Ecole de statistique, biostatistique et sciences actuarielles

Faculté des Sciences

A Bayesian Approach combining Peptide Intensity and Missingness Modelling to analyse Label Free Mass Spectrometry based Proteomics Data

Mémoire présenté en vue de l'obtention du master en statistiques, orientation biostatistique, par :

Philippe Hauchamps

Membres du jury :

Prof. P. Lambert, Promoteur (Université de Liège, Université catholique de Louvain)
Prof. L. Gatto, Promoteur (Université catholique de Louvain)
Prof. B. Govaerts (Université catholique de Louvain)
Prof. L. Clement (Universiteit Gent)

Louvain-La-Neuve
Juin 2021

Acknowledgments

First, I would like to warmly thank my supervisors, Prof. Lambert and Prof. Gatto of UCLouvain, who gave me the opportunity to work on this interesting topic, and, with their availability and kind feedbacks, provided me with precious support to conduct this research.

I would also like to thank Prof. Govaerts of UCLouvain, who made me discover the exciting world of omics data, and for being reader of this thesis, as well as Prof. Clement of UGent, who also accepted to be reader of this thesis.

Finally, I would like to express my thanks of gratitude to my family. Indeed, this thesis is the final step of a four year-long journey towards the master in biostatistics graduation, which actually had to be done in parallel with my existing professional obligations. Therefore, I would like to thank my spouse, Alice, and my sons, Julien and Maxime, for their patience and continued support during this demanding period.

Contents

Chapter 1

Introduction

1.1 Domain of application : proteomics

The work presented in this Ms Thesis report concerns the statistical treatment of mass spectrometry-based proteomics data.

Proteomics is one of the so-called *omics* sciences that developed in the recent decades. These new omics disciplines are aimed at the systematic and high-throughput analyses of biological samples at molecular level, with as underlying goal the better understanding of the functioning of living beings cells.

Figure 1.1 shows the central position of proteomics in the chain of omics disciplines. each of them relating to a specific stage of the central dogma of molecular biology.

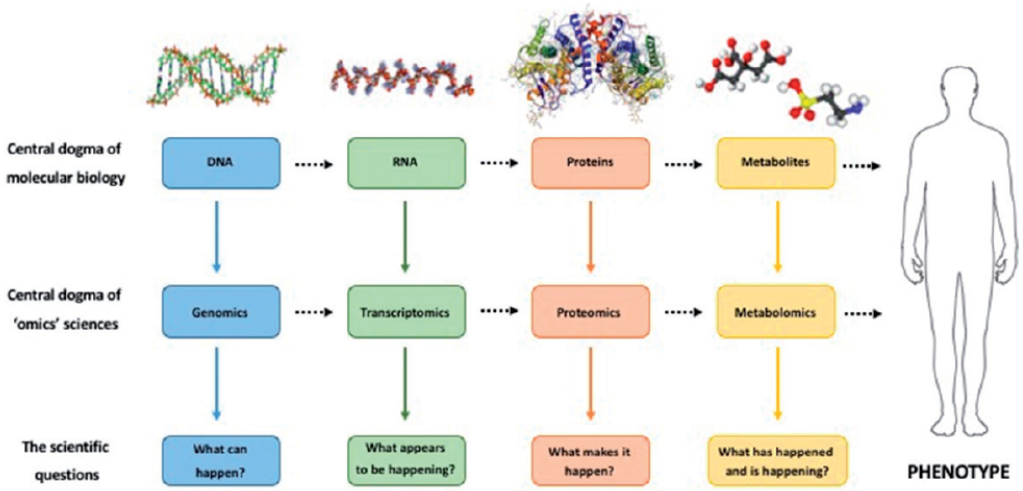


Figure 1.1: The relationship between the central dogma of molecular biology and the omics cascade (in [Araújo et al. 2017])

- *Genomics* studies the *genome*, i.e. the set of genes, coded as DNA molecules and replicated in living beings cells. With a few exceptions, the DNA content is the same in each cell of a given organism, and is mostly stable over time. However this DNA

content does not say anything about what part of it is actually used (i.e. *expressed*) in one particular cell type, at a given time. This is why one can say that genomics focuses on "what can happen?" and not necessarily on "what is actually happening?".

- *Transcriptomics* studies the *transcriptome*, i.e. the set of RNA molecules that are transcripted from their DNA templates and present in some of the cells of a given organism at a given time. An important category of these transcripts encompasses the protein-coding messenger RNA transcripts. However, many RNA transcripts do not code for protein, hence do not fall within the scope of the central dogma of molecular biology. These have different regulatory functions in the process of gene transcription and translation [Frith, Pheasant, and Mattick 2005]. Therefore, RNA analysis can sometimes be found to lack correlation with protein content [Dhingra et al. 2005], this is why one can say that transcriptomics focuses on "what appears to be happening?", in contrast to proteomics, which focuses on "what makes it happen?".

- After genomics and transcriptomics, *proteomics* is the next step in the study of biological systems. Proteins are the functional units in cells, and proteomics studies the *proteome*, which can be defined as the set of proteins found (expressed) in a particular cell or organism at a given time.

- Finally, *metabolomics* studies the smaller molecules (e.g. sugars, lipids, amino acids,...) that are present in cells, tissues and produced by a given organism at a given time (e.g. urine, blood, saliva,...). These substances are by-products of processes and chemical reactions that have happened or are happening in the organism, as the results of the first three steps ("what has happened and is happening?")

According to [Anderson and Leigh Anderson 1998], the goal of proteomics is "a comprehensive, quantitative description of protein expression and its changes under the influence of biological perturbations such as disease or drug treatment". However, besides broad proteins identification and quantification, some other specific goals can also be, for example [Gatto 2019] :

- studying the interactions between proteins forming protein-protein complexes
- tagging the presence of post-transcriptional modification (i.e. phosphorylations)
- assessing the rate at which proteins are produced and degraded
- identifying the cell location(s) where the proteins reside
- ...

The technique of choice to study proteins in a high throughput way is *mass spectrometry* (MS).

1.2 Mass spectrometry - how does it work ?

In order to assess the characteristics of the data that are obtained through mass spectrometry, it is important to understand the basics of this technology. Please note that the content of

this section is mostly adapted from [Gatto 2019].

In bottom-up proteomics using mass spectrometry, a preliminary step is the preparation of the biological samples, during which proteins are digested by a protease (e.g. trypsin). This means that the analytes are in fact peptides, i.e. protein pieces, and not directly the proteins of which the investigator wants to assess the abundance.

When studying complex mixtures, the mass spectrometer is coupled with liquid or gaz chromatography. Chromatography is used as a separation technique. Indeed, before reaching the entry of the mass spectrometer, the analytes need to *elute* from a chromatography column, and the time needed to do so depends on their respective physical properties. Figure 1.2 shows a typical *chromatogram*, which plots the total amount of analytes - measured by the total ion current recorded by the mass spectrometer - as a function of their chromatographic retention time.

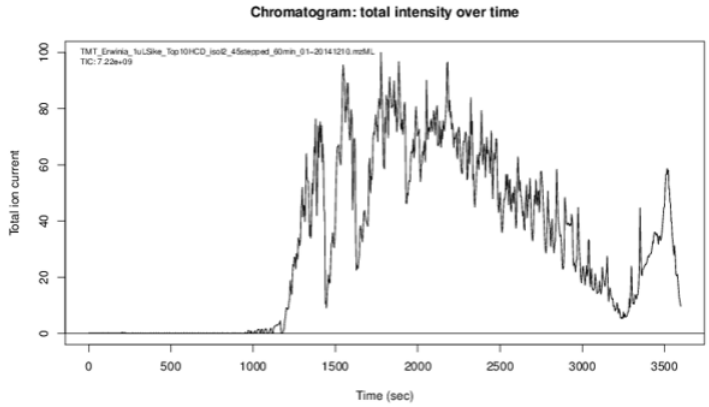


Figure 1.2: A chromatogram, illustrating the total amount of analytes over the retention time

Then, mass spectrometry, which consists in separating ion particules as a function of their m/z (mass-to-charge ratio) is performed. A mass spectrometer can be schematised as composed of three parts (Figure 1.3) :

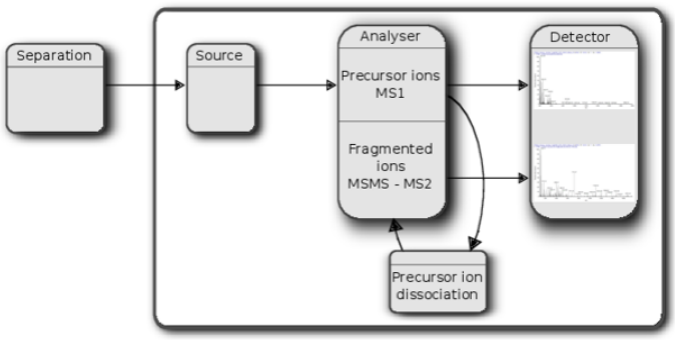


Figure 1.3: A schematic mass spectrometer

- The source, which ionises the molecules: examples of techniques used in this step are Matrix-assisted laser desorption/ionisation (MALDI) or electrospray ionisation (ESI).

- The analyser, which separates the ions according to their m/z ratios. Examples of analyzer types are Time of Flight (TOF) or Orbitrap analyzers.

- The detector, which quantifies the ions.

After a first pass of the ionized molecules in the analyser, one obtains a spectrum, i.e. a set of intensities as a function of increasing m/z. The later spectrum is called a MS1 spectrum - as it is obtained from round 1 in the mass spectrometer - composed of intensity peaks representing the detected molecules, surrounded by background noise. This is illustrated in Figure 1.4, which represents MS data along 3 dimensions: a set of spectra are captured over a series of retention times, resulting from the chromatography step.

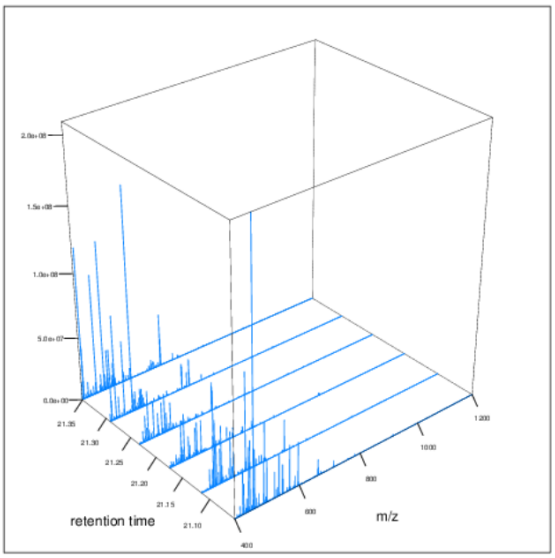


Figure 1.4: MS1 spectra (blue) over retention time (in [Gatto 2019])

However, at the MS1 stage, the only information possessed about the detected peptides are their retention time and their mass-to-charge, which is not enough to infer their identify (i.e. their amino-acid sequence). Therefore, in proteomics, *tandem* mass spectrometry (MSMS, or MS2), is used. In that case, ions are subjected to at least another MS round. First, the mass spectrometer selects a certain number of *precursor* peaks in an MS1 scan (for example 10 or 20) in a specific timespan. This is illustrated in Figure 1.5, where the spectrum on the left is a MS1 spectrum acquired after 21 minutes and 3 seconds of elution (as indicated at the top of the figure). Ten peaks, highlited by dotted vertical lines, have been selected for MS2 analysis.

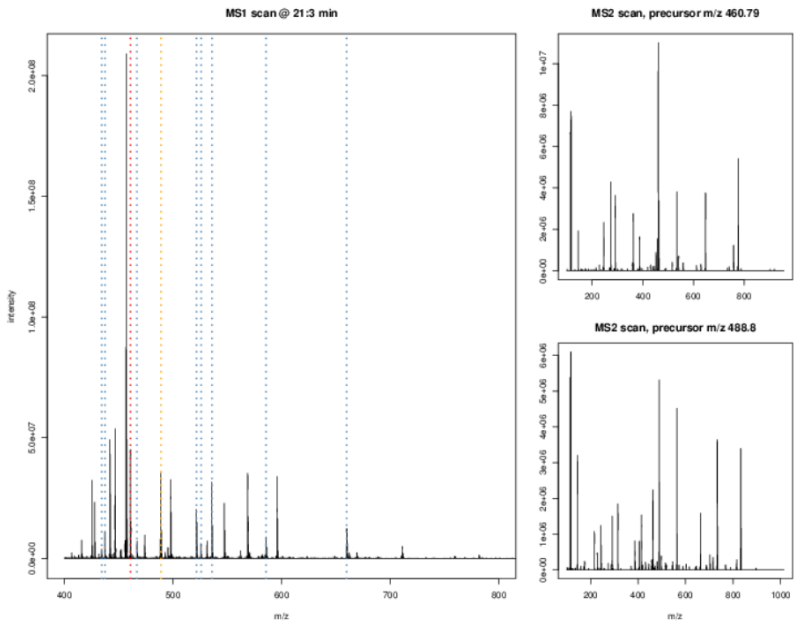


Figure 1.5: Parent ions in the MS1 spectrum (left) and two selected fragment ions MS2 spectra (right) (in [Gatto 2019])

Once a narrow m/z range has been selected (corresponding to one high-intensity peak, supposingly a peptide, and some background noise), it is fragmented into fragment ions. Example of techniques used to create these fragments are collision-induced dissociation (CID), higher energy collisional dissociation (HCD) or electron-transfer dissociation (ETD). These smaller charged molecules are then, in a second round, separated again in the analyser and detected in order to produce a new spectrum, called this time a MS2 spectrum. In the left part of Figure 1.5, among the selected ten peaks, the peak at M/Z 460.79 is highlighted by a red vertical line on the MS1 spectrum, while the peak at M/Z 488.8 is highlighted by an orange vertical line. From these two peaks, the MS2 round generates the two fragment MS2 spectra that are shown in the right part of Figure 1.5, respectively in the top and the bottom plots. The MS2 spectra can then be used to identify the corresponding peptides, in the identification step explained below.

An important remark to make at this stage is that the limited number of MS1 peaks that the spectrometer is able to select, per unit of time, is one of the causes for missing data (see Section 1.3), as only the highest peaks present in the spectra at a specific time are selected for the MS2 round, hence for peptide identification.

Next, in the identification step, the MS2 spectra are used to identify the corresponding peptides. In this step, an MS2 spectrum is search against a database of all possible peptides (chosen according to the anticipated protein population for a given species). Indeed, each of these possible peptides has a well defined predicted fragment spectrum infered from its molecular structure, as illustrated in Figure 1.6. The set of possible predicted spectra can then be tested for matching with the experimental MS2 spectrum to be identified. If there is a sufficiently good match, the MS2 spectrum is assigned the corresponding peptide sequence.

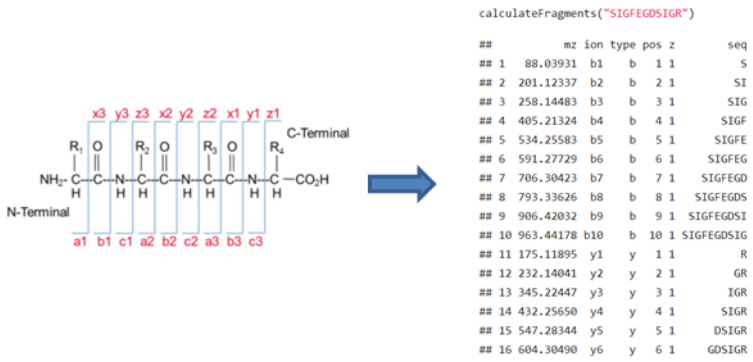


Figure 1.6: The set of possible fragments obtained from a peptide (left) and the corresponding predicted spectrum (right) (in [Gatto 2019])

Note here that the identification is an error prone process, where both false positives and false negatives can occur, leading to identification mismatches. It is possible to assess the statistical significance of a given PSM (*Peptide Spectrum Match*), e.g. by using target vs. decoy search strategy [Elias and Gygi 2010]. In a nutshell, this technique consists in adding to the search database, a set of *decoy* peptides that are known in advance to not exist in the analysed sample. For instance, these can be artificially built by reversing the amino-acid sequences of all peptides included in the target search database. Then, one can use the proportion of matches to decoy peptide sequences to estimate the proportion of false positives in the obtained PSMs, while assessing a reasonable filtering criteria in terms of quality of match.

Finally, the *quantitation* step consists in quantifying the data recorded in the scans. Several techniques are possible depending on the experimental design. The scope of this work will be limited to *Label Free* Quantitation, where the MS1 peaks that match an identified peptide are integrated over its retention time and the area under that extracted ion chromatogram is used to quantify the identified peptide in the current sample (cf. Figure 1.7).

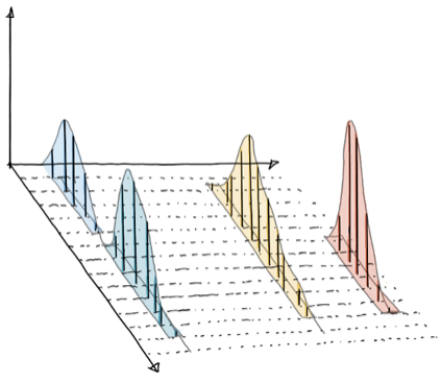


Figure 1.7: Integration of MS1 peak area in Label Free Quantitation (in [Gatto 2019], figure by Johannes Rainer)

Please also note that in Label Free Quantitation, each sample is analysed separately in distinct mass spectrometer runs. This contrasts with *labelled* quantitation (isobaric tagging, SILAC), where several samples are pooled after labelled and analysed together in the same MS run (see Gatto 2019).

As a summary of this section introducing mass spectrometry quantitation for proteomics, Figure 1.8 provides a schematic overview of the different steps of MS data acquisition in the particular case of Label Free Tandem MS based quantitative proteomics.

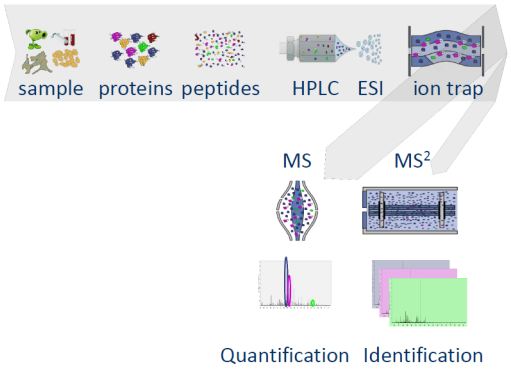


Figure 1.8: Different steps of data acquisition in LF-MS based quantitative proteomics (in [Clement 2019])

.

1.3 Label free MS based quantitative proteomics - data characteristics

This section highlights some typical characteristics of the data obtained in Label Free Mass Spectrometry based Quantitative Proteomics, and the associated challenges for the statistician who needs to make inference from these data. Note that one important source for the content of this section is [Clement 2019].

In a MS based Quantitative Proteomics data analysis, the goal is to perform inference from a high dimensional array of protein intensities data, of which the rows refer to the proteins under study (e.g. 1000+ proteins, but it can be as much as tens of thousands of proteins), and each column represents one data sample. Typically, the number of samples - a few units, up to a few tens is some cases - is much smaller than the number of proteins entering the analysis. Samples can in turn have different characterics that depend on the specific context and scientific question at hand, and the chosen experimental design. Very often, one of the main goals of the inferential analysis is to identify proteins which are *differentially abundant* between two or more sample conditions, with the aim to better understand the impact of these different conditions on the underlying cellular mechanisms at hand. For instance, one could be interested in identifying the proteins (or *biomarkers*) that are differentially expressed between e.g. cells of healthy tissues, and cells affected by a specific disease, or between cells undergoing a drug treatment, and not treated cells. On top of the different conditions to

be investigated, samples can also be characterized by some other variables that are useful to better predict the protein expression profile (additional regressors like cell category or phenotype, or nuisance variables like experimental batch, etc.).

In terms of statistical methods, this work will focus solely on univariate regression methods, whereby one analyses the differential abundance essentially on a protein by protein basis, where the individual protein intensities are considered as the univariate responses in a series of individual single protein regression models. Simultaneous inference can then be obtained thanks to multiple testing corrections (see e.g. [Shaffer 1995]). These univariate methods contrast with multivariate methods (e.g. Principal Components Regression (PCR), Partial Least Squares Regression (PLS),...), where one performs some regression analysis of all protein intensities simultaneously, most often combined with dimensionality reduction, in order to predict one or several condition state(s) considered as response variable(s). Indeed, although multivariate methods are widely used in some omics disciplines like metabolomics, in proteomics, univariate methods are most widely used (see Section 2.1).

One of the main characteristics of proteomics data acquired through mass spectrometry, is their high level of noise. Indeed, the numerous steps that are part of the samples preparation, and the mass spectrometer analysis itself, are subject to a high number of variability sources. For example, peptides, which are chunks of a given protein, can present a variety of chemical modifications (e.g. *post-translational modifications*), which can lead to slight changes in their m/z ratio, in turn inducing widening of their respective spectra. Another important source of peptide intensity variability - for a given protein of a given sample - comes from the different *ionization efficiency* for each peptide. Indeed, depending on the physico-chemical properties of a given peptide, it will be more or less easy to generate ions by the source of the spectrometer, hence leading to differences in measured intensities, even for peptides issued from the same protein sample. These are only two examples of the numerous sources of noise that lead to considerable variability (including peptide intensities outliers) for replicated samples. This, combined with the typically low number of true biological replicates in many lab experiments (number of samples as low as 3 biological replicates per conditions are not rare), leads to considerable challenges for the statistical analysis. In order to improve the quality of the data in the statistical analysis, some techniques of *data normalization*, as a pre-processing step, are usually performed (see also Section 1.4).

As mentioned in Section 1.2, in fact, data acquired by mass spectrometry are not directly measures of protein abundances, but rather peptide intensities. Since the scientific questions studied in proteomics relate to the presence or abundance of proteins, and not peptides, one is then left with two possibilities :

- running a preliminary *summarization* step, in order to somehow convert a set of peptide intensities relating to one protein into one single protein intensity figure per sample. This is done in the majority of available statistical methods for proteomics data. There are a number of ways to perform summarization (see e.g. [Sticker et al. 2020]) of which the description is beyond the scope of this work. However it is important to mention that this step inevitably translates into some loss of available information, and can lead to biaises if not performed adequately.
- introducing peptide intensities directly in the statistical modelling, which has the ad-

vantage of keeping all peptide data information, but at the expense of more complex modelling, and bigger problem size. For example, the huge inter-peptide variability needs to be then taken into account.

Another very important and challenging characteristic of MS based quantitative proteomics data is the high proportion of missing data. Indeed, depending on the dataset at hand, it is not rare to encounter percentages of missing data reaching 50% and even more for some datasets (see e.g. [Lazar et al. 2016]). In general, one can distinguish two[1] main categories of missingness, depending on the characteristics of the missingness mechanism, and its relation with the underlying data values (see e.g. [Molenberghs et al. 2014], Section 1.3):

- Missing completely at random (*MCAR*), when the probability that data are missing is unrelated to the specific values that, in principle, should have been obtained, or the set of observed data.
- Not missing at random (*NMAR*), when the probability that data are missing is related to the specific values that should have been obtained, in addition to the ones actually obtained.

MCAR is less tricky than NMAR, because in the case of MCAR, straightforward handling of missingness by limiting the statistical analysis to complete cases can be done without creating bias. However, in the context of MS based quantitative proteomics data, different sources of missingness have different missingness mechanism characteristics. Herebelow are some examples of missingness sources in tandem MS based quantitative proteomics (see Section 1.2), flagged with their expected type of missingness mechanisms :

- ionization efficiency (already mentioned above) is dependent on physico-chemical properties of peptide chains. Low efficiency leads to low measured intensities and possibly failure to detect the corresponding peptide $\Rightarrow$ NMAR
- in case of MSMS identification, only a subset of present MS1 peaks can be selected for identification in a specific time interval, leading to competition between peptides having similar retention times in the chromatography column $\Rightarrow$ NMAR
- the peptide identification step - see Section 1.2 and Figure 1.6 is a error-prone statistical matching process having false positives and false negatives. Here identifying a link between a higher identification error rate and a low intensity is not so obvious, but cannot be completely ruled out $\Rightarrow$ MCAR or NMAR
- in some samples, it might happen that some peptides are effectively not present at all (true biological absence). From a statistical point of view, this is not really missingness, but rather boils down to obtaining some zero intensity data. Unfortunately, it is impossible to distinguish true biological absence of a peptide in a sample, from missingness induced by the mass spectrometer data acquisition.

Combining together these potential sources of missingness, one could say that in general, all MS based quantitative proteomics datasets should have some NMAR component(s), in that low intensity data tend to be associated with higher proportion of missingness. Therefore, it

[1] actually there is a third category of missingness, more rare and specific : Missing at random (*MAR*) when the probability of missingness may be related to the values of the observed data only.

seems reasonable to include some sort of modelling of the missingness mechanism in the data analysis. In order to do this, again two possibilities exist :

- running a preliminary *imputation* step, whereby some cleverly selected values are assigned to missing data, prior to a (downstream) statistical analysis. This is the most widely used solution, because it does not render the downstream statistical analysis more complex. There exists a wide range of imputation methods (see e.g. [Lazar et al. 2016]) and describing them is beyond the scope of this work. However it is important to realize that careful imputation method selection is far from being a trivial problem, and that imputation can have some important drawbacks. Among these is the fact that the downstream statistical analysis will typically ignore the uncertainty attached to the imputed data values, and hence underestimate the variability attached to model parameters during the inference process. This, in turn, can lead to more false positives.

- including some modelling of the missingness mechanism in the (mainstream) statistical analysis. This is a more elegant approach but leads to more complex statistical models.

1.4 Data preprocessing and bioinformatics pipelines

Before concluding this introductory chapter, let us briefly mention some pre-processing steps that are usually applied to proteomics data, i.e. performed before the (main) statistical analysis step :

1. *Filtering* : from the set of protein intensities obtained from the peptide identification and protein matching steps, some of them are filtered out. These concerns e.g. proteins that are only identified by modification sites, or identified by only one or a few peptides, or proteins that are most likely coming from contaminants of the preparation, ...

2. *Log-transformation* : empirically, one can show that, upon transformation in log, one obtains log-intensity responses that show more uniform sampling variability, in effect transforming multiplicative error structures into additive error structures (cf. Figure 1.9)

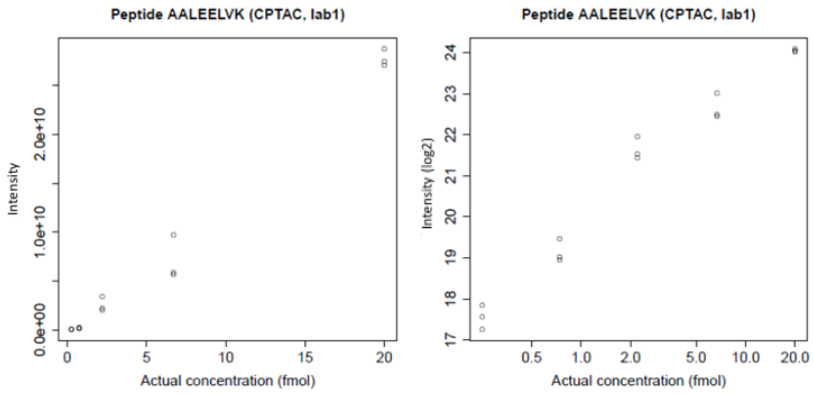


Figure 1.9: Log transformation improves the variability structure (in [Clement 2019])

3. *Normalization* : normalization was already mentioned in Section 1.3, as one of the pre-processing steps to improve the quality of the data. Indeed, even in very clean datasets where only a tiny fraction of proteins have differential abundances, the marginal peptide intensity distribution accross samples can be quite distinct, with considerable sample effects, and therefore, normalization is needed. Different normalization techniques exist (see e.g. [Bolstad et al. 2003]) but describing them is beyond the scope of this work. As an example, Figure 1.10 shows the effect of *quantile normalization*

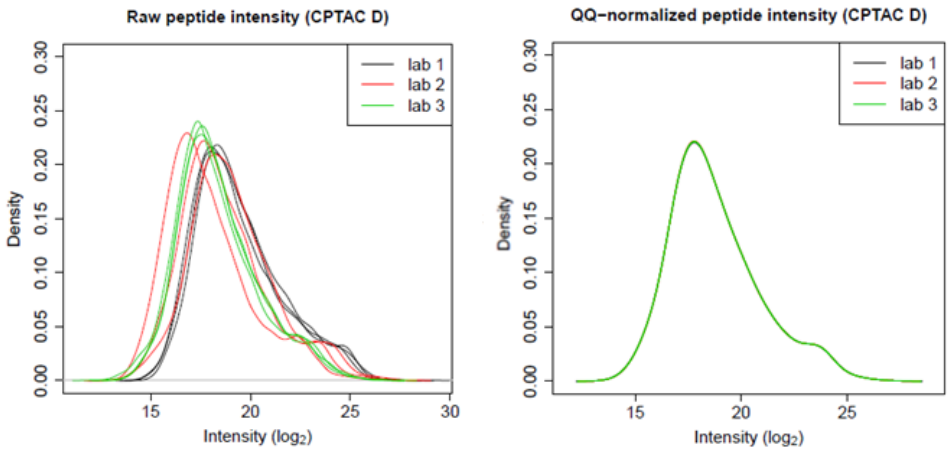


Figure 1.10: Effect of quantile normalization (in [Clement 2019]). In the right hand side plot, all the lines are confounded.

4. As mentioned in Section 1.3, depending on the chosen approach, a distinct missing data *imputation* step, and/or a distinct *summarization* step can possibly be performed as well.

Bringing all together, the data analysis process can then be schematically decomposed into several steps, each step generating output data, which in turn are used as input to subsequent steps. This sequential decomposition is called a *bioinformatic pipeline*. As a summary to this section, typical possible pipelines for MS based quantitative proteomics data are synthetically represented in Figure 1.11.

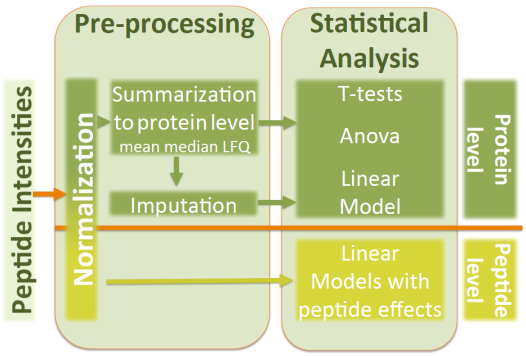


Figure 1.11: Typical MS based quantitative proteomics pipelines (in [Clement 2019])

1.5 Objective and outline

The generic objective of this thesis is to explore the route of complex statistical modelling in order to tackle some of the various data challenges exposed in Section 1.3, including the two following aspects : handling of missingness of different types (MCAR and NMAR), and modelling intensities at peptide level, in an attempt to respectively avoid the imputation and summarization steps, which both come with some drawbacks.

More specifically, this work will focus on selecting a particular form of Bayesian statistical model, and assessing both the practicalities (and possible challenges) of the model implementation, and to which extend the chosen model could bring added value compared to existing methods. As such, this work can be viewed as an exploratory step, which could lay the fundations for a longer term research project. The latter would consist in developing a new statistical analysis procedure, provided as an R package, that would be suited to analyse a wide range of MS based quantitative proteomics datasets.

The dissertation is structured as follows :

- Chapter 2 reviews some relevant litterature. While discussing the different articles, specific attention is paid to the identification of key modelling ingredients, given the research question at hand. This chapter continues with a summary of selected model ingredients, and finally ends with a description of a chosen benchmark dataset, which will be used in the subsequent chapters.
- Applying complex Bayesian multi-parameters models is a challenging task. Therefore, for some of the model implementations, it was resorted to a readily available piece of software, i.e. the Stan platform ([Stan Development Team 2020b]). Chapter 3 is devoted to a description of the Hamiltonian Monte Carlo algorithm that is at the heart of the Stan inferential engine, and to an introduction to the Stan platform itself.
- Chapter 4 focuses on the specification and implementation of a *single protein model*, and its application to the chosen dataset. This model aims at addressing most of the challenges present in the MS based proteomics data, by including most of the model ingredients selected in Chapter 2, at the exception of the hierarchical modelling aspect.
- In Chapter 5, an extension of the single protein model developed in previous chapter, in the form of a hierarchical model, is introduced and implemented. This chapter ends with a short model comparison study, where the performance of the proposed hierarchical model is assessed in comparison to some of other possible models, on the chosen benchmark dataset.
- Finally, in Chapter 6, some conclusions of this work are sketched, and possible directions for further research are indicated.

Chapter 2

Literature review and model ingredients

2.1 Some literature review

2.1.1 Introduction

In this section, a few articles describing existing statistical methods to analyse MS Based Proteomics Data are discussed. Most of these methods are implemented in R packages and available as open source code. For the first three of them, a more detailed description of the core models underlying the methods is provided, because these introduce key modelling ingredients.

Throughout this section, and whenever describing models more formally, one does not necessarily follow the original notations of the corresponding articles, but rather strive to use a common notation, which will also be re-used in subsequent chapters of this dissertation.

2.1.2 Limma

Limma is a widely used statistical R package that is incorporated in many bioinformatics pipelines for different omics data analysis contexts. Its foundations were introduced in Gordon Smyth's seminal paper [Smyth 2004]. Although not originally designed as a proteomics method but rather built for transcriptomics purposes - Limma actually stands for LInear Models for Micro-Arrays - it is nonetheless used in some proteomics pipelines (in conjunction with summarization and possibly imputation techniques - see Section 1.3).

The key methodological idea introduced by Limma is *sample variance moderation*, an ingredient that is ubiquous in subsequently developed, possibly more evolved, methods. This concept is in fact to be related to hierarchical variance modelling, and is hereafter introduced in a proteomics context. Note that in [Smyth 2004], the model exposition is done using a mix of frequentist and Bayesian arguments, while in what follows, one strives to stick as much as possible to the Bayesian paradigm.

Denote by :

- $k : 1, \dots K$ the set of considered proteins in the MS based quantitation analysis

- $i : 1, \dots n_k$ the set of observed samples for protein k (n_k might be k specific due to varying number of NA's per protein)
- $\boldsymbol{y}_k$ the vector ($n_k \times 1$) of log-2 transformed intensities for protein k (e.g. summarized from peptide intensities)

Limma assumes the following linear model :

$$\boldsymbol{y}_k \sim \mathcal{N}(X_k \boldsymbol{\alpha}_k, \sigma_k^2 \operatorname{diag}(\frac{1}{\boldsymbol{w}_k})) \tag{2.1}$$

where :

- $X_k(n_k \times p)$ is the model matrix for protein k, given p chosen regressors
- $\boldsymbol{\alpha_k}$ is the vector of the linear regression coefficients $\alpha_{kj}, j : 1, \dots p$
- σ_k^2 is the (non weighted) residual variance for protein k
- $\boldsymbol{w}_k$ is a n_k-dimensional vector of known weights (most frequently equal to ones)

The key idea of Limma is to introduce a hierarchical structure in the model, by assigning the following conjugate priors :

$$\sigma_k^2 \sim \text{Inv-}\chi^2(\nu_0, s_0^2) \tag{2.2}$$

where Inv-$\chi^2(\nu, s^2)$ is the scaled inverse chi square distribution with ν degrees of freedom and scale parameter s.

Define the following quantities :

$$\begin{aligned} \hat{\boldsymbol{\alpha}}_{\boldsymbol{k}} &= (X_k^T \boldsymbol{W_k} X_k)^{-1} X_k^T \boldsymbol{W_k} \boldsymbol{y_k} \\ s_k^2 &= \frac{(\boldsymbol{y_k} - X_k \hat{\boldsymbol{\alpha}}_{\boldsymbol{k}})^T (\boldsymbol{y_k} - X_k \hat{\boldsymbol{\alpha}}_{\boldsymbol{k}})}{\nu_k} \end{aligned} \tag{2.3}$$

with $\nu_k = n_k - p$ the residual degrees of freedom for protein k.

Then, combining (2.1) and (2.2), one can show that the posterior distribution of σ_k^2 is then:

$$\sigma_k^2 | s_k^2 \sim \text{Inv-}\chi^2(\nu_0 + \nu_k, \tilde{s}_k^2), \text{ with } \tilde{s}_k^2 = \frac{\nu_0 s_0^2 + \nu_k s_k^2}{\nu_0 + \nu_k} \tag{2.4}$$

Actually, in the Limma package, this model is implemented in a frequentist fashion, and translates into a set of so-called *moderated* t-tests, where the null hypothesis is the equality to zero of the corresponding α_{kj} parameter. These frequentist hypothesis tests show an improved power compared to the classical t-tests performed with a non hierarchical version of the linear models. This is due to the increased degrees of freedom - $(\nu_0 + \nu_k)$ instead of ν_k - together with the more stable residual variance estimates, i.e. $\tilde{s}_k^2$, as defined in (2.4).

In Limma, the hyperparameters ν_0 and s_0^2 are estimated using an "empirical Bayes" procedure. The latter basically means that, instead of being set by the user based on prior knowledge, these hyperparameters are estimated from the data themselves. More specifically, the estimation procedure uses moments matching between the theoretical distributions of the

hyperparameters - given the chosen hierarchical model - and their empirical distributions from the data pertaining to the whole set of proteins. More details can be found in [Smyth 2004].

Variance moderation is useful in Omics data analysis situations where the number of samples is low, which is typically true in the proteomics use case. Indeed, it helps providing a more robust estimate of the residual variance for each protein, by borrowing information from the whole proteins set. Therefore, it stabilizes the outcome of the analysis and makes it more reproduceable. Note also that, by applying variance moderation, one makes the implicit assumption that the error variances of the protein intensities, while not being equal, are sample units of a common population of variances.

2.1.3 MSqRob

MSqRob - standing for ROBust MS based Quantitation - is a statistical method dedicated to proteomics quantitative data, acquired through Mass Spectrometry, which is also available as an R package. The method was first described in [Goeminne, Gevaert, and Clement 2016], and further refined in subsequent articles, e.g. [Goeminne, Sticker, et al. 2020].

At the heart of the method is a protein-wise linear model with mixed effects, of which the responses are the intensities at peptide level.

Denote by :

- $k : 1, \ldots, K$ the set of proteins
- $p : 1, \ldots, P_k$ the set of identified peptides of protein k
- $r : 1, \ldots, R$ the set of mass spectrometer runs (equal to the set of samples in Label Free Quantitation)
- y_{kpr} the $\log_2$ transformed intensity of peptide p (among the identified peptides of protein k,) observed in run r

MSqRob assumes the following model for the log intensities (note that by convenience the protein index k, which otherwise could be added everywhere, is here dropped):

$$y_{pr} = \alpha_0 + \boldsymbol{x}_{\boldsymbol{pr}}^T \boldsymbol{\alpha} + \alpha_p^{peptide} + \beta_r^{run} + \epsilon_{pr} \tag{2.5}$$

where :

- α_0 is the intercept;
- $\alpha_p^{peptide}$ is a fixed effect of individual peptide p;
- β_r^{run} is a random run effect ($\beta_r^{run} \sim \mathcal{N}(0, \sigma_{run}^2)$);
- $\boldsymbol{x_{pr}} = (x_{1,pr}, \ldots, x_{m,pr})^T$ is a column vector extracted from the model matrix, related to m chosen predictors (apart from peptide and run);
- $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_m)^T$ is the vector of corresponding regression coefficients;
- $\epsilon_{pr} \overset{i.i.d.}{\sim} \mathcal{N}(0, \sigma^2)$

One notices that using peptide - and not protein - intensities as response of the model, leads to the introduction of :

- peptide effects (fixed in the above model formulation), which take into account the physico-chemical properties of peptides having a peptide-specific impact on their corresponding MS measured intensities (e.g. ionization efficiency, cf. Section 1.3)
- a random run effect, which accounts for the correlation of peptide intensities originating from the same biological sample (run)

On top of this core protein-wise mixed model, MSqRob introduces three modular improvements :

1. Empirical Bayes variance estimation, which shrinks the individual protein variances towards a common prior variance. This improvement implements the Limma moderated variance concept, explained in Section 2.1.2.
2. Ridge regression, which penalizes the size of the model parameters, with the aim to avoid overfitting. Ridge regression is a regularization technique which can be used to handle collinearity problems in linear regression models - see Appendix A.1. In MSqRob, it is implemented by taking advantage of a theoretical link between ridge regression and random effect models - see Appendix A.2. As a result, the peptide fixed effects, and possibly some or all of the remaining fixed effects introduced in (2.5), are in fact replaced by corresponding random effects.
3. M-estimation with Huber weights, which is a particular case of robust regression, aimed at making the estimators more robust with respect to outliers. See Appendix B.1 for more details.

By introducing peptide level modelling, and thanks to the three above ingredients, the authors were able to demonstrate, on some benchmark dataset, noticeable improvement of estimation, sensitivity and specificity compared to some competing summarization based methods, which run the statistical analysis at aggregated protein data level.

2.1.4 ProDA

ProDA is a recently published statistical method [Ahlmann-Eltze and Anders 2020] which is also available as an R package. Unlike MSqRob, proDA opts for modelling intensities at protein level - therefore after a summarization method has been applied to the peptide intensities.

The most interesting characteristic of the proDA method is that it introduces an explicit modelling of the NMAR missingness pattern observed with MS based quantitative proteomics data (see Section 1.3). Indeed, proDA stands for *PRObabilistic Drop-out Analysis*, and its model assumes a dropout probability curve, which links the probability of observing a protein intensity, to the level of its latent intensity value. Hereafter is a short mathematical description of the chosen model.

Denote by :

- $k \in \{1, \dots K\}$ the set of proteins

- $i \in \{1, \dots n\}$ the set of samples
- $\boldsymbol{y}$ the vector of protein (latent) log-2 transformed intensities, with y_{ki} being the log-2 transformed intensity of protein k, in sample i, which can be observed or not (missing data)
- r_{ki} be an indicator variable such that $r_{ki} = 1$ if y_{ki} is observed, and $r_{ki} = 0$ otherwise

ProDA assumes the following model :

$$\begin{aligned} \mathbb{E}(\boldsymbol{y_k}) = \boldsymbol{\mu}_k &= X\boldsymbol{\alpha}_k \\ y_{ki}|\mu_{ki}, \sigma_k^2 &\sim \mathcal{N}(\mu_{ki}, \sigma_k^2) \\ r_{ki}|y_{ki}, \rho_i, \zeta_i^2 &\sim \text{Ber}(\Phi(y_{ki}; \rho_i, \zeta_i^2)) \end{aligned} \tag{2.6}$$

where :

- $\boldsymbol{\mu}_k$ is the vector of mean intensities for protein k
- $\boldsymbol{\alpha}_k$ is the set of regression coefficients for protein k
- X is the model matrix, which is dependent on the design of experiment
- $\text{Ber}(p)$ is the Bernoulli distribution with parameter p
- $\Phi(x; \mu, \sigma^2)$ is the normal cumulative distribution function with mean μ and variance σ^2

In (2.6), the missingness process is modelled using a sigmoïd shaped drop-out curve, which is parameterized with an inflection point ρ_i and a scale ζ_i. The latter parameters are sample dependent. The following priors are also introduced:

$$\begin{aligned} \mu_{ki}|\nu_{0\mu}, \mu_0, \sigma_0^2 &\sim t_{\nu_{0\mu}}(\mu_0, \sigma_0^2) \\ \sigma_k^2|\nu_0, s_0^2 &\sim \text{ Inv-}\chi^2(\nu_0, s_0^2) \end{aligned} \tag{2.7}$$

where

- $t_\nu(\mu, \sigma^2)$ is the Student t distribution with ν degrees of freedom, mean μ and scale parameter σ
- Inv-$\chi^2(\nu, s^2)$ is the scaled inverse chi square distribution with ν degrees of freedom and scale parameter s

Note that the prior specifications for σ_k^2 are completely identical as in Limma - see (2.2).

Then, the model fitting procedure translates into an iterative algorithm applying two sequential steps in each iteration (for more details, see [Ahlmann-Eltze and Anders 2020]) :

- *Maximum a posteriori* approach to find the best estimators for the parameters of the model - i.e. $\boldsymbol{\mu}_k, \sigma_k^2$ for each protein k, and ρ_i, ζ_i^2 for each sample i - given the hyperparameters $\nu_{0\mu}, \mu_0, \sigma_0^2, \nu_0$ and s_0^2;
- Empirical Bayes approach to estimate the hyperparameters, given the parameters obtained in the previous step.

In [Ahlmann-Eltze and Anders 2020], the authors compared the performance of their method with standard proteomics methods, which make use of various imputation techniques. They were able to demonstrate some superiority of proDA, in terms of both False Dicovery Rate control and power, on a specific benchmark dataset.

2.1.5 Other articles

Apart from the previously described articles, there are many other available methods that are commonly used in proteomics bioinformatic pipelines, and describing them all - or even only the most used ones - would be beyond the scope of this dissertation. However, as part of this work, some of the most recent articles addressing (parts of) this work proteomics research question were also investigated. Some of the ideas found in these papers are herebelow sketched briefly.

In [Goeminne, Sticker, et al. 2020], the authors extend the MSqRob methodology (cf. Section 2.1.3) towards a two-step "hurdle model", where both data missingness and intensities are modelled, at peptide level. More specifically, a first step consists in modelling the probability of peptide identification in a logistic regression. From this first step, one aims at identifying the proteins which are differentially detected between two or more conditions. In a second step, one models the peptide intensities, but then conditionally on the identification of these peptides. This second step involves a similar model specification as in MSqRob - cf Equation 2.5 - and uses the same covariates as in the logistic regression step. The *MSqRob-Hurdle* method can then provide p-values for testing the differential detection OR differential abundance of each protein.

[The and Käll 2019] propose a conceptually appealing Bayesian framework where several steps of the Mass Spectrometry data acquisition process of bottom-up MS based proteomics - as described in Section 1.2 - are modelled together in a *probabilistic graphical model* (PGM). This PGM combines several error models, which are often considered separately in typical proteomic pipelines, and the advantage of this combination is, according to the authors, that global False Discovery Rate control in hypothesis testing can then be preserved. The proposed approach leads to a fairly complex Bayesian hierarchical model, of which the various hyperparameters are estimated from the data using an empirical Bayesian approach. Let us mention the following interesting features of the resulting *Triqler* (TRansparent Identification-Quantification-Linked Error Rates) method :

- the identification of *peptide spectrum matches*, or PSM, which refers to the MS2 identification step mentioned in Section 1.2, is one of the explicitly modelled MS steps. This means that the model makes use of the data produced, at yet a lower level than peptide i.e. the PSM level.

- one of the boxes present in the specified PGM is the modelling of the missingness mechanism in a NMAR only fashion, using a sigmoid shaped drop out curve, comparable to proDA (see Section 2.1.4).

Finally, in [Mallikarjun, Richardson, and Swift 2020], which presents the *BayesENproteomics* R package, the authors make use of protein-level data to estimate a complex Bayesian hierarchical model, where one can find back some ingredients that were also present in the above mentioned articles, sometimes in slightly different variants :

- linear regression models, with some regularization penalties (*Elastic Nets* which can be viewed as a trade-off between ridge and Lasso regularization). Regularization is also present in MSqRob, through ridge regression (Section 2.1.3);

- missingness modelling using a logistic regression, but here with the aim to distinguish between MAR and NMAR missingness (adaptive multiple imputation);
- variance moderation, as introduced in Limma (Section 2.1.2), and also used in all other methods mentioned in this section.

2.2 Selection of model ingredients

In this section, a selection of desirable model ingredients is performed. Recall that the objective is to apply a (possibly complex) Bayesian statistical model to proteomics data, with the aim to avoid the imputation and summarization steps. Taking into account this objective, and based on the limited literature review performed in Section 2.1, one comes up with the following features :

Proteinwise linear regression model Among the articles reviewed in Section 2.1, all of them describes methodologies using proteinwise linear regression models as a core. Essentially, the multi-protein aspect - referring to the fact that the same statistical tests need to be performed for a number of proteins - mostly comes into play through the use of multiple testing corrections, or through the use of sample variance moderation techniques (hierarchical model). In this work, the same kind of core approach will therefore be followed, using linear regressions to model the response variables (intensities or missingness, see below), and the same model specifications will be repeated, protein by protein.

Random effects One of the goals of this work is to avoid summarization, and therefore one of the key choices will be to model the response intensities at peptide level. In order to do so for a typical proteomics design of experiment, and as indicated in [Goeminne, Gevaert, and Clement 2016], one will need to include some peptide effect, and some biological sample (or run) effect in order to distinguish between biological and technical variability. These effects are better modelled as random effects. Moreover, having random effects as an ingredient of the model will also allow to apply some sort of ridge regularization through the link between ridge regression and random effect models, as in MSqRob method (see also Appendix A.2).

