then randomly initialise, as appropriate, the variational parameters $\boldsymbol{\phi} = \{\boldsymbol{\mu}, \Sigma\}$ of the variational density q.

ii) Choose a real valued $\epsilon > 0$ that would be considered small enough for convergence to have practically taken place.

iii) Choose an $S \in \mathbb{N}$: The number of samples to consider in each **MC** integration-based approximation of the expectation $\mathbb{E}_{\eta \sim \mathcal{N}(\mathbf{0},I)}$.

Start

Define an appropriate bijective transformation function τ for the latent variables $\boldsymbol{\theta}$, that maps the space of the latent variables to the real space with the same dimension. The transformed, and unconstrained, latent variables are $\boldsymbol{\zeta}$, where $\boldsymbol{\zeta} = \tau(\boldsymbol{\theta})$ and $\boldsymbol{\zeta} \in \mathbb{R}^m$.

Reparameterise the covariance matrix Σ of q. If q takes the form of a "Mean-Field" Gaussian, then set $\boldsymbol{\omega} = \log(\boldsymbol{\sigma})$, where $\boldsymbol{\sigma}$ is a vector of the covariances in the diagonal covariance matrix Σ. If q takes the form of a "Full-Rank" Gaussian, then set $\Sigma = LL^T$, where L is some unconstrained lower-triangular matrix of the same size as Σ.

Define:

i) $\boldsymbol{\eta} = S_{\boldsymbol{\phi}}(\boldsymbol{\zeta}) = L^{-1}(\boldsymbol{\zeta} - \boldsymbol{\mu})$ if q is a "Full-Rank" Gaussian

ii) $\boldsymbol{\eta} = S_{\boldsymbol{\phi}}(\boldsymbol{\zeta}) = diag(\exp(\boldsymbol{\omega}))^{-1}(\boldsymbol{\zeta} - \boldsymbol{\mu})$ if q is a "Mean-Field" Gaussian.

Sample: S values from the m-dimensional standard Gaussian distribution $\mathcal{N}(\mathbf{0}, I)$: $\{\boldsymbol{\eta}_1, \boldsymbol{\eta}_2, \ldots, \boldsymbol{\eta}_S\}$.

The randomly initialised variational parameter values will be $\boldsymbol{\phi}_1 = \{\boldsymbol{\mu}_1, \Sigma_1\}$, or equivalently: $\boldsymbol{\phi}_1 = \{\boldsymbol{\mu}_1, \boldsymbol{\omega}_1\}$ if q takes the form of a "Mean-Field" Gaussian; and $\boldsymbol{\phi}_1 = \{\boldsymbol{\mu}_1, L_1\}$ if q takes the form of a "Full-Rank" Gaussian.

Compute:
$$\begin{aligned}\text{ELBO}_1[q] \approx\ & \tfrac{1}{S}\textstyle\sum_{s=1}^{S}\left(\log\left(p\left(\mathbf{x}|\tau^{-1}\left(S_{\boldsymbol{\phi}_1}^{-1}(\boldsymbol{\eta}_s)\right)\right)p\left(\tau^{-1}\left(S_{\boldsymbol{\phi}_1}^{-1}(\boldsymbol{\eta}_s)\right)\right)\right)\right)\\ & +\tfrac{1}{S}\textstyle\sum_{s=1}^{S}\left(\log\left|\det \mathcal{J}_{\tau^{-1}}\left(S_{\boldsymbol{\phi}_1}^{-1}(\boldsymbol{\eta}_s)\right)\right|\right)\\ & +\tfrac{m}{2}(1+\log(2\pi)) + \tfrac{1}{2}\log|\Sigma_1|\end{aligned} \tag{A.45}$$

```
Set: $i = 1$

While: $|\text{ELBO}_i[q] - \text{ELBO}_{i-1}[q]| \geq \epsilon$ or $i = 1$:
    Resample: $S$ values from the $m$-dimensional standard Gaussian distribution $\mathcal{N}(\mathbf{0}, I)$:
        $\{\boldsymbol{\eta}_1, \boldsymbol{\eta}_2, \ldots, \boldsymbol{\eta}_S\}$.
    Compute: $\nabla_{\boldsymbol{\mu}} ELBO_i[q] \approx \frac{1}{S}\sum_{s=1}^{S}\left(\nabla_{\boldsymbol{\mu}}\log\left(p\left(\mathbf{x}|\tau^{-1}\left(S_{\boldsymbol{\phi}_i}^{-1}(\boldsymbol{\eta}_s)\right)\right)p\left(\tau^{-1}\left(S_{\boldsymbol{\phi}_i}^{-1}(\boldsymbol{\eta}_s)\right)\right)\right)\right)$
                 $+\frac{1}{S}\sum_{s=1}^{S}\left(\nabla_{\boldsymbol{\mu}}\log\left|\det \mathcal{J}_{\tau^{-1}}\left(S_{\boldsymbol{\phi}_i}^{-1}(\boldsymbol{\eta}_s)\right)\right|\right)$    (A.46)
        using automatic differentiation at each of the points $\boldsymbol{\eta}_s$ in the summations.
    Compute: $\boldsymbol{\mu}_{i+1} = \boldsymbol{\mu}_i + \rho_i \cdot \nabla_{\boldsymbol{\mu}} ELBO[q]$

    If $q$ takes the form of a "Mean-Field" Gaussian:
        Compute: $\nabla_{\boldsymbol{\omega}} ELBO_i[q] \approx \frac{1}{S}\sum_{s=1}^{S}\left(\nabla_{\boldsymbol{\omega}}\log\left(p\left(\mathbf{x}|\tau^{-1}\left(S_{\boldsymbol{\phi}_i}^{-1}(\boldsymbol{\eta}_s)\right)\right)p\left(\tau^{-1}\left(S_{\boldsymbol{\phi}_i}^{-1}(\boldsymbol{\eta}_s)\right)\right)\right)\right)$
                 $+\frac{1}{S}\sum_{s=1}^{S}\left(\nabla_{\boldsymbol{\omega}}\log\left|\det \mathcal{J}_{\tau^{-1}}\left(S_{\boldsymbol{\phi}_i}^{-1}(\boldsymbol{\eta}_s)\right)\right|\right)$
                 $+\mathbf{1}$    (A.47)
            using automatic differentiation at each of the points $\boldsymbol{\eta}_s$ in the summations.
        Compute: $\boldsymbol{\omega}_{i+1} = \boldsymbol{\omega}_i + \rho_i \cdot \nabla_{\boldsymbol{\omega}} ELBO[q_{\boldsymbol{\omega}}]$

    If $q$ takes the form of a "Full-Rank" Gaussian:
        Compute: $\nabla_L ELBO_i[q] \approx \frac{1}{S}\sum_{s=1}^{S}\left(\nabla_L\log\left(p\left(\mathbf{x}|\tau^{-1}\left(S_{\boldsymbol{\phi}_i}^{-1}(\boldsymbol{\eta}_s)\right)\right)p\left(\tau^{-1}\left(S_{\boldsymbol{\phi}_i}^{-1}(\boldsymbol{\eta}_s)\right)\right)\right)\right)$
                 $+\frac{1}{S}\sum_{s=1}^{S}\left(\nabla_L\log\left|\det \mathcal{J}_{\tau^{-1}}\left(S_{\boldsymbol{\phi}_i}^{-1}(\boldsymbol{\eta}_s)\right)\right|\right)$
                 $+\left(L^{-1}\right)^T$    (A.48)
            using automatic differentiation at each of the points $\boldsymbol{\eta}_s$ in the summations.
        Compute: $L_{i+1} = L_i + \rho_i \cdot \nabla_L ELBO[q]$
    **Set:** $\boldsymbol{\phi}_{i+1} = \{\boldsymbol{\mu}_{i+1}, \boldsymbol{\omega}_{i+1}\}$, if $q$ takes the form of a "Mean-Field" Gaussian; or
         $\boldsymbol{\phi}_{i+1} = \{\boldsymbol{\mu}_{i+1}, L_{i+1}\}$, if $q$ takes the form of a "Full-Rank" Gaussian.
    **Compute:** $\text{ELBO}_{i+1}[q] \approx \frac{1}{S}\sum_{s=1}^{S}\left(\log\left(p\left(\mathbf{x}|\tau^{-1}\left(S_{\boldsymbol{\phi}_{i+1}}^{-1}(\boldsymbol{\eta}_s)\right)\right)p\left(\tau^{-1}\left(S_{\boldsymbol{\phi}_{i+1}}^{-1}(\boldsymbol{\eta}_s)\right)\right)\right)\right)$
         $+\frac{1}{S}\sum_{s=1}^{S}\left(\log\left|\det\mathcal{J}_{\tau^{-1}}\left(S_{\boldsymbol{\phi}_{i+1}}^{-1}(\boldsymbol{\eta}_s)\right)\right|\right)$
         $+\frac{m}{2}(1+\log(2\pi))+\frac{1}{2}\log|\Sigma_{i+1}|$    (A.45)
    **Set:** $i = i+1$
**End**
```

1.12 Copulas

The final background-based concept to be introduced, that fundamentally underpins the theme and research of this thesis, as indicated by the very title of this thesis, is that of "Copulas". Copulas will be introduced and outlined in this section, to the extent that they will be relevant to the work that will be elaborated on in subsequent chapters.

Copulas are a statistical tool used to model how the individual coordinates, that constitute a given multivariate distribution, are correlated to one another according to that multivariate distribution. In the context of copulas, these individual coordinates are often referred to as marginals - this convention will henceforth be adopted.

A simple, yet substantial, introductory example of a copula is the "bivariate Gaussian copula". Let x_1 and x_2, where $x_1, x_2 \in \mathbb{R}$, denote the two coordinates of a generic bivariate Gaussian density, where the probability distributions of each of its individual coordinates are standard univariate Gaussian densities: I.e., $x_1 \sim \mathcal{N}(0,1)$ and $x_2 \sim \mathcal{N}(0,1)$. Furthermore, let $u_1 = \Phi(x_1; 0, 1)$ and $u_2 = \Phi(x_2; 0, 1)$. The bivariate Gaussian copula density - which is denoted by $c(u_1, u_2; \rho)$ - is then defined such that:

$$\mathcal{N}\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}; \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}\right) = c(u_1, u_2; \rho) \cdot \mathcal{N}(x_1; 0, 1) \cdot \mathcal{N}(x_2; 0, 1) \tag{1.62}$$

In order for a bivariate Gaussian density to have each of its coordinates distributed as $\mathcal{N}(0,1)$ (the density that is observed when the bivariate Gaussian density is projected onto each of its coordinate axes), its mean must be $\begin{pmatrix} 0 \\ 0 \end{pmatrix}$ and its covariance matrix must be $\begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}$, for some $\rho \in (-1, 1)$ (ρ is the correlation between the variables x_1 and x_2): That is, the means and variances of the distributions of each of x_1 and x_2 must be 0 and 1, respectively. Given equation 1.62, it can thus be deduced that:

$$c(u_1, u_2; \rho) = \frac{1}{\sqrt{1-\rho^2}} \exp\left\{-\frac{\rho^2\left(x_1^2 + x_2^2\right) - 2\rho x_1 x_2}{2\left(1-\rho^2\right)}\right\} \tag{1.63}$$

where: $x_1, x_2, u_1, u_2, \rho \in \mathbb{R}$; $x_1 \sim \mathcal{N}(0,1)$; $x_2 \sim \mathcal{N}(0,1)$; $u_1 = \Phi(x_1; 0, 1)$; $u_2 = \Phi(x_1; 0, 1)$; $u_1, u_2 \in (0,1)$; and $\rho \in (-1, 1)$. A full derivation of equation 1.63 is given in appendix section A.13.

The expression of the bivariate Gaussian density in equation 1.62, as a product of a copula density and standard univariate Gaussians, is a particular example of how copulas capture and represent the correlations of the marginals in a multivariate distribution.

It is conventional for the (univariate) distributions of each of the marginals in a copula to be standardised (having a mean of zero and a variance of one), as in the example above. This standardisation is primarily done in aid of mathematical and computational efficiency: Subsequent scaling and shifting based transformations can always be implemented, where appropriate, depending on the context, to account for means and variances that are not standardised. This convention will henceforth be adopted.

Broadly speaking, copulas tend to be primarily represented, summarised and manipulated via their **CDF** forms - denoted by C - as opposed to their density forms - denoted by c (as illustrated above). These two forms are related via the following equation:

$$c(u_1, u_2) = \frac{\partial^2 C(u_1, u_2)}{\partial u_1 \partial u_2} \tag{1.64}$$

For the bivariate Gaussian copula example outlined above, we would have $C(u_1, u_2; \rho) = \int_{u_1} \int_{u_2} c(u_1, u_2; \rho) du_2 du_1 = \Phi_2(\Phi^{-1}(u_1), \Phi^{-1}(u_2); \rho)$, where Φ_2 is the corresponding (joint) **CDF** of the bivariate Gaussian density in equation 1.62.

The individual scalar coordinates of a multivariate Gaussian will necessarily be distributed as univariate Gaussians. However, the converse of this statement is not true: That is, there are various possible multivariate distributions for which the corresponding individual scalar coordinates are distributed as univariate Gaussians. Copulas can be used to differentiate these multivariate distributions, as they each only fundamentally differ in the way that their marginals (individual scalar coordinates) are correlated. As an example, figure 1.7 depicts the contour plots of six common bivariate density functions for which the marginals are distributed as standard Gaussians, together with the corresponding copulas that can be used to identify them (given the correlations of their marginals).

The **CDF** formulae of each of the bivariate copula families/types referenced and illustrated in figure 1.7 are listed in the table in figure 1.8. In the table in figure 1.8, the copulas are parametric bivariate copula families, where ρ, v and δ have been used to denote the respective scalar parameters. The source of figures 1.7 and 1.8 is the website `https://bochang.me/blog/posts/copula/`.

The underlying concepts of the bivariate copula(s) that have so far been introduced can be extrapolated and generalised to their multivariate counterparts: For any distribution, its factorization into a product of marginal densities and a copula of the same dimension always exists and integrates to one (Tran, Blei, and Airoldi, 2015; Nelsen, 2007). However, in the context of **VI**, these multivariate copula extensions have not been found to offer the versatility required to effectively model the dependencies between the appropriate variables - For **VI** augmented with copulas, the latent variables $\mathbf{z}$ are the marginals (Tran, Blei, and Airoldi, 2015; Genest et al., 2009).

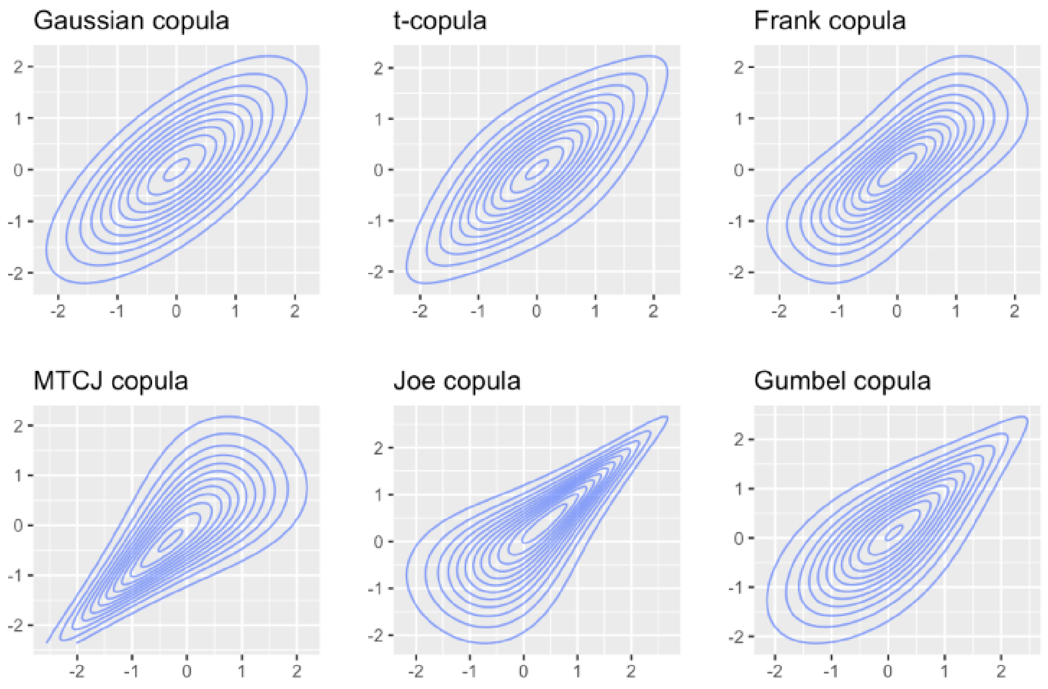


FIGURE 1.7: Contour plots of six common bivariate density functions, each with standard normal marginals, but with the marginals correlated differently

For **VI**, an approach that has been theoretically and empirically shown to be more effective and successful than using a multivariate copula with more than 2 variables, is to use a "Vine" (Tran, Blei, and Airoldi, 2015; Joe, 1996; Kurowicka and Cooke, 2006).

A "vine" is a statistical and mathematical method of factorising a multivariate copula density, with more than 2 variables, into a product of conditional/unconditional bivariate copula densities. In the context of vines, these conditional/unconditional bivariate copula density factors are commonly referred to as pair copulas. If $d \in \mathbb{N}$, for $d > 2$, denotes the number of variables in a given multivariate copula density, then a vine will allow that multivariate copula density to be factorised into a product of $\frac{d(d-1)}{2}$ pair copulas.

There are three main types of vine structures that govern how the original multivariate copula density is factorised:

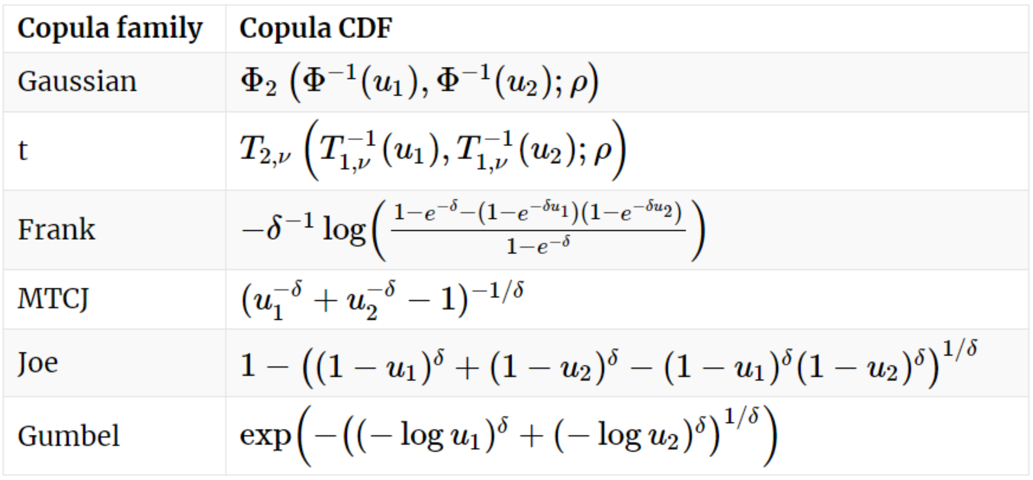
i) The regular vine structure, or "R-vine Structure".

ii) The canonical vine structure, or "C-vine Structure".

iii) The drawable vine structure, or "D-vine Structure".

The C-vine and D-vine structures are actually just particular types of the more generic R-vine structure, so let the general R-vine structure first be considered.

In an R-vine structure, the pair copulas of a multivariate (with more than 2 variables) copula density are formed from the edges of a sequence of mathematical trees. This sequence of trees is generated from an initial tree, the "Proximity Condition", and



Copula family	Copula CDF
Gaussian	$\Phi_2\left(\Phi^{-1}(u_1), \Phi^{-1}(u_2); \rho\right)$
t	$T_{2,\nu}\left(T_{1,\nu}^{-1}(u_1), T_{1,\nu}^{-1}(u_2); \rho\right)$
Frank	$-\delta^{-1}\log\left(\frac{1-e^{-\delta}-(1-e^{-\delta u_1})(1-e^{-\delta u_2})}{1-e^{-\delta}}\right)$
MTCJ	$(u_1^{-\delta}+u_2^{-\delta}-1)^{-1/\delta}$
Joe	$1-\left((1-u_1)^{\delta}+(1-u_2)^{\delta}-(1-u_1)^{\delta}(1-u_2)^{\delta}\right)^{1/\delta}$
Gumbel	$\exp\left(-\left((-\log u_1)^{\delta}+(-\log u_2)^{\delta}\right)^{1/\delta}\right)$

FIGURE 1.8: **CDF** formulae of some common parametric bivariate copulas

a set of rules that govern the formation of subsequent nodes and edges. It is this sequence of trees, depicting the given R-vine structure, that is commonly referred to as the R-vine structure.

The initial tree is simply a mathematical tree for which the nodes are the d ($d \in \mathbb{N}$) marginals of the original unfactorised copula density; and the edges are $d-1$ lines that connect all these nodes together in any configuration: Every node must be connected to at least one other node by an edge, and there must be one less edge than there are nodes.

For a sequence of $n \in \mathbb{N}$ trees that are consecutively labelled T_1, T_2, $\ldots$, T_n (so T_1 is the initial tree), the proximity condition states that if two nodes are joined by an edge in tree T_{j+1}, for some $1 \leq j < n$, where $j \in \mathbb{N}$, then the corresponding edges in tree T_j share a node.

The nodes of the initial tree, T_1, are labelled by the marginals of the multivariate copula density to be factorised. Each edge of the initial tree is labelled by the pair of marginals associated with the two nodes that the edge connects. The edges of tree T_j become the nodes of tree T_{j+1}. In general, an edge that connects two nodes is labelled with all the marginals in those nodes: If a marginal is contained in both nodes, then that marginal is conditional in the edge; if a marginal is conditional in either of the two nodes, then it is also conditional in the edge. The final tree in every R-vine structure is made up of two nodes and an edge connecting them.

Figure 1.9 (Krämer and Schepsmeier, 2011) depicts an example of an R-vine structure with 5 marginals, formulated as per the outline above. For the R-vine structure in figure 1.9:

i) The R-vine structure is intended to find a factorisation of a 5^{th} dimensional (multivariate) copula density, which can be depicted by, say, $c(u_1, u_2, u_3, u_4, u_5)$.

ii) The marginals in the R-vine structure are represented by the numbers in their subscripts (I.e., 1 represents u_1, 2 represents u_2, etc).

iii) There are many possible formulations of the initial tree T_1 that are valid. One of these valid formulations must first be chosen.

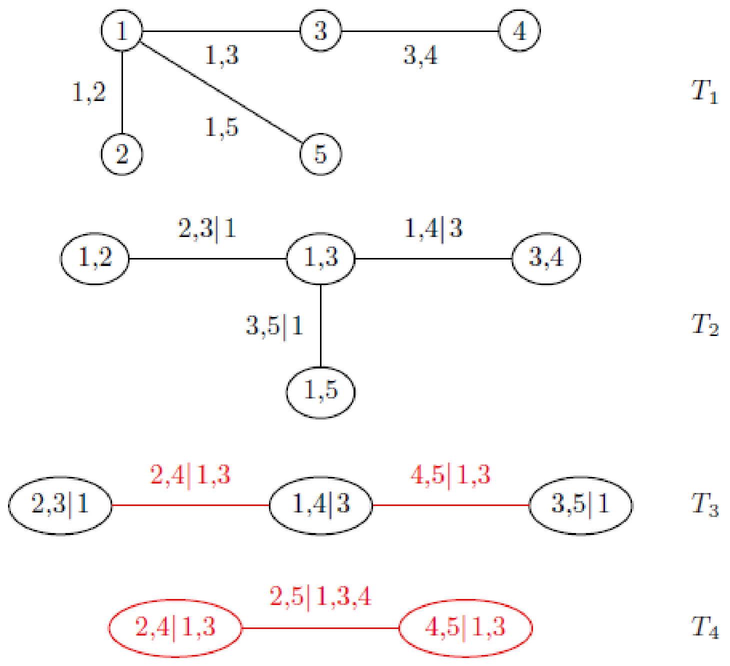


FIGURE 1.9: An example of an R-vine structure with 5 marginals

Given the R-vine structure in figure 1.9, it can thus be deduced that:

$$\begin{aligned} c(u_1, u_2, u_3, u_4, u_5) = & c(u_1, u_2) \cdot c(u_1, u_3) \cdot c(u_1, u_5) \cdot c(u_3, u_4) \\ & \cdot c(u_2, u_3 | u_1) \cdot c(u_1, u_4 | u_3) \cdot c(u_3, u_5 | u_1) \\ & \cdot c(u_2, u_4 | u_1, u_3) \cdot c(u_4, u_5 | u_1, u_3) \\ & \cdot c(u_2, u_5 | u_1, u_3, u_4) \end{aligned} \tag{1.65}$$

The multivariate copula density $c(u_1, u_2, u_3, u_4, u_5)$ has now been factorised into a product of pair copulas, as required.

The C-vine structure is an R-vine structure where all but one of the nodes in tree T_1 are only connected by one edge. The remaining node is connected to all the other nodes by the available edges. The D-vine structure is an R-vine structure where tree T_1 is a path: That is, two nodes in T_1 are only connected by one edge (the end points of the path) and the other nodes are connected by two edges.

Using a vine to express a multivariate copula density as a product of pair copulas allows the multivariate copula density to take the form of a composite copula with a large degree of versatility: By a "composite copula", we are referring to a copula that (partially) belongs to multiple copula families. This composition is achieved by allowing each of the pair copulas in its vine structure decomposition to freely belong to any copula family from an available selection.

Sequential tree selection can be used to find the particular vine structure that is most appropriate for a given **VI** model (Tran, Blei, and Airoldi, 2015; Dissmann et al., 2013); and Bayesian model selection can be used to find the most appropriate copula family that each of the pair copula factors should belong to, from an available selection (Tran, Blei, and Airoldi, 2015; Gruber and Czado, 2015).

As was briefly outlined above, when a **VI** method is said to have been augmented with copulas, the interpretation there, as will henceforth be adopted, is that the surrogate posterior $q(\mathbf{z})$ takes the form:

$$q(\mathbf{z}) = \left[\prod_{i=1}^{d} q_i(z_i)\right] \cdot c\left(Q_1(z_1), Q_2(z_2), \ldots, Q_d(z_d)\right) \tag{1.66}$$

We have $\mathbf{z} = (z_1, z_2, \ldots, z_d)^T$. c is a multivariate copula density of the same dimension as q: I.e., of dimension d (for some $d \in \mathbb{N}$). Each $Q_i(z_i)$, for $i \in \{1, 2, \ldots, d\}$, is the **CDF** form of the density $q_i(z_i)$. The densities $q_i(z_i)$ are univariate densities (see section 1.6).

The expression in square brackets in equation 1.66 is the mean-field approximated form of the surrogate posterior (see section 1.6). The augmentation of the copula $c\left(Q_1(z_1), Q_2(z_2), \ldots, Q_d(z_d)\right)$ to this mean-filed approximation provides a means of effectively dismantling the mean-filed approximation, allowing all dependencies between the latent variables z_i, for $i \in \{1, 2, \ldots, d\}$, to be accounted for. As outlined above, the multivariate copula in equation 1.66 will be factorised into an appropriate product of pair copulas.

2 Thesis Purpose & Method Developed

2.1 Current Methods and the Purpose of this Thesis

One of the crucial elements that underpins the research and purpose of this thesis is an exploration of an augmentation of **MFVI** (see section 1.6) with copulas (see section 1.12). This section outlines the main academic research that has so far been conducted in this regard, and how the results and concepts unveiled in this preceding research lead onto the research in, and purpose of, this thesis.

Arguably the most substantial approach so far taken to augment **MFVI** with copulas is outlined in the academic paper "Copula Variational Inference" (Tran, Blei, and Airoldi, 2015). In this paper, the mean-field approximated surrogate posterior in **MFVI** is multiplied by a multivariate copula density with the same dimension as the latent vector **z**, to form a new surrogate posterior structure (one that no longer assumes independence between the latent coordinates): This copula augmentation/-multiplication is summarised by equation 1.66. The approach and material outlined in the paper "Copula Variational Inference" form part of the basis of the original research that will be subsequently outlined in this thesis. In general, all the core concepts upon which the paper "Copula Variational Inference" builds have already been outlined in chapter 1.

Let the form of the copula augmented surrogate posterior $q(\mathbf{z})$ in the paper "Copula Variational Inference" (Tran, Blei, and Airoldi, 2015) be outlined. This surrogate posterior takes the following form:

$$q(\mathbf{z};\boldsymbol{\lambda},\boldsymbol{\psi}) = \left[\prod_{i=1}^{d} q_i(z_i;\boldsymbol{\lambda}_i)\right] \cdot c\left(Q_1(z_1;\boldsymbol{\lambda}_1), Q_2(z_2;\boldsymbol{\lambda}_2), \ldots, Q_d(z_d;\boldsymbol{\lambda}_d);\boldsymbol{\psi}\right) \tag{2.1}$$

i We have $1 \leq i \leq d$, where $i, d \in \mathbb{N}$. The integer d denotes the dimension of **z**, where $\mathbf{z} = (z_1 \quad z_2 \quad \ldots \quad z_d)^T$ is a vector.

ii The expression in square brackets is the original mean-field approximated surrogate posterior (see section 1.6).

iii The arguments of the augmented copula - $Q_i(z_i)$, for $i \in \{1, 2, \ldots, d\}$ - are the **CDF** counterparts of the probability density factors - $q_i(z_i)$ - in the mean-field component.

iv The sets $\boldsymbol{\lambda}$ and $\boldsymbol{\psi}$ are the variational parameters of the marginal, q_i, and copula, c, densities, respectively.

v We have $\boldsymbol{\lambda} = \{\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2, \ldots, \boldsymbol{\lambda}_d\}$, where each $\boldsymbol{\lambda}_i$, for $i \in \{1, 2, \ldots, d\}$, is the set of variational parameters of the correspondingly labelled univariate probability density $q_i(z_i)$ $(= q_i(z_i; \boldsymbol{\lambda}_i))$.

vi Equation 2.1 takes a very similar form to equation (3) in the paper "Copula Variational Inference" - its counterpart in the aforementioned paper. However, from equation (3) to equation 2.1: $\boldsymbol{\eta}$ has been replaced by $\boldsymbol{\psi}$ to avoid a clash of notation that will subsequently become evident; and each of the densities, q_i, and **CDF**s, Q_i, have been individually labelled with the subscripts that uniquely identify them, for clarity.

As has been briefly outlined previously, this copula augmentation allows all possible dependencies between the latent variables, z_i, for $i \in \{1, 2, \ldots, d\}$ to be accounted for. This dependence accountability approach is a means of addressing one of the most substantial and concerning drawbacks of **MFVI** (see section 1.6): The strong independence assumption of the latent components in $\mathbf{z}$.

This copula augmentation approach to **MFVI** ensures that the resulting optimised surrogate posterior either better or equally approximates the true posterior, compared to an optimised surrogate posterior with just the mean-field structure alone (Tran, Blei, and Airoldi, 2015). However, having to optimise over not just the variational parameters of the mean-field factors, but also over those of the copula density, also means that this copula augmentation approach is more computationally expensive. Nevertheless, initial empirical research into the relative accuracy and computational cost of this copula augmentation of **MFVI**, compared to **MFVI** alone, is yielding promising results (Tran, Blei, and Airoldi, 2015).

The paper "Copula Variational Inference" provides a modified version of the **MFVI**-based **SVI** algorithm (see section 1.9) that not only allows for the required optimisation to be carried out over the variational parameters of the univariate density factors $q_i(z_i)$, but also over the variational parameters of the augmented copula. It is via this means that the paper "Copula Variational Inference" proposes an augmentation of **VI** - specifically **SVI** - with copulas. The general copula **VI** algorithm in the paper "Copula Variational Inference" (Tran, Blei, and Airoldi, 2015) can be outlined as follows:

Copula Variational Inference (Copula **VI**) Algorithm

Input: A model $p(\mathbf{x}, \mathbf{z})$, a dataset $\mathbf{x}$, and a step size sequence ρ_i.

Output: The fitted & optimised Variational Density $q(\mathbf{z})$, which takes the form of equation 2.1.

Initialisation: Randomly initialise the variational parameters in $\boldsymbol{\lambda}$. Set the variational parameters in $\boldsymbol{\psi}$ such that the copula density c is uniform (i.e., $c(\ldots) = 1$). Choose an ϵ that would be considered small enough for convergence to have practically taken place.

Sample: A sufficiently large number of data points from the dataset $\mathbf{x}$ that is to act as a surrogate for subsequent computations involving $\mathbf{x}$ in this algorithm, where required.

```
Factorise the copula density of $q(\mathbf{z})$ into pair copulas using an appropriately chosen
vine structure (see section 1.12).

Start: $t = 1$

While: $\left|\text{ELBO}_{(t)}[q] - \text{ELBO}_{(t-1)}[q]\right| \geq \epsilon$ or $t = 1$:
  Set: $v = 1$
  // Fix $\boldsymbol{\psi}$, maximise over $\boldsymbol{\lambda}$.
  While: $\left|\text{ELBO}_{(v)}[q] - \text{ELBO}_{(v-1)}[q]\right| \geq \epsilon$ or $v = 1$:
    Compute: $\nabla_{\boldsymbol{\lambda}}\text{ELBO}_{(v)}\,[q(\mathbf{z})]$ with the aid of equation 1.16
    Set: $\boldsymbol{\lambda}_{(v+1)} = \boldsymbol{\lambda}_{(v)} + \rho_v \nabla_{\boldsymbol{\lambda}}\text{ELBO}_{(v)}$
    Compute: $\text{ELBO}_{(v+1)}\,[q(\mathbf{z})]$ using the new settings of the variational
      parameters $\boldsymbol{\lambda}_{(v+1)}$ and equation 1.16.
    Set: $v = v + 1$
    Set: $\boldsymbol{\lambda}_{(t+1)} = \boldsymbol{\lambda}_{(v)}$

  Set: $v = 1$
  // Fix $\boldsymbol{\lambda}$, maximise over $\boldsymbol{\psi}$.
  While: $\left|\text{ELBO}_{(v)}[q] - \text{ELBO}_{(v-1)}[q]\right| \geq \epsilon$ or $v = 1$:
    Compute: $\nabla_{\boldsymbol{\psi}}\text{ELBO}_{(v)}\,[q(\mathbf{z})]$ with the aid of equation 1.16
    Set: $\boldsymbol{\psi}_{(v+1)} = \boldsymbol{\psi}_{(v)} + \rho_v \nabla_{\boldsymbol{\psi}}\text{ELBO}_{(v)}$
    Compute: $\text{ELBO}_{(v+1)}\,[q(\mathbf{z})]$ using the new settings of the variational
      parameters $\boldsymbol{\psi}_{(v+1)}$ and equation 1.16.
    Set: $v = v + 1$
    Set: $\boldsymbol{\psi}_{(t+1)} = \boldsymbol{\psi}_{(v)}$

Compute: $\text{ELBO}_{(t+1)}\,[q(\mathbf{z})]$ using the new settings of the variational parameters:
  $\boldsymbol{\lambda}_{(t+1)}$ and $\boldsymbol{\psi}_{(t+1)}$.
Set: $t = t + 1$

End
```

i The variables t and v are iteration counters: I.e., $t, v \in \mathbb{N}$. The variable t denotes the iteration counter of the outer loop, that represents the optimisation process of the algorithm as a whole. The variable v denotes the iteration counter of each of the inner loops, each of which holds one of the variational parameter types fixed, whilst optimising over the other.

ii Double forward slashes have been used to denote comments at key sections of the algorithm, to emphasise how the optimisation process is generally carried out, as was done in the paper "Copula Variational Inference".

iii As regular subscripts have already been used for $\boldsymbol{\lambda}$, to indicate the variational parameters of the individual mean-field factors (i.e., $q_i(z_i; \boldsymbol{\lambda}_i)$, for $i \in \{1, 2, \ldots, d\}$), subscripts with parentheses have been used to denote the values of quantities or of variational parameters in sets at a given iteration. For example: $\text{ELBO}_{(v=2)}[q(\mathbf{z})]$ is the value of the **ELBO** at the second iteration of one of the inner loops; and $\boldsymbol{\lambda}_{(t=1)}$ is the vector set of the randomly initialised values (for the first overall iteration) of the variational parameters in the set $\boldsymbol{\lambda}$.

iv The copula **VI** algorithm outlined above is, in fact, a generalisation of the **SVI** algorithm outlined in section 1.9: The **SVI** algorithm can be considered as a particular sub-branch of the copula **VI** algorithm, in which the algorithm ceases its execution after the first inner while loop (Tran, Blei, and Airoldi, 2015).

v Let it be emphasised that this is just a general algorithm. Precisely how quantities such as $\nabla_{\boldsymbol{\lambda}} \text{ELBO}\left[q(\mathbf{z})\right]$ are calculated depend on the more intricate framework of the precise **SVI**-based algorithm that is implemented (such as **ADVI** - see section 1.11).

To summarise, the copula **VI** algorithm takes the concepts of **MFVI** and **SVI** (see sections 1.6 and 1.9, respectively), and a copula-based structure for the variable surrogate posterior $q(\mathbf{z})$, to carry out the required optimisation in **VI**. The "variable surrogate posterior" structure is a "full-rank" density, made up of a multivariate copula that has been factorised into an equivalent product of pair copulas (via a vine structure), multiplied by the marginal probability densities that, alone, would otherwise be equivalent to an **MFVI** variant of **VI**. The optimisation is carried out over the variational parameters of the marginal factors, and then of the multivariate copula density, in turn. That is, each of these variational parameter types (of the marginals and of the copula density(ies)) are held constant in turn, while optimisation is carried out over the other type.

In the paper "Copula Variational Inference", the precise **SVI**-based algorithm considered is the **BBVI** algorithm (Ranganath, Gerrish, and Blei, 2014). The **BBVI** (black box variational inference) algorithm is a particular **SVI** variant that is typically considered to be the most generic and applicable of the **SVI** algorithms. However, for the class of models for which the more specific **SVI**-based algorithm "automatic differentiation variational inference", or **ADVI** (see section 1.11), can be applied, there are generally clear computational and accuracy advantages to choosing the **ADVI** algorithm over the more generic **BBVI** algorithm (Blei, Kucukelbir, and McAuliffe, 2017; Kucukelbir et al., 2017).

There is no indication in academic material that an **ADVI**-based copula **VI** algorithm has been theoretically established, empirically tested, or considered in general - only for **BBVI**. It is this very avenue of open research on which the original research in this thesis is based, as will now be outlined in the proceeding sections and chapters.

The purpose of this thesis is to outline, in detail, the original research that has been carried out by Daniel William SCOTT, for consideration of an "MSc (by Research) in Statistics" to be awarded to Daniel William SCOTT. The level of detail required must be such that the original research conducted, in its entirety, can be understood by a trained lay-Statistician following their reading of chapter 1. The original research in question will be the theoretical establishment and empirical testing of the copula **VI** algorithm based on **ADVI**, as opposed to the previously considered **BBVI**.

For equation 2.1, when compared to its counterpart in the paper "Copula Variational Inference" - equation (3) - it was noted that $\boldsymbol{\eta}$ had been replaced by $\boldsymbol{\psi}$ to avoid a clash of notation. It is now evident what this clash of notation is: The variable $\boldsymbol{\eta}$ is typically used in **ADVI** (see section 1.11) to denote an alternative concept (a random variable distributed as a standardised Gaussian) to that used in the paper "Copula Variational Inference" for the copula **VI** algorithm (the variational parameters of the multivariate copula density).

2.2 Setup & Approach

This section sets out the approach that was taken to carry out the original research, as well as the prerequisite constraints and conditions that had to be abided by in order for the required objectives to be met. The **ADVI**-based copula **VI** algorithm, that was introduced and briefly referenced in section 2.1, will henceforth be referred to as "the copula augmentation of **ADVI**".

As a means of obtaining an empirical indication as to how the accuracy and computational speed/expense of the copula augmentation of **ADVI** are affected, compared to **ADVI** alone, the common iris dataset (Anderson, 1935) is going to be used as an empirically indicative benchmark. This dataset has been chosen as it is generally considered within the statistical community as one of many standard empirical benchmarks, not just for classification, but for machine learning in general. Furthermore, given the reverence and familiarity of this dataset within the statistical community, the hope is that resulting empirical indications will prompt further research beyond the scope of this MSc thesis.

The iris dataset is a categorical dataset with: 1 categorical target variable; 150 instances; and 4 numerically continuous feature variables on $\mathbb{R}^{++}$. The feature variables are the lengths and widths of the petals and sepals of 150 different iris flowers: 50 iris setosas, 50 iris versicolours, and 50 iris virginicas, as indicated by the categorical target variable.

Recall from section 1.11 that the variables in $\mathbf{z}$, which shall henceforth denote the vector set of latent variables (the vector set of latent variables was denoted by $\boldsymbol{\theta}$ in section 1.11, but its denotation by $\mathbf{z}$ was used more extensively throughout chapter 1), must be continuous in order for **ADVI** to be invoked. However, for the iris dataset, the one target variable is categorical, meaning that its direct inclusion in a Bayesian model $p(\mathbf{x}, \mathbf{z})$ ($\mathbf{x}$ is the dataset) would result in a set $\mathbf{z}$ of discrete latent variables.

A prerequisite for us to be able to invoke **ADVI**, both with and without copula augmentation, given the dataset that we are seeking to invoke the **ADVI** method for (the iris dataset), is to devise a means of modelling the discrete latent variables in $\mathbf{z}$ in a continuous way, whilst ensuring that the gradient $\nabla_{\mathbf{z}} \log p(\mathbf{x}, \mathbf{z})$ is defined (see section 1.11). Such prior $p(\mathbf{z})$ and likelihood $p(\mathbf{x}|\mathbf{z})$ structures of the model $p(\mathbf{x}, \mathbf{z})$ $(= p(\mathbf{x}|\mathbf{z}) * p(\mathbf{z}))$ that allow for a suitable "continuous modelling of the discrete latent variables, that give rise to a defined gradient $\nabla_{\mathbf{z}} \log p(\mathbf{x}, \mathbf{z})$" are outlined in appendix sections B.1 and B.2, respectively.

This "continuous modelling"-based prerequisite, either with or without copula augmentation, is not required for the likes of the **BBVI** or **CAVI** algorithms (see sections 2.1 and 1.7, respectively). If we were to train either of these methods on the iris dataset, we would likely use a discrete, categorical-based model $p(\mathbf{x}, \mathbf{z})$, such as a tailored version of the **GMM** in section 1.8.

The variants of the **ADVI** algorithm that are going to be trained and tested using the iris dataset are:

i **ADVI** with a "mean-field" Gaussian surrogate posterior structure

ii **ADVI** with a "mean-field augmented with copulas" Gaussian surrogate posterior structure

As a means of building upon the mean-field **ADVI** variant that has already been established in preceding research (Kucukelbir et al., 2017), that only considers a Gaussian-based surrogate posterior, only Gaussian copulas will be considered in the original research (so that the surrogate posterior will still be Gaussian-based). Considering an augmentation with other types of copulas could be the subject of future research (see chapter 4).

The most basic of the variants to consider - **ADVI** with a "mean-field" Gaussian surrogate posterior structure - will be used as a benchmark to gauge the accuracy and computational expense trade off of the "mean-field augmented with copulas" Gaussian surrogate posterior structure variant.

Extensive training and testing of the 2 **ADVI** variants outlined above, on the iris dataset, will be carried out using a cross-validation based approach.

2.3 Original Method Developed

The previous two sections in this chapter outlined: the purpose of this thesis; the research objectives of this thesis; the preceding research of other academics, upon which the primary original research in this thesis is inspired and builds; and the originally designed prerequisites needed to carry out the required primary original research. We are now in a position to consider, outline, and elaborate on, the primary original research that has been conducted, as will now be done in this section.

Note that the structures that have been adopted for the prior and the likelihood, as outlined in appendix sections B.1 and B.2, respectively, that are required as prerequisites (see section 2.2), contribute to part of the original research in this thesis. However, these prerequisites are not part of the main subject line of the original research in this thesis (which is why they are considered prerequisites), unlike the original material which shall now be outlined.

Henceforth, the mean-field approach to **ADVI**, as outlined in the paper (Kucukelbir et al., 2017), shall be referred to as "mean-field **ADVI**"; and my original approach to the copula augmentation of this mean-field approach to **ADVI** shall be referred to as "copula **ADVI**".

As briefly outlined in section 2.2, the surrogate posterior in copula **ADVI** shall take the form of a Gaussian, just as for mean-field **ADVI**. However, unlike in mean-field **ADVI**, the surrogate posterior in copula **ADVI** shall take the form of a full-rank Gaussian. Considering the surrogate posterior of copula **ADVI** in the context of equation 2.1, where each of the q_is are its univariate Gaussian components, the multivariate copula, c, must be a Gaussian copula in order for q to take the form of a full-rank Gaussian (also briefly outlined in section 2.2).