Robust regression As indicated in Section 1.3, a characteristics of MS based proteomics data is their noisiness and the presence of numerous outliers. Therefore, robust regression methods will be preferred here to reduce the impact of such data on estimation. In a Bayesian modelling context, the robust regression will be implemented as a Student-t regression (see Appendix B.2). For the sake of simplicity, the degrees of freedom parameter of the Student t distribution will be set to a fixed low value (3 to 5, say) instead of attempting to infer it from the data.

Modelling missingness Another objective is to avoid the use of pre-imputed values for missing data. Therefore, taking inspiration from some of the articles reviewed in Section 2.1 ([Ahlmann-Eltze and Anders 2020], [The and Käll 2019] and [Mallikarjun, Richardson, and Swift 2020]), the chosen approach chooses for the explicit modelling of the missingness mechanism. In particular, the ability to link missingness and intensity (NMAR), in some sort of drop-out curve model, appears to be important. However,

as mentioned in Section 1.3, it is also a fact that missingness in MS based Label Free quantitative proteomics is not purely NMAR, but also contains a MCAR component. Therefore the ability to introduce both types of missingness could be an interesting feature as well.

Hierarchical set-up Finally, the methods discussed in Section 2.1 introduce some kind of hierarchical models, at least to perform variance moderation as in Limma. Indeed, the hierarchical set-up allows to borrow information from the whole set of proteins, which can be useful when the information contained in the data pertaining to a single protein is not sufficient to accurately estimate some of the model parameters. Therefore one would like to include this element in the model proposals.

It is clear that the inclusion of all the ingredients selected above inevitably leads to the construction of a rather complex model. Therefore, in the following chapters, one will opt for a stepwise implementation. Initially, one will consider a simplified version, consisting of a single protein model (Chapter 4), while leaving the multidimensional aspect that comes with the addition of a hierarchical set-up, for a second stage (Chapter 5).

2.3 CPTAC benchmark dataset

The last section of this chapter is devoted to the description of the benchmark dataset that will be used throughout the remainder of this work. This dataset is derived from the CPTAC[1] Technology Assessment Study 6 dataset[2], which has been produced to assess the repeatability and reproducibility of the MS technology for proteomics data analysis. This dataset has also been used as benchmark dataset in the following MSqRob articles : [Goeminne, Gevaert, and Clement 2016] and [Goeminne, Sticker, et al. 2020].

In this benchmark experiment, 48 human UPS1[3] proteins were spiked at 5 different concentrations (from 0.25 to 20 fmol/μl, samples denominated 6A to 6E) in a 60ng/μl yeast lysate (Figure 2.1). Each of these 5 samples of different concentration, all originating from the same biological master preparation, were analysed with 3 different mass spectromers located in different labs (denominated *lab1*, *lab2* and *lab3* in Figure 2.1). Finally, each spectrometer measure was in turn performed 3 times, using Label Free Quantitation. With this CPTAC dataset, the main goal of the data analysis is to identify, from the data samples, which proteins were differentially abundant in the 5 different conditions (corresponding to different concentrations of UPS1 proteins). However, note that in the analysis performed in the remainder of this work, only samples of concentrations A, B and C will be selected. Indeed, according to [Goeminne, Sticker, et al. 2020], higher concentration samples (i.e. 6D and 6E) suffer from greater ionization competition effects, which degrades substantially the quality of the data attached to these 2 samples.

[1] US NCI Clinical Proteomic Tumor Analysis Consortium
[2] https://cptac-data-portal.georgetown.edu/cptac/study/list/10423?scope=Phase+I#10423
[3] UPS : Universal Proteomics Standard

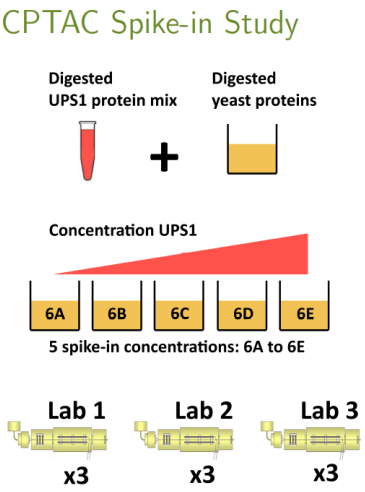


Figure 2.1: Schematic representation of CPTAC benchmark dataset (in [Clement 2019])

What makes this dataset an interesting benchmark dataset is the fact, when comparing different spike-in concentrations, that one knows *beforehand* that only the human UPS1 proteins should be flagged as differentially abundant, whereas the proteins of the yeast *background* should not be flagged. Moreover, one knows the true abundance ratios between the different conditions (*log-fold-changes*), and this therefore allows calculation of the true estimation errors attached to any data analysis methodology.

However, somehow the price to pay to obtain a benchmark dataset is that it is also very specific in nature, in comparison with data encountered in typical protomics data analyses. Indeed, since all samples were created based on the same biological master preparation, no biological variability is in fact present between these samples, which makes the dataset less representative of a true "lab situation".

In the next chapters, one will systematically make use of the peptide level data obtained from the above described subset of the CPTAC database. More specifically, data produced after log transformation and quantile normalization will be used. Note that studying the impact of these two pre-processing steps on the data analysis results would make sense, but goes beyond the scope of this work.

In terms of missingness, the dataset contains, at peptide level, 47% of missing values overall. Figure 2.2 shows the missingness pattern observed, as a function of the condition (concentration 6A to 6C) and as function of the lab (1 to 3). On the one hand, the graph on the left shows that the percentage of missing values seems roughly the same between the different conditions. This was to be expected since one knows that only 48 proteins out of the 1347 identified proteins are differentially abundant between conditions. On the other hand, the graph on the right clearly shows a difference of missing data percentage between the 3 different labs, where, for example, lab 1 has a much higher missingness rate (almost 68%) compared to lab 2 (bearly 25%). This finding will be reflected in the choice of model specification in subsequent chapters.

Figure 2.2: Missingness pattern in the CPTAC dataset

Chapter 3

Hamiltonian Monte-Carlo and Stan software platform

3.1 Introduction

In Chapter 2, a list of desirable model ingredients was selected in order to build a statistical method applied to proteomics data. At this stage, one already realizes that combining these modelling elements, and taking into account the high dimensionality aspect of the data, will obviously lead to high-dimensional complex statistical models.

In Bayesian inference with complex multi-parameters models, the main issue is not to obtain the joint posterior distribution of the model parameters - its expression, up to a multiplicative constant, can be obtained in a rather straighforward way by applying Bayes′theorem. Instead, the key challenge is the ability to generate samples from this joint posterior distribution, in order to answer the inferential question(s) at hand. To implement this sample generation, one typically relies on *Markov Chain Monte Carlo*(*MCMC*) methods ([Metropolis et al. 1953], [Hastings 1970]). In practice however, for non trivial models, implementing such methods to the required level of efficiency and robustness requires a great deal of programming and tuning investment (see e.g. [Gelman et al. 2013], Chapters 11 and 12).

Therefore, instead of building an exact Bayesian inference engine from scratch for the selected model(s), the choice has been made, in this work, to use a piece of software that is readily available to the Bayesian statistical community, i.e. the Stan software platform [Stan Development Team 2020b]. In this chapter, one introduces the Hamiltonian Monte Carlo method, which is at the heart of Stan inferential engine, and further describes some key elements of the Stan software platform itself.

3.2 Hamiltonian Monte Carlo

3.2.1 Conceptual introduction

Hamiltonian Monte Carlo, a.k.a. *Hybrid Monte Carlo*, or *HMC* ([Neal et al. 2011]) is a specific type of MCMC algorithm, which takes its inspiration from the world of physics, and can be best understood by using an analogy with a hypothetical mechanical system. The

mental representation of this analogy is easier for two dimensional distributions, but it can be extended to multidimensional posteriors. Consider a ball with a certain kinetic energy rolling frictionlessly around a landscape with valleys and hills defined by the posterior distribution for which samples need to be drawn. More specifically, the ground shape is determined by the opposite of the log-posterior distribution, such that peaks correspond to zones of low density values, while the deeepest valleys correspond to modes of the distribution. To generate a sample from the target distribution, with the ball lying on an initial position, an initial speed and direction are picked at random, and the ball is then left rolling, possibly eventually turning around. The sample will then correspond to the position of the ball along its path, taken at a pre-specified time interval where the ball is stopped. Then another sample can be generated, by pushing again the ball, from its current position, in another random trajectory. By repeting this a big number of times, one can learn about the shape of the log posterior distribution.

Figure 3.1[1] illustrates an example of generated HMC trajectory, in the simple case of a two-dimensional problem with a bivariate normal posterior distribution. On this picture, the concentric circles are the contours of the target two-dimensional distribution, and the smaller grey dots represent the previously generated samples. The current trajectory, i.e. the series of thick black dots, starts from the right, with the randomly chosen initial speed vector represented as the grey arrow. One can then imagine the resulting trajectory of the ball, curving left as it is attracted by the valley corresponding to the mode of the distribution. The position of the final thick black dot of the trajectory is then taken as the newly created sample. On the axes, the target marginal distributions are drawn, together with the histograms obtained from the previously generated samples.

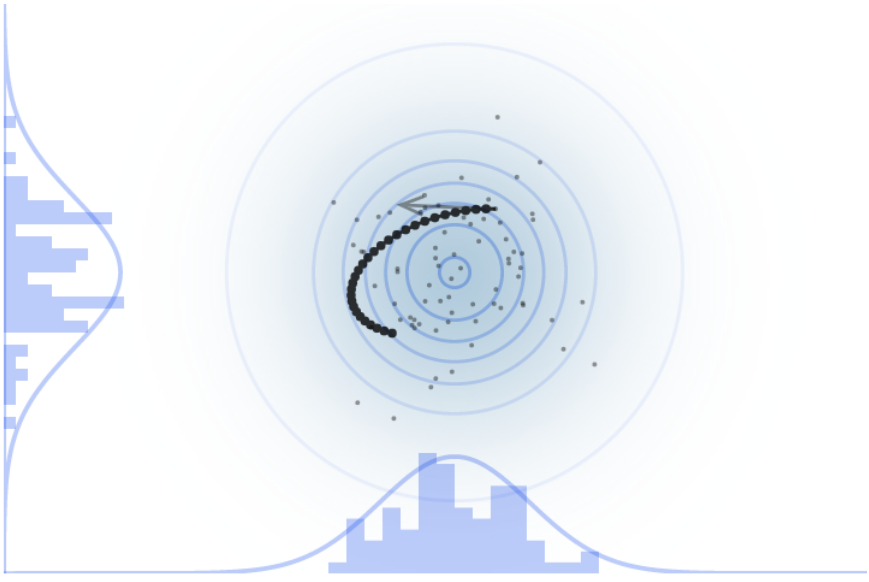


Figure 3.1: An example of HMC trajectory in a bivariate normal posterior distribution case

[1]source : blog of Richard McElreath :“Occasional text on evolutionary anthropology, statistical inference, and the intersection of the two.”; http://elevanth.org/blog/2017/11/28/build-a-better-markov-chain/

3.2.2 Hamilton's equations

Mathematically, the mechanical system described in Section 3.2.1 is governed by *Hamiltonian dynamics*, which translates the conservation of energy principle into a set of mathematical equations, i.e. Hamilton's equations. These equations in turn drive the evolution in time of the different components of the position and speed of the ball introduced above.

Indeed, assume that the position of the ball is described by a d-dimensional vector $\boldsymbol{\theta} = \{\theta_1, \ldots, \theta_j, \ldots, \theta_d\}$, while its speed, or *momentum*, is described by another d-dimensional vector $\boldsymbol{\phi} = \{\phi_1, \ldots, \phi_j, \ldots, \phi_d\}$. Denoting by $p(\boldsymbol{\theta})$ the distribution from which one needs to get samples from, define the *Hamiltonian* as the following quantity :

$$H(\boldsymbol{\theta}, \boldsymbol{\phi}) = U(\boldsymbol{\theta}) + K(\boldsymbol{\phi}) \tag{3.1}$$

where :

- $U(\boldsymbol{\theta}) = -\log p(\boldsymbol{\theta})$ is called the *potential energy*;
- $K(\boldsymbol{\phi}) = \frac{1}{2}\boldsymbol{\phi}^T M^{-1}\boldsymbol{\phi}$ is called the *kinetic energy*, with M a diagonal positive mass matrix - see also section 3.2.3.

Then, Hamilton's equations, governing the trajectory, are :

$$\begin{aligned} \frac{d\theta_j}{dt} &= \frac{\partial H}{\partial \phi_j} \\ \frac{d\phi_j}{dt} &= -\frac{\partial H}{\partial \theta_j} \end{aligned} \tag{3.2}$$

for $j = 1, \ldots, d$.

Using (3.1) and (3.2), one can show that the Hamiltonian $H(\boldsymbol{\theta}, \boldsymbol{\phi})$, representing the total energy, stays invariant along a trajectory, hence enforcing the conservation of energy principle in a frictionless mechanical system.

3.2.3 HMC algorithm

Algorithmically, implementing the physics analogy exposed in Section 3.2.1 and Section 3.2.2 will lead to doubling the number of dimensions of the space explored by the MCMC method. Indeed, for each component θ_j of the model parameters set, one needs to add one momentum variable ϕ_j. Both vectors $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ are then updated together in a specific Metropolis-type algorithm, in which the jumping distribution for $\boldsymbol{\theta}$ is determined largely by $\boldsymbol{\phi}$. The latter comes from the fact that $\boldsymbol{\phi}$, which corresponds to the initial speed vector given to the ball in all directions, is effectively picked at random, while $\boldsymbol{\theta}$, corresponding to the position vector, results from the equations governing the movement of the ball, i.e. (3.1) and (3.2). This explains why the term "Hybrid Monte Carlo" is also used, because HMC combines MCMC with a deterministic simulation method.

In HMC, the posterior density $p(\boldsymbol{\theta}|\mathcal{D})$, with $\mathcal{D}$ representing the data, is therefore augmented by an independent distribution $p(\boldsymbol{\phi})$ on the momenta, thus defining a joint distribution $p(\boldsymbol{\theta}, \boldsymbol{\phi}|\mathcal{D}) = p(\boldsymbol{\phi})p(\boldsymbol{\theta}|\mathcal{D})$. It is usual to give $\boldsymbol{\phi}$ a multivariate normal distribution with

mean 0 and diagonal covariance ("mass") matrix M, already introduced in (3.2). Note that, due to above chosen model specifications, the posterior and prior distributions of $\boldsymbol{\phi}$ are therefore identical : $\boldsymbol{\phi}, (\boldsymbol{\phi}|\mathcal{D}) \sim \mathcal{N}(0, M)$.

Unfortunately, Hamilton's equations (3.2) are partial differential equations (PDE's), of which the solution cannot in general be obtained exactly, but needs numerical simulation using a discretization scheme. For the Hamiltonian system of differential equations, a good scheme is the so-called *leapfrog* scheme. Algorithm 1 (adapted from [Gelman et al. 2013], Section 12.4) shows how to practically update the sample values within a single MCMC iteration. Note that the leapfrog scheme requires from the user the pre-specification of a small step size $\epsilon > 0$ and a number of steps used in each iteration, L. As with any other MCMC algorithms, these iterations are repeated until approximate convergence of the Monte Carlo chains (see Section 3.3.3).

Algorithm 1 t_{th} iteration of HMC simulation

▷ sample from (posterior) distribution of $\boldsymbol{\phi}$

$\boldsymbol{\phi}_t \leftarrow \mathcal{N}(0, M)$

▷ init values for the trajectory at iteration t

$\boldsymbol{\phi}^* \leftarrow \boldsymbol{\phi}_t$; $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}_{t-1}$

▷ mimic physical dynamics by applying L leapfrog steps

for $l \in 1, \ldots, L$ **do**

$\quad \boldsymbol{\phi}^* \leftarrow \boldsymbol{\phi}^* + \frac{1}{2}\epsilon \frac{d}{d\boldsymbol{\theta}} \log p(\boldsymbol{\theta}|\mathcal{D})\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^*}$

$\quad \boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}^* + \epsilon M^{-1}\boldsymbol{\phi}$

$\quad \boldsymbol{\phi}^* \leftarrow \boldsymbol{\phi}^* + \frac{1}{2}\epsilon \frac{d}{d\boldsymbol{\theta}} \log p(\boldsymbol{\theta}|\mathcal{D})\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^*}$

end for

▷ calculate probability of accepting the proposed θ^*

$r = \min\left(\frac{p(\boldsymbol{\theta}^*|\mathcal{D})p(\boldsymbol{\phi}^*)}{p(\boldsymbol{\theta}^{t-1}|\mathcal{D})p(\boldsymbol{\phi}^{t-1})}, 1\right)$

▷ apply acceptation criterium

Sample $u \sim U(0,1)$; $\boldsymbol{\theta}^t = \begin{cases} \boldsymbol{\theta}^* & \text{if } u < r \\ \boldsymbol{\theta}^{t-1} & \text{otherwise.} \end{cases}$

On the one hand, the advantage of HMC compared to other MCMC methods like Gibbs or Metropolis-Hastings - run on the original parameter space $\theta \in \Theta$ - is the expected improvement on the random-walk behaviour, that might be observed with these alternative algorithms. This improvement can be substantial when dealing with difficult-to-sample types of distributions (i.e. displaying challenging contour shapes for blind samplers), which are

often encountered in high dimensional spaces. Indeed, in such cases, HMC is likely to need much less iterations to get representative sampling, because :

- distances between successive generated points are typically larger;
- probabilities of rejection of new samples is typically lower, so fewer calculations will be done in vain.

On the other hand, the price to pay with HMC is that it needs the computation of the analytical gradient of the log posterior (as numerical approximation of the gradient would be computationally too demanding) at each iteration. Therefore the computing power needed at each iteration is greater than for competing MCMC algorithms.

3.2.4 NUTS extension

Actually, the Stan software is based on an extension of HMC, called *No U-Turn Sampler*, a.k.a. *NUTS*, of which the main idea is now sketched. Indeed, one of the difficulty with using the HMC algorithm is that there is a need to properly finetune the number of leapfrog steps per iteration, L :

- If it is too low, the algorithm will not take full advantage of the Hamiltonian trajectories at each iteration, resulting in more iterations to achieve sufficient exploration of the target posterior distribution;
- If it is too high, the trajectories will tend to turn around (*U-Turns*) wasting precious computational resources on exploration with no additional return.

The HMC user can of course try to tune the number of steps by hand, but that is not so easy when the target distribution is complex. Therefore the idea of NUTS is to try to adaptively set the L parameter by figuring out when the path starts to turn around. In order to do this efficiently, it needs to simulate the path in both directions. When the path starts to turn around, NUTS stops the simulation and takes a sample. In practice, beyond this simple idea, the NUTS leads to algorithms that are rather technical, and which come with a number of adaptive optimizations. See [Hoffman and Gelman 2014] for a more detailed description.

Having solved the problem of choosing the L parameter, thanks to NUTS, one is left with the choice of the ϵ parameter, i.e. the step size used in the leap frog discretization scheme. There, one typically relies to heuristics. For example, according to [Gelman et al. 2013], theory suggests that HMC is optimally efficient when its acceptance rate is approximately 65%. Therefore, one can manually tune the ϵ parameter to approach this acceptance rate. If the obtained acceptance rate is too low, then ϵ would need to be decreased. If it is too high, then one can afford to increase the ϵ. In more advanced implementations of NUTS, built-in automatic adaptation of ϵ might be performed, as in the Stan platform (see Section 3.3.2).

3.3 Stan software platform

Stan [Stan Development Team 2020b] (*Sampling Through Adaptive Neighborhoods*[2]) is an open-source software platform developed and supported since 2012 by the Stan development

[2]Stan is also named in honour of Stanislaw Ulam, one of the pioneers of the Monte Carlo method.

team, which originated from Prof. Andrew Gelman's lab (University of Columbia). It is readily and freely available for use, and comes with various interfaces, among which the *rstan* R package [Stan Development Team 2020a], which is used in this work.

3.3.1 Stan model language and inferential engine

At the heart of Stan software is a Domain Specific Language, Stan probabilistic programming language for statistical inference [Carpenter et al. 2017]. Language specific statements are written by the user to specify a Bayesian statistical model step by step. More specifically, these lines goes into defining the log probability density of the data and parameters, with code for looping, conditioning, and so on ([Gelman et al. 2013], Section 12.6). Standard distributions such as the normal, gamma, binomial,... are pre-programmed and arbitrary distributions can be entered by programming directly the log density.

In order to illustrate the Stan probabilistic programming language, a simple example, implementing a linear regression model, is now provided. The model specifications are as follows. Denote by :

- n the number of observations
- p the number of predictors
- X (of dimensions $n \times p$) the model matrix
- $\boldsymbol{y} = (y_1, \dots y_n)$ the response vector
- $\boldsymbol{\beta} = (\beta_1, \dots, \beta_p)$ the regression coefficients parameters
- σ^2 the error variance

The chosen linear regression model is :

$$\begin{aligned} \boldsymbol{y} &\sim \mathcal{N}(X\boldsymbol{\beta}, \sigma^2 I_n) \\ \boldsymbol{\beta} &\sim \mathcal{N}(\boldsymbol{\mu}_{0\beta}, \Sigma_{0\beta}) \\ \sigma^2 &\sim \text{Inv-}\chi^2(\nu_0, s_0^2) \end{aligned} \tag{3.3}$$

The Stan script implementing the model specified in (3.3) is provided as a code sample in Figure 3.2.

As mentioned before, Stan uses NUTS (see Section 3.2.4) as its default inferential engine. A run starts by parsing the model script, and converting it in C++ code, which is in turn compiled into executable code. At this stage, Stan automatically adds some information needed to compute the analytical gradient of the log-posterior distribution specified by the user. This is done throught the use of adjunct analytic differentiation (AAD), of which the main principle is the automatic iterative use of the derivative chain rule through the program statements [Naumann 2012].

In addition to the data, parameters and model statements (see Figure 3.2), a Stan run also needs the user provision of some typical MCMC simulation parameters (number of chains, number of iterations per chain, number of iterations taken in the warm-up phase, etc.), as

```
data {
  int<lower=1>  n;                 // number of observations
  int<lower=1>  p;                 // number of predictors
  vector[n]     y;                 // vector of responses
  matrix[n, p]  X;                 // predictor matrix
  vector[p]     mu0;               // beta prior mean
  matrix[p, p]  Sigma0;            // beta prior covariance matrix
  real          nu0;               // sigma2 prior degrees of freedom
  real<lower=0> s0;                // sigma2 prior scale
}

parameters {
  vector[p]     beta;              // regression coefficients for predictors
  real<lower=0> sigma2;            // error variance
}

model {
  y      ~ normal( X * beta, sqrt(sigma2) );
  beta   ~ multi_normal( mu0, Sigma0 );
  sigma2 ~ scaled_inv_chi_square( nu0, s0 );
}
```

Figure 3.2: An example of Stan model program

well as various control parameters that can be set by default. Initial simulation values can be user supplied or otherwise generated randomly by the engine, which then takes into account the actual boundaries of each model parameter.

3.3.2 Tuning HMC specific parameters

As mentioned in Section 3.2.3, the Hamiltonian Monte Carlo engine, in its raw version, needs the user tuning of its specific algorithmic parameters : the number of leapfrog steps L, the step size ϵ, and the mass matrix M. The NUTS extension resolves the tuning of L (Section 3.2.4), and Stan adds some dynamic adaptation of M and ϵ (see [Stan Development Team 2020b]). These adaptations are performed during the warm-up phase, with the aim to come up with some values for M and ϵ that are reasonably suited to the specific posterior distribution to sample from. This adaptation procedure can in turn also be tuned by some second layer parameters, which are typically set by default. Finally, the user can toggle off this additional warm-up adaptation procedure if it happens not to work well for the specific case at hand.

3.3.3 Stan diagnostics

Stan engine comes with a number of diagnostic tools, that can be used to check whether the inference based on the obtained MCMC chains can be done reliably. In this section, one elaborates on three main diagnostic indicators. The first one - divergence monitoring - is specific to HMC, while the two remaining ones - $\hat{R}$ and effective sample size - are applicable to any MCMC method.

3.3.3.1 Divergences

Divergences are related to the approximation done when discretizating the trajectory of the fictious ball introduced in Section 3.2.1, using the leapfrog scheme (lines 4 to 6 in Algorithm 1).

Indeed, as explained in Section 3.2.2, a key feature of the Hamiltonian dynamics is that the Hamiltonian - defined in (3.1) - is conserved along the trajectory followed by the ball. However, in practice, due to the finite ϵ step used in the leapfrog discretization scheme, this

Hamiltonian will not be fully conserved. Therefore, a *divergence* arises when the simulated Hamiltonian trajectory departs from the true trajectory. Divergence is measured by the absolute difference between the Hamiltonian end value and its initial value. When this divergence is too high, the simulation has "gone off the rails" and cannot be trusted. This can happen, for example, when the stepsize ϵ is too high, related to the local curvature of the posterior distribution.

Stan output reports divergences that occur above a specific threshold as warnings and provide programmatic ways to access which iterations have encountered those divergences.

3.3.3.2 Potential scale reduction statistic, or $\hat{R}$

Like for any MCMC method, the validity of the inferences done with the Stan NUTS engine rely on the stationary assumption of the simulated MCMC chains, hence on the convergence of these chains. The general approach here is to compare different simulated sequences, by "cutting" generated MCMC chains (after discarding the warm-up phase) into fragments of equal length, and/or by generating different MCMC chains from different initial values. One then checks the "good mixing" of these chains, i.e. whether they visually appear to have similar distributional behaviours. For example, Figure 3.3 shows two possible examples of chains that have not mixed.

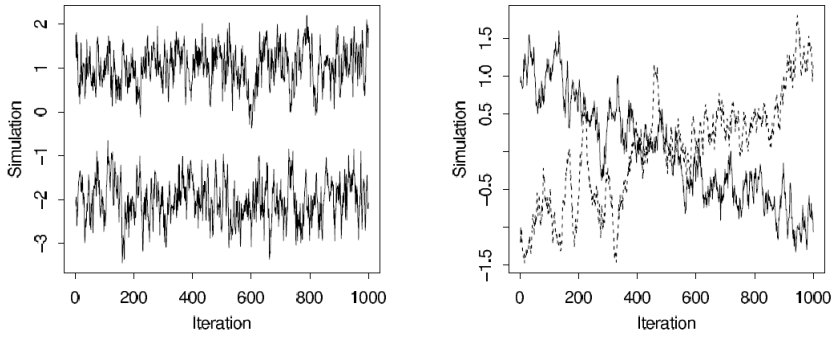


Figure 3.3: Two examples of non converged MCMC chains. On the left plot, either sequence appears to be stable, but they have not converged to the same distribution. On the right plot, both sequences appear to cover the same distribution, but none of them is stationary.

These graphs can provide a useful diagnostic tool but unfortunately, they rely on subjective user interpretation, and even more importantly, they are very cumbersome to apply in high-dimensional problems where quantities of interest can be numerous. Therefore, Stan provides a method to assess convergence by comparing the between- and within- sequence variances, which can be automated for all quantities of interest ([Gelman et al. 2013], Section 11.4). Herebelow, this method, based on the computation of the so-called *potential scale reduction statistic*, or $\hat{R}$, is briefly explained.

For each scalar quantity of interest ψ, let ψ_{ij} be the value obtained for the ith iteration of the jth (fragment of) chain (with $i = 1, \ldots, n; j = 1, \ldots, m$). One start by defining the

following quantities :

$$\begin{aligned}
\bar{\psi}_{.j} &= \frac{1}{n}\sum_{i=1}^{n}\psi_{ij} \\
\bar{\psi}_{..} &= \frac{1}{m}\sum_{j=1}^{m}\bar{\psi}_{.j} \\
B &= \frac{n}{m-1}\sum_{j=1}^{m}(\bar{\psi}_{.j}-\bar{\psi}_{..})^2 \\
s_j^2 &= \frac{1}{n-1}\sum_{i=1}^{n}(\psi_{ij}-\bar{\psi}_{.j})^2 \\
W &= \frac{1}{m}\sum_{j=1}^{m}s_j^2
\end{aligned} \tag{3.4}$$

In (3.4), B and W are the between- and within-chain variances, respectively. The marginal posterior variance of ψ, i.e. $\mathrm{var}(\psi|\mathcal{D})$, can then be estimated using a weighted average of these two quantities :

$$\hat{\mathrm{var}}^{+}(\psi|\mathcal{D}) = \frac{n-1}{n}W + \frac{1}{n}B \tag{3.5}$$

This estimator is unbiaised under stationarity. However, when convergence has not occured yet, it tends to overestimate the true posterior variance of ψ. Meanwhile, for any finite n, the within variance W should be an underestimate of the true posterior variance, because the individual sequences have not had time to range over all the target distribution and, as a result, should have less variability.

Therefore, convergence of the iterative simulation can be monitored by estimating the factor by which the scale of the current distribution for ψ might be reduced if the simulations were continued in the limit $n \to \infty$. This potential scale reduction is estimated by the $\hat{R}$ statistic :

$$\hat{R} = \sqrt{\frac{\hat{\mathrm{var}}^{+}(\psi|\mathcal{D})}{W}} \tag{3.6}$$

[Gelman et al. 2013] recommends to compute the $\hat{R}$ statistics for all quantities of interest, and to acknowledge the convergence if all statistics are sufficiently near 1 (e.g. < 1.1).

3.3.3.3 Effective sample size

Besides the need to monitor convergence, a second technical difficulty posed by MCMC methods is that the samples are typically autocorrelated within a chain. This increases the uncertainty of the estimation of posterior quantities of interest, such as means, variances, or quantiles. In order to quantify this additional uncertainty, one usually uses the concept of *effective sample size* for each parameter. The effective sample size can be defined by considering the statistical efficiency of the average of the simulations $\bar{\psi}_{..}$, as an estimate of the posterior mean, $\mathbb{E}(\psi|\mathcal{D})$ ([Gelman et al. 2013], Section 11.5), and comparing it to the efficiency that

would have been observed with chains of independent samples.

In order to do this, one uses the following asymptotic formula for the variance of the average of a correlated sequence, reusing the same notations as in Section 3.3.3.2 :

$$\lim_{n\to\infty} mn \ \mathrm{var}(\bar{\psi}_{..}) = \left(1 + 2\sum_{t=1}^{\infty} \rho_t\right) \ \mathrm{var}(\psi|\mathcal{D}) \tag{3.7}$$

where ρ_t is the autocorrelation of the sequence ψ at lag t. Then the effective sample size can be defined as :

$$n_{\mathrm{eff}} = \frac{mn}{1 + 2\sum_{t=1}^{\infty} \rho_t} \tag{3.8}$$

Note that in practice, when computing (3.8), the ρ_t need to be estimated, and the sum needs to be truncated to a finite number of terms. Finally, let us mention that [Gelman et al. 2013] considers as acceptable an effective sample size of 10 times the number of sequences.

Chapter 4

Single protein model

4.1 Introduction

In this chapter, which is part of the main body of this work, a single protein statistical model, where the data related to each protein (intensities and missingness) are modelled independently from the data belonging to the other proteins, is developed. In this model, most of the ingredients selected in Section 2.2 are introduced, at the exception of the hierarchical set-up.

This single protein model is not meant to be the target model of this work, but rather a simplified version, as a first step. By doing this, one hopes to avoid the pitfall of bringing all the complexity at once, temporarily leaving out the multi-dimensional nature of the problem. However, in a second step, one will aim at bringing back the hierarchical set-up, with the goal to benefit from borrowing information available from all protein data, in the analysis of the responses for a given protein. This will be the subject of Chapter 5.

4.2 Model description

In this section, the single protein model is specified in mathematical equations. Here, one aims at describing the model in its most generic version, without specifying explicitely the nature of the covariates in the linear regression parts. Later on, one will particularize the model for the specific case of the CPTAC benchmark dataset (see Section 2.3). Also, two equivalent formulation of the same model are introduced, a base formulation (Section 4.2.1), and a data augmented formulation (Section 4.2.2).

4.2.1 Generic model in its base formulation

Following the choices made in Section 2.2, one considers peptide level intensities, and model these jointly with data missingness. Now, assume that a particular protein k has been selected from the dataset ($k \in \{1, ...K\}$, where K is the total number of proteins). The response variables of the single protein model for protein k are then the log-2 transformed peptide intensities, for all peptides belonging to protein k. The same model will then be applied repeatedly for all k. Therefore, in what follows, the index k is dropped and is now considered as implicit.

For protein k :

- Let $\boldsymbol{y} = (y_1, \ldots, y_j, \ldots, y_N)$ be the (complete) vector of log-2 transformed peptide intensities (observed and missing);
- Let $\boldsymbol{r} = (r_1, \ldots, r_j, \ldots, r_N)$ be the vector of observation indicator variables, i.e. $r_j = 1$ if y_j is observed, and $r_j = 0$ if y_j is a missing data.

$\boldsymbol{y}$ and $\boldsymbol{r}$ are the two sets of responses in the model, and N is the total number of peptide intensities, observed or missing, belonging to protein k.

The chosen single protein model is composed of a peptide intensity part and a missingness part, as follows :

$$\begin{aligned} \boldsymbol{y}|\boldsymbol{\alpha}, \boldsymbol{\beta}, \sigma_\epsilon^2 &\sim t_\nu(X\boldsymbol{\alpha} + Z\boldsymbol{\beta}, \sigma_\epsilon^2 I_N) \\ r_j|\boldsymbol{\gamma}, \gamma^y, y_j &\sim \text{Ber}(\Phi((T\boldsymbol{\gamma})_j + \gamma^y(y_j - y_{ref}))) \end{aligned} \tag{4.1}$$

where :

- $X\boldsymbol{\alpha}$ are the fixed effect terms in the intensity regression part;
- $Z\boldsymbol{\beta}$ are the random effect terms in the intensity regression part;
- T is the model matrix for the MAR missingness coefficients;
- $\boldsymbol{\gamma}$ is the vector of MAR missingness regression coefficients;
- γ^y is the NMAR missingness regression coefficient;
- y_{ref} is a fixed reference intensity level;
- $t_\nu(\boldsymbol{\mu}, \Sigma)$ is the multivariate Student distribution with ν degrees of freedom, location vector $\boldsymbol{\mu}$, and scale matrix Σ;
- $\text{Ber}(p)$ is the Bernoulli distribution with parameter p;
- $\Phi(x)$ is the standard normal cumulative distribution function (cdf).

The chosen prior distributions are :

$$\begin{aligned} \boldsymbol{\alpha} &\sim \mathcal{N}(\boldsymbol{\mu}_\alpha, \Sigma_\alpha) \\ \boldsymbol{\beta} &\sim \mathcal{N}(0, G(\boldsymbol{\sigma_r^2})) \\ \boldsymbol{\gamma} &\sim \mathcal{N}(\boldsymbol{\mu}_\gamma, \Sigma_\gamma) \\ \gamma_y &\sim \mathcal{N}(\mu_{\gamma_y}, \sigma^2_{\gamma_y}) \\ \sigma_\epsilon^2 &\sim \text{Inv-}\chi^2(\nu_\epsilon, s_\epsilon^2) \\ \sigma_{r,i}^2 &\sim \text{Inv-}\chi^2(\nu_{r,i}, s_{r,i}^2) \end{aligned} \tag{4.2}$$

where :

- $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ is the multivariate normal distribution, with mean $\boldsymbol{\mu}$ and covariance matrix Σ;

- Inv-$\chi^2(\nu, s^2)$ is the scaled inverse chi square distribution, with ν degrees of freedom and scale parameter s;
- $G(\boldsymbol{\sigma}_r^2) = \text{blockdiag}(\sigma_{r,1}^2 I_{M^{\beta(1)}}, \ldots, \sigma_{r,R}^2 I_{M^{\beta(R)}})$, with R the number of random effects, and $M^{\beta(i)}$ the number of observed levels for random effect i.

And finally, each block of parameters is set to be a priori independent from the other blocks :

$$p(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \gamma_y, \sigma_\epsilon^2, \boldsymbol{\sigma}_r^2) = p(\boldsymbol{\alpha})\ p(\boldsymbol{\beta})\ p(\boldsymbol{\gamma})\ p(\gamma_y)\ p(\sigma_\epsilon^2)\ p(\boldsymbol{\sigma}_r^2) \tag{4.3}$$

A few remarks need to be formulated at this stage :

- In the peptide intensity regression part :
  - in order to apply a robust regression (see discussion in Section 2.2), one chooses the Student distribution as the peptide intensities distribution, and fix the degrees of freedom ν to a low integer value (typically 3 to 5)
  - in this generic model formulation, the choice of variables to consider in the fixed and random effects is left open. However, in accordance with Section 2.2, at least one random effect corresponding to a peptide effect will be introduced. Moreover, in most practical cases of application of this model, a random run effect will need to be introduced in order to distinguish biological variability from technical variability (see again 2.2). In addition, other effects could be modelled as fixed or random effects, according to the modeller's preference. However the residuals dependence structure has to correspond to a "components of variance" model, leading to the above specification for the G matrix.
- In the missingness regression part :
  - a probit regression model, instead of a logit regression model, is chosen. On the one hand, these two models are very similar, and on the other hand, probit will allow for better analytical tractibility, for example when deriving analytically some approximations of posterior distributions (e.g. in Section 4.4 and Section 4.5).
  - a fixed y_{ref} parameter is introduced, in order to improve the interpretability of the intercept γ parameter. This y_{ref} parameter is set to a common value for all protein k models. For instance, it can be taken as the overall median of observed peptide intensities in the entire dataset.
  - one opts for a relatively simple NMAR modelling, where missingness is linearly dependent on underlying intensity, only through a single γ_y parameter. This choice was made to not complexify too much the model, and therefore, there is no interaction between intensity and other variables in the missingness model.
- Regarding the priors :
  - in this single protein model, all prior distribution parameters, i.e. $\boldsymbol{\mu}_\alpha$, Σ_α, $\boldsymbol{\mu}_\gamma$, Σ_γ, μ_{γ_y}, $\sigma_{\gamma_y}^2$, ν_ϵ, s_ϵ^2, $\nu_{r,i}$, $s_{r,i}^2$, $\sigma_{r,i}^2$ are fixed and set before-hand by the modeller.
  - in case there is no a priori information available, prior distribution parameters are selected in order to have (almost) uninformative proper priors, i.e. very large variances for the prior normal distributions of α, β, γ parameters, and low positive parameters for the prior inverse chi square distributions of σ^2 parameters.

4.2.2 Alternative data augmented model formulation

In the subsequent sections of this chapter, often another, yet completely equivalent, formulation of the single protein model will be used. This alternative formulation introduces two sets of additional auxiliary variables, which in turn allow to replace both the Student and the Bernoulli distributions in (4.1), by normal distributions. This technique is called *data augmentation*, and is often used in Bayesian modelling in order to improve analytical and/or algorithmical tractability. See e.g. [Tipping and Lawrence 2003], and [Holmes and Held 2006] for application of the same technique in similar Bayesian contexts.

In this case, the new data-augmented formulation will replace Equation 4.1 by the following one :

$$\begin{aligned} \boldsymbol{y}|\boldsymbol{\alpha},\boldsymbol{\beta},\sigma_\epsilon^2,\boldsymbol{b} &\sim \mathcal{N}(X\boldsymbol{\alpha}+Z\boldsymbol{\beta},\sigma_\epsilon^2\,\mathrm{diag}(\boldsymbol{b})) \\ \boldsymbol{a}|\boldsymbol{\gamma},\gamma^y,\boldsymbol{y} &\sim \mathcal{N}(T\boldsymbol{\gamma}+\gamma^y(\boldsymbol{y}-y_{ref}),\boldsymbol{\sigma}_a^2=\mathbf{1_N}) \\ r_j &= I(a_j>0)\ \forall j:1,\ldots,N \end{aligned} \tag{4.4}$$

where :

- $\boldsymbol{b}$ is a vector of inverse observation weights in the intensity regression part;
- $\boldsymbol{a}$ is a vector of auxiliary "missingness intensity" variables, such that if a_j is highly positive (resp. negative), there is a high chance of observation (resp. missingness) of the corresponding peptide intensity j.

In terms of priors specification, (4.2) is to be completed with :

$$\boldsymbol{b} \sim I.G.(\nu/2,\nu/2) \tag{4.5}$$

where

- $I.G.(A,B)$ denotes the Inverse Gamma distribution with shape parameter A and scale parameter B,
- ν is the degrees of freedom of the Student distribution in eq.(4.1)

An interesting graphical representation of models that is sometimes used by Bayesian statistics practitioners is the probabilistic *Direct Acyclic Graph*, or *DAG*. It has the advantage of summarizing in a concise and visual way the relationships between various parameters of a model, hence facilitating the general understanding, especially of more complex (hierarchical) models. The DAG of the single protein model, in its data augmented formulation, is represented in Figure 4.1.

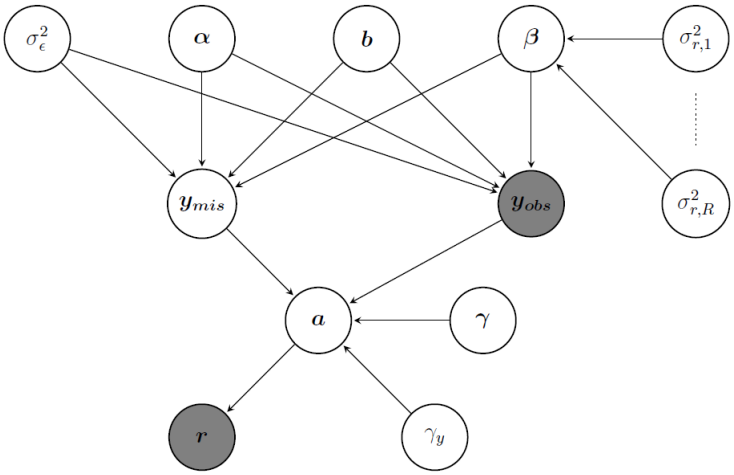


Figure 4.1: Direct Acyclic Graph (DAG) of the single protein model

In a DAG (see e.g. [Bishop 2006], Chapter 8), *nodes* represent variables or vectors of variables, while *directed edges* convey conditional dependences. The observed data components of the DAGs, here filled in light grey, are called *evidence* nodes, corresponding to observed quantities, whilst the model parameters, here not filled, are *hidden* nodes.

4.2.3 Application to the CPTAC Benchmark dataset

In order to apply the generic single protein model to the CPTAC Benchmark dataset (see 2.3), one needs to particularize it by chosing the actual set of variables included in the two regression parts of the model, appearing in eq.(4.1) or (4.4). In the model description below, the compact symbolic form of model formulas applied in the *R* language syntax, is used.

The peptide intensity model formula is :

$$y \sim condition + lab + (1|peptide) \tag{4.6}$$

Basically, eq.(4.6) means that the following regression coefficients will be introduced in the intensity regression part :

- an intercept (α_0);
- for each level of the *condition* variable, except a reference level, an α coefficient correponding to the fixed effect of that particular level, in comparison with the reference level. In the case of the CPTAC dataset, each level of the *condition* variable corresponds to a specific concentration in UPS1 protein (sample of concentration B or C compared to reference concentration level A);
- for each level of the *lab* variable, except a reference level, an α coefficient corresponding to the fixed effect of that particular level, in comparison with the reference level. As indicated in Section 2.3, the *lab* corresponds to a specific spectrometer used for protein abundance measurement, so there will be two such coefficients, since there are three labs in total.

- a β coefficient for each observed peptide effect for the protein at hand. Peptide effect is here the only random effect introduced in the model. Indeed, as mentioned in Section 2.3 the CPTAC dataset is specific in that there is no true biological replicates, only technical replicates, hence there is no need to introduce a random run effect to enable the distinction between biological and technical variability.

The missingness model formula is :

$$r \sim lab + y \tag{4.7}$$

Eq.(4.7) basically means that the following regression coefficients will be introduced regression part :

- an intercept (γ_0);
- for each level of the *lab* variable except a reference level, a γ coefficient corresponding to the effect of that particular lab on the missingness probability, in comparison with the reference level. So again, there will be two such γ coefficients.
- the NMAR coefficent γ_y, modelling the anticipated link between underlying peptide intensity, and probability of missingness (cf. Section 1.3).

Note that introducing γ coefficients related to the *lab* variables, and not to the *condition* ones can be justified from the missingness pattern present in the CPTAC dataset, as briefly described in Section 2.3.

Then, since there is no strong a priori information available on the distribution of the model parameters, weakly informative proper priors will be used. More specifically, one selects the following prior distribution parameter values, particularizing eq.(4.2) :

- Regarding the α prior distributions, one distinguishes the intercept, which is actually a log2 peptide intensity, somehow representative of the dataset, from the other α coefficients, which represent log2 fold changes between different conditions or labs. According to the knowledge at disposal, typical log2 intensities measured by mass spectrometry can vary between 0 and 30, while observed fold changes usually do not fall outside the $[-10; 10]$ interval. Based on this information, one sets[1] :
  - $\alpha_0 \sim \mathcal{N}(\mu = 15, \sigma = 5.823)$ for the intercept (log2 intensity)
  - $\alpha_i \sim \mathcal{N}(\mu = 0, \sigma = 3.882)$ for the other α_i coefficients (log2 fold changes)
  - the α's are set a priori independent
- Regarding the γ prior distributions, there is no available prior information, hence the following (almost) uninformative values are selected :
  - $\gamma_0 \sim \mathcal{N}(\mu = 0, \sigma = 10)$ for the intercept of the missingness regression part
  - $\gamma_i \sim \mathcal{N}(\mu = 0, \sigma = 10)$ for the other MAR γ coefficients
  - $\gamma_y \sim \mathcal{N}(\mu = 0, \sigma = 10)$ for the NMAR γ_y coefficient

[1] the following hyperparameter values lead to normal prior distributions having 99% of their probability mass lying in their respective target intervals

– the γ's are set a priori independent

- Regarding the inverse chi square prior distributions of the σ parameters, very low values for their respective degrees of freedom are chosen, leading to (almost) uninformative prior distributions, although still proper : $\nu_\epsilon = \nu_{r,i} = 0.002$. For the scale parameter, one chooses $s = 1$, but this is less critical as the very low ν parameter will anyway make sure the prior distribution is not informative. Also, note that this choice of hyperameters is the same as choosing $A = 0.001$ and $B = 0.001$ in an equivalent inverse gamma (A, B) parametrization to the inverse chi square parametrization used here.

And finally, the following choices are also made for the remaining (fixed) parameters in the model :

- ν, i.e. the degrees of freedom of the likelihood Student distribution, is fixed at 4;
- y_{ref}, i.e. the reference peptide intensity used in the missingness regression part, is set to the median of all observed peptide intensities, for all proteins of the CPTAC dataset, leading to $y_{ref} = 18.54$.

In terms of model size, it is interesting to note that the number of model parameters varies from protein to protein, and is mainly dependent on the number of peptides identified for the corresponding protein. Indeed, for any protein k of the CPTAC dataset, and given the regression variables selected in the model specification, one can show that the total number of model parameters $n_{param,k}$ is equal to :

$$n_{param,k} = 11 + n_{pept,k}\ (1 + 27\ \%\mathrm{NA}_k) \tag{4.8}$$

where :

- $n_{pept,k}$ is the number of identified peptides for protein k;
- $\%\mathrm{NA}_k$ is the percentage of missing peptide intensities for protein k.

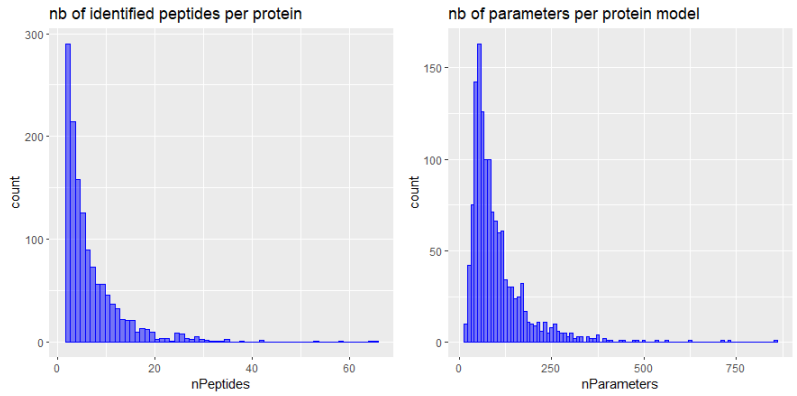


Figure 4.2: Single protein models size

Figure 4.2 provides, on the left hand side, an histogram of the number of identified peptides per protein, and, on the right hand side, an histogram of the resulting number of model

parameters per protein. Table 4.1 provides some descriptive statistics for the same two variables, as well as the number of observed intensities per protein. In total, summing over the $K = 1347$ protein models, there are 136 282 model parameters.

per protein (K=1347)	Min	Max	Mean	Median
nb identified peptides	2	65	6.8	5
nb model parameters	15	854	101	77
nb observed intensities	4	1207	100	62

Table 4.1: Some descriptive statistics related to protein models size

As a final comment of this section, please note that assessing the impact of alternative model choices, especially the choice of regression variables in the two model parts, while being of great interest, is outside the scope of this work.

4.3 Some single protein model results using Stan

4.3.1 Runs for two proteins

In this section, some results of applying the single protein model, to some subsets of data belonging to the CPTAC benchmark dataset, are provided. For that purpose, the equations of the base model formulation (4.1) and (4.2) have been implemented in a Stan model script, using the Stan domain specific language (see 3.3.1). The model script code can be found in Appendix G.1. The Stan engine can then be run, by providing the model with all input data that belong to one specific protein chosen from the CPTAC dataset. For this, the regression variables are selected and the design matrices are built according to the principles described in Section 4.2.3.

Figure 4.3 provides an extract of Stan summary output obtained when running the model on an arbitrarily chosen protein *P08311ups*, which has been identified with 10 peptides. The Stan run was configured to use 4 chains of 2000 MCMC iterations, and took 242 seconds to run on one single core of our laptop. On the one hand, Stan output provides the empirical mean, standard deviation and quantiles of the simulated marginal posterior distributions of the different model parameters. This should allow us to perform inference based on these parameter distributions. On the other hand, some element of diagnostics related to each parameter are also provided, i.e. the standard error of the mean, the effective sample size n_{eff} and the $\hat{R}$ indicator (see Section 3.3.3).

	mean	se_mean	sd	2.5%	25%	50%	75%	97.5%	n_eff	Rhat
alpha_Intercept[1]	16.25	0.03	0.63	15.00	15.85	16.26	16.64	17.49	551.10	1.01
alpha_condition6B[1]	1.62	0.00	0.10	1.42	1.55	1.61	1.68	1.82	2561.70	1.00
alpha_condition6C[1]	3.27	0.00	0.11	3.06	3.20	3.27	3.34	3.48	2120.33	1.00
alpha_labLTQ-OrbitrapO_65[1]	0.23	0.01	0.18	-0.11	0.11	0.23	0.34	0.58	720.39	1.01
alpha_labLTQ-OrbitrapW_56[1]	-0.14	0.01	0.17	-0.46	-0.25	-0.14	-0.03	0.20	741.53	1.01
beta_peptideAAFKGDSGGPLLCNNVAHGIVSYGK[1]	-0.94	0.03	0.62	-2.16	-1.32	-0.92	-0.56	0.26	565.02	1.01
beta_peptideAQEGLRPGTLCTVAGWGR[1]	1.66	0.03	0.62	0.44	1.29	1.66	2.04	2.90	566.24	1.01
beta_peptideENTQQHITAR[1]	-1.23	0.03	0.62	-2.48	-1.61	-1.22	-0.84	-0.02	565.14	1.01
beta_peptideGDSGGPLLCNNVAHGIVSYGK[1]	0.39	0.03	0.62	-0.83	0.02	0.40	0.76	1.60	561.61	1.01
beta_peptideIFGSYDPR[1]	0.30	0.03	0.62	-0.93	-0.08	0.29	0.69	1.53	555.46	1.01
beta_peptideNVNPVALPR[1]	2.00	0.03	0.61	0.81	1.62	2.00	2.38	3.19	557.64	1.01
beta_peptideRENTQQHITAR[1]	-0.37	0.03	0.63	-1.62	-0.76	-0.37	0.03	0.85	578.79	1.01
beta_peptideSSGVPPEVFTR[1]	0.53	0.03	0.61	-0.69	0.15	0.53	0.90	1.73	556.31	1.01
beta_peptideTIQNDIMLLQLSR[1]	-3.53	0.03	0.64	-4.86	-3.92	-3.52	-3.13	-2.28	574.27	1.01
beta_peptideVSSFLPWIR[1]	1.13	0.03	0.62	-0.11	0.75	1.13	1.51	2.35	565.38	1.01
gamma_Intercept[1]	-0.64	0.00	0.17	-0.98	-0.75	-0.64	-0.53	-0.32	2380.45	1.00
gamma_labLTQ-OrbitrapO_65[1]	1.36	0.00	0.23	0.91	1.20	1.37	1.51	1.81	2582.89	1.00
gamma_labLTQ-OrbitrapW_56[1]	1.72	0.00	0.25	1.25	1.54	1.71	1.88	2.21	2854.88	1.00
gammY[1]	0.37	0.00	0.05	0.27	0.33	0.37	0.41	0.48	3554.32	1.00
sigmaEps[1]	0.39	0.00	0.03	0.33	0.37	0.39	0.41	0.47	845.13	1.00
sigmaRdm_peptide[1]	1.78	0.01	0.49	1.11	1.44	1.69	2.01	2.98	2253.05	1.00
contrast_6B-6A[1]	1.62	0.00	0.10	1.42	1.55	1.61	1.68	1.82	2561.70	1.00
contrast_6C-6A[1]	3.27	0.00	0.11	3.06	3.20	3.27	3.34	3.48	2120.33	1.00

Figure 4.3: Single protein model fit on P083111ups protein

In this case, the diagnostic indicators are good: the MCMC chains have converged ($\hat{R}$ all almost equal to 1.0) and the uncertainty associated with the simulation is quite low (the mean standard errors are low, and the effective sample sizes are more than reasonable).

Note that, since this protein is part of the UPS1 protein set, it should be flagged as differentially abundant when comparing the different concentration conditions, and one can compare the true log fold changes to the estimated ones. Given the concentrations attached to the 6A, 6B and 6C conditions (resp. 0.25, 0.74 and 2.22 fmol/μl), the true log fold changes are resp. 1.57 and 3.15, which are not that far from the observed median of the corresponding marginal posterior distributions (resp. 1.61 and 3.27).

From this output, one can also notice that for this protein, the lab effect on intensities does not seem to be significant, while the peptide random effects tend to be of material size, as can be seen from the median of the posterior distributions of the β coefficients, as well as the median value of 1.69 for σ_{rdm}.

Finally, when looking at the missingness part of the model, the γ parameters are all significant. As expected, γ_{lab2} and γ_{lab3} are positive since lab2 and lab3 tend to have relatively less missing data than lab1 (see Figure 2.2). Another important point is also the positive and significant γ_Y coefficient (0.37), showing that there seems to be a positive link between intensity and data observation, as could be expected from the discussion in Section 1.3.

Now, Figure 4.4 provides the same output, this time for protein $P11938$, which is a yiest protein (i.e. not UPS1), and so the true log fold changes related to the concentration conditions should be all equal to zero. Note that this particular protein has been identified with only 3 peptides. This time the Stan run took 88 seconds on one single core of our laptop.

	mean	se_mean	sd	2.5%	25%	50%	75%	97.5%	n_eff	Rhat
alpha_Intercept[1]	15.13	1.17	2.07	11.79	13.83	14.63	16.35	19.82	3.15	1.69
alpha_condition6B[1]	0.10	0.01	0.20	-0.31	-0.03	0.10	0.23	0.51	844.32	1.02
alpha_condition6C[1]	-0.14	0.19	0.34	-0.70	-0.37	-0.22	0.05	0.63	3.19	1.62
alpha_labLTQ-OrbitrapO_65[1]	1.65	0.95	1.50	-1.96	1.12	2.10	2.58	3.70	2.48	2.28
alpha_labLTQ-OrbitrapW_56[1]	0.70	0.61	1.09	-2.14	0.29	0.89	1.35	2.46	3.13	1.69
beta_peptideDADAHDSLNDIDQLAR[1]	0.09	0.06	1.08	-1.82	-0.35	0.03	0.43	2.36	299.57	1.01
beta_peptideLQEQAQLLR[1]	0.43	0.31	1.18	-1.62	-0.21	0.47	0.95	2.84	14.04	1.11
beta_peptideSLITDEDTPTAIAR[1]	-0.45	0.59	1.40	-2.80	-1.13	-0.59	0.25	1.97	5.55	1.27
gamma_Intercept[1]	8.88	10.31	15.51	-22.79	2.90	13.95	19.53	29.14	2.27	2.84
gamma_labLTQ-OrbitrapO_65[1]	-0.83	3.69	5.95	-8.06	-4.66	-2.82	0.03	13.54	2.60	2.09
gamma_labLTQ-OrbitrapW_56[1]	-0.42	0.21	3.21	-7.08	-1.80	-0.51	0.89	6.37	224.00	1.04
gammY[1]	2.24	4.70	7.04	-14.11	0.33	4.94	6.73	10.34	2.24	3.00
sigmaEps[1]	0.41	0.01	0.10	0.26	0.34	0.39	0.46	0.63	84.84	1.05
sigmaRdm_peptide[1]	1.32	0.08	1.50	0.31	0.61	0.92	1.46	4.89	313.11	1.01
contrast_6B-6A[1]	0.10	0.01	0.20	-0.31	-0.03	0.10	0.23	0.51	844.32	1.02
contrast_6C-6A[1]	-0.14	0.19	0.34	-0.70	-0.37	-0.22	0.05	0.63	3.19	1.62

Figure 4.4: Single protein model fit on P11938 protein

For this protein, one notices that, although the log fold changes related to concentration conditions seem to be more or less properly estimated, there is however a problem. Indeed the diagnostic measures provided by Stan are not good : some $\hat{R}$ coefficients are way above the conventional threshold of 1.10 showing that the MCMC chains have not mixed. This is combined with very low effective sample sizes and very high mean standard errors for most model parameters, but this is particularly true for the missingness parameters $\boldsymbol{\gamma}$. These parameters have in turn a very high estimated posterior standard deviation, although this should be looked at with care, as the chains have not mixed. However, some additional runs with many more iterations by chain were not able to solve the issue.

Bottomline, one can interpret this issue as a model identifiability problem. It seems indeed that, for this particular protein, there is not enough information contained in the data (relating to this protein alone) to properly identify the missingness pattern, at least when a link with latent intensity is modelled. This could be linked to the small number of peptides identified for this protein, which provides much less data points than in the previous protein case. This "wild guess" will be somehow confirmed when introducing the hierarchical model in Chapter 5.

4.3.2 Discussion

Although the chosen single protein model is interesting, as it contains rich specification about the missingness pattern and its link with intensity, this model also comes with its issues. The two contrasted examples shown in Section 4.3.1 illustrates the two main issues identified so far :

1. For some of the proteins, the model seems to show identifiability issues when fit on the data relating solely to the protein at hand. As mentioned, this could be the case when the protein is identified with relatively few proteins. In Chapter 5, one will try to improve the model identifiability by "borrowing information" from the whole set of proteins in order to estimate the model for one protein.

2. At least in this specific implementation using the Stan NUTS engine, the posterior distribution simulation takes a long time to run, typically a few minutes by protein. Knowing that the CPTAC benchmark dataset contains 1347 identified proteins, this

could lead to prohibitive total run times. One also notes that in typical proteomics dataset, one can easily reach the tens of thousands proteins! Of course, this issue can be partly mitigated if one can parallelize the protein runs on a solid IT cluster with many cores, since these single protein models can be simulated separately, per protein. However, this long run time justifies in itself a further investigation on distributional approximation techniques that could potentially help to decrease it. This is the object of the remainder of the current chapter.

4.4 Laplace approximations

In order to decrease the computation time needed to simulate the joint posterior distribution obtained in the single protein model, some posterior distribution approximation techniques need to be investigated. In this section, one concentrates on Laplace approximations, which basically consist in approximating some exact joint posterior distribution by a multivariate gaussian distribution.

Note that the theoretical content of this section is mainly adapted from [Gelman et al. 2013], Section 4.1 and Section 13.5.

4.4.1 General principle

Assume that the vector of model parameters $\boldsymbol{\theta}$ has a posterior distribution $p(\boldsymbol{\theta}|\mathcal{D})$ where $\mathcal{D}$ represents the observed data. Further assume that this posterior density is unimodal and roughly symmetric. In that case, it could be convenient, for analytical tractability, to approximate it using a multivariate normal distribution. For this, one first needs to find the mode of the distribution, $\hat{\boldsymbol{\theta}}$, and this can be done using standard numerical optimization routines. Indeed, a Taylor series expansion of $\log p(\boldsymbol{\theta}|\mathcal{D})$, centered at the posterior mode, gives :

$$\log p(\boldsymbol{\theta}|\mathcal{D}) = \log p(\hat{\boldsymbol{\theta}}|\mathcal{D}) + \frac{1}{2}(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^T \Big[\frac{\partial^2}{\partial \boldsymbol{\theta}^2} \log p(\boldsymbol{\theta}|\mathcal{D})\Big]_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}} (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}) + \dots, \tag{4.9}$$

Therefore :

$$p(\boldsymbol{\theta}|\mathcal{D}) \approx \mathcal{N}(\hat{\boldsymbol{\theta}}, [\mathcal{I}(\hat{\boldsymbol{\theta}})]^{-1}), \tag{4.10}$$

where $\mathcal{I}(\boldsymbol{\theta})$ is the *observed information* :

$$\mathcal{I}(\boldsymbol{\theta}) = -\Big[\frac{\partial^2}{\partial \boldsymbol{\theta}^2} \log p(\boldsymbol{\theta}|\mathcal{D})\Big].$$

Calculating the Laplace approximation of the posterior distribution therefore involves calculating all the second order derivatives with respect to the model parameters, evaluated at the node. These in turn can be calculated analytically, when possible given the form of the exact posterior distribution, or computed using numerical approximations. Indeed, the Hessian matrix is often calculated or approximated within the optimization routines typically used to find the mode of the distribution, such that it can often be obtained as a by-product. This technique is therefore relatively straightforward. For example, Stan provides to the user

a specific run mode where the MCMC NUTS Engine is replaced by a quasi Newton mode finding algorithm (Broyden–Fletcher–Goldfarb–Shanno, a.k.a. BFGS algorithm, see [Kelley 1999]). As mentioned briefly in Section 3.3, Stan takes advantage of algorithmic differentiation in order to automatically obtain the exact first order derivatives of the posterior density. Hence, based on the same model script as used with the MCMC NUTS Engine, Stan can numerically approximate the Hessian matrix needed in the opimization algorithm and provide samples of the Laplace approximation of the joint posterior.

Some results obtained when running the Stan engine to calculate the Laplace approximation on the single protein model will be presented in Section 4.4.4.

4.4.2 Posterior approximation using conditionals and marginals

It is well known (see e.g. [Gelman et al. 2013], Section 4.1) that when the number of model parameters is high, which is the case in the single protein model, the Laplace approximation often performs poorly. However, there is a way to bring Laplace approximations a bit further. Indeed, assume that one can split the whole set of parameters $\boldsymbol{\theta}$ into two subsets $\boldsymbol{\theta} = (\boldsymbol{\psi}, \boldsymbol{\phi})$, for which one can derive the conditional posterior distribution of $\boldsymbol{\psi}$ given the other set of parameters ($\boldsymbol{\phi}$), and can easily generate samples from it (e.g. because it has the form of a standard distribution). The joint posterior distribution satisfies :

$$\begin{aligned} p(\boldsymbol{\psi}, \boldsymbol{\phi}|\mathcal{D}) &= p(\boldsymbol{\psi}|\boldsymbol{\phi}, \mathcal{D})p(\boldsymbol{\phi}|\mathcal{D}) \\ &\approx p(\boldsymbol{\psi}|\boldsymbol{\phi}, \mathcal{D})p_{approx.}(\boldsymbol{\phi}|\mathcal{D}) \end{aligned} \tag{4.11}$$

where $p_{approx.}(\boldsymbol{\phi}|\mathcal{D})$ is the Laplace approximation of $p(\boldsymbol{\phi}|\mathcal{D})$. Using equations (4.11), approximating the joint posterior now involves four steps :

1. calculating the marginal posterior distribution of $\boldsymbol{\phi}$;
2. finding its mode using an optimization routine and inferring its Laplace approximation using (4.10) ;
3. sampling from the normal approximation of the marginal distribution of $\boldsymbol{\phi}$ obtained in step 2;
4. given the samples of $\boldsymbol{\phi}$ obtained in step 3, generating samples of $\boldsymbol{\psi}$ using its posterior conditional distribution $p(\boldsymbol{\psi}|\boldsymbol{\phi}, \mathcal{D})$.

The advantage of this slightly more complex approach is that the range of possible joint posterior distribution approximations is bigger, since only the $\boldsymbol{\phi}$ parameters marginal distribution is approximated using a multivariate normal distribution. Now, the marginal distribution of $\boldsymbol{\psi}$ can have a different shape, triggered by the parametric form of the conditonal distribution of $\boldsymbol{\psi}$, given $\boldsymbol{\phi}$. However, one of the challenges is the adequate choice of the parameters split into the two sets $\boldsymbol{\phi}$ and $\boldsymbol{\psi}$. On the one hand, one should try to keep as less parameters as possible in $\boldsymbol{\phi}$ - since Laplace Approximation performs better with few parameters - and on the other hand, the conditional distribution of $\boldsymbol{\psi}$ needs to be a standard distribution, from which samples can be easily drawn.

4.4.3 Application to the single protein model

In this section, the principles described above are applied to the single protein model of Section 4.2. For technical reasons, one will choose here yet another formulation of the model, which is a kind of "hybrid" between the data augmented formulation (Section 4.2.2) and the base formulation (Section 4.2.1) whereby only the $\boldsymbol{b}$ auxiliary variables are introduced, but not the $\boldsymbol{a}$ auxiliary variables. On the one hand, the introduction of the $\boldsymbol{b}$ variables is mathematically helpful to handle the Student distribution. On the other hand, the $\boldsymbol{a}$ auxiliary variables have truncated conditional posterior distributions which are not well approximated by multivariate gaussians. Therefore the formulation of the model used in this section replaces equation (4.1) or (4.4) by a combination of these, as follows :

$$\begin{aligned} \boldsymbol{y}|\boldsymbol{\alpha},\boldsymbol{\beta},\sigma_\epsilon^2,\boldsymbol{b} &\sim \mathcal{N}(X\boldsymbol{\alpha}+Z\boldsymbol{\beta},\sigma_\epsilon^2\,\mathrm{diag}(\boldsymbol{b})) \\ r_j|\boldsymbol{\gamma},\gamma^y,y_j &\sim \mathrm{Ber}(\Phi((T\boldsymbol{\gamma})_j+\gamma^y(y_j-y_{ref}))) \\ \boldsymbol{b} &\sim \mathrm{I.G.}(\nu/2,\nu/2) \end{aligned} \tag{4.12}$$

Denote by $\boldsymbol{y}_{obs}$ (resp. $\boldsymbol{y}_{mis}$) the set of observed (resp. missing) peptide intensities, such that $\boldsymbol{y}=(\boldsymbol{y}_{obs},\boldsymbol{y}_{mis})$. Note first that the observed data are $\mathcal{D}=(\boldsymbol{y}_{obs},\boldsymbol{r})$. Then, the following parametrization choice is made :

$$\begin{aligned} \boldsymbol{\psi} &= (\boldsymbol{\alpha},\boldsymbol{\beta}) \\ \boldsymbol{\phi} &= (\boldsymbol{y}_{mis},\boldsymbol{\gamma},\gamma_y,\log\sigma_\epsilon^2,\log\boldsymbol{\sigma}_r^2,\log\boldsymbol{b}) \end{aligned} \tag{4.13}$$

In (4.13), the scale related parameters $(\sigma_\epsilon^2,\boldsymbol{\sigma}_r^2,\boldsymbol{b})$ are included in $\boldsymbol{\phi}$ after log transformation. This is because the aim is to approximate their marginal distribution using a normal distribution, and these parameters are only defined on the positive domain, hence there is hope to enhance the quality of the normal approximation by applying this log-transformation.

In a first step, one needs to derive the conditional posterior of $\boldsymbol{\psi}$:

$$\begin{aligned} p(\boldsymbol{\psi}|\boldsymbol{\phi},\mathcal{D}) &= p(\boldsymbol{\psi}|\boldsymbol{y}_{obs},\boldsymbol{y}_{mis},\sigma_\epsilon^2,\boldsymbol{\sigma}_r^2,\boldsymbol{b}) \\ &\propto p(\boldsymbol{y}|\boldsymbol{\alpha},\boldsymbol{\beta},\sigma_\epsilon^2,\boldsymbol{b})p(\boldsymbol{\alpha},\boldsymbol{\beta}|\boldsymbol{\sigma}_r^2) \end{aligned} \tag{4.14}$$

Combining (4.14) and (4.12), and taking into account the model priors for $\boldsymbol{\psi}=(\boldsymbol{\alpha},\boldsymbol{\beta})$ - see (4.2), one can show that :

$$\begin{aligned} \boldsymbol{\psi}|\boldsymbol{\phi},\mathcal{D} &\sim \mathcal{N}(\boldsymbol{\mu}(\boldsymbol{\phi}),\Sigma(\boldsymbol{\phi})),\ \text{where :} \\ \Sigma(\boldsymbol{\phi}) &= [\frac{1}{\sigma_\epsilon^2}W^T\,\mathrm{diag}(1/\boldsymbol{b})W+\Sigma_{\alpha,\beta}^{-1}(\boldsymbol{\sigma}_r^2)]^{-1} \\ \boldsymbol{\mu}(\boldsymbol{\phi}) &= \Sigma(\boldsymbol{\phi})\frac{1}{\sigma_\epsilon^2}W^T\,\mathrm{diag}(1/\boldsymbol{b})\ \boldsymbol{y}+\Sigma_{\alpha,\beta}^{-1}(\boldsymbol{\sigma}_r^2)\ \boldsymbol{\mu}_{\alpha,\beta} \end{aligned} \tag{4.15}$$

where :

$$\begin{aligned} W &= [XZ],\ \text{where [ ] refers to matrix columns binding.} \\ \Sigma_{\alpha,\beta}(\boldsymbol{\sigma}_r^2) &= \mathrm{blockdiag}(\Sigma_\alpha,G(\boldsymbol{\sigma}_r^2))\ \text{is the prior variance of } \boldsymbol{\psi} \\ \boldsymbol{\mu}_{\alpha,\beta} &= (\boldsymbol{\mu}_\alpha,\boldsymbol{\mu}_\beta=(0,\ldots,0))\ \text{is the prior mean of } \boldsymbol{\psi} \end{aligned} \tag{4.16}$$

In a second step, one needs to infer the marginal posterior distribution of $\boldsymbol{\phi}$:

$$p(\boldsymbol{\phi}|\mathcal{D}) = \frac{p(\boldsymbol{\psi}, \boldsymbol{\phi}|\mathcal{D})}{p(\boldsymbol{\psi}|\boldsymbol{\phi}, \mathcal{D})} \tag{4.17}$$

Since the left hand side of Equation (4.17) does not depend on $\boldsymbol{\psi}$, the right hand side should be valid for all possible values of $\boldsymbol{\psi}$, and in particular if one imposes $\boldsymbol{\psi} = \boldsymbol{\mu}(\boldsymbol{\phi})$. Using (4.15),

$$\left.\frac{1}{p(\boldsymbol{\psi}|\boldsymbol{\phi}, \mathcal{D})}\right|_{\boldsymbol{\psi}=\boldsymbol{\mu}(\boldsymbol{\phi})} \propto |\Sigma(\boldsymbol{\phi})|^{\frac{1}{2}} \tag{4.18}$$

Moreover, the numerator in eq.(4.17) expands to :

$$\begin{aligned} p(\boldsymbol{\psi}, \boldsymbol{\phi}|\mathcal{D}) &\propto p(\mathcal{D}|\boldsymbol{\psi}, \boldsymbol{\phi}) p(\boldsymbol{\psi}, \boldsymbol{\phi}) \\ &\propto p(\boldsymbol{r}|\boldsymbol{y}_{obs}, \boldsymbol{\psi}, \boldsymbol{\phi}) p(\boldsymbol{y}_{obs}|\boldsymbol{\psi}, \boldsymbol{\phi}) p(\boldsymbol{\psi}, \boldsymbol{\phi}) \\ &\propto p(\boldsymbol{r}|\boldsymbol{y}_{obs}, \boldsymbol{\phi}) p(\boldsymbol{y}_{obs}|\boldsymbol{\psi}, \boldsymbol{\phi}) p(\boldsymbol{\psi}, \boldsymbol{\phi}) \end{aligned} \tag{4.19}$$

since the distribution of $\boldsymbol{r}$, conditionally on $\boldsymbol{\phi}$, does not depend on $\boldsymbol{\psi}$. Now developing the last factor of (4.19) :

$$\begin{aligned} p(\boldsymbol{\psi}, \boldsymbol{\phi}) &= p(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{y}_{mis}, \boldsymbol{\gamma}, \gamma_y, \log \sigma_\epsilon^2, \log \boldsymbol{\sigma}_r^2, \log \boldsymbol{b}) \\ &\propto p(\boldsymbol{y}_{mis}|\boldsymbol{\alpha}, \boldsymbol{\beta}, \sigma_\epsilon^2, \boldsymbol{b}) p(\boldsymbol{\alpha}) p(\boldsymbol{\beta}|\boldsymbol{\sigma}_r^2) p(\boldsymbol{\sigma}_r^2) p(\sigma_\epsilon^2) p(\boldsymbol{b}) p(\boldsymbol{\gamma}, \gamma_Y) \\ &\quad \boldsymbol{\sigma}_r^2 \sigma_\epsilon^2 \boldsymbol{b} \end{aligned} \tag{4.20}$$

Bringing all together, the marginal posterior distribution of $\boldsymbol{\phi}$ is :

$$\begin{aligned} p(\boldsymbol{\phi}|\mathcal{D}) &= p(\boldsymbol{y}_{mis}, \boldsymbol{\gamma}, \gamma_y, \log \sigma_\epsilon^2, \log \boldsymbol{\sigma}_r^2, \log \boldsymbol{b}|\boldsymbol{y}_{obs}, \boldsymbol{r}) \\ &\propto p(\boldsymbol{r}|\boldsymbol{y_{obs}}, \boldsymbol{y_{mis}}, \boldsymbol{\gamma}, \gamma_Y) \left[p(\boldsymbol{y}|\boldsymbol{\alpha}, \boldsymbol{\beta}, \sigma_\epsilon^2, \boldsymbol{b}) p(\boldsymbol{\alpha}) p(\boldsymbol{\beta}|\boldsymbol{\sigma}_r^2)\right]\big|_{\boldsymbol{\psi}=\boldsymbol{\mu}(\boldsymbol{\phi})} \\ &\quad p(\boldsymbol{\sigma}_r^2) p(\sigma_\epsilon^2) p(\boldsymbol{b}) p(\boldsymbol{\gamma}, \gamma_Y) \boldsymbol{\sigma}_r^2 \sigma_\epsilon^2 \boldsymbol{b} \; |\Sigma(\boldsymbol{\phi})|^{\frac{1}{2}} \end{aligned} \tag{4.21}$$

Again, in order to find the mode and Hessian matrix of the marginal posterior distribution described in (4.21), one can take advantage of the Stan engine and its algorithmic differentiation capabilities. To implement this, a slightly different Stan script, which is provided in Appendix G.2, has been developed. Then, as indicated in Section 4.4.1, Stan can also generate a random sample of $\boldsymbol{\phi}$ values using (4.21). When this is done, in order to get a random sample of $(\boldsymbol{\alpha}, \boldsymbol{\beta})$ values, it suffices to sample from the multivariate normal distribution in (4.15).

4.4.4 Results of Laplace approximations

In this section, some results obtained while applying some Laplace approximations to the single protein model are presented. For this, the *P08311ups* protein, for which good results were obtained with the model in Section 4.3.1, was selected. Two types of Laplace approximation were implemented, i.e. Laplace approximation on all parameters at once as described in Section 4.4.1, and Laplace approximation with parameters split as exposed in Section 4.4.3. The CPU execution times were then recorded, and the approximated marginal distributions obtained for various model parameters were compared with the "exact" ones (i.e. "exact" up

to Monte Carlo errors), previously obtained with Stan NUTS engine.

In terms of execution time, the Laplace approximations are compared with the MCMC runs performed in Section 4.3.1, which provided 4,000 samples of the posterior distributions in total (excluding the warm-up part). As expected, Table 4.2 shows a dramatic improvement of execution time, as the Laplace approximation on all parameters at once (here labelled *Laplace approxi simple*) runs in less than 1 second of CPU time, while the Laplace approximation with parameters split (here labelled *Laplace approxi cond/marg*), runs in about 15 seconds. This is to be compared with the MCMC run time of more than 4 minutes.

Algorithm	Average CPU execution time (s)
Stan NUTS	258
Laplace approxi simple	0.9
Laplace approxi cond/marg	15.8

Table 4.2: Comparison of CPU execution time on one single protein (*P08311ups*) between Stan NUTS and Laplace approximations (average run time for 10 runs and $4,000$ obtained samples).

The posterior marginal distributions are compared graphically for some of the model parameters in Figure 4.5, Figure 4.6, Figure 4.7 and Figure 4.8. Note that this particular subset of parameters was selected in order to show the typical observed behaviours. The full set of graphical results can be found in Appendix C.1.

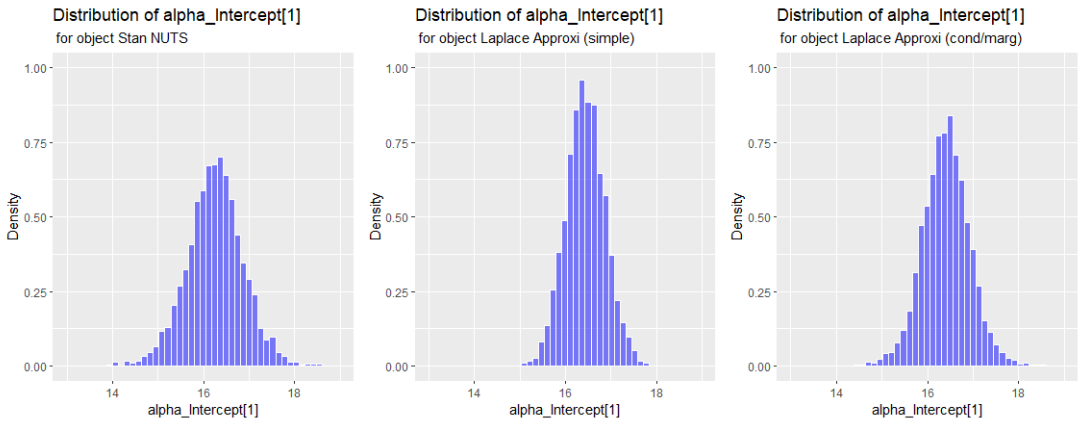


Figure 4.5: Comparison of marginal posterior distributions for $\alpha_{intercept}$

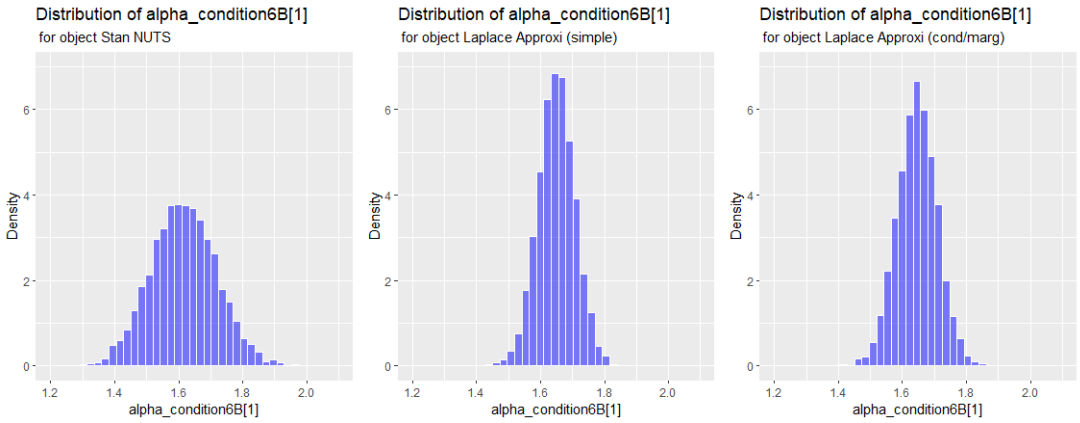


Figure 4.6: Comparison of marginal posterior distributions for α_{cond6B}

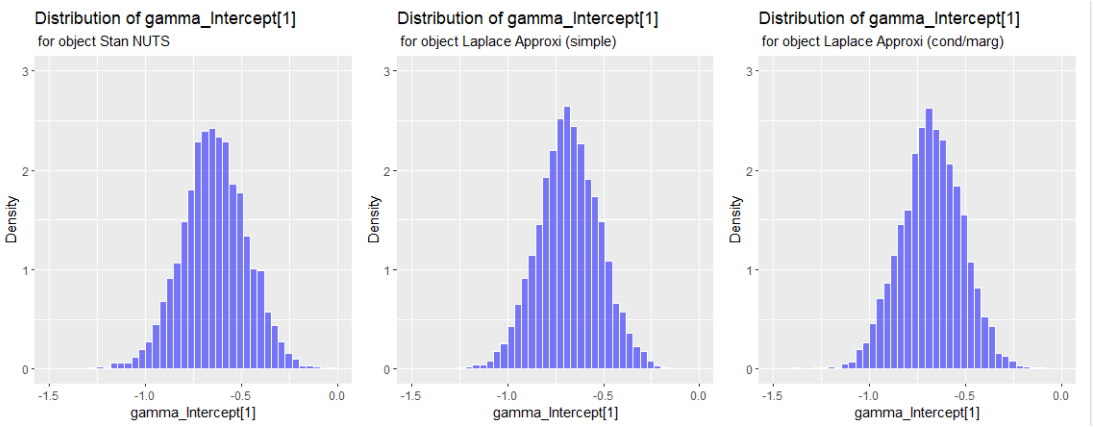


Figure 4.7: Comparison of marginal posterior distributions for $\gamma_{intercept}$

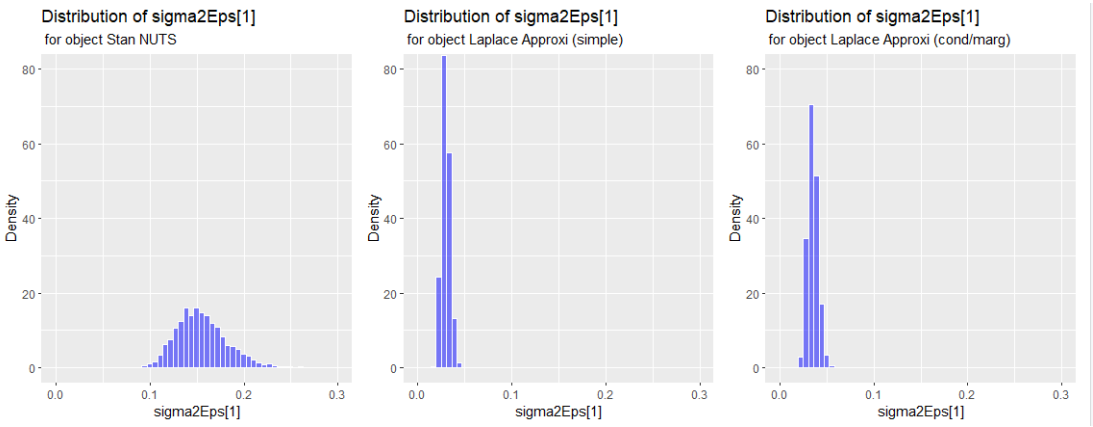


Figure 4.8: Comparison of marginal posterior distributions for σ_ϵ^2

From these graphs, one can identify the following patterns :

- In terms of mode identification of the marginal distributions, the two types of Laplace approximations seem to do a good job, since for most parameters, the mode of the approximations seems to correspond to the exact mode. There is one notable exception (see Figure 4.8) : the marginal distribution of σ_ϵ^2 (i.e. the residual variance) is completely off, and this similarly for both Laplace approximation types. This rather strange behaviour will be examined in paragraph 4.4.5. For the rest, the approximated marginal modes are OK, sometimes slightly off as in Figure 4.6, but not dramatically.

- The distributions of the γ parameters, which are the coefficients of the missingness regression part of the model, are accurately approximated by the two types of Laplace approximations. Indeed, not only the mode, but also the dispersion and the normal shape of the distributions remarkably match. This is shown in Figure 4.7, as well as the other γ graphs in Appendix C.1.

- Regarding the remaining parameters, one can identify a recurring pattern, in that the approximated distribution tend to underestimate the exact dispersion. This underestimation can be sometimes severe (e.g. Figure C.9), and unfortunately also shows up for the main parameters of inferential interest, i.e. the α's.

- Comparing now the two types of Laplace approximation, one can identify that the more evolved one, applying a split between the conditionals and marginals of two sets of parameters, does only slightly better in estimating the dispersion of a handful of model parameters posterior distributions (for instance, the $\alpha_{intercept}$ in Figure 4.5, or the variance of the peptide random effect $\sigma^2_{rdm,peptide}$ in Figure C.8).

Bottomline, these graphs suggest that the Laplace approximations, as tried in this section, although running very quickly in terms of CPU time, will not be the panacea for what inference is concerned. Indeed, while these approximations would be suitable to find good point estimates for most model parameters (at the notable exception of σ_ϵ^2), they would lead to a variability underestimation for o.a. the $\boldsymbol{\alpha}$ parameters, hence to an excess of false positive rate when identifying differentially abundant proteins. However, one needs to emphasize that these conclusions might be highly dependent on the particular choices made for the model formulation as well as for the parameters split (4.13). In particular, looking at the form of the posterior of $\boldsymbol{\psi}$ obtained in (4.15), the $\boldsymbol{\psi}$ posterior variability underestimation is likely to be linked to the sharp underestimation of σ_ϵ^2. As will be advocated in Section 4.4.5, this in turn might be caused by the presence of the missing data $\boldsymbol{y}_{mis}$ in the $\boldsymbol{\phi}$ set of parameters. This would mean that such a parametrization would fail to properly account for the uncertainty present in the missing data, hence leading to the underestimation of the α parameters dispersion.

Therefore, in order to improve the results, some attempts were made to develop different ways to apply Laplace approximation using the principles exposed in Section 4.4.2. In particular other ways to split the parameters into the $\boldsymbol{\psi}$ and $\boldsymbol{\phi}$ sets were tried, also using the data augmentated formulation of the model - see eq.(4.4). Unfortunately, these attempts did not work out. On the one hand, these attempts sometimes lead to intractable conditional posterior distributions for $\boldsymbol{\psi}$, or from which it was impossible to generate samples in an acceptable execution time. On the other hand, they could also sometimes lead to $\boldsymbol{\phi}$ parameters for which the normal approximation was behaving quite poorly. All in all, it was not possible

to obtain Laplace approximations performing better than those shown in the previous graphs, while running in improved execution times, compared to the Stan NUTS engine.

4.4.5 Investigating the mode issue for σ_ϵ^2

Before closing this section on Laplace approximations, a brief digression is made. Indeed, in Section 4.4.4, it was reported that the implemented Laplace distribution approximations produce a posterior mode for σ_ϵ^2 that is very different from the true mode (sharp underestimation). Here one tries to shed some light on this strange phenomenon, which was at first attributed to an implementation bug. In fact, this is probably due to the introduction of missing data in the model (4.1), as part of the $\boldsymbol{\psi}$ set of parameters - see (4.13).

To show this, consider the following simpler model, where there is one single vector of intensity responses, $\boldsymbol{y}$, which contains N_{obs} observations. The intensities are modelled as i.i.d., normal distributions with a common location parameter μ:

$$y_i \sim \mathcal{N}(\mu, \sigma^2) \text{ with } p(\mu, \sigma^2) \propto 1 \tag{4.22}$$

Straightforward analytical derivation of the mode of the joint posterior of (μ, σ^2) gives (see Appendix D) :

$$\begin{aligned} \hat{\mu} &= \bar{\boldsymbol{y}} \\ \hat{\sigma}^2 &= \frac{||\boldsymbol{y}_{obs} - \hat{\mu}||^2}{N} \end{aligned} \tag{4.23}$$

In (4.23), one finds back the familiar expressions of the maximum likelihood estimators of μ and σ^2 for a normal distribution, as obtained in a classical frequentist setting. However, if one now introduces, here in a somewhat artifical way, some missing data responses in the $\boldsymbol{y}$ vector, so that there are now N_{obs} observed responses, and N_{mis} missing ones :

$$y_i \sim \mathcal{N}(\mu, \sigma^2) \text{ with } \boldsymbol{y} = (\boldsymbol{y}_{obs}, \boldsymbol{y}_{mis}) \text{ and } p(\mu, \sigma^2, \boldsymbol{y}_{mis}) \propto 1 \tag{4.24}$$

the analytical derivation of the mode of the joint posterior of $(\mu, \sigma^2, \boldsymbol{y_{mis}})$ now gives (see again Appendix D) :

$$\begin{aligned} \hat{\mu} &= \bar{\boldsymbol{y}}_{obs} \\ \hat{y}_{mis,i} &= \hat{\mu} \ \forall i : 1, \ldots, N_{mis} \\ \hat{\sigma}^2 &= \frac{||\boldsymbol{y}_{obs} - \hat{\mu}||^2}{N_{obs} + N_{mis}} \end{aligned} \tag{4.25}$$

However, when calculating the mode of the marginal distribution of (μ, σ^2) (see Appendix D), one obtains :

$$\begin{aligned} \hat{\mu} &= \bar{\boldsymbol{y}}_{obs} \\ \hat{\sigma}^2 &= \frac{||\boldsymbol{y}_{obs} - \hat{\mu}||^2}{N_{obs}} \end{aligned} \tag{4.26}$$

Equations (4.25) and (4.26) have important implications :

- The joint mode value for σ^2 is smaller than the marginal mode value. If $N_{mis} \approx N_{obs}$, it is about twice smaller.

- Therefore, when calculating the Laplace approximation for the whole set of parameters in the model (4.24), one is approximating the joint posterior distribution by a multivariate normal, of which the mode corresponds to (4.25). However, given the properties of multinormal distributions, this means that the mode of the approximate *marginal* distribution of σ^2 will *also* correspond to the value in formula (4.25) which is about twice smaller than the true marginal mode (4.26)! In this simple model, this phenomonon clearly appears because of the addition of the missing data $\boldsymbol{y}_{mis}$ as new dimensions in the joint posterior distribution of the model parameters.

Of course, (4.24) is a dummy model, as there is actually little point in introducing missing responses in this simple case. However, one finds back the same kind of behaviour in the more complex single protein model, where the introduction of missing responses does make sense. Therefore, it could well be that the mode mismatch phenomenon observed in Section 4.4.4 could be linked to similar distributional characteristics as those described here, although the development above, done in a much simpler context, cannot be considered as a formal proof.

4.5 Variational Bayes approximation

In this section, another type of posterior distributional approximation, namely *Mean Field Variational Bayes* (MFVB) approximations, is investigated.

4.5.1 Elements of MFVB inference

Mean Field Variational Bayes is a family of parameter estimation techniques that was developed in the late 1990's in the field of Computer Science and Machine Learning. They were applied to statistical inference a bit later, in the late 2000's. In this section, some elements explaining the main ideas behind MFVB are introduced. These are adapted from [Faes, Ormerod, and Wand 2011]. More detailed expositions targetting statistician readership can be found e.g. in [Ormerod and Wand 2010] and [Blei, Kucukelbir, and McAuliffe 2017]. A thorough mathematical description can also be found in [Coulibaly 2019].

Consistently with Section 4.4, consider a Bayesian model with a vector of parameters $\boldsymbol{\theta} \in \Theta$ having a posterior distribution $p(\boldsymbol{\theta}|\mathcal{D})$ where $\mathcal{D}$ represents the observed data. Further assume that $\mathcal{D}$ and $\boldsymbol{\theta}$ are continuous random vectors. Let q be an arbitrary density function over Θ. Then, one can show ([Ormerod and Wand 2010]) that the log of the marginal likelihood $\log p(\mathcal{D})$ satisfies :

$$\log p(\mathcal{D}) = \log \underline{\mathrm{p}}(\mathcal{D}; q) + KL(p||q) \tag{4.27}$$

where

$$\underline{\mathrm{p}}(\mathcal{D}; q) \equiv \exp \int_{\Theta} q(\boldsymbol{\theta}) \log \left\{ \frac{p(\mathcal{D}, \boldsymbol{\theta})}{q(\boldsymbol{\theta})} \right\} d\boldsymbol{\theta} \tag{4.28}$$

and

$$KL(p||q) = q(\boldsymbol{\theta}) \log \Big(\frac{q(\boldsymbol{\theta})}{p(\boldsymbol{\theta}|\mathcal{D}} \Big) d\boldsymbol{\theta} \tag{4.29}$$

The latter positive quantity is known as the *Kullback-Leibler divergence* between distributions $p(\boldsymbol{\theta}|\mathcal{D})$ and $q(\boldsymbol{\theta})$. It is minimized when the two distributions are equal :

$$q(\boldsymbol{\theta}) = q_{exact}(\boldsymbol{\theta}) = p(\boldsymbol{\theta}|\mathcal{D}),$$

the exact posterior density function. However, since the latter is intractable for most models of practical interest, the idea behind Variational Bayes approximations is to substitute it by an approximate distribution q, while placing restrictions on the form of q to achieve tractability. In particular, in the case of *Mean Field* Variational Bayes, product density restrictions are imposed, as follows :

$$q(\boldsymbol{\theta}) = \prod_{i=1}^{M} q_i(\boldsymbol{\theta}_i) \text{ for some partition } \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_M\} \text{ of } \boldsymbol{\theta}. \tag{4.30}$$

Under this restriction, the optimal densities, with respect to minimum Kullback-Leibler divergence, can be shown to satisfy ([Ormerod and Wand 2010]) :

$$q_i^* \propto \exp\{\mathbb{E}_{-\boldsymbol{\theta}_i} \log p(\boldsymbol{\theta}_i|\text{rest})\},\ 1 \leq i \leq M, \tag{4.31}$$

where $\mathbb{E}_{-\boldsymbol{\theta}_i}$ denotes expectation with respect to the density $\prod_{j \neq i} q_j(\boldsymbol{\theta}_j)$, and

$$\text{rest} \equiv \{\mathcal{D}, \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{i-1}, \boldsymbol{\theta}_{i+1}, \ldots, \boldsymbol{\theta}_M\}$$

is the set containing the rest of the random vectors in the model, apart from $\boldsymbol{\theta}_i$. Note that the densities $p(\boldsymbol{\theta}_i|\text{rest})$ are known in the Bayesian literature as the *full conditionals*.

In practice, the set of equations (4.31) gives rise to an iterative algorithm scheme, as will be illustrated in Section 4.5.2.

4.5.2 Application of MFVB to the single protein model

Now, the MFVB principles described before are applied on the single protein model. Throughout this section, and for analytical tractability reasons, one uses the data augmented formulation of the model described in Section 4.2.2, illustrated as a Direct Acyclic Graph in Figure 4.1. Basically, working out the MFVB algorithm for a particular Bayesian model involves the following steps :

1. specifying a product density restriction, implied from the choice of a specific partition of $\boldsymbol{\theta}$, cf. (4.30);
2. deriving the full conditionals $p(\boldsymbol{\theta}_i|\text{rest})$, for all subset of parameters $\boldsymbol{\theta}_i$;
3. deriving the formulas of the approximated marginal distributions, using (4.31);
4. implying an iterative algorithm, in order to reach global convergence of the parameters of all q_i distributions derived in step 3.

Step 1

For the generic single protein model, the following product density restriction is chosen :

$$q(\boldsymbol{\alpha}, \boldsymbol{\beta}, \sigma_\epsilon^2, \boldsymbol{\sigma}_r^2, \boldsymbol{b}, \boldsymbol{y}_{mis}, \boldsymbol{a}, \boldsymbol{\gamma}, \gamma_y) = q(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{y}_{mis})\ q(\sigma_\epsilon^2)\ q(\boldsymbol{\sigma}_r^2)\ q(\boldsymbol{b})\ q(\boldsymbol{a})\ q(\boldsymbol{\gamma}, \gamma_y) \tag{4.32}$$

The choice of (4.32) might seem rather arbitrary. However it is the result of a trade-off between analytical tractability of steps 2 and 3, and the wilingness to come up with an accurate approximation for the posterior density, hence avoiding too stringent density restriction. On the one hand, in this specific case, it appeared crucial to keep the missing data parameters $\boldsymbol{y}_{mis}$ in a common joint q density with the fixed and random effects $(\boldsymbol{\alpha}, \boldsymbol{\beta})$ in order not to inject too much information in the approximated density of $(\boldsymbol{\alpha}, \boldsymbol{\beta})$. On the other hand, the discontinuity introduced by the relation between the $\boldsymbol{r}$ data and the $\boldsymbol{a}$ auxiliary variables (see (4.4)), for example, did not allow to mix the missingness parameters $\boldsymbol{a}, \boldsymbol{\gamma}, \gamma_y$ with the other parameters in a common joint q density.