In the context of copula **ADVI**, equation 2.1 can thus be rewritten as follows:

$$\begin{aligned} q(\mathbf{z};\boldsymbol{\mu},\Sigma) &= q(\mathbf{z};\boldsymbol{\mu},\boldsymbol{\sigma}^2,\boldsymbol{\rho}) \\ &= \left[\prod_{i=1}^{d}\mathcal{N}_i(z_i;\mu_i,\sigma_i^2)\right]\cdot c\left(\Phi_1(z_1;\mu_1,\sigma_1^2),\Phi_2(z_2;\mu_2,\sigma_2^2),\ldots,\Phi_d(z_d;\mu_d,\sigma_d^2);\boldsymbol{\rho}\right) \end{aligned} \tag{2.2}$$

where $d \in \mathbb{N}$, and:

$$\boldsymbol{\mu} = \begin{pmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_d \end{pmatrix}, \boldsymbol{\sigma} = \begin{pmatrix} \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_d \end{pmatrix}, \boldsymbol{\rho} = \begin{pmatrix} \rho_{12} \\ \rho_{13} \\ \vdots \\ \rho_{23} \\ \vdots \\ \rho_{(d-1)d} \end{pmatrix} \text{ and } \Sigma = \begin{pmatrix} \sigma_1^2 & \rho_{12}\sigma_1\sigma_2 & \dots & \rho_{1d}\sigma_1\sigma_d \\ \rho_{12}\sigma_1\sigma_2 & \sigma_2^2 & \dots & \rho_{2d}\sigma_2\sigma_d \\ \vdots & \vdots & \ddots & \vdots \\ \rho_{1d}\sigma_1\sigma_d & \rho_{2d}\sigma_2\sigma_d & \dots & \sigma_d^2 \end{pmatrix} \tag{2.3}$$

Note that: $\boldsymbol{\mu}$ is a d-dimensional vector; $\boldsymbol{\sigma}$ is a d-dimensional vector; $\boldsymbol{\rho}$ is a $\frac{d(d-1)}{2}$-dimensional vector; and Σ is a $d \times d$ covariance matrix.

In mean-field **ADVI**, we would optimise over the variational parameters in $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$, whilst holding $\boldsymbol{\rho}$ fixed at $\mathbf{0}$ (Tran, Blei, and Airoldi, 2015; Kucukelbir et al., 2017): When $\boldsymbol{\rho} = \mathbf{0}$, then $c = 1$ in equation 2.2. However, in copula **ADVI**, we shall optimise over each of the variational parameters in $\boldsymbol{\mu}$, $\boldsymbol{\sigma}$ and $\boldsymbol{\rho}$.

In the particular case of a full-rank Gaussian expressed as a product of a multivariate Gaussian copula and its univariate Gaussian components, its associated covariance matrix provides access to each of the parameters of the multivariate Gaussian copula (see equation 2.3): Recall that a Gaussian copula is parameterised by all of the Pearson correlations associated with the Gaussian. As a result, we need not factorise the multivariate Gaussian copula into a product of pair copulas, one for each of the variational parameters of the multivariate Gaussian copula, as would need to be done in the general case (see section 1.12).

Let the general form of the **ELBO** for both the mean-field and copula **ADVI** algorithms, in the context of the adopted prior, likelihood (see appendix sections B.1 and B.2, respectively) and surrogate posterior structures, be derived as follows:

$$\begin{aligned} ELBO[q(\mathbf{z})] &= \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}\left[\log\left(p(\mathbf{x}|\tau^{-1}(\mathbf{z}))p(\tau^{-1}(\mathbf{z}))\right) + \log|\det \mathcal{J}_{\tau^{-1}}(\mathbf{z})|\right] \\ &\quad + \mathbb{H}_{\mathbf{z}\sim q(\mathbf{z})}[q(\mathbf{z})] \\ &= \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}\left[\log\left(p(\mathbf{x}|\mathbf{z})p(\mathbf{z})\right) + \log|\det I_d|\right] + \mathbb{H}_{\mathbf{z}\sim q(\mathbf{z})}[q(\mathbf{z})] \\ &= \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}\left[\log p(\mathbf{z}) + \log p(\mathbf{x}|\mathbf{z}) + \log(1)\right] + \mathbb{H}_{\mathbf{z}\sim q(\mathbf{z})}[q(\mathbf{z})] \\ &= \mathbb{E}_{\eta\sim\mathcal{N}(\mathbf{0},I)}\left[\log p(\mathbf{z}) + \log p(\mathbf{x}|\mathbf{z}) + 0\right] + \mathbb{H}_{\mathbf{z}\sim q(\mathbf{z})}[q(\mathbf{z})] \\ &= \mathbb{E}_{\eta\sim\mathcal{N}(\mathbf{0},I)}[\log p(\mathbf{z}) + \log p(\mathbf{x}|\mathbf{z})] + \frac{d}{2}(1 + \log(2\pi)) + \frac{1}{2}\log|\Sigma| \end{aligned} \tag{2.4}$$

i The first equality in derivation 2.4 follows from an application of equation 1.49, but with the unconstrained (and continuous) latent variables vector denoted by $\mathbf{z}$ instead of $\boldsymbol{\xi}$.

ii The adopted prior and likelihood structures assume an initial latent variables vector, $\mathbf{z}$, in the space $\mathbb{R}^d$, for some $d \in \mathbb{N}$, where $\mathbf{z}$ is a d-dimensional vector (see appendix sections B.1 and B.2). That is, this latent variables vector already lives in the unconstrained real space, meaning that the transformation τ in derivation 2.4, which was taken from equation 1.49, would be an identity transformation. Therefore $\tau^{-1}(\mathbf{z}) = \mathbf{z}$ and $\mathcal{J}_{\tau^{-1}}(\mathbf{z}) = I_d$.

iii The third equality in derivation 2.4 follows from the fact that the determinant of an identity matrix is always equal to one. Hence $|\det I_d| = 1$.

iv The fourth equality in derivation 2.4 follows from an application of equation 1.59 (where it has been noted that $f\left(S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta})\right) = f(\boldsymbol{\xi})$ in that equation), but with the unconstrained (and continuous) latent variables vector denoted by $\mathbf{z}$ instead of $\boldsymbol{\xi}$.

v Just as in section 1.11, $\boldsymbol{\eta}$ is a vector of random variables, distributed as a standardised Gaussian of the same dimension.

vi The last equality in derivation 2.4 follows from an application of equation 1.50, where $d \in \mathbb{N}$ is the dimension of the Gaussian q, and Σ is its covariance matrix.

Recall from section 1.11 that, in **ADVI**, we require all variational parameters, over which the optimisation is to be carried out, to live in the unconstrained real space. To ensure that all variational parameters live in the unconstrained real space, appropriate reparameterisations may be required. The variational parameters in $\boldsymbol{\mu}$ already live in the unconstrained real space, and thus do not require any such reparameterisations; and the variational parameters in $\boldsymbol{\sigma}$ can be appropriately reparameterised via the transformation $\omega_i = \log(\sigma_i)$, where ω_i is the unconstrained counterpart of σ_i (see section 1.11).

The variational parameters in $\boldsymbol{\rho}$, being Pearson correlations, live in the real space $[-1, 1]$. Latent variables that have a perfect linear correlation with an existing latent variable in a **VI** model (i.e., the Pearson correlation between such variables is -1 or 1) can be discarded, as such variables can be precisely modelled and accounted for separately, without the need for statistical inference. Therefore, considering equation 2.2, we can further assert that, for each $\rho_i \in \boldsymbol{\rho}$, we have $\rho_i \in (-1, 1)$. Therefore, the variational parameters in $\boldsymbol{\rho}$ can be appropriately reparameterised via transformation 2.5, where α_i is the unconstrained counterpart of ρ_i (i.e., $\alpha_i \in \mathbb{R}$).

$$\alpha_i = \tan\left(\frac{\pi}{2}\rho_i\right) \tag{2.5}$$

Note that, although the surrogate posterior in copula **ADVI** takes the form of a full-rank Gaussian, just as for the Cholesky-based full-rank **ADVI** variant outlined in section 1.11, the differing approaches of these two **ADVI** variants can now be better appreciated. In the "full-rank case" in section 1.11 (Kucukelbir et al., 2017), the covariance matrix of the surrogate posterior was factorised via a Cholesky factorisation, to obtain unconstrained Cholesky-based variational parameters. In copula **ADVI**, the variances and Pearson correlations (the variational parameters of the Gaussian copula(s)) of the covariance matrix of the surrogate posterior are considered separately, each reparameterised to ensure that they are unconstrained as required.

The gradient ascent steps, in the **ADVI** optimisation, for each of the variational parameter types - $\boldsymbol{\mu}$, $\boldsymbol{\sigma}$ and $\boldsymbol{\rho}$ - given equation 1.48, can thus be summarised as follows:

$$\begin{aligned}
\boldsymbol{\mu}_{i+1} &= \boldsymbol{\mu}_i + \delta_i \nabla_{\boldsymbol{\mu}} ELBO[q(\mathbf{z})] \\
\boldsymbol{\omega}_{i+1} &= \boldsymbol{\omega}_i + \delta_i \nabla_{\boldsymbol{\omega}} ELBO[q(\mathbf{z})] \\
\boldsymbol{\alpha}_{i+1} &= \boldsymbol{\alpha}_i + \delta_i \nabla_{\boldsymbol{\alpha}} ELBO[q(\mathbf{z})]
\end{aligned} \tag{2.6}$$

(to avoid a conflict of notation, the step size sequence is now denoted by δ_i, as opposed to ρ_i as was done previously)

Equivalent expressions for each of the gradient ascent quantities in equations 2.6, that will allow for their values to be calculated using automatic differentiation, given settings of the variational parameters in $\boldsymbol{\mu}$, $\boldsymbol{\omega}$ and $\boldsymbol{\alpha}$, and given a sample of $\boldsymbol{\eta}$ (distributed as $\mathcal{N}(\mathbf{0}, I)$), are derived in appendix section B.3 and summarised as follows:

$$\begin{aligned}
\nabla_{\boldsymbol{\mu}} ELBO[q(\mathbf{z})] &= \mathbb{E}_{\boldsymbol{\eta} \sim \mathcal{N}(\mathbf{0},I)} \left[\nabla_{\mathbf{z}} \log p(\mathbf{z}) + \nabla_{\mathbf{z}} \log p(\mathbf{x}|\mathbf{z})\right] \\
\nabla_{\boldsymbol{\omega}} ELBO[q(\mathbf{z})] &= \mathbb{E}_{\boldsymbol{\eta} \sim \mathcal{N}(\mathbf{0},I)} \left[(\nabla_{\mathbf{z}} \log p(\mathbf{z}) + \nabla_{\mathbf{z}} \log p(\mathbf{x}|\mathbf{z})) \cdot \nabla_{\boldsymbol{\omega}} \left[\Sigma^{\frac{1}{2}} \boldsymbol{\eta}\right]\right] + \frac{\nabla_{\boldsymbol{\omega}} |\Sigma|}{2\,|\Sigma|} \\
\nabla_{\boldsymbol{\alpha}} ELBO[q(\mathbf{z})] &= \mathbb{E}_{\boldsymbol{\eta} \sim \mathcal{N}(\mathbf{0},I)} \left[(\nabla_{\mathbf{z}} \log p(\mathbf{z}) + \nabla_{\mathbf{z}} \log p(\mathbf{x}|\mathbf{z})) \cdot \nabla_{\boldsymbol{\alpha}} \left[\Sigma^{\frac{1}{2}} \boldsymbol{\eta}\right]\right] + \frac{\nabla_{\boldsymbol{\alpha}} |\Sigma|}{2\,|\Sigma|}
\end{aligned} \tag{2.7}$$

A full and concise outline of my originally designed "copula **ADVI**" algorithm can be summarised as follows:

Copula Automatic Differentiation Variational Inference (Copula **ADVI**) Algorithm

---

Input: The model $p(\mathbf{x}, \mathbf{z}) = p(\mathbf{z}) * p(\mathbf{x}|\mathbf{z})$, where $p(\mathbf{z})$ and $p(\mathbf{x}|\mathbf{z})$ take the forms outlined in appendix sections B.1 and B.2, respectively; a dataset $\mathbf{x}$; and a step size sequence δ_i.

Output: The fitted & optimised Variational Density $q(\mathbf{z})$, which takes the form of equation 2.2.

Initialisation: Choose an ϵ that would be considered small enough for convergence to have practically taken place; a suitable sample size for $\boldsymbol{\eta}$, for the **MC** integration steps; and settings of the hyperparameters ω_T, ω_F and s, for $p(\mathbf{z})$ (see appendix section B.1). Set up (or fit) the parameter functions $\boldsymbol{\mu}(\mathbf{z})$ and $\Sigma(\mathbf{z})$ for the likelihood $p(\mathbf{x}|\mathbf{z})$, ideally using an initial training dataset (see appendix section B.2).

Sample: Our chosen number of $\boldsymbol{\eta}$ values randomly from the distribution $\mathcal{N}(\mathbf{0}, I)$ (each of $\boldsymbol{\eta}$ and $\mathcal{N}(\mathbf{0}, I)$ have the same dimension as $\mathbf{z}$).

```
Start
For: "Class" in $\{(\omega_T \ \ \omega_F \ \ \dots \ \ \omega_F)^T, (\omega_F \ \ \omega_T \ \ \dots \ \ \omega_F)^T, \dots, (\omega_F \ \ \omega_F \ \ \dots \ \ \omega_T)^T\}$:
    Set: $i = 1$
    Set: $\boldsymbol{\mu}_1 = "Class", \boldsymbol{\omega}_1 = \mathbf{0}$ and $\boldsymbol{\alpha}_1 = \mathbf{0}$
    Compute: $\text{ELBO}_1[q(\mathbf{z})]$ using equations and derivations 2.3, 2.4 and 2.5.
    While: $|\text{ELBO}_i[q(\mathbf{z})] - \text{ELBO}_{i-1}[q(\mathbf{z})]| \geq \epsilon$ or $i = 1$:
        Compute: $\nabla_{\boldsymbol{\mu}}\text{ELBO}_i[q(\mathbf{z})]$, $\nabla_{\boldsymbol{\omega}}\text{ELBO}_i[q(\mathbf{z})]$ and $\nabla_{\boldsymbol{\alpha}}\text{ELBO}_i[q(\mathbf{z})]$ with the aid
            of equations 2.7.
        Set: $\boldsymbol{\mu}_{i+1} = \boldsymbol{\mu}_i + \delta_i \nabla_{\boldsymbol{\mu}}\text{ELBO}_i[q(\mathbf{z})]$
        Set: $\boldsymbol{\omega}_{i+1} = \boldsymbol{\omega}_i + \delta_i \nabla_{\boldsymbol{\omega}}\text{ELBO}_i[q(\mathbf{z})]$
        Set: $\boldsymbol{\alpha}_{i+1} = \boldsymbol{\alpha}_i + \delta_i \nabla_{\boldsymbol{\alpha}}\text{ELBO}_i[q(\mathbf{z})]$
        **Compute:** $\text{ELBO}_{i+1}\left[q(\mathbf{z})\right]$ using the new settings of the variational parameters
          $\boldsymbol{\mu}_{i+1}$, $\boldsymbol{\omega}_{i+1}$ and $\boldsymbol{\alpha}_{i+1}$.
        **Set:** $i = i + 1$
    **Set:** $\text{ELBO}_{("Class")}\left[q(\mathbf{z})\right] = \text{ELBO}_{i}\left[q(\mathbf{z})\right]$
**Prediction:** The "Class" with the largest value of $\text{ELBO}_{("Class")}\left[q(\mathbf{z})\right]$.
**End**
```

Note that the copula **ADVI** algorithm outlined immediately above is fundamentally different in structure to the copula **VI** algorithm outlined in the paper (Tran, Blei, and Airoldi, 2015), as summarised in section 2.1. In particular; in the "copula **VI** algorithm", the optimisation is first carried out over the non-copula variational parameters, and then over the copula variational parameters: However, in the "copula **ADVI** algorithm", the optimisation is carried out over both the copula and non-copula variational parameters at the same time.

Just as for the copula **VI** algorithm in the paper (Tran, Blei, and Airoldi, 2015), the copula augmentation of the underlying mean-field approximation in the copula **ADVI** algorithm can be effectively cancelled out by fixing/bypassing certain setting and steps in the algorithm, such that the multivariate copula of the surrogate posterior (see equation 2.2) is always equal to one.

The copula **ADVI** algorithm can thus be viewed as a generalisation of a variant of the mean-field **ADVI** algorithm in the paper (Kucukelbir et al., 2017) (see section 1.11), where such a mean-field **ADVI** algorithm is the special case of fixing $\boldsymbol{\alpha} = \mathbf{0}$ (i.e., by bypassing "**Set:** $\boldsymbol{\alpha}_{i+1} = \boldsymbol{\alpha}_i + \delta_i \nabla_{\boldsymbol{\alpha}} \text{ELBO}_i\left[q(\mathbf{z})\right]$") in the copula **ADVI** algorithm above.

Furthermore note that, just as for the adopted prior and likelihood structures (see appendix sections B.1 and B.2, respectively), the copula **ADVI** algorithm outlined above has been designed specifically for categorisation (suitable for the likes of the iris dataset). This is most evident in the copula **ADVI** algorithm outlined above via the introduction of the novel "repeated optimisation for the same data point, **x**, for each of its possible classes". The optimisation is not performed once per data point, as is typical in **VI**, but instead "once per class, per data point". The class predicted for the data point **x** is the one that results in the largest value of the maximised **ELBO**.

3 Experimental Testing, Results & Findings

This chapter starts by summarising the means by which the copula **ADVI** algorithm outlined in section 2.3 was set up, executed and tested in a practical context. We will then outline how such setting up, execution and testing can also be done for the mean-field **ADVI** algorithm (see section 2.2) in a way that allows for the empirical results of these two algorithms to be directly compared. This chapter will then finish with a summary of the results of this testing, together with an analysis and interpretation of these results.

3.1 The Empirical & Practical Setup

Recall from section 2.2 that the iris dataset is to be used as the empirically indicative benchmark in our testing of the copula **ADVI** algorithm. A Python script that will allow for an execution of the copula **ADVI** algorithm in section 2.3, on the iris dataset, for our testing purposes, is provided in appendix section B.4.

The chosen settings of the hyperparameters and numerical constants for the copula **ADVI** algorithm, as per section 2.3, were:

i $s = 1$

ii $\omega_T = 5$

iii $\omega_F = -5$

iv $\epsilon = 0.01$

v "sample size for $\boldsymbol{\eta}$" $= 100$

In general, the best choice of hyperparameters and numerical constants is largely and in part: Dependent on the context; dependent on the available computational power and available resources; and an ambiguous one, open to interpretation. The settings $s = 1$, $\omega_T = 5$ and $\omega_F = -5$ were chosen, as this set up means the prior has regions of very distinct probability density (the difference between the Gaussian centres, around one-hot vectors based on ω_T and ω_F, is substantially larger than their standard deviations, based on s), as required (see figure B.1 and appendix section B.1). The settings $\epsilon = 0.01$, and "sample size for $\boldsymbol{\eta}$" $= 100$ were chosen, as they were found to allow the copula **ADVI** algorithm to function very well without taking very extreme values (ϵ being very small and the sample size being very large).

The iris dataset is available through the "sklearn" (scikit-learn) package in Python, and it is via this means that the iris dataset was obtained and manipulated in the main copula **ADVI** Python script in appendix section B.4.

Google Colab, a server-based **IDE** compatible with Python, was the **IDE** used for all testing-based executions of Python scripts in this thesis. Google Colab offers greater time consistently in the execution of scripts of a given computational expense than other **IDE**s, as executions are performed (remotely) on a Google server, as opposed to on my local machine (at any given moment, other programs on my local machine could slow down the execution of locally executed Python scripts). It is for this reason that Google Colab was chosen for testing, as it is the time taken to execute the lines of code between the Python variables $t1$ and $t2$ near the bottom of the Python script in appendix section B.4 that is used to gauge the underlying computational expense of their execution.

Recall from the last paragraph of section 2.2 that a cross-validation based approach is to be used to carry out the required testing. Following preliminary testing of variants of the Python script in appendix section B.4, it was decided that a 2-fold cross-validation based approach would be used for testing. As both the copula **ADVI** and mean-field **ADVI** variants, on the iris dataset, for varying cross-validation folds, were found to make correct predictions to a very high degree of accuracy, the smallest possible cross-validation fold (a fold of 2) was chosen to be used to more greatly draw out the inaccuracies in both the copula and mean-field variants of the **ADVI** algorithm (particularly for comparative testing purposes).

The Python package used to practically carry out the required automatic differentiation steps, for both the copula and mean-field variants of the **ADVI** algorithm, is the "autograd" package `https://github.com/HIPS/autograd` (see appendix section B.4). In particular, the "jacobian" method in the "autograd" package was used to calculate the derivative of a given function with respect the input vector of that function (see examples of the application of this method in appendix section B.4).

In theory, automatic differentiation should allow for the derivatives of functions to be calculated with a constant time complexity (see section 1.10). In practice, it was found that the manually calculated (and thus directly programmed) derivatives of functions execute significantly faster than their automatic differentiation counterparts, that were automatically calculated using the "autograd" package. Therefore, as it was still practically possible, I manually calculated the derivatives, with respect to the latent vector, of the logs of the prior and likelihood, and included them in the main copula **ADVI** Python script (see appendix section B.4). Appendix section B.5 provides manual derivations of the required derivatives, of the logs of the prior and likelihood: That is, of $\nabla_{\mathbf{z}} \log p(\mathbf{z})$ and $\nabla_{\mathbf{z}} \log p(\mathbf{x}|\mathbf{z})$. The significant work required to manually calculate just $\nabla_{\mathbf{z}} \log p(\mathbf{z})$ and $\nabla_{\mathbf{z}} \log p(\mathbf{x}|\mathbf{z})$, as outlined in appendix section B.5, is a prime example of the computational benefit provided by automatic differentiation.

In the main copula **ADVI** Python script (see appendix section B.4), a limit to the number of iterations of each gradient ascent step was set, as a means of avoiding potential case scenarios in which the **ELBO** does not converge, or does not converge sufficiently quickly, given the set convergence measure of ϵ. An iteration limit of 100 was set, meaning that for a prediction of the type of any given iris flower, $\mathbf{x}$, no more than 300 gradient ascent steps will be executed (as there are 3 possible classes of iris flower). However, during testing, no set of gradient ascent steps ever reached this iteration limit of 100 (see appendix section B.6 for an indication of this).

A relatively simple 'exponentially decreasing' step size sequence, δ_i, was chosen for our practical applications and testing of the copula and mean-field variants of the

ADVI algorithm. Our chosen step size sequence is given by equation 3.1: δ_i is the step size at iteration i; L is the iteration limit; δ_f is the first step size (when $i = 1$); and δ_l is the step size should $i = L$.

$$\delta_i = \delta_f * \left(\frac{\delta_l}{\delta_f}\right)^{\frac{i-1}{L-1}} \tag{3.1}$$

As δ_i is an 'exponentially decreasing' step size sequence, we must have $\delta_f > \delta_l$. As outlined in the Python script in appendix section B.4, our chosen settings of the step size based constants are: $\delta_f = 0.01$, $\delta_l = 0.001$ and $L = 100$. This step size sequence was chosen as it is one that is relatively simple, yet highly effective, in allowing the **ELBO** to converge as intended.

The printed output of running the main copula **ADVI** Python script outlined in appendix section B.4 is 150 lines of text of the form "18 Setosa 137 207.80917624700123".

i The first number is the index of an iris flower instance in the iris dataset, whose feature variables values, $\mathbf{x}$, were tested in the algorithm.

ii The first number is followed by either "Setosa", "Versicolour" or "Virginica", which is the prediction of the algorithm, given an execution of the algorithm about $\mathbf{x}$, for the instance whose index is denoted by the first number.

iii The second number denotes the total number of iterations of the gradient ascent step (the limit has been set to 300) required for the prediction to be made.

iv The final number is a measure of the total computational time taken for an execution of all the gradient ascent steps, before the resulting prediction can be made.

v The 150 lines of text correspond to all of the indices, from 0 to 149, inclusive, in the iris dataset. As a cross-validation approach has been used for testing, each of the 150 instances in the iris dataset will be used exactly once for testing.

For example, an output of "18 Setosa 137 207.80917624700123" means that the iris flower instance with an index of 18 was tested, and the algorithm predicted that this was a setosa iris flower: 137 gradient ascent iterations were required, and it took a relative computational time of approximately 207.81, for this prediction to be made. Recall that the iris flower instances with indices: from 0 to 49 are setosas, from 50 to 99 are versicolours, and from 100 to 149 are virginicas. Therefore, in this case, the algorithm correctly predicted that this iris flower instance is a setosa.

As briefly outlined in section 2.3: Copula **ADVI** is the algorithm used when the line of code "alpha = alpha + (delta * grad_alpha_ELBO())" is included in the Python script in appendix section B.4; and a variant of mean-field **ADVI**, with a structure comparable to that of copula **ADVI**, is the algorithm used when the same line of code "alpha = alpha + (delta * grad_alpha_ELBO())" is excluded.

The fact that the structure of this variant of mean-field **ADVI** is comparable to that of copula **ADVI**, means that an inclusion or exclusion of the line of code "alpha = alpha + (delta * grad_alpha_ELBO())" can be used to best gauge the accuracy and computational expense trade off of the copula augmentation itself. In **ADVI**, the gradient ascent steps are what are generally used to gauge the overall computational expense of the algorithm. For this reason, and to avoid, as far as practically possible, the computational caveats of a high-level programming language such as Python from providing false indications as to the computational expense of the algorithm itself (for example, consider how a directly coded derivative executes faster than one calculated using the "autograd" package), only the time taken to execute the gradient ascent steps will be used to gauge the computational expense of the algorithm (refer to the Python variables $t1$ and $t2$ near the bottom of the Python script in appendix section B.4).

3.2 Statistical Testing of the Method Developed

Using the Python-based setup outlined in section 3.1, the copula **ADVI** algorithm and its structurally analogous mean-field **ADVI** variant were each executed twice using 2-fold cross-validation. The results of these executions are summarised in appendix section B.6. Each of these algorithms were executed twice in order to provide a more substantial set of experimental results, and to ensure that repeated testing has been carried out for every instance in the iris dataset, for each algorithm, to help better identify potential anomalies or outliers.

As is considered conventional among many statisticians, any values within a numerical sample that have z-scores outside the range $(-3, 3)$ will be considered as outliers, and effectively disregarded. By this measure, the numerical samples in the experimental results (see appendix section B.6) possess the following outliers:

i For the sample of "Number of Gradient Ascent Iterations Required", for the copula **ADVI** algorithm, the value of 260, with a z-score of approximately 3.01, is an outlier.

ii For the sample of "Relative Execution Time", for the copula **ADVI** algorithm, the values of 601.0164018429994 and 621.7980240750176, with z-scores of approximately 3.07 and 3.36, respectively, are outliers.

iii For the sample of "Relative Execution Time", for the mean-field **ADVI** algorithm, the values of 343.98187364300054 and 372.4262367520114, with z-scores of approximately 3.01 and 3.69, respectively, are outliers.

We have relatively large samples (generally speaking, a 'large' sample is one with more than 30 instances) for both the copula and mean-field **ADVI** algorithms, in the experimental results. Therefore, for the numerical samples of "Number of Gradient Ascent Iterations Required" and "Relative Execution Time", as we do not know the standard deviations of any of the corresponding 4 populations (of these two numerical samples, and of these two algorithms), the Student's t-test can be used as a suitable statistical significance test for these sample types, as appropriate. As the populations of possible test results of the copula and mean-field **ADVI** algorithms effectively have infinite numbers of members (there are effectively an infinite number of combinations of possible samples for the **MC** integration steps, and an infinite number of possible settings of the hyperparameters, etc), a binomial test can be used as a suitable statistical significance test for the proportion of inaccurate predictions between these two algorithms.

Let the p-values, that can be used to gauge statistical significance, of the numerical samples of each of "Number of Gradient Ascent Iterations Required" and "Relative Execution Time", and of the proportion of inaccurate predictions, between the copula and mean-field **ADVI** algorithms in the experimental results (see appendix section B.6), for the respective significance tests, now be derived.

In general, the t-value for a Student's t-test is calculated using the following equation:

$$t = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}} \tag{3.2}$$

where: $\bar{x}_1$, s_1 and n_1 are the mean, standard deviation and size, respectively, of one of the samples. Likewise, $\bar{x}_2$, s_2 and n_2 are the mean, standard deviation and size, respectively, of the other sample. Note that the labeling of the two samples must be such that $\bar{x}_1 \geq \bar{x}_2$.

Given the numerical samples in appendix section B.6 and the outliers identified above, the following (rounded) key statistics can thus be inferred:

Numerical Sample	Mean, $\bar{x}$	Std, s	Size, n
Copula, Iterations	164.91	31.04	299
Copula, Time	375.23	70.64	298
MF, Iterations	162.65	29.93	300
MF, Time	216.05	40.66	298

where "Copula", "MF", "Iterations" and "Time" are short-hand labels for 'the copula **ADVI** algorithm', 'the mean-field **ADVI** algorithm', 'the numerical sample type "Number of Gradient Ascent Iterations Required"' and 'the numerical sample type "Relative Execution Time"', respectively. For example, "Copula, Iterations" refers to the numerical sample for the copula **ADVI** algorithm that is a measure of the number of gradient ascent iterations required, with a first value of 135 and a final value of 179 (see appendix section B.6).

Given the unrounded counterparts of the values in the table above, and equation 3.2, the t-values of the "Iterations" and "Time" numerical sample types, each between the copula and mean-field **ADVI** algorithms, are 0.91 and 33.71 (each rounded), respectively.

Copula-based **VI**, in general, is already understood to be computationally more expensive than the mean-field based counterparts (Tran, Blei, and Airoldi, 2015), as is implied by the $\bar{x}$ values of "Copula, Time" and "MF, Time". Therefore, a one-tail test will be used for the "Time" numerical sample type. However, it is not known whether the number of gradient ascent iterations required will generally be larger for either copula-based or mean-field based **VI** algorithms. Therefore, a two-tail test will be used for the "Iterations" numerical sample type.

For a one-tail Student's t-test, with 594 degrees of freedom ($df = n_1 + n_2 - 2$) and a t-value of 33.71 (or rather, its unrounded counterpart, as outlined above), the corresponding p-value is 3.58×10^{-12} (to 3 significant figures). For a two-tail Student's t-test, with 597 degrees of freedom and a t-value of 0.91 (or rather, its unrounded counterpart, as outlined above), the corresponding p-value is 0.365 (to 3 significant figures).

Now let the proportions of inaccurate predictions be considered. Recall from section 3.1 that the instances with indices: from 0 to 49 are setosas; from 50 to 99 are versicolours; and from 100 to 149 are virginicas. The copula and mean-field **ADVI** algorithms incorrectly predicted 4 and 11, respectively, out of their 300 tested instances. Therefore, given the binomial distribution, we have:

$$
\begin{aligned}
p\left(x \leq 4 | p = \frac{11}{300}\right) &= \sum_{i=1}^{4} C_i^{300} \left(\frac{11}{300}\right)^i \left(\frac{289}{300}\right)^{300-i} \\
&= 0.0138 \text{ (to 3 significant figures)}
\end{aligned}
\tag{3.3}
$$

and

$$
\begin{aligned}
p\left(x \geq 11 | p = \frac{4}{300}\right) &= 1 - \sum_{i=1}^{10} C_i^{300} \left(\frac{4}{300}\right)^i \left(\frac{296}{300}\right)^{300-i} \\
&= 0.0205 \text{ (to 3 significant figures)}
\end{aligned}
\tag{3.4}
$$

3.3 Interpretations & Conclusions

The p-values of the sample types between the copula and mean-field **ADVI** algorithms, as established and derived in section 3.2, can be summarised as follows:

i For the numerical sample type "Number of Gradient Ascent Iterations Required" in appendix section B.6, the associated p-value is 0.365 (to 3 significant figures).

ii For the numerical sample type "Relative Execution Time" in appendix section B.6, the associated p-value is 3.58×10^{-12} (to 3 significant figures).

iii For the proportion of inaccurate predictions of the copula **ADVI** algorithm, assuming the results of the mean-field **ADVI** algorithm, the associated p-value is 0.0138 (to 3 significant figures).

iv For the proportion of inaccurate predictions of the mean-field **ADVI** algorithm, assuming the results of the copula **ADVI** algorithm, the associated p-value is 0.0205 (to 3 significant figures).

Given the associated p-values outlined above, and assuming a common α level of, say, 0.05, the following conclusions can thus be made:

i There is insufficient evidence to suggest a statistically significant difference in the "Number of Gradient Ascent Iterations Required" between the copula and mean-field **ADVI** algorithms.

ii There is very strong evidence to suggest a statistically significant difference in the "Relative Execution Time" between the copula and mean-field **ADVI** algorithms. The copula **ADVI** algorithm is more computationally expensive than the mean-field **ADVI** algorithm.

iii There is reasonable evidence to suggest a statistically significant difference in the proportion of inaccurate predictions between the copula and mean-field **ADVI** algorithms. The copula **ADVI** algorithm is more accurate than the mean-field **ADVI** algorithm.

The fact that the p-value associated with the "Number of Gradient Ascent Iterations Required" (i.e., 0.365) is relative large suggests that there is, in fact, strong evidence that the "Number of Gradient Ascent Iterations Required" for the copula and mean-field **ADVI** algorithm are effectively the same, given our setup. The augmentation of our chosen variant of the mean-field **ADVI** algorithm with copulas seemingly has little to no influence over the number of gradient ascent iterations required for the underlying **ELBO** to converge to a suitable local maximum.

Note that the p-value associated with the "Relative Execution Time" is smaller than all the other p-values considered by several orders of magnitude. This highlights the significant extent to which the experimental results in section 3.2 and appendix section B.6 imply that the copula **ADVI** algorithm is more computationally expensive than its structurally analogous mean-field counterpart. The p-values associated with the proportions of inaccurate predictions of the algorithms (i.e., 0.0138 and 0.0205) are relatively small, and certainly below the common α level adopted, but are not nearly as small as the aforementioned p-value. Such results imply that any small relative gains to be made in the accuracy, through an augmentation of the mean-field **ADVI** algorithm with copulas, could be met with a relatively large increase in the required computational expense.

Recall from section 1.4 that other commonly used Bayesian-compatible approximation algorithms, such as **MCMC**, that can be used to carry out the same objective as **VI** in general, tend to be more accurate than **VI**, but are also more computationally expensive. Likewise, the copula **ADVI** algorithm has been shown to be more accurate than its mean-field counterpart, when both are used to carry out the same objective, but is also more computationally expensive. Although it is known, to a certain extent, how mean-field **ADVI** compares in accuracy and computational expense to other commonly used Bayesian-compatible approximation algorithms, such as **MCMC**, it is still not known how copula **ADVI** compares to these other algorithms. For example, could it be that copula **ADVI** ranks in between both mean-field **ADVI** and **MCMC** when considering both accuracy and computational expense in turn? If so, or otherwise, how does the accuracy and computational expense of copula **ADVI** then compare to mean-field **ADVI** and **MCMC**? Such limitations and avenues for future research will be outlined and explored in more detail in chapter 4.

To summarise, based on the analysis of the experimental results, as outlined in section 3.2:

i There is strong evidence to suggest that the number of gradient ascent iterations required for a generic instance, for which we seek a prediction to be made, is not influenced by our choice of algorithm: Either the copula **ADVI** algorithm or the mean-field **ADVI** algorithm.

ii There is strong evidence to suggest that the copula **ADVI** algorithm is more accurate than the mean-field **ADVI** algorithm.

iii There is very strong evidence to suggest that the copula **ADVI** algorithm is more computationally expensive than the mean-field **ADVI** algorithm.

iv There is reasonable evidence to suggest that the copula **ADVI** algorithm will offer small to moderate improvements in the predictive accuracy, at the expense of a relatively large increase in the computational cost required for the algorithm to execute, when compared to the mean-field **ADVI** algorithm.

v Both copula **ADVI** and **MCMC** (as well as other commonly used Bayesian-compatible approximation algorithms that can be used to carry out the same objective) are more accurate and more computationally expensive than the mean-field **ADVI** algorithm. However, it is not known how copula **ADVI** and **MCMC** (and the other such commonly used algorithms) compare to each other in these regards.

vi Note that the experimental testing, analysis, interpretations and conclusions outlined above are only in the context of the iris dataset and the "continuous categorical model" adopted, as outlined in appendix sections B.1 and B.2 (the model is $p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x}|\mathbf{z}) * p(\mathbf{z})$).

vii It is suspected that the interpretations and conclusions outlined above extend beyond the scope of the iris dataset and the "continuous categorical model", although further empirical testing and theoretical analysis would need to be done for this statement to be asserted with sufficient confidence.

3.4 Applying Copula ADVI on the MNIST Dataset

Following my completion of the original research development, testing and interpretations outlined in the preceding chapters and sections, I proceeded to further extend and test my algorithm in the context of a more complex dataset. Until now, all empirical testing and implementation has been done in the context of the common iris dataset. However, this is a small and relatively simple dataset that, to a certain degree, is limited in its provisions of some of the more complex aspects of other real world datasets.

The more complex dataset in question, against which my algorithm can be further implemented, assessed and evaluated, is the common **MNIST** dataset (LeCun et al., 1998). The **MNIST** dataset package consists of two elements: A training dataset and a testing dataset. These two datasets each consist of 784 feature variables, with each feature variable corresponding to each pixel in a 28x28 image grid (note that $28 * 28 = 784$), consecutively from left to right, and then from top to bottom (in the same order as how pixels are scanned on a typical computer screen). The training dataset has 60,000 instances, and the testing dataset has 10,000 instances. For each dataset, each entry for a given feature variable and instance is a discrete integer value from 0 to 255, inclusive, corresponding to the grey scale value of a pixel (0 is white, and 255 is black). Each dataset has a single target variable that takes a categorical value of one of the ten numerical digits between 0 and 9, inclusive, that corresponds to an intended hand-written digit on the 28x28 image grid, encompassed by the corresponding 784 feature variables.

In seeking an implementation of the algorithm that I developed on the more complex **MNIST** dataset, considerations must be paid to key properties of the **MNIST** dataset that are not present in the iris dataset. Such properties, if not accounted for, will undermine the implementation of the copula **ADVI** algorithm. For example:

i There are features in the **MNIST** training dataset that take the same value (a value of zero) across each of the 60,000 instances, and thus do not contribute information from which more precise and accurate predictions can be made.

ii The variance of many/most of the features in the **MNIST** training dataset have a variance of zero, or have a very small variance, within the 10 subsets categorised by each of the 10 values that the target variable can take (even after excluding those that have been identified in the bullet point above). This is problematic, due to the fact that the likelihood for the model adopted takes a

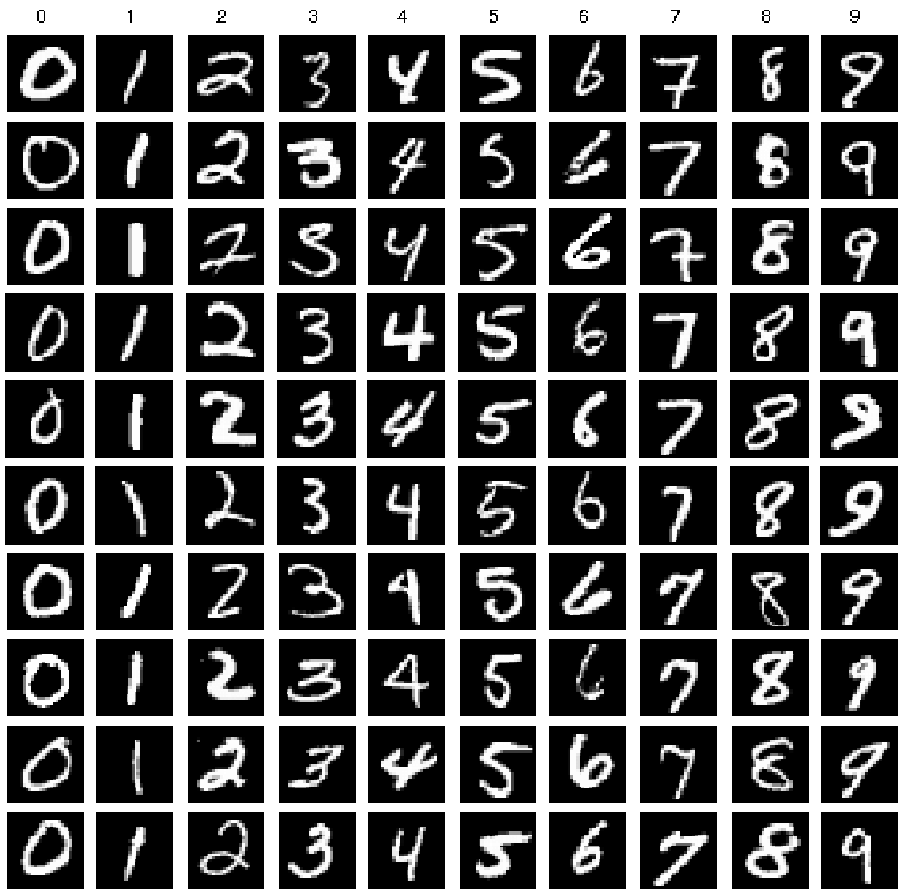


FIGURE 3.1: A sample of graphically depicted instances from the **MNIST** dataset. The column labels (from 0 to 9) at the top of the figure state the associated target variable value of the instances in that column.

Gaussian form (see appendix section B.2), thus resulting in extreme values for small variance values, and is undefined for zero variance values.

iii There are a small number of features, within the 10 subsets categorised by each of the 10 values that the target variable can take, that have a far larger variance than most of the other features. These features correspond to the pixels that tend to be on the edges of the digits being drawn. Depending on natural variations as to how a given digit is drawn, these pixel may be full illuminated, partially illuminated, or not illuminated at all (they are less consistently illuminated, or not illuminated).

In order to address the caveats outlined in the bullet points above, I first applied a series of pre-processing steps to the **MNIST** dataset. These pre-processing steps can be outlined as follows:

i First, drop all of the feature variables in the **MNIST** training dataset that have an overall variance of zero.

ii Next, standardise the resulting truncated dataset (also required for the next pre-processing step).

iii Apply **PCA** on the dataset, retaining the corresponding target variable values for each of the instances. This truncation, standardisation, and then **PCA** application, of the **MNIST** training dataset addresses all of the variance-based caveats identified in the preceding series of bullet points (the first bullet point is addressed by the truncation, the second bullet point is addressed by the **PCA** application, and the third bullet point is addressed by the standardisation).

iv Select the number of principle components to use, which is between 1 and the number of feature variables in the truncated dataset, that results after the first pre-processing stage (i.e., 717). The number of principle components to use is a hyperparameter that should be chosen to be a large as reasonably possible, limited by a pragmatic judgement of the computational expense available.

v Calculate the required $\boldsymbol{\beta}$ vector (which will be a 10-dimensional vector). Then calculate the means and variances of each of the selected principle components, for each of the 10 subsets categorised by each of the 10 values that the target variable can take. This pre-processing stage is identical to that of the iris dataset - see appendix section B.4.

vi Having extracted all the information required from the **MNIST** training dataset, this dataset can be dropped, allowing for the required pre-processing of the **MNIST** testing dataset to now be considered.

vii First, drop the same features variables in the **MNIST** testing dataset, as was done for the **MNIST** training dataset.

viii Next, for each entry i in the truncated testing dataset, replace it with $\frac{i-m}{s^2}$, where m and s^2 are the mean and variance, respectively, of the corresponding non-standardised feature variable in the **MNIST** training dataset in which the entry i would otherwise be located. That is, we are effectively standardising the truncated testing dataset in the context of the truncated training dataset.

ix Take the corresponding **PCA** loadings that were calculated when **PCA** was applied on the truncated and standardised training dataset. Use these loadings to calculate the 'pseudo' principle components, in the context of the previous **PCA** application, for each of the entries in the resulting testing dataset.

A complete application (including the pre-processing steps outlined above) of my originally developed Copula **ADVI** algorithm on the **MNIST** dataset is provided by the Python script in appendix section B.7. In this script, 300 principle components were chosen, with most of the other hyperparameters taking the same values as they did for the iris dataset application in appendix section B.4.

An initial long-term execution of a variant of the script in appendix section B.7 indicates that the Copula **ADVI** algorithm is able to make very accurate predictions on the **MNIST** dataset, much like for the iris dataset. However, even given appropriate adjustments to the hyperparameters to reduce computational expense, the script in appendix section B.7 was found to take too long to execute for appropriate testing to be conducted. After having analysed various aspects of the script, it was found that the autograd package likely contributes the most computational expense to the script, and is proving to be a bottle neck now that the Copula **ADVI** application has been scaled up to this extent (from the iris dataset to the **MNIST** dataset).

This setback could perhaps offer crucial support in favour of the research limitation outlined in section 4.2, that the use of a high-level programming language such as

Python could be critically hindering the testing and execution of the algorithm. I would say that this is the most crucial element to consider, should further research, testing and extensions be made to the Copula **ADVI** algorithm.

4 Limitations & Avenues for Future Research

4.1 Current Summary, Following Chapter 3

First, let the consequences of the interpretations and conclusions summarised in section 3.3 be considered. The interpretations made led to the conclusion that copula **ADVI**, just as for **MCMC** and the likes, is more computationally expensive and more accurate than mean-field **ADVI**. Therefore, just as is the case when deciding whether to use **MCMC** (or the likes), or **VI** in general (Blei, Kucukelbir, and McAuliffe, 2017), particular attention should be paid to the scenario in which either copula **ADVI** or mean-field **ADVI** will be used (if we have already decided that one of these two algorithms are to be used). For example:

i What computational resources are available for an analysis and evaluation of the data?

ii How important is it for accurate predictions to be made? What degree of error could be permitted?

iii Given the context, do we suspect that the latent variables are going to be strongly correlated, or not?