Steps 2 and 3

Calculating the form of the full conditionals of all the parameter sets in (4.32) and implying the q^* distributions using (4.31) involves somewhat tedious mathematical derivations, although straightforward in the applied principles. These lead to optimal approximated densities $q^*(\cdot)$, such that :

$$\begin{aligned}
(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{y}_{mis}) &\overset{q^*}{\sim} \mathcal{N}(\boldsymbol{\mu}_{q^*(\alpha,\beta,y_{mis})}, \Sigma_{q^*(\alpha,\beta,y_{mis})}) \\
(\boldsymbol{\gamma}, \gamma_y) &\overset{q^*}{\sim} \mathcal{N}(\boldsymbol{\mu}_{q^*(\gamma,\gamma_y)}, \Sigma_{q^*(\gamma,\gamma_y)}) \\
q^*(\boldsymbol{a}) &= \text{product of } N \text{ truncated univariate normal densities } q^*(a_i), \text{ such that} \\
a_i &\overset{q^*}{\sim} \left\{ \begin{array}{l} \mathcal{TN}^+(\mu_{a,i}^*, \sigma_{a,i}^2 = 1) \text{ when } r_i = 1 \\ \mathcal{TN}^-(\mu_{a,i}^*, \sigma_{a,i}^2 = 1) \text{ when } r_i = 0 \end{array} \right\} \\
\sigma_\epsilon^2 &\overset{q^*}{\sim} I.G.(A_{q^*(\sigma_\epsilon^2)}, B_{q^*(\sigma_\epsilon^2)}) \\
q^*(\boldsymbol{\sigma}_r^2) &= \text{product of } R \text{ univariate inverse gamma densities } q^*(\sigma_{r,i}^2), \text{ such that} \\
\sigma_{r,i}^2 &\overset{q^*}{\sim} I.G.(A_{q^*(\sigma_{r,i}^2)}, B_{q^*(\sigma_{r,i}^2)}) \\
q^*(\boldsymbol{b}) &= \text{product of } N \text{ univariate inverse gamma densities } q^*(b_i), \text{ such that} \\
b_i &\overset{q^*}{\sim} I.G.(A_{q^*(b_i)}, B_{q^*(b_i)})
\end{aligned} \tag{4.33}$$

The expressions of the various distribution parameters introduced in (4.33) can be found in Appendix E.1.

Step 4

Given the expressions obtained in the previous steps, the optimal parameters of the approximated densities may be obtained iteratively, using Algorithm 2. Most of the mathematical notations used there come either from the single protein model specification (Section 4.2.2), or from eq.(4.33). Some additional auxiliary quantities were also defined in 4.16.

Algorithm 2 Iterative scheme for obtaining MFVB approximated densities for the single protein model

1: Initialize: $\begin{cases} \sigma^2_{q(\gamma_y)} = 1 \ ; \ \mu_{q(\gamma_y)} = 0 \ ; \ \boldsymbol{\mu}_a = 0_N \ ; \ \boldsymbol{\mu}_{q(a)} = 2(2\boldsymbol{r}-1)\phi(0) \ ; \\ \mathbb{E}_q[\frac{1}{\boldsymbol{b}}] = 1_N \ ; \ \mathbb{E}_q[\frac{1}{\sigma^2_\epsilon}] = 1 \ ; \ \mathbb{E}_q[\frac{1}{\boldsymbol{\sigma}^2_r}] = 1_R \end{cases}$

2: **repeat**

3: $\mathbb{E}_q[D] \leftarrow \text{diag}(\mathbb{E}_q[\frac{1}{\boldsymbol{b}}]) \ ; \ \mathbb{E}_q[D_{obs}] \leftarrow diag(\mathbb{E}_q[\frac{1}{\boldsymbol{b}_{obs}}]) \ ; \ \mathbb{E}_q[D_{mis}] \leftarrow \text{diag}(\mathbb{E}_q[\frac{1}{\boldsymbol{b}_{mis}}])$

4: $\mathbb{E}_q[\Sigma^{-1}_{\alpha\beta}(\boldsymbol{\sigma}^2_r)] \leftarrow \begin{bmatrix} \Sigma^{-1}_\alpha & 0 \\ 0 & \text{blockdiag}\left(\mathbb{E}_q[\frac{1}{\sigma^2_{r,1}}]I_{M^{\beta(1)}}, \ldots, \mathbb{E}_q[\frac{1}{\sigma^2_{r,R}}]I_{M^{\beta(R)}}\right) \end{bmatrix}$

5: $\Sigma_{q(\alpha,\beta,y_{mis})} \leftarrow \begin{bmatrix} \mathbb{E}_q[\frac{1}{\sigma^2_\epsilon}]W^T\mathbb{E}_q[D]W + \mathbb{E}_q[\Sigma^{-1}_{\alpha\beta}(\boldsymbol{\sigma}^2_r)] & -\mathbb{E}_q[\frac{1}{\sigma^2_\epsilon}]W^T_{mis}\mathbb{E}_q[D_{mis}] \\ -\mathbb{E}_q[\frac{1}{\sigma^2_\epsilon}]\mathbb{E}_q[D_{mis}]W_{mis} & (\sigma^2_{q(\gamma_y)} + \mu^2_{q(\gamma_y)})I_{N_{mis}} + \mathbb{E}_q[\frac{1}{\sigma^2_\epsilon}]\mathbb{E}_q[D_{mis}] \end{bmatrix}^{-1}$

6: $\boldsymbol{\mu}_{q(\alpha,\beta,y_{mis})} \leftarrow \Sigma_{q(\alpha,\beta,y_{mis})} \begin{bmatrix} \mathbb{E}_q[\frac{1}{\sigma^2_\epsilon}]W^T_{obs}\mathbb{E}_q[D_{obs}]\boldsymbol{y}_{obs} + \mathbb{E}_q[\Sigma^{-1}_{\alpha\beta}(\boldsymbol{\sigma}^2_r)]\boldsymbol{\mu}_{\alpha\beta} \\ \mu_{q(\gamma_y)}(\boldsymbol{\mu}_{q(a)} - T_{mis}\boldsymbol{\mu}_{q(\gamma)} + \mu_{q(\gamma_y)}y_{ref}) \end{bmatrix}$

7: $\Sigma_{q(\gamma,\gamma_y)} \leftarrow \left\{[T\ (\boldsymbol{\mu}_{q(y)} - y_{ref})]^T[T\ (\boldsymbol{\mu}_{q(y)} - y_{ref})] + \begin{bmatrix} 0 & 0 \\ 0 & \text{tr}(\Sigma_{q(y_{mis})}) \end{bmatrix} + \Sigma^{-1}_{\gamma\gamma_y}\right\}^{-1}$

8: $\boldsymbol{\mu}_{q(\gamma,\gamma_y)} \leftarrow \Sigma_{q(\gamma,\gamma_y)}\left([T\ (\boldsymbol{\mu}_{q(y)} - y_{ref})]\boldsymbol{\mu}_{q(a)} + \Sigma^{-1}_{\gamma\gamma_y}\boldsymbol{\mu}_{\gamma\gamma_y}\right)$

9: $\boldsymbol{\mu}_a \leftarrow T\boldsymbol{\mu}_{q(\gamma)} + \mu_{q(\gamma_y)}(\boldsymbol{\mu}_{q(y)} - y_{ref})$

10: $\boldsymbol{\mu}_{q(a)} \leftarrow \boldsymbol{\mu}_a + \frac{(2\boldsymbol{r}-1)\odot\phi(\boldsymbol{\mu}_a)}{\Phi\{(2\boldsymbol{r}-1)\odot\boldsymbol{\mu}_a\}}$

11: $B_{q(\sigma^2_\epsilon)} \leftarrow \frac{\nu_\epsilon s^2_\epsilon}{2} + \frac{1}{2}\left\{(\boldsymbol{\mu}_{q(y)} - W\boldsymbol{\mu}_{q(\alpha,\beta)})^T\mathbb{E}_q[D](\boldsymbol{\mu}_{q(y)} - W\boldsymbol{\mu}_{q(\alpha,\beta)}) + \text{tr}\left(\mathbb{E}_q[D]\begin{bmatrix} -W_{obs} & 0 \\ -W_{mis} & I_{N_{mis}} \end{bmatrix}\Sigma_{q(\alpha,\beta,y)}\begin{bmatrix} -W_{obs} & 0 \\ -W_{mis} & I_{N_{mis}} \end{bmatrix}^T\right)\right\}$

12: $\mathbb{E}_q[\frac{1}{\sigma^2_\epsilon}] \leftarrow (\frac{\nu_\epsilon}{2} + \frac{N}{2})/B_{q(\sigma^2_\epsilon)}$

13: **for all** $i \in \{1, \ldots, R\}$ **do**

14: $B_{q(\sigma^2_{r,i})} \leftarrow \frac{\nu_{r,i}s^2_{r,i}}{2} + \frac{1}{2}\left\{(\boldsymbol{\mu}^{(r,i)}_{q(\alpha,\beta)})^T\boldsymbol{\mu}^{(r,i)}_{q(\alpha,\beta)} + \text{tr}(\Sigma^{(r,i)}_{q(\alpha,\beta)})\right\}$

15: $\mathbb{E}_q[\frac{1}{\sigma^2_{r,i}}] \leftarrow (\frac{\nu_{r,i}}{2} + \frac{M^{\beta(i)}}{2})/B_{q(\sigma^2_{r,i})}$

16: **end for**

17: $B_{q(b)} \leftarrow \frac{\nu}{2} + \frac{1}{2}\mathbb{E}_q[\frac{1}{\sigma^2_\epsilon}]\left\{(\boldsymbol{\mu}_{q(y)} - W\boldsymbol{\mu}_{q(\alpha,\beta)}) \odot (\boldsymbol{\mu}_{q(y)} - W\boldsymbol{\mu}_{q(\alpha,\beta)}) + \text{diag}([-W\ I_N]\ \Sigma_{q(\alpha,\beta,y)}\ [-W\ I_N]^T)\right\}$

18: $\mathbb{E}_q[\frac{1}{\boldsymbol{b}}] \leftarrow (\frac{\nu}{2} + \frac{1}{2})/B_{q(b)}$

19: **until** convergence, i.e. ELBO(q) converges to a maximum

Note that the initialization step needs the choice of some initial values for at least some of the parameters. Since the algorithm is only guaranteed to converge to a local optimum (see Section 4.5.3), some of these choices can be amended, such that, when running different trials with different initial values, there is more chance to attain a global optimum.

4.5.3 Convergence monitoring of the MFVB algoritm

Convergence of algorithm 2 needs to be monitored. Fortunately, there is no need to control the convergence of each single parameter in (4.33) separately. Indeed one can monitor a single scalar quantity called the *ELBO*, i.e. *Evidence Lower BOund*, which is defined as follows (Blei, Kucukelbir, and McAuliffe 2017) :

$$\text{ELBO}(q) = \mathbb{E}_q[\log(p(\boldsymbol{\theta}, \mathcal{D}))] - \mathbb{E}_q[\log(q(\boldsymbol{\theta}))] \tag{4.34}$$

One can show (Blei, Kucukelbir, and McAuliffe 2017) that this quantity is equal to the opposite of the Kullback-Leibler divergence - defined in (4.29) - plus $\log(p(\mathcal{D}))$, the latter being a constant with respect to q. Therefore, in order to reach convergence, and minimize the KL divergence, ELBO(q) should reach a maximum.

Moreover (see Gelman et al. 2013 Section 13.7), each iteration of the MFVB iterative algorithm can only increase the ELBO. The algorithm is therefore always guaranteed to converge to at least a local optimum.

Deriving the analytical expression of the ELBO for the single protein model is a complicated mathematical derivation task, which leads to the detailed formula found in Appendix E.2.

4.5.4 Results of MFVB approximation

In this section, some results obtained with the MFVB approximation, as described in Section 4.5.2, are provided. For this, Algorithm 2 was implemented in R and was run on the data pertaining to again the *P08311ups* protein. After having checked the good convergence of the algorithm (ELBO gently reaching a maximum, see Figure 4.9), the CPU execution times were recorded (Table 4.3). Additionally, the approximated marginal distributions obtained with MFVB were compared with the "exact" ones, previously obtained with Stan NUTS engine.

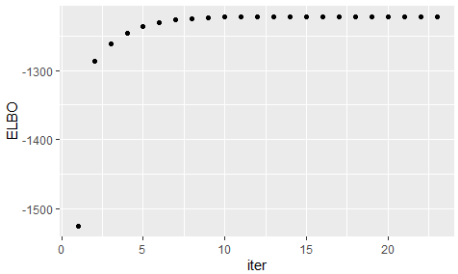


Figure 4.9: Convergence of the ELBO as a function of iteration number

Algorithm	Average CPU execution time (s)
Stan NUTS	258
MFVB approxi	3.4

Table 4.3: Comparison of CPU execution time on one single protein (*P08311ups*) between Stan NUTS and MFVB approximation (average run time for 10 runs and 4,000 obtained samples).

As Table 4.3 shows, the MFVB iterative algorithm is extremely efficient, as it takes only a handful of seconds to run, compared with the MCMC run time of more than 4 minutes.

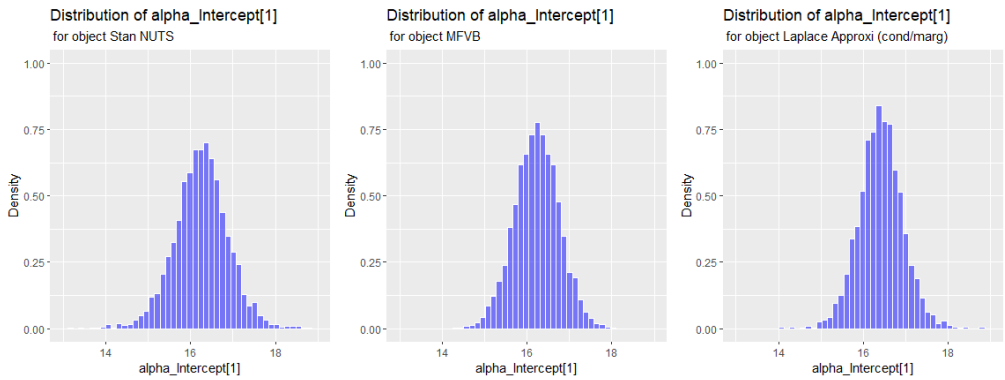


Figure 4.10: Comparison of marginal posterior distributions for $\alpha_{intercept}$

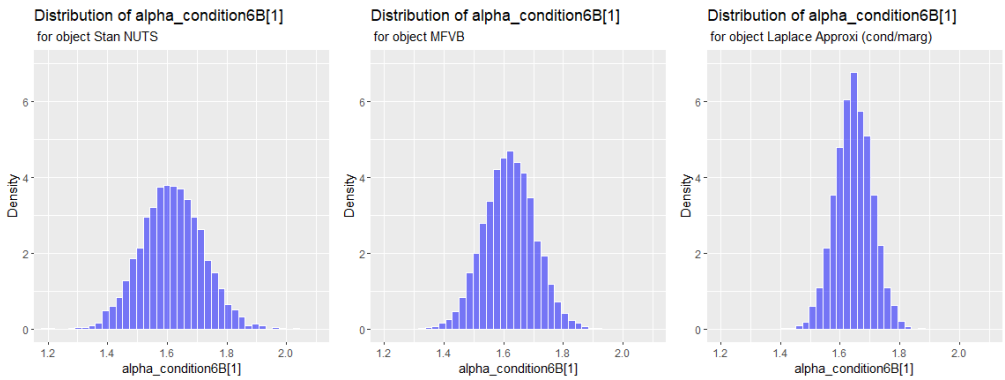


Figure 4.11: Comparison of marginal posterior distributions for α_{cond6B}

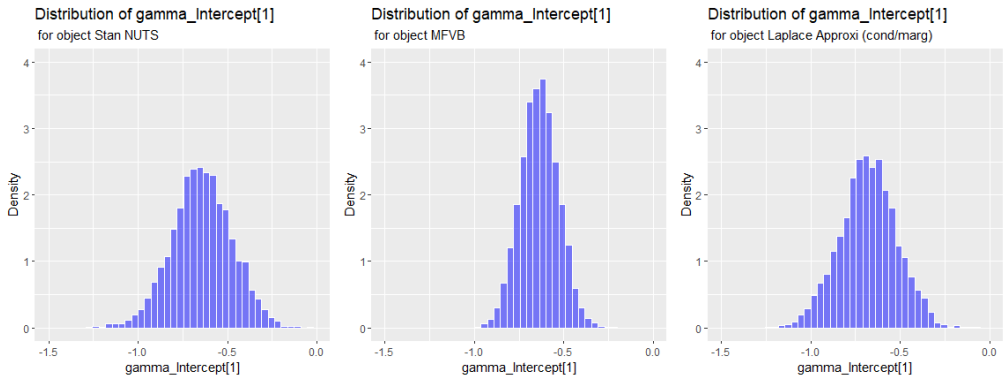


Figure 4.12: Comparison of marginal posterior distributions for $\gamma_{intercept}$

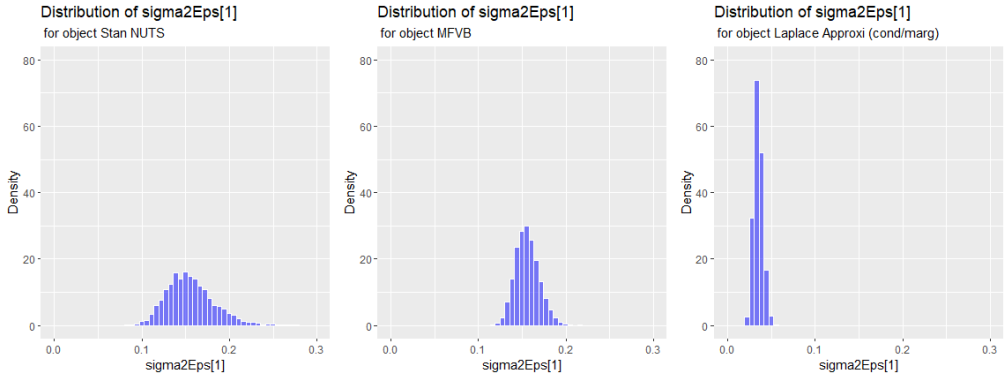


Figure 4.13: Comparison of marginal posterior distributions for σ_ϵ^2

As for Laplace approximations, the posterior marginal distributions comparisons are done in Figure 4.10, Figure 4.11, Figure 4.12 and Figure 4.13, where the previously obtained results with the Laplace approximation (parameters split version) are also included. The full set of graphical results can be found in Appendix C.2. From these graphs, one can make the following statements :

- The MFVB approximation is very good at identifying correctly the mode of the posterior distributions of all parameters. Contrary to the Laplace approximations calculated before, there is no noticeable mode biais for the α parameters. Moreover, the mode mismatch issue related to the σ_ϵ^2, discussed in Section 4.4.5, has completely vanished.

- However, when looking at the dispersion of the approximated distribution, one can find a similar issue - although it seems less severe - as the one observed with the Laplace approximations. Indeed, it seems that the MFVB approximated distribution tend to systematically underestimate the dispersion of the parameters distributions. And, as a side note, this phenomenon also appears here for the γ type parameters (see Figure 4.7), while it was not the case with the Laplace approximations.

Table 4.4 displays the mean and std deviation of the alpha coefficients, comparing the exact (Stan MCMC) and MFVB approximated posterior distributions. Recall that these α

parameters are typically the most important ones for inference, as all the others could be considered as nuisance parameters. These numbers confirm the observations made from the graphs.

α param	Stan mean	MFVB mean	Stan sd	MFVB sd
Intercept	16.25	16.23	0.63	0.52
cond6B	1.62	1.62	0.10	0.09
cond6C	3.27	3.27	0.11	0.09
lab2	0.23	0.23	0.18	0.12
lab3	-0.14	-0.13	0.17	0.11

Table 4.4: Comparison of means and standard deviations of main α posterior distributions.

Actually, this tendency of MFVB approximations to underestimate the true dispersion is well described in the literature ([Rue, Martino, and Chopin 2009], [Gelman et al. 2013], Section 13.7). It seems this issue is less critical when sample sizes are big enough. However, this is typically not the case for mass spectrometry proteomics data, as mentioned in Section 1.3.

Bottomline, this means that MFVB approximations, although very quick and accurate for posterior distribution mode finding, may not be used directly for inference in the specific use case that is addressed in this work. Indeed, this would, as with Laplace approximations, lead to a variability underestimation for o.a. the α parameters, hence to an excess of false positive rate when identifying differentially abundant proteins.

Chapter 5

Hierarchical model

5.1 Motivation - from the single protein model to a hierarchical model

This chapter introduces an extension of the single protein model, where the data related to all proteins (intensities and missingness) are modelled together, in a hierarchical set-up. This will allow, when fitting the model to the data pertaining to one protein, to *borrow* information from the data relating to the other proteins. When doing this, it is implicitly assumed that the protein data belonging to a dataset obtained from an experiment, do belong to the same population, sharing some similarities, which could indeed make sense. This idea has been introduced in the early days of omics statistics, e.g. in the Limma package (see Section 2.1 and [Smyth 2004]). When information is borrowed from the whole dataset, one actually contraints the set of model parameters for one protein, to preferrably lie within plausible values, given the data observed for the population of all proteins at hand. It is actually very similar to providing informative priors, except that the prior information is obtained from the dataset itself.

When running the single protein model on two randomly selected proteins, in Section 4.3.1, a potential model identifiability issue was found. Then the assumption was made that this could be due to a lack of information present in the corresponding set of peptide intensities, for the proteins identified with only a few peptides. Here, additional insight is provided into this identifiability issue. In the following, multiple runs of the single protein model are performed on the data of each protein, one by one. However, this is not done using the Stan NUTS engine, but using the MFVB approximation implemention implemented in Section 4.5, for the following reasons :

- firstly, as was shown in Section 4.5.4, the CPU time to run a single protein model is about 50 times shorter for the MFVB approximation, than for the Stan exact simulation;
- secondly, as was demonstrated in Section 4.3.1, when the presumed model identifiability pops up wit Stan NUTS engine, it comes together with chain mixing problems, so that one cannot then be confident about even the location of the obtained posterior distribution in that case;
- finally, although MFVB approximation leads to underestimation of the posterior dispersion, the mode location was found to be reliable (see Section 4.5.4).

Then, for each protein, and for each model parameter, the mean and standard deviation of the obtained marginal posterior distribution are spotted. Note that these can be calculated analytically from the various expression in Appendix E.1. From this exercise, histograms can be built for the set of means (resp. set of standard deviations) of the obtained posteriors for the whole population of proteins. However, when building the histograms, proteins are split into categories, as a function of the corresponding number of identified peptides, according to Table 5.1.

Prot. category	nb peptides	#proteins	#UPS1 proteins
A	< 5	662	20
B	5 to 9	401	11
C	≥ 10	286	6

Table 5.1: Protein categories, function of number of identified peptides

In this analysis, one concentrates more specifically on the γ parameters, i.e. the regression coefficients of the missingness part of the model. In Figure 5.1, these histograms are compared, between protein categories A (very few peptides) and C (10+ peptides), for the γ_y coefficient, linking missingness and intensities. A more complete set of histograms, for the entire set of γ model parameters, can be found in Appendix C.3

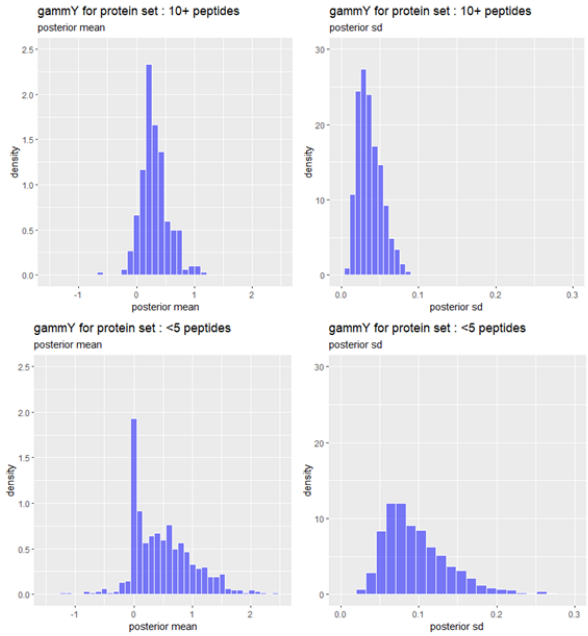


Figure 5.1: Means and standard deviations obtained for γ_y, using non informative priors. Comparison between the proteins of cat. C with 10+ peptides (above), and the proteins of cat. A with < 5 peptides (below)

Looking at Figure 5.1, it is clear that, for the proteins having very few peptides, the set of γ_y posterior distributions is rather ill-behaved. Indeed, the posterior mean dispersion is high,

and the associated standard deviations have high values as well. This shows that there is a high uncertainty associated to this parameter. Moreover, there is a very high peak at zero, and it was checked that this peak was associated with protein runs for which the chains could not mix. In comparison, the proteins with 10+ peptides lead to γ_y posterior distributions behaving much better, with a much lower dispersion of the mean, and much lower values of standard deviations as well. This is an indication that good identifiability of the γ_y coefficient is associated with a higher number of identified peptides. Moreover, the graphs shown in Appendix C.3 seem to suggest that for the other γ parameters, the conclusions are roughly the same.

In order to get some intuition about what improvement a hierarchical set-up could bring, consider the fit of the histograms obtained for the γ model parameters for the category C proteins (10+ peptides), obtained for parametric normal densities. One obtains the following fitted means and standard deviations (Table 5.2) :

γ parameter	Normal fitted mean	Normal fitted standard deviation
Intercept	-0.16	0.52
Lab2	1.09	0.64
Lab3	0.76	0.53
y	0.31	0.24

Table 5.2: Fitted normal density parameters for the γ parameters empirical distributions (category C proteins, 10+ peptides)

Then, the above fitted normal densities are used as new *informative* priors for the γ parameters in the single protein model - cf. eq.(4.2). Finally, the single protein model is run again, proteinwise, in its MFVB implementation, as before, and the corresponding histograms for the γ parameters are built again. The obtained histograms, again for parameter γ_y are displayed in Figure 5.2 - the same histograms for the other γ parameters are shown in Appendix C.3.

Although not perfect, the histograms displayed in Figure 5.2, where the axis for the mean histograms were intentially kept the same as before, show a visible improvement in terms of identifiability of the γ_y coefficients, for the A category proteins (< 5 peptides). Indeed, the histogram of the posterior mean has now a dispersion that is comparable for the two protein categories, and the peak at zero shown in Figure 5.1 has now vanished. Regarding the standard deviations, they now gently increase from category C (10+ peptides) to category A protein (< 5 peptides), which seems perfectly sound. The same conclusions seem to hold - and are in fact sometimes even more visible - for the other γ parameters as well (see again Appendix C.3).

All of the above tends to demonstrate that a hierarchical set-up for the γ parameters, where information from all proteins is borrowed to estimate the parameters of one protein, seems promising in order to improve the model identifiability issue spotted with the single protein model version.

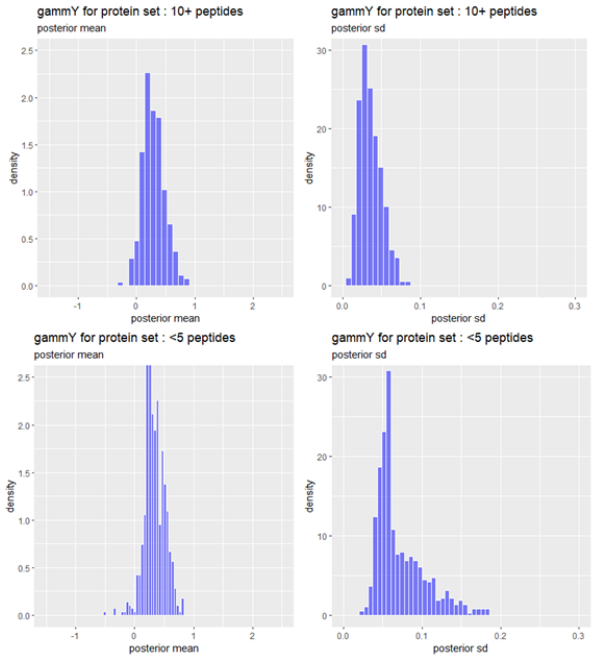


Figure 5.2: Means and standard deviations obtained for γ_y, using informative priors. Comparison between the proteins of cat. C with 10+ peptides (above), and the proteins of cat. A with < 5 peptides (below)

5.2 Hierarchical model description

In this section, one specifies the hierarchical model in mathematical equations. However, instead of describing the model from scratch, only the differences with the single protein model - described in Section 4.2 - are depicted. In particular, the two regression parts of the model assigning distributions to the two sets of responses (4.1) - or (4.4) in the data augmented version - stay exactly the same. However, the specification differences appear in the following priors :

$$\begin{aligned} \gamma_{i,k} &\sim \mathcal{N}(\mu_{0\gamma_i}, \sigma^2_{0\gamma_i}) \ ; \ \gamma_{y,k} \sim \mathcal{N}(\mu_{0\gamma_y}, \sigma^2_{0\gamma_y}), \ \forall \text{ protein } k \\ \sigma^2_{\epsilon,k} &\sim \text{Inv-}\chi^2(\nu_{0\epsilon}, s^2_{0\epsilon}), \ \forall \text{ protein } k \\ \sigma^2_{r,i,k} &\sim \text{Inv-}\chi^2(\nu_{0r_i}, s^2_{0r_i}), \ \forall \text{ protein } k \end{aligned} \tag{5.1}$$

$\mu_{0\gamma_i}, \sigma^2_{0\gamma_i}, \mu_{0\gamma_y}, \sigma^2_{0\gamma_y}, \nu_{0\epsilon}, s^2_{0\epsilon}, \nu_{0r_i},$ and $s^2_{0r_i}$ are the introduced hyper-parameters of the hierarchical model, linking together the a priori distributions of the model parameters of the different proteins. Note that, on top of hyperparameters related to the γ's, some hyperparameters related to the σ^2 were also included, following in that respect the same idea as in the Limma method [Smyth 2004]. However, hyper-priors were not specified - or equivalently, uniform improper hyper-priors were specified - counting on the fact that the posterior distributions of the hyper-parameters will be implied from data pertaining to a high number of proteins.

Figure 5.3 illustrates the Direct Acyclic Graph of the data augmented version of this hierarchical model, where the newly introduced hyperparameters appear in dashed contours. However, this DAG representation only displays a tiny part of the full hierarchical model. Indeed, all hyperparameters are in fact also linked, in the same fashion, to the corresponding parameters of all protein sub-models.

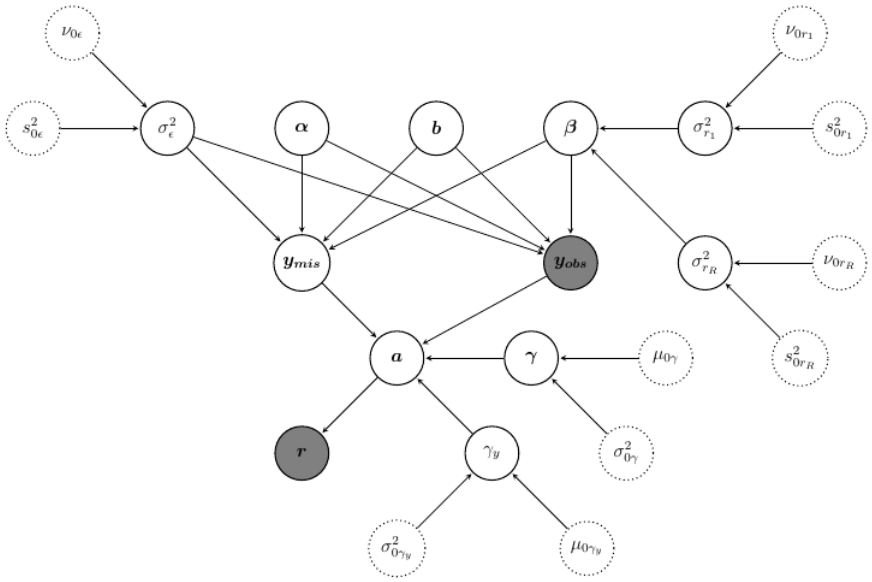


Figure 5.3: Direct Acyclic Graph (DAG) of the hierarchical model

In terms of model size, and when applied to the specific variable set attached to the CPTAC dataset (see Section 4.2.3), the hierarchical model leads to 8 hyperparameters which, added to the whole set of proteinwise model parameters, lead to a total number of parameters of 136 290 ! Moreover, contrary to the single protein case, it is here not possible to run the different protein parts independently, meaning that the whole model has to run at once in the simulation engine.

The above specified hierarchical model was implemented in Stan, using the same script as for the single protein model (Appendix G.1), but then run in a "multi-protein mode", and with the introduction of dedicated flags to activate the hierarchical set-up and the hyperparameters. Unfortunately however, it quickly appeared that the huge number of parameters was not tractable by the Stan NUTS engine, even if only a subset as small as 100 proteins was selected. This "non tractability" issue did not only translate into huge execution times, but also into the fact that the runs systematically ended up with MCMC chains that had clearly not mixed, with unacceptably high levels of $\hat{R}$'s indicators. An approximated approach, as described in the following section, is therefore again mandatory.

5.3 Hierarchical model - MFVB implementation

5.3.1 Application of MFVB to the hierarchical model

In this section, the principles of MFVB approximations exposed in Section 4.5 are applied to the hierarchical model. The motivation is three-fold :

- since the exact implementation in Stan NUTS is not tractable, one needs to find more efficient methods to run the hierarchical model;
- even if MFVB has a tendency to underestimate the posterior dispersion (cf. Section 4.5.4), one could still use MFVB for posterior mode identification and for comparing the hierarchical model with other models.
- among the approximations implemented for the single protein model, MFVB proved to be the most reliable (at least for mode identification) and was as CPU-efficient as the Laplace approximations.

Step 1 : deciding the product form of the approximated density

For the hierarchical model, the following product density restriction is imposed, where k is the protein index and K is the number of proteins :

$$\begin{aligned} & q([\boldsymbol{\alpha}_k, \boldsymbol{\beta}_k, \sigma^2_{\epsilon,k}, \boldsymbol{\sigma}^2_{r,k}, \boldsymbol{b}_k, \boldsymbol{y}_{mis,k}, \boldsymbol{a}_k, \boldsymbol{\gamma}_k, \gamma_{y,k}]_{k=1}^K, \boldsymbol{\mu}_{0\gamma}, \mu_{0\gamma_y}, \boldsymbol{\sigma}^2_{0\gamma}, \sigma^2_{0\gamma_y}, \nu_{0\epsilon}, s^2_{0\epsilon}, \boldsymbol{\nu}_{0r}, \boldsymbol{s}^2_{0r}) \\ & = \prod_{k=1}^{K} \left\{ q(\boldsymbol{\alpha}_k, \boldsymbol{\beta}_k, \boldsymbol{y}_{mis,k})\ q(\sigma^2_{\epsilon,k})\ q(\boldsymbol{\sigma}^2_{r,k})\ q(\boldsymbol{b}_k)\ q(\boldsymbol{a}_k)\ q(\boldsymbol{\gamma}_k, \gamma_{y,k}) \right\} \\ & \quad q(\boldsymbol{\mu}_{0\gamma}, \mu_{0\gamma_y})\ q(\boldsymbol{\sigma}^2_{0\gamma}, \sigma^2_{0\gamma_y})\ q(\nu_{0\epsilon})\ q(s^2_{0\epsilon})\ q(\boldsymbol{\nu}_{0r})\ q(\boldsymbol{s}^2_{0r}) \end{aligned} \tag{5.2}$$

Steps 2 and 3 : calculating the full conditionals and implying the optimal q^* densities

These two steps lead to optimal approximated densities $q^*(\cdot)$ such that (where N_k is the total amount of latent observations for protein k) :

$$\begin{aligned} (\boldsymbol{\alpha}_k, \boldsymbol{\beta}_k, \boldsymbol{y}_{mis,k}) & \overset{q^*}{\sim} \mathcal{N}(\boldsymbol{\mu}_{q^*(\alpha_k,\beta_k,y_{mis,k})}, \Sigma_{q^*(\alpha_k,\beta_k,y_{mis,k})}) \\ (\boldsymbol{\gamma}_k, \gamma_{y,k}) & \overset{q^*}{\sim} \mathcal{N}(\boldsymbol{\mu}_{q^*(\gamma_k,\gamma_{y,k})}, \Sigma_{q^*(\gamma_k,\gamma_{y,k})}) \\ q^*(\boldsymbol{a}_k) & = \text{product of } N_k \text{ truncated univariate normal densities } q^*(a_{i,k}), \text{ such that} \\ a_{i,k} & \overset{q^*}{\sim} \left\{ \begin{array}{l} \mathcal{TN}^+(\mu^*_{a,i,k}, \sigma^2_{a,i,k} = 1) \text{ when } r_{i,k} = 1 \\ \mathcal{TN}^-(\mu^*_{a,i,k}, \sigma^2_{a,i,k} = 1) \text{ when } r_{i,k} = 0 \end{array} \right\} \\ \sigma^2_{\epsilon,k} & \overset{q^*}{\sim} I.G.(A_{q^*(\sigma^2_{\epsilon,k})}, B_{q^*(\sigma^2_{\epsilon,k})}) \\ q^*(\boldsymbol{\sigma}^2_{r,k}) & = \text{product of } R \text{ univariate inverse gamma densities } q^*(\sigma^2_{r,i,k}), \text{ such that} \\ \sigma^2_{r,i,k} & \sim I.G.(A_{q^*(\sigma^2_{r,i,k})}, B_{q^*(\sigma^2_{r,i,k})}) \\ q^*(\boldsymbol{b}_k) & = \text{product of } N_k \text{ univariate inverse gamma densities } q^*(b_{i,k}) \text{ such that} \\ b_{ik} & \overset{q^*}{\sim} I.G.(A_{q^*(b_{i,k})}, B_{q^*(b_{i,k})}) \end{aligned} \tag{5.3}$$

$$
\begin{aligned}
q^*(\boldsymbol{\mu}_{0\gamma}, \mu_{0\gamma_y}) &= \text{product of } (M_\gamma + 1) \text{ univariate normal densities } q^*(\mu_{0\gamma_i}), \text{ such that} \\
\mu_{0\gamma_i} &\overset{q^*}{\sim} \mathcal{N}(\mu_{q^*(\mu_{0\gamma_i})}, \sigma^2_{q^*(\mu_{0\gamma_i})}), \forall i : 1, \ldots, (M_\gamma + 1) \\
q^*(\boldsymbol{\sigma}^2_{0\gamma}, \sigma^2_{0\gamma_y}) &= \text{product of } (M_\gamma + 1) \text{ univariate inverse gamma densities } q^*(\sigma^2_{0\gamma_i}), \text{ such that} \\
\sigma^2_{0\gamma_i} &\overset{q^*}{\sim} I.G.(A_{q^*(\sigma^2_{0\gamma_i})}, B_{q^*(\sigma^2_{0\gamma_i})}), \forall i : 1, \ldots, (M_\gamma + 1) \\
\frac{\nu_{0\epsilon}}{2} &\overset{q^*}{\sim} \nu \text{ distrib.}(\kappa_{q^*(\nu_{0\epsilon})}, \eta_{q^*(\nu_{0\epsilon})}) \\
s^2_{0\epsilon} &\overset{q^*}{\sim} \mathcal{G}(A_{q^*(s^2_{0\epsilon})}, B_{q^*(s^2_{0\epsilon})}) \\
q^*(\frac{\nu_{0r}}{2}) &= \text{product of } R \text{ univariate distributions } q^*(\frac{\nu_{0r_i}}{2}) \text{ such that} \\
\frac{\nu_{0r_i}}{2} &\overset{q^*}{\sim} \nu\text{distrib.}(\kappa_{q^*(\nu_{0r_i})}, \eta_{q^*(\nu_{0r_i})}) \\
q^*(s^2_{0r}) &= \text{product of } R \text{ univariate gamma distributions } q^*(s^2_{0r_i}) \text{ such that} \\
s^2_{0r_i} &\overset{q^*}{\sim} \mathcal{G}(A_{q^*(s^2_{0r_i})}, B_{q^*(s^2_{0r_i})})
\end{aligned}
\tag{5.4}
$$

Detailed expressions for the various distribution parameters introduced in (5.3) and (5.4) can be found in Appendix E.3. Note that the optimal q^* densities attached to the ν hyper-parameters - i.e. the degrees of freedom of the σ^2 priors - have non standard distributions, which, in (5.3), are denominated as "ν distrib.(κ, η)". Their log-densities have the following expression :

$$
\log q^*(\frac{\nu}{2}; \kappa, \eta) = \kappa[(\frac{\nu}{2}) \log(\frac{\nu}{2}) - \log \Gamma(\frac{\nu}{2})] - (\kappa + \eta)(\frac{\nu}{2}) + const. \tag{5.5}
$$

where $\Gamma(x)$ is the gamma function, η and κ are positive, and η is typically small compared to κ. For the sake of running the MFVB algorithm, as well as for marginal posterior sampling, these "ν" distributions have been approximated by gamma distributions having the same mode and curvature at the mode. The choice of a gamma distribution as an approximation has been driven by an idea exposed in [Jullion and Lambert 2007]. The interested reader can find some details of this approximation in Appendix E.5, as well as some graphs demonstrating its accuracy.

Step 4 - building the MFVB iterative algorithm

Given the expressions obtained in the previous steps, the optimal parameters of the approximated densities may be obtained iteratively, using Algorithms 3 and 4, where the used notations were introduced in Section 4.2.2 and Section 5.2, as well as in formulas (5.3) and (4.16). The pseudo-code of the GammaApproxi(κ, η) procedure can be found in Appendix E.5.

Step 5 - deriving the expression for the ELBO

Since the convergence of the algorithm needs to be checked, and one wants to avoid monitoring separately each single parameter of (5.3) and (5.4), one also needs to derive the analytical expression for the ELBO. In the hierarchical model case, the outcome is a one full page long expression, which is provided in Appendix E.4.