The answers to these questions will highlight whether copula **ADVI** or mean-field **ADVI** will be a more suitable choice in any given case scenario.

4.2 Research Limitations

This section considers and outlines the limitations primarily and/or explicitly attributed to the original research itself. Furthermore, a direct consequence of each of these limitations are avenues of future research (there are other avenues of future research that are not just based on these research limitations, as will be outlined in section 4.3).

Let the fact that only the iris dataset was used for testing be the first limitation to consider. More rigorous empirical testing could have been done using additional testing datasets, that would hopefully corroborate the interpretations and conclusions outlined in section 3.3. In particular, datasets with an alternative structure, such as those for which continuous numerical (as opposed to categorical) predictions need to be made, could have been used. Such datasets would require a different model, $p(\mathbf{x}|\mathbf{z})$, from the "continuous categorical model" used throughout chapters 2 and 3, and could thus help to rule out the possibility of the interpretations and conclusions resulting, at least in part, from the particular "continuous categorical model" used (the interpretations and conclusions should be independent from any particular model).

Building on the latter points raised in the previous paragraph, the use of alternative "continuous categorical" models could also be explored. That is, testing other such "continuous categorical" models, that take a different approach and have a different structure to that used in the original research, that still satisfy the original requirements, to see if any differences in the interpretations and conclusions ultimately arise. In particular, a model could be developed and tested that does not require an execution of the algorithm "per class, per data point" (see section 2.3), but rather just "per data point", as is conventional for **VI** in general.

Alternative values of the hyperparameters to those set in section 3.1 and appendix section B.4 could have been more systematically explored. The chosen values of the hyperparameters followed from an exploratory setting up of draft versions of the copula **ADVI** algorithm, the final works of which are provided in section 2.3, section 3.1 and appendix section B.4. However, no attempt was made to more formally and systematically test alternative values of the hyperparameters, as a means of gauging how such setting may influence the result, interpretations and conclusions.

Recall from section 2.2 that only Gaussian copulas were considered in the original research. This sole consideration was adopted to maintain a Gaussian surrogate posterior, and thus allow much of the original research to build on developments already outlined in the paper (Kucukelbir et al., 2017). However, an exploration of augmentations of alternative copula types could have been pursued, subject to an appropriate development of those aspects of the copula **ADVI** algorithm that currently assume a Gaussian surrogate posterior (for example, consider the closed-form expression of the entropy given by equation 1.50).

Recall from section 2.3 that the gradient ascent approach of the copula **ADVI** algorithm in this thesis is different in structure from that of a more generic copula **VI** setup, as outlined in the article (Tran, Blei, and Airoldi, 2015). That is: In copula **VI** (Tran, Blei, and Airoldi, 2015), the optimisation (gradient ascent) is first carried out over the non-copula variational parameters, and then over the copula variational parameters; and in copula **ADVI** (section 2.3), the optimisation (gradient ascent) is carried out over both the copula and non-copula variational parameters at the same time. However, it is not known whether an implementation of either of these gradient ascent approaches, or even an alternative approach, fundamentally influences the results, interpretations or conclusions made.

The choice of the step size sequence for the practical implementation and empirical testing of the copula **ADVI** algorithm, as outlined in equation 3.1, is another possible source of bias, influencing the resulting empirical results, interpretations and/or conclusions: Particularly as this is the only step size sequence that was used in testing. It is suspected that the conclusion that "the number of gradient ascent iterations required is not influenced by our choice of algorithm" (see section 3.3) may have been more heavily influenced by this limitation than the other conclusions made. Furthermore, as outlined in section 1.9, our copula **ADVI** algorithm (which is also an **SVI** algorithm) is certain to converge to a local optimum of the **ELBO**, should the step size sequence abide by a Robbins-Monro schedule. However, the step size sequence of equation 3.1 does not abide by such a schedule, the lack of convergence guarantees of which may have unduly influenced the empirical results, and thus the interpretations and conclusions subsequently drawn.

Recall from section 3.1 that many aspects of the empirical and practical setup arose as a means of bypassing, as far as reasonably and practically possible, the computational caveats of Python (being a high-level programming language). However, a more effective way of stemming the influence of such higher-level programming language caveats could be to program the copula **ADVI** algorithm in a lower-level programming language (such as C, or even variants of assembly language). Although preceding research has already confirmed the larger computational expense of a copula augmented **VI** algorithm (Tran, Blei, and Airoldi, 2015), the larger computational expense of copula **ADVI** compared to its mean-field counterpart may not be as substantial and significant as indicated, interpreted and concluded in sections 3.2 and 3.3. That is, the significant "Relative Execution Time" difference interpreted between the mean-field and copula **ADVI** algorithms may be substantially attributed to (unaccounted for) high-level programming language caveats, and not just the algorithms themselves.

Finally, consider the t-tests conducted in section 3.2 and interpreted in section 3.3. T-tests, in the context of how they have been conducted in section 3.2, assume that the samples, about which the tests are conducted, are drawn from populations distributed as Gaussians (where the population standard deviations are unknown). However, it may not be appropriate to assume that the complete populations of the "Number of Gradient Ascent Iterations Required" or the "Relative Execution Time", about which the t-tests were conducted in section 3.2, attributed to either the mean-field or copula **ADVI** algorithms, are distributed as Gaussians. Further theoretical analysis may reveal an expectation for such underlying populations to follow alternative distributions, thus nullifying our current use of the t-test.

4.3 Avenues for Future Research

As noted in section 4.2, each of the research limitations present avenues for future research. The avenues for future research, corresponding to each of the research limitations identified in section 4.2, in order, can be briefly summarised as follows (refer to section 4.2 for further details):

i Conduct empirical testing using additional datasets - not just the iris dataset.

ii Conduct empirical testing using additional datasets that differ in structure. For example, for which continuous numerical predictions need to be made.

iii Conduct empirical testing using a variety of models, $p(\mathbf{x}, \mathbf{z})$ - not just the one model outlined in appendix sections B.1 and B.2, and section 3.1 (where $p(\mathbf{x}, \mathbf{z}) = p(\mathbf{z}) * p(\mathbf{x}|\mathbf{z})$).

iv Include an element of systematic testing of alternative hyperparameter values in the empirical testing.

v Conduct empirical testing using a variety of copula types - not just the Gaussian copula.

vi Conduct empirical testing using both gradient ascent approaches identified, in section 2.3 and in the article (Tran, Blei, and Airoldi, 2015).

vii Conduct empirical testing using a variety of step size sequences, δ_i - not just the one outlined in equation 3.1. In particular, include step size sequences that satisfy a Robbins-Monro schedule.

viii Conduct empirical testing using a low-level, or at least a lower-level, programming language.

ix Carry out further theoretical analysis on the copula **ADVI** algorithm, and its mean-field counterpart, to establish whether the distributions, or characteristics of such distributions, of the underlying populations of the numerical measures, can be determined. For example, we could potentially establish: what the ranges of the numerical measures are; whether the numerical measures would be symmetrically distributed; etc.

However, there are avenues for future research that arise as they go beyond the scope of the original research framework. These avenues arose at differing stages of the development, conduct, testing and evaluation of the original research.

Let the first of these avenues to consider be the one that can be succinctly summarised by the question "How does copula **ADVI** compare to **MCMC**?". Recall from section 3.3 that it was concluded that copula **ADVI** is more computationally expensive and more accurate than mean-field **ADVI**. It was already understood via preceding research that **MCMC** is more computationally expensive and more accurate than mean-field **ADVI** (Kucukelbir et al., 2017; Robert, Casella, and Casella, 1999). However, how would, for example, the general accuracy and computational expense of copula **ADVI** and **MCMC** compare? It is strongly suspected that **MCMC** would be more computationally expensive and more accurate than copula **ADVI**, as tends to be the case for **MCMC** compared to **VI** in general (see section 1.4). However, no such investigations or research have yet been conducted. Furthermore, similar such questions and research topics can also be posed for copula **ADVI** compared to other commonly used Bayesian-compatible approximation algorithms, that can be used to carry out the same objective.

Recall from section 2.3 that we were able to access the variational parameters of the Gaussian copula via the covariance/correlation matrix of the corresponding full-rank Gaussian density (as the univariate components were also distributed as Gaussians). Therefore, a pair copula factorisation and a consideration of the corresponding vine structure, of the multivariate Gaussian copula, were not required. However, should a multivariate copula type with computable pair copulas and pair copula derivatives be considered, without access to the same amenable mathematical features of the Gaussian copula, a restructuring of the copula **ADVI** algorithm and an exploration into different vine structures could thus be conducted.

Consider the dimension, $d \in \mathbb{N}$, of the latent vector variable, $\mathbf{z}$, where $\mathbf{z} \in \mathbb{R}^d$. It was outlined in section 2.3 that for the corresponding Gaussian surrogate posterior $q(\mathbf{z})$, there would be $\frac{d(d-1)}{2}$ copula variational parameters. However, this fact still holds for a generic copula augmentation of a mean-field structured surrogate posterior. Note that the number of copula variational parameters increases/decreases in a squared fashion with the dimension, d, of $\mathbf{z}$ (this is not the case for the non-copula variational parameters). It is thus suspected that the computational expense of the corresponding copula **ADVI** algorithm, or variants of it, could scale accordingly, with the dimension, d, of $\mathbf{z}$. However, an investigation into this potential was not a subject of the original research in this thesis, and can thus be considered another avenue of future research: This avenue of future research, in an empirical context, can be considered as being closely linked to the avenue "conduct empirical testing using additional datasets that differ in structure", where the differing structures result in latent vector variables with varying dimensions.

Arguably the most important and underpinning avenue of future research that could be undertaken would be to attempt to theoretically confirm the interpretations and conclusions made in section 3.3. Although we could further increase our confidence in the interpretations and conclusions made through further empirical experimentation - more samples, varied datasets, varying cross-validation folds, etc - such an approach will always lack absolute confirmations, the likes of which can only be obtained on a theoretical basis. Therefore, an attempt to make such (often very difficult) theoretical confirmations could be considered a worthwhile investment: Especially given the potential future significance of **VI** in general (see section 1.4).

A Chapter 1 Derivations

A.1 Derivation of the Evidence of the Exponential-Normal Model in Section 1.4

Let I denote an indicator function, where

$$I(Z \geq 0) = \begin{cases} 1, & Z \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{A.1}$$

Therefore:

$$\begin{aligned}
p(X) &= \int_Z p(X,Z)dZ \\
&= \int_Z p(X|Z)p(Z)dZ \quad (\text{As } p(X,Z) = p(X|Z) * p(Z)) \\
&= \int_{-\infty}^{\infty} \left(\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{X-\mu}{\sigma}\right)^2} \right) p(Z)dZ \quad (\text{As } p(X|Z) \text{ is a Gaussian density}) \\
&= \int_{-\infty}^{\infty} \left(\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{X-\mu}{\sigma}\right)^2} \right) \left(\lambda e^{-\lambda Z} I(Z \geq 0) \right) dZ \\
&\qquad (\text{As } p(Z) \text{ is an Exponential density}) \\
&= \int_{-\infty}^{\infty} \left(\frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(X-Z)^2} \right) \left(e^{-Z} I(Z \geq 0) \right) dZ \quad (\text{As } \mu = Z, \sigma = 1 \ \& \ \lambda = 1) \\
&= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-\frac{1}{2}X^2 + XZ - \frac{1}{2}Z^2} e^{-Z} I(Z \geq 0) dZ \\
&= \frac{1}{\sqrt{2\pi}} \int_{0}^{\infty} e^{-\frac{1}{2}X^2} e^{XZ} e^{-\frac{1}{2}Z^2} e^{-Z} dZ \quad (\text{Given the definition of } I(Z \geq 0)) \\
&= \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}X^2} \int_{0}^{\infty} e^{XZ} e^{-\frac{1}{2}Z^2} e^{-Z} dZ
\end{aligned} \tag{A.2}$$

A.2 Isolating the evidence from the generic VI Objective

$$
\begin{aligned}
\mathcal{KL}\left(q(\mathbf{z})||p(\mathbf{z}|\mathbf{x})\right) &= \mathbb{E}_{q(\mathbf{z})\in Q}\left[\log\left(\frac{q(\mathbf{z})}{p(\mathbf{z}|\mathbf{x})}\right)\right] \quad 1.8 \\
&= \mathbb{E}_{q(\mathbf{z})\in Q}\left[\log\left(\frac{q(\mathbf{z})}{p(\mathbf{x},\mathbf{z})/p(\mathbf{x})}\right)\right] \quad (\text{As } p(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{x},\mathbf{z})}{p(\mathbf{x})}) \\
&= \mathbb{E}_{q(\mathbf{z})\in Q}\left[\log\left(\frac{q(\mathbf{z})p(\mathbf{x})}{p(\mathbf{x},\mathbf{z})}\right)\right] \\
&= \mathbb{E}_{q(\mathbf{z})\in Q}\left[\log\left(\frac{q(\mathbf{z})}{p(\mathbf{x},\mathbf{z})}\right) + \log\left(p(\mathbf{x})\right)\right] \\
&= \mathbb{E}_{q(\mathbf{z})\in Q}\left[-\log\left(\frac{p(\mathbf{x},\mathbf{z})}{q(\mathbf{z})}\right) + \log\left(p(\mathbf{x})\right)\right] \\
&= \mathbb{E}_{q(\mathbf{z})\in Q}\left[-\log\left(\frac{p(\mathbf{x},\mathbf{z})}{q(\mathbf{z})}\right)\right] + \mathbb{E}_{q(\mathbf{z})\in Q}\left[\log\left(p(\mathbf{x})\right)\right] \\
&\qquad (\text{Due to the Linearity of } \mathbb{E}) \\
&= \mathbb{E}_{q(\mathbf{z})\in Q}\left[-\log\left(\frac{p(\mathbf{x},\mathbf{z})}{q(\mathbf{z})}\right)\right] + \log\left(p(\mathbf{x})\right) \\
&\qquad (\text{As the latter term does not depend on } \mathbf{z}) \\
&= -\mathbb{E}_{q(\mathbf{z})\in Q}\left[\log\left(\frac{p(\mathbf{x},\mathbf{z})}{q(\mathbf{z})}\right)\right] + \log\left(p(\mathbf{x})\right)
\end{aligned} \tag{A.3}
$$

A.3 Derivation of the ELBO via Jensen's Inequality

The inequality $\log\left(p(\mathbf{x})\right) \geq \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}\left[\log\left(p(\mathbf{x},\mathbf{z})\right)\right] + \mathbb{H}(\mathbf{z})$ can be derived as follows:

$$
\begin{aligned}
\log\left(p(\mathbf{x})\right) &= \log\left(\int_{\mathbf{z}} p(\mathbf{x},\mathbf{z})d\mathbf{z}\right) \\
&= \log\left(\int_{\mathbf{z}} q(\mathbf{z})\frac{p(\mathbf{x},\mathbf{z})}{q(\mathbf{z})}d\mathbf{z}\right) \\
&\geq \int_{\mathbf{z}} q(\mathbf{z})\log\left(\frac{p(\mathbf{x},\mathbf{z})}{q(\mathbf{z})}\right)d\mathbf{z} \quad (\text{By Jensen's Inequality (Jordan et al., 1999)}) \\
&= \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}\left[\log\left(\frac{p(\mathbf{x},\mathbf{z})}{q(\mathbf{z})}\right)\right] \quad (\text{Given the definition of } \mathbb{E}) \\
&= \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}\left[\log\left(p(\mathbf{x},\mathbf{z})\right) - \log\left(q(\mathbf{z})\right)\right] \\
&= \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}\left[\log\left(p(\mathbf{x},\mathbf{z})\right)\right] - \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}\left[\log\left(q(\mathbf{z})\right)\right] \quad (\text{Given the linearity of } \mathbb{E}) \\
&= \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}\left[\log\left(p(\mathbf{x},\mathbf{z})\right)\right] + \mathbb{H}(\mathbf{z}) \quad (\text{where } \mathbb{H}(\mathbf{z}) = -\mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}\left[\log\left(q(\mathbf{z})\right)\right])
\end{aligned} \tag{A.4}
$$

The quantity $\mathbb{H}(\mathbf{z}) = -\mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}\left[\log\left(q(\mathbf{z})\right)\right]$ is the Shannon entropy (Yang, 2017).

A.4 MFVI Factors in the CAVI Algorithm, for a 3D Case

Let an expression for the **ELBO**, given that $q(\mathbf{z}) = q_1(z_1)q_2(z_2)q_3(z_3)$, first be obtained, as follows:

$$
\begin{aligned}
\text{ELBO}\left[q(\mathbf{z})\right] &= \mathbb{E}_{q(\mathbf{z})\in Q}\left[\log\left(\frac{p(\mathbf{x},\mathbf{z})}{q(\mathbf{z})}\right)\right] \quad \text{(By equation 1.16)}\\
&= \mathbb{E}_{q(\mathbf{z})\in Q}\left[\log\left(\frac{p(\mathbf{x},\mathbf{z})}{q_1(z_1)q_2(z_2)q_3(z_3)}\right)\right]\\
&= \int_{\mathbf{z}} \log\left(\frac{p(\mathbf{x},\mathbf{z})}{q_1(z_1)q_2(z_2)q_3(z_3)}\right) q_1(z_1)q_2(z_2)q_3(z_3)d\mathbf{z}\\
&\qquad \text{(Given the definition of } \mathbb{E}\text{)}\\
&= \int_{\mathbf{z}} \left[\log\left(p(\mathbf{x},\mathbf{z})\right) - log(q_1(z_1)) - log(q_2(z_2)) - log(q_3(z_3))\right]\\
&\qquad q_1(z_1)q_2(z_2)q_3(z_3)d\mathbf{z}\\
&= \int_{z_1,z_2,z_3} \left[\log\left(p(\mathbf{x},\mathbf{z})\right) - log(q_1(z_1)) - log(q_2(z_2)) - log(q_3(z_3))\right]\\
&\qquad q_1(z_1)q_2(z_2)q_3(z_3)dz_1dz_2dz_3\\
&= \int_{z_1,z_2,z_3} \log\left(p(\mathbf{x},\mathbf{z})\right) q_1(z_1)q_2(z_2)q_3(z_3)dz_1dz_2dz_3\\
&\quad - \int_{z_1,z_2,z_3} \log\left(q_1(z_1)\right) q_1(z_1)q_2(z_2)q_3(z_3)dz_1dz_2dz_3\\
&\quad - \int_{z_1,z_2,z_3} \log\left(q_2(z_2)\right) q_1(z_1)q_2(z_2)q_3(z_3)dz_1dz_2dz_3\\
&\quad - \int_{z_1,z_2,z_3} \log\left(q_3(z_3)\right) q_1(z_1)q_2(z_2)q_3(z_3)dz_1dz_2dz_3\\
&= \int_{z_1,z_2,z_3} \log\left(p(\mathbf{x},\mathbf{z})\right) q_1(z_1)q_2(z_2)q_3(z_3)dz_1dz_2dz_3\\
&\quad - \int_{z_1} \log\left(q_1(z_1)\right) q_1(z_1)dz_1 * \int_{z_2,z_3} q_2(z_2)q_3(z_3)dz_2dz_3\\
&\quad - \int_{z_2} \log\left(q_2(z_2)\right) q_2(z_2)dz_2 * \int_{z_1,z_3} q_2(z_1)q_3(z_3)dz_1dz_3\\
&\quad - \int_{z_3} \log\left(q_3(z_3)\right) q_3(z_3)dz_3 * \int_{z_1,z_2} q_1(z_1)q_2(z_2)dz_1dz_2
\end{aligned} \tag{A.5}
$$

As each of z_1, z_2 and z_3 are assumed to be mutually independent, where $q_1(z_1)$, $q_2(z_2)$ and $q_3(z_3)$ are probability densities, then $\int_{z_i,z_j} q_i(z_i)q_j(z_j)dz_idz_j = 1$, for $i,j \in \{1,2,3\}$ and $i \neq j$. Thus:

$$\begin{aligned}
\text{ELBO}\,[q(\mathbf{z})] &= \int_{z_1,z_2,z_3} \log\left(p(\mathbf{x},\mathbf{z})\right) q_1(z_1) q_2(z_2) q_3(z_3) dz_1 dz_2 dz_3 \\
&\quad - \int_{z_1} \log\left(q_1(z_1)\right) q_1(z_1) dz_1 \\
&\quad - \int_{z_2} \log\left(q_2(z_2)\right) q_2(z_2) dz_2 \\
&\quad - \int_{z_3} \log\left(q_3(z_3)\right) q_3(z_3) dz_3 \qquad (\beta_*) \\
&= \int_{z_1} q_1(z_1) \left[\int_{z_2,z_3} \log\left(p(\mathbf{x},\mathbf{z})\right) q_2(z_2) q_3(z_3) dz_2 z_3 - \log\left(q_1(z_1)\right) \right] dz_1 \\
&\quad - \int_{z_2} \log\left(q_2(z_2)\right) q_2(z_2) dz_2 - \int_{z_3} \log\left(q_3(z_3)\right) q_3(z_3) dz_3 \qquad (\beta_1) \\
&= \int_{z_2} q_2(z_2) \left[\int_{z_1,z_3} \log\left(p(\mathbf{x},\mathbf{z})\right) q_1(z_1) q_3(z_3) dz_1 z_3 - \log\left(q_2(z_2)\right) \right] dz_2 \\
&\quad - \int_{z_1} \log\left(q_1(z_1)\right) q_1(z_1) dz_1 - \int_{z_3} \log\left(q_3(z_3)\right) q_3(z_3) dz_3 \qquad (\beta_2) \\
&= \int_{z_3} q_3(z_3) \left[\int_{z_1,z_2} \log\left(p(\mathbf{x},\mathbf{z})\right) q_1(z_1) q_2(z_2) dz_1 z_2 - \log\left(q_3(z_3)\right) \right] dz_3 \\
&\quad - \int_{z_1} \log\left(q_1(z_1)\right) q_1(z_1) dz_1 - \int_{z_2} \log\left(q_2(z_2)\right) q_2(z_2) dz_2 \qquad (\beta_3)
\end{aligned} \tag{A.6}$$

Each of equations β_1, β_2 & β_3 follow from equation β_*, given the factorisation of each of z_1, z_2 & z_3, respectively, into their outer integral expressions.

By the Euler-Lagrange equation, we thus have:

$$\frac{\delta \text{ELBO}[q(\mathbf{z})]}{\delta q_1} = \int_{z_2,z_3} \log\left(p(\mathbf{x},\mathbf{z})\right) q_2(z_2) q_3(z_3) dz_2 dz_3 - \log\left(q_1(z_1)\right) - 1 - 1 = 0 \tag{A.7}$$

$$\frac{\delta \text{ELBO}[q(\mathbf{z})]}{\delta q_2} = \int_{z_1,z_3} \log\left(p(\mathbf{x},\mathbf{z})\right) q_1(z_1) q_3(z_3) dz_1 dz_3 - \log\left(q_2(z_2)\right) - 1 - 1 = 0 \tag{A.8}$$

$$\frac{\delta \text{ELBO}[q(\mathbf{z})]}{\delta q_3} = \int_{z_1,z_2} \log\left(p(\mathbf{x},\mathbf{z})\right) q_1(z_1) q_2(z_2) dz_1 dz_2 - \log\left(q_3(z_3)\right) - 1 - 1 = 0 \tag{A.9}$$

Equations A.7, A.8 and A.9 follow from application of the Euler-Lagrange equation to β_1, β_2 and β_3, for derivatives with respect to q_1, q_2 and q_3, respectively, where we are setting the respective integrals equal to zero, corresponding to anticipated local maxima of the **ELBO** (given an appropriately chosen model such that the **ELBO** has a maximum).

As a notational shorthand, let $\mathbb{E}_{-i}$ denote the expectation with respect to $\mathbf{z}$, excluding its i^{th} component, distributed as $\frac{q(\mathbf{z})}{q_i(z_i)}$. Given equation A.7, we have:

$$\begin{aligned}
\int_{z_2,z_3} \log(p(\mathbf{x},\mathbf{z}))q_2(z_2)q_3(z_3)dz_2dz_3 - \log(q_1(z_1)) - 1 - 1 &= 0 \\
\int_{z_2,z_3} \log(p(\mathbf{x},\mathbf{z}))q_2(z_2)q_3(z_3)dz_2dz_3 &= \log(q_1(z_1)) + 2 \\
\mathbb{E}_{(z_2 \quad z_3)^T \sim q_2(z_2)q_3(z_3)}\left[\log(p(\mathbf{x},\mathbf{z}))\right] &= \log(q_1(z_1)) + 2 \\
\log(q_1(z_1)) &= \mathbb{E}_{(z_2 \quad z_3)^T \sim q_2(z_2)q_3(z_3)}\left[\log(p(\mathbf{x},\mathbf{z}))\right] - 2 \\
\log(q_1(z_1)) &= \mathbb{E}_{-1}\left[\log(p(\mathbf{x},\mathbf{z}))\right] - 2 \\
q_1(z_1) &= e^{-2} * e^{\mathbb{E}_{-1}[\log(p(\mathbf{x},\mathbf{z}))]} \\
q_1(z_1) &\propto e^{\mathbb{E}_{-1}[\log(p(\mathbf{x},\mathbf{z}))]} \\
q_1(z_1) &= \alpha_1 e^{\mathbb{E}_{-1}[\log(p(\mathbf{x},\mathbf{z}))]}
\end{aligned} \tag{A.10}$$

In derivation A.10, the proportion in the penultimate line is required to ensure that $q_1(z_1)$ integrates to one over its domain for subsequent iterations of the corresponding **CAVI** algorithm: $e^{-2} * e^{\mathbb{E}_{-1}[\log(p(\mathbf{x},\mathbf{z}))]}$ will not necessarily be a probability density - only proportional to one. That is, α_1 is an appropriate normalisation constant, which in general will depend on: the chosen model $p(\mathbf{x},\mathbf{z})$; the variational parameters of all of the mean-field factors $q_j(z_j)$; and the number of mean-field factors (there are 3 factors in this case).

Given the symmetry among the subscript labelling in equations A.7, A.8 and A.9, and the fact that A.10 follows from equation A.7, it can thus be deduced that:

$$\begin{aligned}
q_1(z_1) &= \alpha_1 e^{\mathbb{E}_{-1}[\log(p(\mathbf{x},\mathbf{z}))]} \\
q_2(z_2) &= \alpha_2 e^{\mathbb{E}_{-2}[\log(p(\mathbf{x},\mathbf{z}))]} \\
q_3(z_3) &= \alpha_3 e^{\mathbb{E}_{-3}[\log(p(\mathbf{x},\mathbf{z}))]}
\end{aligned} \tag{A.11}$$

A.5 Derivation of the Joint Model and Surrogate Posterior for the GMM

Let $\mathbf{c} = \{c_1, c_2, \ldots, c_n\}$. Hence, given that $\boldsymbol{\mu}$ and $\mathbf{c}$ encompass all the latent variables of the **GMM**, the required Bayesian joint model and surrogate posterior to derive are $p(\boldsymbol{\mu}, \mathbf{c}, \mathbf{x})$ and $q(\boldsymbol{\mu}, \mathbf{c})$, respectively. Furthermore, distribution 1.27 therefore provides us with an expression of the likelihood of the **GMM**, given by:

$$p(x_i|\boldsymbol{\mu}, c_i) = \mathcal{N}(x_i; c_i^T\boldsymbol{\mu}, 1) \tag{A.12}$$

Given the conditional on the left-hand side of equation A.12, we thus have: $p(c_i, x_i|\boldsymbol{\mu}) = p(x_i|\boldsymbol{\mu}, c_i) * p(c_i)$. As each of the c_i values are sampled independently, and the corresponding x_i values depend directly on their c_i counterparts, then it can be further deduced that:

$$p(\mathbf{c},\mathbf{x}|\boldsymbol{\mu}) = \prod_{i=1}^{n} p(c_i, x_i|\boldsymbol{\mu}) = \prod_{i=1}^{n} p(c_i)p(x_i|\boldsymbol{\mu}, c_i) \tag{A.13}$$

The conditional on the left-hand side of equation A.13 can be expressed as $p(\mathbf{c},\mathbf{x}|\boldsymbol{\mu}) = \frac{p(\boldsymbol{\mu},\mathbf{c},\mathbf{x})}{p(\boldsymbol{\mu})}$. Furthermore, as is the case for the c_i values (outlined above), as each of the μ_i values are sampled independently, then $p(\boldsymbol{\mu}) = \prod_{k=1}^{K} p(\mu_k)$. Given the two equivalences outlined in this paragraph, and the results summarised in equation A.13, it can thus be deduced that:

$$\begin{aligned} p(\boldsymbol{\mu},\mathbf{c},\mathbf{x}) &= p(\boldsymbol{\mu})p(\mathbf{c},\mathbf{x}|\boldsymbol{\mu}) \\ &= p(\boldsymbol{\mu})\prod_{i=1}^{n} p(c_i, x_i|\boldsymbol{\mu}) \\ &= p(\boldsymbol{\mu})\prod_{i=1}^{n} p(c_i)p(x_i|\boldsymbol{\mu}, c_i) \\ &= \prod_{k=1}^{K} p(\mu_k)\prod_{i=1}^{n} p(c_i)p(x_i|\boldsymbol{\mu}, c_i) \end{aligned} \tag{A.14}$$

(the first, and latter two, equivalent expressions of the joint model in equation A.14 are summarised in equation 1.28).

Recall that the surrogate posterior $q(\mathbf{z})$ of the **VI** method, to which this appendix section relates (see section 1.8), is to have the mean-field approximation applied to it. Therefore, given that the latent variables are $\{\boldsymbol{\mu}, \mathbf{c}\} = \{\mu_1, \dots, \mu_K, c_1, \dots, c_n\}$, the surrogate posterior is thus given by:

$$\begin{aligned} q(\boldsymbol{\mu},\mathbf{c}) &= q_{\boldsymbol{\mu}}(\boldsymbol{\mu})q_{\mathbf{c}}(\mathbf{c}) \\ &= q_{\mu,1}(\mu_1)q_{\mu,2}(\mu_2)\dots q_{\mu,K}(\mu_K)q_{c,1}(c_1)q_{c,2}(c_2)\dots q_{c,n}(c_n) \\ &= \prod_{k=1}^{K} q_{\mu,k}(\mu_k)\prod_{i=1}^{n} q_{\mathbf{c},i}(c_i) \end{aligned} \tag{A.15}$$

where $q_{\mu,k}$ and $q_{\mathbf{c},i}$ are the one-dimensional surrogate posterior components, and $q_{\boldsymbol{\mu}}$ and $q_{\mathbf{c}}$ are the surrogate posterior component types, of the complete surrogate posterior q, for the latent variable types $\boldsymbol{\mu}$ and $\mathbf{c}$, respectively, within the mean-field approximation.

A.6 CAVI Solution of the Surrogate Posterior Factor Type $q_{\mathbf{c},i}(c_i)$ for the GMM

This appendix section outlines a derivation for the **CAVI** solution - the general form of which is given by $q_j(z_j) = \alpha_j e^{\mathbb{E}_{-j}[\log p(\mathbf{x},\mathbf{z})]}$ (equation 1.24) - for the surrogate posterior factor type $q_{\mathbf{c},i}(c_i)$ in the **GMM** of section 1.8.

In derivation A.16, the variables α_m for $m \in \{1,2,3\}$ are constants, and the variable i in the surrogate posterior factor type $q_{\mathbf{c},i}(c_i)$ takes the particular value of s, where

s is equal to precisely one of the integers in the set $\{1, 2, \ldots, n\}$ (one of the allowed values of i). Given the **CAVI** algorithm in section 1.7, all of the other latent variables in the **GMM**, not being considered - $\{\mu_1, \mu_2, \ldots, \mu_K, c_1, c_2, \ldots, c_{s-1}, c_{s+1}, \ldots, c_n\}$ - are treated as constants (held fixed), and thus will ultimately be absorbed into the series of constants α_m.

$$
\begin{aligned}
\log\left(q_{\mathbf{c},s}(c_s)\right) &= \log\left(\alpha_{\mathbf{c},s} e^{\mathbb{E}_{-s}[\log p(\boldsymbol{\mu},\mathbf{c},\mathbf{x})]}\right) \\
&= \log\left(\alpha_{\mathbf{c},s}\right) + \mathbb{E}_{-s}[\log p(\boldsymbol{\mu},\mathbf{c},\mathbf{x})] \\
&= \alpha_1 + \mathbb{E}_{-s}[\log p(\boldsymbol{\mu},\mathbf{c},\mathbf{x})] \\
&= \alpha_1 + \mathbb{E}_{-s}\left[\sum_{k=1}^{K}\left(\log p(\mu_k)\right) + \sum_{i=1}^{n}\left(\log p(c_i)\right) + \sum_{j=1}^{n}\left(\log p(x_j|\boldsymbol{\mu}, c_j)\right)\right] \\
&= \alpha_1 + \mathbb{E}_{-s}\left[\sum_{k=1}^{K}\left(\log p(\mu_k)\right)\right] + \mathbb{E}_{-s}\left[\sum_{i=1}^{n}\left(\log p(c_i)\right)\right] \\
&\qquad + \mathbb{E}_{-s}\left[\sum_{j=1}^{n}\left(\log p(x_j|\boldsymbol{\mu}, c_j)\right)\right] \\
&= \alpha_2 + \mathbb{E}_{-s}\left[\sum_{i=1}^{n}\left(\log p(c_i)\right)\right] + \mathbb{E}_{-s}\left[\sum_{j=1}^{n}\left(\log p(x_j|\boldsymbol{\mu}, c_j)\right)\right] \\
&= \alpha_2 + \sum_{i=1}^{n}\mathbb{E}_{-s}\left[\log p(c_i)\right] + \sum_{j=1}^{n}\mathbb{E}_{-s}\left[\log p(x_j|\boldsymbol{\mu}, c_j)\right] \\
&= \alpha_2 + \mathbb{E}_{-s}\left[\log p(c_s)\right] + \sum_{i=1, i\neq s}^{n}\mathbb{E}_{-s}\left[\log p(c_i)\right] + \mathbb{E}_{-s}\left[\log p(x_s|\boldsymbol{\mu}, c_s)\right] \\
&\qquad + \sum_{j=1, j\neq s}^{n}\mathbb{E}_{-s}\left[\log p(x_j|\boldsymbol{\mu}, c_j)\right] \\
&= \alpha_3 + \mathbb{E}_{-s}\left[\log p(c_s)\right] + \mathbb{E}_{-s}\left[\log p(x_s|\boldsymbol{\mu}, c_s)\right] \\
&= \alpha_3 + \log\left(p(c_s)\right) + \mathbb{E}_{-s}\left[\log p(x_s|\boldsymbol{\mu}, c_s)\right]
\end{aligned}
\tag{A.16}
$$

The following numbered list contains comments, with each number corresponding to the order of the equality in derivation A.16 to which the comment applies (i.e., comment 1. is for the first equality, comment 2. is for the second equality, etc).

1. By equation 1.24. $\mathbb{E}_{-s}$ is the expectation with respect to all of the latent variables in the **GMM** except s, distributed as $\frac{q(\boldsymbol{\mu},\mathbf{c})}{q_{\mathbf{c},s}(c_s)}$.
3. Where $a_1 = \log\left(\alpha_{\mathbf{c},s}\right)$, given that $\alpha_{\mathbf{c},s}$ is a constant.
4. By equation 1.30.
5. Given the linearity of expectations, such as $\mathbb{E}_{-s}$.
6. Where $\alpha_2 = \alpha_1 + \mathbb{E}_{-s}\left[\sum_{k=1}^{K}\left(\log p(\mu_k)\right)\right]$. As the term $\mathbb{E}_{-s}\left[\sum_{k=1}^{K}\left(\log p(\mu_k)\right)\right]$ does not depend on c_s, then it is a constant term (all of the latent variables in this term, none of which are c_s, are held fixed).
7. Given the linearity of $\mathbb{E}_{-s}$.
8. Breaking away the terms where $i = s$ in each of the summations.
9. Where $\alpha_3 = \alpha_2 + \sum_{i=1, i\neq s}^{n}\mathbb{E}_{-s}\left[\log p(c_i)\right] + \sum_{j=1, j\neq s}^{n}\mathbb{E}_{-s}\left[\log p(x_j|\boldsymbol{\mu}, c_j)\right]$. As

each of the summations $\sum_{i=1,i\neq s}^{n} \mathbb{E}_{-s}\left[\log p(c_i)\right]$ and $\sum_{j=1,j\neq s}^{n} \mathbb{E}_{-s}\left[\log p(x_j|\boldsymbol{\mu}, c_j)\right]$ have been formulated with c_s having been broken away, then these are constant terms in this context.

10. As the expression $\log p(c_s)$ only depends on c_s, and thus not on any of the variables that $\mathbb{E}_{-s}$ is with respect to, then $\mathbb{E}_{-s}\left[\log p(c_s)\right] = \log\left(p(c_s)\right)$.

As an initialisation of the variational parameters ϕ for $q_{\mathbf{c},s}(c_s)$ in equation 1.31, required for the **CAVI** algorithm outlined in section 1.7, let each of the ϕ values be set such that they are equal. This is done to simplify the remainder of the derivation in this appendix section. As there are K such ϕ values, and their sum must equal 1 - requirement for the categorical distribution - then each ϕ value will be sent to $\frac{1}{K}$. An interpretation that follows from this initialisation is that, prior to any evidence, c_s is equal to each of the K possible one-hot vectors with a probability of $\frac{1}{K}$. Therefore:

$$p(c_i) = \frac{1}{K} \tag{A.17}$$

Let c_{sk} denote the value of the k^{th} element in the one-hot vector c_s, which will be either a 0 or a 1. By distribution 1.27, we have $p(x_s|\boldsymbol{\mu}, c_s) = \mathcal{N}(x_s; c_s^T\boldsymbol{\mu}, 1)$. As $c_s^T\boldsymbol{\mu} = \mu_k$ for the one value of k such that $c_{sk} = 1$, where all other values of k would result in $c_{sk} = 0$, then it can be deduced that:

$$p(x_s|\boldsymbol{\mu}, c_s) = \prod_{k=1}^{K} p(x_s|\mu_k)^{c_{sk}} \tag{A.18}$$

where, in general:

$$p(x_i|\mu_k) = \mathcal{N}(x_i; \mu_k, 1) \tag{A.19}$$

We can now make the following inferences about the term $\mathbb{E}_{-s}\left[\log p(x_s|\boldsymbol{\mu}, c_s)\right]$ from derivation A.16:

$$
\begin{aligned}
\mathbb{E}_{-s}\left[\log p(x_s|\boldsymbol{\mu}, c_s)\right] &= \mathbb{E}_{-s}\left[\log\left(\prod_{k=1}^{K} p(x_s|\mu_k)^{c_{sk}}\right)\right] \\
&= \mathbb{E}_{-s}\left[\sum_{k=1}^{K} \log p(x_s|\mu_k)^{c_{sk}}\right] \\
&= \mathbb{E}_{-s}\left[\sum_{k=1}^{K} c_{sk} \log p(x_s|\mu_k)\right] \\
&= \sum_{k=1}^{K} \mathbb{E}_{-s}\left[c_{sk} \log p(x_s|\mu_k)\right] \\
&= \sum_{k=1}^{K} c_{sk}\mathbb{E}_{-s}\left[\log p(x_s|\mu_k)\right] \\
&= \sum_{k=1}^{K} c_{sk}\mathbb{E}_{-s}\left[\log\left(\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}(x_s-\mu_k)^2}\right)\right] \\
&= \sum_{k=1}^{K} c_{sk}\mathbb{E}_{-s}\left[-\frac{1}{2}\left(x_s-\mu_k\right)^2 + \log\left(\frac{1}{\sqrt{2\pi}}\right)\right] \\
&= \sum_{k=1}^{K} c_{sk}\mathbb{E}_{-s}\left[-\frac{1}{2}\left(x_s^2 - 2x_s\mu_k + \mu_k^2\right) + \log\left(\frac{1}{\sqrt{2\pi}}\right)\right] \\
&= \sum_{k=1}^{K} c_{sk}\mathbb{E}_{-s}\left[-\frac{1}{2}x_s^2 + x_s\mu_k - \frac{1}{2}\mu_k^2 + \log\left(\frac{1}{\sqrt{2\pi}}\right)\right] \\
&= \sum_{k=1}^{K} c_{sk}\left(\mathbb{E}_{-s}\left[-\frac{1}{2}x_s^2\right] + \mathbb{E}_{-s}\left[x_s\mu_k\right] + \mathbb{E}_{-s}\left[-\frac{1}{2}\mu_k^2\right]\right) \\
&\qquad + \sum_{k=1}^{K} c_{sk}\left(\mathbb{E}_{-s}\left[\log\left(\frac{1}{\sqrt{2\pi}}\right)\right]\right) \\
&= \sum_{k=1}^{K} c_{sk}\left(-\frac{1}{2}x_s^2 + x_s\mathbb{E}_{-s}\left[\mu_k\right] - \frac{1}{2}\mathbb{E}_{-s}\left[\mu_k^2\right] + \log\left(\frac{1}{\sqrt{2\pi}}\right)\right) \\
&= \sum_{k=1}^{K} c_{sk}\left(-\frac{1}{2}x_s^2 + \log\left(\frac{1}{\sqrt{2\pi}}\right)\right) \\
&\qquad + \sum_{k=1}^{K} c_{sk}\left(x_s\mathbb{E}_{-s}\left[\mu_k\right] - \frac{1}{2}\mathbb{E}_{-s}\left[\mu_k^2\right]\right) \\
&= -\frac{1}{2}x_s^2 + \log\left(\frac{1}{\sqrt{2\pi}}\right) + \sum_{k=1}^{K} c_{sk}\left(x_s\mathbb{E}_{-s}\left[\mu_k\right] - \frac{1}{2}\mathbb{E}_{-s}\left[\mu_k^2\right]\right)
\end{aligned}
\tag{A.20}
$$

The following numbered list contains comments, with each number corresponding to the order of the equality in derivation A.20 to which the comment applies (i.e., comment 1. is for the first equality, comment 2. is for the second equality, etc).

1. By equation A.18
4. Given the linearity of $\mathbb{E}_{-s}$.
5. As none of the c_{sk} values, for $k \in \{1, 2, \ldots, K\}$ depend on any of the latent variables in the **GMM** that are not c_s, that $\mathbb{E}_{-s}$ is with respect to.

6. Given the definition of the normal distribution, and equation A.19.
10. Given the linearity of $\mathbb{E}_{-s}$. Please note that the expression on the right-hand side of this equality, that would have otherwise been expressed as a single summation, has been broken up into two summations so that the expression fits on the page.
11. Recall from section 1.5 that observations $\mathbf{x}$ are fixed in the **VI** method variants that we have adopted, which includes **CAVI** (see section 1.7): Hence $\mathbb{E}_{-s}\left[-\frac{1}{2}x_s^2\right] = -\frac{1}{2}x_s^2$ and $\mathbb{E}_{-s}\left[x_s\mu_k\right] = x_s\mathbb{E}_{-s}\left[\mu_k\right]$. Furthermore, as $\log\left(\frac{1}{\sqrt{2\pi}}\right)$ and $-\frac{1}{2}$ are constant, then $\mathbb{E}_{-s}\left[\log\left(\frac{1}{\sqrt{2\pi}}\right)\right] = \log\left(\frac{1}{\sqrt{2\pi}}\right)$ and $\mathbb{E}_{-s}\left[-\frac{1}{2}\mu_k^2\right] = -\frac{1}{2}\mathbb{E}_{-s}\left[\mu_k^2\right]$.
13. As $c_{sk} = 1$ for precisely one value of k in the set $\{1, 2, \dots, K\}$, where $c_{sk} = 0$ for every other value of k, and as neither $-\frac{1}{2}x_s^2$ nor $\log\left(\frac{1}{\sqrt{2\pi}}\right)$ depend on k (x_s does not depend on k once it has been observed, which is the case here), then $\sum_{k=1}^{K} c_{sk}\left(-\frac{1}{2}x_s^2 + \log\left(\frac{1}{\sqrt{2\pi}}\right)\right) = -\frac{1}{2}x_s^2 + \log\left(\frac{1}{\sqrt{2\pi}}\right)$.