Algorithm 3 Iterative scheme for obtaining MFVB approximated densities for the hierarchical model - Part 1

procedure UPDATEPROTPARAMS(k) ▷ Updates all model parameters for protein k

$\mathbb{E}_q[D_k] \leftarrow \text{diag}(\mathbb{E}_q[\frac{1}{\boldsymbol{b},k}])$; $\mathbb{E}_q[D_{obs,k}] \leftarrow \text{diag}(\mathbb{E}_q[\frac{1}{\boldsymbol{b}_{obs,k}}])$; $\mathbb{E}_q[D_{mis,k}] \leftarrow \text{diag}(\mathbb{E}_q[\frac{1}{\boldsymbol{b}_{mis,k}}])$

$$\mathbb{E}_q[\Sigma^{-1}_{\alpha\beta,k}(\boldsymbol{\sigma}^2_{r,k})] \leftarrow \begin{bmatrix} \Sigma_\alpha^{-1} & 0 \\ 0 & \text{blockdiag}\left(\mathbb{E}_q[\frac{1}{\sigma^2_{r,1,k}}]I_{M^{\beta(1)}}, \ldots, \mathbb{E}_q[\frac{1}{\sigma^2_{r,R,k}}]I_{M^{\beta(R)}}\right) \end{bmatrix}$$

$$\Sigma_{q(\alpha_k,\beta_k,y_{mis,k})} \leftarrow \begin{bmatrix} \mathbb{E}_q[\frac{1}{\sigma^2_{\epsilon,k}}]W_k^T\mathbb{E}_q[D_k]W_k & -\mathbb{E}_q[\frac{1}{\sigma^2_{\epsilon,k}}]W^T_{mis,k}\mathbb{E}_q[D_{mis,k}] \\ \quad + \mathbb{E}_q[\Sigma^{-1}_{\alpha\beta,k}(\boldsymbol{\sigma}^2_{r,k})] & \\ -\mathbb{E}_q[\frac{1}{\sigma^2_{\epsilon,k}}]\mathbb{E}_q[D_{mis,k}]W_{mis,k} & (\sigma^2_{q(\gamma_{y,k})} + \mu^2_{q(\gamma_{y,k})})I_{N_{mis,k}} \\ & \quad + \mathbb{E}_q[\frac{1}{\sigma^2_{\epsilon,k}}]\mathbb{E}_q[D_{mis,k}] \end{bmatrix}^{-1}$$

$$\boldsymbol{\mu}_{q(\alpha_k,\beta_k,y_{mis,k})} \leftarrow \Sigma_{q(\alpha_k,\beta_k,y_{mis,k})} \begin{bmatrix} \mathbb{E}_q[\frac{1}{\sigma^2_{\epsilon,k}}]W^T_{obs,k}\mathbb{E}_q[D_{obs,k}]\boldsymbol{y}_{obs,k} + \mathbb{E}_q[\Sigma^{-1}_{\alpha\beta,k}(\boldsymbol{\sigma}^2_{r,k})]\boldsymbol{\mu}_{\alpha\beta} \\ \mu_{q(\gamma_{y,k})}(\boldsymbol{\mu}_{q(a_k)} - T_{mis,k}\boldsymbol{\mu}_{q(\gamma_k)} + \mu_{q(\gamma_{y,k})}y_{ref}) \end{bmatrix}$$

$$\Sigma_{q(\gamma_k,\gamma_{y,k})} \leftarrow \left\{[T_k\ (\boldsymbol{\mu}_{q(y_k)} - y_{ref})]^T[T_k\ (\boldsymbol{\mu}_{q(y_k)} - y_{ref})] + \begin{bmatrix} 0 & 0 \\ 0 & \text{tr}(\Sigma_{q(y_k)}) \end{bmatrix} + \text{diag}(\mathbb{E}_q[\frac{1}{\boldsymbol{\sigma}^2_{0\gamma}}])\right\}^{-1}$$

$$\boldsymbol{\mu}_{q(\gamma_k,\gamma_{y,k})} \leftarrow \Sigma_{q(\gamma_k,\gamma_{y,k})}\left([T_k\ (\boldsymbol{\mu}_{q(y_k)} - y_{ref})]\boldsymbol{\mu}_{q(a_k)} + \text{diag}(\mathbb{E}_q[\frac{1}{\boldsymbol{\sigma}^2_{0\gamma}}])\ \boldsymbol{\mu}_{q(\mu_{0\gamma\gamma_y})}\right)$$

$$\boldsymbol{\mu}_{a_k} \leftarrow T_k\boldsymbol{\mu}_{q(\gamma_k)} + \mu_{q(\gamma_{y,k})}(\boldsymbol{\mu}_{q(y_k)} - y_{ref})$$

$$\boldsymbol{\mu}_{q(a_k)} \leftarrow \boldsymbol{\mu}_{a_k} + \frac{(2\boldsymbol{r}_k - 1)\odot\phi(\boldsymbol{\mu}_{a_k})}{\Phi\{(2\boldsymbol{r}_k - 1)\odot\boldsymbol{\mu}_{a_k}\}}$$

$$A_{q(\sigma^2_{\epsilon,k})} \leftarrow \mathbb{E}_q[\tfrac{\nu_{0\epsilon}}{2}] + \tfrac{N_k}{2}$$

$$B_{q(\sigma^2_{\epsilon,k})} \leftarrow \mathbb{E}_q[\tfrac{\nu_{0\epsilon}}{2}]\mathbb{E}_q[s^2_{0\epsilon}] + \tfrac{1}{2}\Big\{(\boldsymbol{\mu}_{q(y_k)} - W_k\boldsymbol{\mu}_{q(\alpha_k,\beta_k)})^T\mathbb{E}_q[D_k](\boldsymbol{\mu}_{q(y_k)} - W_k\boldsymbol{\mu}_{q(\alpha_k,\beta_k)}) + \text{tr}\left(\mathbb{E}_q[D_k]\begin{bmatrix} -W_{obs,k} & 0 \\ -W_{mis,k} & I_{N_{mis,k}} \end{bmatrix}\Sigma_{q(\alpha_k,\beta_k,y_k)}\begin{bmatrix} -W_{obs,k} & 0 \\ -W_{mi,ks} & I_{N_{mis,k}} \end{bmatrix}^T\right)\Big\}$$

$$\mathbb{E}_q[\tfrac{1}{\sigma^2_{\epsilon,k}}] \leftarrow A_{q(\sigma^2_{\epsilon,k})}/B_{q(\sigma^2_{\epsilon,k})}; \mathbb{E}_q[\log(\sigma^2_{\epsilon,k})] \leftarrow \log(B_{q(\sigma^2_{\epsilon,k})}) - \psi(A_{q(\sigma^2_{\epsilon,k})})$$

for $i \in \{1, \ldots, R\}$ **do**

$$A_{q(\sigma^2_{r,i,k})} \leftarrow \mathbb{E}_q[\tfrac{\nu_{0,r,i}}{2}] + \tfrac{M_k^{\beta(i)}}{2}$$

$$B_{q(\sigma^2_{r,i,k})} \leftarrow \mathbb{E}_q[\tfrac{\nu_{0,r,i}}{2}]\mathbb{E}_q[s^2_{0,r,i}] + \tfrac{1}{2}\left\{(\boldsymbol{\mu}^{(r,i)}_{q(\alpha_k,\beta_k)})^T\boldsymbol{\mu}^{(r,i)}_{q(\alpha_k,\beta_k)} + \text{tr}(\Sigma^{(r,i)}_{q(\alpha_k,\beta_k)})\right\}$$

$$\mathbb{E}_q[\tfrac{1}{\sigma^2_{r,i,k}}] \leftarrow A_{q(\sigma^2_{r,i,k})}/B_{q(\sigma^2_{r,i,k})}\ ;\ \mathbb{E}_q[\log(\sigma^2_{r,i,k})] \leftarrow \log(B_{q(\sigma^2_{r,i,k})}) - \psi(A_{q(\sigma^2_{r,i,k})})$$

end for

$$B_{q(b_k)} \leftarrow \tfrac{\nu}{2} + \tfrac{1}{2}\mathbb{E}_q[\tfrac{1}{\sigma^2_{\epsilon,k}}]\Big\{(\boldsymbol{\mu}_{q(y_k)} - W_k\boldsymbol{\mu}_{q(\alpha_k,\beta_k)}) \odot (\boldsymbol{\mu}_{q(y_k)} - W_k\boldsymbol{\mu}_{q(\alpha_k,\beta_k)}) + \text{diag}([-W_k\ I_{N_k}]\ \Sigma_{q(\alpha_k,\beta_k,y_k)}\ [-W_k\ I_{N_k}]^T)\Big\}$$

$$\mathbb{E}_q[\tfrac{1}{\boldsymbol{b}_k}] \leftarrow (\tfrac{\nu}{2} + \tfrac{1}{2})/B_{q(b_k)}$$

end procedure

Algorithm 4 Iterative scheme for obtaining MFVB approximated densities for the hierarchical model - Part 2

for $k \in \{1, \ldots, K\}$ **do**

Initialize: $\begin{cases} \sigma^2_{q(\gamma_{y,k})} = 1 \ ; \ \mu_{q(\gamma_{y,k})} = 0 \ ; \ \boldsymbol{\mu}_{a,k} = 0_{N_k} \ ; \ \boldsymbol{\mu}_{q(a_k)} = 2(2\boldsymbol{r}_k - 1)\phi(0); \\ \mathbb{E}_q[\frac{1}{\boldsymbol{b}_k}] = 1_{N_k} \ ; \ \mathbb{E}_q[\frac{1}{\sigma^2_{\epsilon,k}}] = 1 \ ; \ \mathbb{E}_q[\frac{1}{\boldsymbol{\sigma}^2_{r,k}}] = 1_R \end{cases}$

end for

Initialize: $\begin{cases} \mathbb{E}_q[\frac{\nu_{0\epsilon}}{2}] = \log(K) \ ; \ \mathbb{E}_q[s^2_{0\epsilon}] = 1 \ ; \ \mathbb{E}_q[\frac{\nu_{0r_i}}{2}] = \log(K) \ ; \ \mathbb{E}_q[s^2_{0r_i}] = 1 \\ \mathbb{E}_q[\Sigma_{0\gamma\gamma_y}] = \mathrm{diag}(1_{M^\gamma+1}) \ ; \ \mathbb{E}_q[\mu_{0\gamma\gamma_y}] = 0_{(M^\gamma+1)} \end{cases}$

repeat

  for $k \in \{1, \ldots, K\}$ **do**

    UpdateProtParams(k)

  end for

  for all γ_i **do**

    $\mu_{q(\mu_{0\gamma_i})} \leftarrow \frac{1}{K} \sum_{k=1}^K \mu_{q(\gamma_{i,k})}$

    $\sigma^2_{q(\mu_{0\gamma_i})} \leftarrow (K\mathbb{E}_q[\frac{1}{\sigma^2_{0\gamma_i}}])^{-1}$

    $B_{q(\sigma^2_{0\gamma_i})} \leftarrow \frac{1}{2}\big\{ \sum_{k=1}^K (\mu_{q(\gamma_{i,k})} - \mu_{q(\mu_{0\gamma_i})})^2 + \sum_{k=1}^K \sigma^2_{q(\gamma_{i,k})} + K\sigma^2_{q(\mu_{0\gamma_i})} \big\}$

    $\mathbb{E}_q[\frac{1}{\sigma^2_{0\gamma_i}}] \leftarrow (\frac{K}{2} - 1)/B_{q(\sigma^2_{0\gamma_i})}$

  end for

  $A_{q(s^2_{0\epsilon})} \leftarrow K\mathbb{E}_q[\frac{\nu_{0\epsilon}}{2}] + 1$

  $B_{q(s^2_{0\epsilon})} \leftarrow \mathbb{E}_q[\frac{\nu_{0\epsilon}}{2}] \sum_{k=1}^K \mathbb{E}_q[\frac{1}{\sigma^2_{\epsilon_k}}]$

  $\mathbb{E}_q[s^2_{0\epsilon}] \leftarrow A_{q(s^2_{0\epsilon})}/B_{q(s^2_{0\epsilon})} \ ; \ \mathbb{E}_q[\log(s^2_{0\epsilon})] \leftarrow \psi(A_{q(s^2_{0\epsilon})}) - \log(B_{q(s^2_{0\epsilon})})$

  $\eta_{q(\nu_{0\epsilon})} \leftarrow \sum_{k=1}^K \mathbb{E}_q[\log(\sigma^2_{\epsilon,k})] + \mathbb{E}_q[s^2_{0\epsilon}] \sum_{k=1}^K \mathbb{E}_q[\frac{1}{\sigma^2_{\epsilon,k}}] - K(\mathbb{E}_q[\log(s^2_{0\epsilon})] + 1)$

  $(A_{q(\nu_{0\epsilon})}, B_{q(\nu_{0\epsilon})}) \leftarrow$ GammaApproxi$(K, \eta_{q(\nu_{0\epsilon})}) \ ; \ \mathbb{E}_q[\frac{\nu_{0\epsilon}}{2}] \leftarrow A_{q(\nu_{0\epsilon})}/B_{q(\nu_{0\epsilon})}$

  for $i \in \{1, \ldots, R\}$ **do**

    $A_{q(s^2_{0r_i})} \leftarrow K\mathbb{E}_q[\frac{\nu_{0r_i}}{2}] + 1$

    $B_{q(s^2_{0r_i})} \leftarrow \mathbb{E}_q[\frac{\nu_{0r_i}}{2}] \sum_{k=1}^K \mathbb{E}_q[\frac{1}{\sigma^2_{r,i,k}}]$

    $\mathbb{E}_q[s^2_{0r_i}] \leftarrow A_{q(s^2_{0r_i})}/B_{q(s^2_{0r_i})} \ ; \ \mathbb{E}_q[\log(s^2_{0r_i})] \leftarrow \psi(A_{q(s^2_{0r_i})}) - \log(B_{q(s^2_{0r_i})})$

    $\eta_{q(\nu_{0r_i})} \leftarrow \sum_{k=1}^K \mathbb{E}_q[\log(\sigma^2_{r,i,k})] + \mathbb{E}_q[s^2_{0r_i}] \sum_{k=1}^K \mathbb{E}_q[\frac{1}{\sigma^2_{r,i,k}}] - K(\mathbb{E}_q[\log(s^2_{0r_i})] + 1)$

    $(A_{q(\frac{\nu_{0r_i}}{2})}, B_{q(\frac{\nu_{0r_i}}{2})}) \leftarrow$ GammaApproxi$(K, \eta_{q(\nu_{0r_i})}) \ ; \ \mathbb{E}_q[\frac{\nu_{0r_i}}{2}] \leftarrow A_{q(\frac{\nu}{2})}/B_{q(\frac{\nu}{2})}$

  end for

until convergence, i.e. ELBO(q) converges to a maximum

5.3.2 MFVB Results for the hierarchical model

In this section, some results obtained with the MFVB implementation of the hierarchical model are presented. To obtain these, the algorithm described in Section 5.3.1 was implemented in R, and was run on the CPTAC dataset. As was the case for the single protein model, good convergence of the ELBO could be reached (see Figure 5.4), which is also an indicator of correctness of the formulas obtained for the approximate densities (Appendix E.3) and that of a correct implementation of the corresponding MFVB iterative algorithm (Algorithm 3 and 4 in Section 5.3.1).

Note that, in order to reach acceptable execution time, the algorithm had to be parallelized, such that, in each iteration, the procedure updating the set of protein model parameters (Algorithm 3) could be called on several cores simultenously. Thanks to that, the full algorithm applied on the 1347 proteins ran in about two hours on 6 CPU cores.

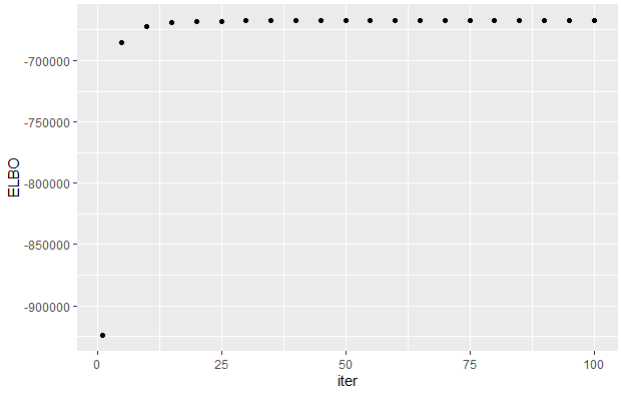


Figure 5.4: Convergence of the ELBO as a function of iteration number

In terms of results, one concentrates here more specifically on the approximated marginal posterior distribution of the hyperparameters. Starting with the hyperparameters related to the σ^2-alike model parameters (error variances and peptide random effect variances), Table 5.3 displays some descriptive statistics of the obtained approximate posterior distributions.

hyperparameter	mean	sd	$\sqrt{\text{mean}}$
$\nu_{0\epsilon}$	7.16	0.26	
$s^2_{0\epsilon}$	0.14	0.002	0.37
$\nu_{0r,pept}$	8.40	0.31	
$s^2_{0r,pept}$	1.67	0.02	1.29

Table 5.3: Hyperparmeters related to the σ^2 parameters - MFVB hierarchical model fitted on all proteins

From Table 5.3, one can make the following observations :

- the σ^2 related hyperparameters show posterior standard deviations that are quite small compared to their mean

- $\nu_{0\epsilon}$ has a mean around 7 while $s^2_{0\epsilon}$ has a mean of $0.14 = (0.37)^2$. These values are more or less equivalent to providing a common prior for all $\sigma^2_{\epsilon,k}$, with roughly the same information content as 7 prior observations having all a residual standard deviation of 0.37. This is a rather weak prior in comparison with the number of data observations available for an average protein (i.e. 100, cf. Table 4.1).

- $\nu_{0r,pept}$ has a mean around 8 while $s^2_{0r,pept}$ has a mean of $1.67 = (1.29)^2$. These values are more or less equivalent to providing a common prior for all $\sigma^2_{r,pept,k}$, with roughly the same information content as 8 prior peptides having all a random peptide effect standard deviation of 1.29. Here, this is a rather strong prior, considering the average number of identified peptides per protein (i.e. 6.8, cf. Table 4.1).

Considering now the gamma related hyperparameters, one can find some results in Table 5.4.

γ parameter	mean(μ_0)	sd(μ_0)	mean(σ^2_0)	sd(σ^2_0)	$\sqrt{\text{mean}(\sigma^2_0)}$
Intercept	0.30	0.07	5.98	0.23	2.45
Lab2	0.99	0.06	4.71	0.18	2.17
Lab3	0.56	0.06	5.44	0.21	2.33
y	0.46	0.01	0.24	0.01	0.49

Table 5.4: Hyperparmeters related to the γ parameters - MFVB hierarchical model fitted on all proteins

From Table 5.4, there are two important findings to be reported :

- as for the σ^2 related hyperparameters, the standard deviations of all gamma hyperparameters are quite small compared to their mean.

- the distribution of all σ^2_0 hyperparameters are centered around values that are quite high, compared to the corresponding μ_0 parameters. The latter comparison is more easily performed using the last column of Table 5.4, where the square root of the posterior mean of the dispersion hyperparameters is provided.

The first finding, and its implications, will be looked at in more details in Section 5.4. However, the second finding is disappointing as it basically implies that the hierarchical priors for the gamma parameters are weak. However, the hierarchical set-up was introduced with the hope to improve the identifiability of the single protein model (see Section 5.1). Since this identifiability issue was specifically present for the gamma parameters, noticeable improvement can only be possible if the hierarchical priors for the gamma parameters are strong enough, hence if the corresponding dispersion hyperparameters are small enough.

This disappointing result is, for example, confirmed in Figure 5.5, showing the distribution of the protein-wise γ_y posterior means and standard deviations. This graph shows very similar characteristics (peak at zero, wide range of values for the posterior mean, high values for the standard deviations) as those of Figure 5.1 in section 5.1.

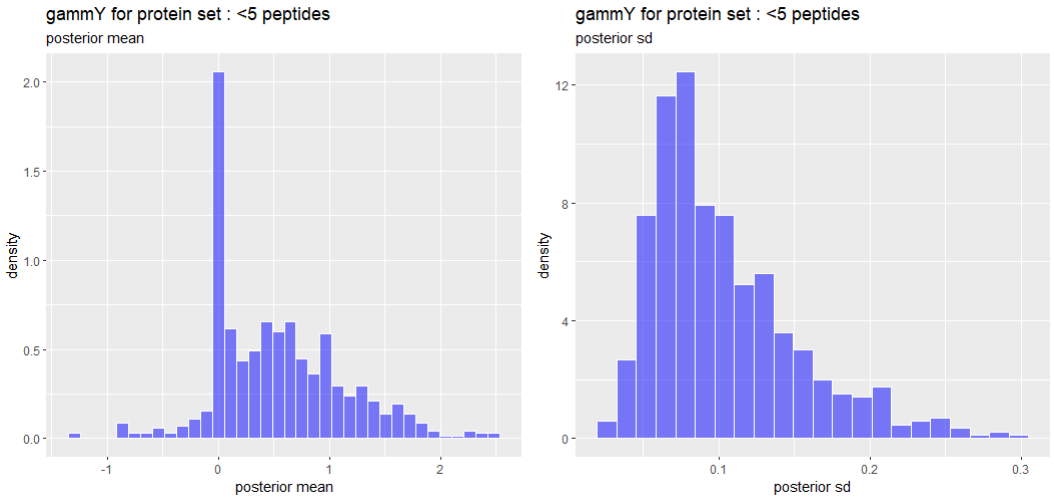


Figure 5.5: Means and standard deviations obtained for γ_y, using the hierarchical model, for the proteins with < 5 peptides

The latter observation basically discards the hierarchical model in its current design, as the identifiability issue observed with the single protein model is not solved.

As a side remark, note that unfortunately one cannot rely, as was the case with the single protein model, to a reference "exact" MCMC implementation to benchmark the MFVB approximated distributions. As mentioned previously, it was not possible to obtain such a benchmark from Stan NUTS. This is undoubtedly a weakness as one is therefore unable to assess to which extend MFVB approximation here underestimate the dispersion of the various parameters. However, it is believed that the conclusions drawn in this section are nonetheless valid, given they are made based on values taken by the mode of the hyperparameters, not their dispersion.

5.4 Two-step empirical Bayes approach

5.4.1 Motivation

In this new section, a two-step empirical Bayes approach is described, with the aim to overcome the limitations of the hierarchical model discussed in Section 5.3.

In order to introduce the idea behind this new approach, consider running again the hierarchical model, but this time keeping only the proteins having 10 or more identified peptide in the dataset. This is similar to what was done in Section 5.1, but using the hierarchical model instead of the single protein model. Now Table 5.5 is obtained for the gamma related hyperparameters :

γ parameter	mean(μ_0)	sd(μ_0)	mean(σ_0^2)	sd(σ_0^2)	$\sqrt{\text{mean}(\sigma_0^2)}$	variance ratio
Intercept	-0.17	0.03	0.28	0.02	0.53	0.9969
Lab2	1.09	0.04	0.44	0.04	0.67	0.9964
Lab3	0.76	0.03	0.31	0.03	0.55	0.9971
y	0.31	0.01	0.06	0.005	0.24	0.9983

Table 5.5: Hyperparmeters related to the γ parameters - MFVB hierarchical model fitted on proteins with 10+ peptides

Firstly, Table 5.5 shows that the σ_0^2 hyperparameters have now smaller values, leading to hyper-priors that are stronger. Actually, one can notice that these values are very similar to the one found in the discussion of Section 5.1 (see Table 5.2). One can therefore expect a firm improvement in terms of identifiability of the gamma parameters for each protein.

Secondly, as was already the case in the results shown in Section 5.3.2 (Table 5.4), the standard deviation of the μ_0 parameters are very small compared to the mean of the corresponding σ_0^2 parameters. These values correpond to variance ratios defined as :

$$\text{variance ratio}(\gamma_i) = \frac{\text{posterior mean}(\sigma_{0,i}^2)}{(\text{posterior mean}(\sigma_{0,i}^2) + (\text{posterior sd}(\mu_{0,i}))^2)}, \tag{5.6}$$

which are all very close to 1, as disclosed in the last column of Table 5.5.

The latter observation seems to support the use of an *empirical Bayes* method. In such a method, one would replace the distributions of the μ_0 and σ_0^2 hyperparameter of each γ parameter, by a single value, a pointwise estimate, here set equal to their posterior mean.

These two findings lead to the following proprosal for a *two-step empirical Bayes procedure* :

1. run the MFVB implementation of the hierarchical model, using only the proteins with "sufficiently high" number of identified peptides (e.g. 10+, as was used here). Obtain the pointwise estimates (mode) of all hyperparameters of the model.

2. run the single protein version of the model on all proteins, but using, as informative priors for the γ and σ^2 parameters, the pointwise estimates obtained in step 1.

The advantages of the following procedure are the following :

- from the MFVB algorithm, only pointwise estimates are used, for which it is now known that MFVB provides good results. However, inference itself can now be done using the exact MCMC engine.

- there is hope to solve the identifiability issue observed for the γ parameters, thanks to the informative priors obtained within the hierarchical set-up in step 1.

- in terms of computation time, the exact MCMC inference part is run with the single protein model, which can be run in parallel.

- one can hope to further speed up step 2 by providing good first guesses of the model parameters, obtained from the MFVB step, as an input to the MCMC step.

This procedure can be valid as soon as two important conditions are met :

- Firstly, applying an empirical Bayes method for the hyperparameters should not cut too much variability. It seems that this can be justified given the typically high number of proteins at hand, and was demonstrated by the variance ratios shown in Table 5.5.
- Secondly, including in step 1, only the proteins with a "sufficiently high" number of identified peptides, should not lead to biased model parameters priors used in the missingness regression part of the model. The latter condition is arguably a bit more difficult to justify. Conceptually however, one cannot think of any obvious reason why the missingness pattern of peptide intensities should be linked to the number of peptides identified for a given protein.

In an attempt to support the latter assertion, Table 5.6 displays the point estimates obtained for the γ hyperparameters when varying the minimum number of peptide threshold used in step 1. One can see that these hyperparameters point estimates do not vary a great deal, when slightly changing the peptide threshold, but these variations are still significant compared to their associated uncertainties. Also there is a noticeable trend for the μ_0(Intercept) parameter, where its pointwise estimate seems to increase as the number of peptides threshold increases. The latter findings are unfortunately not fully supportive of the proposed peptide threshold approach, and would deserve more investigation, for example by assessing whether they are also present when applying the proposed approach on other (benchmark) proteomics datasets.

pointwise estimate	8+ pept.	10+ pept.	12+ pept.	15+ pept.	sd for 10+ pept.
μ_0(Intercept)	-0.12	-0.17	-0.24	-0.27	0.03
μ_0(lab2)	1.06	1.10	1.13	1.09	0.04
μ_0(lab3)	0.73	0.76	0.77	0.77	0.03
μ_0(y)	0.33	0.31	0.27	0.27	0.01
nb proteins	396	284	201	125	
average peptide intensity	18.80	18.86	18.91	18.92	

Table 5.6: Point estimates of γ hyperparameters as a function of number of peptides threshold.

5.5 Model comparison study

In this section, some results obtained when applying the two-step empirical Bayes approach described above, on the CPTAC dataset, are presented and discussed. More specifically, a short model comparison study is done, where specific attention is devoted to the obtained posterior distributions of the log fold changes between conditions (α_{cond} parameters), in different model settings. In this study, one takes advantage of the knowledge of the 'true' log fold change between conditions (see Section 2.3).

5.5.1 Compared models

The compared models are the following :

Model 1 : *Simple Linear* The first model is a simple linear regression model, containing the following ingredients :

- the peptide intensity model has the following formula : $y \sim condition+lab+peptide$
- the log transformed peptide intensities are modelled by a normal distribution
- there is no modelling of missingness
- weakly informative priors are used, as described in Section 4.2.3, there is no hierarchical prior

Model 2: ***Robust Linear*** The second model is a robust extension of model 1, where the log transformed peptide intensities are modelled by a Student-t distribution with $\nu = 4$ degrees of freedom.

Model 3: ***Robust Hierarchical*** The third model adds a hierarchical prior for the error variance parameter σ_ϵ^2, therefore enabling variance moderation, similarly to Limma. This hierarchical model is implemented using a two-step empirical Bayes approach, similar to the one described in Section 5.4, but without using any threshold on the protein number of peptides when performing the MFVB step. Indeed, for this model, there was no need for such a threshold as there seemed to be no model identifiability issue linked to small numbers of identified peptides per protein.

Model 4: ***Full Hierarchical*** The fourth model is the full hierarchical model, as described in the previous sections of this chapter. The two-step empirical Bayes approach here uses a threshold on the protein number of peptides set to 10.

Detailed model specifications for the first three models, which are compared with the full hierarchical model, are provided in Appendix F.1. In terms of implementation, it was here taken advantage of the flexibility and modularity offered by the *StanProt* package, which was developed as part of this work (see Section 5.6). Indeed, all models could be implemented in the Stan NUTS engine, using the same model script (Appendix G.1), and complemented by the MFVB algorithm described in Section 5.3.1, whenever the two-step empirical Bayes approach was needed, i.e. for models 3 and 4.

5.5.2 Execution times

The respective total execution times obtained for the four competing models, when run on the same 6 cores desktop, are provided in Table 5.7. When a two-step empirical Bayes approach is used, the total execution time is split into two parts, i.e. for the MFVB step, and for the MCMC Stan NUTS step.

Model	MFVB step	MCMC step	Total execution time
Simple Linear	–	1h04	1h04
Robust Linear	–	1h22	1h22
Robust Hierarchical	0h47	1h22	2h09
Full Hierarchical	1h40	11h35	13h15

Table 5.7: Execution times for the four competing models

Table 5.7 shows that there is a huge increase in computation time, when adding the missingness part on top of the intensity regression part. Indeed, the Full Hierarchical model needs more than 13 hours to run, compared to execution times between one and two hours

for the other models. Of these 13 hours, most of it is spent in running the MCMC step with the single protein model, in parallel, using Stan NUTS engine. In comparison, the addition of the hierarchical set-up for the error variance (Robust Hierarchical model vs. Robust Linear model) is relatively cheap in terms of computation time.

5.5.3 Comparing the RMSE for the log fold changes

Now focusing on the log fold changes between the different concentration conditions (α_{cond} parameters), one selects two contrasts of interest : condition B vs. condition A (hereafter mentioned as B vs. A), which corresponds to the value of $\alpha_{cond,B}$, and condition C vs. condition B (C vs. B), which corresponds to the difference $(\alpha_{cond,C} - \alpha_{cond,B})$. Using the benchmark property of the dataset, one can assess that the 'true' log fold change for these two contrasts, are both equal to 1.575 for UPS proteins (see section 2.3) and to 0 for non UPS proteins. Therefore, denoting by $c_{k,tgt}$ the target fold change corresponding to protein k (i.e. 1.575 or 0), one can compute the Root Mean Square Error (RMSE) for the four competing models, and for the two considered contrasts, according to the following formula :

$$RMSE = \sqrt{\frac{1}{K}\sum_{k=1}^{K}(\hat{c}_k - c_{k,tgt})^2} \tag{5.7}$$

where K in the number of measured proteins and $\hat{c}_k$ is a point estimate obtained from the posterior distribution of the contrast being considered. Here, the chosen point estimate is the median of the sampled posterior distribution. These calculations lead to the results provided in Table 5.8, where the standard deviations of the RMSE have also been computed, according to (5.8) (see [Faber 1999]) :

$$\hat{\sigma}(RMSE) \approx RMSE \sqrt{\frac{1}{2K}} \tag{5.8}$$

Model	RMSE (B vs. A)	RMSE (C vs. B)
Simple Linear	0.193±0.004	0.169±0.003
Robust Linear	0.176±0.003	0.159±0.003
Robust Hierarchical	0.175±0.003	0.160±0.003
Full Hierarchical	0.170±0.003	0.159±0.003

Table 5.8: Root Mean Square Error (RMSE) for each model, for each of the contrasts of interest. For each RMSE, an estimate of its standard deviation is also provided.

From Table 5.8, and using the Simple Linear model as an initial benchmark, one can notice a significant improvement in the RMSE when adding the robust regression element. However, although there might be a slight additional RMSE gain provided by the Full Hierarchical model, compared with the other models which do not include missingness, this gain is small in the B vs. A case - and here not significant - , while it is not even visible in the C vs. B case. Note that since the A condition is the one with the lowest UPS protein concentration, it could well be that modelling a link between low abundance and data missingness could be more beneficial in that case. However, the results in Table 5.8 suggest that the improvement, in terms of RMSE, could mainly come from the robust regression, and not from the modelling of the missingness mechanism.

5.5.4 Volcano plots

One way to assess the inferential performance of the competing models, is to look at their respective errors when classifying proteins between differentially abundant or not differentially abundant. For example, one can compare the number of false positive and false negative when applying a specific classification rule. In a frequentist setting, one would use hypothesis testing to decide, based on a statistical model, whether a given protein is differentially abundant between two conditions, and apply a multiple test correction to obtained adjusted p-values. However, in a Bayesian setting, there is no such concept as a p-value. Instead, one needs to look at the location and dispersion of the posterior distributions of the relevant linear combinations of model parameters. Here, the following decision rule will be applied :

1. choose a positive log2 fold change detection threshold lFC_{thresh}, for example 0.5 or 1. For example, it can be set according to the expert opinion of the investigator.
2. calculate a posterior probability that protein k is differentially abundant as $p_{DA}(k) = p(c_k > lFC_{thresh}|\mathcal{D})$. Note that in the CPTAC benchmark case, a one sided comparison can be used, as the sample of higher concentration is well known.
3. classify proteins as differentially abundant, when the probability calculated above is higher than a chosen level, set between 0.5 and 0.99, depending on the selected level of conservatism.

By applying the above described decision rule, and by varying the values of the posterior probability threshold level, one obtains, for example, the results provided in Table 5.9, for the B vs. A contrast, and with $lFC_{thresh} = 0.5$.

Model	Indicator	Proba level				
		50%	60%	70%	80%	90%
Simple Linear	FP	9	7	2	0	0
	FN	4	5	5	6	6
Robust Linear	FP	5	5	3	0	0
	FN	4	5	5	5	6
Robust Hierarchical	FP	5	4	2	0	0
	FN	4	4	5	5	5
Full Hierarchical	FP	4	3	1	1	0
	FN	3	3	4	4	5

Table 5.9: false positives (FP) and false negatives (FN) for the four competing models, for the B vs. A contrast, and with $lFC_{thresh} = 0.5$

The outcome of the decision rule above can also be illustrated graphically in a volcano plot. In Figure 5.6, such a plot is displayed for each of the four models. In these graphs, the x axis represents the estimated log fold change (here the median of the posterior distribution), while the y axis represents the posterior probability of being differentially abundant, as assessed by the model. The red dots are UPS proteins (truly differentially abundant), while the blue dots are non UPS proteins (truly non differentially abundant). Therefore, an ideal model would assign all red points in the top right corner, near ($x = 1.575$; $y \approx 1$) and all blue points near the origin.

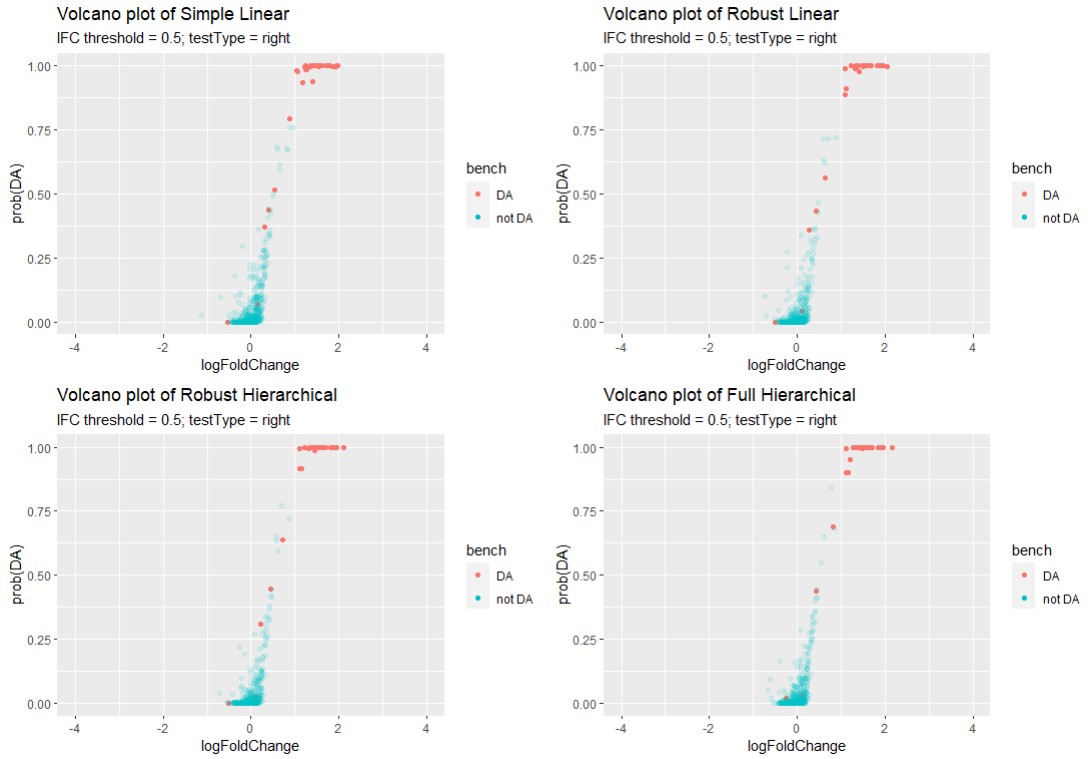


Figure 5.6: Volcano plots for the four competing models, for the B vs. A contrast, and with $lFC_{thresh} = 0.5$

Bottomline, Table 5.9 and Figure 5.6 lead to the same observations :

- All models seem to perform remarkably well, as for most probability threshold levels, the numbers of false positive and false negative are very low compared to the total number of proteins - 1347, of which 37 are differentially abundants. Even the simple linear model already performs well. This is probably due to the low variability of the CPTAC dataset samples. Indeed, as mentioned in Section 2.3, samples do not present here any biological variability, but only technical variability.

- There seems to be a small, but visible and consistent classification performance gain when using the Full Hierarchical model, compared to the models that do not consider the link between missingness and low intensity. As is suggested from Table 5.9, this gain can be quantified as having about 2 additional proteins that are correctly classified, among the total set of 1347 proteins.

In order to illustrate the later point, a deeper look is taken at the results concerning two such 'borderline' proteins, that are differentially classified in Table 5.9, when focusing on the column with a probability mass threshold of 60%. The posterior distributions obtained for the B vs. A contrast, for these two proteins, are provided in Figure 5.7 and Figure 5.8. In these figures, a comparison is done between the Full Hierarchical model, and the Robust Hierarchical model, which does not model the missingness mechanism. On each histogram, a red dashed vertical line has been added, showing the log-2 fold change threshold used in

the classification rule, i.e. 0.5% Basically, this means that the protein at hand is classified as differently abundant if the probability mass on the right of this level is greater than 60%.

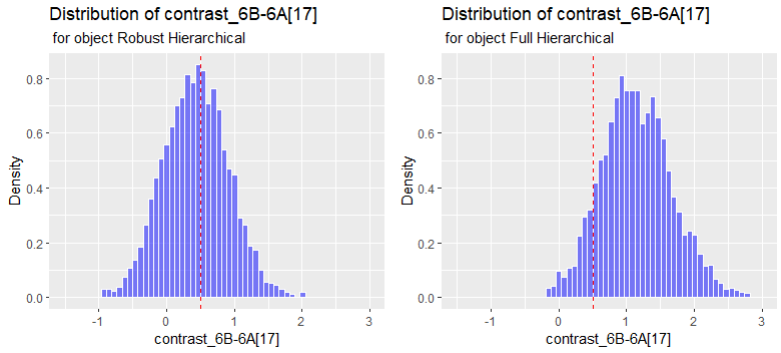


Figure 5.7: Comparison of posterior distributions obtained for the B vs. A contrast, between Robust Hierarchical and Full Hierarchical models, for protein P00441ups

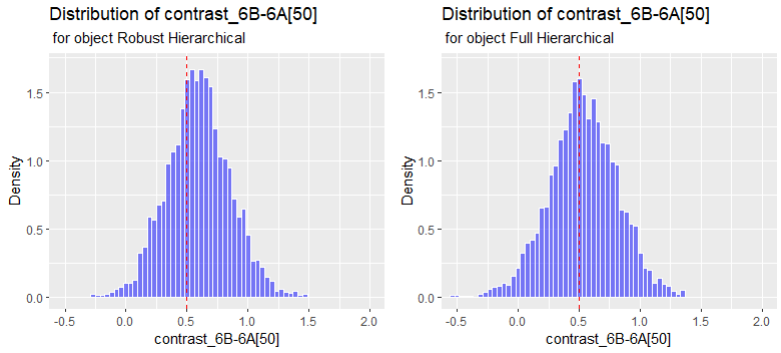


Figure 5.8: Comparison of posterior distributions obtained for the B vs. A contrast, between Robust Hierarchical and Full Hierarchical models, for protein P02294

In Figure 5.7, the considered protein, P00441ups, is a UPS protein. The Full Hierarchical model correctly identifies it as differentially abundant, while the other models obtain a false negative (considering a mass probability threshold of 60%). Investigating the posterior distributions, one can clearly notice that this is due to an improvement of the location, which is nearer the true value (1.575). The data pertaining to this protein reveal that this UPS protein was identified through only 5 peptides, with 66% of missing data, and actually there were much more missing data for condition A than for condition B. Therefore, the γ_y parameter of the full hierarchical model, which is positive and significant (0.64 ± 0.13) leads to a positive correction of the estimated log fold change.

In Figure 5.7, the considered protein, P02295, is not a UPS protein. The Full Hierarchical model does not identify it as non differentially abundant, while the other models obtain a false positive (again considering a mass probability threshold of 60%). Here, the posterior distributions are more similar to each other, and it is a combination of slightly lower location and slightly higher dispersion (0.53 ± 0.28 vs. 0.59 ± 0.26) that leads to avoiding a false

positive for the Full Hierarchical model.

In Appendix F.2, similar tables and volcano plots as in Table 5.9 and Figure 5.6 are provided for $lFC_{thresh} = 1.0$ and for the other contrast of interest (C vs. B). The observations that can be made from these additional results are in general similar to the ones discussed above, although these alternative choices of threshold parameter and contrast can sometimes lead to less discriminative results between models.

5.5.5 Discussion

The results obtained in Section 5.5.3 and Section 5.5.4 with the Full Hierarchical model, implemented in the two-step empirical Bayes procedure, are encouraging :

- in terms of RMSE, the Full Hierarchical model seems to show a small improvement, compared to the Simple Linear model. However, as advocated in Section 5.5.3, this improvement seems to be mostly due to the robust component of the intensity regression part of the model, and not to the modelling of the missingness mechanism.
- in terms of classification performance, which combines information from both location and uncertainty of the log fold changes, the full hierarchical model seems to show a small, but visible and consistent improvement compared to the other models, which do not take missingness into account.

However, these findings are obviously to be taken with great care, since :

- the performance improvement noticed with the Full Hierarchical model is small, related to only a couple of proteins among more than one thousand.
- the CPTAC dataset is a single instance of data realization. One would need to confirm the obtained results by a complementary simulation study. However, performing such a study is challenging, given the long run time needed by the Full Hierarchical model.
- these results would also deserve confirmation when applying the models on other type of experimental datasets, where biological variability is present.

As an important final remark of this section, there is still an open question related to the actual values of log fold change and probability mass thresholds the user should choose. Indeed, these parameters need to be set to define the classification decision rule, between differentially abundant and not differentially abundant proteins. This is obviously an important consideration when applying the statistical method on “real life” datasets, when the “truth” about differential abundance is not known. However, this topic is left for future work.

5.6 StanProt R package

As briefly mentioned in section 5.5.1, the various analysis presented throughout this work were facilitated by the implementation of the described models and methods in an R package, named *StanProt*. This allowed to benefit from the object orientation programming techniques to obtain a flexible and extendable piece of software. From a user perspective, the StanProt package offers the following advantages :

- it takes as input data, a standard object of the class *SummarizedExperiment*, implemented in the R package of the same name[1]. The latter is a widely used package of the *Bioconductor* platform[2], which is a popular platform in the Bioinformatics community. The necessary pre-processing steps to transform a SummarizedExperiment object into the internal model data inputs are performed automatically when calling methods of the package.

- it encapsulates interfaces to the various inferential engines used in this work : Stan NUTS, Stan MAP (Maximum A Posteriori) that was used for Laplace approximations, as well as the ad hoc implemented MFVB approximate inference algorithms. As a result, these engines are exposed to the user in a seamless way, as the package provides common ways to define inputs and retrieve outputs.

- it allows flexible tuning of the model specifications in different respects :

  - the user provides R style synthetic formulas to specify the chosen regressors in the intensity and missingness parts of the model, as in (4.6) and (4.7). It is therefore straightforward to e.g. add or remove a regressor, to replace a fixed effect by a random effect, etc. without having to amend material parts of the user analysis program.
  - additionally most ingredients of the model can be activated or de-activated using dedicated flags : robust regression, modelling of (NMAR or MCAR) missingness, hierarchical priors for the γ or σ^2 type of parameters, etc.

- it provides detailed outputs in the form of summaries or samples of the posterior distributions of the various model parameters, or user specified contrasts on the α parameters. Also, different plot types can be obtained using dedicated methods of the package. Indeed, most of the plots presented in Chapter 4 and Chapter 5 (histograms, volcano plots, ...) were implemented in the StanProt package itself.

[1] https://doi.org/doi:10.18129/B9.bioc.SummarizedExperiment
[2] https://bioconductor.org/about/

Chapter 6

Conclusions

Analysing Mass Spectrometry based proteomics experimental data is a demanding task. Indeed, as was pointed out in Chapter 1, these data possess intrinsic characteristics that are inherent to the technology used to produce them, which in turn lead to serious challenges for the statistician. Among these challenges are the high proportion of missing data, with a missingness process that is Not Missing At Random (NMAR), and the fact that the quantities of interest - protein abundances - are not obtained directly but need to be estimated from measurable quantities, i.e. peptide intensities. In bioinformatics pipeline settings, these two challenges are often addressed by implementing pre-processing steps, i.e. imputation and summarization, which come with their benefits but also drawbacks, as was pointed out in Chapter 1. Therefore, the research question motivating this work was to assess whether Bayesian statistical modelling could constitute an alternative direction to address these data challenges, both from a theoretical and practical point of view.

In Chapter 2, some litterature review was done, and it appeared that several recent articles had handled similar proteomics data challenges in a variety of ways, either within a Bayesian or a frequentist paradigm, but without necessarily addressing all these challenges at once. This review however highlighted a number of specific ideas, or model ingredients, that were coming back regularly. Therefore, a list of such key ingredients was constructed, as a deliverable of this litterature review. Moreover, as a by-product, a suitable benchmark dataset - the CPTAC dataset - was selected, from these mentioned in the litterature, in order to support the subsequent testing of the implemented models.

Then, in order to speed-up the implementation work of MCMC algorithms, decision was made to use the Stan plateform. Hence, Chapter 3 provided a summary of the key elements of this plateform, and introduced the Hamiltonian Monte Carlo algorithm, which is the main pillar of the Stan inferential engine.

In Chapter 4, a single protein model, modelling both intensity and missingness at peptide level, was developed and tested on the CPTAC dataset. Some initial runs of the model using HMC, with Stan NUTS simulation engine, reported two main issues, namely the long CPU execution time, and a model identifiability problem for some of - but not all - proteins. In an attempt to improve the long run times, several computationally efficient techniques to approximate the posterior distributions were investigated, i.e. some Laplace approxima-

tions and some Mean Field Variational Bayes (MFVB) approximations. However, with both type of approximations, systematic underestimation of the posterior dispersion, o.a. for the main parameters of inferential interest, was observed. Therefore, it was not possible to obtain an approximation method that was "good enough" for straightforward use in inference. Nevertheless, along the road, the MFVB distributional approximation method, in its specific implementation to the single protein model, proved to be quick in execution time, and showed reliability when used for posterior mode finding.

In order to address the model identifiability issue reported for the single protein model, a hierarchical model extension was motivated and developed in Chapter 5. Unfortunately, adding a hierarchical set-up, hence the need to run the model on all proteins at once, lead to a very high dimensional model parameters space, hence to intractable execution times for the Stan HMC engine. Therefore, an ad hoc MFVB approximate inference algorithm was also developed for the hierarchical version of the model. The results obtained with the MFVB approximation allowed to motivate the idea of a two-step empirical Bayes procedure, with the aim to come up with tractable execution times, while greatly limiting the posterior dispersion underestimation. Finally, a short model comparison study was done, and showed that the proposed approach, when applied to the CPTAC dataset, was slightly but consistently more performant, in terms of protein classification accuracy, than more simple model counterparts.

Bottomline, the high level conclusion of this thesis is that Bayesian statistical models do possess the potential to address the various data challenges conveyed by proteomics data. In particular, they have the required flexibility to model complex data generation processes as these encountered in proteomics, like for Label Free MS based quantitation. They also have the ability to faithfully render the accumulation of uncertainty inherent to sequential multi-step experimental designs. However, their complexity lies in their practical implementation, as obtaining samples from a huge dimensional multivariate distribution is challenging, and model identifiability problems can occur. In this work, these issues were partly addressed by proposing an ad hoc two-step approach, combining posterior distribution approximations in a hierarchical model and empirical Bayes estimation of the hyper-parameters. However, there is still room for additional work, both for improving the remaining CPU execution time, and for confirming the validity and benefits of the approach on other datasets.

In terms of future research, there are several possible directions :

- In order to further decrease the CPU execution time, some additional posterior distribution approximation techniques would certainly deserve some investigation. For instance, one can mention *integrated nested Laplace approximations* ([Rue, Martino, and Chopin 2009]).

- In order to benchmark the implementation of the MFVB approximation for the hierarchical model, it would certainly be good to have an exact MCMC implementation of the model. However, as mentioned in Section 5.2, it was not possible to obtain a conclusive run of the hierarchical model on a meaningful number of proteins, using the Stan NUTS engine. Therefore, it could be beneficial to use the expressions obtained for the full conditionals of the model parameters (Section 5.3.1), as a starting point for the implementation of an ad hoc Gibbs MCMC sampler for this model.

- Also, to comfort the validity of the proposed approach, further investigation should be made about the influence of the number of peptides threshold introduced in Section 5.4, for example by further analysing the sensitivity of the statistical results to this threshold, for different datasets.
- In terms of scope extension, it would certainly be interesting to test the proposed approach on other datasets, possessing true biological variability. Moreover, the model could also be easily extended to handle other type of experimental designs e.g. Labelled MS quantitation, affinity purification experiments, etc.
- Finally, as mentioned at the end of Section 5.5, in order to use the proposed approach as a built-in statistical method, readily available to analyse different proteomics datasets, specific attention should be devoted to the choice of the threshold parameters involved in the classification of proteins between differentially abundant and not differentially abundant.

Appendix A

Ridge regression and link with linear mixed models

In this appendix, one recalls the concept of ridge regression in linear regression models, and shows a link between ridge regression and linear mixed models.

A.1 Ridge estimator in linear regression models

Consider the linear regression model :

$$\boldsymbol{y} = X\boldsymbol{\beta} + \boldsymbol{\epsilon} \tag{A.1}$$

where : $\boldsymbol{y}(n \times 1)$ is the response vector
$X(n \times p)$ is a know matrix of predictor variables
$\boldsymbol{\beta}(p \times 1)$ is a vector of unknow coefficients
$\boldsymbol{\epsilon}(n \times 1) \sim \mathcal{N}(0, \sigma^2 I)$

The *Ordinary Least Squares* (OLS) estimator :

$$\hat{\beta} = (X^T X)^{-1} X^T \boldsymbol{y} \tag{A.2}$$

is obtained by minimizing :

$$(\boldsymbol{y} - X\boldsymbol{\beta})^T (\boldsymbol{y} - X\boldsymbol{\beta}) \tag{A.3}$$

The *ridge* estimator :

$$\hat{\beta}_{ridge} = (X^T X + K)^{-1} X^T \boldsymbol{y} \tag{A.4}$$

where K is a diagonal matrix with positive diagonal elements $\lambda_j \geq 0$ for $j : 1, \ldots, p$,
is obtained by minimizing (*Penalized Least Squares*) :

$$(\boldsymbol{y} - X\boldsymbol{\beta})^T (\boldsymbol{y} - X\boldsymbol{\beta}) + \sum_{j=1}^{p} \lambda_j \|\beta_j\|^2 \tag{A.5}$$

Possible motivations for using the ridge estimator, instead of the OLS estimator, are the following :

- to provide an estimator even when $n < p$, or in other cases of multicollinearity ($\text{rank}(X) < p$).
- to regularize, i.e. to shrink the β_j's towards 0 to avoid overfitting (better stability). In particular, if the number of observations, n, is small, the ridge estimator will perform more regularization.
- $\hat{\beta}_{ridge}$ is biaised but might have a smaller *Mean Squared Error* (i.e. $\mathbb{E}(\hat{\boldsymbol{\beta}} - \boldsymbol{\beta})^2$) than the *OLS* estimator.

Note that an issue arising when using the ridge estimator is how to choose adequately the penalty coefficients λ_j's (hyper-parameters).

A.2 Link between ridge regression and linear mixed models

Consider the linear mixed model :

$$\boldsymbol{y} = X\boldsymbol{\beta} + Z\boldsymbol{u} + \boldsymbol{\epsilon}, \text{ with } \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{\epsilon} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \sigma_u^2 I & 0 \\ 0 & \sigma_\epsilon^2 I \end{bmatrix} \right) \tag{A.6}$$

where : $X(n \times p)$ is a know design matrix for the fixed effects
$Z(n \times m)$ is a know design matrix for the random effects

One can show ([Wand 2003], [Muñoz Maldonado 2009]) that, for given values of σ_u^2 and σ_ϵ^2, computing the Best Linear Unbiaised Estimator ($BLUE$) of $X\boldsymbol{\beta} + Z\boldsymbol{u}$ is equivalent to solving the penalized least squares problem :

$$\begin{bmatrix} \hat{\boldsymbol{\beta}} \\ \hat{\boldsymbol{u}} \end{bmatrix} = \underset{\beta,u}{\text{argmin}}(\|\boldsymbol{y} - X\boldsymbol{\beta} - Z\boldsymbol{u}\|^2 + \lambda\|u\|^2), \text{ where } \lambda \equiv \frac{\sigma_\epsilon^2}{\sigma_u^2} \tag{A.7}$$

Therefore, there exists a direct link between :

- Ridge regularization of some β coefficients in a standard linear model setting, and,
- Having the related effects defined as random and normally distributed in a linear mixed model setting.