Extending derivation A.16, given what has since been established and outlined in this appendix section, we thus have:

$$\begin{aligned}
\log\left(q_{\mathbf{c},s}(c_s)\right) &= \alpha_3 + \log\left(p(c_s)\right) + \mathbb{E}_{-s}\left[\log p(x_s|\boldsymbol{\mu}, c_s)\right] \\
&= \alpha_3 + \log\left(\frac{1}{K}\right) + \mathbb{E}_{-s}\left[\log p(x_s|\boldsymbol{\mu}, c_s)\right] \\
&= \alpha_3 - \log\left(K\right) + \mathbb{E}_{-s}\left[\log p(x_s|\boldsymbol{\mu}, c_s)\right] \\
&= \alpha_4 + \mathbb{E}_{-s}\left[\log p(x_s|\boldsymbol{\mu}, c_s)\right] \\
&= \alpha_4 - \frac{1}{2}x_s^2 + \log\left(\frac{1}{\sqrt{2\pi}}\right) + \sum_{k=1}^{K} c_{sk}\left(x_s\mathbb{E}_{-s}\left[\mu_k\right] - \frac{1}{2}\mathbb{E}_{-s}\left[\mu_k^2\right]\right) \\
&= \alpha_5 + \sum_{k=1}^{K} c_{sk}\left(x_s\mathbb{E}_{-s}\left[\mu_k\right] - \frac{1}{2}\mathbb{E}_{-s}\left[\mu_k^2\right]\right)
\end{aligned} \tag{A.21}$$

As was the case for derivation A.16, the variables α_m for $m \in \{3, 4, 5\}$ are constants. Following the "numbered list of comments" convention for derivations A.16 and A.20, we have:

1. Given the equality of the first and last expressions in derivation A.16.
2. By equation A.17.
4. It was outlined earlier in section 1.8 that "After the observations $\mathbf{x}$ have been generated, we disregard all of the **GMM** settings that resulted in the generation of this dataset, except for the number of Gaussian clusters K". That is, the value of K is already preset, and can thus be treated as a constant. It is thus absorbed into the series of constants α_m, where $\alpha_4 = \alpha_3 - \log\left(K\right)$.
5. Given the equality of the first and last expressions in derivation A.20.
6. As x_s is fixed, and thus treated as a constant, then $-\frac{1}{2}x_s^2 + \log\left(\frac{1}{\sqrt{2\pi}}\right)$ is a constant, where $\alpha_5 = \alpha_4 - \frac{1}{2}x_s^2 + \log\left(\frac{1}{\sqrt{2\pi}}\right)$.

For the surrogate posterior factor type $q_{\mathbf{c},s}(c_s)$, let a case be considered where $c_s = (1 \quad 0 \quad \dots \quad 0)^T$: That is, where $k = 1$. Hence:

$$\begin{aligned} q_{\mathbf{c},s}(c_s = (1 \quad 0 \quad \dots \quad 0)^T) &= \exp\left(\alpha_5 + \sum_{k=1}^{K} c_{sk}\left(x_s\mathbb{E}_{-s}\left[\mu_k\right] - \frac{1}{2}\mathbb{E}_{-s}\left[\mu_k^2\right]\right)\right) \\ &= \exp\left(\alpha_5 + x_s\mathbb{E}_{-s}\left[\mu_1\right] - \frac{1}{2}\mathbb{E}_{-s}\left[\mu_1^2\right]\right) \\ &= \exp\left(\alpha_5\right) * \exp\left(x_s\mathbb{E}_{-s}\left[\mu_1\right] - \frac{1}{2}\mathbb{E}_{-s}\left[\mu_1^2\right]\right) \\ &\propto \exp\left(x_s\mathbb{E}_{-s}\left[\mu_1\right] - \frac{1}{2}\mathbb{E}_{-s}\left[\mu_1^2\right]\right) \end{aligned} \tag{A.22}$$

Given the "numbered list of comments" convention used for the derivations above, now applied to derivation A.22, we have:

1. Given the equality of the first and last expressions in derivation A.21.
2. As $k = 1$, then $c_{s,1} = 1$ and $c_{sr} = 0$ for $r \in \{2, 3, \dots, K\}$. Therefore $\sum_{k=1}^{K} c_{sk}\left(x_s\mathbb{E}_{-s}\left[\mu_k\right] - \frac{1}{2}\mathbb{E}_{-s}\left[\mu_k^2\right]\right) = x_s\mathbb{E}_{-s}\left[\mu_1\right] - \frac{1}{2}\mathbb{E}_{-s}\left[\mu_1^2\right]$.
4. As α_5 is a constant, then so is $\exp\left(\alpha_5\right)$. Hence this proportionality holds.

An essential component of the ultimate "$q_{\mathbf{c},i}(c_i)$ **CAVI** solution" is the softmax function, also known as the "Normalised Exponential Transformation", which can be defined as follows (Bridle, 1989):

$$\text{Softmax}(A_c) = \frac{\exp\left(A_c\right)}{\sum_{b=1}^{B}\exp\left(A_b\right)} \tag{A.23}$$

where: c is an index-based integer value from the set $\{1, 2, \dots, B\}$; $B \in \mathbb{N}$; and A is a function that is parameterised by single members from the set $\{1, 2, \dots, B\}$, as denoted by the subscripts for A in equation A.23.

Recall from equation 1.31 that $\phi_{i,k}$ is the probability, according to the surrogate posterior $q_{\mathbf{c},i}(c_i)$, that c_i is equal to the one-hot vector with the 1 in the k^{th} position. Therefore $\phi_{s,1} = q_{\mathbf{c},s}(c_s = (1 \quad 0 \quad \dots \quad 0)^T)$. Finally, as we require $\sum_{k=1}^{K}\phi_{s,k} = 1$, given the definition of a categorical distribution, then for the case of equation 1.31 where $i = s$, considering the specific ϕ value where $k = 1$, we have:

$$\begin{aligned} \phi_{s,1} &= q_{\mathbf{c},s}(c_s = (1 \quad 0 \quad \dots \quad 0)^T) \\ &\propto \exp\left(x_s\mathbb{E}_{-s}\left[\mu_1\right] - \frac{1}{2}\mathbb{E}_{-s}\left[\mu_1^2\right]\right) \quad \text{(By Derivation A.22)} \\ \phi_{s,1} &= \frac{\exp\left(x_s\mathbb{E}_{-s}\left[\mu_1\right] - \frac{1}{2}\mathbb{E}_{-s}\left[\mu_1^2\right]\right)}{\sum_{k=1}^{K}\exp\left(x_s\mathbb{E}_{-s}\left[\mu_k\right] - \frac{1}{2}\mathbb{E}_{-s}\left[\mu_k^2\right]\right)} \\ &= \text{Softmax}\left(x_s\mathbb{E}_{-s}\left[\mu_1\right] - \frac{1}{2}\mathbb{E}_{-s}\left[\mu_1^2\right]\right) \quad \text{(By Equation A.23)} \end{aligned} \tag{A.24}$$

The penultimate line in derivation A.24 follows from a combination of the proportional requirement for $\phi_{s,1}$, as outlined in derivation A.22, and an appropriate normalisation adjustment that ensures the equation $\sum_{k=1}^{K} \phi_{s,k} = 1$ is abided by.

Although the derivation of the value $\phi_{s,1}$ above only considers the case where $k = 1$, it is evident that this derivation process generalises for any arbitrarily chosen value of $k \in \{1, 2, \ldots, K\}$. Furthermore, as s represents a specific yet arbitrary value of i, then the same generalisation is also applicable for the i indexing. Therefore, a complete generalisation of the results of derivation A.24 is summarised by the following equation:

$$\phi_{i,k} = Softmax\left(x_i \mathbb{E}_{-i}\left[\mu_k\right] - \frac{1}{2}\mathbb{E}_{-i}\left[\mu_k^2\right]\right) = \frac{\exp\left(x_i \mathbb{E}_{-i}\left[\mu_k\right] - \frac{1}{2}\mathbb{E}_{-i}\left[\mu_k^2\right]\right)}{\sum_{r=1}^{K} \exp\left(x_i \mathbb{E}_{-i}\left[\mu_r\right] - \frac{1}{2}\mathbb{E}_{-i}\left[\mu_r^2\right]\right)} \tag{A.25}$$

(this equation is given as equation 1.32 in section 1.8).

A.7 CAVI Solution of the Surrogate Posterior Factor Type $q_{\mu,k}(\mu_k)$ for the GMM

This appendix section outlines a derivation for the **CAVI** solution - the general form of which is given by $q_j(z_j) = \alpha_j e^{\mathbb{E}_{-j}[\log p(\mathbf{x},\mathbf{z})]}$ (equation 1.24) - for the surrogate posterior factor type $q_{\mu,k}(\mu_k)$ in the **GMM** of section 1.8.

In derivations A.26, A.27 and A.29, the variables α_m for $m \in \{1, 2, 3, 4, 5, 6, 7\}$ are constants, and the variable k in the surrogate posterior factor type $q_{\mu,k}(\mu_k)$ takes the particular value of t, where t is equal to precisely one of the integers in the set $\{1, 2, \ldots, K\}$ (one of the allowed values of k). Given the **CAVI** algorithm in section 1.7, all of the other latent variables in the **GMM**, not being considered - $\{\mu_1, \mu_2, \ldots, \mu_{t-1}, \mu_{t+1}, \ldots, \mu_K, c_1, c_2, \ldots, c_n\}$ - are treated as constants (held fixed), and thus will ultimately be absorbed into the series of constants α_m.

Let c_{jk} denote the value of the k^{th} element in the one-hot vector c_j, which will be either a 0 or a 1.

$$
\begin{aligned}
\log\left(q_{\mu,t}(\mu_t)\right) &= \log\left(\alpha_{\mu,t} e^{\mathbb{E}_{-t}[\log p(\boldsymbol{\mu},\mathbf{c},\mathbf{x})]}\right) \\
&= \log\left(\alpha_{\mu,t}\right) + \mathbb{E}_{-t}\left[\log p(\boldsymbol{\mu},\mathbf{c},\mathbf{x})\right] \\
&= \alpha_1 + \mathbb{E}_{-t}[\log p(\boldsymbol{\mu},\mathbf{c},\mathbf{x})] \\
&= \alpha_1 + \mathbb{E}_{-t}\left[\sum_{k=1}^{K}\left(\log p(\mu_k)\right) + \sum_{i=1}^{n}\left(\log p(c_i)\right) + \sum_{j=1}^{n}\left(\log p(x_j|\boldsymbol{\mu},c_j)\right)\right] \\
&= \alpha_1 + \mathbb{E}_{-t}\left[\sum_{k=1}^{K}\left(\log p(\mu_k)\right)\right] + \mathbb{E}_{-t}\left[\sum_{i=1}^{n}\left(\log p(c_i)\right)\right] \\
&\qquad + \mathbb{E}_{-t}\left[\sum_{j=1}^{n}\left(\log p(x_j|\boldsymbol{\mu},c_j)\right)\right] \\
&= \alpha_2 + \mathbb{E}_{-t}\left[\sum_{k=1}^{K}\left(\log p(\mu_k)\right)\right] + \mathbb{E}_{-t}\left[\sum_{j=1}^{n}\left(\log p(x_j|\boldsymbol{\mu},c_j)\right)\right] \\
&= \alpha_2 + \sum_{k=1}^{K}\mathbb{E}_{-t}\left[\log p(\mu_k)\right] + \sum_{j=1}^{n}\mathbb{E}_{-t}\left[\log p(x_j|\boldsymbol{\mu},c_j)\right] \\
&= \alpha_2 + \mathbb{E}_{-t}\left[\log p(\mu_t)\right] + \sum_{k=1,k\neq t}^{K}\mathbb{E}_{-t}\left[\log p(\mu_k)\right] \\
&\qquad + \sum_{j=1}^{n}\mathbb{E}_{-t}\left[\log p(x_j|\boldsymbol{\mu},c_j)\right] \\
&= \alpha_2 + \log p(\mu_t) + \sum_{k=1,k\neq t}^{K}\mathbb{E}_{-t}\left[\log p(\mu_k)\right] + \sum_{j=1}^{n}\mathbb{E}_{-t}\left[\log p(x_j|\boldsymbol{\mu},c_j)\right] \\
&= \alpha_3 + \log p(\mu_t) + \sum_{j=1}^{n}\mathbb{E}_{-t}\left[\log p(x_j|\boldsymbol{\mu},c_j)\right] \\
&= \alpha_3 + \log\left(\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{1}{2}\left(\frac{\mu_t}{\sigma}\right)^2}\right) + \sum_{j=1}^{n}\mathbb{E}_{-t}\left[\log\prod_{k=1}^{K}p(x_j|\mu_k)^{c_{jk}}\right] \\
&= \alpha_3 + \log\left(\frac{1}{\sigma\sqrt{2\pi}}\right) - \frac{1}{2}\left(\frac{\mu_t}{\sigma}\right)^2 + \sum_{j=1}^{n}\mathbb{E}_{-t}\left[\sum_{k=1}^{K}\log p(x_j|\mu_k)^{c_{jk}}\right] \\
&= \alpha_4 - \frac{\mu_t^2}{2\sigma^2} + \sum_{j=1}^{n}\mathbb{E}_{-t}\left[\sum_{k=1}^{K}c_{jk}\log p(x_j|\mu_k)\right] \\
&= \alpha_4 - \frac{\mu_t^2}{2\sigma^2} + \sum_{j=1}^{n}\mathbb{E}_{-t}\left[c_{jt}\log p(x_j|\mu_t)\right] \\
&= \alpha_4 - \frac{\mu_t^2}{2\sigma^2} + \sum_{j=1}^{n}\left(\mathbb{E}_{-t}\left[c_{jt}\right]\log p(x_j|\mu_t)\right)
\end{aligned}
\tag{A.26}
$$

The following numbered list contains comments, with each number corresponding to the order of the equality in derivation A.26 to which the comment applies (i.e., comment 1. is for the first equality, comment 2. is for the second equality, etc).

1. By equation 1.24. $\mathbb{E}_{-t}$ is the expectation with respect to all of the latent

variables in the **GMM** except s, distributed as $\frac{q(\boldsymbol{\mu},\mathbf{c})}{q_{\boldsymbol{\mu},t}(\mu_t)}$.

3. Where $\alpha_1 = \log\left(\alpha_{\boldsymbol{\mu},t}\right)$, given that $\alpha_{\boldsymbol{\mu},t}$ is a constant.
4. By equation 1.30.
5. Given the linearity of expectations, such as $\mathbb{E}_{-t}$.
6. Where $\alpha_2 = \alpha_1 + \mathbb{E}_{-t}\left[\sum_{i=1}^{n}\left(\log p(c_i)\right)\right]$. As the term $\mathbb{E}_{-t}\left[\sum_{i=1}^{n}\left(\log p(c_i)\right)\right]$ does not depend on μ_t, then it is a constant term (all of the latent variables in this term, none of which are μ_t, are held fixed).
7. Given the linearity of $\mathbb{E}_{-t}$.
8. Breaking away the term where $k = t$ in the summation for $\mathbb{E}_{-t}\left[\log p(\mu_k)\right]$, given that this summation includes a term for t.
9. As the expression $\log p(\mu_t)$ only depends on μ_t, and thus not on any of the variables that $\mathbb{E}_{-t}$ is with respect to, then $\mathbb{E}_{-t}\left[\log p(\mu_t)\right] = \log\left(p(\mu_t)\right)$.
10. Where $\alpha_3 = \alpha_2 + \sum_{k=1,k\neq t}^{K}\mathbb{E}_{-t}\left[\log p(\mu_k)\right]$. As the summation $\sum_{k=1,k\neq t}^{K}\mathbb{E}_{-t}\left[\log p(\mu_k)\right]$ has been formulated with μ_t having been broken away, then this is a constant term in this context.
11. By distribution 1.25, it can be deduced that $p(\mu_t) = \frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{1}{2}\left(\frac{\mu_t}{\sigma}\right)^2}$. By equation A.18, it can also be deduced that $p(x_j|\boldsymbol{\mu}, c_j) = \prod_{k=1}^{K} p(x_j|\mu_k)^{c_{jk}}$.
13. Where $\alpha_4 = \alpha_3 + \log\left(\frac{1}{\sigma\sqrt{2\pi}}\right)$. As σ is a hyperparameter, then $\log\left(\frac{1}{\sigma\sqrt{2\pi}}\right)$ can effectively be treated as a constant in the **CAVI** algorithm.
14. As we are considering the case where $k = t$, then the c_{kj} values will be equal to 1 only when $k = t$, and be equal to 0 otherwise. Therefore $\sum_{k=1}^{K} c_{jk}\log p(x_j|\mu_k) = c_{jt}\log p(x_j|\mu_t)$.
15. As the x_j variables are observations, then they can be treated as constants (they are fixed). Furthermore, as μ_t is only with respect to the latent variable t, that the expectation $\mathbb{E}_{-t}$ is not with respect to, then it can be deduced that $\mathbb{E}_{-t}\left[c_{jt}\log p(x_j|\mu_t)\right] = \mathbb{E}_{-t}\left[c_{jt}\right]\log p(x_j|\mu_t)$.

$$
\begin{aligned}
\log\left(q_{\mu,t}(\mu_t)\right) &= \alpha_4 - \frac{\mu_t^2}{2\sigma^2} + \sum_{j=1}^{n}\left(\mathbb{E}_{-t}\left[c_{jt}\right]\log p(x_j|\mu_t)\right) \\
&= \alpha_4 - \frac{\mu_t^2}{2\sigma^2} + \sum_{j=1}^{n}\left(\phi_{j,t}\log\left(\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}(x_j-\mu_t)^2}\right)\right) \\
&= \alpha_4 - \frac{\mu_t^2}{2\sigma^2} + \sum_{j=1}^{n}\left(\phi_{j,t}\left[\log\left(\frac{1}{\sqrt{2\pi}}\right) - \frac{1}{2}(x_j-\mu_t)^2\right]\right) \\
&= \alpha_4 - \frac{\mu_t^2}{2\sigma^2} + \sum_{j=1}^{n}\left(\phi_{j,t}\log\left(\frac{1}{\sqrt{2\pi}}\right) - \frac{1}{2}\phi_{j,t}(x_j-\mu_t)^2\right) \\
&= \alpha_4 - \frac{\mu_t^2}{2\sigma^2} + \sum_{j=1}^{n}\left(\phi_{j,t}\log\left(\frac{1}{\sqrt{2\pi}}\right)\right) + \sum_{j=1}^{n}\left(-\frac{1}{2}\phi_{j,t}(x_j-\mu_t)^2\right) \\
&= \alpha_5 - \frac{\mu_t^2}{2\sigma^2} + \sum_{j=1}^{n}\left(-\frac{1}{2}\phi_{j,t}(x_j-\mu_t)^2\right) \\
&= \alpha_5 - \frac{\mu_t^2}{2\sigma^2} + \sum_{j=1}^{n}\left(-\frac{1}{2}\phi_{j,t}\left(x_j^2 - 2x_j\mu_t + \mu_t^2\right)\right) \\
&= \alpha_5 - \frac{\mu_t^2}{2\sigma^2} + \sum_{j=1}^{n}\left(-\frac{1}{2}\phi_{j,t}x_j^2 + \phi_{j,t}x_j\mu_t - \frac{1}{2}\phi_{j,t}\mu_t^2\right) \\
&= \alpha_5 - \frac{1}{2\sigma^2}\mu_t^2 - \frac{1}{2}\sum_{j=1}^{n}\left(\phi_{j,t}x_j^2\right) + \mu_t\sum_{j=1}^{n}\left(\phi_{j,t}x_j\right) - \frac{1}{2}\mu_t^2\sum_{j=1}^{n}\left(\phi_{j,t}\right) \\
&= \alpha_5 + \left(-\frac{1}{2\sigma^2} - \frac{1}{2}\sum_{j=1}^{n}\phi_{j,t}\right)\mu_t^2 + \left(\sum_{j=1}^{n}\phi_{j,t}x_j\right)\mu_t + \left(-\frac{1}{2}\sum_{j=1}^{n}\phi_{j,t}x_j^2\right) \\
&= \alpha_6 - \frac{1}{2}\left(\frac{1}{\sigma^2} + \sum_{j=1}^{n}\phi_{j,t}\right)\mu_t^2 + \left(\sum_{j=1}^{n}\phi_{j,t}x_j\right)\mu_t
\end{aligned}
\tag{A.27}
$$

The following numbered list contains comments, with each number corresponding to the order of the equality in derivation A.27 to which the comment applies (i.e., comment 1. is for the first equality, comment 2. is for the second equality, etc).

1. Given the equality of the first and last expressions in derivation A.26.
2. By equation A.19, it can thus be deduced that $p(x_j|\mu_t) = \frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}(x_j-\mu_t)^2}$. Furthermore, we have $\mathbb{E}_{-t}\left[c_{jt}\right] = \phi_{j,t}$: Considering all of the latent variables of the **GMM** excluding t, we would expect the t^{th} element of the one-hot vector c_j to take, as an average, the probability of setting that element as the 1 in the categorical distribution for $q_{\mathbf{c},j}(c_j)$ (as we do not have the true posterior, see equation 1.31) - i.e., $\phi_{j,t}$.
6. Where $\alpha_5 = \alpha_4 + \sum_{j=1}^{n}\left(\phi_{j,t}\log\left(\frac{1}{\sqrt{2\pi}}\right)\right)$. As none of the terms in the summand $\phi_{j,t}\log\left(\frac{1}{\sqrt{2\pi}}\right)$ depend directly on the latent variable μ_t, then $\sum_{j=1}^{n}\left(\phi_{j,t}\log\left(\frac{1}{\sqrt{2\pi}}\right)\right)$ can be treated as a constant (terms that are not μ_t are fixed in place by the **CAVI** algorithm).
10. Grouping by the 0^{th}, 1^{st} and 2^{nd} powers of μ_t.

11. Where $\alpha_6 = \alpha_5 - \frac{1}{2}\sum_{j=1}^{n}\phi_{j,t}x_j^2$. As none of the terms in the summand $\phi_{j,t}x_j^2$ depend directly on the latent variable μ_t, then $\frac{1}{2}\sum_{j=1}^{n}\phi_{j,t}x_j^2$ can be treated as a constant.

For the final derivation involving the surrogate posterior factor type $q_{\mu,k}(\mu_k)$ (derivation A.29), in obtaining an expression for $q_{\mu,t}(\mu_t)$, let the following intermediary simplifications be made:

$$\beta_1 = \frac{1}{\sigma^2} + \sum_{j=1}^{n}\phi_{j,t} \quad \text{and} \quad \beta_2 = \sum_{j=1}^{n}\phi_{j,t}x_j \tag{A.28}$$

Therefore:

$$\begin{aligned}
q_{\mu,t}(\mu_t) &= \exp\left(\alpha_6 - \frac{1}{2}\left(\frac{1}{\sigma^2} + \sum_{j=1}^{n}\phi_{j,t}\right)\mu_t^2 + \left(\sum_{j=1}^{n}\phi_{j,t}x_j\right)\mu_t\right)\\
&= \exp\left(\alpha_6 - \frac{1}{2}\beta_1\mu_t^2 + \beta_2\mu_t\right)\\
&= \exp\left(\alpha_6 - \frac{1}{2}\beta_1\left(\mu_t^2 - \frac{2\beta_2}{\beta_1}\mu_t\right)\right)\\
&= \exp\left(\alpha_6 - \frac{1}{2}\beta_1\left(\left(\mu_t - \frac{\beta_2}{\beta_1}\right)^2 - \frac{\beta_2^2}{\beta_1^2}\right)\right)\\
&= \exp\left(\alpha_6 - \frac{1}{2}\beta_1\left(\mu_t - \frac{\beta_2}{\beta_1}\right)^2 + \frac{\beta_2^2}{2\beta_1}\right)\\
&= \exp\left(\alpha_6 + \frac{\beta_2^2}{2\beta_1}\right) * \exp\left(-\frac{1}{2}\left(\beta_1^{1/2}\left(\mu_t - \frac{\beta_2}{\beta_1}\right)\right)^2\right)\\
&= \alpha_7 * \exp\left(-\frac{1}{2}\left(\frac{\mu_t - \beta_1^{-1}\beta_2}{\beta_1^{-1/2}}\right)^2\right)
\end{aligned} \tag{A.29}$$

Given the 'numbered list of comments' convention used for derivations A.26 and A.27 above, now applied to derivation A.29, we have:

1. Given the equality of the first and last expressions in derivation A.27.
2. By the intermediary simplifications summarised in A.28.
4. Completing the square for $\mu_t^2 - \frac{2\beta_2}{\beta_1}\mu_t$.
7. Where $\alpha_7 = \exp\left(\alpha_6 + \frac{\beta_2^2}{2\beta_1}\right)$. Note that, by inspection of the definitions of β_1 and β_2 in equations A.28, neither β_1 nor β_2 depend directly on μ_t. Therefore, as none of the terms in the exponent $\alpha_6 + \frac{\beta_2^2}{2\beta_1}$ depend directly on the latent variable μ_t, then $\exp\left(\alpha_6 + \frac{\beta_2^2}{2\beta_1}\right)$ can be treated as a constant.

By equation 1.33, we thus know that $q_{\mu,t}(\mu_t) = \mathcal{N}(\mu_t; m_t, s_t^2)$. By inspection of the final equality in derivation A.29, noting that the final expression takes the standard form of a univariate Gaussian density, it can hence be deduced that the mean and standard deviation of the normal distribution for $q_{\mu,t}(\mu_t)$ are $m_t = \beta_1^{-1}\beta_2$ and $s_t = \beta_1^{-1/2}$, respectively. Substituting back for the intermediary simplifications β_1 and β_2, using equations A.28, we therefore have:

$$m_t = \frac{\sum_{j=1}^{n} \phi_{j,t} x_j}{\frac{1}{\sigma^2} + \sum_{j=1}^{n} \phi_{j,t}} \quad \text{and} \quad s_t = \left(\frac{1}{\sigma^2} + \sum_{j=1}^{n} \phi_{j,t} \right)^{-1/2} \tag{A.30}$$

As t represents a particular, yet arbitrarily chosen, value of k, then the derivations outlined above generalise for a generic indexing of k. Therefore, a complete generalisation of the results A.30 are summarised by the following equations:

$$m_k = \frac{\sum_{i=1}^{n} \phi_{i,k} x_i}{\frac{1}{\sigma^2} + \sum_{i=1}^{n} \phi_{i,k}} \quad \text{and} \quad s_k = \left(\frac{1}{\sigma^2} + \sum_{i=1}^{n} \phi_{i,k} \right)^{-1/2} \tag{A.31}$$

(these equations are given as equations 1.34 in section 1.8).

A.8 First and Second Moment Equivalents of the Latent Variable μ_k in the GMM

The first and second moments of the latent variable μ_k, in question, that we are seeking to find the equivalents for, are $\mathbb{E}_{-i}\left[\mu_k\right]$ and $\mathbb{E}_{-i}\left[\mu_k^2\right]$, respectively. As defined previously, $\mathbb{E}_{-i}$ is the expectation with respect to all of the latent variables in the **GMM** except i, which is in $\{1, 2, \ldots, n\}$ (i.e., $\{\mu_1, \mu_2, \ldots, \mu_K, c_1, c_2, \ldots, c_{i-1}, c_{i+1}, \ldots, c_n\}$), distributed as $\frac{q(\boldsymbol{\mu}, \mathbf{c})}{q_{\mathbf{c},i}(c_i)}$.

The latest information that we have at any particular moment in time regarding the distribution of μ_k, given any particular value of k in $\{1, 2, \ldots, K\}$ and any presented set of observations $\mathbf{x}$, is summarised by the probability density function $q_{\boldsymbol{\mu},k}(\mu_k)$, given the values of its variational parameters at that particular moment in time.

By equation 1.33, it can thus be deduced that $\mathbb{E}[\mu_k] = m_k$ (as the Gaussian distribution is symmetric, with mean m_k). As this distribution for μ_k only depends directly on the latent variable μ_k itself, then a removal of the latent variable c_i from the set of variables that the general expectation $\mathbb{E}$ is with respect to - which will result in $\mathbb{E}_{-i}$ - will not affect the result of the expectation in this case. Therefore:

$$\mathbb{E}_{-i}[\mu_k] = m_k \tag{A.32}$$

Furthermore, by equation 1.33, the variance of the distribution for μ_k is $\text{Var}(\mu_k) = s_k^2$. Hence:

$$\begin{aligned} \text{Var}(\mu_k) &= \mathbb{E}_{-i}\left[\mu_k^2\right] - \left(\mathbb{E}_{-i}[\mu_k]\right)^2 \\ s_k^2 &= \mathbb{E}_{-i}\left[\mu_k^2\right] - (m_k)^2 \quad \text{(By equation A.32)} \\ \mathbb{E}_{-i}\left[\mu_k^2\right] &= s_k^2 + m_k^2 \end{aligned} \tag{A.33}$$

A.9 Derivation of the ELBO for the GMM

This appendix section provides a full derivation of the evidence lower bound - or **ELBO** - for the **GMM** in section 1.8.

First, consider the following derivation:

$$\begin{aligned}
\mathrm{ELBO}[q] &= \mathbb{E}_{\boldsymbol{\mu},\mathbf{c}\sim q(\boldsymbol{\mu},\mathbf{c})}\left[\log\left(\frac{p(\boldsymbol{\mu},\mathbf{c},\mathbf{x})}{q(\boldsymbol{\mu},\mathbf{c})}\right)\right] \\
&= \mathbb{E}\left[\log\left(p(\boldsymbol{\mu},\mathbf{c},\mathbf{x})\right) - \log\left(q(\boldsymbol{\mu},\mathbf{c})\right)\right] \\
&= \mathbb{E}\left[\log\left(\prod_{k=1}^{K} p(\mu_k)\prod_{i=1}^{n} p(c_i)p(x_i|\boldsymbol{\mu},c_i)\right) - \log\left(\prod_{k=1}^{K} q_{\boldsymbol{\mu},k}(\mu_k)\prod_{i=1}^{n} q_{\mathbf{c},i}(c_i)\right)\right] \\
&= \mathbb{E}\left[\log\left(\prod_{k=1}^{K} p(\mu_k)\prod_{i=1}^{n} p(c_i)p(x_i|\boldsymbol{\mu},c_i)\right)\right] \\
&\qquad - \mathbb{E}\left[\log\left(\prod_{k=1}^{K} q_{\boldsymbol{\mu},k}(\mu_k)\prod_{i=1}^{n} q_{\mathbf{c},i}(c_i)\right)\right] \\
&= \mathbb{E}\left[\log\left(\prod_{k=1}^{K} p(\mu_k)\right) + \log\left(\prod_{i=1}^{n} p(c_i)p(x_i|\boldsymbol{\mu},c_i)\right)\right] \\
&\qquad - \mathbb{E}\left[\log\left(\prod_{k=1}^{K} q_{\boldsymbol{\mu},k}(\mu_k)\right) + \log\left(\prod_{i=1}^{n} q_{\mathbf{c},i}(c_i)\right)\right] \\
&= \mathbb{E}\left[\sum_{k=1}^{K}\left(\log p(\mu_k)\right) + \sum_{i=1}^{n}\left(\log p(c_i)p(x_i|\boldsymbol{\mu},c_i)\right)\right] \\
&\qquad - \mathbb{E}\left[\sum_{k=1}^{K}\left(\log q_{\boldsymbol{\mu},k}(\mu_k)\right) + \sum_{i=1}^{n}\left(\log q_{\mathbf{c},i}(c_i)\right)\right] \\
&= \mathbb{E}\left[\sum_{k=1}^{K}\left(\log p(\mu_k)\right) + \sum_{i=1}^{n}\left(\log p(c_i)\right) + \sum_{i=1}^{n}\left(\log p(x_i|\boldsymbol{\mu},c_i)\right)\right] \\
&\qquad - \mathbb{E}\left[\sum_{k=1}^{K}\left(\log q_{\boldsymbol{\mu},k}(\mu_k)\right) + \sum_{i=1}^{n}\left(\log q_{\mathbf{c},i}(c_i)\right)\right] \\
&= \mathbb{E}\left[\sum_{k=1}^{K}\left(\log p(\mu_k)\right)\right] + \mathbb{E}\left[\sum_{i=1}^{n}\left(\log p(c_i)\right)\right] + \mathbb{E}\left[\sum_{i=1}^{n}\left(\log p(x_i|\boldsymbol{\mu},c_i)\right)\right] \\
&\qquad - \mathbb{E}\left[\sum_{k=1}^{K}\left(\log q_{\boldsymbol{\mu},k}(\mu_k)\right)\right] - \mathbb{E}\left[\sum_{i=1}^{n}\left(\log q_{\mathbf{c},i}(c_i)\right)\right] \\
&= \sum_{k=1}^{K}\mathbb{E}\left[\log p(\mu_k)\right] + \sum_{i=1}^{n}\mathbb{E}\left[\log p(c_i)\right] + \sum_{i=1}^{n}\mathbb{E}\left[\log p(x_i|\boldsymbol{\mu},c_i)\right] \\
&\qquad - \sum_{k=1}^{K}\mathbb{E}\left[\log q_{\boldsymbol{\mu},k}(\mu_k)\right] - \sum_{i=1}^{n}\mathbb{E}\left[\log q_{\mathbf{c},i}(c_i)\right]
\end{aligned} \tag{A.34}$$

The following numbered list contains comments, with each number corresponding to the order of the equality in derivation A.34 to which the comment applies (i.e., comment 1. is for the first equality, comment 2. is for the second equality, etc).

1. By the definition of the **ELBO**, given equation 1.16, in the context of the **GMM**.
2. Henceforth in this derivation, it will be assumed that the expectations are given with respect to the latent variables in the **GMM**, distributed as the surrogate posterior. That is, $\mathbb{E} = \mathbb{E}_{\boldsymbol{\mu},\mathbf{c}\sim q(\boldsymbol{\mu},\mathbf{c})}$.
3. By equations 1.28 and 1.29.
4. Given the linearity of expectations.
8. Given the linearity of expectations.
9. Given the linearity of expectations.

Now let derivations for each of $\mathbb{E}\left[\log p(\mu_k)\right]$, $\mathbb{E}\left[\log p(c_i)\right]$, $\mathbb{E}\left[\log p(x_i|\boldsymbol{\mu}, c_i)\right]$, $\mathbb{E}\left[\log q_{\boldsymbol{\mu},k}(\mu_k)\right]$ and $\mathbb{E}\left[\log q_{\mathbf{c},i}(c_i)\right]$ be established for the **ELBO** of the **GMM** (see derivation A.34), based on what has already been deduced in section 1.8 and its associated appendix sections, leading up to this appendix section (i.e., A.5, A.6, A.7 & A.8). In keeping with the previous notational conventions, let c_{ik} denote the value of the k^{th} element of the one-hot vector c_i, which will be either a 0 or a 1.

Given that $q_{\mathbf{c},i}(c_i)$ takes the form of a categorical distribution, as has already been established via equation 1.31, where each ϕ_{ik} value indicates the probability of the 1 in the one-hot vector c_i being at the k^{th} position according to the distribution $q_{\mathbf{c},i}$, then the probability density $q_{\mathbf{c},i}(c_i)$ can be expressed as the following weighted product:

$$q_{\mathbf{c},i}(c_i) = \prod_{k=1}^{K} {\phi_{ik}}^{c_{ik}} \tag{A.35}$$

Let the following five derivations now be considered:

$$\begin{aligned}
\mathbb{E}\left[\log p(\mu_k)\right] &= \mathbb{E}\left[\log\left(\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{1}{2}\left(\frac{\mu_k}{\sigma}\right)^2}\right)\right] \\
&= \mathbb{E}\left[\log\left(\frac{1}{\sigma\sqrt{2\pi}}\right) - \frac{1}{2}\left(\frac{\mu_k}{\sigma}\right)^2\right] \\
&= \mathbb{E}\left[\log\left(\sqrt{\frac{1}{2\pi\sigma^2}}\right)\right] - \mathbb{E}\left[\frac{\mu_k^2}{2\sigma^2}\right] \\
&= \log\left(\sqrt{\frac{1}{2\pi\sigma^2}}\right) - \frac{\mathbb{E}\left[\mu_k^2\right]}{2\sigma^2} \\
&= \log\left(\left(2\pi\sigma^2\right)^{-1/2}\right) - \frac{m_k^2 + s_k^2}{2\sigma^2} \\
&= -\frac{1}{2}\log\left(2\pi\sigma^2\right) - \frac{1}{2\sigma^2}\left(m_k^2 + s_k^2\right)
\end{aligned} \tag{A.36}$$

$$\begin{aligned}
\mathbb{E}\left[\log p(c_i)\right] &= \mathbb{E}\left[\log\left(\frac{1}{K}\right)\right] \\
&= \mathbb{E}\left[-\log K\right] \\
&= -\log K
\end{aligned} \tag{A.37}$$

$$
\begin{aligned}
\mathbb{E}\left[\log p(x_i|\boldsymbol{\mu}, c_i)\right] &= \mathbb{E}\left[\log\left(\prod_{k=1}^{K} p(x_i|\mu_k)^{c_{ik}}\right)\right] \\
&= \mathbb{E}\left[\sum_{k=1}^{K} \log\left(p(x_i|\mu_k)^{c_{ik}}\right)\right] \\
&= \mathbb{E}\left[\sum_{k=1}^{K} c_{ik} \log p(x_i|\mu_k)\right] \\
&= \mathbb{E}\left[\sum_{k=1}^{K} c_{ik} \log\left(\frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(x_i-\mu_k)^2}\right)\right] \\
&= \mathbb{E}\left[\sum_{k=1}^{K} c_{ik} \left(\log\left(\frac{1}{\sqrt{2\pi}}\right) - \frac{1}{2}(x_i-\mu_k)^2\right)\right] \\
&= \mathbb{E}\left[\sum_{k=1}^{K} \left(c_{ik} \log\left(\frac{1}{\sqrt{2\pi}}\right)\right) - \sum_{k=1}^{K}\left(\frac{1}{2} c_{ik}(x_i-\mu_k)^2\right)\right] \\
&= \mathbb{E}\left[\sum_{k=1}^{K} \left(-\frac{1}{2} c_{ik} \log(2\pi)\right)\right] - \mathbb{E}\left[\sum_{k=1}^{K}\left(\frac{1}{2} c_{ik}\left(x_i^2 - 2x_i\mu_k + \mu_k^2\right)\right)\right] \\
&= \mathbb{E}\left[-\frac{1}{2}\log(2\pi)\sum_{k=1}^{K} c_{ik}\right] - \mathbb{E}\left[\frac{1}{2}\sum_{k=1}^{K}\left(x_i^2 c_{ik} - 2x_i\mu_k c_{ik} + \mu_k^2 c_{ik}\right)\right] \\
&= -\frac{1}{2}\log(2\pi)\mathbb{E}\left[\sum_{k=1}^{K} c_{ik}\right] - \frac{1}{2}\mathbb{E}\left[\sum_{k=1}^{K}\left(x_i^2 c_{ik} - 2x_i\mu_k c_{ik} + \mu_k^2 c_{ik}\right)\right] \\
&= -\frac{1}{2}\log(2\pi)\sum_{k=1}^{K} \mathbb{E}\left[c_{ik}\right] - \frac{1}{2}\sum_{k=1}^{K}\mathbb{E}\left[x_i^2 c_{ik} - 2x_i\mu_k c_{ik} + \mu_k^2 c_{ik}\right] \\
&= -\frac{1}{2}\log(2\pi)\sum_{k=1}^{K} \phi_{i,k} - \frac{1}{2}\sum_{k=1}^{K}\mathbb{E}\left[x_i^2 c_{ik}\right] - \frac{1}{2}\sum_{k=1}^{K}\mathbb{E}\left[-2x_i\mu_k c_{ik}\right] \\
&\qquad - \frac{1}{2}\sum_{k=1}^{K}\mathbb{E}\left[\mu_k^2 c_{ik}\right] \\
&= -\frac{1}{2}\log(2\pi)\sum_{k=1}^{K} \phi_{i,k} - \frac{1}{2}x_i^2\sum_{k=1}^{K}\mathbb{E}\left[c_{ik}\right] + x_i\sum_{k=1}^{K}\mathbb{E}[\mu_k]\mathbb{E}[c_{ik}] \\
&\qquad - \frac{1}{2}\sum_{k=1}^{K}\mathbb{E}[\mu_k^2]\mathbb{E}[c_{ik}] \\
&= -\frac{1}{2}\log(2\pi)\sum_{k=1}^{K} \phi_{i,k} - \frac{1}{2}x_i^2\sum_{k=1}^{K}\phi_{i,k} + x_i\sum_{k=1}^{K} m_k\phi_{i,k} \\
&\qquad - \frac{1}{2}\sum_{k=1}^{K}\left(m_k^2 + s_k^2\right)\phi_{i,k} \\
&= -\frac{1}{2}\log(2\pi) - \frac{1}{2}x_i^2 + x_i\sum_{k=1}^{K}\phi_{i,k}m_k - \frac{1}{2}\sum_{k=1}^{K}\phi_{i,k}\left(m_k^2 + s_k^2\right)
\end{aligned}
\tag{A.38}
$$

$$
\begin{aligned}
\mathbb{E}\left[\log q_{\mu,k}(\mu_k)\right] &= \mathbb{E}\left[\log\left(\frac{1}{s_k\sqrt{2\pi}}e^{-\frac{1}{2}\left(\frac{\mu_k-m_k}{s_k}\right)^2}\right)\right] \\
&= \mathbb{E}\left[\log\left(\frac{1}{s_k\sqrt{2\pi}}\right)-\frac{1}{2}\left(\frac{\mu_k-m_k}{s_k}\right)^2\right] \\
&= \mathbb{E}\left[\log\left(\sqrt{\frac{1}{2\pi s_k^2}}\right)-\left(\frac{\mu_k^2-2\mu_k m_k+m_k^2}{2s_k^2}\right)\right] \\
&= \mathbb{E}\left[-\frac{1}{2}\log 2\pi s_k^2\right]-\mathbb{E}\left[\frac{\mu_k^2-2\mu_k m_k+m_k^2}{2s_k^2}\right] \\
&= -\frac{1}{2}\log\left(2\pi s_k^2\right)-\frac{\mathbb{E}[\mu_k^2]-2m_k\mathbb{E}[\mu_k]+m_k^2}{2s_k^2} \\
&= -\frac{1}{2}\log\left(2\pi s_k^2\right)-\frac{(m_k^2+s_k^2)-2m_k(m_k)+m_k^2}{2s_k^2} \\
&= -\frac{1}{2}\log\left(2\pi s_k^2\right)-\frac{m_k^2+s_k^2-2m_k^2+m_k^2}{2s_k^2} \\
&= -\frac{1}{2}\log\left(2\pi s_k^2\right)-\frac{s_k^2}{2s_k^2} \\
&= -\frac{1}{2}\log\left(2\pi s_k^2\right)-\frac{1}{2}
\end{aligned}
\tag{A.39}
$$

$$
\begin{aligned}
\mathbb{E}\left[\log q_{\mathbf{c},i}(c_i)\right] &= \mathbb{E}\left[\log\left(\prod_{k=1}^{K}{\phi_{i,k}}^{c_{ik}}\right)\right] \\
&= \mathbb{E}\left[\sum_{k=1}^{K}\log\left({\phi_{i,k}}^{c_{ik}}\right)\right] \\
&= \mathbb{E}\left[\sum_{k=1}^{K}c_{ik}\log\phi_{i,k}\right] \\
&= \sum_{k=1}^{K}\mathbb{E}\left[c_{ik}\log\phi_{i,k}\right] \\
&= \sum_{k=1}^{K}\mathbb{E}\left[c_{ik}\right]\log\phi_{i,k} \\
&= \sum_{k=1}^{K}\phi_{i,k}\log\phi_{i,k}
\end{aligned}
\tag{A.40}
$$

The following list contains comments relevant to the 5 derivations above: A.36, A.37, A.38, A.39 & A.40.

i The expectations in each of the derivations are with respect to the latent variables in the **GMM**, distributed as the surrogate posterior, just as for derivation A.34. That is: $\mathbb{E} = \mathbb{E}_{\mu,\mathbf{c}\sim q(\mu,\mathbf{c})}$.

ii The expectation function $\mathbb{E}$ is linear.

iii As $\mathbb{E}$ is only with respect to the latent variables - μ_k and c_i (each c_i is expressed as a series of binary c_{ik} variables), for $k \in \{1, 2, \ldots, K\}$ and $i \in \{1, 2, \ldots, n\}$ - then all terms that only involve the other variables (m_k, s_k, $\phi_{i,k}$, σ & K, for $k \in \{1, 2, \ldots, K\}$ and $i \in \{1, 2, \ldots, n\}$) in the derivations above can be linearly taken out of the expectation function $\mathbb{E}$.

iv Given distribution 1.25, and equations 1.33 and A.19, and the general closed-form definition of a univariate Gaussian density, it can thus be deduced that: $p(\mu_k) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\mu_k}{\sigma}\right)^2}$, $q_{\boldsymbol{\mu},k}(\mu_k) = \frac{1}{s_k\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\mu_k - m_k}{s_k}\right)^2}$ and $p(x_i|\mu_k) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(x_i - \mu_k)^2}$, respectively.

v Given that the "equal ϕ value" initialisation outlined in appendix section A.6 still holds in this context, as this is an initialisation of the **CAVI** algorithm for which the **ELBO** is being derived in this appendix section, then equation A.17 applies: That is, $p(c_i) = \frac{1}{K}$ for all $i \in \{1, 2, \ldots, n\}$.

vi Given equation A.18, and the fact that s is a specific but arbitrary value of i in this equation, it can thus be deduced that, in general: $p(x_i|\boldsymbol{\mu}, c_i) = \prod_{k=1}^{K} p(x_i|\mu_k)^{c_{ik}}$ for all $i \in \{1, 2, \ldots, n\}$.

vii Equation A.35 has been substituted for the first line in derivation A.40.

viii By equations A.32 and A.33, it can thus be deduced that $\mathbb{E}[\mu_k] = \mathbb{E}_i[\mathbb{E}_{-i}[\mu_k]] = \mathbb{E}_i[m_k] = m_k$ and $\mathbb{E}[\mu_k^2] = \mathbb{E}_i[\mathbb{E}_{-i}[\mu_k^2]] = \mathbb{E}_i[s_k^2 + m_k^2] = s_k^2 + m_k^2$, respectively, where $\mathbb{E}_{-i}$ takes the same definition as for equations A.32 and A.33, and $\mathbb{E}_i = \mathbb{E}_{c_i \sim q_{\mathbf{c},i}(c_i)}$, each for $i \in \{1, 2, \ldots, n\}$.

ix Given the outline provided in appendix section A.7, we have $\mathbb{E}_{-k}\left[c_{ik}\right] = \phi_{i,k}$ for some $i \in \{1, 2, \ldots, n\}$ and some $k \in \{1, 2, \ldots, K\}$: Considering all of the latent variables of the **GMM** excluding k, we would expect the k^{th} element of the one-hot vector c_i to take, as an average, the probability of setting that element as the 1 in the categorical distribution for $q_{\mathbf{c},i}(c_i)$ (as we do not have the true posterior, see equation 1.31) - i.e., $\phi_{i,k}$. Therefore $\mathbb{E}[c_{ik}] = \mathbb{E}_k[\mathbb{E}_{-k}[c_{ik}]] = \mathbb{E}_k[\phi_{i,k}] = \phi_{i,k}$, where $\mathbb{E}_{-k}$ takes the same definition as has been outlined previously for this notation, and $\mathbb{E}_k = \mathbb{E}_{\mu_k \sim q_{\mu,k}(\mu_k)}$, each for $k \in \{1, 2, \ldots, K\}$.

x As the value of μ_k, as a result of sampling from distribution 1.25, is independent of the variable's index k, then we can assume that the latent variables μ_k and c_{ik} are uncorrelated. Hence $\mathbb{E}[\mu_k c_{ik}] = \mathbb{E}[\mu_k]\mathbb{E}[c_{ik}]$ and $\mathbb{E}[\mu_k^2 c_{ik}] = \mathbb{E}[\mu_k^2]\mathbb{E}[c_{ik}]$. These inferences are used in derivation A.38.

Therefore, a concise closed-form definition of the **ELBO** for the **GMM** in section 1.8, given in terms of the observations $\mathbf{x}$, the variational parameters - m_k, s_k and $\phi_{i,k}$, for $i \in \{1, 2, \ldots, n\}$ and $k \in \{1, 2, \ldots, K\}$ - and the hyperparameters - σ and K - is:

$$\begin{aligned}\text{ELBO}[q] = &\sum_{k=1}^{K} \mathbb{E}\left[\log p(\mu_k)\right] + \sum_{i=1}^{n} \mathbb{E}\left[\log p(c_i)\right] + \sum_{i=1}^{n} \mathbb{E}\left[\log p(x_i|\boldsymbol{\mu}, c_i)\right] \\ &- \sum_{k=1}^{K} \mathbb{E}\left[\log q_{\boldsymbol{\mu},k}(\mu_k)\right] - \sum_{i=1}^{n} \mathbb{E}\left[\log q_{\mathbf{c},i}(c_i)\right]\end{aligned} \tag{A.41}$$

where:

$$
\begin{aligned}
\mathbb{E}\left[\log p(\mu_k)\right] &= -\frac{1}{2}\log\left(2\pi\sigma^2\right) - \frac{1}{2\sigma^2}\left(m_k^2 + s_k^2\right) \\
\mathbb{E}\left[\log p(c_i)\right] &= -\log K \\
\mathbb{E}\left[\log p(x_i|\boldsymbol{\mu}, c_i)\right] &= -\frac{1}{2}\log\left(2\pi\right) - \frac{1}{2}x_i^2 + x_i\sum_{k=1}^{K}\phi_{i,k}m_k - \frac{1}{2}\sum_{k=1}^{K}\phi_{i,k}\left(m_k^2 + s_k^2\right) \\
\mathbb{E}\left[\log q_{\boldsymbol{\mu},k}(\mu_k)\right] &= -\frac{1}{2}\log\left(2\pi s_k^2\right) - \frac{1}{2} \\
\mathbb{E}\left[\log q_{\mathbf{c},i}(c_i)\right] &= \sum_{k=1}^{K}\phi_{i,k}\log\phi_{i,k}
\end{aligned}
\tag{A.42}
$$

A.10 CAVI algorithm for the GMM in Python

A full programmatic demonstration of the execution of the **CAVI** algorithm for the **GMM** in section 1.8 is provided as follows, with $\epsilon = 0.001$, $\sigma = 0.5$, $K = 10$ and $n = 100$. This program is in Python, with the inclusion of the common third-party module NumPy (Numerical Python).

```python
import numpy as np

##########################################################

# A dataset, x_vector, is first generated using the Gaussian and
# categorical distributions of the GMM.

# Hyperparameters
sigma = 0.5         # Standard deviation of the set of mus
K = 10              # Number of mus
n = 100             # Number of cluster assignments / data points

# Sampling the K mu values via the appropriate Gaussian distribution,
# for the vector mu_vector, that will be used in the generation of the
# dataset x_vector.
mu_vector = np.random.normal(0, sigma, K)

# Sampling the n c values via the appropriate categorical distribution,
# for the mixture assignment vector c_vector, that will be used in the
# generation of the dataset x_vector.
c_vector = np.random.default_rng(seed=42).integers(low=0, high=K,
                           size=n)

# Initialising the generated dataset vector x_vector.
x_vector = []

# Generating each of the n data points in turn, for the dataset vector
# x_vector, via the appropriate Gaussian distribution.
for i in range(n):
    c_i_transpose_mu = mu_vector[c_vector[i]]
    x_i = np.random.normal(c_i_transpose_mu, 1, 1)[0]
    x_vector.append(x_i)

# Final GMM generated dataset.
x_vector = np.array(x_vector)

##########################################################
```

```
# CAVI Algorithm for the GMM

# We retain x_vector, and now assume that we do not know mu_vector or
# c_vector.
# We now try to approximate mu_vector and c_vector, via the surrogate
# variational parameters m_vector, s_vector and phi_matrix, given
# x_vector.

# Introducing and initialising the variational parameters of the
# surrogate posterior q.
m_vector = np.zeros(K)              # A K dimensional vector of zeros
# A K dimensional vector with elements all equal to ((sigma + 1)/2)
s_vector = np.ones(K) * ((sigma + 1)/2)
# An n by K matrix with elements all equal to 1/K
phi_matrix = np.full((n, K), (1/K))

# Small change used to determine convergence in CAVI
epsilon = 0.001


# Defining the callable function that will be used to update each of
# the phi values at each iteration of the CAVI algorithm.
def phi_update():
    # The vector row_exp_sum_vector will contain i sums, of the
    # exponentials of the elements in each of the i rows of
    # phi_matrix, for normalisation purposes.
    row_exp_sum_vector = np.array([])
    for i in range(n):
        row_exp_sum = 0.0
        for k in range(K):
            row_exp_sum += np.exp(phi_matrix[i, k])
        row_exp_sum_vector = np.append(row_exp_sum_vector, row_exp_sum)

    # Each of the elements in phi_matrix are then updated using the
    # definition of the Softmax function.
    for i in range(n):
        for k in range(K):
            softmax_argument = ((x_vector[i] * m_vector[k])
                 - ((1/2) * (((s_vector[k])**2) + ((m_vector[k])**2))))
            old_phi_matrix_value = phi_matrix[i, k]
            phi_matrix[i, k] = (np.exp(softmax_argument) /
                                            row_exp_sum_vector[i])
            # The normalisation vector row_exp_sum_vector is then
            # updated accordingly.
            row_exp_sum_vector[i] = (row_exp_sum_vector[i]
                              - old_phi_matrix_value + phi_matrix[i, k])
    return


# Defining the callable function that will be used to update each of
# the m_k and s_k values at each iteration of the CAVI algorithm.
def m_and_s_update():
    for k in range(K):
        # The summation needed in the calculation of m_k and s_k
        phi_sum = 0.0
        # The summation needed in the calculation of m_k
        phi_times_x_sum = 0.0

        for i in range(n):
            phi_sum += phi_matrix[i, k]
            phi_times_x_sum += (phi_matrix[i, k] * x_vector[i])

        m_vector[k] = phi_times_x_sum / ((1 / (sigma**2)) + phi_sum)
        s_vector[k] = ((1 / (sigma**2)) + phi_sum)**(-1/2)
    return


# Defining the callable function that will be used to compute the
# value of the ELBO at each iteration.
def ELBO():
    # Initialising each of the 5 summations that constitute the ELBO.
    # E denotes the appropriate expectation.
    sum_E_log_p_mu_k = 0.0
    sum_E_log_p_c_i = 0.0
    sum_E_log_p_x_i_given_mu_c_i = 0.0
    sum_E_log_q_mu_k = 0.0
    sum_E_log_q_c_i = 0.0

    for k in range(K):
        sum_E_log_p_mu_k += (((-1/2) * np.log(2 * np.pi * (sigma**2)))
                    + ((-1/(2*(sigma**2))) * ((m_vector[k]**2)
                        + (s_vector[k]**2))))
        sum_E_log_q_mu_k += (((-1/2)
                      * np.log(2 * np.pi * (s_vector[k]**2))) - (1/2))

    for i in range(n):
        # One of the summations needed in the calculation of
        # sum_E_log_p_x_i_given_mu_c_i.
        sum_phi_m_k = 0.0
        # One of the summations needed in the calculation of
        # sum_E_log_p_x_i_given_mu_c_i.
        sum_phi_m_k_2_plus_s_k_2 = 0.0
        # The summation needed in the calculation of sum_E_log_q_c_i.
        sum_phi_log_phi = 0.0
        for k in range(K):
            sum_phi_m_k += (phi_matrix[i][k] * m_vector[k])
            sum_phi_m_k_2_plus_s_k_2 += (phi_matrix[i][k]
                                * ((m_vector[k]**2) + (s_vector[k]**2)))
            sum_phi_log_phi += (phi_matrix[i][k]
                                         * np.log(phi_matrix[i][k]))

        sum_E_log_p_c_i += (-np.log(K))
        sum_E_log_p_x_i_given_mu_c_i += (((-1/2) * np.log(2*np.pi))
                    + ((-1/2) * (x_vector[i]**2))
                    + (x_vector[i] * sum_phi_m_k)
                    + ((-1/2) * sum_phi_m_k_2_plus_s_k_2))
        sum_E_log_q_c_i += sum_phi_log_phi

    # The expectation of the log of the joint model.
    ELBO_value = (sum_E_log_p_mu_k + sum_E_log_p_c_i
                                + sum_E_log_p_x_i_given_mu_c_i)
    # The expectation of the log of the surrogate posterior.
    ELBO_value -= (sum_E_log_q_mu_k + sum_E_log_q_c_i)

    return ELBO_value

# Initialising the ELBO prior and current values with arbitrary
# extreme values that will allow the algorithm to start.
prior_ELBO_value = 10
current_ELBO_value = 5

# The iterative loop for the CAVI algorithm.
while (abs(current_ELBO_value - prior_ELBO_value) >= epsilon):
    phi_update()        # Update the phi values
    m_and_s_update()    # Update the m and s values
    # With the variational parameters having been updated, we need to
    # re-calculate the value of the ELBO.
    prior_ELBO_value = current_ELBO_value
    current_ELBO_value = ELBO()
```

A.11 A Full Derivation of the Elliptical Standardisation Transformation, in the One-Dimensional Case, for ADVI

This appendix section is an extension and further elaboration of derivation 1.59, in section 1.11, in the specific case where $\boldsymbol{\xi}$ and $\boldsymbol{\eta}$ are one-dimensional (scalars): ξ and η.

As ξ and η are one-dimensional in the **ADVI** algorithm, then $\xi \sim \mathcal{N}(\mu, \sigma^2)$ for some $\mu \in \mathbb{R}$ and some $\sigma \in \mathbb{R}^+$, and $\eta \sim \mathcal{N}(0,1)$. The elliptical standardisation transformation function and its inverse are then defined by $\eta = S_{\boldsymbol{\phi}}(\xi) = \frac{\xi-\mu}{\sigma}$ and $\xi = S_{\boldsymbol{\phi}}^{-1}(\eta) = \eta\sigma + \mu$, respectively - see equations 1.56 and 1.58. As the surrogate posterior q models the distribution of ξ, and $\xi \sim \mathcal{N}(\mu, \sigma^2)$, then $q(\xi) = \mathcal{N}(\xi; \mu, \sigma^2)$.

In this appendix section, let the function f be defined as in section 1.11, in the case where $\boldsymbol{\xi}$ is one-dimensional:

$$f(\xi) = \log\left(p(\mathbf{x}|\tau^{-1}(\xi))p(\tau^{-1}(\xi))\right) + \log|\det \mathcal{J}_{\tau^{-1}}(\xi)| \tag{A.43}$$

Given that $\xi = S_{\boldsymbol{\phi}}^{-1}(\eta) = \eta\sigma + \mu$, it can thus be deduced that $\frac{d\xi}{d\eta} = \sigma$. Therefore $d\xi = \sigma d\eta$.

Hence:

$$\begin{aligned}
\mathbb{E}_{\xi\sim q(\xi)}[f(\xi)] &= \int_\xi \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\xi-\mu}{\sigma}\right)^2} f(\xi) d\xi \quad (\text{As } q(\xi) = \mathcal{N}(\xi;\mu,\sigma^2)) \\
&= \int_\eta \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\xi-\mu}{\sigma}\right)^2} f(\xi)\sigma d\eta \quad (\text{As } d\xi = \sigma d\eta) \\
&= \int_\eta \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\xi-\mu}{\sigma}\right)^2} f\left(S_{\boldsymbol{\phi}}^{-1}(\eta)\right)\sigma d\eta \quad (\text{As } \xi = S_{\boldsymbol{\phi}}^{-1}(\eta)) \\
&= \int_\eta \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\eta)^2} f\left(S_{\boldsymbol{\phi}}^{-1}(\eta)\right)\sigma d\eta \quad (\text{As } \eta = \frac{\xi-\mu}{\sigma}) \\
&= \int_\eta \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(\eta)^2} f\left(S_{\boldsymbol{\phi}}^{-1}(\eta)\right) d\eta \quad (\text{The } \sigma\text{s in the integrand cancel out.}) \\
&= \mathbb{E}_{\eta\sim\mathcal{N}(0,1)}\left[f\left(S_{\boldsymbol{\phi}}^{-1}(\eta)\right)\right]
\end{aligned} \tag{A.44}$$

The result of derivation A.44 is the same as for 1.59 in section 1.11, except that derivation A.44 has been derived via a more first principled basis, and is a specific one-dimensional case example of the broader result provided in derivation 1.59.

A.12 Derivations of Intermediary Components Required for the ADVI Algorithm in Section 1.11

The **ADVI** algorithm at the end of section 1.11 outlines full expressions of key intermediary components, required for the algorithm, that are derived from combinations of results that preceded the algorithm in the same section. The key intermediary components in question are: $ELBO[q]$, $\nabla_{\mu} ELBO[q]$, $\nabla_{\omega} ELBO[q]$ & $\nabla_{L} ELBO[q]$. This appendix section provides the derivations that led to each of these full expressions.

A full expression of $ELBO[q]$, which is required for its computation, is given by the result of the following derivation:

$$\begin{aligned}
ELBO[q] &= \mathbb{E}_{\xi \sim q(\xi)} \left[\log \left(p(\mathbf{x}|\tau^{-1}(\xi)) p(\tau^{-1}(\xi)) \right) + \log |\det \mathcal{J}_{\tau^{-1}}(\xi)| \right] \\
&\quad + \mathbb{H}_{\xi \sim q(\xi)}[q(\xi)] \\
&= \mathbb{E}_{\xi \sim q(\xi)} \left[\log \left(p(\mathbf{x}|\tau^{-1}(\xi)) p(\tau^{-1}(\xi)) \right) + \log |\det \mathcal{J}_{\tau^{-1}}(\xi)| \right] \\
&\quad + \frac{m}{2}(1 + \log(2\pi)) + \frac{1}{2} \log |\Sigma| \\
&= \mathbb{E}_{\eta \sim \mathcal{N}(\mathbf{0}, I)} \left[\log \left(p \left(\mathbf{x} | \tau^{-1} \left(S_{\phi}^{-1}(\eta) \right) \right) p \left(\tau^{-1} \left(S_{\phi}^{-1}(\eta) \right) \right) \right) \right] \\
&\quad + \mathbb{E}_{\eta \sim \mathcal{N}(\mathbf{0}, I)} \left[\log \left| \det \mathcal{J}_{\tau^{-1}} \left(S_{\phi}^{-1}(\eta) \right) \right| \right] \\
&\quad + \frac{m}{2}(1 + \log(2\pi)) + \frac{1}{2} \log |\Sigma| \\
&\approx \frac{1}{S} \sum_{s=1}^{S} \left(\log \left(p \left(\mathbf{x} | \tau^{-1} \left(S_{\phi}^{-1}(\eta_s) \right) \right) p \left(\tau^{-1} \left(S_{\phi}^{-1}(\eta_s) \right) \right) \right) \right) \\
&\quad + \frac{1}{S} \sum_{s=1}^{S} \left(\log \left| \det \mathcal{J}_{\tau^{-1}} \left(S_{\phi}^{-1}(\eta_s) \right) \right| \right) \\
&\quad + \frac{m}{2}(1 + \log(2\pi)) + \frac{1}{2} \log |\Sigma|
\end{aligned} \tag{A.45}$$

i The first equality follows from equation 1.49.

ii The second equality follows from equation 1.50, where $m \in \mathbb{N}$ is the dimensionality of ξ. That is, $\xi \in \mathbb{R}^m$.

iii The third equality follow from derivation 1.59 and by the linearity of expectations.

iv The fourth and final (approximate) equality follows from approximation 1.61.

A full expression of $\nabla_{\mu} ELBO[q]$, which is required for its computation, is given by the result of the following derivation:

$$
\begin{aligned}
\nabla_\mu ELBO[q] &= \nabla_\mu \mathbb{E}_{\xi\sim q(\xi)}\left[\log\left(p(\mathbf{x}|\tau^{-1}(\boldsymbol{\xi}))p(\tau^{-1}(\boldsymbol{\xi}))\right) + \log|\det \mathcal{J}_{\tau^{-1}}(\boldsymbol{\xi})|\right] \\
&\quad + \nabla_\mu \mathbb{H}_{\xi\sim q(\xi)}[q(\boldsymbol{\xi})] \\
&= \nabla_\mu \mathbb{E}_{\xi\sim q(\xi)}\left[\log\left(p(\mathbf{x}|\tau^{-1}(\boldsymbol{\xi}))p(\tau^{-1}(\boldsymbol{\xi}))\right) + \log|\det \mathcal{J}_{\tau^{-1}}(\boldsymbol{\xi})|\right] \\
&\quad + \mathbf{0} \\
&= \nabla_\mu \mathbb{E}_{\eta\sim\mathcal{N}(\mathbf{0},I)}\left[\log\left(p\left(\mathbf{x}|\tau^{-1}\left(S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta})\right)\right)p\left(\tau^{-1}\left(S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta})\right)\right)\right)\right] \\
&\quad + \nabla_\mu \mathbb{E}_{\eta\sim\mathcal{N}(\mathbf{0},I)}\left[\log\left|\det \mathcal{J}_{\tau^{-1}}\left(S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta})\right)\right|\right] \\
&= \mathbb{E}_{\eta\sim\mathcal{N}(\mathbf{0},I)}\left[\nabla_\mu \log\left(p\left(\mathbf{x}|\tau^{-1}\left(S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta})\right)\right)p\left(\tau^{-1}\left(S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta})\right)\right)\right)\right] \\
&\quad + \mathbb{E}_{\eta\sim\mathcal{N}(\mathbf{0},I)}\left[\nabla_\mu \log\left|\det \mathcal{J}_{\tau^{-1}}\left(S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta})\right)\right|\right] \\
&\approx \frac{1}{S}\sum_{s=1}^{S}\left(\nabla_\mu \log\left(p\left(\mathbf{x}|\tau^{-1}\left(S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta}_s)\right)\right)p\left(\tau^{-1}\left(S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta}_s)\right)\right)\right)\right) \\
&\quad + \frac{1}{S}\sum_{s=1}^{S}\left(\nabla_\mu \log\left|\det \mathcal{J}_{\tau^{-1}}\left(S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta}_s)\right)\right|\right)
\end{aligned}
\tag{A.46}
$$

i The first equality follows from equation 1.49.

ii The second equality follows from equation 1.51.

iii The third equality follow from derivation 1.59 and by the linearity of expectations.

iv The fourth equality follows from the first of equations 1.60.

v The fifth and final (approximate) equality follows from approximation 1.61.

A full expression of $\nabla_\omega ELBO[q]$, which is required for its computation, is given by the result of the following derivation:

$$
\begin{aligned}
\nabla_{\omega} ELBO[q] &= \nabla_{\omega} \mathbb{E}_{\xi \sim q(\xi)} \left[\log \left(p(\mathbf{x}|\tau^{-1}(\xi)) p(\tau^{-1}(\xi)) \right) + \log |\det \mathcal{J}_{\tau^{-1}}(\xi)| \right] \\
&\quad + \nabla_{\omega} \mathbb{H}_{\xi \sim q(\xi)}[q(\xi)] \\
&= \nabla_{\omega} \mathbb{E}_{\xi \sim q(\xi)} \left[\log \left(p(\mathbf{x}|\tau^{-1}(\xi)) p(\tau^{-1}(\xi)) \right) + \log |\det \mathcal{J}_{\tau^{-1}}(\xi)| \right] \\
&\quad + \mathbf{1} \\
&= \nabla_{\omega} \mathbb{E}_{\eta \sim \mathcal{N}(\mathbf{0},I)} \left[\log \left(p \left(\mathbf{x} | \tau^{-1} \left(S_{\phi}^{-1}(\eta) \right) \right) p \left(\tau^{-1} \left(S_{\phi}^{-1}(\eta) \right) \right) \right) \right] \\
&\quad + \nabla_{\omega} \mathbb{E}_{\eta \sim \mathcal{N}(\mathbf{0},I)} \left[\log \left| \det \mathcal{J}_{\tau^{-1}} \left(S_{\phi}^{-1}(\eta) \right) \right| \right] + \mathbf{1} \\
&= \mathbb{E}_{\eta \sim \mathcal{N}(\mathbf{0},I)} \left[\nabla_{\omega} \log \left(p \left(\mathbf{x} | \tau^{-1} \left(S_{\phi}^{-1}(\eta) \right) \right) p \left(\tau^{-1} \left(S_{\phi}^{-1}(\eta) \right) \right) \right) \right] \\
&\quad + \mathbb{E}_{\eta \sim \mathcal{N}(\mathbf{0},I)} \left[\nabla_{\omega} \log \left| \det \mathcal{J}_{\tau^{-1}} \left(S_{\phi}^{-1}(\eta) \right) \right| \right] + \mathbf{1} \\
&\approx \frac{1}{S} \sum_{s=1}^{S} \left(\nabla_{\omega} \log \left(p \left(\mathbf{x} | \tau^{-1} \left(S_{\phi}^{-1}(\eta_s) \right) \right) p \left(\tau^{-1} \left(S_{\phi}^{-1}(\eta_s) \right) \right) \right) \right) \\
&\quad + \frac{1}{S} \sum_{s=1}^{S} \left(\nabla_{\omega} \log \left| \det \mathcal{J}_{\tau^{-1}} \left(S_{\phi}^{-1}(\eta_s) \right) \right| \right) + \mathbf{1}
\end{aligned}
\tag{A.47}
$$

i The first equality follows from equation 1.49.

ii The second equality follows from equation 1.53.

iii The third equality follow from derivation 1.59 and by the linearity of expectations.

iv The fourth equality follows from the second of equations 1.60.

v The fifth and final (approximate) equality follows from approximation 1.61.

A full expression of $\nabla_L ELBO[q]$, which is required for its computation, is given by the result of the following derivation:

$$\begin{aligned}
\nabla_L ELBO[q] &= \nabla_L \mathbb{E}_{\xi\sim q(\xi)}\left[\log\left(p(\mathbf{x}|\tau^{-1}(\xi))p(\tau^{-1}(\xi))\right) + \log|\det \mathcal{J}_{\tau^{-1}}(\xi)|\right] \\
&\quad + \nabla_L \mathbb{H}_{\xi\sim q(\xi)}[q(\xi)] \\
&= \nabla_L \mathbb{E}_{\xi\sim q(\xi)}\left[\log\left(p(\mathbf{x}|\tau^{-1}(\xi))p(\tau^{-1}(\xi))\right) + \log|\det \mathcal{J}_{\tau^{-1}}(\xi)|\right] \\
&\quad + \left(L^{-1}\right)^T \\
&= \nabla_L \mathbb{E}_{\eta\sim\mathcal{N}(\mathbf{0},I)}\left[\log\left(p\left(\mathbf{x}|\tau^{-1}\left(S_{\phi}^{-1}(\eta)\right)\right)p\left(\tau^{-1}\left(S_{\phi}^{-1}(\eta)\right)\right)\right)\right] \\
&\quad + \nabla_L \mathbb{E}_{\eta\sim\mathcal{N}(\mathbf{0},I)}\left[\log\left|\det \mathcal{J}_{\tau^{-1}}\left(S_{\phi}^{-1}(\eta)\right)\right|\right] + \left(L^{-1}\right)^T \\
&= \mathbb{E}_{\eta\sim\mathcal{N}(\mathbf{0},I)}\left[\nabla_L \log\left(p\left(\mathbf{x}|\tau^{-1}\left(S_{\phi}^{-1}(\eta)\right)\right)p\left(\tau^{-1}\left(S_{\phi}^{-1}(\eta)\right)\right)\right)\right] \\
&\quad + \mathbb{E}_{\eta\sim\mathcal{N}(\mathbf{0},I)}\left[\nabla_L \log\left|\det \mathcal{J}_{\tau^{-1}}\left(S_{\phi}^{-1}(\eta)\right)\right|\right] + \left(L^{-1}\right)^T \\
&\approx \frac{1}{S}\sum_{s=1}^{S}\left(\nabla_L \log\left(p\left(\mathbf{x}|\tau^{-1}\left(S_{\phi_i}^{-1}(\eta_s)\right)\right)p\left(\tau^{-1}\left(S_{\phi_i}^{-1}(\eta_s)\right)\right)\right)\right) \\
&\quad + \frac{1}{S}\sum_{s=1}^{S}\left(\nabla_L \log\left|\det \mathcal{J}_{\tau^{-1}}\left(S_{\phi_i}^{-1}(\eta_s)\right)\right|\right) + \left(L^{-1}\right)^T
\end{aligned} \tag{A.48}$$

i The first equality follows from equation 1.49.

ii The second equality follows from equation 1.55.

iii The third equality follow from derivation 1.59 and by the linearity of expectations.

iv The fourth equality follows from the third of equations 1.60.

v The fifth and final (approximate) equality follows from approximation 1.61.

A.13 Derivation of a formula for the bivariate Gaussian copula density

The bivariate Gaussian copula density, $c(u_1, u_2; \rho)$, where $u_1 = \Phi(x_1; 0, 1)$ and $u_2 = \Phi(x_2; 0, 1)$, is defined via the following equation:

$$\mathcal{N}\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}; \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}\right) = c(u_1, u_2; \rho) \cdot \mathcal{N}(x_1; 0, 1) \cdot \mathcal{N}(x_2; 0, 1) \tag{A.49}$$

The formulae of the standard univariate Gaussian densities in equation A.49 are:

$$\mathcal{N}(x_1; 0, 1) = \frac{1}{\sqrt{2\pi}}\exp\left\{-\frac{1}{2}x_1^2\right\} \quad \text{and} \quad \mathcal{N}(x_2; 0, 1) = \frac{1}{\sqrt{2\pi}}\exp\left\{-\frac{1}{2}x_2^2\right\} \tag{A.50}$$

Given the general formula of a generic multivariate Gaussian density - $\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \Sigma) = \frac{1}{(2\pi)^{d/2}}|\Sigma|^{-1/2}\exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\}$ - where $d \in \mathbb{N}$ is the number of variables that constitute the multivariate Gaussian density, it can thus be deduced that:

$$
\begin{aligned}
\mathcal{N}\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}; \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}\right) &= \frac{1}{(2\pi)^{2/2}} \cdot \left| \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix} \right|^{-\frac{1}{2}} \\
&\quad \cdot \exp\left\{ -\frac{1}{2} \begin{pmatrix} x_1 - 0 \\ x_2 - 0 \end{pmatrix}^T \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}^{-1} \begin{pmatrix} x_1 - 0 \\ x_2 - 0 \end{pmatrix} \right\} \\
&= \frac{1}{(2\pi)^1} \cdot \frac{1}{\sqrt{1-\rho^2}} \\
&\quad \cdot \exp\left\{ -\frac{1}{2} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}^T \begin{pmatrix} \frac{1}{1-\rho^2} & -\frac{\rho}{1-\rho^2} \\ -\frac{\rho}{1-\rho^2} & \frac{1}{1-\rho^2} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \right\} \\
&= \frac{1}{2\pi\sqrt{1-\rho^2}} \cdot \exp\left\{ -\frac{1}{2(1-\rho^2)} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}^T \begin{pmatrix} 1 & -\rho \\ -\rho & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \right\} \\
&= \frac{1}{2\pi\sqrt{1-\rho^2}} \exp\left\{ -\frac{1}{2(1-\rho^2)} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}^T \begin{pmatrix} x_1 - \rho x_2 \\ -\rho x_1 + x_2 \end{pmatrix} \right\} \\
&= \frac{1}{2\pi\sqrt{1-\rho^2}} \exp\left\{ -\frac{1}{2(1-\rho^2)} \left(x_1^2 - 2\rho x_1 x_2 + x_2^2 \right) \right\}
\end{aligned}
\tag{A.51}
$$

Combining equations A.49, A.50, and the result of derivation A.51, it can thus be deduced that:

$$
\begin{aligned}
c(u_1, u_2; \rho) &= \frac{\mathcal{N}\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}; \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}\right)}{\mathcal{N}(x_1; 0, 1) \cdot \mathcal{N}(x_2; 0, 1)} \\
&= \frac{\frac{1}{2\pi\sqrt{1-\rho^2}} \exp\left\{ -\frac{1}{2(1-\rho^2)} \left(x_1^2 - 2\rho x_1 x_2 + x_2^2 \right) \right\}}{\frac{1}{\sqrt{2\pi}} \exp\left\{ -\frac{1}{2} x_1^2 \right\} \cdot \frac{1}{\sqrt{2\pi}} \exp\left\{ -\frac{1}{2} x_2^2 \right\}} \\
&= \frac{\frac{1}{2\pi\sqrt{1-\rho^2}} \exp\left\{ -\frac{x_1^2 - 2\rho x_1 x_2 + x_2^2}{2(1-\rho^2)} \right\}}{\frac{1}{2\pi} \exp\left\{ -\frac{1}{2} x_1^2 - \frac{1}{2} x_2^2 \right\}} \\
&= \frac{1}{\sqrt{1-\rho^2}} \exp\left\{ -\frac{x_1^2 - 2\rho x_1 x_2 + x_2^2}{2(1-\rho^2)} + \frac{1}{2} x_1^2 + \frac{1}{2} x_2^2 \right\} \\
&= \frac{1}{\sqrt{1-\rho^2}} \exp\left\{ -\frac{x_1^2 - 2\rho x_1 x_2 + x_2^2}{2(1-\rho^2)} + \frac{x_1^2 (1-\rho^2)}{2(1-\rho^2)} + \frac{x_2^2 (1-\rho^2)}{2(1-\rho^2)} \right\} \\
&= \frac{1}{\sqrt{1-\rho^2}} \exp\left\{ \frac{-x_1^2 + 2\rho x_1 x_2 - x_2^2 + x_1^2 - \rho^2 x_1^2 + x_2^2 - \rho^2 x_2^2}{2(1-\rho^2)} \right\} \\
&= \frac{1}{\sqrt{1-\rho^2}} \exp\left\{ \frac{2\rho x_1 x_2 - \rho^2 x_1^2 - \rho^2 x_2^2}{2(1-\rho^2)} \right\} \\
&= \frac{1}{\sqrt{1-\rho^2}} \exp\left\{ -\frac{\rho^2 \left(x_1^2 + x_2^2 \right) - 2\rho x_1 x_2}{2\left(1-\rho^2\right)} \right\}
\end{aligned}
\tag{A.52}
$$

Given the constraints that are generally applicable to the parameters of cumulative Gaussian distributions and Gaussian densities, it can thus be deduced that the result of derivation A.52 is subject to the following constraints:

i $x_1, x_2, u_1, u_2, \rho \in \mathbb{R}$

ii $x_1 \sim \mathcal{N}(0, 1)$

iii $x_2 \sim \mathcal{N}(0, 1)$

iv $u_1, u_2 \in (0, 1)$

v $\rho \in (-1, 1)$

B Derivations, Scripts & Results of the Original Research

B.1 The Prior Structure of the Baseline ADVI Algorithm

The aim of this appendix section is to outline a prior structure $p(\mathbf{z})$ for which: An initial vector $\mathbf{z}$ containing discrete categorical latent variables can be modelled in a continuous way; and the gradient $\nabla_{\mathbf{z}} \log p(\mathbf{x}, \mathbf{z})$ is defined, as are required for the **ADVI** algorithm (see section 1.11).

Let a simple (discrete) categorical random variable ω_D, that can only take one of two categorical values - "True" or "False" - first be considered. To model ω_D in a continuous way, we can use a continuous variable, ω, on $\mathbb{R}$, that is intrinsically linked to ω_D, where: $\omega \approx \omega_T$ when $\omega_D =$ "True"; and $\omega \approx \omega_F$ when $\omega_D =$ "False", for some $\omega_T, \omega_F \in \mathbb{R}$.

A graph of $p(\omega_D)$ could be represented by two vertical bars on a bar plot, the relative heights of which correspond to the probabilities of $\omega_D =$ "True" and $\omega_D =$ "False", where $p(\omega_D = \text{"True"}) + p(\omega_D = \text{"False"}) = 1$. As ω is continuous on $\mathbb{R}$, $p(\omega)$ can not simply be modelled as two vertical bars on a plot. However, an approximation to this could be obtained by using a continuous plot (the probability density function $p(\omega)$ being continuous on $\mathbb{R}$ would imply that the variable ω is also continuous on $\mathbb{R}$), where:

i $p(\omega)$ is close to zero when $\omega \not\approx \omega_T$ and $\omega \not\approx \omega_F$

ii $p(\omega)$ is close to $p(\omega_D = \text{"True"})$ when $\omega \approx \omega_T$

iii $p(\omega)$ is close to $p(\omega_D = \text{"False"})$ when $\omega \approx \omega_F$

iv As $p(\omega)$ is a probability density function, and ω is on $\mathbb{R}$, we must ensure that $\int_{-\infty}^{\infty} p(\omega) d\omega = 1$.

Let $m_i \in \mathbb{N}$ denote the number of categorical values that a discrete categorical latent variable z_i, in $\mathbf{z}$, can take. Each discrete categorical latent variable z_i, in $\mathbf{z}$, can then be equivalently represented by an appropriate vector of m_i variables of the type ω_D. This is achieved by ordering each of the categorical values that z_i can take, and equating each of these to an ordered vector of variables of the type ω_D, where all but one of the variables take a value of "False", with the "True" value being in the corresponding position of the called categorical value.

For example, if z_i could be "Red", "Green" or "Blue", in this chosen order, then:

i $z_i =$ "Red" is equivalent to $(\omega_{D,1}, \omega_{D,2}, \omega_{D,3})^T = (\text{"True"}, \text{"False"}, \text{"False"})^T$

ii $z_i =$ "Green" is equivalent to $(\omega_{D,1}, \omega_{D,2}, \omega_{D,3})^T = (\text{"False"}, \text{"True"}, \text{"False"})^T$

iii $z_i =$ "Blue" is equivalent to $(\omega_{D,1}, \omega_{D,2}, \omega_{D,3})^T = (\text{"False"}, \text{"False"}, \text{"True"})^T$

As the iris dataset, to which the required prior structure will be applied, only has one categorical target variable, then $\mathbf{z} = (z_1)$. This can effectively be interpreted as $\mathbf{z}$ and the one z_i (in $\mathbf{z}$) being equivalent in this context. This is the only structure of $\mathbf{z}$ that will henceforth be considered.

As $\nabla_{\mathbf{z}} \log p(\mathbf{x}, \mathbf{z}) = \nabla_{\mathbf{z}} \log (p(\mathbf{x}|\mathbf{z})p(\mathbf{z})) = \nabla_{\mathbf{z}} [\log p(\mathbf{x}|\mathbf{z}) + \log p(\mathbf{z})] = \nabla_{\mathbf{z}} \log p(\mathbf{x}|\mathbf{z}) + \nabla_{\mathbf{z}} \log p(\mathbf{z})$ and $\nabla_{\mathbf{z}} \log p(\mathbf{z}) = \frac{\nabla_{\mathbf{z}} p(\mathbf{z})}{p(\mathbf{z})}$, then for $\nabla_{\mathbf{z}} \log p(\mathbf{x}, \mathbf{z})$ to be defined, we require $p(\mathbf{z}) > 0$ for all $\mathbf{z}$ (as $p(\mathbf{z})$ is a probability density function, then it can not take negative values), and $\nabla_{\mathbf{z}} p(\mathbf{z})$ to be defined for all $\mathbf{z}$.

Given the "ordered vector of bi-categorical variables ω_D" framework outlined above, in order to meet the "continuous modelling" and "defined gradient" conditions outlined in section 1.11 and the first paragraph of this appendix section: Let each ω_D be approximated by an appropriate ω; ensure that each $p(\omega)$ is continuous on $\mathbb{R}$ (the probability density function $p(\omega)$ being continuous on $\mathbb{R}$ would imply that the variable ω is also continuous on $\mathbb{R}$); and ensure that each $\nabla_{\omega} p(\omega)$ is defined, and $p(\omega) > 0, \forall \omega \in \mathbb{R}$.

Given what has so far been outlined above, an evident choice of model for $p(\omega)$, as an approximation to $p(\omega_D)$, is the "sum of Gaussians" model: $p(\omega) = \beta_T \mathcal{N}(\omega; \omega_T, s^2) + \beta_F \mathcal{N}(\omega; \omega_F, s^2)$, where $\omega_T, \omega_F \in \mathbb{R}$ and $s \in \mathbb{R}^{++}$ are hyperparameters; $\beta_T = p(\omega_D =$ "True"); and $\beta_F = p(\omega_D =$ "False"). Subject to an appropriate choice of values for $\omega_T, \omega_F \in \mathbb{R}$ and $s \in \mathbb{R}^{++}$, this underlying "sum of Gaussians" model for the prior structure ensures that all the required conditions of the **ADVI** algorithm are met.

A sketch of $y = p(\omega) = \beta_T \mathcal{N}(\omega; \omega_T, s^2) + \beta_F \mathcal{N}(\omega; \omega_F, s^2)$ is given in figure B.1 in the case where $\omega_T > \omega_F$ and $\beta_T < \beta_F$. Note that whether $\omega_T > \omega_F$ or $\omega_T < \omega_F$ is a purely arbitrary choice (they are just numerical labels for nominal data).

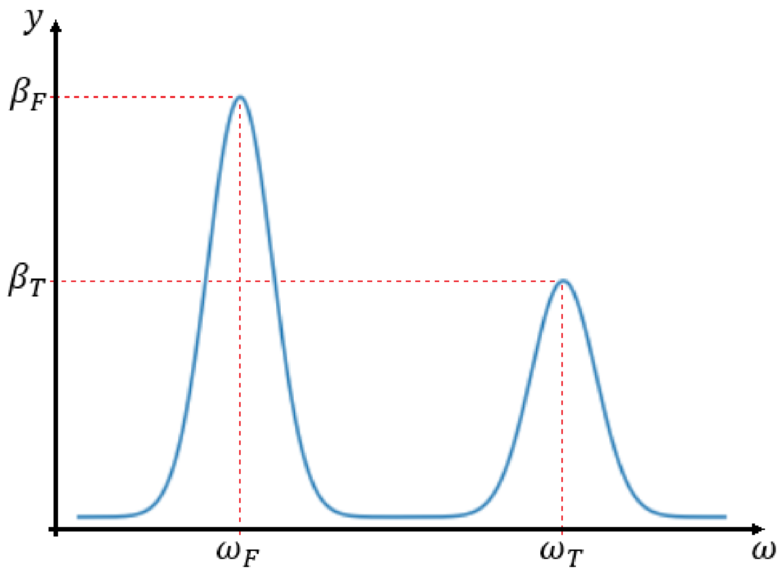


FIGURE B.1: A sketch of a constituent component of the "sum of Gaussians" model

For the "colour category latent variable" example given above, our adopted prior model would then be:

$$p(\mathbf{z}) = \beta_R \mathcal{N}(\mathbf{z}, \boldsymbol{\omega}_R, S) + \beta_G \mathcal{N}(\mathbf{z}, \boldsymbol{\omega}_G, S) + \beta_B \mathcal{N}(\mathbf{z}, \boldsymbol{\omega}_B, S) \tag{B.1}$$

where:

i $\mathbf{z} = \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix}$ and $S = \begin{pmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & s \end{pmatrix}$.

ii $\boldsymbol{\omega}_R = \begin{pmatrix} \omega_T \\ \omega_F \\ \omega_F \end{pmatrix}$, $\boldsymbol{\omega}_G = \begin{pmatrix} \omega_F \\ \omega_T \\ \omega_F \end{pmatrix}$ and $\boldsymbol{\omega}_B = \begin{pmatrix} \omega_F \\ \omega_F \\ \omega_T \end{pmatrix}$.

iii $\beta_R = p(z_i = \text{"Red"})$, $\beta_G = p(z_i = \text{"Green"})$ and $\beta_B = p(z_i = \text{"Blue"})$.

iv $\mathbf{z}, \boldsymbol{\omega}_R, \boldsymbol{\omega}_G, \boldsymbol{\omega}_B \in \mathbb{R}^3$, $s \in \mathbb{R}^{++}$ and $\beta_R, \beta_G, \beta_B \in (0, 1)$

This model can be extrapolated as necessary, in the fashion that is evidently laid out by equation B.1, for any dimension d of the latent vector $\mathbf{z}$, where $d \in \mathbb{N}$ and $d \geq 2$. For example, if z_i could take one of 4 categorical values, $p(\mathbf{z})$ would be a sum of four proportionately scaled 4-dimensional Gaussians, resulting in $p(\mathbf{z})$ having 4 distinct peaks, at those numerical values that have been chosen to denote each of the 4 categorical values that z_i can take. In any case, it is evident that: $\mathbf{z}$ is continuous; $p(\mathbf{z}) > 0, \forall \mathbf{z} \in \mathbb{R}^d$; and $\nabla_{\mathbf{z}} p(\mathbf{z})$ is defined $\forall \mathbf{z} \in \mathbb{R}^d$, as required. Furthermore, for the example above where $\beta_R + \beta_G + \beta_B = 1$, as is required for a probability density function:

$$\begin{aligned}
\int_{-\infty}^{\infty} p(\mathbf{z}) d\mathbf{z} &= \int_{-\infty}^{\infty} \left(\beta_R \mathcal{N}(\mathbf{z}, \boldsymbol{\omega}_R, S) + \beta_G \mathcal{N}(\mathbf{z}, \boldsymbol{\omega}_G, S) + \beta_B \mathcal{N}(\mathbf{z}, \boldsymbol{\omega}_B, S)\right) d\mathbf{z} \\
&= \int_{-\infty}^{\infty} \beta_R \mathcal{N}(\mathbf{z}, \boldsymbol{\omega}_R, S) d\mathbf{z} + \int_{-\infty}^{\infty} \beta_G \mathcal{N}(\mathbf{z}, \boldsymbol{\omega}_G, S) d\mathbf{z} + \int_{-\infty}^{\infty} \beta_B \mathcal{N}(\mathbf{z}, \boldsymbol{\omega}_B, S) d\mathbf{z} \\
&= \beta_R \int_{-\infty}^{\infty} \mathcal{N}(\mathbf{z}, \boldsymbol{\omega}_R, S) d\mathbf{z} + \beta_G \int_{-\infty}^{\infty} \mathcal{N}(\mathbf{z}, \boldsymbol{\omega}_G, S) d\mathbf{z} + \beta_B \int_{-\infty}^{\infty} \mathcal{N}(\mathbf{z}, \boldsymbol{\omega}_B, S) d\mathbf{z} \\
&= (\beta_R * 1) + (\beta_G * 1) + (\beta_B * 1) \\
&= \beta_R + \beta_G + \beta_B \\
&= 1
\end{aligned} \tag{B.2}$$

Note that, in effect, $\lim_{s \to 0} p(\mathbf{z}) \Leftrightarrow p(z_i)$, where z_i is a discrete categorical variable and $\mathbf{z}$ is the vector of continuous variables, whereby $p(\mathbf{z})$ approximates $p(z_i)$. It would thus seem tempting to set the hyperparameter s to be as small as practically possible. However, should s take too small a value, values of gradients that are based on the prior would then be very large in the vicinity of the numerical vector points in the vector space that have been chosen to denote each of the categorical values of z_i (consider points ω_T and ω_F in figure B.1). These large gradient values are problematic, as to prevent "overshooting" in an optimisation of the **ADVI** algorithm, we would require countering small step sizes that would be too small to be feasible for optimisation throughout all other regions with more moderate gradient values.