This link is effective, when the λ_j coefficients of the ridge regression are taken to be the same as in eq. (A.7).

Appendix B

Robust estimation and link with Student-t likelihood

In this appendix, one introduces the class of robust M-estimators, and shows a conceptual link between robust M-estimation and Student-t likelihood.

B.1 Robust M-estimators

Note that the content of this section is adapted from [Croux and Dehon 2014], and [Venables and Ripley 2002].)

In order to introduce robust M-estimators, consider the M-estimation of a location parameter. Assume univariate observations $x_1, \ldots, x_n$ generated as $x_i = \mu + \sigma\epsilon_i$, such that the ϵ_i's are *i.i.d.* with as probability density function $f_\epsilon(\cdot) = f_0(\cdot)$. Supposingly, one would like to fit a statistical model to these data, and let $f_{\mu,\sigma}$ be the chosen model distribution (e.g. $\mathcal{N}(\mu, \sigma)$). In particular, the objective is to find a robust estimate of the location parameter μ .

The class of M-estimators contains the Maximum Likelihood (*ML*) estimator, $\hat{\mu}_{ML}$, as a special case. Recall that, when calculating $\hat{\mu}_{ML}$, the first order condition is given by :

$$\sum_{i=1}^{n} \psi_{ML}\Big(\frac{x_i - \hat{\mu}_{ML}}{\sigma}\Big) = 0, \text{ with } \psi_{ML}(u) = \frac{-f_0'(u)}{f_0(u)} \text{ (score function)} \tag{B.1}$$

Note that in the specific case of the normal distribution : $\psi_{ML}(u) = u$, leading to $\hat{\mu}_{ML} = \bar{X}$ (i.e. the sample mean).

Without being formal here[1], let us refer to the *robustness* of an estimator, as the property of not being too sensitive to the presence of outliers. Then, it is well known that :

- On the one hand, ML estimators are (asymptotically) efficient (under some conditions), but are often not robust. One example of such non robust ML estimators is the sample mean.

[1] Robustness metrics can be formally defined, see e.g. ([Croux and Dehon 2014]).

- On the other hand, robust estimators can have poor relative efficiency (e.g. 64% efficiency for the median as a location estimator for the normal distribution).

A M-estimator of location, $\hat{\mu}$, is defined as the solution of the equation :

$$\sum_{i=1}^{n} \psi\Big(\frac{x_i - \hat{\mu}}{\hat{\sigma}_0}\Big) = 0 \tag{B.2}$$

with ψ any non decreasing, odd function, and $\hat{\sigma}_0$ set to a robust estimate of σ, for example the Median Absolute Deviation (MAD).

A common choice for ψ is the Huber ψ function, defined as :

$$\psi_b(u) = \max(\min(u, b), -b)$$

with $b = 1.345$ giving a 95% relative efficiency for the M-estimator applied to the normal distribution.

Note that eq. (B.2) can also be written as :

$$\sum_{i=1}^{n} w_i\Big(\frac{x_i - \hat{\mu}}{\hat{\sigma}_0}\Big) = 0, \text{ with } w_i = \psi\Big(\frac{x_i - \hat{\mu}}{\hat{\sigma}_0}\Big) / \Big(\frac{x_i - \hat{\mu}}{\hat{\sigma}_0}\Big) \tag{B.3}$$

which is equivalent to the MLE equation (B.1) in the normal distribution case, but with unequal w_i weights (e.g. Huber weights) assigned to each observation.

Figure B.1 illustrates the shape of the ψ function for four commonly used M-estimators.

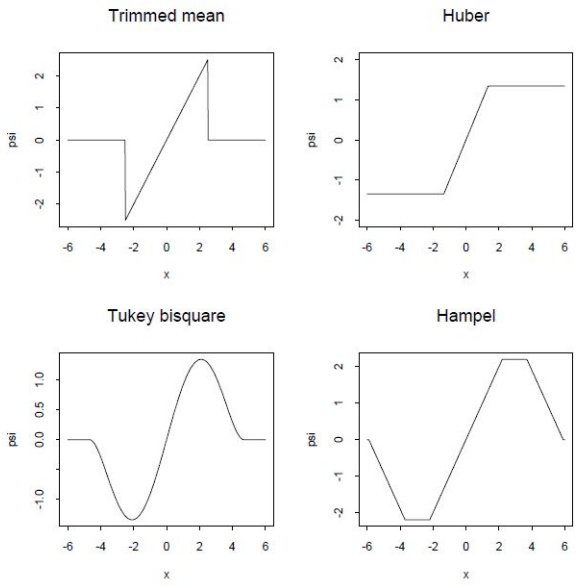


Figure B.1: ψ functions for four common M-estimators.

Eq. (B.3) suggests an iterative implementation to find $\hat{\mu}$:

- Start from a chosen initial $\hat{\mu}$, for example the median
- Repeat the following 2 steps until convergence:
  1. update $\hat{\sigma}_0$ and the Huber weights w_i's as a function of the current value $\hat{\mu}$
  2. update $\hat{\mu}$, as the weighted sample mean, using the newly computed w_i's.

As a final remark, one can mention that it can be shown (see e.g. [Venables and Ripley 2002]) that the robust estimation idea can be easily extended to the linear regression case, i.e. when $\mu = X\boldsymbol{\beta}$, and in that case :

- $\hat{\sigma}_0$ is then a robust estimator of the scale parameter of the residuals;
- the above described iterative estimation procedure is then implemented as an iterative least square ($IWLS$) algorithm.

B.2 Link between M estimators and Student-t likelihood

Coming back to the initial problem of Section B.1, one now chooses, instead of the normal distribution, the Student-t as model distribution :

$$f_{\mu,\sigma,\nu}(x) = \frac{\Gamma((\nu+1)/2)}{\Gamma(\nu/2)\sqrt{\nu\pi}\sigma}\left(1+\frac{1}{\nu}\left(\frac{x-\mu}{\sigma}\right)^2\right)^{-(\nu+1)/2} \tag{B.4}$$

Applying eq. (B.1), one can easily show in that case that the MLE for μ, $\hat{\mu}_{ML,t}$ leads to $\psi_{ML}(u) = \frac{u}{1+u^2/\nu}$ (instead of $\psi_{ML}(u) = u$ for the normal distribution).

Figure B.2 shows graphically the shape of the obtained ψ function and weight function, for different values of ν of the Student-t distribution. The Huber ψ and weight functions are also plotted on the same graph.

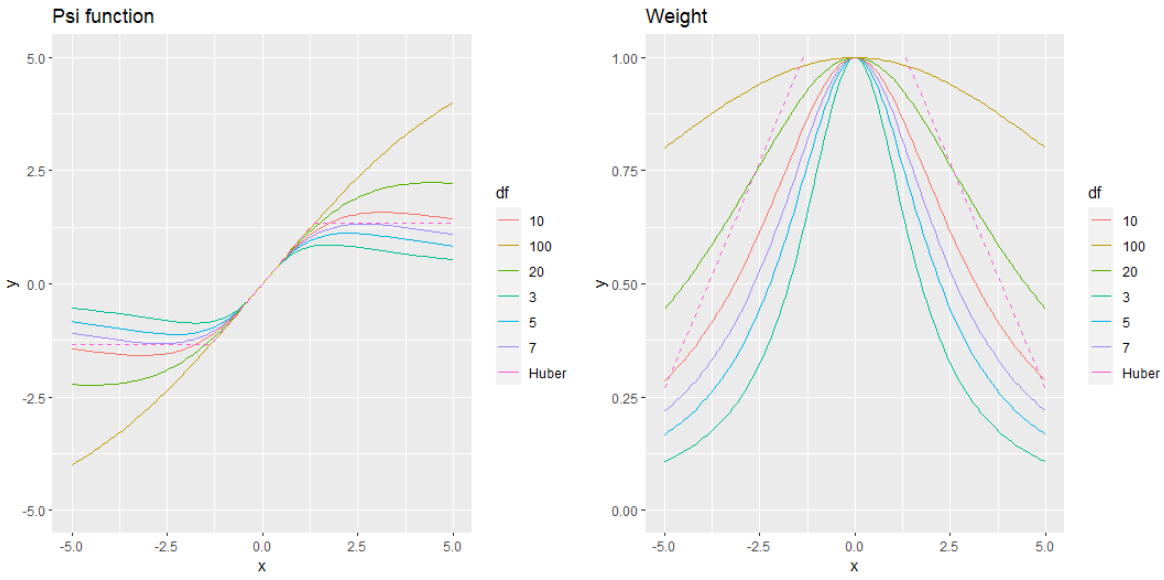


Figure B.2: ψ and weight functions for the $\hat{\mu}_{ML,t}$ estimator.

The above graph implies that one can view the use of the Student-t distribution, instead of the Normal distribution, as a way to come up with a robust estimate for the location parameter μ. Indeed, the weight assigned to each observation in the calculation of the $\hat{\mu}$ estimator, decreases as these observations deviate from the average. The strength of robustness is driven by the degrees of freedom parameter, ν : the smaller the ν, the more robust the location estimator tends to be.

Appendix C

Distributional approximation : results for the single protein model

C.1 Approximated posterior distributions with Laplace approximations

In this section, one provides the full set of graphs that compare the obtained marginal posterior distributions for various parameters, using the Laplace approximations described in Section 4.4 of the main text, to the exact marginal posterior distributions obtained with Stan NUTS.

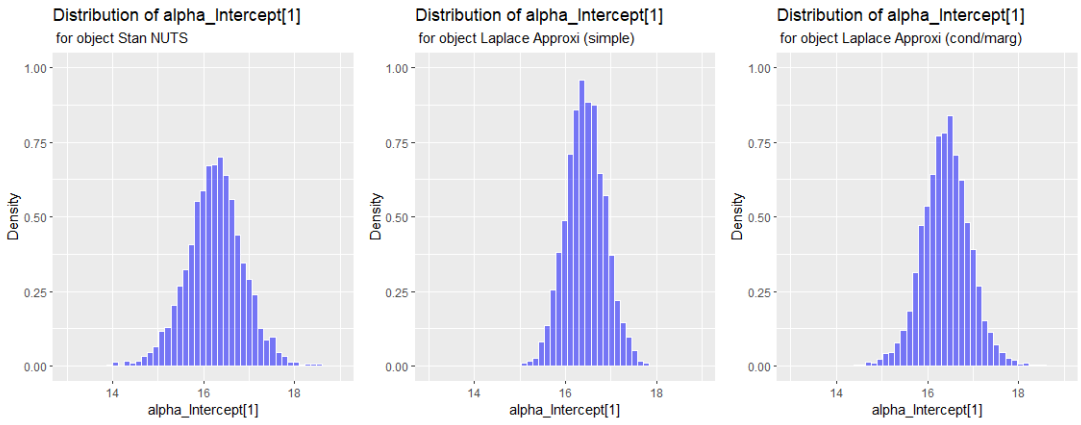


Figure C.1: Comparison of marginal posterior distributions for $\alpha_{intercept}$

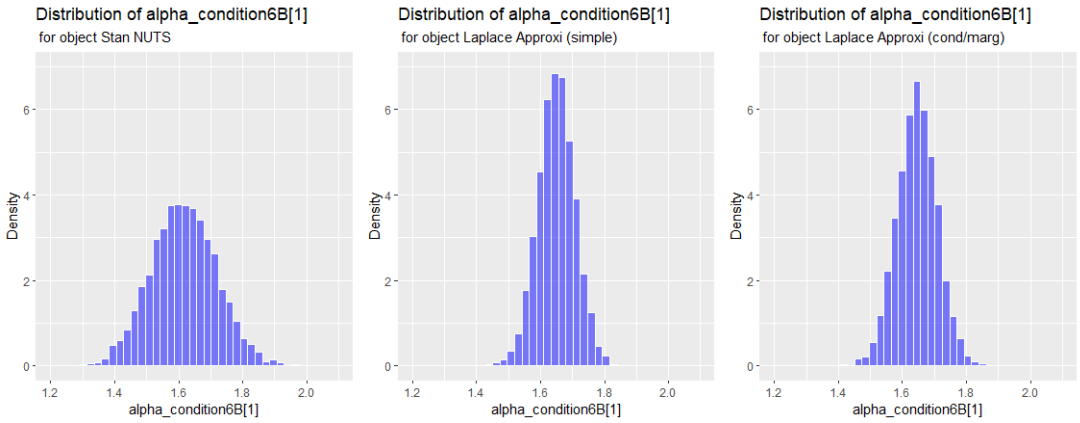


Figure C.2: Comparison of marginal posterior distributions for α_{cond6B}

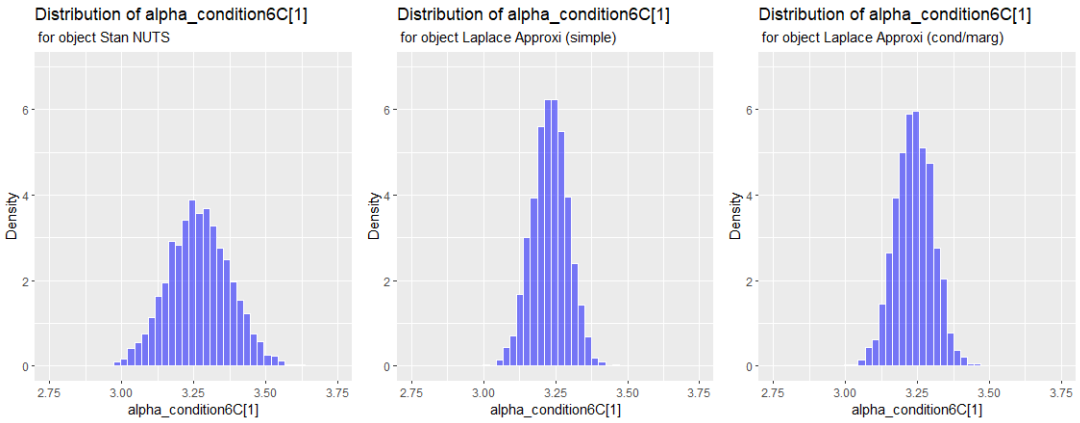


Figure C.3: Comparison of marginal posterior distributions for α_{cond6C}

Figure C.4: Comparison of marginal posterior distributions for α_{lab2}

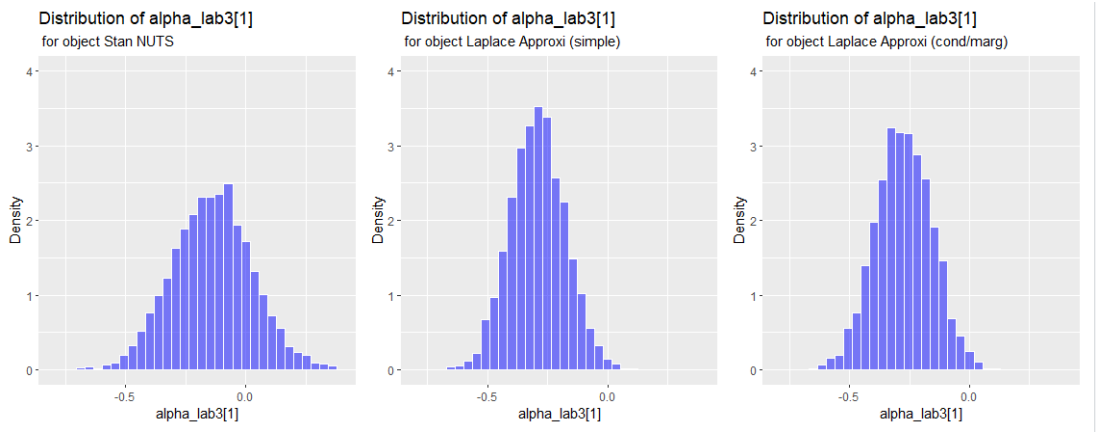


Figure C.5: Comparison of marginal posterior distributions for α_{lab3}

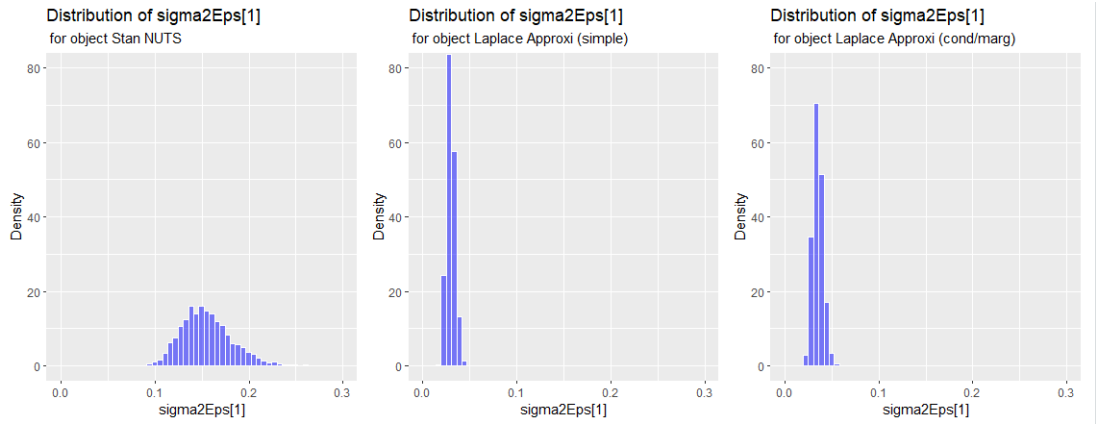


Figure C.6: Comparison of marginal posterior distributions for σ^2_ϵ

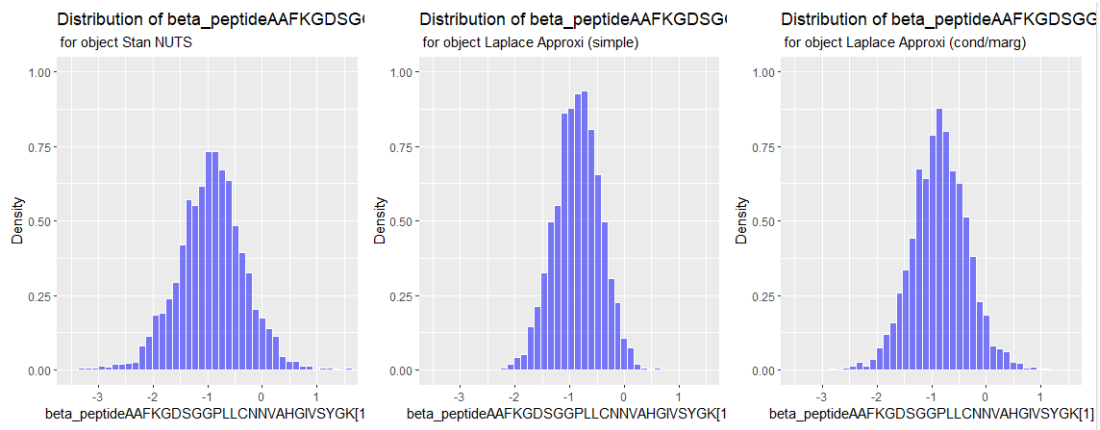


Figure C.7: Comparison of marginal posterior distributions for $\beta_{peptide1}$ (one example of random peptide effect)

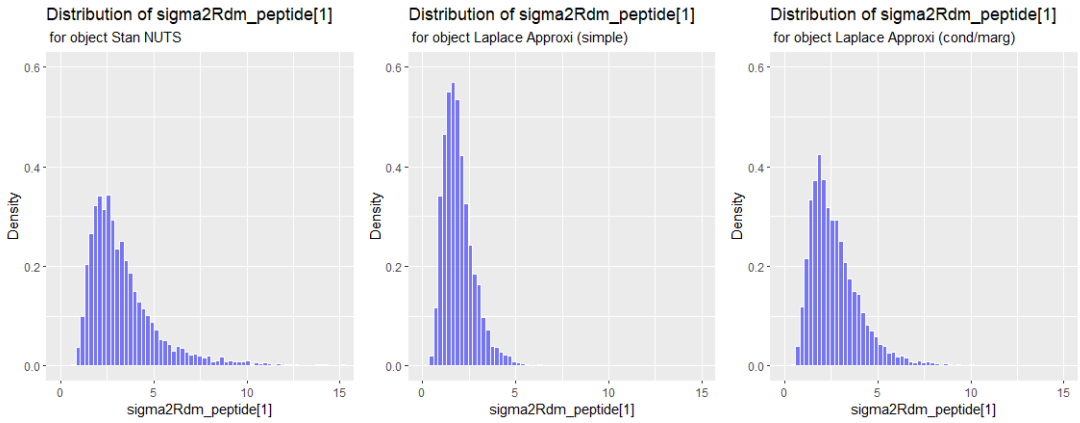


Figure C.8: Comparison of marginal posterior distributions for $\sigma^2_{rdm,peptide}$

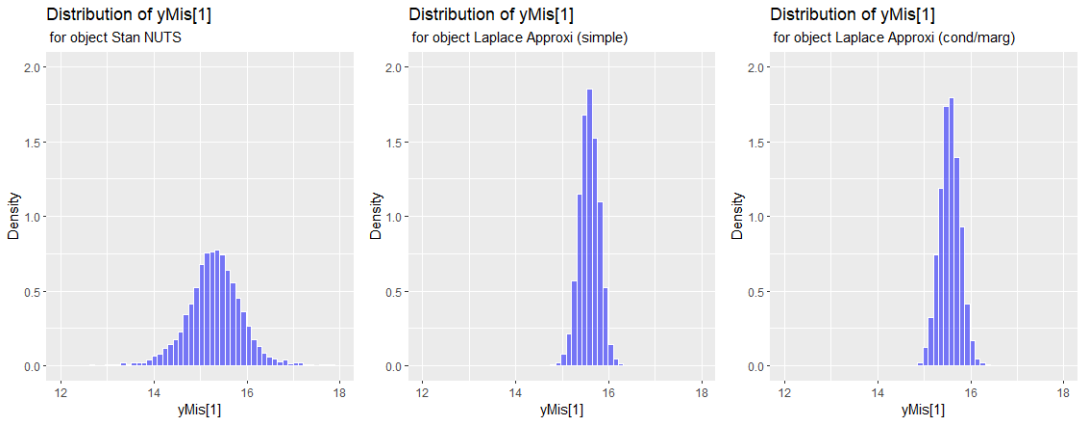


Figure C.9: Comparison of marginal posterior distributions for $ymis_1$ (one example of missing data)

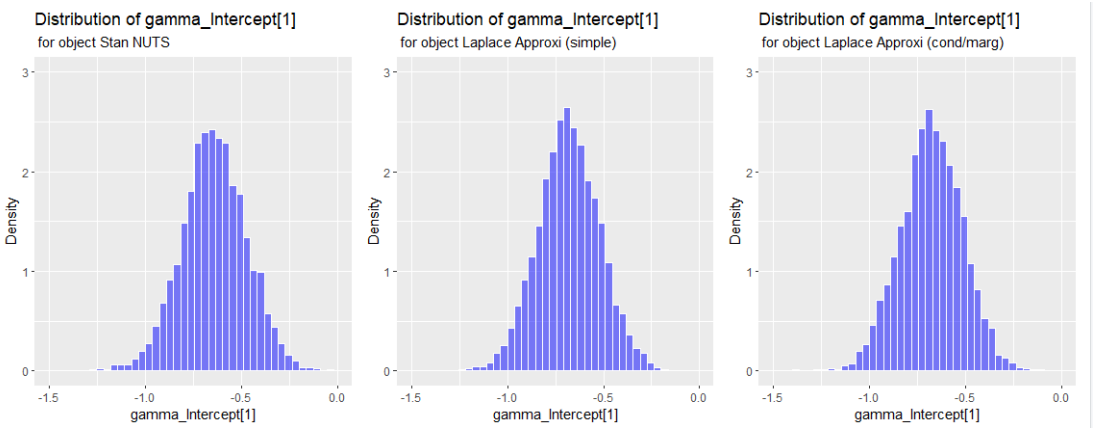


Figure C.10: Comparison of marginal posterior distributions for $\gamma_{intercept}$

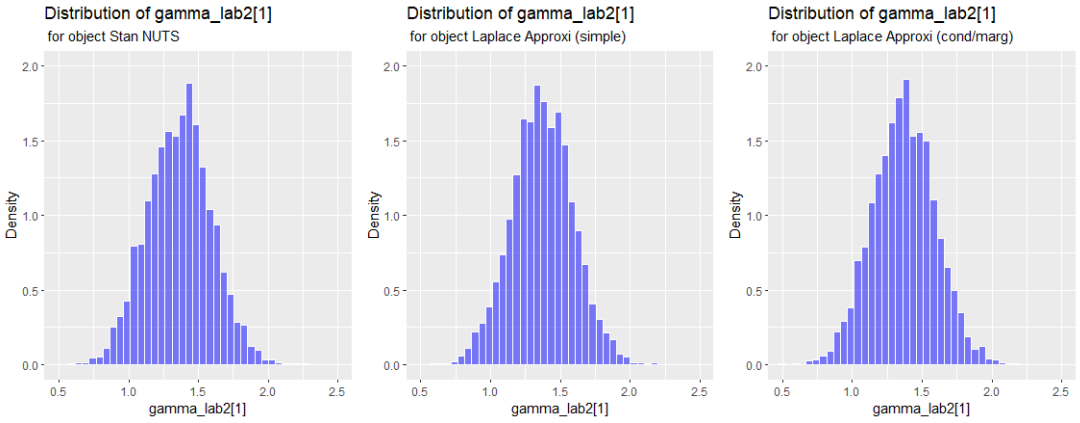


Figure C.11: Comparison of marginal posterior distributions for γ_{lab2}

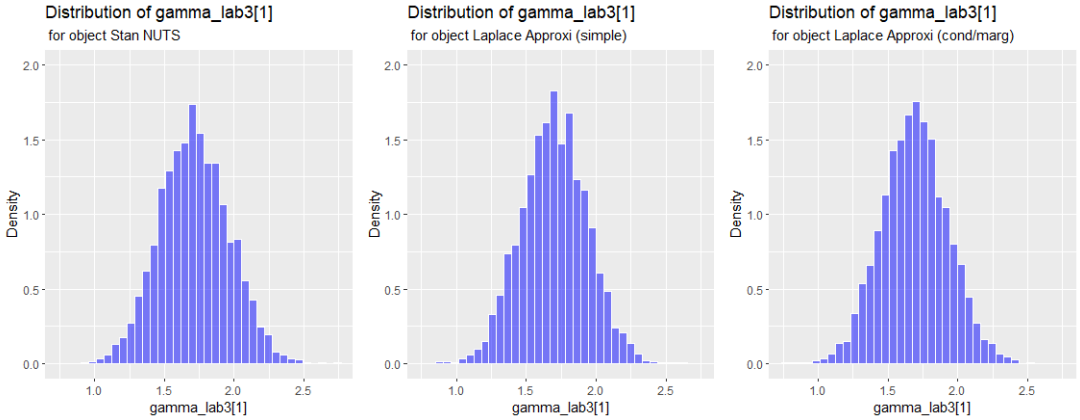


Figure C.12: Comparison of marginal posterior distributions for γ_{lab3}

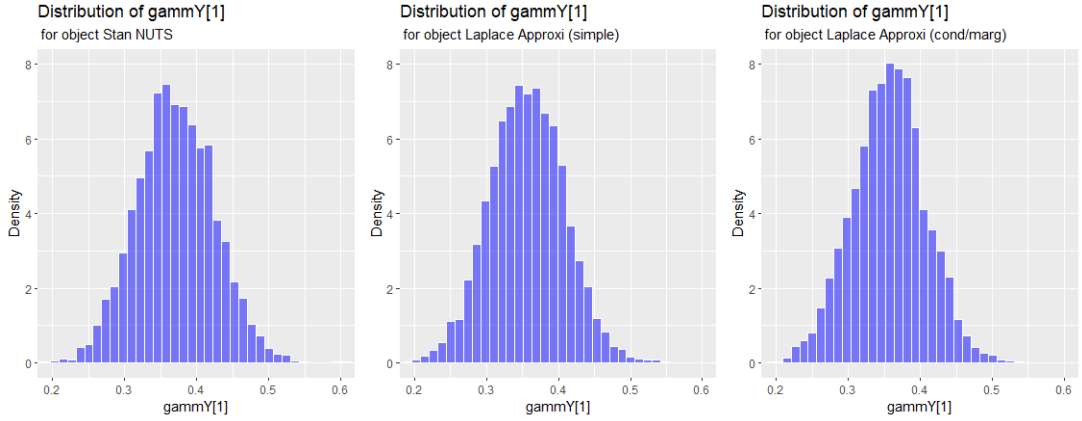


Figure C.13: Comparison of marginal posterior distributions for γ_Y

C.2 Approximated posterior distributions with MFVB

In this section, one provides the full set of graphs that compare the obtained marginal posterior distributions for various parameters, using the MFVB approximation described in Section 4.5 of the main text, to the exact marginal posterior distributions obtained with Stan NUTS. Comparison with Laplace approximation with parameters split (cf. Section 4.4) is also made.

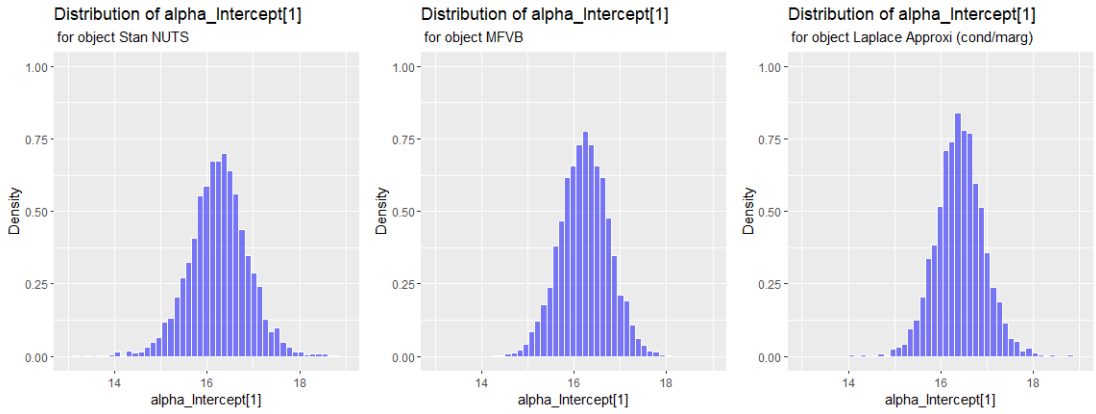


Figure C.14: Comparison of marginal posterior distributions for $\alpha_{intercept}$

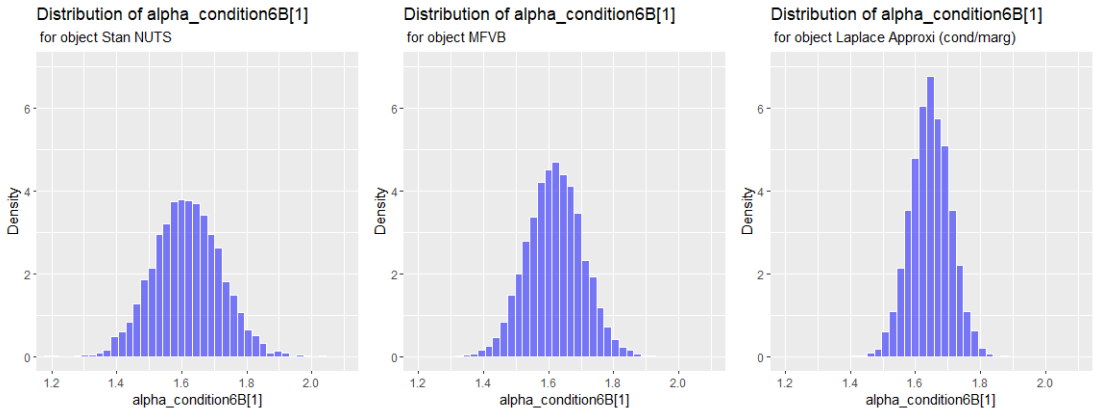


Figure C.15: Comparison of marginal posterior distributions for α_{cond6B}

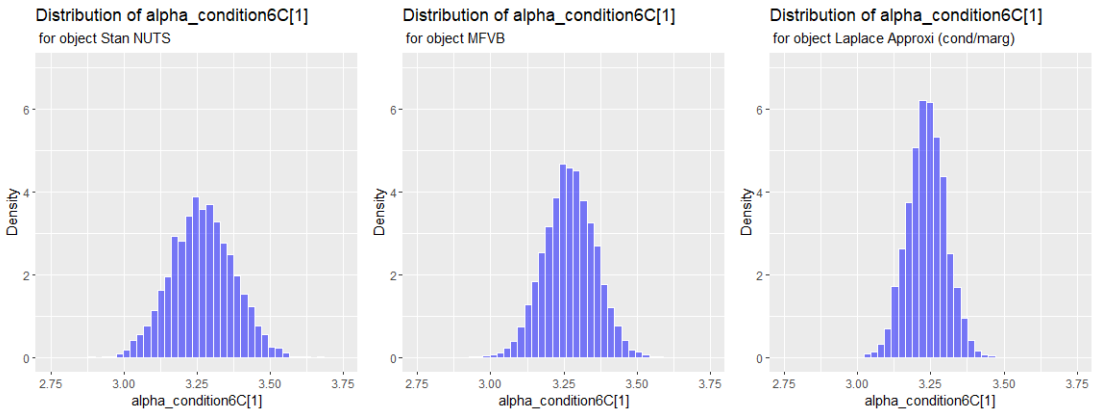


Figure C.16: Comparison of marginal posterior distributions for α_{cond6C}

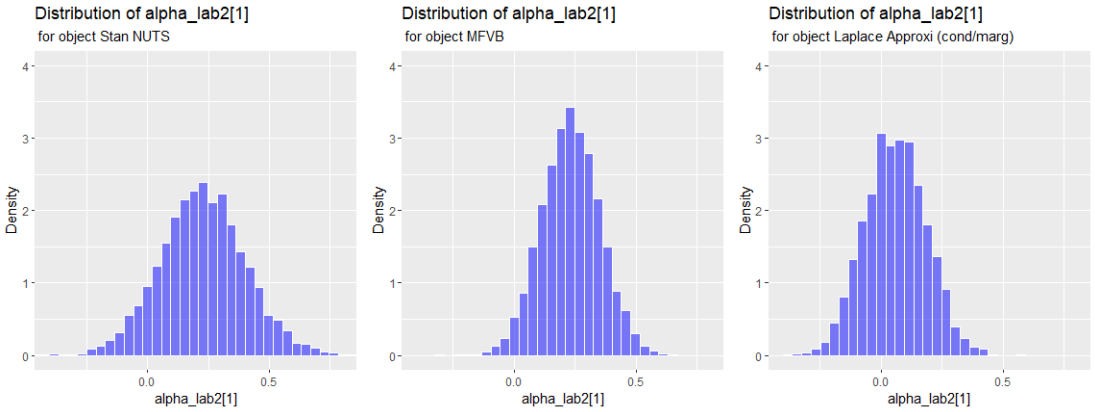


Figure C.17: Comparison of marginal posterior distributions for α_{lab2}

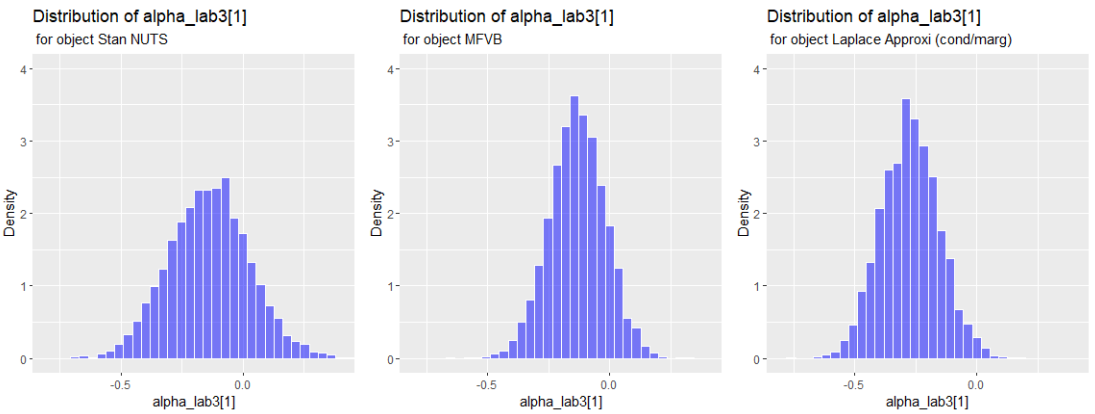


Figure C.18: Comparison of marginal posterior distributions for α_{lab3}

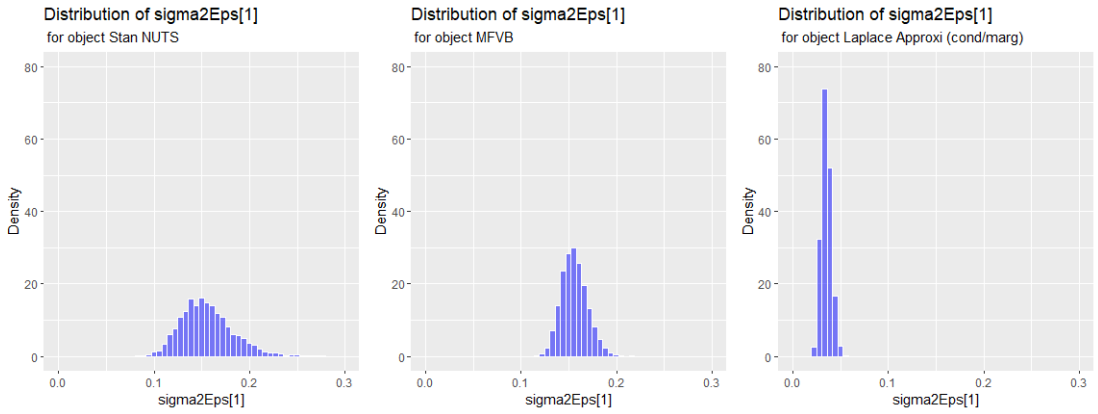


Figure C.19: Comparison of marginal posterior distributions for σ^2_ϵ

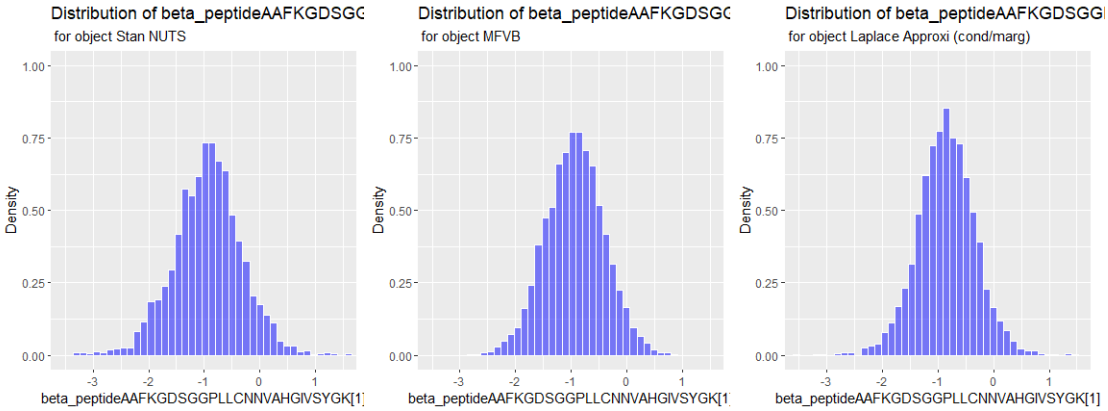


Figure C.20: Comparison of marginal posterior distributions for $\beta_{peptide1}$ (one example of random peptide effect)

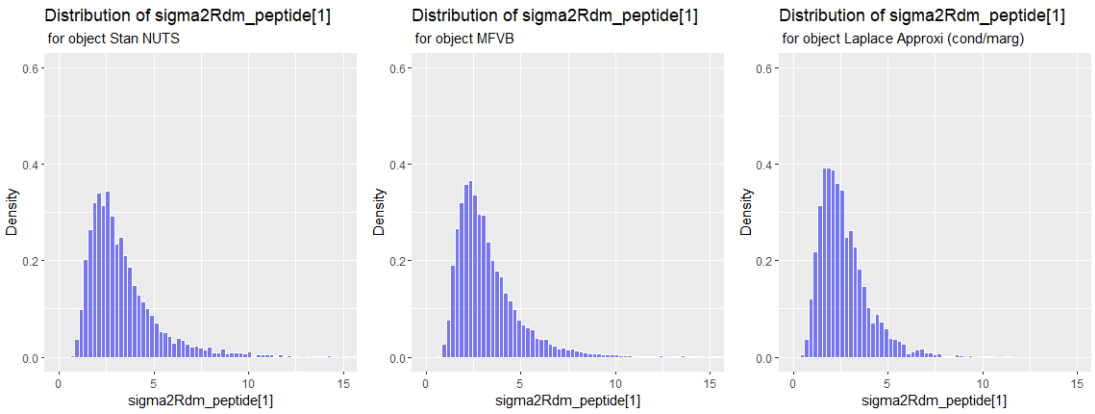


Figure C.21: Comparison of marginal posterior distributions for $\sigma^2_{rdm,peptide}$

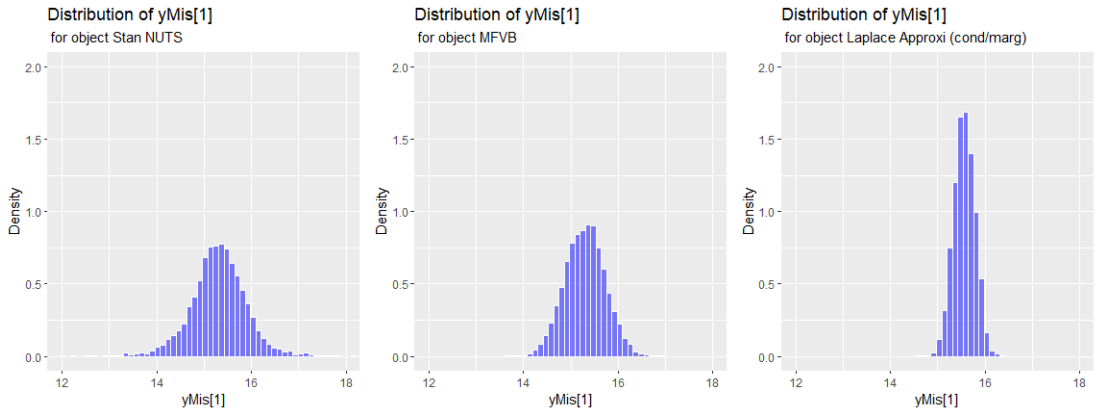


Figure C.22: Comparison of marginal posterior distributions for $ymis_1$ (one example of missing data)

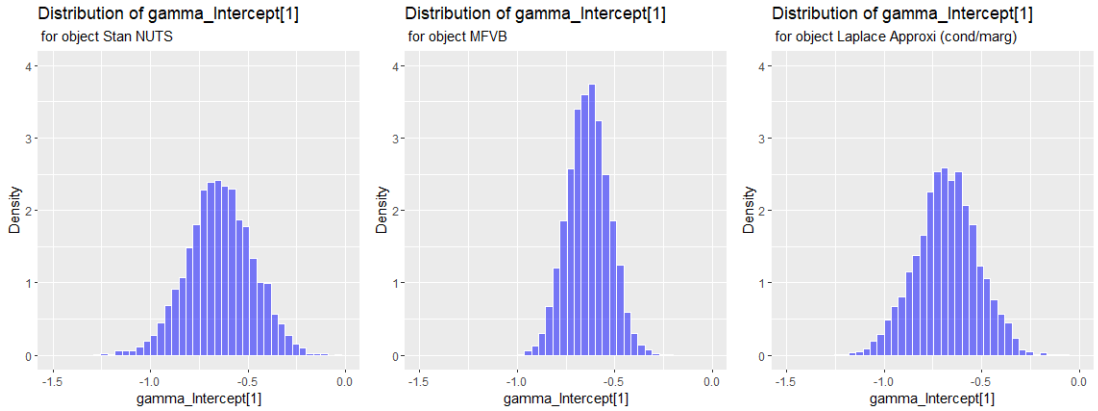


Figure C.23: Comparison of marginal posterior distributions for $\gamma_{intercept}$

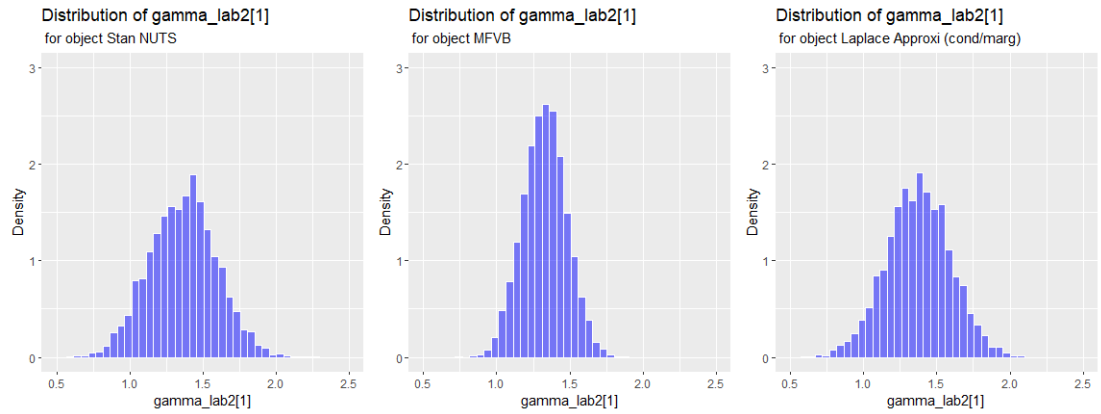


Figure C.24: Comparison of marginal posterior distributions for γ_{lab2}

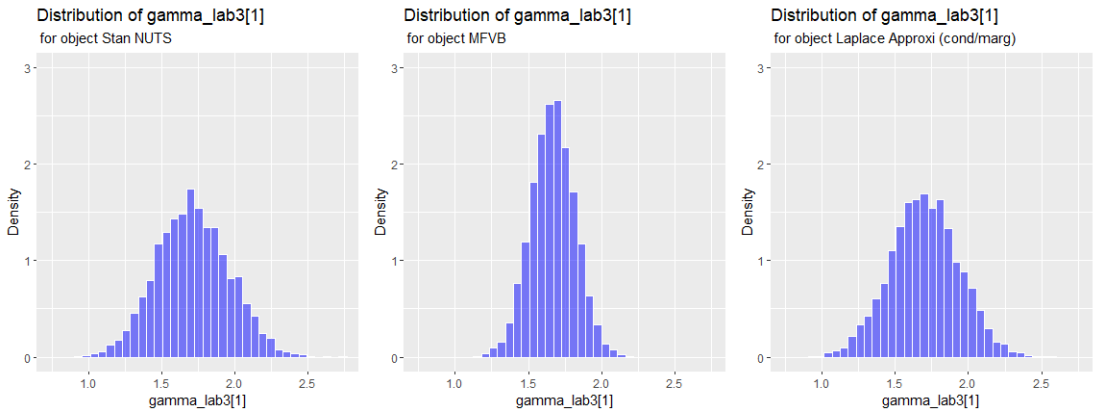


Figure C.25: Comparison of marginal posterior distributions for γ_{lab3}

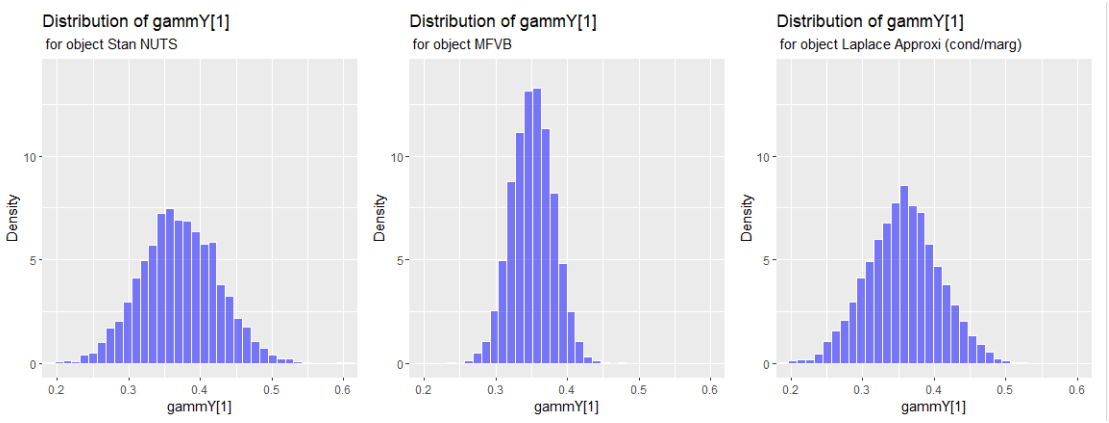


Figure C.26: Comparison of marginal posterior distributions for γ_Y

C.3 Histograms of means and standard deviations of the obtained posterior distributions

C.3.1 γ parameters obtained with non informative priors

$\gamma_{Intercept}$ parameter

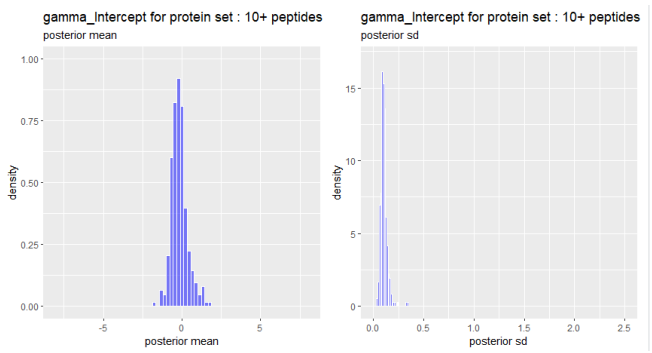


Figure C.27: Histogram of posterior means and standard deviations obtained for $\gamma_{Intercept}$, for the proteins with 10+ peptides.