As a means of best approximating $p(z_i)$ with $p(\mathbf{z})$, without having to make the hyperparameter s too small, we leverage a more fundamental property of the probability densities of discrete variables, by ensuring that there are "discrete regions of probability density" in $p(\mathbf{z})$. As depicted in figure B.1, this can be achieved by ensuring that the magnitude of the difference between ω_T and ω_F is significantly larger than s. The probability density in $p(\mathbf{z})$ will then lie predominantly in regions significantly closer to one of the numerical points - $(\omega_T \quad \omega_F \quad \omega_F \quad \dots)^T$, $(\omega_F \quad \omega_T \quad \omega_F \quad \dots)^T$, etc - that denote each of the categorical values of z_i.

B.2 The Likelihood Structure of the Baseline ADVI Algorithm

The aim of this appendix section is to outline a suitable likelihood structure, $p(\mathbf{x}|\mathbf{z})$, assuming $\mathbf{z}$ is modelled as a vector of continuous variables (see appendix section B.1), for which the gradient $\nabla_{\mathbf{z}} \log p(\mathbf{x}, \mathbf{z})$ is defined, as required for the **ADVI** algorithm (see section 1.11).

Let the "ordered vector of bi-categorical variables ω_D" framework outlined in appendix section B.1 be invoked as a means of modelling $\mathbf{z}$ as a vector of continuous variables: Each of the ω_D variables in $\mathbf{z}$ will be mapped to, and modelled by, a continuous variable ω that, in training, will only take a value of either ω_T or ω_F.

As $\nabla_{\mathbf{z}} \log p(\mathbf{x}, \mathbf{z}) = \nabla_{\mathbf{z}} \log (p(\mathbf{x}|\mathbf{z})p(\mathbf{z})) = \nabla_{\mathbf{z}} [\log p(\mathbf{x}|\mathbf{z}) + \log p(\mathbf{z})] = \nabla_{\mathbf{z}} \log p(\mathbf{x}|\mathbf{z}) + \nabla_{\mathbf{z}} \log p(\mathbf{z})$ and $\nabla_{\mathbf{z}} \log p(\mathbf{x}|\mathbf{z}) = \frac{\nabla_{\mathbf{z}} p(\mathbf{x}|\mathbf{z})}{p(\mathbf{x}|\mathbf{z})}$, then for $\nabla_{\mathbf{z}} \log p(\mathbf{x}, \mathbf{z})$ to be defined, we require $p(\mathbf{x}|\mathbf{z}) > 0$ for all $\mathbf{x}$ and $\mathbf{z}$ (as $p(\mathbf{x}|\mathbf{z})$ is a probability density function, then it can not take negative values), and $\nabla_{\mathbf{z}} p(\mathbf{x}|\mathbf{z})$ to be defined for all $\mathbf{x}$ and $\mathbf{z}$.

In keeping with the underlying Gaussian-based framework in the **ADVI** academic article (Kucukelbir et al., 2017), the likelihood $p(\mathbf{x}|\mathbf{z})$ shall take the form of a Gaussian density. This structure has been chosen predominantly as a means of leveraging the amenable mathematical features and properties of Gaussians, that will allow for computational speed up and closed-form evaluations in sections of the **ADVI** algorithm that would not otherwise have been possible (see section 1.11 and the **ADVI** academic article referenced above). The likelihood $p(\mathbf{x}|\mathbf{z})$, and thus the Gaussian density that it equates to, has the same dimension as the observations $\mathbf{x}$. It is assumed that each component, x_i, of $\mathbf{x}$ belongs to the space $\mathbb{R}$: Thus $\mathbf{x} \in \mathbb{R}^d$, for some $d \in \mathbb{N}$. Therefore, the domain and codomain of $p(\mathbf{x}|\mathbf{z})$ are $\mathbb{R}^d$ and $\mathbb{R}^{++}$, respectively.

Lets assume we have a dataset of numerically continuous feature variables and discrete categorical target variable(s). Each of the discrete (set of) values that the target variable(s) take will be denoted by an ordered continuous vector of one ω_T value, and ω_F values in the remaining positions, given the "continuous modelling" framework outlined above and in appendix section B.1. The process by which the likelihood is fitted, given such a dataset, is outlined as follows:

i We first need to specify an order for the discrete (set of) values that the target variable(s) take. Each of these ordered (set of) values will then be mapped to a modified one-hot vector, with an ω_T in the position corresponding to the ordered position of the value (set) that the modified one-hot vector represents, and ω_F values in all other positions. Each of these modified one-hot vectors have a length equal to the number of discrete (set of) values.

ii We then need to calculate the mean and the variance of the data for each of the feature variables, attributed to each of the discrete (set of) categorical values that the target variable(s) can take. It is assumed that each calculated variance will be greater than zero. This establishes a mean vector $\boldsymbol{\mu}$ and a mean-field (diagonal) covariance matrix Σ associated with each of the discrete (set of) categorical values.

iii Finally, we need to find a continuous vector function $\boldsymbol{\mu}(\mathbf{z})$ and a continuous matrix function $\Sigma(\mathbf{z})$, for which these functions are equal to each of the empirically calculated $\boldsymbol{\mu}$ and Σ in the previous step when $\mathbf{z}$ takes the value that corresponds to each of these given categorical values (or value sets). For any value of $\mathbf{z}$, we must ensure that $\Sigma(\mathbf{z})$ is a valid mean-field covariance matrix.

The covariance matrix $\Sigma(\mathbf{z})$ of the Gaussian likelihood $p(\mathbf{x}|\mathbf{z})$ is set to be a mean-field (diagonal) covariance matrix to avoid the time complexity issue that arises when $\mathbf{x}$ is very large (which is often the case in practice for **VI**): If $d \in \mathbb{N}$ is the dimension of $\mathbf{x}$, then $\Sigma(\mathbf{z})$ is a $d \times d$ matrix. In a generic $d \times d$ covariance matrix, there are d variances and $\frac{1}{2}d(d-1)$ covariances: When d is large, there are significantly more covariances than variances, so disregarding the covariances can significantly reduce the required computational expense.

The vector function $\boldsymbol{\mu}(\mathbf{z})$ can take the form of a linear function, as $\boldsymbol{\mu}(\mathbf{z})$ is a valid mean vector for any value of $\mathbb{R}^d$. However, as each of the variance (diagonal) components in $\Sigma(\mathbf{z})$ must be greater than zero (required for a Gaussian), a valid form could be an exponential of a linear combination of logarithms (we can not use a linear form directly, as a linear function can take negative values if the domain is $\mathbb{R}^d$).

Let the process of fitting such a likelihood be illustrated with a simple & contrived example dataset. Lets say we have the dataset summarised in the following table:

t	f_1	f_2
Red	3	7
Red	2	9
Green	4	15
Green	6	17
Green	5	19
Blue	8	13
Blue	9	11

i Let it be assumed that this dataset has one target variable, denoted by t, with the remaining two variables thus being feature variables, denoted by f_1 and f_2.

ii Each of f_1 and f_2 are numerically continuous variables on $\mathbb{R}$. The target variable, t, is a (discrete) categorical variable, that can take one of three categorical values: "Red", "Green" or "Blue".

iii Let the categorical values that t can take have the order: "Red", "Green", "Blue". t can then be continuously re-modelled by the continuous vector variable $\mathbf{t}' = (t_r \quad t_g \quad t_b)^T$, where $t_r, t_g, t_b \in \mathbb{R}$ (the subscripts r, g and b denote the categorical values "Red", "Green" and "Blue", respectively). We thus have: $t =$ "Red" $\Leftrightarrow \mathbf{t}' = (\omega_T \quad \omega_F \quad \omega_F)^T$; $t =$ "Green" $\Leftrightarrow \mathbf{t}' = (\omega_F \quad \omega_T \quad \omega_F)^T$; and $t =$ "Blue" $\Leftrightarrow \mathbf{t}' = (\omega_F \quad \omega_F \quad \omega_T)^T$.

iv Let $\mathbf{f} = \begin{pmatrix} f_1 \\ f_2 \end{pmatrix}$. The likelihood that we are looking to fit is thus $p(\mathbf{f}|\mathbf{t}^{'})$.

v Let: $\boldsymbol{\mu}_r$ denote the mean vector of the feature variables data attributed to a target variable value of $t =$ "Red"; and Σ_r denote the mean-field covariance matrix of the feature variables data attributed to a target variable value of $t =$ "Red". These same underlying definitions apply to $\boldsymbol{\mu}_g$, $\boldsymbol{\mu}_b$, Σ_g and Σ_b, but with the subscript r, for "Red", replaced by either g, for "Green", or b, for "Blue".

vi Let the hyperparmeters ω_T and ω_F be set to the values 5 and -5, respectively. Therefore: $t =$ "Red" $\Leftrightarrow \mathbf{t}^{'} = (5 \quad -5 \quad -5)^T$; $t =$ "Green" $\Leftrightarrow \mathbf{t}^{'} = (-5 \quad 5 \quad -5)^T$; and $t =$ "Blue" $\Leftrightarrow \mathbf{t}^{'} = (-5 \quad -5 \quad 5)^T$.

vii We will assume that the dataset provided in the table above is a sample, taken from a larger population. Therefore, Bessel's correction will be used in the calculation of all variances.

Given the dataset provided in the table above, and the preceding definitions and setup, quantities such as $\boldsymbol{\mu}_g$ and Σ_g can be directly calculated as follows:

$$\begin{aligned} \boldsymbol{\mu}_g &= \begin{pmatrix} \frac{4+6+5}{3} \\ \frac{15+17+19}{3} \end{pmatrix} = \begin{pmatrix} 5 \\ 17 \end{pmatrix} \\ \Sigma_g &= \begin{pmatrix} \frac{(4-5)^2+(6-5)^2+(5-5)^2}{3-1} & 0 \\ 0 & \frac{(15-17)^2+(17-17)^2+(19-17)^2}{3-1} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 4 \end{pmatrix} \end{aligned} \tag{B.3}$$

A complete set of the required empirically calculated vector and matrix quantities is given in the following table:

Subscript	$\boldsymbol{\mu}$	Σ
r	$\begin{pmatrix} 2.5 \\ 8 \end{pmatrix}$	$\begin{pmatrix} 0.5 & 0 \\ 0 & 2 \end{pmatrix}$
g	$\begin{pmatrix} 5 \\ 17 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & 4 \end{pmatrix}$
b	$\begin{pmatrix} 8.5 \\ 12 \end{pmatrix}$	$\begin{pmatrix} 0.5 & 0 \\ 0 & 2 \end{pmatrix}$

The vector and matrix functions - $\boldsymbol{\mu}(\mathbf{t}^{'})$ and $\Sigma(\mathbf{t}^{'})$ - that are used to parametrically define the fitted likelihood, $p(\mathbf{f}|\mathbf{t}^{'})$, can then be derived as follows:

$$
\begin{aligned}
\boldsymbol{\mu}(\mathbf{t}^{'}) &= \boldsymbol{\mu}\begin{pmatrix} t_r \\ t_g \\ t_b \end{pmatrix} \\
&= \frac{t_r - \omega_F}{\omega_T - \omega_F}\boldsymbol{\mu}_r + \frac{t_g - \omega_F}{\omega_T - \omega_F}\boldsymbol{\mu}_g + \frac{t_b - \omega_F}{\omega_T - \omega_F}\boldsymbol{\mu}_b \\
&= \frac{t_r - (-5)}{5 - (-5)}\begin{pmatrix} 2.5 \\ 8 \end{pmatrix} + \frac{t_g - (-5)}{5 - (-5)}\begin{pmatrix} 5 \\ 17 \end{pmatrix} + \frac{t_b - (-5)}{5 - (-5)}\begin{pmatrix} 8.5 \\ 12 \end{pmatrix} \\
&= \frac{t_r + 5}{5 + 5}\begin{pmatrix} 5/2 \\ 8 \end{pmatrix} + \frac{t_g + 5}{5 + 5}\begin{pmatrix} 5 \\ 17 \end{pmatrix} + \frac{t_b + 5}{5 + 5}\begin{pmatrix} 17/2 \\ 12 \end{pmatrix} \\
&= \frac{t_r}{10}\begin{pmatrix} 5/2 \\ 8 \end{pmatrix} + \frac{1}{2}\begin{pmatrix} 5/2 \\ 8 \end{pmatrix} + \frac{t_g}{10}\begin{pmatrix} 5 \\ 17 \end{pmatrix} + \frac{1}{2}\begin{pmatrix} 5 \\ 17 \end{pmatrix} + \frac{t_b}{10}\begin{pmatrix} 17/2 \\ 12 \end{pmatrix} + \frac{1}{2}\begin{pmatrix} 17/2 \\ 12 \end{pmatrix} \\
&= \begin{pmatrix} t_r/4 \\ 4t_r/5 \end{pmatrix} + \begin{pmatrix} 5/4 \\ 4 \end{pmatrix} + \begin{pmatrix} t_g/2 \\ 17t_g/10 \end{pmatrix} + \begin{pmatrix} 5/2 \\ 17/2 \end{pmatrix} + \begin{pmatrix} 17t_b/20 \\ 6t_b/5 \end{pmatrix} + \begin{pmatrix} 17/4 \\ 6 \end{pmatrix} \\
&= \begin{pmatrix} \frac{1}{4}t_r + \frac{1}{2}t_g + \frac{17}{20}t_b + \frac{5}{4} + \frac{5}{2} + \frac{17}{4} \\ \frac{4}{5}t_r + \frac{17}{10}t_g + \frac{6}{5}t_b + 4 + \frac{17}{2} + 6 \end{pmatrix} \\
&= \begin{pmatrix} \frac{1}{4}t_r + \frac{1}{2}t_g + \frac{17}{20}t_b + 8 \\ \frac{4}{5}t_r + \frac{17}{10}t_g + \frac{6}{5}t_b + \frac{37}{2} \end{pmatrix}
\end{aligned}
\tag{B.4}
$$

$$
\begin{aligned}
\Sigma(\mathbf{t}') &= \Sigma\begin{pmatrix} t_r \\ t_g \\ t_b \end{pmatrix} \\
&= \exp\left\{ \frac{t_r-\omega_F}{\omega_T-\omega_F}\log(\Sigma_r) + \frac{t_g-\omega_F}{\omega_T-\omega_F}\log(\Sigma_g) + \frac{t_b-\omega_F}{\omega_T-\omega_F}\log(\Sigma_b) \right\} \\
&= \exp\left\{ \frac{t_r-(-5)}{5-(-5)}\log\begin{pmatrix} 0.5 & 0 \\ 0 & 2 \end{pmatrix} + \frac{t_g-(-5)}{5-(-5)}\log\begin{pmatrix} 1 & 0 \\ 0 & 4 \end{pmatrix} \right. \\
&\qquad\qquad\qquad \left. + \frac{t_b-(-5)}{5-(-5)}\log\begin{pmatrix} 0.5 & 0 \\ 0 & 2 \end{pmatrix} \right\} \\
&= \exp\left\{ \frac{t_r+5}{5+5}\begin{pmatrix} \log(1/2) & 0 \\ 0 & \log(2) \end{pmatrix} + \frac{t_g+5}{5+5}\begin{pmatrix} \log(1) & 0 \\ 0 & \log(4) \end{pmatrix} \right\} \\
&\qquad \cdot \exp\left\{ \frac{t_b+5}{5+5}\begin{pmatrix} \log(1/2) & 0 \\ 0 & \log(2) \end{pmatrix} \right\} \\
&= \exp\left\{ \frac{t_r}{10}\begin{pmatrix} \log(1/2) & 0 \\ 0 & \log(2) \end{pmatrix} + \frac{1}{2}\begin{pmatrix} \log(1/2) & 0 \\ 0 & \log(2) \end{pmatrix} \right\} \\
&\qquad \cdot \exp\left\{ \frac{t_g}{10}\begin{pmatrix} \log(1) & 0 \\ 0 & \log(4) \end{pmatrix} + \frac{1}{2}\begin{pmatrix} \log(1) & 0 \\ 0 & \log(4) \end{pmatrix} \right\} \\
&\qquad \cdot \exp\left\{ \frac{t_b}{10}\begin{pmatrix} \log(1/2) & 0 \\ 0 & \log(2) \end{pmatrix} + \frac{1}{2}\begin{pmatrix} \log(1/2) & 0 \\ 0 & \log(2) \end{pmatrix} \right\} \\
&= \exp\left\{ \begin{pmatrix} t_r\log(1/2)/10 & 0 \\ 0 & t_r\log(2)/10 \end{pmatrix} + \begin{pmatrix} \log(1/2)/2 & 0 \\ 0 & \log(2)/2 \end{pmatrix} \right\} \\
&\qquad \cdot \exp\left\{ \begin{pmatrix} t_g\log(1)/10 & 0 \\ 0 & t_g\log(4)/10 \end{pmatrix} + \begin{pmatrix} \log(1)/2 & 0 \\ 0 & \log(4)/2 \end{pmatrix} \right\} \\
&\qquad \cdot \exp\left\{ \begin{pmatrix} t_b\log(1/2)/10 & 0 \\ 0 & t_b\log(2)/10 \end{pmatrix} + \begin{pmatrix} \log(1/2)/2 & 0 \\ 0 & \log(2)/2 \end{pmatrix} \right\} \\
&= \exp\left\{ \begin{pmatrix} \frac{\log\left(\frac{1}{2}\right)}{10}t_r + \frac{\log(1)}{10}t_g + \frac{\log\left(\frac{1}{2}\right)}{10}t_b & 0 \\ 0 & \frac{\log(2)}{10}t_r + \frac{\log(4)}{10}t_g + \frac{\log(2)}{10}t_b \end{pmatrix} \right\} \\
&\qquad \cdot \exp\left\{ \begin{pmatrix} \frac{\log\left(\frac{1}{2}\right)}{2} + \frac{\log(1)}{2} + \frac{\log\left(\frac{1}{2}\right)}{2} & 0 \\ 0 & \frac{\log(2)}{2} + \frac{\log(4)}{2} + \frac{\log(2)}{2} \end{pmatrix} \right\} \\
&= \exp\left\{ \begin{pmatrix} \alpha(\mathbf{t}') & 0 \\ 0 & \beta(\mathbf{t}') \end{pmatrix} \right\} \\
&= \begin{pmatrix} e^{\alpha(\mathbf{t}')} & 0 \\ 0 & e^{\beta(\mathbf{t}')} \end{pmatrix}
\end{aligned} \tag{B.5}
$$

where $\alpha(\mathbf{t}') = \frac{\log\left(\frac{1}{2}\right)}{10}t_r + \frac{\log(1)}{10}t_g + \frac{\log\left(\frac{1}{2}\right)}{10}t_b + \frac{\log(1)}{2} + \log\left(\frac{1}{2}\right)$ and $\beta(\mathbf{t}') = \frac{\log(2)}{10}t_r + \frac{\log(4)}{10}t_g + \frac{\log(2)}{10}t_b + \frac{\log(4)}{2} + \log(2)$ in equation B.5.

By inspection of derivations B.4 and B.5 (particularly the second of their equalities), it can be noted that:

$$\mu\begin{pmatrix}5\\-5\\-5\end{pmatrix}=\mu_r,\quad \mu\begin{pmatrix}-5\\5\\-5\end{pmatrix}=\mu_g\quad\text{and}\quad \mu\begin{pmatrix}-5\\-5\\5\end{pmatrix}=\mu_b, \tag{B.6}$$

and

$$\Sigma\begin{pmatrix}5\\-5\\-5\end{pmatrix}=\Sigma_r,\quad \Sigma\begin{pmatrix}-5\\5\\-5\end{pmatrix}=\Sigma_g\quad\text{and}\quad \Sigma\begin{pmatrix}-5\\-5\\5\end{pmatrix}=\Sigma_b, \tag{B.7}$$

as required.

Given that the observations vector **f** is a 2-dimensional vector, and given the definitions, derivations and setup outlined above, the fitted likelihood is thus given as follows:

$$\begin{aligned}p\left(\mathbf{f}|\mathbf{t}^{'}\right)&=\mathcal{N}\left(\mathbf{f};\boldsymbol{\mu}(\mathbf{t}^{'}),\Sigma(\mathbf{t}^{'})\right)\\&=\frac{1}{(2\pi)^{2/2}}\left|\Sigma(\mathbf{t}^{'})\right|^{-\frac{1}{2}}\exp\left\{-\frac{1}{2}\left(\mathbf{f}-\boldsymbol{\mu}(\mathbf{t}^{'})\right)^{T}\left(\Sigma(\mathbf{t}^{'})\right)^{-1}\left(\mathbf{f}-\boldsymbol{\mu}(\mathbf{t}^{'})\right)\right\}\\&=\frac{1}{2\pi}\left|\Sigma(\mathbf{t}^{'})\right|^{-\frac{1}{2}}\exp\left\{-\frac{1}{2}\left(\mathbf{f}-\boldsymbol{\mu}(\mathbf{t}^{'})\right)^{T}\Sigma^{-1}(\mathbf{t}^{'})\left(\mathbf{f}-\boldsymbol{\mu}(\mathbf{t}^{'})\right)\right\}\end{aligned} \tag{B.8}$$

where $\left(\mathbf{f}-\boldsymbol{\mu}(\mathbf{t}^{'})\right)=\begin{pmatrix}f_1-(\frac{1}{4}t_r+\frac{1}{2}t_g+\frac{17}{20}t_b+8)\\f_2-(\frac{4}{5}t_r+\frac{17}{10}t_g+\frac{6}{5}t_b+\frac{37}{2})\end{pmatrix}=\begin{pmatrix}f_1-\frac{1}{4}t_r-\frac{1}{2}t_g-\frac{17}{20}t_b-8\\f_2-\frac{4}{5}t_r-\frac{17}{10}t_g-\frac{6}{5}t_b-\frac{37}{2}\end{pmatrix}$ and $\Sigma^{-1}(\mathbf{t}^{'})=\left(\Sigma(\mathbf{t}^{'})\right)^{-1}$.

B.3 Derivations of the Required ELBO Derivatives in Copula ADVI

In this appendix section, just as in section 2.3, my original approach to the copula augmentation of the mean-field approach to **ADVI**, as outlined in the paper (Kucukelbir et al., 2017), is referred to as "copula **ADVI**".

This appendix section outlines derivations of each of the required **ELBO** derivatives in copula **ADVI**, each of which are specified in equations 2.6: $\nabla_{\mu}ELBO[q(\mathbf{z})]$, $\nabla_{\omega}ELBO[q(\mathbf{z})]$ & $\nabla_{\alpha}ELBO[q(\mathbf{z})]$.

Say that $\mathbf{z}\sim q(\mathbf{z};\boldsymbol{\mu},\Sigma)$, where $q(\mathbf{z};\boldsymbol{\mu},\Sigma)$ is defined as in equation 2.2. To transform **z** to $\boldsymbol{\eta}$, where $\boldsymbol{\eta}\sim\mathcal{N}(\mathbf{0},I)$, and where **z** and $\boldsymbol{\eta}$ have the same dimension - i.e., to standardise the full-rank Gaussian $q(\mathbf{z};\boldsymbol{\mu},\Sigma)$ - we require the following transformation (this is the elliptical standardisation step):

$$\boldsymbol{\eta}=\Sigma^{-\frac{1}{2}}(\mathbf{z}-\boldsymbol{\mu}) \tag{B.9}$$

The "unconstrained" reparameterisation transformations of the of the variational parameters in copula **ADVI**, where appropriate, are $\alpha_i=\tan\left(\frac{\pi}{2}\rho_i\right)$ (equation 2.5) and

$\omega_i = \log(\sigma_i)$ (see section 2.3). Hence $\rho_i = \frac{2}{\pi}\arctan(\alpha_i)$ and $\sigma_i = e^{\omega_i}$. Substituting these equations into the general form of the covariance matrix of the surrogate posterior, as expressed by the last of equations 2.3, gives:

$$
\begin{aligned}
\Sigma &= \begin{pmatrix} \sigma_1^2 & \rho_{12}\sigma_1\sigma_2 & \cdots & \rho_{1d}\sigma_1\sigma_d \\ \rho_{12}\sigma_1\sigma_2 & \sigma_2^2 & \cdots & \rho_{2d}\sigma_2\sigma_d \\ \vdots & \vdots & \ddots & \vdots \\ \rho_{1d}\sigma_1\sigma_d & \rho_{2d}\sigma_2\sigma_d & \cdots & \sigma_d^2 \end{pmatrix} \\
&= \begin{pmatrix} e^{2\omega_1} & \frac{2}{\pi}e^{\omega_1+\omega_2}\arctan(\alpha_{12}) & \cdots & \frac{2}{\pi}e^{\omega_1+\omega_d}\arctan(\alpha_{1d}) \\ \frac{2}{\pi}e^{\omega_1+\omega_2}\arctan(\alpha_{12}) & e^{2\omega_2} & \cdots & \frac{2}{\pi}e^{\omega_2+\omega_d}\arctan(\alpha_{2d}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{2}{\pi}e^{\omega_1+\omega_d}\arctan(\alpha_{1d}) & \frac{2}{\pi}e^{\omega_2+\omega_d}\arctan(\alpha_{2d}) & \cdots & e^{2\omega_d} \end{pmatrix}
\end{aligned}
\tag{B.10}
$$

Consider a generic function y; the fact that Σ can be expressed as a function of $\boldsymbol{\omega}$ and $\boldsymbol{\alpha}$; and the fact that equation B.9 implies that $\mathbf{z} = \Sigma^{\frac{1}{2}}\boldsymbol{\eta} + \boldsymbol{\mu}$. The following equalities can then be established:

$$
\begin{aligned}
\nabla_{\boldsymbol{\mu}}[y] &= \frac{\partial y}{\partial \boldsymbol{\mu}} = \frac{\partial y}{\partial \mathbf{z}} \cdot \frac{\partial \mathbf{z}}{\partial \boldsymbol{\mu}} = \nabla_{\mathbf{z}}[y] \cdot \nabla_{\boldsymbol{\mu}}[\mathbf{z}] = \nabla_{\mathbf{z}}[y] \cdot \nabla_{\boldsymbol{\mu}}[\Sigma^{\frac{1}{2}}\boldsymbol{\eta} + \boldsymbol{\mu}] = \nabla_{\mathbf{z}}[y] \cdot \mathbf{1} = \nabla_{\mathbf{z}}[y] \\
\nabla_{\boldsymbol{\omega}}[y] &= \frac{\partial y}{\partial \boldsymbol{\omega}} = \frac{\partial y}{\partial \mathbf{z}} \cdot \frac{\partial \mathbf{z}}{\partial \boldsymbol{\omega}} = \nabla_{\mathbf{z}}[y] \cdot \nabla_{\boldsymbol{\omega}}[\mathbf{z}] = \nabla_{\mathbf{z}}[y] \cdot \nabla_{\boldsymbol{\omega}}[\Sigma^{\frac{1}{2}}\boldsymbol{\eta} + \boldsymbol{\mu}] = \nabla_{\mathbf{z}}[y] \cdot \nabla_{\boldsymbol{\omega}}[\Sigma^{\frac{1}{2}}\boldsymbol{\eta}] \\
\nabla_{\boldsymbol{\alpha}}[y] &= \frac{\partial y}{\partial \boldsymbol{\alpha}} = \frac{\partial y}{\partial \mathbf{z}} \cdot \frac{\partial \mathbf{z}}{\partial \boldsymbol{\alpha}} = \nabla_{\mathbf{z}}[y] \cdot \nabla_{\boldsymbol{\alpha}}[\mathbf{z}] = \nabla_{\mathbf{z}}[y] \cdot \nabla_{\boldsymbol{\alpha}}[\Sigma^{\frac{1}{2}}\boldsymbol{\eta} + \boldsymbol{\mu}] = \nabla_{\mathbf{z}}[y] \cdot \nabla_{\boldsymbol{\alpha}}[\Sigma^{\frac{1}{2}}\boldsymbol{\eta}]
\end{aligned}
\tag{B.11}
$$

Equivalent expressions of each of $\nabla_{\boldsymbol{\mu}}ELBO[q(\mathbf{z})]$, $\nabla_{\boldsymbol{\omega}}ELBO[q(\mathbf{z})]$ and $\nabla_{\boldsymbol{\alpha}}ELBO[q(\mathbf{z})]$, that will allow us to obtain values for each of these gradients during the optimisation process in (copula) **ADVI**, can be derived as follows (please refer to the list of comments proceeding the following 3 derivations):

$$
\begin{aligned}
\nabla_{\boldsymbol{\mu}}ELBO[q(\mathbf{z})] &= \nabla_{\boldsymbol{\mu}}\left[\mathbb{E}_{\boldsymbol{\eta}\sim\mathcal{N}(\mathbf{0},I)}[\log p(\mathbf{z}) + \log p(\mathbf{x}|\mathbf{z})] + \frac{d}{2}(1+\log(2\pi)) + \frac{1}{2}\log|\Sigma|\right] \\
&= \nabla_{\boldsymbol{\mu}}\mathbb{E}_{\boldsymbol{\eta}\sim\mathcal{N}(\mathbf{0},I)}[\log p(\mathbf{z}) + \log p(\mathbf{x}|\mathbf{z})] \\
&\qquad + \nabla_{\boldsymbol{\mu}}\left[\frac{d}{2}(1+\log(2\pi))\right] + \nabla_{\boldsymbol{\mu}}\left[\frac{1}{2}\log|\Sigma|\right] \\
&= \nabla_{\mathbf{z}}\mathbb{E}_{\boldsymbol{\eta}\sim\mathcal{N}(\mathbf{0},I)}[\log p(\mathbf{z}) + \log p(\mathbf{x}|\mathbf{z})] + \mathbf{0} + \mathbf{0} \\
&= \mathbb{E}_{\boldsymbol{\eta}\sim\mathcal{N}(\mathbf{0},I)}[\nabla_{\mathbf{z}}[\log p(\mathbf{z}) + \log p(\mathbf{x}|\mathbf{z})]] \\
&= \mathbb{E}_{\boldsymbol{\eta}\sim\mathcal{N}(\mathbf{0},I)}\left[\nabla_{\mathbf{z}}\log p(\mathbf{z}) + \nabla_{\mathbf{z}}\log p(\mathbf{x}|\mathbf{z})\right]
\end{aligned}
\tag{B.12}
$$

$$\begin{aligned}
\nabla_{\omega} ELBO[q(\mathbf{z})] &= \nabla_{\omega}\left[\mathbb{E}_{\eta\sim\mathcal{N}(\mathbf{0},I)}[\log p(\mathbf{z}) + \log p(\mathbf{x}|\mathbf{z})] + \frac{d}{2}(1+\log(2\pi)) + \frac{1}{2}\log|\Sigma|\right] \\
&= \nabla_{\omega}\mathbb{E}_{\eta\sim\mathcal{N}(\mathbf{0},I)}[\log p(\mathbf{z}) + \log p(\mathbf{x}|\mathbf{z})] \\
&\qquad + \nabla_{\omega}\left[\frac{d}{2}(1+\log(2\pi))\right] + \nabla_{\omega}\left[\frac{1}{2}\log|\Sigma|\right] \\
&= \mathbb{E}_{\eta\sim\mathcal{N}(\mathbf{0},I)}[\nabla_{\omega}[\log p(\mathbf{z}) + \log p(\mathbf{x}|\mathbf{z})]] + \mathbf{0} + \frac{1}{2}\nabla_{\omega}[\log|\Sigma|] \\
&= \mathbb{E}_{\eta\sim\mathcal{N}(\mathbf{0},I)}\left[\nabla_{\mathbf{z}}[\log p(\mathbf{z}) + \log p(\mathbf{x}|\mathbf{z})]\cdot\nabla_{\omega}\left[\Sigma^{\frac{1}{2}}\boldsymbol{\eta}\right]\right] + \frac{1}{2}\left(\frac{\nabla_{\omega}|\Sigma|}{|\Sigma|}\right) \\
&= \mathbb{E}_{\eta\sim\mathcal{N}(\mathbf{0},I)}\left[(\nabla_{\mathbf{z}}\log p(\mathbf{z}) + \nabla_{\mathbf{z}}\log p(\mathbf{x}|\mathbf{z}))\cdot\nabla_{\omega}\left[\Sigma^{\frac{1}{2}}\boldsymbol{\eta}\right]\right] + \frac{\nabla_{\omega}|\Sigma|}{2|\Sigma|}
\end{aligned} \tag{B.13}$$

$$\begin{aligned}
\nabla_{\alpha} ELBO[q(\mathbf{z})] &= \nabla_{\alpha}\left[\mathbb{E}_{\eta\sim\mathcal{N}(\mathbf{0},I)}[\log p(\mathbf{z}) + \log p(\mathbf{x}|\mathbf{z})] + \frac{d}{2}(1+\log(2\pi)) + \frac{1}{2}\log|\Sigma|\right] \\
&= \nabla_{\alpha}\mathbb{E}_{\eta\sim\mathcal{N}(\mathbf{0},I)}[\log p(\mathbf{z}) + \log p(\mathbf{x}|\mathbf{z})] \\
&\qquad + \nabla_{\alpha}\left[\frac{d}{2}(1+\log(2\pi))\right] + \nabla_{\alpha}\left[\frac{1}{2}\log|\Sigma|\right] \\
&= \mathbb{E}_{\eta\sim\mathcal{N}(\mathbf{0},I)}[\nabla_{\alpha}[\log p(\mathbf{z}) + \log p(\mathbf{x}|\mathbf{z})]] + \mathbf{0} + \frac{1}{2}\nabla_{\alpha}[\log|\Sigma|] \\
&= \mathbb{E}_{\eta\sim\mathcal{N}(\mathbf{0},I)}\left[\nabla_{\mathbf{z}}[\log p(\mathbf{z}) + \log p(\mathbf{x}|\mathbf{z})]\cdot\nabla_{\alpha}\left[\Sigma^{\frac{1}{2}}\boldsymbol{\eta}\right]\right] + \frac{1}{2}\left(\frac{\nabla_{\alpha}|\Sigma|}{|\Sigma|}\right) \\
&= \mathbb{E}_{\eta\sim\mathcal{N}(\mathbf{0},I)}\left[(\nabla_{\mathbf{z}}\log p(\mathbf{z}) + \nabla_{\mathbf{z}}\log p(\mathbf{x}|\mathbf{z}))\cdot\nabla_{\alpha}\left[\Sigma^{\frac{1}{2}}\boldsymbol{\eta}\right]\right] + \frac{\nabla_{\alpha}|\Sigma|}{2|\Sigma|}
\end{aligned} \tag{B.14}$$

i The first equality in each of derivations B.12, B.13 and B.14 follow from an application of the equivalence of the first and last expressions in derivation 2.4.

ii As the scalar expression $\frac{d}{2}(1+\log(2\pi))$ does not depend on either $\boldsymbol{\mu}$, $\boldsymbol{\omega}$ or $\boldsymbol{\alpha}$, then the derivatives of this expression with respect to each of $\boldsymbol{\mu}$, $\boldsymbol{\omega}$ and $\boldsymbol{\alpha}$ will be equal to $\mathbf{0}$.

iii As Σ does not depend on $\boldsymbol{\mu}$ (see equations B.10), then $\nabla_{\mu}\left[\frac{1}{2}\log|\Sigma|\right] = \mathbf{0}$ (relevant to derivation B.12).

iv For the third equality in derivation B.12, and for the forth equalities in each of derivations B.13 and B.14, equality of the first and last expressions in each of derivations B.11 ($\nabla_{\mu}[y] = \nabla_{\mathbf{z}}[y]$, $\nabla_{\omega}[y] = \nabla_{\mathbf{z}}[y]\cdot\nabla_{\omega}[\Sigma^{\frac{1}{2}}\boldsymbol{\eta}]$ and $\nabla_{\alpha}[y] = \nabla_{\mathbf{z}}[y]\cdot\nabla_{\alpha}[\Sigma^{\frac{1}{2}}\boldsymbol{\eta}]$, respectively) have been applied, where appropriate.

v For the forth equality in derivation B.12, and for the third equalities in each of derivations B.13 and B.14, as $\boldsymbol{\eta}$ (distributed as $\mathcal{N}(\mathbf{0},I)$) does not depend on either $\boldsymbol{\mu}$, $\boldsymbol{\omega}$ or $\boldsymbol{\alpha}$, then $\nabla_{\mu}\mathbb{E}_{\eta\sim\mathcal{N}(\mathbf{0},I)}[y] = \mathbb{E}_{\eta\sim\mathcal{N}(\mathbf{0},I)}[\nabla_{\mu}y]$, $\nabla_{\omega}\mathbb{E}_{\eta\sim\mathcal{N}(\mathbf{0},I)}[y] = \mathbb{E}_{\eta\sim\mathcal{N}(\mathbf{0},I)}[\nabla_{\omega}y]$ and $\nabla_{\alpha}\mathbb{E}_{\eta\sim\mathcal{N}(\mathbf{0},I)}[y] = \mathbb{E}_{\eta\sim\mathcal{N}(\mathbf{0},I)}[\nabla_{\alpha}y]$, for a generic function y.

vi Automatic differentiation can now be used to calculate values for each of the derivatives in the last expressions of each of derivations B.12, B.13 and B.14; given set values of the variational parameters in $\boldsymbol{\mu}$, $\boldsymbol{\omega}$ and $\boldsymbol{\alpha}$; and given a sample of $\boldsymbol{\eta}$ (distributed as $\mathcal{N}(\mathbf{0}, I)$), as required for **MC** integration.

B.4 Copula ADVI on the Iris Dataset, in Python

In this appendix section, just as in section 2.3, my original approach to the copula augmentation of the mean-field approach to **ADVI**, as outlined in the paper (Kucukelbir et al., 2017), is referred to as "copula **ADVI**".

The following script is a Python script that executes the copula **ADVI** algorithm on the iris dataset, via a cross-validation based approach for testing purposes. Please refer to the comments in the script, and section 3.1, for guidance as to how the script has been written, compiled and structured.

```
# Thinly-wrapped numpy, for automatic differentiation
import autograd.numpy as np

from autograd import jacobian
import pandas as pd
import random
import time

# The iris dataset is available through the sklearn (scikit-learn)
# library
from sklearn.datasets import load_iris

#############################################

iris_dataset = load_iris().data
iris_dataset = pd.DataFrame(iris_dataset)

#############################################

# Hyperparameters
epsilon = 0.01          # Convergence Measure
iter_limit = 100        # Iteration Limit
delta_first = 0.01      # Initial Step Size
delta_last = 0.001      # Final Step Size
sample_size = 100       # Eta Sample Size
cross_val_fold = 2      # Cross Validation Fold Number

z_pos = 5               # Omega_T, for the Prior
z_neg = -5              # Omega_F, for the Prior
s = 1                   # Prior Gaussian Component std

#############################################

# The order of the classes is: "Setosa", "Versicolour", "Virginica"
class_list = np.array([[z_pos, z_neg, z_neg], [z_neg, z_pos, z_neg],
                       [z_neg, z_neg, z_pos]])

# Set of the sampled eta values
eta_set = np.random.normal(0, 1, sample_size*3).reshape(sample_size, 3)
eta = np.array([0.0, 0.0, 0.0])

# Random sets of indices of the iris flowers, used for testing in
# consecutive folds of cross-validation
testing_index_list = np.arange(150)
random.shuffle(testing_index_list)
testing_index_list = testing_index_list.reshape(cross_val_fold,
                         int(len(testing_index_list) / cross_val_fold))

##############################################

# z is 3D, x is 4D


def prior(z_a):
    Gaus_seto = ((-0.5)*(((z_a[0]-z_pos)/s)**2)
              + (-0.5)*(((z_a[1]-z_neg)/s)**2)
              + (-0.5)*(((z_a[2]-z_neg)/s)**2))
    Gaus_seto = (1/((2*np.pi)**(1.5))) * np.exp(Gaus_seto)

    Gaus_vers = ((-0.5)*(((z_a[0]-z_neg)/s)**2)
              + (-0.5)*(((z_a[1]-z_pos)/s)**2)
              + (-0.5)*(((z_a[2]-z_neg)/s)**2))
    Gaus_vers = (1/((2*np.pi)**(1.5))) * np.exp(Gaus_vers)

    Gaus_virg = ((-0.5)*(((z_a[0]-z_neg)/s)**2)
              + (-0.5)*(((z_a[1]-z_neg)/s)**2)
              + (-0.5)*(((z_a[2]-z_pos)/s)**2))
    Gaus_virg = (1/((2*np.pi)**(1.5))) * np.exp(Gaus_virg)

    return ((beta_seto*Gaus_seto) + (beta_vers*Gaus_vers)
          + (beta_virg*Gaus_virg))


# Manually calculated gradient, w.r.t. z, of the log of the prior
def grad_log_prior(z_b):
    Gaus_seto = ((-0.5)*(((z_b[0]-z_pos)/s)**2)
              + (-0.5)*(((z_b[1]-z_neg)/s)**2)
              + (-0.5)*(((z_b[2]-z_neg)/s)**2))
    Gaus_seto = (1/((2*np.pi)**(1.5))) * np.exp(Gaus_seto)

    Gaus_vers = ((-0.5)*(((z_b[0]-z_neg)/s)**2)
              + (-0.5)*(((z_b[1]-z_pos)/s)**2)
              + (-0.5)*(((z_b[2]-z_neg)/s)**2))
    Gaus_vers = (1/((2*np.pi)**(1.5))) * np.exp(Gaus_vers)

    Gaus_virg = ((-0.5)*(((z_b[0]-z_neg)/s)**2)
              + (-0.5)*(((z_b[1]-z_neg)/s)**2)
              + (-0.5)*(((z_b[2]-z_pos)/s)**2))
    Gaus_virg = (1/((2*np.pi)**(1.5))) * np.exp(Gaus_virg)

    comp_1 = ((beta_seto*Gaus_seto*((z_pos-z_b[0])/s))
           + (beta_vers*Gaus_vers*((z_neg-z_b[0])/s))
           + (beta_virg*Gaus_virg*((z_neg-z_b[0])/s)))

    comp_2 = ((beta_seto*Gaus_seto*((z_neg-z_b[1])/s))
           + (beta_vers*Gaus_vers*((z_pos-z_b[1])/s))
           + (beta_virg*Gaus_virg*((z_neg-z_b[1])/s)))

    comp_3 = ((beta_seto*Gaus_seto*((z_neg-z_b[2])/s))
           + (beta_vers*Gaus_vers*((z_neg-z_b[2])/s))
           + (beta_virg*Gaus_virg*((z_pos-z_b[2])/s)))

    return np.array([comp_1 / prior(z_b), comp_2 / prior(z_b),
                     comp_3 / prior(z_b)])
```

```python
def log_likelihood(z_c):
    mean = ((((z_c[0]-z_neg)/(z_pos-z_neg))*mean_seto)
          + (((z_c[1]-z_neg)/(z_pos-z_neg))*mean_vers)
          + (((z_c[2]-z_neg)/(z_pos-z_neg))*mean_virg))
    var = np.exp((((z_c[0]-z_neg)/(z_pos-z_neg))*np.log(var_seto))
               + (((z_c[1]-z_neg)/(z_pos-z_neg))*np.log(var_vers))
               + (((z_c[2]-z_neg)/(z_pos-z_neg))*np.log(var_virg)))
    log_likelihoods = (np.log(1 / np.sqrt(2*np.pi*var))
                            - (0.5 * (((x-mean)**2) / var)))
    return sum(log_likelihoods)


# Manually calculated gradient, w.r.t. z, of the log of the likelihood
def grad_log_likelihood(z_d):
    comp_1 = 0.0
    comp_2 = 0.0
    comp_3 = 0.0

    for i in range(4):
        mu_i = ((((z_d[0]-z_neg)/(z_pos-z_neg))*mean_seto[i])
              + (((z_d[1]-z_neg)/(z_pos-z_neg))*mean_vers[i])
              + (((z_d[2]-z_neg)/(z_pos-z_neg))*mean_virg[i]))
        var_i = np.exp(
                (((z_d[0]-z_neg)/(z_pos-z_neg))*np.log(var_seto[i]))
              + (((z_d[1]-z_neg)/(z_pos-z_neg))*np.log(var_vers[i]))
              + (((z_d[2]-z_neg)/(z_pos-z_neg))*np.log(var_virg[i])))
        del_z_mu_i = np.array([mean_seto[i] / (z_pos-z_neg),
                               mean_vers[i] / (z_pos-z_neg),
                               mean_virg[i] / (z_pos-z_neg)])
        del_z_var_i = np.array(
                       [(np.log(var_seto[i])*var_i) / (z_pos-z_neg),
                        (np.log(var_vers[i])*var_i) / (z_pos-z_neg),
                        (np.log(var_virg[i])*var_i) / (z_pos-z_neg)])
        del_z_mu_i_squared = np.array(
                           [(2*mean_seto[i]*mu_i) / (z_pos-z_neg),
                            (2*mean_vers[i]*mu_i) / (z_pos-z_neg),
                            (2*mean_virg[i]*mu_i) / (z_pos-z_neg)])

        comp_1 = (comp_1
            + ((del_z_var_i[0] + del_z_mu_i_squared[0]
                - (2*x[i]*del_z_mu_i[0])) / var_i)
            - ((del_z_var_i[0]*(((x[i])**2) - (2*x[i]*mu_i)
                + (mu_i**2))) / (var_i**2)))
        comp_2 = (comp_2
            + ((del_z_var_i[1] + del_z_mu_i_squared[1]
                - (2*x[i]*del_z_mu_i[1])) / var_i)
            - ((del_z_var_i[1]*(((x[i])**2) - (2*x[i]*mu_i)
                + (mu_i**2))) / (var_i**2)))
        comp_3 = (comp_3
            + ((del_z_var_i[2] + del_z_mu_i_squared[2]
                - (2*x[i]*del_z_mu_i[2])) / var_i)
            - ((del_z_var_i[2]*(((x[i])**2) - (2*x[i]*mu_i)
                + (mu_i**2))) / (var_i**2)))

    comp_1 = (-0.5)*comp_1
    comp_2 = (-0.5)*comp_2
    comp_3 = (-0.5)*comp_3

    return np.array([comp_1, comp_2, comp_3])


# Covariance matrix of the surrogate posterior
```

```python
def Sigma(omega_e, alpha_e):
    a_11 = np.exp(2*omega_e[0])
    a_22 = np.exp(2*omega_e[1])
    a_33 = np.exp(2*omega_e[2])
    a_12 = ((2/np.pi) * np.exp(omega_e[0]+omega_e[1])
                      * np.arctan(alpha_e[0]))
    a_13 = ((2/np.pi) * np.exp(omega_e[0]+omega_e[2])
                      * np.arctan(alpha_e[1]))
    a_23 = ((2/np.pi) * np.exp(omega_e[1]+omega_e[2])
                      * np.arctan(alpha_e[2]))
    return np.array([[a_11, a_12, a_13], [a_12, a_22, a_23],
                     [a_13, a_23, a_33]])


# Square root of the covariance matrix of the surrogate posterior
def Sigma_sqrt(omega_f, alpha_f):
    # Calculating the square root of the target matrix via an
    # eigendecomposition approach
    eigenvalues, eigenvectors = np.linalg.eig(Sigma(omega_f, alpha_f))
    sqrt_eigenval_matrix = np.sqrt(np.diag(eigenvalues))
    eigenvectors_inv_mat = np.linalg.inv(eigenvectors)
    return np.matmul(np.matmul(eigenvectors, sqrt_eigenval_matrix),
                     eigenvectors_inv_mat)


def ELBO():
    ELBO_value = 0.0
    entropy = ((1.5 * (1 + np.log(2*np.pi)))
              + (0.5 * np.log(np.linalg.det(Sigma(omega, alpha)))))

    # The summation in MC integration
    for samp in range(sample_size):
        eta = eta_set[samp]
        z_i = np.matmul(Sigma_sqrt(omega, alpha), eta) + mu
        ELBO_value = ELBO_value + np.log(prior(z_i))
        ELBO_value = ELBO_value + log_likelihood(z_i)

    return (ELBO_value / sample_size) + entropy


def grad_mu_ELBO():
    grad_ELBO_value = np.array([0.0, 0.0, 0.0])

    # The summation in MC integration
    for samp in range(sample_size):
        eta = eta_set[samp]
        z_i = np.matmul(Sigma_sqrt(omega, alpha), eta) + mu
        grad_ELBO_value = grad_ELBO_value + grad_log_prior(z_i)
        grad_ELBO_value = grad_ELBO_value + grad_log_likelihood(z_i)

    return grad_ELBO_value / sample_size


def grad_omega_ELBO():
    grad_ELBO_value = np.array([0.0, 0.0, 0.0])
    grad_omega_det_Sigma = jacobian(
            lambda omega_g: np.linalg.det(Sigma(omega_g, alpha)))

    # The summation in MC integration
    for samp in range(sample_size):
        eta = eta_set[samp]
        z_i = np.matmul(Sigma_sqrt(omega, alpha), eta) + mu
        grad_omega_Sigma_sqrt_eta = jacobian(
            lambda omega_h: np.matmul(Sigma_sqrt(omega_h, alpha), eta))
        grad_ELBO_value = (grad_ELBO_value
                + (grad_log_prior(z_i)
                    * np.diag(grad_omega_Sigma_sqrt_eta(omega))))
        grad_ELBO_value = (grad_ELBO_value
                + (grad_log_likelihood(z_i)
                    * np.diag(grad_omega_Sigma_sqrt_eta(omega))))

    return ((grad_ELBO_value / sample_size)
                + (grad_omega_det_Sigma(omega)
                    / (2*np.linalg.det(Sigma(omega, alpha)))))


def grad_alpha_ELBO():
    grad_ELBO_value = np.array([0.0, 0.0, 0.0])
    grad_alpha_det_Sigma = jacobian(
                lambda alpha_j: np.linalg.det(Sigma(omega, alpha_j)))

    # The summation in MC integration
    for samp in range(sample_size):
        eta = eta_set[samp]
        z_i = np.matmul(Sigma_sqrt(omega, alpha), eta) + mu
        grad_alpha_Sigma_sqrt_eta = jacobian(
            lambda alpha_k: np.matmul(Sigma_sqrt(omega, alpha_k), eta))
        grad_ELBO_value = (grad_ELBO_value
                + (grad_log_prior(z_i)
                    * np.diag(grad_alpha_Sigma_sqrt_eta(alpha))))
        grad_ELBO_value = (grad_ELBO_value
                + (grad_log_likelihood(z_i)
                    * np.diag(grad_alpha_Sigma_sqrt_eta(alpha))))

    return ((grad_ELBO_value / sample_size)
                + (grad_alpha_det_Sigma(alpha)
                    / (2*np.linalg.det(Sigma(omega, alpha)))))


#############################################

# Each iteration of this for loop corresponds to a fold in
# cross-validation
for testing_indices in testing_index_list:
    current_xs = iris_dataset.iloc[testing_indices]

    # The setosa flowers are those with indices from 0 to 49
    # The versicolour flowers are those with indices from 50 to 99
    # The virginica flowers are those with indices from 100 to 149
    training_indices_seto = np.setdiff1d(
                            np.arange(0, 50), testing_indices)
    training_indices_vers = np.setdiff1d(
                            np.arange(50, 100), testing_indices)
    training_indices_virg = np.setdiff1d(
                            np.arange(100, 150), testing_indices)

    # Each of the following are 4-dimensional vectors (numpy arrays)
    mean_seto = iris_dataset.iloc[training_indices_seto].mean()
    mean_seto = mean_seto.to_numpy()
    mean_vers = iris_dataset.iloc[training_indices_vers].mean()
    mean_vers = mean_vers.to_numpy()
    mean_virg = iris_dataset.iloc[training_indices_virg].mean()
    mean_virg = mean_virg.to_numpy()
    var_seto = iris_dataset.iloc[training_indices_seto].var()
    var_seto = var_seto.to_numpy()
    var_vers = iris_dataset.iloc[training_indices_vers].var()
    var_vers = var_vers.to_numpy()
    var_virg = iris_dataset.iloc[training_indices_virg].var()
    var_virg = var_virg.to_numpy()

    # Each beta is a proportion of the respective flower types in a
    # given training dataset
    beta_seto = (len(training_indices_seto)
                    / (len(training_indices_seto)
                     + len(training_indices_vers)
                     + len(training_indices_virg)))
    beta_vers = (len(training_indices_vers)
                    / (len(training_indices_seto)
                     + len(training_indices_vers)
                     + len(training_indices_virg)))
    beta_virg = (len(training_indices_virg)
                    / (len(training_indices_seto)
                     + len(training_indices_vers)
                     + len(training_indices_virg)))

    for testing_index in testing_indices:
        x = current_xs.loc[testing_index].to_numpy()
        ELBO_list = []
        total_iteration_counter = 0
        total_key_time = 0.0
        for current_class in class_list:
            mu = current_class
            omega = np.array([0.0, 0.0, 0.0])
            alpha = np.array([0.0, 0.0, 0.0])

            current_ELBO = ELBO()
            iteration_counter = 0

            # This ensures that the statement in the following while
            # loop is always true for the first iteration
            last_ELBO = epsilon + 1

            while (abs(current_ELBO - last_ELBO) >= epsilon
                    and iteration_counter < iter_limit):
                iteration_counter += 1
                total_iteration_counter += 1
                last_ELBO = current_ELBO
                delta = (delta_first
                        * ((delta_last/delta_first)
                            **((iteration_counter-1)/(iter_limit-1))))

                # The gradient ascent step, which is timed for each
                # iteration
                t1 = time.perf_counter()
                mu = mu + (delta * grad_mu_ELBO())
                omega = omega + (delta * grad_omega_ELBO())
                alpha = alpha + (delta * grad_alpha_ELBO())
                current_ELBO = ELBO()
                t2 = time.perf_counter()

                total_key_time += (t2 - t1)

            ELBO_list.append(current_ELBO)

        if (ELBO_list[0] >= ELBO_list[1]
                and ELBO_list[0] >= ELBO_list[2]):
            print(testing_index, 'Setosa', total_iteration_counter,
                  total_key_time)
        elif (ELBO_list[1] >= ELBO_list[0]
                and ELBO_list[1] >= ELBO_list[2]):
            print(testing_index, 'Versicolour',
            total_iteration_counter, total_key_time)
        elif (ELBO_list[2] >= ELBO_list[0]
                and ELBO_list[2] >= ELBO_list[1]):
            print(testing_index, 'Virginica', total_iteration_counter,
            total_key_time)
```

B.5 The Manual Derivations of $\nabla_{\mathbf{z}} \log p(\mathbf{z})$ and $\nabla_{\mathbf{z}} \log p(\mathbf{x}|\mathbf{z})$

The prior $p(\mathbf{z})$ and likelihood $p(\mathbf{x}|\mathbf{z})$ are assumed to take the forms outlined in appendix sections B.1 and B.2, respectively, in the context of the iris dataset.

Therefore, given equation B.1, and with reference to the proceeding comments, we have:

$$p(\mathbf{z}) = \beta_{seto}\mathcal{N}(\mathbf{z}, \boldsymbol{\omega}_{seto}, S) + \beta_{vers}\mathcal{N}(\mathbf{z}, \boldsymbol{\omega}_{vers}, S) + \beta_{virg}\mathcal{N}(\mathbf{z}, \boldsymbol{\omega}_{virg}, S) \tag{B.15}$$

i "seto" refers to the setosa iris flower; "vers" refers to the versicolour iris flower; and "virg" refers to the virginica iris flower.

ii Let $\boldsymbol{\omega}_{seto} = \begin{pmatrix} \omega_T \\ \omega_F \\ \omega_F \end{pmatrix}$, $\boldsymbol{\omega}_{vers} = \begin{pmatrix} \omega_F \\ \omega_T \\ \omega_F \end{pmatrix}$ and $\boldsymbol{\omega}_{virg} = \begin{pmatrix} \omega_F \\ \omega_F \\ \omega_T \end{pmatrix}$.

iii β_{seto}, β_{vers} and β_{virg} are the proportions of setosa, versicolour and virginica iris flowers, respectively, in a given training dataset used to fit the prior $p(\mathbf{z})$.

iv Recall that $\mathbf{z} = \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix}$ and $S = \begin{pmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & s \end{pmatrix}$.

As a shorthand notation, let $G_{seto}(\mathbf{z}) = \mathcal{N}(\mathbf{z}, \boldsymbol{\omega}_{seto}, S)$, $G_{vers}(\mathbf{z}) = \mathcal{N}(\mathbf{z}, \boldsymbol{\omega}_{vers}, S)$ and $G_{virg}(\mathbf{z}) = \mathcal{N}(\mathbf{z}, \boldsymbol{\omega}_{virg}, S)$.

Given what has been outlined above, it is evident that each of the Gaussians $G_{seto}(\mathbf{z})$, $G_{vers}(\mathbf{z})$ and $G_{virg}(\mathbf{z})$ can be expressed as follows:

$$\begin{aligned}
G_{seto}(\mathbf{z}) &= \frac{1}{(2\pi)^{3/2}} \exp\left\{ -\frac{1}{2}\left(\frac{z_1 - \omega_T}{s}\right)^2 - \frac{1}{2}\left(\frac{z_2 - \omega_F}{s}\right)^2 - \frac{1}{2}\left(\frac{z_3 - \omega_F}{s}\right)^2 \right\} \\
G_{vers}(\mathbf{z}) &= \frac{1}{(2\pi)^{3/2}} \exp\left\{ -\frac{1}{2}\left(\frac{z_1 - \omega_F}{s}\right)^2 - \frac{1}{2}\left(\frac{z_2 - \omega_T}{s}\right)^2 - \frac{1}{2}\left(\frac{z_3 - \omega_F}{s}\right)^2 \right\} \\
G_{virg}(\mathbf{z}) &= \frac{1}{(2\pi)^{3/2}} \exp\left\{ -\frac{1}{2}\left(\frac{z_1 - \omega_F}{s}\right)^2 - \frac{1}{2}\left(\frac{z_2 - \omega_F}{s}\right)^2 - \frac{1}{2}\left(\frac{z_3 - \omega_T}{s}\right)^2 \right\}
\end{aligned} \tag{B.16}$$

Given the first of equations B.16, it can thus be deduced that:

$$\begin{aligned}
\nabla_{z_1}\left[G_{seto}(\mathbf{z})\right] &= \nabla_{z_1}\left[\frac{1}{(2\pi)^{3/2}}\exp\left\{-\frac{1}{2}\left(\frac{z_1-\omega_T}{s}\right)^2-\frac{1}{2}\left(\frac{z_2-\omega_F}{s}\right)^2-\frac{1}{2}\left(\frac{z_3-\omega_F}{s}\right)^2\right\}\right] \\
&= \frac{1}{(2\pi)^{3/2}}\nabla_{z_1}\left[\exp\left\{-\frac{1}{2}\left(\frac{z_1-\omega_T}{s}\right)^2-\frac{1}{2}\left(\frac{z_2-\omega_F}{s}\right)^2-\frac{1}{2}\left(\frac{z_3-\omega_F}{s}\right)^2\right\}\right] \\
&= \frac{1}{(2\pi)^{3/2}}\cdot\nabla_{z_1}\left[-\frac{1}{2}\left(\frac{z_1-\omega_T}{s}\right)^2-\frac{1}{2}\left(\frac{z_2-\omega_F}{s}\right)^2-\frac{1}{2}\left(\frac{z_3-\omega_F}{s}\right)^2\right] \\
&\qquad\cdot\exp\left\{-\frac{1}{2}\left(\frac{z_1-\omega_T}{s}\right)^2-\frac{1}{2}\left(\frac{z_2-\omega_F}{s}\right)^2-\frac{1}{2}\left(\frac{z_3-\omega_F}{s}\right)^2\right\} \\
&= G_{seto}(\mathbf{z})\cdot\nabla_{z_1}\left[-\frac{1}{2}\left(\frac{z_1-\omega_T}{s}\right)^2\right] \\
&= G_{seto}(\mathbf{z})\cdot\left(-\frac{1}{2}\right)\cdot 2\cdot\frac{z_1-\omega_T}{s} \\
&= \frac{\omega_T-z_1}{s}G_{seto}(\mathbf{z})
\end{aligned} \tag{B.17}$$

Given the patterns of z_1, z_2, z_3, ω_T and ω_F among equations B.16, and derivations analogous to that of derivation B.17, it is thus evident that:

$$\begin{aligned}
&\nabla_{z_1}\left[G_{seto}(\mathbf{z})\right] = \frac{\omega_T-z_1}{s}G_{seto}(\mathbf{z}), \quad \nabla_{z_1}\left[G_{vers}(\mathbf{z})\right] = \frac{\omega_F-z_1}{s}G_{vers}(\mathbf{z}), \\
&\nabla_{z_1}\left[G_{virg}(\mathbf{z})\right] = \frac{\omega_F-z_1}{s}G_{virg}(\mathbf{z}), \quad \nabla_{z_2}\left[G_{seto}(\mathbf{z})\right] = \frac{\omega_F-z_2}{s}G_{seto}(\mathbf{z}), \\
&\nabla_{z_2}\left[G_{vers}(\mathbf{z})\right] = \frac{\omega_T-z_2}{s}G_{vers}(\mathbf{z}), \quad \nabla_{z_2}\left[G_{virg}(\mathbf{z})\right] = \frac{\omega_F-z_2}{s}G_{virg}(\mathbf{z}), \\
&\nabla_{z_3}\left[G_{seto}(\mathbf{z})\right] = \frac{\omega_F-z_3}{s}G_{seto}(\mathbf{z}), \quad \nabla_{z_3}\left[G_{vers}(\mathbf{z})\right] = \frac{\omega_F-z_3}{s}G_{vers}(\mathbf{z}), \quad \text{and} \\
&\nabla_{z_3}\left[G_{virg}(\mathbf{z})\right] = \frac{\omega_T-z_3}{s}G_{virg}(\mathbf{z})
\end{aligned} \tag{B.18}$$

A manual derivation of $\nabla_{\mathbf{z}}\log p(\mathbf{z})$ can thus be given as follows:

$$
\begin{aligned}
\nabla_{\mathbf{z}} \log p(\mathbf{z}) &= \begin{pmatrix} \nabla_{z_1} \\ \nabla_{z_2} \\ \nabla_{z_3} \end{pmatrix} \log p(\mathbf{z}) \\
&= \begin{pmatrix} \nabla_{z_1} \log p(\mathbf{z}) \\ \nabla_{z_2} \log p(\mathbf{z}) \\ \nabla_{z_3} \log p(\mathbf{z}) \end{pmatrix} \\
&= \begin{pmatrix} \frac{\nabla_{z_1} p(\mathbf{z})}{p(\mathbf{z})} \\ \frac{\nabla_{z_2} p(\mathbf{z})}{p(\mathbf{z})} \\ \frac{\nabla_{z_3} p(\mathbf{z})}{p(\mathbf{z})} \end{pmatrix} \\
&= \frac{1}{p(\mathbf{z})} \begin{pmatrix} \nabla_{z_1} p(\mathbf{z}) \\ \nabla_{z_2} p(\mathbf{z}) \\ \nabla_{z_3} p(\mathbf{z}) \end{pmatrix} \\
&= \frac{1}{p(\mathbf{z})} \begin{pmatrix} \nabla_{z_1} \left[\beta_{seto} G_{seto}(\mathbf{z}) + \beta_{vers} G_{vers}(\mathbf{z}) + \beta_{virg} G_{virg}(\mathbf{z})\right] \\ \nabla_{z_2} \left[\beta_{seto} G_{seto}(\mathbf{z}) + \beta_{vers} G_{vers}(\mathbf{z}) + \beta_{virg} G_{virg}(\mathbf{z})\right] \\ \nabla_{z_3} \left[\beta_{seto} G_{seto}(\mathbf{z}) + \beta_{vers} G_{vers}(\mathbf{z}) + \beta_{virg} G_{virg}(\mathbf{z})\right] \end{pmatrix} \\
&= \frac{1}{p(\mathbf{z})} \begin{pmatrix} \beta_{seto} \nabla_{z_1} \left[G_{seto}(\mathbf{z})\right] + \beta_{vers} \nabla_{z_1} \left[G_{vers}(\mathbf{z})\right] + \beta_{virg} \nabla_{z_1} \left[G_{virg}(\mathbf{z})\right] \\ \beta_{seto} \nabla_{z_2} \left[G_{seto}(\mathbf{z})\right] + \beta_{vers} \nabla_{z_2} \left[G_{vers}(\mathbf{z})\right] + \beta_{virg} \nabla_{z_2} \left[G_{virg}(\mathbf{z})\right] \\ \beta_{seto} \nabla_{z_3} \left[G_{seto}(\mathbf{z})\right] + \beta_{vers} \nabla_{z_3} \left[G_{vers}(\mathbf{z})\right] + \beta_{virg} \nabla_{z_3} \left[G_{virg}(\mathbf{z})\right] \end{pmatrix} \\
&= \frac{1}{p(\mathbf{z})} \begin{pmatrix} \beta_{seto} \left(\frac{\omega_T - z_1}{s} G_{seto}(\mathbf{z})\right) + \beta_{vers} \left(\frac{\omega_F - z_1}{s} G_{vers}(\mathbf{z})\right) + \beta_{virg} \left(\frac{\omega_F - z_1}{s} G_{virg}(\mathbf{z})\right) \\ \beta_{seto} \left(\frac{\omega_F - z_2}{s} G_{seto}(\mathbf{z})\right) + \beta_{vers} \left(\frac{\omega_T - z_2}{s} G_{vers}(\mathbf{z})\right) + \beta_{virg} \left(\frac{\omega_F - z_2}{s} G_{virg}(\mathbf{z})\right) \\ \beta_{seto} \left(\frac{\omega_F - z_3}{s} G_{seto}(\mathbf{z})\right) + \beta_{vers} \left(\frac{\omega_F - z_3}{s} G_{vers}(\mathbf{z})\right) + \beta_{virg} \left(\frac{\omega_T - z_3}{s} G_{virg}(\mathbf{z})\right) \end{pmatrix}
\end{aligned}
\tag{B.19}
$$

i The fifth equality in derivation B.19 follows from a substitution of equation B.15.

ii The sixth equality in derivation B.19 follows from the fact that β_{seto}, β_{vers} and β_{virg} do not depend on any of z_1, z_2 and z_3.

iii The last equality in derivation B.19 follows from substitutions of each of equations B.18.

iv The last expression in derivation B.19 is the expression that is used to explicitly calculate $\nabla_{\mathbf{z}} \log p(\mathbf{z})$ in the main copula **ADVI** Python script (see appendix section B.4).

Recall from appendix section B.2 that the likelihood $p(\mathbf{x}|\mathbf{z})$ is a mean-field Gaussian of the same dimension as $\mathbf{x}$. As $\mathbf{x}$ has a dimension of 4 for the iris dataset (see section 2.2), and given the general structure of $p(\mathbf{x}|\mathbf{z})$ as per appendix section B.2, the likelihood can thus be expressed as follows (please refer to the proceeding comments):

$$
p(\mathbf{x}|\mathbf{z}) = \prod_{i=1}^{4} \left[\frac{1}{\sqrt{2\pi v_i(\mathbf{z})}} \exp\left\{ -\frac{\left(x_i - \mu_i(\mathbf{z})\right)^2}{2v_i(\mathbf{z})} \right\} \right] \tag{B.20}
$$

where

$$
\mu_i(\mathbf{z}) = \frac{z_1 - \omega_F}{\omega_T - \omega_F} \mu_{i,seto} + \frac{z_2 - \omega_F}{\omega_T - \omega_F} \mu_{i,vers} + \frac{z_3 - \omega_F}{\omega_T - \omega_F} \mu_{i,virg} \tag{B.21}
$$

and

$$v_i(\mathbf{z}) = \exp\left\{\frac{z_1 - \omega_F}{\omega_T - \omega_F}\log\left(v_{i,seto}\right) + \frac{z_2 - \omega_F}{\omega_T - \omega_F}\log\left(v_{i,vers}\right) + \frac{z_3 - \omega_F}{\omega_T - \omega_F}\log\left(v_{i,virg}\right)\right\} \tag{B.22}$$

i Just as for the prior above: "seto" refers to the setosa iris flower; "vers" refers to the versicolour iris flower; and "virg" refers to the virginica iris flower.

ii $\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}$ and $\mathbf{z} = \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix}$.

iii Let $i = 1$, $i = 2$, $i = 3$ and $i = 4$ refer to the feature variables "sepal length", "sepal width", "petal length" and "petal width", respectively. The ordering of the feature variables in this sense is arbitrary, but has been chosen in this way as this is the ordering of the iris feature variables in the "sklearn" Python package, from where the iris dataset is sourced in the main copula **ADVI** Python script (see appendix section B.4).

iv The quantities $\mu_{i,seto}$ and $v_{i,seto}$ are the mean and variance values, respectively, of feature i for the setosa flowers in the training dataset used to fit the likelihood. Analogous definitions apply to the quantities $\mu_{i,vers}$, $\mu_{i,virg}$, $v_{i,vers}$ and $v_{i,virg}$.

v The mean and variance functions given by equations B.21 and B.22, respectively, are analogous in structure to the third expressions derivations B.4 and B.5, respectively.

vi Just as for the prior, let the order of the target classes be: "setosa", "versicolour", "virginica" (for example, when $\mathbf{z} = (\omega_T \quad \omega_F \quad \omega_F)^T$, then $\mu_i(\mathbf{z}) = \mu_{i,seto}$).

An expressible form of the scalar derivative $\nabla_{z_k} \log p(\mathbf{x}|\mathbf{z})$, for some $k \in \{1, 2, 3\}$, can thus be derived as follows:

$$
\begin{aligned}
\nabla_{z_k} \log p(\mathbf{x}|\mathbf{z}) &= \nabla_{z_k} \log \left(\prod_{i=1}^{4} \left[\frac{1}{\sqrt{2\pi v_i(\mathbf{z})}} \exp\left\{ -\frac{(x_i - \mu_i(\mathbf{z}))^2}{2v_i(\mathbf{z})} \right\} \right] \right) \\
&= \nabla_{z_k} \sum_{i=1}^{4} \left[\log \left(\frac{1}{\sqrt{2\pi v_i(\mathbf{z})}} \exp\left\{ -\frac{(x_i - \mu_i(\mathbf{z}))^2}{2v_i(\mathbf{z})} \right\} \right) \right] \\
&= \nabla_{z_k} \sum_{i=1}^{4} \left[\log \left(\frac{1}{\sqrt{2\pi v_i(\mathbf{z})}} \right) - \frac{(x_i - \mu_i(\mathbf{z}))^2}{2v_i(\mathbf{z})} \right] \\
&= \nabla_{z_k} \sum_{i=1}^{4} \left[-\frac{1}{2} \log\left(2\pi v_i(\mathbf{z})\right) - \frac{x_i^2 - 2x_i\mu_i(\mathbf{z}) + \mu_i^2(\mathbf{z})}{2v_i(\mathbf{z})} \right] \\
&= -\frac{1}{2} \nabla_{z_k} \sum_{i=1}^{4} \left[\log\left(2\pi v_i(\mathbf{z})\right) + \frac{x_i^2 - 2x_i\mu_i(\mathbf{z}) + \mu_i^2(\mathbf{z})}{v_i(\mathbf{z})} \right] \\
&= -\frac{1}{2} \sum_{i=1}^{4} \left[\nabla_{z_k} \left[\log\left(2\pi v_i(\mathbf{z})\right)\right] + \nabla_{z_k} \left[\frac{x_i^2 - 2x_i\mu_i(\mathbf{z}) + \mu_i^2(\mathbf{z})}{v_i(\mathbf{z})} \right] \right] \\
&= -\frac{1}{2} \sum_{i=1}^{4} \left[\frac{\nabla_{z_k}[2\pi v_i(\mathbf{z})]}{2\pi v_i(\mathbf{z})} \right] - \frac{1}{2} \sum_{i=1}^{4} \left[\frac{v_i(\mathbf{z}) \nabla_{z_k}[x_i^2 - 2x_i\mu_i(\mathbf{z}) + \mu_i^2(\mathbf{z})]}{v_i^2(\mathbf{z})} \right. \\
&\qquad \left. - \frac{(x_i^2 - 2x_i\mu_i(\mathbf{z}) + \mu_i^2(\mathbf{z})) \nabla_{z_k}[v_i(\mathbf{z})]}{v_i^2(\mathbf{z})} \right] \\
&= -\frac{1}{2} \sum_{i=1}^{4} \left[\frac{2\pi \nabla_{z_k}[v_i(\mathbf{z})]}{2\pi v_i(\mathbf{z})} + \frac{\nabla_{z_k}[x_i^2 - 2x_i\mu_i(\mathbf{z}) + \mu_i^2(\mathbf{z})]}{v_i(\mathbf{z})} \right] \\
&\qquad + \frac{1}{2} \sum_{i=1}^{4} \left[\frac{(x_i^2 - 2x_i\mu_i(\mathbf{z}) + \mu_i^2(\mathbf{z})) \nabla_{z_k}[v_i(\mathbf{z})]}{v_i^2(\mathbf{z})} \right] \\
&= -\frac{1}{2} \sum_{i=1}^{4} \left[\frac{\nabla_{z_k}[v_i(\mathbf{z})]}{v_i(\mathbf{z})} + \frac{-2x_i \nabla_{z_k}[\mu_i(\mathbf{z})] + \nabla_{z_k}[\mu_i^2(\mathbf{z})]}{v_i(\mathbf{z})} \right] \\
&\qquad + \frac{1}{2} \sum_{i=1}^{4} \left[\frac{(x_i^2 - 2x_i\mu_i(\mathbf{z}) + \mu_i^2(\mathbf{z})) \nabla_{z_k}[v_i(\mathbf{z})]}{v_i^2(\mathbf{z})} \right] \\
&= -\frac{1}{2} \sum_{i=1}^{4} \left[\frac{\nabla_{z_k}[v_i(\mathbf{z})] - 2x_i \nabla_{z_k}[\mu_i(\mathbf{z})] + \nabla_{z_k}[\mu_i^2(\mathbf{z})]}{v_i(\mathbf{z})} \right. \\
&\qquad \left. - \frac{(x_i^2 - 2x_i\mu_i(\mathbf{z}) + \mu_i^2(\mathbf{z})) \nabla_{z_k}[v_i(\mathbf{z})]}{v_i^2(\mathbf{z})} \right]
\end{aligned}
\tag{B.23}
$$

i The first equality in derivation B.23 follows from a substitution of equation B.20.

ii The seventh equality in derivation B.23 follows, in part, from an application of the quotient rule for differentiation.

iii As x_i, for each $i \in \{1,2,3,4\}$, does not depend on z_k, for any $k \in \{1,2,3\}$, then $\nabla_{z_k}[x_i^2 - 2x_i\mu_i(\mathbf{z}) + \mu_i^2(\mathbf{z})] = \nabla_{z_k}[x_i^2] + \nabla_{z_k}[-2x_i\mu_i(\mathbf{z})] + \nabla_{z_k}[\mu_i^2(\mathbf{z})] = 0 - 2x_i\nabla_{z_k}[\mu_i(\mathbf{z})] + \nabla_{z_k}[\mu_i^2(\mathbf{z})] = -2x_i\nabla_{z_k}[\mu_i(\mathbf{z})] + \nabla_{z_k}[\mu_i^2(\mathbf{z})]$. This fact is reflected in the ninth equality in derivation B.23.

Expressible forms of the quantities $\nabla_{z_k}[\mu_i(\mathbf{z})]$, $\nabla_{z_k}[v_i(\mathbf{z})]$ and $\nabla_{z_k}\left[\mu_i^2(\mathbf{z})\right]$ from derivation B.23, in the particular case where $k = 1$, can be derived as follows:

$$\begin{aligned}
\nabla_{z_1}[\mu_i(\mathbf{z})] &= \nabla_{z_1}\left[\frac{z_1-\omega_F}{\omega_T-\omega_F}\mu_{i,seto}+\frac{z_2-\omega_F}{\omega_T-\omega_F}\mu_{i,vers}+\frac{z_3-\omega_F}{\omega_T-\omega_F}\mu_{i,virg}\right]\\
&= \nabla_{z_1}\left[\frac{z_1-\omega_F}{\omega_T-\omega_F}\mu_{i,seto}\right]\\
&= \frac{\mu_{i,seto}}{\omega_T-\omega_F}
\end{aligned} \tag{B.24}$$

$$\begin{aligned}
\nabla_{z_1}[v_i(\mathbf{z})] &= \nabla_{z_1}\Bigg[\exp\Bigg\{\frac{z_1-\omega_F}{\omega_T-\omega_F}\log\left(v_{i,seto}\right)+\frac{z_2-\omega_F}{\omega_T-\omega_F}\log\left(v_{i,vers}\right)\\
&\qquad\qquad+\frac{z_3-\omega_F}{\omega_T-\omega_F}\log\left(v_{i,virg}\right)\Bigg\}\Bigg]\\
&= \nabla_{z_1}\Bigg[\frac{z_1-\omega_F}{\omega_T-\omega_F}\log\left(v_{i,seto}\right)+\frac{z_2-\omega_F}{\omega_T-\omega_F}\log\left(v_{i,vers}\right)\\
&\qquad\qquad+\frac{z_3-\omega_F}{\omega_T-\omega_F}\log\left(v_{i,virg}\right)\Bigg]\cdot v_i(\mathbf{z})\\
&= \nabla_{z_1}\left[\frac{z_1-\omega_F}{\omega_T-\omega_F}\log\left(v_{i,seto}\right)\right]\cdot v_i(\mathbf{z})\\
&= \frac{\log\left(v_{i,seto}\right)}{\omega_T-\omega_F}v_i(\mathbf{z})
\end{aligned} \tag{B.25}$$

$$\begin{aligned}
\nabla_{z_1}\left[\mu_i^2(\mathbf{z})\right] &= \nabla_{z_1}\left[\left(\frac{z_1-\omega_F}{\omega_T-\omega_F}\mu_{i,seto}+\frac{z_2-\omega_F}{\omega_T-\omega_F}\mu_{i,vers}+\frac{z_3-\omega_F}{\omega_T-\omega_F}\mu_{i,virg}\right)^2\right]\\
&= \nabla_{z_1}\left[\left(\frac{z_1\mu_{i,seto}}{\omega_T-\omega_F}-\frac{\omega_F\mu_{i,seto}}{\omega_T-\omega_F}+\frac{z_2-\omega_F}{\omega_T-\omega_F}\mu_{i,vers}+\frac{z_3-\omega_F}{\omega_T-\omega_F}\mu_{i,virg}\right)^2\right]\\
&= \nabla_{z_1}\left[\left(\frac{z_1\mu_{i,seto}}{\omega_T-\omega_F}+A\right)^2\right]\\
&= \nabla_{z_1}\left[\frac{z_1^2\mu_{i,seto}^2}{(\omega_T-\omega_F)^2}+\frac{2Az_1\mu_{i,seto}}{\omega_T-\omega_F}+A^2\right]\\
&= \frac{2z_1\mu_{i,seto}^2}{(\omega_T-\omega_F)^2}+\frac{2A\mu_{i,seto}}{\omega_T-\omega_F}\\
&= \frac{2z_1\mu_{i,seto}^2}{(\omega_T-\omega_F)^2}+\frac{2\mu_{i,seto}}{\omega_T-\omega_F}\left(\mu_i(\mathbf{z})-\frac{z_1\mu_{i,seto}}{\omega_T-\omega_F}\right)\\
&= \frac{2z_1\mu_{i,seto}^2}{(\omega_T-\omega_F)^2}+\frac{2\mu_{i,seto}\mu_i(\mathbf{z})}{\omega_T-\omega_F}-\frac{2z_1\mu_{i,seto}^2}{(\omega_T-\omega_F)^2}\\
&= \frac{2\mu_{i,seto}\mu_i(\mathbf{z})}{\omega_T-\omega_F}
\end{aligned} \tag{B.26}$$

i The first equalities in derivations B.24 and B.26 each follow from a substitution of equation B.21. The first equality in derivation B.25 follows from a substitution of equation B.22.

ii The term A in derivation B.26 is the sum of all terms in $\mu_i(\mathbf{z})$ that do not include z_1, given equation B.21: $A = \mu_i(\mathbf{z}) - \frac{z_1\mu_{i,seto}}{\omega_T-\omega_F} = -\frac{\omega_F\mu_{i,seto}}{\omega_T-\omega_F} + \frac{z_2-\omega_F}{\omega_T-\omega_F}\mu_{i,vers} + \frac{z_3-\omega_F}{\omega_T-\omega_F}\mu_{i,virg}$.

Given the pattern of the appearances of z_1, z_2, z_3, ω_T and ω_F in equations B.21 and B.22, and derivations analogous to those of B.24, B.25 and B.26, it is thus evident that:

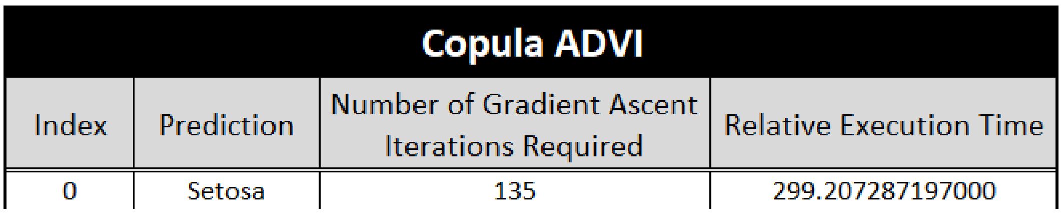
$$\begin{aligned}
&\nabla_{z_1}[v_i(\mathbf{z})] = \frac{\mu_{i,seto}}{\omega_T-\omega_F}, \quad \nabla_{z_1}[v_i(\mathbf{z})] = \frac{\log\left(v_{i,seto}\right)}{\omega_T-\omega_F}v_i(\mathbf{z}), \quad \nabla_{z_1}\left[\mu_i^2(\mathbf{z})\right] = \frac{2\mu_{i,seto}\mu_i(\mathbf{z})}{\omega_T-\omega_F},\\
&\nabla_{z_2}[v_i(\mathbf{z})] = \frac{\mu_{i,vers}}{\omega_T-\omega_F}, \quad \nabla_{z_2}[v_i(\mathbf{z})] = \frac{\log\left(v_{i,vers}\right)}{\omega_T-\omega_F}v_i(\mathbf{z}), \quad \nabla_{z_2}\left[\mu_i^2(\mathbf{z})\right] = \frac{2\mu_{i,vers}\mu_i(\mathbf{z})}{\omega_T-\omega_F},\\
&\nabla_{z_3}[v_i(\mathbf{z})] = \frac{\mu_{i,virg}}{\omega_T-\omega_F}, \quad \nabla_{z_3}[v_i(\mathbf{z})] = \frac{\log\left(v_{i,virg}\right)}{\omega_T-\omega_F}v_i(\mathbf{z}) \quad \text{and}\\
&\nabla_{z_3}\left[\mu_i^2(\mathbf{z})\right] = \frac{2\mu_{i,virg}\mu_i(\mathbf{z})}{\omega_T-\omega_F}
\end{aligned} \tag{B.27}$$

A manual derivation of $\nabla_{\mathbf{z}}\log p(\mathbf{x}|\mathbf{z})$ can thus be obtained via a combination of equations B.27 and the following:

$$\begin{aligned}
\nabla_{\mathbf{z}}\log p(\mathbf{x}|\mathbf{z}) &= \begin{pmatrix}\nabla_{z_1}\log p(\mathbf{x}|\mathbf{z})\\ \nabla_{z_2}\log p(\mathbf{x}|\mathbf{z})\\ \nabla_{z_3}\log p(\mathbf{x}|\mathbf{z})\end{pmatrix}\\
\nabla_{z_k}\log p(\mathbf{x}|\mathbf{z}) &= -\frac{1}{2}\sum_{i=1}^{4}\left[\frac{\nabla_{z_k}[v_i(\mathbf{z})] + \nabla_{z_k}[\mu_i^2(\mathbf{z})] - 2x_i\nabla_{z_k}[\mu_i(\mathbf{z})]}{v_i(\mathbf{z})}\right.\\
&\qquad\left. - \frac{(x_i^2 - 2x_i\mu_i(\mathbf{z}) + \mu_i^2(\mathbf{z}))\nabla_{z_k}[v_i(\mathbf{z})]}{v_i^2(\mathbf{z})}\right]\\
k &\in \{1,2,3\}
\end{aligned} \tag{B.28}$$

Equations B.27 and B.28 are what were used to explicitly calculate $\nabla_{\mathbf{z}}\log p(\mathbf{x}|\mathbf{z})$ in the main copula **ADVI** Python script (see appendix section B.4).