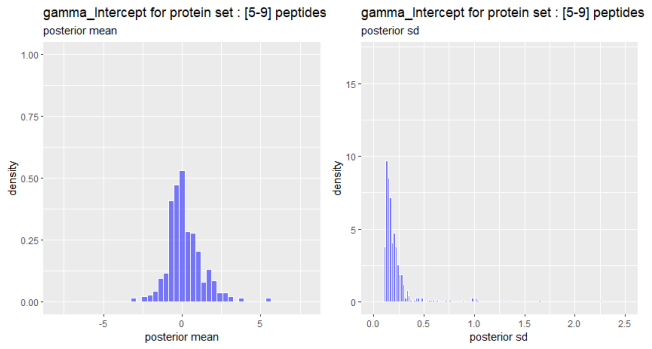


Figure C.28: Histogram of posterior means and standard deviations obtained for $\gamma_{Intercept}$, for the proteins with $[5-9]$ peptides.

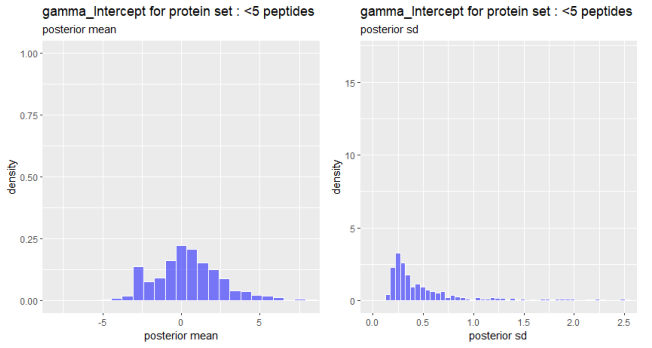


Figure C.29: Histogram of posterior means and standard deviations obtained for $\gamma_{Intercept}$, for the proteins with < 5 peptides.

γ_{lab2} parameter

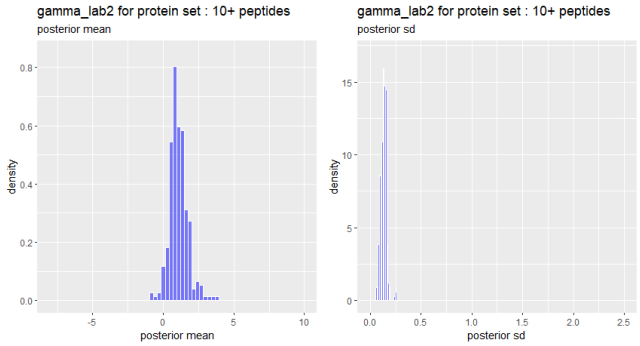


Figure C.30: Histogram of posterior means and standard deviations obtained for γ_{lab2}, for the proteins with 10+ peptides.

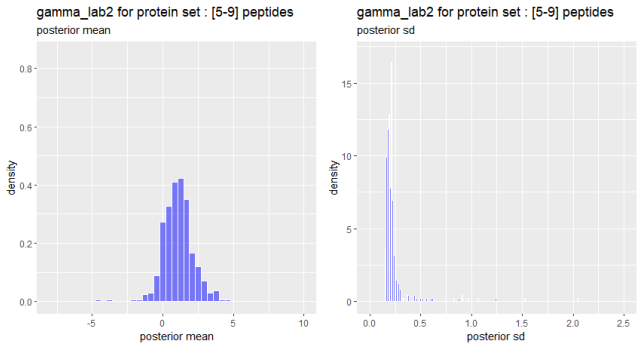


Figure C.31: Histogram of posterior means and standard deviations obtained for γ_{lab2}, for the proteins with $[5-9]$ peptides.

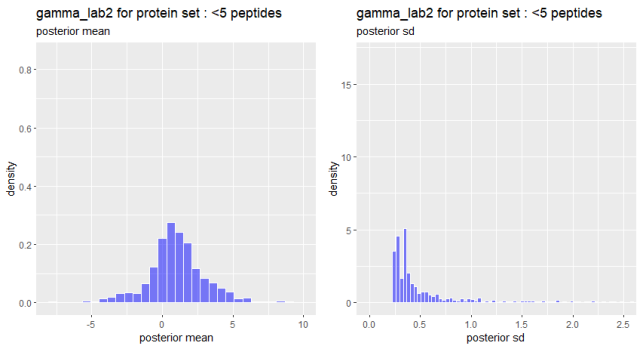


Figure C.32: Histogram of posterior means and standard deviations obtained for γ_{lab2}, for the proteins with < 5 peptides.

γ_{lab3} parameter

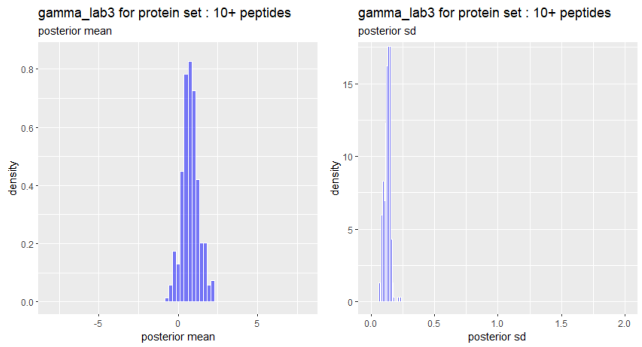


Figure C.33: Histogram of posterior means and standard deviations obtained for γ_{lab3}, for the proteins with 10+ peptides.

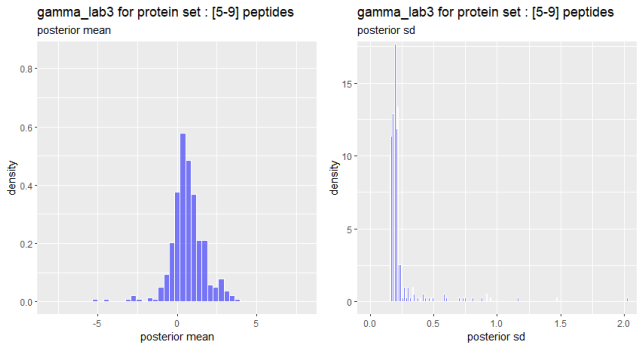


Figure C.34: Histogram of posterior means and standard deviations obtained for γ_{lab3}, for the proteins with $[5-9]$ peptides.

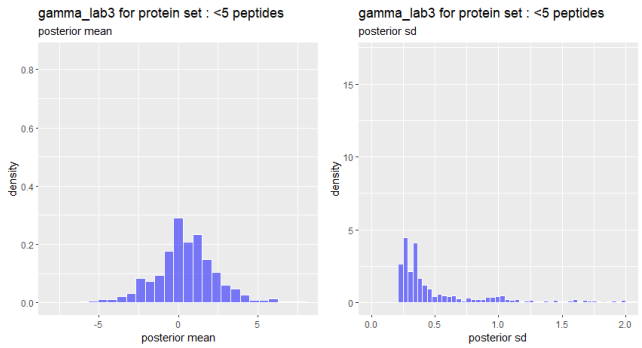


Figure C.35: Histogram of posterior means and standard deviations obtained for γ_{lab3}, for the proteins with < 5 peptides.

γ_y parameter

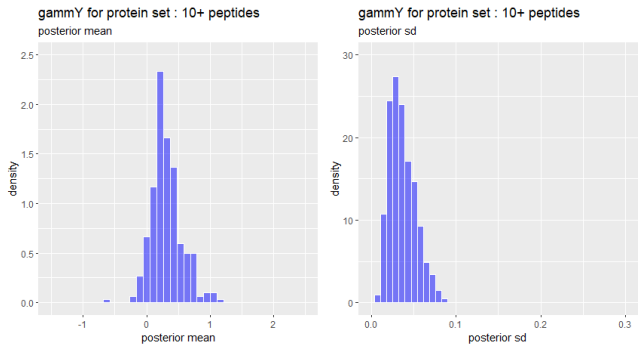


Figure C.36: Histogram of posterior means and standard deviations obtained for γ_Y, for the proteins with 10+ peptides.

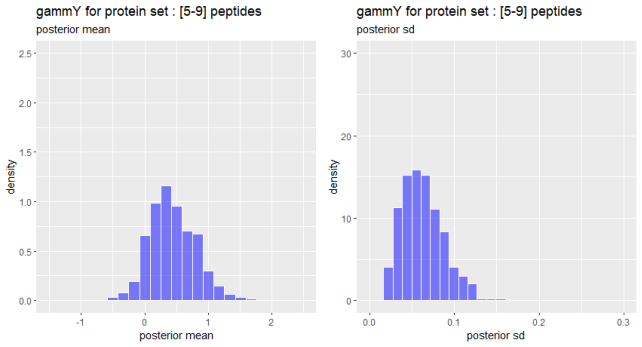


Figure C.37: Histogram of posterior means and standard deviations obtained for γ_Y, for the proteins with $[5-9]$ peptides.

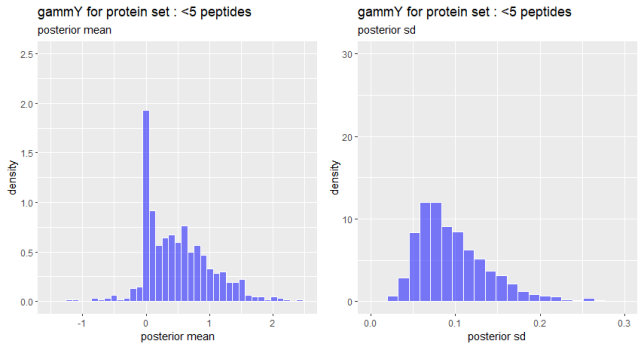


Figure C.38: Histogram of posterior means and standard deviations obtained for γ_Y, for the proteins with < 5 peptides.

C.3.2 γ parameters obtained with informative priors

$\gamma_{Intercept}$ parameter

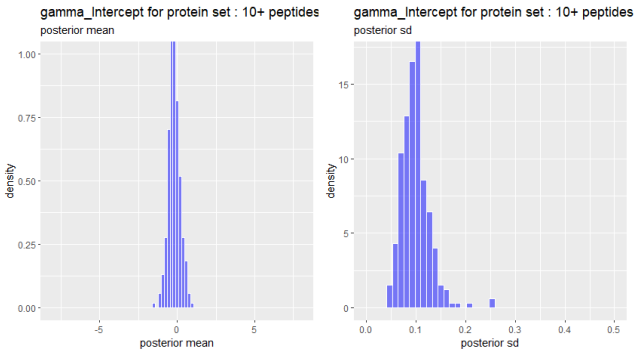


Figure C.39: Histogram of posterior means and standard deviations obtained for $\gamma_{Intercept}$, for the proteins with 10+ peptides.

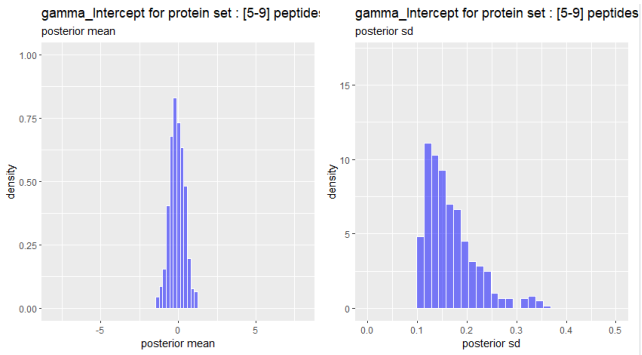


Figure C.40: Histogram of posterior means and standard deviations obtained for $\gamma_{Intercept}$, for the proteins with $[5-9]$ peptides.

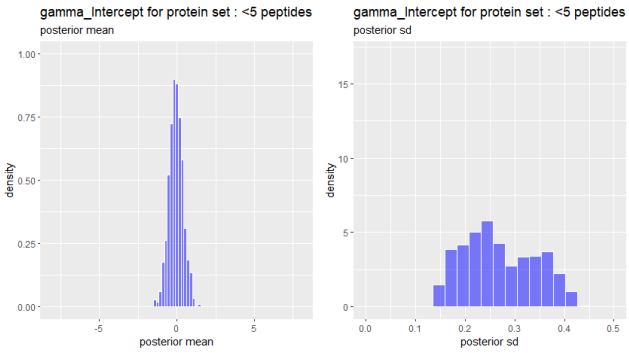


Figure C.41: Histogram of posterior means and standard deviations obtained for $\gamma_{Intercept}$, for the proteins with < 5 peptides.

γ_{lab2} parameter

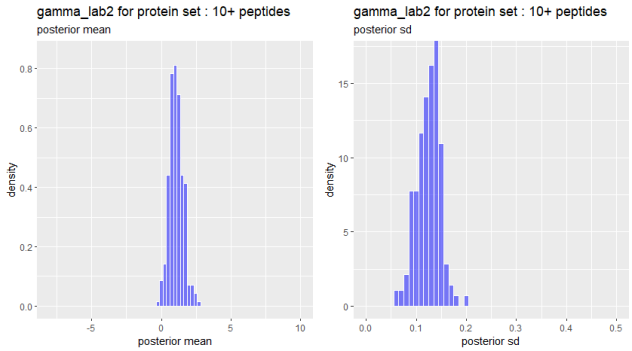


Figure C.42: Histogram of posterior means and standard deviations obtained for γ_{lab2}, for the proteins with 10+ peptides.

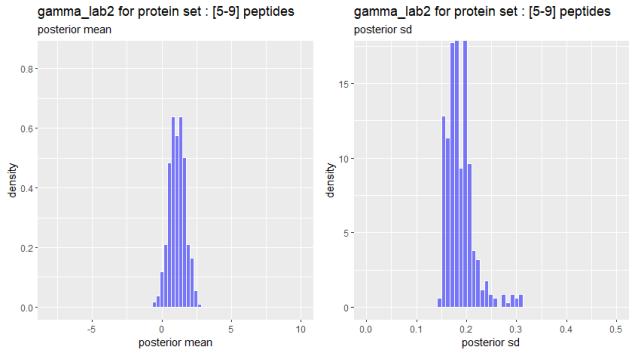


Figure C.43: Histogram of posterior means and standard deviations obtained for γ_{lab2}, for the proteins with $[5-9]$ peptides.

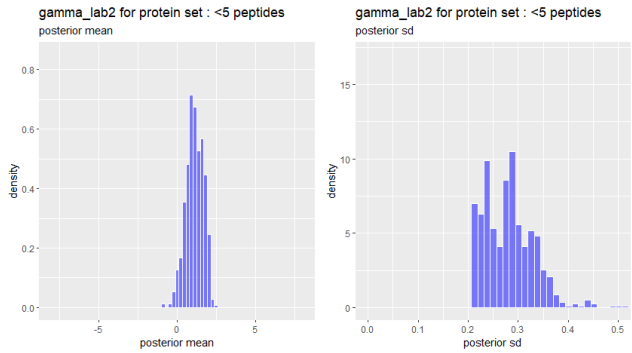


Figure C.44: Histogram of posterior means and standard deviations obtained for γ_{lab2}, for the proteins with < 5 peptides.

γ_{lab3} parameter

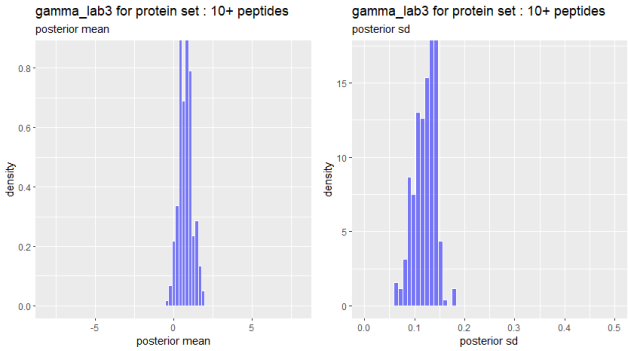


Figure C.45: Histogram of posterior means and standard deviations obtained for γ_{lab3}, for the proteins with 10+ peptides.

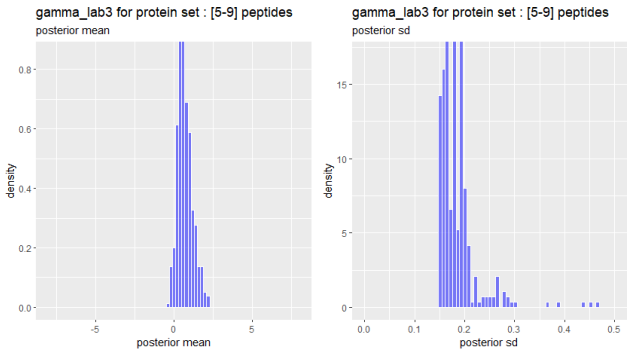


Figure C.46: Histogram of posterior means and standard deviations obtained for γ_{lab3}, for the proteins with $[5-9]$ peptides.

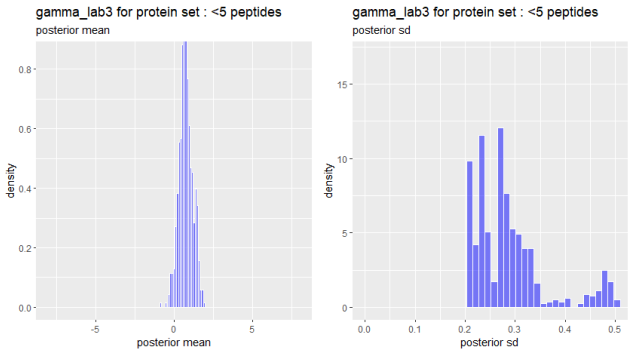


Figure C.47: Histogram of posterior means and standard deviations obtained for γ_{lab3}, for the proteins with < 5 peptides.

γ_y parameter

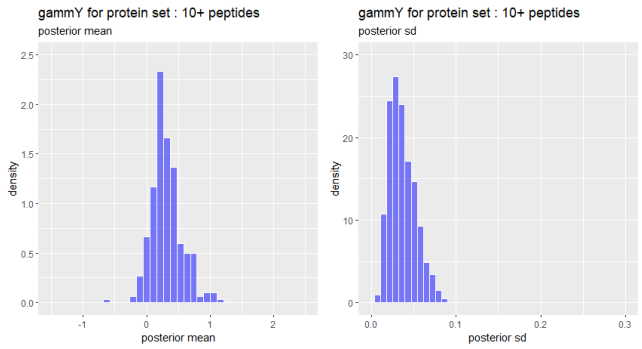


Figure C.48: Histogram of posterior means and standard deviations obtained for γ_Y, for the proteins with 10+ peptides.

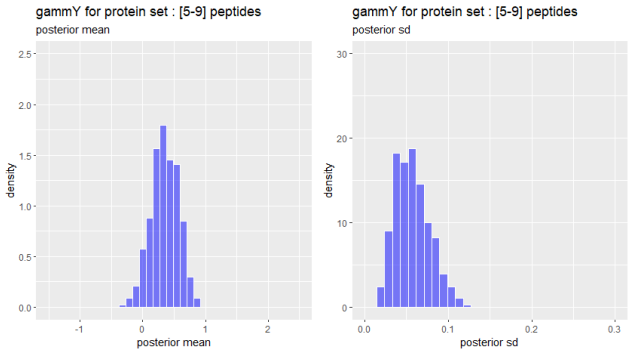


Figure C.49: Histogram of posterior means and standard deviations obtained for γ_Y, for the proteins with $[5-9]$ peptides.

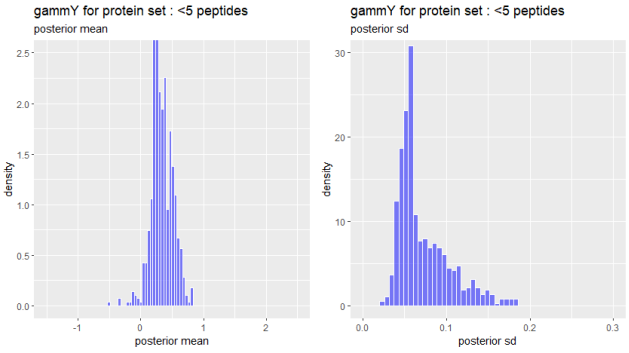


Figure C.50: Histogram of posterior means and standard deviations obtained for γ_Y, for the proteins with < 5 peptides.

Appendix D

Posterior mode of σ^2 in an elementary normal model with missing data

In a first step, consider a simple model where there is one single vector of intensity responses, $\boldsymbol{y}$, which contains N_{obs} observations. The intensities are modelled as i.i.d., normal distributions with a common location parameter μ:

$$y_i \sim \mathcal{N}(\mu, \sigma^2) \text{ with } p(\mu, \sigma^2) \propto 1 \tag{D.1}$$

The log of the joint posterior is :

$$\log p(\mu, \sigma^2 | \boldsymbol{y}_{obs}) = -\frac{N_{obs}}{2} \log(\sigma^2) - \frac{1}{2\sigma^2} (\boldsymbol{y}_{obs} - \mu)^T (\boldsymbol{y}_{obs} - \mu) + const. \tag{D.2}$$

One can derive the mode of the posterior distribution :

$$\begin{aligned} \frac{\partial}{\partial \mu} \log p(\mu, \sigma^2 | \boldsymbol{y}_{obs}) = 0 \Rightarrow \mu = \frac{\sum_{i=1}^{N_{obs}} y_{obs,i}}{N_{obs}} \\ \frac{\partial}{\partial \sigma^2} \log p(\mu, \sigma^2 | \boldsymbol{y}_{obs}) = 0 \Rightarrow \sigma^2 = \frac{(\boldsymbol{y}_{obs} - \mu)^T (\boldsymbol{y}_{obs} - \mu)}{N_{obs}} \end{aligned} \tag{D.3}$$

Combining the two parts of (D.3) gives:

$$\begin{aligned} \hat{\mu} &= \bar{y}_{obs} \\ \hat{\sigma}^2 &= \frac{||\boldsymbol{y}_{obs} - \hat{\mu}||^2}{N_{obs}} \end{aligned} \tag{D.4}$$

In a second step, some missing data are introduced in the response vector $\boldsymbol{y}$, so that there are now N_{obs} observed responses, and N_{mis} missing ones :

$$y_i \sim \mathcal{N}(\mu, \sigma^2) \text{ with } \boldsymbol{y} = (\boldsymbol{y}_{obs}, \boldsymbol{y}_{mis}) \text{ and } p(\mu, \sigma^2, \boldsymbol{y}_{mis}) \propto 1 \tag{D.5}$$

The log of the joint posterior is now :

$$\begin{aligned} \log(p(\mu, \sigma^2, \boldsymbol{y}_{mis} | \boldsymbol{y}_{obs})) = -\frac{N_{obs} + N_{mis}}{2} \log(\sigma^2) - \\ \frac{1}{2\sigma^2} [(\boldsymbol{y}_{obs} - \mu)^T (\boldsymbol{y}_{obs} - \mu) + (\boldsymbol{y}_{mis} - \mu)^T (\boldsymbol{y}_{mis} - \mu)] + const. \end{aligned} \tag{D.6}$$

Again, one can derive the mode :

$$\begin{aligned}
\frac{\partial}{\partial \mu} \log(p(\mu, \sigma^2, \boldsymbol{y}_{mis}|\boldsymbol{y}_{obs})) = 0 \Rightarrow \mu &= \frac{\sum_{i=1}^{N_{obs}} y_{obs,i} + \sum_{i=1}^{N_{mis}} y_{mis,i}}{N_{obs} + N_{mis}} \\
\frac{\partial}{\partial \sigma^2} \log(p(\mu, \sigma^2, \boldsymbol{y}_{mis}|\boldsymbol{y}_{obs})) = 0 \Rightarrow \sigma^2 &= \frac{(\boldsymbol{y}_{obs} - \mu)^T(\boldsymbol{y}_{obs} - \mu) + (\boldsymbol{y}_{mis} - \mu)^T(\boldsymbol{y}_{mis} - \mu)}{N_{obs} + N_{mis}} \\
\frac{\partial}{\partial y_{mis,i}} \log(p(\mu, \sigma^2, \boldsymbol{y}_{mis}|\boldsymbol{y}_{obs})) = 0 \Rightarrow y_{mis,i} &= \mu \ \forall i : 1, \ldots, N_{mis}
\end{aligned} \tag{D.7}$$

Combining the three parts of (D.7) gives:

$$\begin{aligned}
\hat{\mu} &= \bar{y}_{obs} \\
\hat{y_{mis,i}} &= \hat{\mu} \ \forall i : 1, \ldots, N_{mis} \\
\hat{\sigma}^2 &= \frac{||\boldsymbol{y}_{obs} - \hat{\mu}||^2}{N_{obs} + N_{mis}}
\end{aligned} \tag{D.8}$$

Finally, one can calculate the marginal posterior distribution of (μ, σ^2) :

$$\begin{aligned}
p(\mu, \sigma^2|\boldsymbol{y}_{obs}) &= \frac{p(\mu, \sigma^2, \boldsymbol{y}_{mis}|\boldsymbol{y}_{obs})}{p(\boldsymbol{y}_{mis}|\mu, \sigma^2)} \\
\text{with } y_{mis,i} &\sim \mathcal{N}(\mu, \sigma^2) \ \forall i : 1, \ldots, N_{mis}
\end{aligned} \tag{D.9}$$

Expression (D.9) holds for all values of $\boldsymbol{y}_{mis}$. Therefore, setting $\boldsymbol{y}_{mis} = \mu \times \boldsymbol{1_n}$, one obtains :

$$p(\mu, \sigma^2|\boldsymbol{y}_{obs})) \propto p(\mu, \sigma^2, \boldsymbol{y_{mis}}|\boldsymbol{y}_{obs})(\sigma^2)^{N_{mis}/2} \tag{D.10}$$

Combining (D.10) with (D.6) gives :

$$\log(p(\mu, \sigma^2|\boldsymbol{y}_{obs})) = -\frac{N_{obs}}{2} \log(\sigma^2) - \frac{1}{2\sigma^2}[(\boldsymbol{y}_{obs} - \mu)^T(\boldsymbol{y}_{obs} - \mu)] + const. \tag{D.11}$$

The above expression is exactly the same as (D.2), and therefore the mode of the marginal posterior distribution $p(\mu, \sigma^2|\boldsymbol{y}_{obs})$ is :

$$\begin{aligned}
\hat{\mu} &= \bar{y}_{obs} \\
\hat{\sigma}^2 &= \frac{||\boldsymbol{y}_{obs} - \hat{\mu}||^2}{N_{obs}}
\end{aligned} \tag{D.12}$$

Appendix E

Analytical expressions for MFVB approximations

E.1 Parameters of the MFVB approximated densities for the single protein model

In this appendix, one can find detailed analytical expressions for the parameters of the MFVB approximated densities obtained in (4.33), when applying MFVB approximation to the single protein model. Using (4.31) with the product density restriction (4.32), one can show that the optimal approximated densities $q^*(\cdot)$ are such that : (where additional definitions introduced in (4.16) are used):

$$
\begin{aligned}
&(\alpha, \beta, y_{mis}) \overset{q^*}{\sim} \mathcal{N}(\mu_{q^*(\alpha,\beta,y_{mis})}, \Sigma_{q^*(\alpha,\beta,y_{mis})}), \text{ where} \\
&\Sigma_{q^*(\alpha,\beta,y_{mis})} = \left[\begin{array}{ll} \mathbb{E}_{q^*}[\frac{1}{\sigma_\epsilon^2}] W^T \mathbb{E}_{q^*}[D] W + \mathbb{E}_{q^*}[\Sigma_{\alpha\beta}^{-1}(\sigma_r^2)] & -\mathbb{E}_{q^*}[\frac{1}{\sigma_\epsilon^2}] W_{mis}^T \mathbb{E}_{q^*}[D_{mis}] \\ -\mathbb{E}_{q^*}[\frac{1}{\sigma_\epsilon^2}] \mathbb{E}_{q^*}[D_{mis}] W_{mis} & (\sigma^2_{q^*(\gamma_y)} + \mu^2_{q^*(\gamma_y)}) I_{N_{mis}} + \mathbb{E}_{q^*}[\frac{1}{\sigma_\epsilon^2}] \mathbb{E}_{q*}[D_{mis}] \end{array} \right]^{-1} \\
&\boldsymbol{\mu}_{q^*(\alpha,\beta,y_{mis})} = \Sigma_{q^*(\alpha,\beta,y_{mis})} \left[\begin{array}{l} \mathbb{E}_{q^*}[\frac{1}{\sigma_\epsilon^2}] W_{obs}^T \mathbb{E}_{q^*}[D_{obs}] \boldsymbol{y}_{obs} + \mathbb{E}_{q^*}[\Sigma_{\alpha\beta}^{-1}(\sigma_r^2)] \boldsymbol{\mu}_{\alpha\beta} \\ \mu_{q^*(\gamma_y)} (\boldsymbol{\mu}_{q^*(a)} - T_{mis}\ \boldsymbol{\mu}_{q^*(\gamma)} + \mu_{q^*(\gamma_y)} y_{ref}) \end{array} \right]
\end{aligned}
\tag{E.1}
$$

$$
\begin{aligned}
&(\boldsymbol{\gamma}, \gamma_y) \overset{q^*}{\sim} \mathcal{N}(\mu_{q^*(\gamma,\gamma_y)}, \Sigma_{q^*(\gamma,\gamma_y)}), \text{ where} \\
&\Sigma_{q^*(\gamma,\gamma_y)} = \left\{ [T\ (\boldsymbol{\mu}_{q^*(y)} - y_{ref})]^T [T\ (\boldsymbol{\mu}_{q^*(y)} - y_{ref})] + \left[\begin{array}{ll} 0 & 0 \\ 0 & \mathrm{tr}(\Sigma_{q^*(y_{mis})}) \end{array} \right] + \Sigma_{\gamma\gamma_y}^{-1} \right\}^{-1} \\
&\boldsymbol{\mu}_{q^*(\gamma,\gamma_y)} = \Sigma_{q^*(\gamma,\gamma_y)} \left([T\ (\boldsymbol{\mu}_{q^*(y)} - y_{ref})] \boldsymbol{\mu}_{q^*(a)} + \Sigma_{\gamma\gamma_y}^{-1} \boldsymbol{\mu}_{\gamma\gamma_y} \right)
\end{aligned}
\tag{E.2}
$$

$$
\begin{aligned}
q^*(\boldsymbol{a}) &= \prod_{i=1}^{N} q^*(a_i), \text{ such that} \\
a_i &\overset{q^*}{\sim} \left\{ \begin{array}{l} \mathcal{TN}^+(\mu^*_{a,i}, \sigma^2_{a,i} = 1) \text{ when } r_i = 1 \\ \mathcal{TN}^-(\mu^*_{a,i}, \sigma^2_{a,i} = 1) \text{ when } r_i = 0 \end{array} \right\}, \text{ where} \\
\mu^*_{a,i} &= (T \boldsymbol{\mu}_{q^*(\gamma)})_i + \mu_{q^*(\gamma_y)} (\mu_{q^*(y),i} - y_{ref})
\end{aligned}
\tag{E.3}
$$

$$
\begin{aligned}
\sigma_\epsilon^2 &\overset{q^*}{\sim} I.G.(A_{q^*(\sigma_\epsilon^2)}, B_{q^*(\sigma_\epsilon^2)}), \text{ where} \\
A_{q^*(\sigma_\epsilon^2)} &= \frac{\nu_\epsilon}{2} + \frac{N}{2} \\
B_{q^*(\sigma_\epsilon^2)} &= \frac{\nu_\epsilon s_\epsilon^2}{2} + \frac{1}{2}\Big\{(\boldsymbol{\mu}_{q^*(y)} - W\boldsymbol{\mu}_{q^*(\alpha,\beta)})^T \mathbb{E}_{q^*}[D](\boldsymbol{\mu}_{q^*(y)} - W\boldsymbol{\mu}_{q^*(\alpha,\beta)}) \\
&\quad + \operatorname{tr}\left(\mathbb{E}_{q^*}[D] \begin{bmatrix} -W_{obs} & 0 \\ -W_{mis} & I_{N_{mis}} \end{bmatrix} \Sigma_{q^*(\alpha,\beta,y)} \begin{bmatrix} -W_{obs} & 0 \\ -W_{mis} & I_{N_{mis}} \end{bmatrix}^T\right)\Big\}
\end{aligned} \tag{E.4}
$$

$$
\begin{aligned}
q^*(\boldsymbol{\sigma}_r^2) &= \prod_{i=1}^{R} q^*(\sigma_{r_i}^2), \text{ such that} \\
\sigma_{r_i}^2 &\overset{q^*}{\sim} I.G.(A_{q^*(\sigma_{r,i}^2)}, B_{q^*(\sigma_{r,i}^2)}), \text{ where} \\
A_{q^*(\sigma_{r,i}^2)} &= \frac{\nu_{r,i}}{2} + \frac{M^{\beta^{(i)}}}{2} \\
B_{q^*(\sigma_{r,i}^2)} &= \frac{\nu_{r,i} s_{r,i}^2}{2} + \frac{1}{2}\Big\{(\boldsymbol{\mu}_{q^*(\alpha,\beta)}^{(r,i)})^T \boldsymbol{\mu}_{q^*(\alpha,\beta)}^{(r,i)} + \operatorname{tr}(\Sigma_{q^*(\alpha,\beta)}^{(r,i)})\Big\}
\end{aligned} \tag{E.5}
$$

$$
\begin{aligned}
q^*(\boldsymbol{b}) &= \prod_{i=1}^{N} q^*(b_i), \text{ such that} \\
b_i &\overset{q^*}{\sim} I.G.(A_{q^*(b_i)}, B_{q^*(b_i)}), \text{ where} \\
A_{q^*(b_i)} &= \frac{\nu}{2} + \frac{1}{2} \\
B_{q^*(b_i)} &= \frac{\nu}{2} + \frac{1}{2}\mathbb{E}_{q^*}[\frac{1}{\sigma_\epsilon^2}]\Big\{\big((\boldsymbol{\mu}_{q^*(y)} - W\boldsymbol{\mu}_{q^*(\alpha,\beta)})_i\big)^2 + \big([-W\ I_N]\ \Sigma_{q^*(\alpha,\beta,y)}\ [-W\ I_N]^T\big)_{(i,i)}\Big\}
\end{aligned} \tag{E.6}
$$

E.2 Expression of the ELBO for the single protein model

At each iteration, the ELBO needs to be updated applying (4.34), which, combined with the expressions of Section E.1, and after some developments, gives :

$$
\begin{aligned}
ELBO(q) = & \boldsymbol{R}^T \log(\Phi(\boldsymbol{\mu}_a)) + (1_N - \boldsymbol{R})^T \log(1_N - \Phi(\boldsymbol{\mu}_a)) \\
& - \frac{1}{2} \operatorname{tr}([T\ (\boldsymbol{\mu}_{q(y)}) - y_{ref})]^T [T\ (\boldsymbol{\mu}_{q(y)}) - y_{ref})] \Sigma_{q(\gamma,\gamma_y)}) \\
& - \frac{1}{2} \operatorname{tr}(\Sigma_{q(y)})(\mu_q^2(\gamma_y) + \sigma_q^2(\gamma_y)) \\
& + \frac{1}{2} \log|\Sigma_{q(\gamma,\gamma_y)}| - \frac{1}{2}(\boldsymbol{\mu}_{q(\gamma,\gamma_y)} - \boldsymbol{\mu}_{\gamma\gamma_y})^T \Sigma_{\gamma\gamma_y}^{-1} (\boldsymbol{\mu}_{q(\gamma,\gamma_y)} - \boldsymbol{\mu}_{\gamma\gamma_y}) \\
& - \frac{1}{2} \operatorname{tr}(\Sigma_{\gamma\gamma_y}^{-1} \Sigma_{q(\gamma,\gamma_y)}) \\
& - \frac{1}{2} \mathbb{E}_q[\frac{1}{\sigma_\epsilon^2}] \big\{ (\boldsymbol{\mu}_{q(y)} - W\boldsymbol{\mu}_{q(\alpha,\beta)})^T \mathbb{E}_q[D](\boldsymbol{\mu}_{q(y)} - W\boldsymbol{\mu}_{q(\alpha,\beta)}) \\
& \qquad + \operatorname{tr}\left(\mathbb{E}_q[D] \begin{bmatrix} -W_{obs} & 0 \\ -W_{mis} & I_{N_{mis}} \end{bmatrix} \Sigma_{q(\alpha,\beta,y)} \begin{bmatrix} -W_{obs} & 0 \\ -W_{mis} & I_{N_{mis}} \end{bmatrix}^T \right) \big\} \\
& - \frac{1}{2} \big\{ (\boldsymbol{\mu}_{q(\alpha,\beta)} - \boldsymbol{\mu}_{\alpha\beta})^T \mathbb{E}_q[\Sigma_{\alpha\beta}^{-1}(\boldsymbol{\sigma}_r^2)](\boldsymbol{\mu}_{q(\alpha,\beta)} - \boldsymbol{\mu}_{\alpha\beta}) + \operatorname{tr}(\mathbb{E}_q[\Sigma_{\alpha\beta}^{-1}(\boldsymbol{\sigma}_r^2)]\Sigma_{q(\alpha,\beta)}) \big\} \\
& + \frac{1}{2} \log|\Sigma_{q(\alpha,\beta,y_{mis})}| \\
& + \mathbb{E}_q[\frac{1}{\sigma_\epsilon^2}](B_{q(\sigma_\epsilon^2)} - \frac{\nu_\epsilon s_\epsilon^2}{2}) - (\frac{\nu_\epsilon}{2} + \frac{N}{2}) \log(B_{q(\sigma_\epsilon^2)}) \\
& + \sum_{i=1}^{R} \mathbb{E}_q[\frac{1}{\sigma_{r,i}^2}](B_{q(\sigma_{r,i}^2)} - \frac{\nu_{r,i} s_{r,i}^2}{2}) - (\frac{\nu_{r,i}}{2} + \frac{M^{\beta(i)}}{2}) \log(B_{q(\sigma_{r,i}^2)}) \\
& + \sum_{i=1}^{N} \mathbb{E}_q[\frac{1}{b_i}](B_{q(b_i)} - \frac{\nu}{2}) - \frac{\nu+1}{2} \sum_{i=1}^{N} \log(B_{q(b_i)}) \\
& + const.
\end{aligned}
\tag{E.7}
$$

E.3 Parameters of the MFVB approximated densities for the hierarchical model

In this appendix, one can find detailed analytical expressions for the parameters of the MFVB approximated densities obtained in (5.3), when applying MFVB approximation to the hierarchical model. Using (4.31) with the product density restriction (5.2), one can show that the optimal approximated densities $q^*(\cdot)$ are such that :

$$
\begin{aligned}
&(\boldsymbol{\alpha}_k, \boldsymbol{\beta}_k, \boldsymbol{y}_{mis,k}) \overset{q^*}{\sim} \mathcal{N}(\boldsymbol{\mu}_{q^*(\alpha_k,\beta_k,y_{mis,k})}, \Sigma_{q^*(\alpha_k,\beta_k,y_{mis,k})}), \text{ where} \\
&\Sigma_{q^*(\alpha_k,\beta_k,y_{mis,k})} = \begin{bmatrix} \mathbb{E}_{q^*}[\frac{1}{\sigma^2_{\epsilon,k}}] W_k^T \mathbb{E}_{q^*}[D_k] W_k & -\mathbb{E}_{q^*}[\frac{1}{\sigma^2_{\epsilon,r}}] W_{mis,k}^T \mathbb{E}_{q^*}[D_{mis},k] \\ \quad + \mathbb{E}_{q^*}[\Sigma^{-1}_{\alpha,\beta,k}(\boldsymbol{\sigma}^2_{r,k})] & \\ -\mathbb{E}_{q^*}[\frac{1}{\sigma^2_{\epsilon,k}}] \mathbb{E}_{q^*}[D_{mis,k}] W_{mis,k} & (\sigma^2_{q^*(\gamma_{y,k})} + \mu^2_{q^*(\gamma_{y,k})}) I_{N_{mis,k}} \\ & \quad + \mathbb{E}_{q^*}[\frac{1}{\sigma^2_{\epsilon,k}}] \mathbb{E}_{q^*}[D_{mis,k}] \end{bmatrix}^{-1} \\
&\boldsymbol{\mu}_{q^*(\alpha_k,\beta_k,y_{mis,k})} = \Sigma_{q^*(\alpha_k,\beta_k,y_{mis,k})} \begin{bmatrix} \mathbb{E}_{q^*}[\frac{1}{\sigma^2_{\epsilon,k}}] W_{obs,k}^T \mathbb{E}_{q^*}[D_{obs,k}] \boldsymbol{y}_{obs,k} + \mathbb{E}_{q^*}[\Sigma^{-1}_{\alpha,\beta,k}(\boldsymbol{\sigma}^2_{r,k})] \boldsymbol{\mu}_{\alpha\beta} \\ \mu_{q^*(\gamma_{y,k})}(\boldsymbol{\mu}_{q(a_k)} - T_{mis,k} \boldsymbol{\mu}_{q^*(\gamma_k)}) + \mu_{q^*(\gamma_{y,k})} y_{ref}) \end{bmatrix}
\end{aligned}
\tag{E.8}
$$

$$
\begin{aligned}
&(\boldsymbol{\gamma}_k, \gamma_{y,k}) \overset{q^*}{\sim} \mathcal{N}(\boldsymbol{\mu}_{q^*(\gamma_k,\gamma_{y,k})}, \Sigma_{q^*(\gamma_k,\gamma_{y,k})}), \text{ where} \\
&\Sigma_{q^*(\gamma_k,\gamma_{y,k})} = \big\{ [T_k \ (\boldsymbol{\mu}_{q^*(y_k)} - y_{ref})]^T [T_k \ (\boldsymbol{\mu}_{q^*(y_k)} - y_{ref})] + \begin{bmatrix} 0 & 0 \\ 0 & \mathrm{tr}(\Sigma_{q^*(y_{mis,k})}) \end{bmatrix} + \mathrm{diag}(\mathbb{E}^*_q[\frac{1}{\boldsymbol{\sigma}^2_{0\gamma}}]) \big\}^{-1} \\
&\boldsymbol{\mu}_{q^*(\gamma_k,\gamma_{y,k})} = \Sigma_{q^*(\gamma_k,\gamma_{y,k})} \big((T_k, (\boldsymbol{\mu}_{q^*(y_k)} - y_{ref})) \boldsymbol{\mu}_{q^*(a_k)} + \mathrm{diag}(\mathbb{E}^*_q[\frac{1}{\boldsymbol{\sigma}^2_{0\gamma}}]) \ \boldsymbol{\mu}_{q^*(\mu_{0\gamma\gamma y})} \big)
\end{aligned}
\tag{E.9}
$$

$$
\begin{aligned}
q^*(\boldsymbol{a}_k) &= \prod_{i=1}^{N} q^*(a_{i,k}), \text{ such that} \\
a_{i,k} &\overset{q^*}{\sim} \left\{ \begin{array}{l} \mathcal{TN}^+(\mu^*_{a,i,k}, \sigma^2_{a,i,k} = 1) \text{ when } r_{i,k} = 1 \\ \mathcal{TN}^-(\mu^*_{a,i,k}, \sigma^2_{a,i,k} = 1) \text{ when } r_{i,k} = 0 \end{array} \right\}, \text{ where} \\
\boldsymbol{\mu}_{a^*_{i,k}} &= (T_k \boldsymbol{\mu}_{q^*(\gamma_k)})_i + \mu_{q^*(\gamma_{y,k})} (\boldsymbol{\mu}_{q^*(y_k),i} - y_{ref})
\end{aligned}
\tag{E.10}
$$

$$
\sigma^2_{\epsilon,k} \overset{q^*}{\sim} I.G.(A_{q^*(\sigma^2_{\epsilon,k})}, B_{q^*(\sigma^2_{\epsilon,k})}), \text{ where}
$$

$$
\begin{aligned}
A_{q^*(\sigma^2_{\epsilon,k})} &= \mathbb{E}_{q^*}[\frac{\nu_{0\epsilon}}{2}] + \frac{N_k}{2} \\
B_{q^*(\sigma^2_{\epsilon,k})} &= \mathbb{E}_{q^*}[\frac{\nu_{0\epsilon}}{2}] \mathbb{E}_{q^*}[s^2_{0\epsilon}] + \frac{1}{2} \big\{ (\boldsymbol{\mu}_{q^*(y_k)} - W_k \boldsymbol{\mu}_{q^*(\alpha_k,\beta_k)})^T \mathbb{E}_{q^*}[D_k] (\boldsymbol{\mu}_{q^*(y_k)} - W_k \boldsymbol{\mu}_{q^*(\alpha_k,\beta_k)}) \\
&\quad + \mathrm{tr} \left(\mathbb{E}_{q^*}[D_k] \begin{bmatrix} -W_{obs,k} & 0 \\ -W_{mis,k} & I_{N_{mis,k}} \end{bmatrix} \Sigma_{q^*(\alpha_k,\beta_k,y_k)} \begin{bmatrix} -W_{obs,k} & 0 \\ -W_{mi,ks} & I_{N_{mis,k}} \end{bmatrix}^T \right) \big\}
\end{aligned}
\tag{E.11}
$$

$$
\begin{aligned}
q^*(\boldsymbol{\sigma}^2_{r,k}) &= \prod_{i=1}^{R} q^*(\sigma^2_{r,i,k}), \text{ such that} \\
\sigma^2_{r,i,k} &\overset{q^*}{\sim} I.G.(A_{q^*(\sigma^2_{r,i,k})}, B_{q^*(\sigma^2_{r,i,k})}), \text{ where} \\
A_{q^*(\sigma^2_{r,i,k})} &= \mathbb{E}_{q^*}[\frac{\nu_{0,r,i}}{2}] + \frac{M_k^{\beta^{(i)}}}{2} \\
B_{q^*(\sigma^2_{r,i,k})} &= \mathbb{E}_{q^*}[\frac{\nu_{0,r,i}}{2}]\mathbb{E}_{q^*}[s^2_{0,r,i}] + \frac{1}{2}\big\{(\boldsymbol{\mu}^{(r,i)}_{q^*(\alpha_k,\beta_k)})^T \boldsymbol{\mu}^{(r,i)}_{q^*(\alpha_k,\beta_k)} + \text{tr}(\Sigma^{(r,i)}_{q^*(\alpha_k,\beta_k)})\big\}
\end{aligned}
\tag{E.12}
$$

$$
\begin{aligned}
q^*(\boldsymbol{b}_k) &= \prod_{i=1}^{N_k} q^*(b_{i,k}), \text{ such that} \\
b_{i,k} &\overset{q^*}{\sim} I.G.(A_{q^*(b_{i,k})}, B_{q^*(b_{i,k})}), \text{ where} \\
A_{q^*(b_{i,k})} &= \frac{\nu}{2} + \frac{1}{2} \\
B_{q^*(b_{i,k})} &= \frac{\nu}{2} + \frac{1}{2}\mathbb{E}_{q^*}[\frac{1}{\sigma^2_{\epsilon,k}}]\big\{\big((\boldsymbol{\mu}_{q^*(y_k)} - W_k\boldsymbol{\mu}_{q^*(\alpha_k,\beta_k)})_i\big)^2 \\
&\quad + \big([-W_k\ I_{N_k}]\ \Sigma_{q^*(\alpha_k,\beta_k,y_k)}\ [-W_k\ I_{N_k}]^T]\big)_{i,i}\big\}
\end{aligned}
\tag{E.13}
$$

$$
\begin{aligned}
q^*(\boldsymbol{\mu}_{0\gamma}, \mu_{0\gamma_y}) &= \prod_{i=1}^{(M_\gamma+1)} q^*(\mu_{0\gamma_i}), \text{ such that} \\
\mu_{0\gamma_i} &\overset{q^*}{\sim} \mathcal{N}(\mu_{q^*(\mu_{0\gamma_i})}, \sigma^2_{q^*(\mu_{0\gamma_i})}), \text{ where} \\
\mu_{q^*(\mu_{0\gamma_i})} &= \frac{1}{K}\sum_{k=1}^{K} \mu_{q^*(\gamma_{i,k})} \\
\sigma^2_{q^*(\mu_{0\gamma_i})} &= (K\mathbb{E}_{q^*}[\frac{1}{\sigma^2_{0\gamma_i}}])^{-1}
\end{aligned}
\tag{E.14}
$$

$$
\begin{aligned}
q^*(\boldsymbol{\sigma}^2_{0\gamma}, \sigma^2_{0\gamma_y}) &= \prod_{i=1}^{(M_\gamma+1)} q^*(\sigma^2_{0\gamma_i}), \text{ such that} \\
\sigma^2_{0\gamma_i} &\overset{q^*}{\sim} (I.G.(A_{q^*(\sigma^2_{0\gamma_i})}, B_{q^*(\sigma^2_{0\gamma_i})}), \text{ where} \\
A_{q^*(\sigma^2_{0\gamma_i})} &= \frac{K}{2} - 1 \\
B_{q^*(\sigma^2_{0\gamma_i})} &= \frac{1}{2}\big\{\sum_{k=1}^{K}(\mu_{q^*(\gamma_{i,k})} - \mu_{q^*(\mu_{0\gamma_i})})^2 + \sum_{k=1}^{K}\sigma^2_{q^*(\gamma_{i,k})} + K\sigma^2_{q^*(\mu_{0\gamma_i})}\big\}
\end{aligned}
\tag{E.15}
$$

$$
\begin{aligned}
\frac{\nu_{0\epsilon}}{2} &\overset{q^*}{\sim} \nu \text{ distrib.}(\kappa_{q^*(\nu_{0\epsilon})}, \eta_{q^*(\nu_{0\epsilon})}), \text{ where} \\
\kappa_{q^*(\nu_{0\epsilon})} &= K \\
\eta_{q^*(\nu_{0\epsilon})} &= \sum_{k=1}^{K} \mathbb{E}_{q^*}[\log(\sigma^2_{\epsilon,k})] + \mathbb{E}_{q^*}[s^2_{0\epsilon}] \sum_{k=1}^{K} \mathbb{E}_{q^*}[\frac{1}{\sigma^2_{\epsilon,k}}] - K(\mathbb{E}_{q^*}[\log(s^2_{0\epsilon})] + 1)
\end{aligned}
\tag{E.16}
$$

$$
\begin{aligned}
s^2_{0\epsilon} &\overset{q^*}{\sim} \mathcal{G}(A_{q^*(s^2_{0\epsilon})}, B_{q^*(s^2_{0\epsilon})}), \text{ where} \\
A_{q^*(s^2_{0\epsilon})} &= K\mathbb{E}_{q^*}[\frac{\nu_{0\epsilon}}{2}] + 1 \\
B_{q^*(s^2_{0\epsilon})} &= \mathbb{E}_{q^*}[\frac{\nu_{0\epsilon}}{2}] \sum_{k=1}^{K} \mathbb{E}_{q^*}[\frac{1}{\sigma^2_{\epsilon_k}}]
\end{aligned}
\tag{E.17}
$$

$$
\begin{aligned}
q^*(\frac{\boldsymbol{\nu}_{0r}}{2}) &= \prod_{i=1}^{R} q^*(\frac{\nu_{0r_i}}{2}), \text{ such that} \\
\frac{\nu_{0r_i}}{2} &\overset{q^*}{\sim} \nu \text{ distrib.}(\kappa_{q^*(\nu_{0r_i})}), \eta_{q^*(\nu_{0r_i})}), \text{ where} \\
\kappa_{q^*(\nu_{0r_i})} &= K \\
\eta_{q^*(\nu_{0r_i})} &= \sum_{k=1}^{K} \mathbb{E}_{q^*}[\log(\sigma^2_{r,i,k})] + \mathbb{E}_{q^*}[s^2_{0r_i}] \sum_{k=1}^{K} \mathbb{E}_{q^*}[\frac{1}{\sigma^2_{r,i,k}}] - K(\mathbb{E}_{q^*}[\log(s^2_{0r_i})] + 1)
\end{aligned}
\tag{E.18}
$$

$$
\begin{aligned}
q^*(\boldsymbol{s}^2_{0r}) &= \prod_{i=1}^{R} q^*(s^2_{0r_i}), \text{ such that} \\
s^2_{0r_i} &\overset{q^*}{\sim} \mathcal{G}(A_{q^*(s^2_{0r_i})}, B_{q^*(s^2_{0r_i})}), \text{ where} \\
A_{q^*(s^2_{0r_i})} &= K\mathbb{E}_{q^*}[\frac{\nu_{0r_i}}{2}] + 1 \\
B_{q^*(s^2_{0r_i})} &= \mathbb{E}_{q_*}[\frac{\nu_{0r_i}}{2}] \sum_{k=1}^{K} \mathbb{E}_{q^*}[\frac{1}{\sigma^2_{r,i,k}}]
\end{aligned}
\tag{E.19}
$$