B.6 The Experimental Results (on the Iris Dataset)



Copula ADVI			
Index	Prediction	Number of Gradient Ascent Iterations Required	Relative Execution Time
0	Setosa	135	299.207287197000

0	Setosa	134	297.812484534999
1	Setosa	128	297.882809426009
1	Setosa	127	287.582823711003
2	Setosa	129	290.911041862997
2	Setosa	130	293.721680191002
3	Setosa	138	320.374275282001
3	Setosa	129	287.914902614986
4	Setosa	151	349.042822983003
4	Setosa	135	301.322336124998
5	Setosa	139	321.767806411009
5	Setosa	144	351.689257210995
6	Setosa	132	296.315585894024
6	Setosa	133	331.732846411016
7	Setosa	132	310.152029095000
7	Setosa	146	338.950988802997
8	Setosa	132	307.041226992996
8	Setosa	131	287.920139053000
9	Setosa	128	300.641726104991
9	Setosa	127	286.113953407000
10	Setosa	140	306.672172047001
10	Setosa	142	320.721228090034
11	Setosa	130	295.799935074994
11	Setosa	151	327.066514702015
12	Setosa	122	301.694441613995
12	Setosa	123	274.953705062005
13	Setosa	125	275.538113120992
13	Setosa	128	282.629141013991
14	Setosa	154	347.034293943012
14	Setosa	159	361.258835124998
15	Setosa	153	339.256983640996
15	Setosa	156	356.839183387992
16	Setosa	136	314.595518213999
16	Setosa	141	344.896543749999
17	Setosa	126	283.468683110008
17	Setosa	134	313.278476942999
18	Setosa	140	290.886761291004
18	Setosa	145	336.103121231995
19	Setosa	141	335.085683567000
19	Setosa	139	311.344570519999
20	Setosa	128	279.330432573999
20	Setosa	125	277.353310373011
21	Setosa	133	301.662768509020
21	Setosa	138	337.207760803996
22	Setosa	137	311.662936680998
22	Setosa	139	296.365997448001
23	Setosa	135	309.386783521993
23	Setosa	131	286.056341766999

24	Setosa	136	318.375674809996
24	Setosa	131	290.444472566994
25	Setosa	126	290.757640158001
25	Setosa	121	296.594701142981
26	Setosa	133	303.103841431990
26	Setosa	126	280.833080858004
27	Setosa	133	303.345812318005
27	Setosa	133	327.986068209007
28	Setosa	130	323.796910952002
28	Setosa	126	278.877440987995
29	Setosa	132	309.783574407998
29	Setosa	140	323.011581229000
30	Setosa	128	284.907537306004
30	Setosa	128	282.449498130006
31	Setosa	130	302.199756672012
31	Setosa	126	279.323132152006
32	Setosa	148	342.688697105983
32	Setosa	173	399.022115469004
33	Setosa	154	351.690795369017
33	Setosa	163	403.731153034987
34	Setosa	141	305.820828069003
34	Setosa	128	265.115892440000
35	Setosa	129	293.873245742002
35	Setosa	129	281.753930671986
36	Setosa	136	323.404442801999
36	Setosa	130	295.228196934989
37	Setosa	134	310.046525055999
37	Setosa	137	337.382109168997
38	Setosa	125	290.131486270000
38	Setosa	128	325.418471484999
39	Setosa	131	300.334764575001
39	Setosa	133	310.272882774998
40	Setosa	131	285.908557205984
40	Setosa	128	284.376125194998
41	Setosa	133	304.857699047996
41	Setosa	132	328.922657430999
42	Setosa	130	305.640573934994
42	Setosa	128	313.419117237994
43	Setosa	142	328.742392276002
43	Setosa	145	360.558860683015
44	Setosa	132	289.711628655004
44	Setosa	143	313.296158381002
45	Setosa	123	272.970781294996
45	Setosa	127	279.903362736007
46	Setosa	137	300.196155778001
46	Setosa	133	298.795637066997
47	Setosa	131	309.743520460009

47	Setosa	130	283.156325120993
48	Setosa	138	311.912463143993
48	Setosa	139	349.330036537999
49	Setosa	128	313.187311682004
49	Setosa	131	288.052963864001
50	Versicolour	175	398.627948251001
50	Versicolour	172	388.985204362999
51	Versicolour	163	381.680450300998
51	Versicolour	153	377.827010811994
52	Versicolour	159	360.808848915981
52	Versicolour	157	353.140376670997
53	Versicolour	160	377.323264841009
53	Versicolour	160	360.799975254991
54	Versicolour	154	347.173217143998
54	Versicolour	160	354.749560436998
55	Versicolour	163	374.958495937000
55	Versicolour	145	361.145214424002
56	Versicolour	156	359.319142102998
56	Versicolour	184	398.440722837999
57	Versicolour	158	359.607621527004
57	Versicolour	151	331.663488278987
58	Versicolour	159	370.362282639001
58	Versicolour	167	388.978415659998
59	Versicolour	160	370.195436464999
59	Versicolour	148	328.105128569985
60	Versicolour	163	384.263583452988
60	Versicolour	148	361.680218440993
61	Versicolour	149	376.169195657999
61	Versicolour	159	349.367431973003
62	Versicolour	159	390.737849970999
62	Versicolour	154	340.504918596000
63	Versicolour	153	340.653978527994
63	Versicolour	159	348.607332505993
64	Versicolour	143	354.299718330999
64	Versicolour	148	326.338255585997
65	Versicolour	148	367.670711001001
65	Versicolour	157	345.746136103998
66	Versicolour	151	334.231297221997
66	Versicolour	160	351.354238351017
67	Versicolour	154	338.006052631995
67	Versicolour	146	326.415278446995
68	Versicolour	169	393.344249396988
68	Versicolour	172	392.371436606994
69	Versicolour	157	369.911610159992
69	Versicolour	142	354.527456611984
70	Virginica	173	427.886490917011
70	Virginica	189	427.106825535989

71	Versicolour	147	321.248770828993
71	Versicolour	147	333.349919800000
72	Versicolour	171	381.706991037999
72	Versicolour	175	384.530900493000
73	Versicolour	149	323.320900947992
73	Versicolour	164	374.108236219024
74	Versicolour	164	372.334372264987
74	Versicolour	148	336.851370509002
75	Versicolour	162	356.225561724000
75	Versicolour	152	340.963319120998
76	Versicolour	172	397.516848156999
76	Versicolour	155	341.759332890002
77	Versicolour	190	440.987656746005
77	Versicolour	176	435.750606744995
78	Versicolour	161	372.522425103983
78	Versicolour	168	385.768436757014
79	Versicolour	155	363.921282375998
79	Versicolour	147	320.929824603999
80	Versicolour	148	332.014365517997
80	Versicolour	162	373.510620622002
81	Versicolour	157	353.561782673021
81	Versicolour	143	355.351331316000
82	Versicolour	157	364.584161182004
82	Versicolour	159	364.585114371016
83	Versicolour	184	423.472792275003
83	Versicolour	178	394.733158353987
84	Versicolour	162	378.649326608998
84	Versicolour	151	336.851791283001
85	Versicolour	173	377.644552013001
85	Versicolour	166	365.220063168988
86	Versicolour	153	375.676152795984
86	Versicolour	156	354.730313856001
87	Versicolour	154	318.346059989999
87	Versicolour	164	341.011583911998
88	Versicolour	158	371.571187353001
88	Versicolour	145	316.591043718996
89	Versicolour	158	365.008847544993
89	Versicolour	159	361.494722617007
90	Versicolour	144	353.596070861018
90	Versicolour	148	323.523138560016
91	Versicolour	168	381.967575729001
91	Versicolour	149	333.315780630997
92	Versicolour	156	363.602806293003
92	Versicolour	147	321.310583360993
93	Versicolour	150	334.422011451973
93	Versicolour	158	368.063194647999
94	Versicolour	143	314.743550453999

94	Versicolour	155	353.736236016001
95	Versicolour	145	355.396910155992
95	Versicolour	147	323.509807633017
96	Versicolour	161	367.400033586000
96	Versicolour	147	325.371263725988
97	Versicolour	163	382.095795218992
97	Versicolour	149	332.850516387003
98	Versicolour	156	361.953127222001
98	Versicolour	144	315.945204835999
99	Versicolour	143	312.341990148001
99	Versicolour	153	337.008933918003
100	Virginica	225	530.096783601992
100	Virginica	231	503.214276419004
101	Virginica	185	432.326670659000
101	Virginica	196	441.083706054001
102	Virginica	212	464.846018128998
102	Virginica	207	464.422276185998
103	Virginica	199	460.148522557999
103	Virginica	198	443.343073968995
104	Virginica	202	454.147841236994
104	Virginica	216	498.639051764993
105	Virginica	236	536.011418374003
105	Virginica	226	560.110424347009
106	Virginica	170	419.227926412018
106	Virginica	183	414.775328621988
107	Virginica	200	468.563257555008
107	Virginica	217	489.755083394997
108	Virginica	199	434.867352101999
108	Virginica	206	462.953639713996
109	Virginica	234	571.943471243972
109	Virginica	235	524.159832143010
110	Virginica	198	454.783584525999
110	Virginica	187	460.548433806998
111	Virginica	194	443.393957597998
111	Virginica	182	402.958000046030
112	Virginica	208	481.259163914992
112	Virginica	198	435.308136683999
113	Virginica	187	463.883445054987
113	Virginica	191	420.363620711001
114	Virginica	204	470.400654036013
114	Virginica	197	443.594882022995
115	Virginica	209	461.219831262007
115	Virginica	194	430.368136456992
116	Virginica	185	421.500963945999
116	Virginica	195	421.389507351005
117	Virginica	231	568.407774650979
117	Virginica	236	520.381201682996

118	Virginica	260	601.016401842999
118	Virginica	249	621.798024075017
119	Virginica	171	377.105129380010
119	Virginica	171	379.257997643002
120	Virginica	227	497.711564215002
120	Virginica	209	467.181501179997
121	Virginica	184	401.004678714003
121	Virginica	198	447.548401677042
122	Virginica	239	559.248989411047
122	Virginica	219	496.133036578001
123	Virginica	185	433.059713254999
123	Virginica	177	429.522579870999
124	Virginica	205	475.105092501993
124	Virginica	201	493.356121730917
125	Virginica	195	492.423888925000
125	Virginica	211	462.431842718006
126	Virginica	175	427.786188819001
126	Virginica	191	432.631190192991
127	Virginica	186	423.377631166005
127	Virginica	173	380.570838054000
128	Virginica	198	433.289777769999
128	Virginica	199	440.297661368001
129	Virginica	187	411.422525903998
129	Virginica	189	425.862039033017
130	Virginica	214	469.726072230993
130	Virginica	198	437.473768204985
131	Virginica	225	497.348504712001
131	Virginica	234	552.196262361001
132	Virginica	211	479.223588425016
132	Virginica	202	493.172221091997
133	Versicolour	181	408.889385016998
133	Versicolour	173	379.951249822002
134	Virginica	175	436.541905847005
134	Virginica	177	390.845528883017
135	Virginica	232	521.390511338009
135	Virginica	229	501.640551573000
136	Virginica	209	511.668017480992
136	Virginica	216	472.246659208016
137	Virginica	182	417.774206852995
137	Virginica	189	418.026313909038
138	Virginica	181	397.063832061019
138	Virginica	189	428.174745789994
139	Virginica	194	442.508985096988
139	Virginica	209	480.141814389995
140	Virginica	232	527.793121120003
140	Virginica	213	442.448502573001
141	Virginica	212	477.990336691997

141	Virginica	215	296.833135775001
142	Virginica	190	419.290760510000
142	Virginica	197	450.172769474996
143	Virginica	220	485.975370638993
143	Virginica	212	469.142794130006
144	Virginica	219	485.200305530992
144	Virginica	239	536.084444654021
145	Virginica	211	477.667898823995
145	Virginica	208	508.746400189999
146	Virginica	198	455.096390380006
146	Virginica	197	446.279798610004
147	Virginica	198	445.914356481984
147	Virginica	184	402.933746113985
148	Virginica	205	480.351202225010
148	Virginica	198	445.593285515002
149	Virginica	186	425.703713527999
149	Virginica	179	395.764770024003

Mean-Field ADVI			
Index	Prediction	Number of Gradient Ascent Iterations Required	Relative Execution Time
0	Setosa	133	176.120570859005
0	Setosa	131	161.082846599989
1	Setosa	127	189.978151162005
1	Setosa	127	168.085198544995
2	Setosa	131	177.120186942989
2	Setosa	127	161.469348128999
3	Setosa	127	160.705235314999
3	Setosa	132	173.568412189015
4	Setosa	134	178.203845219000
4	Setosa	133	169.198446854999
5	Setosa	133	175.376355994995
5	Setosa	131	171.173009825000
6	Setosa	137	169.175249069008
6	Setosa	132	176.631863864009
7	Setosa	129	189.639045404022
7	Setosa	130	171.040283380001
8	Setosa	131	179.862599665008
8	Setosa	125	165.171953621995
9	Setosa	126	166.314183440010
9	Setosa	123	160.538300905995
10	Setosa	136	180.692057497998
10	Setosa	138	185.800895318006
11	Setosa	138	182.586242958950
11	Setosa	131	172.439293787006

12	Setosa	127	167.474240327017
12	Setosa	128	172.168092088007
13	Setosa	127	156.642887989995
13	Setosa	125	167.309544147014
14	Setosa	157	236.327545306980
14	Setosa	157	206.714862349977
15	Setosa	147	193.618443787998
15	Setosa	156	206.962101863977
16	Setosa	148	199.192704684999
16	Setosa	134	176.366452653992
17	Setosa	138	187.745835991972
17	Setosa	137	169.435069492010
18	Setosa	137	207.809176247001
18	Setosa	137	182.919502305998
19	Setosa	148	196.248307767975
19	Setosa	138	195.056843461002
20	Setosa	128	192.019549893986
20	Setosa	136	181.114158896005
21	Setosa	131	173.674938551992
21	Setosa	131	162.603793213009
22	Setosa	134	176.422762482003
22	Setosa	142	189.425069820999
23	Setosa	130	173.441276794006
23	Setosa	128	161.455877859999
24	Setosa	129	169.190615194993
24	Setosa	130	163.326121305002
25	Setosa	126	194.988257080021
25	Setosa	129	171.457889856992
26	Setosa	139	184.361206642992
26	Setosa	139	171.731629433987
27	Setosa	138	183.046606850981
27	Setosa	131	161.794431141010
28	Setosa	132	174.659766188995
28	Setosa	137	169.782602152015
29	Setosa	127	168.631016504992
29	Setosa	128	170.168898511999
30	Setosa	131	164.942283763990
30	Setosa	128	170.208546102006
31	Setosa	125	155.474518657991
31	Setosa	128	168.721068390998
32	Setosa	149	222.937748826005
32	Setosa	147	195.558935372999
33	Setosa	158	238.481344380999
33	Setosa	156	207.879671122012
34	Setosa	125	156.748621031997
34	Setosa	128	186.639808643999
35	Setosa	132	164.346666362987

35	Setosa	126	165.486233733990
36	Setosa	137	203.944795306993
36	Setosa	138	183.880312505998
37	Setosa	134	168.868358878003
37	Setosa	135	178.996733682001
38	Setosa	130	195.429970108005
38	Setosa	129	170.336709254004
39	Setosa	130	162.726757990001
39	Setosa	131	177.335391458003
40	Setosa	131	163.113501154992
40	Setosa	127	175.387353620999
41	Setosa	128	170.685004726001
41	Setosa	127	159.067595728996
42	Setosa	130	198.715129416999
42	Setosa	133	163.730003372003
43	Setosa	137	181.117886118003
43	Setosa	148	184.678695999991
44	Setosa	132	200.106016480998
44	Setosa	132	166.333205984001
45	Setosa	127	187.531801700995
45	Setosa	127	167.675175879989
46	Setosa	137	179.643135038999
46	Setosa	135	181.235747734999
47	Setosa	128	169.131526518001
47	Setosa	125	175.958602272000
48	Setosa	138	181.488653265996
48	Setosa	136	183.829710288004
49	Setosa	130	171.584428429994
49	Setosa	134	165.109525389008
50	Versicolour	157	211.561310817005
50	Versicolour	164	202.918191439002
51	Versicolour	163	217.118744624025
51	Versicolour	157	197.350142585996
52	Versicolour	161	212.634487595000
52	Versicolour	159	210.061063218005
53	Versicolour	154	229.690615570000
53	Versicolour	154	206.431666609982
54	Versicolour	158	208.176307326000
54	Versicolour	153	188.276445979005
55	Versicolour	161	214.919902064997
55	Versicolour	155	210.836861494997
56	Versicolour	160	239.947061600996
56	Versicolour	164	217.684541893011
57	Versicolour	153	200.902082576983
57	Versicolour	158	198.894278485993
58	Versicolour	149	194.955016523001
58	Versicolour	162	216.050480177998

59	Versicolour	153	205.240347740997
59	Versicolour	160	225.800205930999
60	Versicolour	155	208.838249627002
60	Versicolour	157	214.510020671999
61	Versicolour	153	231.675736425997
61	Versicolour	159	211.719174236994
62	Versicolour	149	184.161855208996
62	Versicolour	145	191.579371985002
63	Versicolour	154	229.960293514990
63	Versicolour	155	204.139684622995
64	Versicolour	149	224.242873209002
64	Versicolour	149	196.732464599998
65	Versicolour	163	204.490924617997
65	Versicolour	150	197.671623186997
66	Versicolour	152	189.781948169002
66	Versicolour	166	218.234902090003
67	Versicolour	155	208.900213204990
67	Versicolour	148	182.404933997000
68	Virginica	170	226.245127024001
68	Virginica	170	230.596697563003
69	Versicolour	150	197.955241139996
69	Versicolour	149	185.935156120996
70	Virginica	175	257.052065494990
70	Virginica	187	247.820245556005
71	Versicolour	149	221.992986589999
71	Versicolour	156	207.291886979011
72	Versicolour	162	202.250470935997
72	Virginica	172	229.572185268999
73	Versicolour	149	185.542167788999
73	Versicolour	147	192.749071098002
74	Versicolour	151	185.405136344992
74	Versicolour	160	216.082667008011
75	Versicolour	163	219.422193439006
75	Versicolour	155	198.772947766996
76	Versicolour	154	192.966324531002
76	Versicolour	165	217.102438605998
77	Versicolour	184	239.885075207003
77	Versicolour	187	260.694601773998
78	Versicolour	155	227.302545148988
78	Versicolour	157	207.843915240997
79	Versicolour	149	230.939681180001
79	Versicolour	147	193.725389892001
80	Versicolour	150	197.777190996010
80	Versicolour	144	190.623751826013
81	Versicolour	157	193.555802143004
81	Versicolour	158	211.250372465999
82	Versicolour	149	196.583431284995

82	Versicolour	148	185.894965120980
83	Virginica	175	232.895015100995
83	Virginica	167	208.919430489015
84	Versicolour	165	220.446305512998
84	Versicolour	149	192.682118263996
85	Versicolour	157	230.643113016980
85	Versicolour	153	197.791706705997
86	Versicolour	158	195.910444591019
86	Versicolour	167	223.536711459977
87	Versicolour	153	228.968388303002
87	Versicolour	150	188.992137657001
88	Versicolour	149	222.265606482007
88	Versicolour	149	201.321347834999
89	Versicolour	152	225.510500825987
89	Versicolour	152	202.981379129012
90	Versicolour	153	200.503413899004
90	Versicolour	149	184.189613683003
91	Versicolour	153	233.390733836000
91	Versicolour	152	190.076397503005
92	Versicolour	147	185.856174415994
92	Versicolour	143	191.554912406001
93	Versicolour	156	193.719176963979
93	Versicolour	144	193.033757219032
94	Versicolour	150	222.192551716034
94	Versicolour	151	200.592916837001
95	Versicolour	149	196.175308451005
95	Versicolour	149	188.845884275997
96	Versicolour	150	229.297362476992
96	Versicolour	150	199.220232527002
97	Versicolour	151	201.302536189999
97	Versicolour	149	185.354248030003
98	Versicolour	152	227.317951579994
98	Versicolour	152	201.643273931002
99	Versicolour	151	225.431768844997
99	Versicolour	150	198.822044724998
100	Virginica	224	338.457750877991
100	Virginica	218	269.004195140027
101	Virginica	182	239.634273421000
101	Virginica	189	233.282648902000
102	Virginica	220	292.813322126021
102	Virginica	209	261.148349206974
103	Virginica	187	247.867449441000
103	Virginica	181	227.874074076003
104	Virginica	207	317.352916752001
104	Virginica	202	250.390835297015
105	Virginica	224	277.126851870001
105	Virginica	224	300.419712322996

106	Virginica	163	207.751436323998
106	Virginica	172	227.083190554008
107	Virginica	204	267.688085361001
107	Virginica	201	251.167861824998
108	Virginica	191	296.912145643991
108	Virginica	186	230.759978177995
109	Virginica	236	310.326191976997
109	Virginica	242	319.673102364998
110	Virginica	188	280.733526721982
110	Virginica	186	232.083809244003
111	Virginica	185	243.380497448997
111	Virginica	191	236.999902288989
112	Virginica	198	260.882904514991
112	Virginica	196	241.973885368997
113	Virginica	185	242.818905136995
113	Virginica	177	223.075697712000
114	Virginica	199	267.123339661999
114	Virginica	193	244.631268027997
115	Virginica	204	306.134122104002
115	Virginica	198	247.788715191997
116	Virginica	184	272.102472769020
116	Virginica	186	246.037208625008
117	Virginica	242	372.426236752011
117	Virginica	235	314.646451627993
118	Virginica	245	324.170386424007
118	Virginica	243	302.048878948015
119	Virginica	182	242.247017303019
119	Virginica	176	233.979537162002
120	Virginica	212	278.346020477016
120	Virginica	212	268.808973235003
121	Virginica	181	239.948561597001
121	Virginica	179	234.868162627004
122	Virginica	220	273.489328298006
122	Virginica	225	295.278771966002
123	Virginica	190	253.581019274017
123	Virginica	171	218.115417962992
124	Virginica	203	270.345697628997
124	Virginica	203	268.380529163008
125	Virginica	199	262.758556278997
125	Virginica	198	248.990126456996
126	Virginica	175	275.510221432005
126	Virginica	175	230.965268736990
127	Virginica	176	267.179765855002
127	Virginica	175	229.524818143001
128	Virginica	198	301.225683244982
128	Virginica	196	257.619645455003
129	Versicolour	184	231.711877422989

129	Versicolour	186	230.637025561998
130	Virginica	201	303.429061058011
130	Virginica	203	267.511914004001
131	Virginica	229	281.882377957012
131	Virginica	229	343.981873643000
132	Virginica	205	270.008131306998
132	Virginica	215	285.015908257948
133	Versicolour	174	229.449650061003
133	Versicolour	172	230.078568673006
134	Virginica	169	214.229202766998
134	Virginica	179	236.193840727996
135	Virginica	225	283.014836553988
135	Virginica	227	303.608020009000
136	Virginica	210	278.282597700006
136	Virginica	207	259.714055369999
137	Virginica	189	250.107274837003
137	Virginica	184	243.175012926998
138	Virginica	175	260.285272776998
138	Virginica	174	227.462182527002
139	Virginica	198	259.245260090001
139	Virginica	199	274.972512879005
140	Virginica	215	316.247902212977
140	Virginica	213	279.560490926998
141	Virginica	200	264.257141721987
141	Virginica	200	253.381090612993
142	Virginica	174	220.305673462992
142	Virginica	180	235.826600545000
143	Virginica	215	286.330613108002
143	Virginica	217	296.432417679001
144	Virginica	224	330.540486273004
144	Virginica	221	290.051995093996
145	Virginica	199	244.107485886985
145	Virginica	205	258.156277771000
146	Virginica	182	238.877723096001
146	Virginica	176	221.128657142995
147	Virginica	192	253.839022833007
147	Virginica	191	261.254757341996
148	Virginica	219	292.240693182975
148	Virginica	199	246.243809733003
149	Virginica	172	214.630656760009
149	Virginica	177	232.054860812993

B.7 Copula ADVI on the MNIST Dataset, in Python

As per the convention of previous sections and appendix sections (2.3, 3.1, B.4), my originally designed algorithm is referred to as "Copula **ADVI**". The following script is a Python script that executes the Copula **ADVI** algorithm on the **MNIST** dataset. Please refer to the comments in the script, chapter 3.4, and the previous appendix sections in this appendix, for guidance as to how the following script has been structured.

The output of this particular script is 10,000 printed lines, each resembling the text "A: 5 P: 4 – 167 348.607332505993", corresponding to each of the instances in the **MNIST** testing dataset. The line of text "A: 5 P: 4 – 167 348.607332505993", for example, would mean that, for the **MNIST** testing instance being considered:

i The actual target value of the **MNIST** testing instance is 5.

ii After this instance had been run through the Copula ADVI algorithm, the algorithm predicted that this **MNIST** testing instance takes a target value of 4.

iii A total of 167 gradient ascent iterations were required for the algorithm to be executed for this given **MNIST** testing instance.

iv The relative time taken for these 167 gradient ascent iterations to execute was 348.607332505993.

```python
import pandas as pd
import time

# The autograd package is the underpinning automatic differentiation
# package in this script
import autograd.numpy as np
from autograd import jacobian

# The PCA method within the sklearn library is used to carry out the
# required PCA pre-processing functions outlined in chapter 5
from sklearn.decomposition import PCA

#############################################

# Hyperparameters
epsilon = 0.01              # Convergence Measure
iter_limit = 100            # Iteration Limit
rho_first = 0.01            # Initial Step Size
rho_last = 0.001            # Final Step Size
S = 100                     # MC Integration Sample Size
PCA_comp = 300              # Number of PCA components
num_of_targets = 10         # Number of target labels

z_pos = 5                   # Positive prior/class label
z_neg = -5                  # Negative prior/class label
prior_stds = 1              # Prior Gaussian Component std

#############################################

# Importing the MNIST training dataset
MNIST = pd.read_csv('mnist_train.csv', header=None, index_col=0,
                        dtype=np.float64)

MNIST.index = MNIST.index.astype(np.int64)
```

```
raw_data_mean = MNIST.mean()     # Means of each of the features
raw_data_std = MNIST.std()       # Stds of each of the features

# If raw_data_std = 0 for a given column/feature, the values in that
# column/feature will become NaN values (can not divide by zero), thus
# resulting in the removal of those columns/features by the dropna
# attribute
MNIST = (MNIST - raw_data_mean) / raw_data_std
MNIST.dropna(axis=1, inplace=True)

# Applying PCA on the MNIST training dataset, whilst retaining the
# original target variable values
X_r = np.array(MNIST)
pca = PCA(n_components=PCA_comp)
X_r = pca.fit(X_r).transform(X_r)
MNIST = pd.DataFrame(X_r, index=MNIST.index)

# Calculating the required beta vector, and the required means and
# variances of the selected principle components across each of the
# values that the retained target variable can take
beta = np.array([0.0]*num_of_targets)
mean = np.array([[0.0]*PCA_comp]*num_of_targets)
var = np.array([[0.0]*PCA_comp]*num_of_targets)

for i in range(num_of_targets):
    beta[i] = MNIST.index.value_counts().loc[i] / MNIST.shape[0]
    mean[i] = MNIST.loc[i].mean().to_numpy()
    var[i] = MNIST.loc[i].var().to_numpy()

#############################################

# Importing the MNIST testing dataset, overwriting the variable
# containing the pre-processed MNIST training dataset, as we have now
# extracted all the information required from that dataset
MNIST = pd.read_csv('mnist_test.csv', header=None, index_col=0,
                        dtype=np.float64)

# Ensuring the testing dataset dataframe has the correct number of
# rows and columns
MNIST.index = MNIST.index.astype(np.int64)
MNIST = MNIST.iloc[:, [i for i in range(PCA_comp)]]

# Importing, standardising (in the context of the training dataset)
# and truncating a copy of the MNIST testing dataset, for pre
# processing purposes
MNIST_2 = pd.read_csv('mnist_test.csv', header=None, index_col=0,
                        dtype=np.float64)
MNIST_2.index = MNIST_2.index.astype(np.int64)
MNIST_2 = (MNIST_2 - raw_data_mean) / raw_data_std
MNIST_2.dropna(axis=1, inplace=True)

# Using the PCA loadings that were calculated when PCA was applied on
# the truncated and standardised training dataset, to calculate
# appropriate pseudo principle components for the testing dataset
for i in range(MNIST.shape[0]):
    for j in range(PCA_comp):
        MNIST.iloc[i, j] = sum(MNIST_2.iloc[i].to_numpy()
                                   * pca.components_[j])

#############################################

# z is {num_of_targets}D, x is {PCA_comp}D
class_list = (np.identity(num_of_targets) * (z_pos - z_neg)) + z_neg
```

```
eta_set = np.random.normal(0, 1, S*num_of_targets).reshape(S,
                                num_of_targets)


def prior(z_a):
    Gaus = np.array([0.0]*num_of_targets)
    for i in range(num_of_targets):
        for j in range(num_of_targets):
            Gaus[i] = Gaus[i] + (-0.5)*(((z_a[j]
                        - class_list[i][j])/prior_stds)**2)
        Gaus[i] = (1/(((2*np.pi)**(num_of_targets/2))
                * (prior_stds**num_of_targets))) * np.exp(Gaus[i])
    return sum(beta*Gaus)


def grad_log_prior(z_b):
    Gaus = np.array([0.0]*num_of_targets)
    z_minus_mu = np.array([[0.0]*num_of_targets]*num_of_targets)
    for i in range(num_of_targets):
        for j in range(num_of_targets):
            Gaus[i] = Gaus[i] + (-0.5)*(((z_b[j]
                        - class_list[i][j])/prior_stds)**2)
            z_minus_mu[i][j] = z_b[i] - class_list[i][j]
        Gaus[i] = (1/(((2*np.pi)**(num_of_targets/2))
                    * (prior_stds**num_of_targets))) * np.exp(Gaus[i])
    return ((-1 / (sum(beta*Gaus) * (prior_stds**2)))
                        * np.matmul(z_minus_mu, beta*Gaus))


def log_likelihood(z_c):
    mu = np.array([0.0]*PCA_comp)
    sigma_2 = np.array([0.0]*PCA_comp)
    for i in range(num_of_targets):
        mu = mu + (((z_c[i] - z_neg) / (z_pos - z_neg)) * mean[i])
        sigma_2 = sigma_2 + (((z_c[i] - z_neg) / (z_pos - z_neg))
                        * np.log(var[i]))
    sigma_2 = np.exp(sigma_2)
    log_likelihoods = (-0.5) * np.log(2*np.pi*sigma_2)
    log_likelihoods = log_likelihoods - (((x - mu)**2) / (2*sigma_2))
    return sum(log_likelihoods)


def grad_log_likelihood(z_d):
    mu = np.array([0.0]*PCA_comp)
    sigma_2 = np.array([0.0]*PCA_comp)
    grad_log_likelihood = np.array([0.0]*num_of_targets)
    for i in range(num_of_targets):
        mu = mu + (((z_d[i] - z_neg) / (z_pos - z_neg)) * mean[i])
        sigma_2 = sigma_2 + (((z_d[i] - z_neg) / (z_pos - z_neg))
                    * np.log(var[i]))
    sigma_2 = np.exp(sigma_2)
    for i in range(PCA_comp):
        grad_log_likelihood = grad_log_likelihood + ((
                - (sigma_2[i] * np.log(var[:, i]))
                + (((x[i])**2) * np.log(var[:, i]))
                - (2 * x[i] * mu[i] * np.log(var[:, i]))
                + (2 * x[i] * mean[:, i])
                + (((mu[i])**2) * np.log(var[:, i]))
                - (2 * mu[i] * mean[:, i])
            ) / (2 * sigma_2[i] * (z_pos-z_neg)))
    return grad_log_likelihood
```

```
# I was unable to more succinctly define this function, whereby the
# autograd package would still execute as intended
def Sigma(omega_e, alpha_e):
    a_00 = np.exp(2*omega_e[0])
    a_11 = np.exp(2*omega_e[1])
    a_22 = np.exp(2*omega_e[2])
    a_33 = np.exp(2*omega_e[3])
    a_44 = np.exp(2*omega_e[4])
    a_55 = np.exp(2*omega_e[5])
    a_66 = np.exp(2*omega_e[6])
    a_77 = np.exp(2*omega_e[7])
    a_88 = np.exp(2*omega_e[8])
    a_99 = np.exp(2*omega_e[9])
    a_01 = ((2/np.pi) * np.exp(omega_e[0]+omega_e[1])
                                            * np.arctan(alpha_e[0]))
    a_02 = ((2/np.pi) * np.exp(omega_e[0]+omega_e[2])
                                            * np.arctan(alpha_e[1]))
    a_03 = ((2/np.pi) * np.exp(omega_e[0]+omega_e[3])
                                            * np.arctan(alpha_e[2]))
    a_04 = ((2/np.pi) * np.exp(omega_e[0]+omega_e[4])
                                            * np.arctan(alpha_e[3]))
    a_05 = ((2/np.pi) * np.exp(omega_e[0]+omega_e[5])
                                            * np.arctan(alpha_e[4]))
    a_06 = ((2/np.pi) * np.exp(omega_e[0]+omega_e[6])
                                            * np.arctan(alpha_e[5]))
    a_07 = ((2/np.pi) * np.exp(omega_e[0]+omega_e[7])
                                            * np.arctan(alpha_e[6]))
    a_08 = ((2/np.pi) * np.exp(omega_e[0]+omega_e[8])
                                            * np.arctan(alpha_e[7]))
    a_09 = ((2/np.pi) * np.exp(omega_e[0]+omega_e[9])
                                            * np.arctan(alpha_e[8]))
    a_12 = ((2/np.pi) * np.exp(omega_e[1]+omega_e[2])
                                            * np.arctan(alpha_e[9]))
    a_13 = ((2/np.pi) * np.exp(omega_e[1]+omega_e[3])
                                            * np.arctan(alpha_e[10]))
    a_14 = ((2/np.pi) * np.exp(omega_e[1]+omega_e[4])
                                            * np.arctan(alpha_e[11]))
    a_15 = ((2/np.pi) * np.exp(omega_e[1]+omega_e[5])
                                            * np.arctan(alpha_e[12]))
    a_16 = ((2/np.pi) * np.exp(omega_e[1]+omega_e[6])
                                            * np.arctan(alpha_e[13]))
    a_17 = ((2/np.pi) * np.exp(omega_e[1]+omega_e[7])
                                            * np.arctan(alpha_e[14]))
    a_18 = ((2/np.pi) * np.exp(omega_e[1]+omega_e[8])
                                            * np.arctan(alpha_e[15]))
    a_19 = ((2/np.pi) * np.exp(omega_e[1]+omega_e[9])
                                            * np.arctan(alpha_e[16]))
    a_23 = ((2/np.pi) * np.exp(omega_e[2]+omega_e[3])
                                            * np.arctan(alpha_e[17]))
    a_24 = ((2/np.pi) * np.exp(omega_e[2]+omega_e[4])
                                            * np.arctan(alpha_e[18]))
    a_25 = ((2/np.pi) * np.exp(omega_e[2]+omega_e[5])
                                            * np.arctan(alpha_e[19]))
    a_26 = ((2/np.pi) * np.exp(omega_e[2]+omega_e[6])
                                            * np.arctan(alpha_e[20]))
    a_27 = ((2/np.pi) * np.exp(omega_e[2]+omega_e[7])
                                            * np.arctan(alpha_e[21]))
    a_28 = ((2/np.pi) * np.exp(omega_e[2]+omega_e[8])
                                            * np.arctan(alpha_e[22]))
    a_29 = ((2/np.pi) * np.exp(omega_e[2]+omega_e[9])
                                            * np.arctan(alpha_e[23]))
    a_34 = ((2/np.pi) * np.exp(omega_e[3]+omega_e[4])
                                            * np.arctan(alpha_e[24]))
    a_35 = ((2/np.pi) * np.exp(omega_e[3]+omega_e[5])
                                             * np.arctan(alpha_e[25]))
    a_36 = ((2/np.pi) * np.exp(omega_e[3]+omega_e[6])
                                             * np.arctan(alpha_e[26]))
    a_37 = ((2/np.pi) * np.exp(omega_e[3]+omega_e[7])
                                             * np.arctan(alpha_e[27]))
    a_38 = ((2/np.pi) * np.exp(omega_e[3]+omega_e[8])
                                             * np.arctan(alpha_e[28]))
    a_39 = ((2/np.pi) * np.exp(omega_e[3]+omega_e[9])
                                             * np.arctan(alpha_e[29]))
    a_45 = ((2/np.pi) * np.exp(omega_e[4]+omega_e[5])
                                             * np.arctan(alpha_e[30]))
    a_46 = ((2/np.pi) * np.exp(omega_e[4]+omega_e[6])
                                             * np.arctan(alpha_e[31]))
    a_47 = ((2/np.pi) * np.exp(omega_e[4]+omega_e[7])
                                             * np.arctan(alpha_e[32]))
    a_48 = ((2/np.pi) * np.exp(omega_e[4]+omega_e[8])
                                             * np.arctan(alpha_e[33]))
    a_49 = ((2/np.pi) * np.exp(omega_e[4]+omega_e[9])
                                             * np.arctan(alpha_e[34]))
    a_56 = ((2/np.pi) * np.exp(omega_e[5]+omega_e[6])
                                             * np.arctan(alpha_e[35]))
    a_57 = ((2/np.pi) * np.exp(omega_e[5]+omega_e[7])
                                             * np.arctan(alpha_e[36]))
    a_58 = ((2/np.pi) * np.exp(omega_e[5]+omega_e[8])
                                             * np.arctan(alpha_e[37]))
    a_59 = ((2/np.pi) * np.exp(omega_e[5]+omega_e[9])
                                             * np.arctan(alpha_e[38]))
    a_67 = ((2/np.pi) * np.exp(omega_e[6]+omega_e[7])
                                             * np.arctan(alpha_e[39]))
    a_68 = ((2/np.pi) * np.exp(omega_e[6]+omega_e[8])
                                             * np.arctan(alpha_e[40]))
    a_69 = ((2/np.pi) * np.exp(omega_e[6]+omega_e[9])
                                             * np.arctan(alpha_e[41]))
    a_78 = ((2/np.pi) * np.exp(omega_e[7]+omega_e[8])
                                             * np.arctan(alpha_e[42]))
    a_79 = ((2/np.pi) * np.exp(omega_e[7]+omega_e[9])
                                             * np.arctan(alpha_e[43]))
    a_89 = ((2/np.pi) * np.exp(omega_e[8]+omega_e[9])
                                             * np.arctan(alpha_e[44]))
    return np.array(
       [[a_00, a_01, a_02, a_03, a_04, a_05, a_06, a_07, a_08, a_09],
        [a_01, a_11, a_12, a_13, a_14, a_15, a_16, a_17, a_18, a_19],
        [a_02, a_12, a_22, a_23, a_24, a_25, a_26, a_27, a_28, a_29],
        [a_03, a_13, a_23, a_33, a_34, a_35, a_36, a_37, a_38, a_39],
        [a_04, a_14, a_24, a_34, a_44, a_45, a_46, a_47, a_48, a_49],
        [a_05, a_15, a_25, a_35, a_45, a_55, a_56, a_57, a_58, a_59],
        [a_06, a_16, a_26, a_36, a_46, a_56, a_66, a_67, a_68, a_69],
        [a_07, a_17, a_27, a_37, a_47, a_57, a_67, a_77, a_78, a_79],
        [a_08, a_18, a_28, a_38, a_48, a_58, a_68, a_78, a_88, a_89],
        [a_09, a_19, a_29, a_39, a_49, a_59, a_69, a_79, a_89, a_99]])


def Sigma_sqrt(omega_f, alpha_f):
    eigenvalues, eigenvectors = np.linalg.eig(Sigma(omega_f, alpha_f))
    sqrt_eigenval_matrix = np.sqrt(np.diag(eigenvalues))
    eigenvectors_inv_mat = np.linalg.inv(eigenvectors)
    return np.matmul(np.matmul(eigenvectors, sqrt_eigenval_matrix),
                          eigenvectors_inv_mat)


def ELBO():
    total_MC_expectation_val = 0.0
    entropy = 0.5 * num_of_targets * (1 + np.log(2*np.pi))
    entropy = (entropy
               + (0.5 * np.log(np.linalg.det(Sigma(omega, alpha)))))
    for s in range(S):
        eta = eta_set[s]
        z_i = np.matmul(Sigma_sqrt(omega, alpha), eta) + mu
        total_MC_expectation_val = (total_MC_expectation_val
                                    + np.log(prior(z_i)))
        total_MC_expectation_val = (total_MC_expectation_val
                                    + log_likelihood(z_i))
    return (total_MC_expectation_val / S) + entropy


def grad_mu_ELBO():
    total_MC_expectation_val = np.array([0.0]*num_of_targets)
    for s in range(S):
        eta = eta_set[s]
        z_i = np.matmul(Sigma_sqrt(omega, alpha), eta) + mu
        total_MC_expectation_val = (total_MC_expectation_val
                                    + grad_log_prior(z_i))
        total_MC_expectation_val = (total_MC_expectation_val
                                    + grad_log_likelihood(z_i))
    return total_MC_expectation_val / S


def grad_omega_ELBO():
    total_MC_expectation_val = np.array([0.0]*num_of_targets)
    grad_omega_det_Sigma = jacobian(
               lambda omega_g: np.linalg.det(Sigma(omega_g, alpha)))
    # hgold - Half Grad Omega Log Determinant
    hgold_Sigma = (grad_omega_det_Sigma(omega)
                        / (2 * np.linalg.det(Sigma(omega, alpha))))
    for s in range(S):
        eta = eta_set[s]
        z_i = np.matmul(Sigma_sqrt(omega, alpha), eta) + mu
        # gossre - Grad Omega Sigma Square Root Eta
        gossre = jacobian(
            lambda omega_h: np.matmul(Sigma_sqrt(omega_h, alpha), eta))
        total_MC_expectation_val = (total_MC_expectation_val
                  + np.matmul(grad_log_prior(z_i), gossre(omega)))
        total_MC_expectation_val = (total_MC_expectation_val
               + np.matmul(grad_log_likelihood(z_i), gossre(omega)))
    return (total_MC_expectation_val / S) + hgold_Sigma


def grad_alpha_ELBO():
    total_MC_expectation_val = np.array([0.0]
               * int(((num_of_targets**2) - num_of_targets) / 2))
    grad_alpha_det_Sigma = jacobian(
               lambda alpha_i: np.linalg.det(Sigma(omega, alpha_i)))
    # hgald - Half Grad Alpha Log Determinant
    hgald_Sigma = (grad_alpha_det_Sigma(alpha)
               / (2 * np.linalg.det(Sigma(omega, alpha))))
    for s in range(S):
        eta = eta_set[s]
        z_i = np.matmul(Sigma_sqrt(omega, alpha), eta) + mu
        # gassre - Grad Alpha Sigma Square Root Eta
        gassre = jacobian(
            lambda alpha_j: np.matmul(Sigma_sqrt(omega, alpha_j), eta))
        total_MC_expectation_val = (total_MC_expectation_val
               + np.matmul(grad_log_prior(z_i), gassre(alpha)))
        total_MC_expectation_val = (total_MC_expectation_val
               + np.matmul(grad_log_likelihood(z_i), gassre(alpha)))
    return (total_MC_expectation_val / S) + hgald_Sigma


#############################################

for test_index in range(MNIST.shape[0]):
    x = MNIST.iloc[test_index].to_numpy()
    actual_class = MNIST.iloc[test_index].name
    ELBO_list = []
    total_iteration_counter = 0
    total_key_time = 0.0

    for current_testing_class in class_list:
        mu = current_testing_class
        omega = np.array([0.0]*num_of_targets)
        alpha = np.array([0.0]
                    * int(((num_of_targets**2) - num_of_targets) / 2))

        current_ELBO = ELBO()
        last_ELBO = epsilon + 1
        iteration_counter = 0

        while (abs(current_ELBO - last_ELBO) >= epsilon
                                and iteration_counter < iter_limit):
            iteration_counter += 1
            total_iteration_counter += 1
            last_ELBO = current_ELBO
            rho = rho_first * ((rho_last/rho_first)**
                             ((iteration_counter-1)/(iter_limit-1)))
            t1 = time.perf_counter()
            mu = mu + (rho * grad_mu_ELBO())
            omega = omega + (rho * grad_omega_ELBO())
            alpha = alpha + (rho * grad_alpha_ELBO())
            current_ELBO = ELBO()
            t2 = time.perf_counter()

            total_key_time += (t2 - t1)

        ELBO_list.append(current_ELBO)

    for i in range(num_of_targets):
        if ELBO_list[i] == max(ELBO_list):
            print('A:', actual_class, 'P:', i, '--',
                             total_iteration_counter, total_key_time)
```

Bibliography

Amari, Shun-Ichi (1998). “Natural gradient works efficiently in learning”. In: *Neural computation* 10.2, pp. 251–276.

Anderson, Edgar (1935). “The irises of the Gaspe Peninsula”. In: *Bull. Am. Iris Soc.* 59, pp. 2–5.

Anderson, James R and Carsten Peterson (1987). “A mean field theory learning algorithm for neural networks”. In: *Complex Systems* 1, pp. 995–1019.

Barber, David and Wim Wiegerinck (1998). “Tractable variational structures for approximating graphical models”. In: *Advances in Neural Information Processing Systems* 11.

Baydin, Atilim Gunes et al. (2018). “Automatic differentiation in machine learning: a survey”. In: *Journal of Marchine Learning Research* 18, pp. 1–43.

Bishop, Christopher et al. (1997). “Approximating posterior distributions in belief networks using mixtures”. In: *Advances in neural information processing systems* 10.

Bishop, Christopher M and Nasser M Nasrabadi (2006). *Pattern recognition and machine learning*. Vol. 4. 4. Springer.

Blei, David M, Alp Kucukelbir, and Jon D McAuliffe (2017). “Variational inference: A review for statisticians”. In: *Journal of the American statistical Association* 112.518, pp. 859–877.

Bottou, Léon and Yann Cun (2003). “Large scale online learning”. In: *Advances in neural information processing systems* 16.

Bridle, John (1989). “Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters”. In: *Advances in neural information processing systems* 2.

Cholesky, AL (1924). “Note sur une méthode de résolution des équations normales provenant de l’application de la méthode des moindres carrésa un syteme d’équations linéaires en nombre inférieura celui des inconnues (Published six years after Cholesky’s death by Benoit)”. In: *Bull. Géodésique* 2, pp. 67–77.

Dissmann, Jeffrey et al. (2013). “Selecting and estimating regular vine copulae and application to financial returns”. In: *Computational Statistics & Data Analysis* 59, pp. 52–69.

Gelfand, Alan E and Adrian FM Smith (1990). “Sampling-based approaches to calculating marginal densities”. In: *Journal of the American statistical association* 85.410, pp. 398–409.

Geman, Stuart and Donald Geman (1984). “Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images”. In: *IEEE Transactions on pattern analysis and machine intelligence* 6, pp. 721–741.

Genest, Christian et al. (2009). “Modeling and measurement of multivariate risk in insurance and finance”. In: *Insurance: Mathematics & Economics* 44.2, pp. 143–145.

Gruber, Lutz and Claudia Czado (2015). “Sequential bayesian model selection of regular vine copulas”. In.

Hastings, W Keith (1970). “Monte Carlo sampling methods using Markov chains and their applications”. In.

Hennig, Philipp (2011). “Approximate inference in graphical models”. PhD thesis. University of Cambridge.

Hoffman, Matthew D, Andrew Gelman, et al. (2014). “The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo.” In: *J. Mach. Learn. Res.* 15.1, pp. 1593–1623.

Hoffman, Matthew D et al. (2013). “Stochastic variational inference”. In: *Journal of Machine Learning Research*.

Joe, Harry (1996). “Families of m-variate distributions with given margins and m (m-1)/2 bivariate dependence parameters”. In: *Lecture notes-monograph series*, pp. 120–141.

Jordan, Michael I et al. (1999). “An introduction to variational methods for graphical models”. In: *Machine learning* 37.2, pp. 183–233.

Joyce, James M (2011). “Kullback-leibler divergence”. In: *International encyclopedia of statistical science*. Springer, pp. 720–722.

Knowles, David and Tom Minka (2011). “Non-conjugate variational message passing for multinomial and binary regression”. In: *Advances in Neural Information Processing Systems* 24.

Krämer, Nicole and Ulf Schepsmeier (2011). “Introduction to vine copulas”. In.

Kucukelbir, Alp et al. (2017). “Automatic differentiation variational inference”. In: *Journal of machine learning research*.

Kullback, Solomon and Richard A Leibler (1951). “On information and sufficiency”. In: *The annals of mathematical statistics* 22.1, pp. 79–86.

Kurowicka, Dorota and Roger M Cooke (2006). *Uncertainty analysis with high dimensional dependence modelling*. John Wiley & Sons.

Kushner, Harold J. and George Yin (1997). “Stochastic Approximation Algorithms and Applications”. In: *Applied Mathematics*.

Lauritzen, Steffen L and David J Spiegelhalter (1988). “Local computations with probabilities on graphical structures and their application to expert systems”. In: *Journal of the Royal Statistical Society: Series B (Methodological)* 50.2, pp. 157–194.

LeCun, Yann et al. (1998). “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.

Metropolis, Nicholas et al. (1953). “Equation of state calculations by fast computing machines”. In: *The journal of chemical physics* 21.6, pp. 1087–1092.

Minka, Tom et al. (2005). *Divergence measures and message passing*. Tech. rep. Citeseer.

Nelsen, Roger B (2007). *An introduction to copulas*. Springer science & business media.

Olive, David J (2014). *Statistical theory and inference*. Springer.

Opper, Manfred and David Saad (2001). *Advanced mean field methods: Theory and practice*. MIT press.

Pearl, Judea (1988). *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan kaufmann.

Ranganath, Rajesh, Sean Gerrish, and David Blei (2014). “Black box variational inference”. In: *Artificial intelligence and statistics*. PMLR, pp. 814–822.

Robbins, Herbert and Sutton Monro (1951). “A stochastic approximation method”. In: *The annals of mathematical statistics*, pp. 400–407.

Robert, Christian P, George Casella, and George Casella (1999). *Monte Carlo statistical methods*. Vol. 2. Springer.

Roberts, Stephen J et al. (1998). “Bayesian approaches to Gaussian mixture modeling”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20.11, pp. 1133–1142.

Saul, Lawrence and Michael Jordan (1995). “Exploiting tractable substructures in intractable networks”. In: *Advances in neural information processing systems* 8.

Spall, James C (2005). *Introduction to stochastic search and optimization: estimation, simulation, and control*. John Wiley & Sons.

Tran, Dustin, David Blei, and Edo M Airoldi (2015). "Copula variational inference". In: *Advances in neural information processing systems* 28.

Wainwright, Martin J, Michael I Jordan, et al. (2008). "Graphical models, exponential families, and variational inference". In: *Foundations and Trends® in Machine Learning* 1.1–2, pp. 1–305.

Wengert, Robert Edwin (1964). "A simple automatic derivative evaluation program". In: *Communications of the ACM* 7.8, pp. 463–464.

Winn, John, Christopher M Bishop, and Tommi Jaakkola (2005). "Variational message passing." In: *Journal of Machine Learning Research* 6.4.

Yang, Xitong (2017). "Understanding the variational lower bound". In: *variational lower bound, ELBO, hard attention* 13, pp. 1–4.

Zhang, Cheng et al. (2018). "Advances in variational inference". In: *IEEE transactions on pattern analysis and machine intelligence* 41.8, pp. 2008–2026.