E.4 Expression of the ELBO for the hierarchical model

Denote by $ELBO(q,k)$ the contribution to the ELBO coming from the model parameters related to protein k. Applying (4.34), combining with the expressions of Section E.3, and after some developments, one gets the following expression:

$$
\begin{aligned}
ELBO(q,k) = {} & \boldsymbol{R}_k^T \log(\Phi(\boldsymbol{\mu}_{a_k})) + (1_{N_k} - \boldsymbol{R}_k)^T \log(1_{N_k} - \Phi(\boldsymbol{\mu}_{a_k})) \\
& - \frac{1}{2} \operatorname{tr}\left([T_k\ (\boldsymbol{\mu}_{q(y_k)}) - y_{ref})]^T [T_k\ (\boldsymbol{\mu}_{q(y_k)}) - y_{ref})]\ \Sigma_{q(\gamma_k,\gamma_{y,k})} \right) \\
& - \frac{1}{2} \operatorname{tr}(\Sigma_{q(y,k)})(\mu_q^2(\gamma_{y,k}) + \sigma_q^2(\gamma_{y,k})) \\
& + \frac{1}{2} \log|\Sigma_{q(\gamma_k,\gamma_{y,k})}| - \frac{1}{2}(\boldsymbol{\mu}_{q(\gamma_k,\gamma_{y,k})} - \boldsymbol{\mu}_{\gamma_k\gamma_{y,k}})^T \Sigma^{-1}_{\gamma\gamma_{y,k}} (\boldsymbol{\mu}_{q(\gamma_k,\gamma_{y,k})} - \boldsymbol{\mu}_{\gamma_k\gamma_{y,k}}) \\
& - \frac{1}{2} \operatorname{tr}(\Sigma^{-1}_{\gamma\gamma_y}(\Sigma_{q(\gamma_k,\gamma_{y,k})} + \Sigma_{q(\mu_{0\gamma\gamma_y})})) \\
& - \frac{1}{2} \mathbb{E}_q[\frac{1}{\sigma^2_{\epsilon,k}}] \Big\{ (\boldsymbol{\mu}_{q(y_k)} - W_k \boldsymbol{\mu}_{q(\alpha_k,\beta_k)})^T \mathbb{E}_q[D_k] (\boldsymbol{\mu}_{q(y_k)} - W_k \boldsymbol{\mu}_{q(\alpha_k,\beta_k)}) \\
& \qquad + \operatorname{tr}\left(\mathbb{E}_q[D_k] \begin{bmatrix} -W_{obs,k} & 0 \\ -W_{mis,k} & I_{N_{mis,k}} \end{bmatrix} \Sigma_{q(\alpha_k,\beta_k,y_k)} \begin{bmatrix} -W_{obs,k} & 0 \\ -W_{mis,k} & I_{N_{mis,k}} \end{bmatrix}^T \right) \Big\} \\
& - \frac{1}{2} \Big\{ (\boldsymbol{\mu}_{q(\alpha_k,\beta_k)} - \boldsymbol{\mu}_{\alpha\beta})^T \mathbb{E}_q[\Sigma^{-1}_{\alpha\beta}(\boldsymbol{\sigma}_r^2)] (\boldsymbol{\mu}_{q(\alpha_k,\beta_k)} - \boldsymbol{\mu}_{\alpha\beta}) + \operatorname{tr}(\mathbb{E}_q[\Sigma^{-1}_{\alpha\beta}(\boldsymbol{\sigma}_r^2)] \Sigma_{q(\alpha_k,\beta_k)}) \Big\} \\
& + \frac{1}{2} \log|\Sigma_{q(\alpha_k,\beta_k,y_{mis,k})}| \\
& + \mathbb{E}_q[\frac{1}{\sigma^2_{\epsilon,k}}] B_{q(\sigma^2_{\epsilon,k})} - A_{q(\sigma^2_{\epsilon,k})} \log(B_{q(\sigma^2_{\epsilon,k})}) + \log(\Gamma(A_{q(\sigma^2_{\epsilon,k})})) \\
& + \sum_{i=1}^{R} \Big\{ \mathbb{E}_q[\frac{1}{\sigma^2_{r,i,k}}] B_{q(\sigma^2_{i,r,k})} - A_{q(\sigma^2_{i,r,k})} \log(B_{q(\sigma^2_{i,r,k})}) + \log(\Gamma(A_{q(\sigma^2_{i,r,k})})) \Big\} \\
& + \sum_{i=1}^{N} \mathbb{E}_q[\frac{1}{b_{i,k}}] (B_{q(b_{i,k})} - \frac{\nu}{2}) - \frac{\nu+1}{2} \sum_{i=1}^{N} \log(B_{q(b_{i,k})}) \\
& + const.
\end{aligned}
\tag{E.20}
$$

Then, the total ELBO is the sum of the contributions of the parameters of each protein k, and of the contribution of the hyperparameters :

$$
\begin{aligned}
ELBO(q) = & \sum_{k=1}^{K} ELBO(q,k) \\
& + \sum_{i=1}^{M^\gamma+1} \mathbb{E}_q[\frac{1}{\sigma^2_{0\gamma_i}}] B_{q(\sigma^2_{0\gamma_i})} - (\frac{K}{2}-1) \sum_{i=1}^{M^\gamma+1} \log(B_{q(\sigma^2_{0\gamma_i})}) - \frac{1}{2} \sum_{i=1}^{M^\gamma+1} \log(\sigma^2_{q(\gamma_{0i})}) \\
& - \left\{ A_{q(s^2_{0\epsilon})} \log(B_{q(s^2_{0\epsilon})}) - \log(\Gamma(A_{q(s^2_{0\epsilon})})) + (A_{q(s^2_{0\epsilon})} - 1)\mathbb{E}_q[\log(s^2_{0\epsilon})] - B_{q(s^2_{0\epsilon})}\mathbb{E}_q[s^2_{0\epsilon}] \right\} \\
& - \sum_{i=1}^{R} \left\{ A_{q(s^2_{0r_i})} \log(B_{q(s^2_{0r_i})}) - \log(\Gamma(A_{q(s^2_{0r_i})})) + (A_{q(s^2_{0r_i})} - 1)\mathbb{E}_q[\log(s^2_{0r_i})] - B_{q(s^2_{0r_i})}\mathbb{E}_q[s^2_{0r_i}] \right\} \\
& + \mathbb{E}_q[\frac{\nu_{0\epsilon}}{2}] \sum_{k=1}^{K} \mathbb{E}_q[\log(\sigma^2_{\epsilon,k})] - A_{q(\frac{\nu_{0\epsilon}}{2})} \log(B_{q(\frac{\nu_{0\epsilon}}{2})}) + \log(\Gamma(A_{q(\frac{\nu_{0\epsilon}}{2})})) \\
& + \sum_{i=1}^{R} \mathbb{E}_q[\frac{\nu_{0r_i}}{2}] \sum_{k=1}^{K} \mathbb{E}_q[\log(\sigma^2_{r,i,k})] - A_{q(\frac{\nu_{0r_i}}{2})} \log(B_{q(\frac{\nu_{0r_i}}{2})}) + \log(\Gamma(A_{q(\frac{\nu_{0r_i}}{2})})) \\
& + const.
\end{aligned}
\tag{E.21}
$$

E.5 Approximations of the ν distribution

In equation (5.3), the ν parameters have their MFVB approximated log-densities of the following form :

$$
\log q^*(\frac{\nu}{2}; \kappa, \eta) = \kappa[(\frac{\nu}{2}) \log(\frac{\nu}{2}) - \log\Gamma(\frac{\nu}{2})] - (\kappa+\eta)(\frac{\nu}{2}) + const. \tag{E.22}
$$

where $\Gamma(x)$ is the gamma function, η and κ are positive, and η is typically small compared to κ. $\mathbb{E}_q[\frac{\nu}{2}]$ has to be calculated at each iteration of the MFVB algorithm, and samples from the ν distribution are also needed for distributional comparison purposes.

A first straightforward solution can be to use numerical integration to calculate the multiplicative constant, the mean and quantiles. However this leads to numerical problems when κ gets big, which is typically true since $\kappa = K$ (number of proteins).

Following an idea exposed in [Jullion and Lambert 2007], Section 3.1.2 , the following Stirling's approximation of the factorial function can be used :

$$
\log(n!) \approx (n+\frac{1}{2}) \log(n) - n + \frac{1}{2} \log(2\pi) \tag{E.23}
$$

Replacing then $\log(\Gamma(x)) = \log(x!) - \log(x)$ by its approximation in (E.22), one then gets :

$$
\frac{\nu}{2} \overset{\text{approx.}}{\sim} \mathcal{G}(\frac{\kappa}{2}+1, \eta)
$$

However, the above Gamma approximation suffers from a slight bias in the mode, as will be shown later in this section. However, it can be improved if its parameters $(A_{(\frac{\nu}{2})}, B_{(\frac{\nu}{2})})$ are calculated by matching the mode, as well as the local curvature at the mode, of the exact ν

Algorithm 5 Procedure to calculate a $\mathcal{G}(A, B)$ approximation of a ν *distribution*(κ, η)

procedure GAMMAAPPROXI(κ, η)

▷ find the mode numerically

$\mu \leftarrow \text{root}\big(\kappa(\log(x) + 1 - \psi(x)) - \eta = 0\big)$ where $\psi(x)$ is the digamma function

▷ calculate an approximate variance from the local curvature at the mode

$\sigma^2 \leftarrow -1/\{\kappa(\frac{1}{\mu} - \psi'(\mu))\}$ where $\psi'(x)$ is the trigamma function

▷ obtain A et B by mode and curvature matching

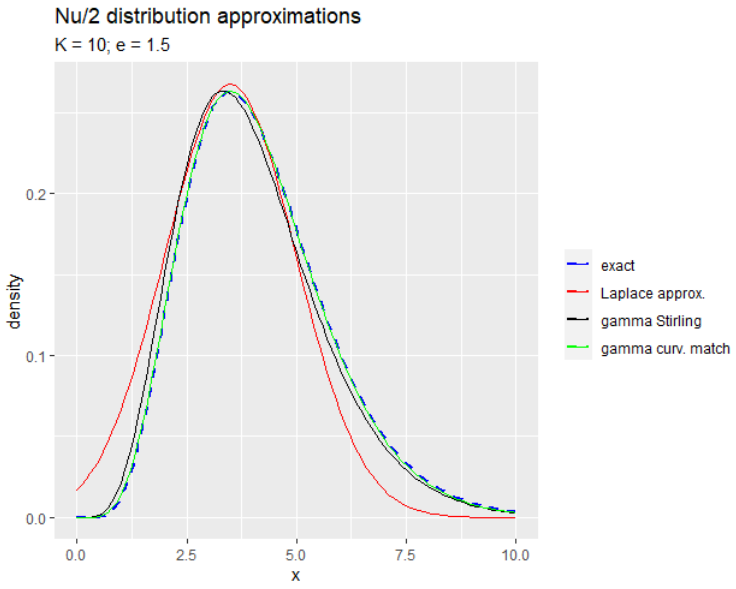
$A \leftarrow \frac{\mu^2}{\sigma^2} + 1 \; ; \; B \leftarrow \frac{\mu}{\sigma^2}$

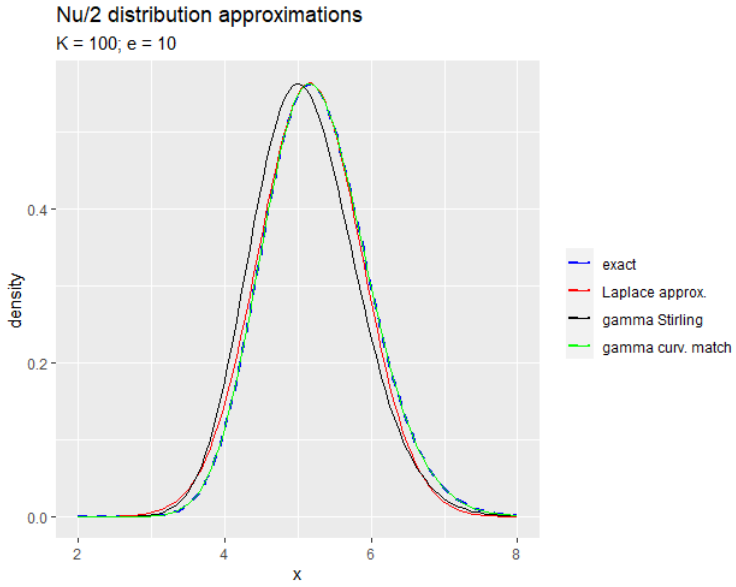
return (A, B)

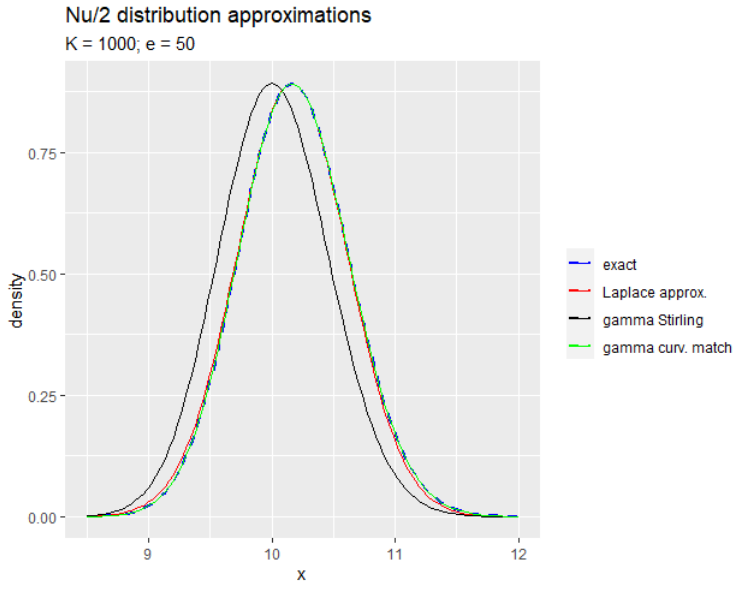
end procedure

distribution. This leads to Algorithm 5.

In the graphs below, the accuracy of the approximation is checked, for typical values of κ and η. On these graphs, one can also find the Gamma distribution using Stirling factorial approximation formula (E.23), as well as the Laplace approximation, which leads to an approximation by a normal distribution. One observes that the Gamma approximation with mode and curvature matching performs quite well for all sets of tested parameters. Laplace approximation performs also quite well for non small values of κ, while a mode bias appears for the Gamma Stirling approximation, when κ value is high.




Nu/2 distribution approximations
K = 100; e = 10
density
0.0
0.2
0.4
2
4
6
8
x
exact
Laplace approx.
gamma Stirling
gamma curv. match


Nu/2 distribution approximations
K = 1000; e = 50
density
0.00
0.25
0.50
0.75
9
10
11
12
x
exact
Laplace approx.
gamma Stirling
gamma curv. match

Appendix F

Supplementary material for the model comparison study

F.1 Specifications of the competing models

In this section, the models introduced in the model comparison study of Section 5.5 are specified in details, using the same notations as in Section 4.2.1 of the main text. Note that in what follows, the protein index k has been dropped.

F.1.1 Simple Linear Model

$$\begin{aligned} \boldsymbol{y}|\boldsymbol{\alpha}, \sigma_\epsilon^2 &\sim \mathcal{N}(X\boldsymbol{\alpha}, \sigma_\epsilon^2 I_N) \\ \boldsymbol{\alpha} &\sim \mathcal{N}(\boldsymbol{\mu}_\alpha, \Sigma_\alpha) \\ \sigma_\epsilon^2 &\sim \text{Inv-}\chi^2(\nu_\epsilon, s_\epsilon^2) \end{aligned} \tag{F.1}$$

The model matrix X is particularized to the CPTAC dataset by using the following synthetic formula :

$$y \sim condition + lab + peptide$$

The peptide effects are therefore modelled as fixed effects. Weakly informative prior parameters are selected, according to section 4.2.3. There is no modelling of missingness.

F.1.2 Robust Linear Model

$$\begin{aligned} \boldsymbol{y}|\boldsymbol{\alpha}, \sigma_\epsilon^2 &\sim t_\nu(X\boldsymbol{\alpha}, \sigma_\epsilon^2 I_N) \text{ , with } \nu = 4 \\ \boldsymbol{\alpha} &\sim \mathcal{N}(\boldsymbol{\mu}_\alpha, \Sigma_\alpha) \\ \sigma_\epsilon^2 &\sim \text{Inv-}\chi^2(\nu_\epsilon, s_\epsilon^2) \end{aligned} \tag{F.2}$$

The model matrix X is particularized to the CPTAC dataset by using the following synthetic formula :

$$y \sim condition + lab + peptide$$

The peptide effects are therefore modelled as fixed effects. Weakly informative prior parameters are selected, according to section 4.2.3. There is no modelling of missingness.

F.1.3 Robust Hierarchical Model

$$\begin{aligned} \boldsymbol{y}|\boldsymbol{\alpha}, \sigma_\epsilon^2 &\sim t_\nu(X\boldsymbol{\alpha}, \sigma_\epsilon^2 I_N) \text{ , with } \nu = 4 \\ \boldsymbol{\alpha} &\sim \mathcal{N}(\boldsymbol{\mu}_\alpha, \Sigma_\alpha) \\ \sigma_\epsilon^2 &\sim \text{Inv-}\chi^2(\nu_{0\epsilon}, s_{0\epsilon}^2) \\ p(\nu_{0\epsilon}, s_{0\epsilon}^2) &\propto 1 \end{aligned} \tag{F.3}$$

In Equation (F.3), the hyperparameters $\nu_{0\epsilon}$ and $s_{0\epsilon}^2$ are the same for each protein k. Also, the model matrix X is particularized to the CPTAC dataset by using the following synthetic formula :

$$y \sim condition + lab + peptide$$

The peptide effects are therefore modelled as fixed effects. For all model parameters except σ_ϵ^2, weakly informative prior parameters are selected, according to section 4.2.3. There is no modelling of missingness.

F.2 Additional results of the model comparison study

In this section, additional indicators tables and volcano plots are provided, relating to the assessment of the inferential performance of the competing models (Section 5.5.4).

Model	Indicator	Proba level				
		50%	60%	70%	80%	90%
Simple Linear	FP	0	0	0	0	0
	FN	6	8	9	12	15
Robust Linear	FP	0	0	0	0	0
	FN	5	7	8	8	11
Robust Hierarchical	FP	0	0	0	0	0
	FN	5	6	8	8	11
Full Hierarchical	FP	0	0	0	0	0
	FN	4	5	8	8	8

Table F.1: false positives (FP) and false negatives (FN) for the four competing models, for the B vs. A contrast, and with $lFC_{thresh} = 1.0$

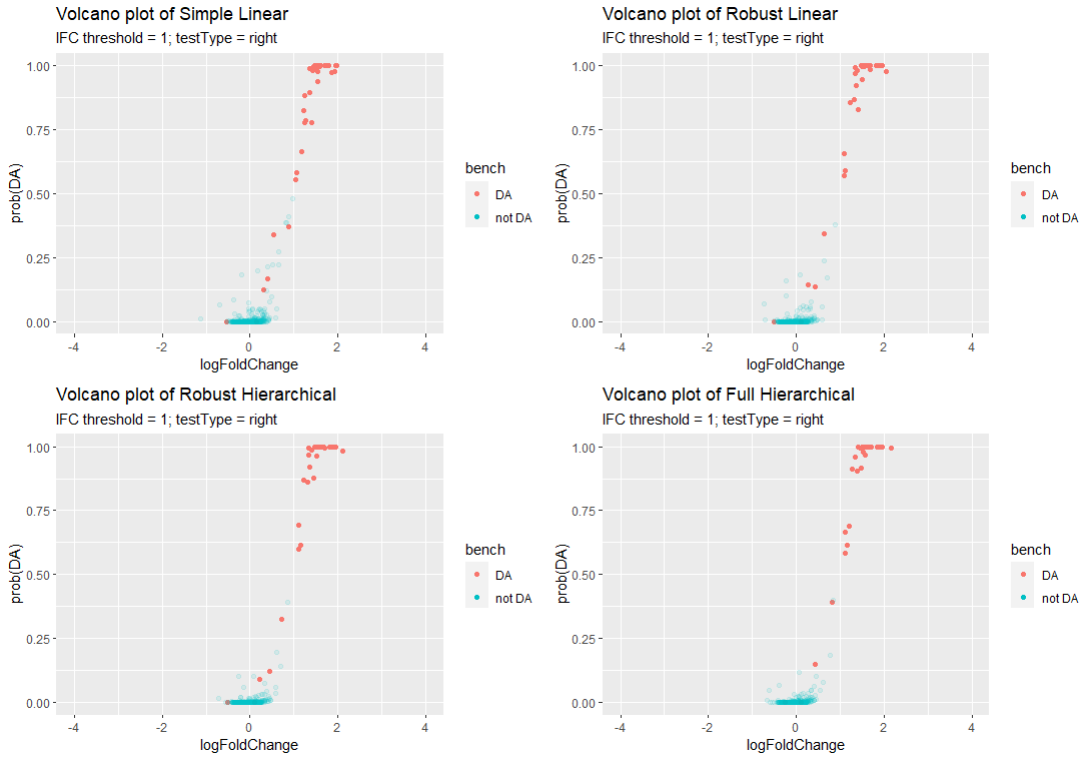


Figure F.1: Volcano plots for the four competing models, for the B vs. A contrast, and with $lFC_{thresh} = 1.0$

Model	Indicator	Proba level				
		50%	60%	70%	80%	90%
Simple Linear	FP	5	1	1	1	0
	FN	1	1	1	2	3
Robust Linear	FP	6	2	1	1	0
	FN	1	1	1	2	3
Robust Hierarchical	FP	6	1	1	1	1
	FN	1	1	1	1	2
Full Hierarchical	FP	4	3	1	0	0
	FN	1	1	1	2	2

Table F.2: false positives (FP) and false negatives (FN) for the four competing models, for the C vs. B contrast, and with $lFC_{thresh} = 0.5$

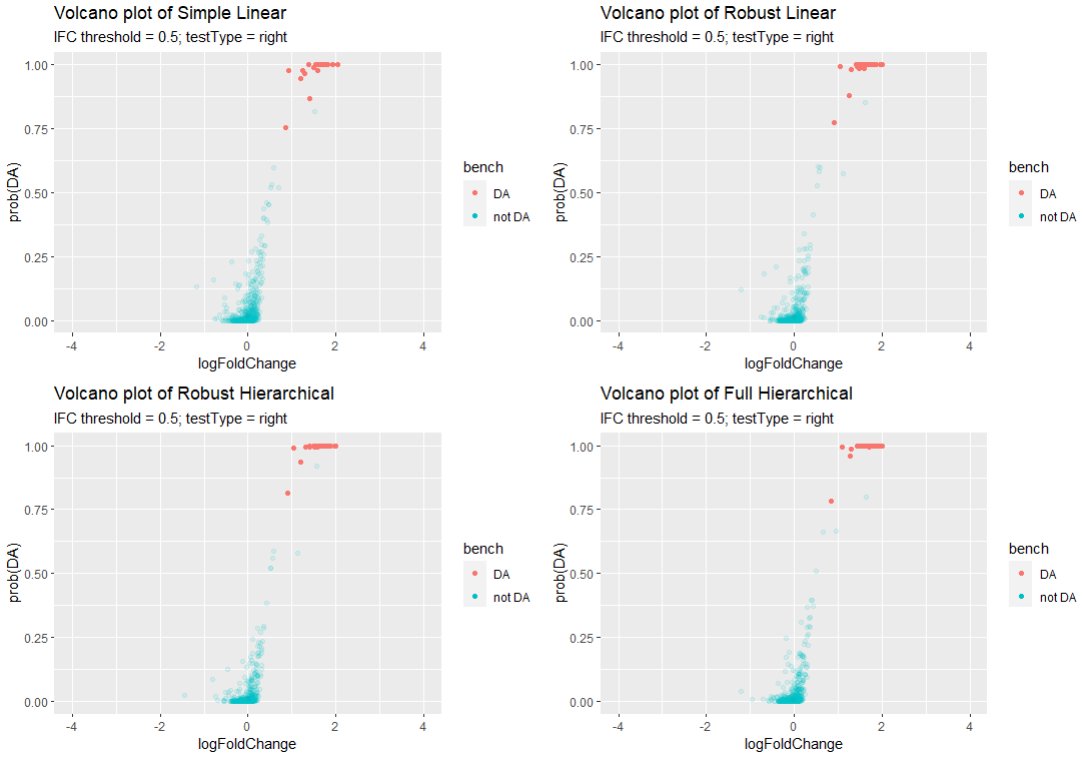


Figure F.2: Volcano plots for the four competing models, for the C vs. B contrast, and with $lFC_{thresh} = 0.5$

Model	Indicator	Proba level				
		50%	60%	70%	80%	90%
Simple Linear	FP	1	1	1	1	0
	FN	3	3	5	7	8
Robust Linear	FP	2	1	1	0	0
	FN	2	2	4	4	7
Robust Hierarchical	FP	2	1	1	1	0
	FN	2	3	4	4	6
Full Hierarchical	FP	1	1	0	0	0
	FN	2	2	3	4	5

Table F.3: false positives (FP) and false negatives (FN) for the four competing models, for the C vs. B contrast, and with $lFC_{thresh} = 1.0$

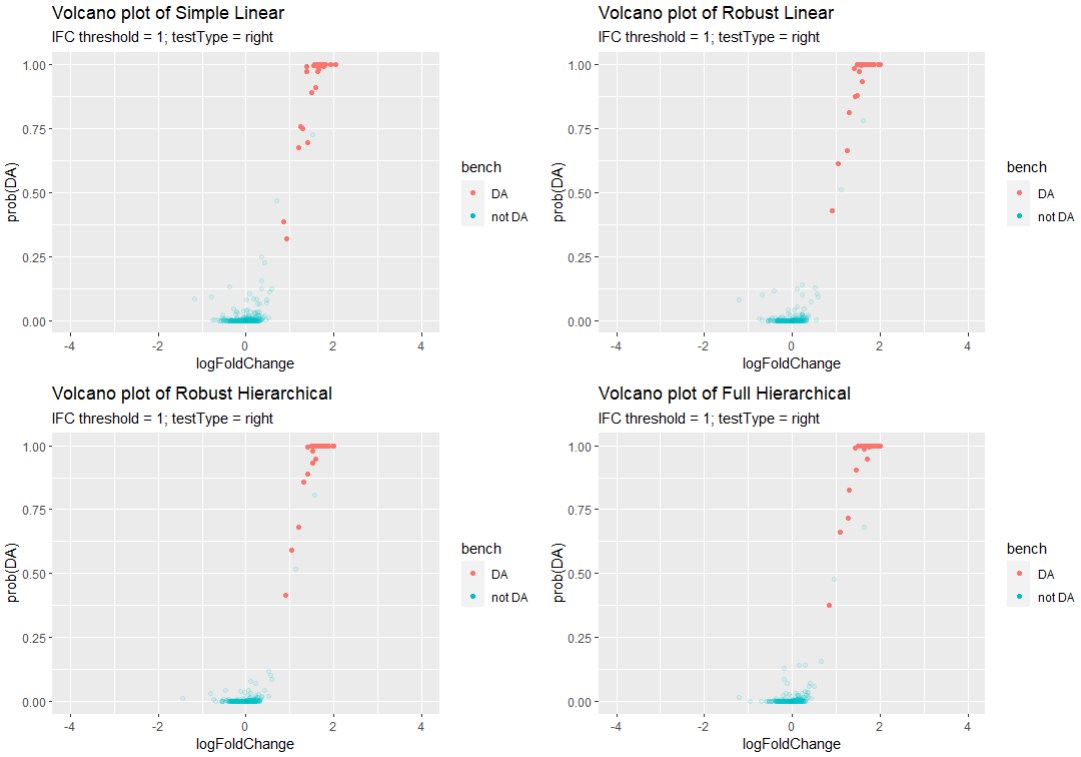


Figure F.3: Volcano plots for the four competing models, for the C vs. B contrast, and with $lFC_{thresh} = 1.0$

Appendix G

Stan scripts

G.1 Code of Stan hierarchical model

In the following, one provides the code of the Stan model script that describes the models used in this work in Stan domain specific language. Note that this script is generic, so that depending on some input flags, it can be used both for the single protein model introduced in Chapter 4, as well as the hierarchical model introduced in Chapter 5.

```
// saved as PeptideLevelWithMixedMissingness.stan

data {
  int<lower=1> K;                    // number of proteins
  int<lower=0> nRdm;                 // number of random effects
  int<lower=1> NObs[K];              // number of observations per protein
  int<lower=1> NMis[K];              // number of missing data per protein
  int<lower=1> MAlpha[K];            // number of alpha's (fixed effects coefs) per protein
  int<lower=0> MBeta[K];             // number of beta's (random effects coefs) per protein
  int<lower=1> MBetaPerRdmEffect[nRdm*K]; // same as before, per random effect
  int<lower=1> MGamma[K];            // number of gamma coeffs (missingness model)
  int<lower=1> NObsSum;              // total number of observations (=sum(NObs))
  int<lower=1> NMisSum;              // total number of missing data (=sum(NMis))
  int<lower=1> XObsLen;              // dimension of the flat vector containing all model matrices
                                     // for the fixed effects of the observed responses
  int<lower=1> XMisLen;              // dimension of the flat vector containing all model matrices
                                     // for the fixed effects of the missing responses
  int<lower=0> ZObsLen;              // dimension of the flat vector containing all model matrices
                                     // for the random effects of the observed responses
  int<lower=0> ZMisLen;              // dimension of the flat vector containing all model matrices
                                     // for the random effects of the missing responses
  int<lower=1> TObsLen;              // dimension of the flat vector containing all model matrices
                                     // for the fixed effects of the missing indicators equal 0
  int<lower=1> TMisLen;              // dimension of the flat vector containing all model matrices
                                     // for the fixed effects of the missing indicators equal 1

  row_vector[XObsLen] XObs;          // concatenation of model matrices in a flat row vector format,
                                     // for the fixed effects of the observed responses
  row_vector[XMisLen] XMis;          // concatenation of model matrices in a flat row vector format,
                                     // for the fixed effects of the missing responses
  vector[NObsSum] yObs;              // observed responses (log-2 intensities of peptides)
  row_vector[ZObsLen] ZObs;          // concatenation of model matrices in a flat row vector format,
                                     // for the random effects of the observed responses
  row_vector[ZMisLen] ZMis;          // concatenation of model matrices in a flat row vector format,
                                     // for the random effects of the missing responses
  row_vector[TObsLen] TObs;          // concatenation of model matrices in a flat row vector format,
                                     // for the fixed effects of the missing indicators equal 0
  row_vector[TMisLen] TMis;          // concatenation of model matrices in a flat row vector format,
                                     // for the fixed effects of the missing indicators equal 1

  // general model settings
  int<lower=0, upper=1> t_likelihood;      // use student likelihood (robust version),
                                           // otherwise normal likelihood
  int<lower=1> t_fixedDF;                  // the fixed df parameter for the student likelihood
  int<lower=0, upper=1> modelMiss;         // model missingness (1) or not (0)
  int<lower=0, upper=1> NMAR;              // include NMAR component, yes (1) or not (0)
  vector[K] refIntensity;                  // ref intensity set in the NMAR term
                                           // of the missingness model
  int<lower=0, upper=1> sigmaHierarch;     // hierarchical set-up for sigma (1) or not (0)
  int<lower=0, upper=1> sigmaRdmHierarch;  // hierarchical set-up for sigma of random effects (1)
                                           // or not (0)
```

```
  int<lower=0, upper=1> gammaHierarch;      // hierarchical set-up for gamma (1) or not (0)

  // priors for intensity model
  real priorIntensityMean;                  // prior distribution for intercept : mean
  real priorIntensitySd;                    // prior distribution for intercept : std dev
  real priorFoldChangeMean;                 // prior distribution for fold changes : mean
  real priorFoldChangeSd;                   // prior distribution for fold changes : std dev
  real priorNuEpsilon;                      // prior inv chi2 distribution for error variances :
                                            // nu parameter
  real priorScaleEpsilon;                   // prior inv chi2 distribution for error variances :
                                            // scale parameter
  real priorNuRdm;                          // prior inv chi2 distribution for random effects variances :
                                            // nu parameter
  real priorScaleRdm;                       // prior inv chi2 distribution for random effects variances :
                                            // scale parameter

  // priors for missingness model
  real priorGammaInterceptMean;             // prior normal distribution for intercept
                                            // in missingness model : mean
  real priorGammaInterceptSd;               // prior normal distribution for intercept
                                            // in missingness model : std dev
  real priorGammaTermsMean[max(MGamma[1]-1,1)];// prior normal distribution for the other MAR
                                            // coefficients in missingness model : mean
  real priorGammaTermsSd[max(MGamma[1]-1,1)];  // prior normal distribution for the other MAR
                                            // coefficients in missingness model : std dev
  real priorGammaYMean;                     // prior normal distribution for NMAR coefficient
                                            // in missingness model : mean
  real priorGammaYSd;                       // prior normal distribution for NMAR coefficient
                                            // in missingness model : std dev

  // contrasts
  int<lower=0> nContrast[K];                // nb of contrasts per protein
  int<lower=0> contrastCoefLen;             // dimension of the flat vector containing all contrast
                                            // coefficients
  vector[contrastCoefLen] contrastCoefs;// concatenation of all constrast coefficient vectors
}


transformed data {
  int NMAlphaObs[K];
  int NMBetaObs[K];
  int NMGammaObs[K];
  int NMAlphaMis[modelMiss?K:0];
  int NMBetaMis[modelMiss?K:0];
  int NMGammaMis[modelMiss?K:0];
  int MAlphaSum;
  int MBetaSum;
  int MGammaSum;
  int MGammaMax;
  int NContrastSum;

  // store N*M for all k
  for(k in 1:K){
    NMAlphaObs[k] = NObs[k] * MAlpha[k];
    NMBetaObs[k] = NObs[k] * MBeta[k];
    NMGammaObs[k] = NObs[k] * MGamma[k];
    if(modelMiss){
      NMAlphaMis[k] = NMis[k] * MAlpha[k];
      NMBetaMis[k] = NMis[k] * MBeta[k];
      NMGammaMis[k] = NMis[k] * MGamma[k];
    }
  }
  MAlphaSum = sum(MAlpha);
  MBetaSum = sum(MBeta);
  MGammaSum = sum(MGamma);
  MGammaMax = max(MGamma);
  NContrastSum = sum(nContrast);
}

parameters {

  // * parameters of intensity model *

  // coefficients for fixed effects
  vector[MAlphaSum] alpha;
  // coefficients for random effects
  vector[nRdm>0?MBetaSum:0] beta;
  // error variances
  real<lower=0> sigma2Eps[K];
  // variances of random effects
  vector<lower=0> [nRdm*K] sigma2Rdm;


  // * hyperparameters of hierarchical intensity model *

  // degrees of freedom hyperparameter for error variances
  real<lower=1> nu0Eps[sigmaHierarch?1:0];
  // scale hyperparameter for error variances
  real<lower=0> s0Eps2[sigmaHierarch?1:0];
  // degrees of freedom hyperparameter for variances of rdm effects
```

```
  real<lower=1> nu0Rdm[nRdm*sigmaRdmHierarch];
  // scale hyperparameter for variances of rdm effects
  real<lower=0> s0Rdm2[nRdm*sigmaRdmHierarch];

  // * parameters of missingness model *

  // coefficients for predictors in probit regression of missingness (MAR)
  vector[modelMiss?MGammaSum:0] gamma;
  // coefficients for Y predictor in probit regression of missingness (NMAR)
  vector[modelMiss*NMAR?K:0] gammY;
  // missing responses
  vector[modelMiss?NMisSum:0] yMis;

  // * hyperparameters of hierarchical missingness model *

  // mean hyperparameters for gamma parameters
  vector [modelMiss*gammaHierarch?(MGammaMax+NMAR):0] mu0gm;
  // scale hyperparameters for gamma parameters
  vector<lower=0> [modelMiss*gammaHierarch?(MGammaMax+NMAR):0] sigma20gm;

}

model {
  int yobspos;
  int zobspos;
  int xobspos;
  int tobspos;
  int ymispos;
  int zmispos;
  int xmispos;
  int tmispos;
  int apos;
  int bpos;
  int cpos;
  int spos;

  yobspos = 1;
  xobspos = 1;
  zobspos = 1;
  tobspos = 1;
  ymispos = 1;
  xmispos = 1;
  zmispos = 1;
  tmispos = 1;
  apos = 1;
  bpos = 1;
  cpos = 1;
  spos = 1;


  for(k in 1:K){
    vector[MAlpha[k]] currentAlpha;
    matrix[NObs[k], MAlpha[k]] XO;

    vector[NObs[k]] YO;

    currentAlpha = segment(alpha, apos, MAlpha[k]);

    // likelihood of observed responses

    YO = segment(yObs, yobspos, NObs[k]);
    XO = to_matrix( segment(XObs, xobspos, NMAlphaObs[k]), NObs[k], MAlpha[k]);

    if(nRdm > 0){
      vector[MBeta[k]] currentBeta;
      vector[nRdm] currentSigma2Rdm;
      matrix[NObs[k], MBeta[k]] ZO;
      int rpos = 1;

      currentBeta = segment(beta, bpos, MBeta[k]);
      currentSigma2Rdm = segment(sigma2Rdm, spos, nRdm);
      ZO = to_matrix( segment(ZObs, zobspos, NMBetaObs[k]), NObs[k], MBeta[k]);

      if(t_likelihood){
        YO ~ student_t(t_fixedDF, XO * currentAlpha + ZO * currentBeta, sqrt(sigma2Eps[k]) );
      } else {
        YO ~ normal( XO * currentAlpha + ZO * currentBeta, sqrt(sigma2Eps[k]) );
      }


      for(r in 1:nRdm){
        int MCol = MBetaPerRdmEffect[(k-1)*nRdm+r];
        segment(currentBeta, rpos, MCol) ~
            normal(0, sqrt(currentSigma2Rdm[r]));
        rpos += MCol;
      }

    } else {
      if(t_likelihood){
        YO ~ student_t(t_fixedDF, XO * currentAlpha, sqrt(sigma2Eps[k]) );
```

```
  } else {
    YO ~ normal( XO * currentAlpha, sqrt(sigma2Eps[k]) );
  }

}

// priors for Alpha's
// intercept
alpha[apos] ~ normal(priorIntensityMean, priorIntensitySd);
// logFoldChanges all
if(MAlpha[k] > 1){
  segment(alpha, apos+1, MAlpha[k]-1) ~ normal( priorFoldChangeMean, priorFoldChangeSd);
}

yobspos += NObs[k];
xobspos += NMAlphaObs[k];
zobspos += NMBetaObs[k];


if (modelMiss == 1){

  matrix[modelMiss?NMis[k]:0, modelMiss?MAlpha[k]:0] XM;
  matrix[modelMiss?NObs[k]:0, modelMiss?MGamma[k]:0] TO;
  matrix[modelMiss?NMis[k]:0, modelMiss?MGamma[k]:0] TM;
  vector[MGamma[k]] currentGamma;

  vector[NMis[k]] YM;

  int Obs[NObs[k]];
  int Mis[NMis[k]];

  // missing responses distributions
  YM = segment(yMis, ymispos, NMis[k]);
  XM = to_matrix( segment(XMis, xmispos, NMAlphaMis[k]), NMis[k], MAlpha[k]);

  if(nRdm > 0){
    vector[MBeta[k]] currentBeta;
    matrix[NMis[k], MBeta[k]] ZM;

    currentBeta = segment(beta, bpos, MBeta[k]);

    ZM = to_matrix( segment(ZMis, zmispos, NMBetaMis[k]), NMis[k], MBeta[k]);

    if(t_likelihood){
      YM ~ student_t( t_fixedDF, XM * currentAlpha + ZM * currentBeta, sqrt(sigma2Eps[k]) );
    } else {
      YM ~ normal( XM * currentAlpha + ZM * currentBeta, sqrt(sigma2Eps[k]) );
    }

    // NOTE : currentBeta already has a prior defined (in likelihood section of observed data)
  } else {
    if(t_likelihood){
      YM ~ student_t( t_fixedDF, XM * currentAlpha, sqrt(sigma2Eps[k]));
    } else {
      YM ~ normal( XM * currentAlpha, sqrt(sigma2Eps[k]));
    }

  }

  // likelihood of non missingness of observed responses
  currentGamma = segment(gamma, cpos, MGamma[k]);
  TO = to_matrix( segment(TObs, tobspos, NMGammaObs[k]), NObs[k], MGamma[k]);
  Obs = rep_array(1, NObs[k]);

  { vector[NObs[k]] temp;
    if(NMAR){
      temp = Phi_approx( TO * currentGamma + (YO - refIntensity[k]) * gammY[k] );
    } else {
      temp = Phi_approx( TO * currentGamma );
    }
    for(n in 1:NObs[k]){
      Obs[n] ~ bernoulli(fmin(fmax(temp[n], 0.000001), 0.999999));
    }
  }


  // likelihood of missingness of missing responses
  TM = to_matrix( segment(TMis, tmispos, NMGammaMis[k]), NMis[k], MGamma[k]);
  Mis = rep_array(0, NMis[k]);

  { vector[NMis[k]] temp;
    if(NMAR){
      temp = Phi_approx( TM * currentGamma + (YM - refIntensity[k]) * gammY[k] );
    } else {
      temp = Phi_approx( TM * currentGamma );
    }
    for(n in 1:NMis[k]){
      Mis[n] ~ bernoulli(fmin(fmax(temp[n], 0.000001), 0.999999));
    }
  }
```

```
      // priors for gamma and gammaY
      if(gammaHierarch == 1){
        segment(gamma, cpos, MGamma[k]) ~ normal( mu0gm[1:MGamma[k]], sqrt(sigma20gm[1:MGamma[k]]) );

        // NMAR coefficient
        if(NMAR){
          gammY[k] ~ normal(mu0gm[MGamma[k]+1], sqrt(sigma20gm[MGamma[k]+1]));
        }
      } else {
        // intercept
        gamma[cpos] ~ normal(priorGammaInterceptMean, priorGammaInterceptSd);
        // other MAR coefficients
        if(MGamma[k] > 1){
          segment(gamma, cpos+1, MGamma[k]-1) ~ normal( priorGammaTermsMean, priorGammaTermsSd);
        }
        // NMAR coefficient
        if(NMAR){
          gammY[k] ~ normal(priorGammaYMean, priorGammaYSd);
        }
      }


      cpos += MGamma[k];
      ymispos += NMis[k];
      xmispos += NMAlphaMis[k];
      zmispos += NMBetaMis[k];
      tobspos += NMGammaObs[k];
      tmispos += NMGammaMis[k];

    } // end if(modelMiss)

    apos += MAlpha[k];
    bpos += MBeta[k];
    spos += nRdm;
  } // end loop on proteins

  // priors for sigma
  if(sigmaHierarch == 1){
    sigma2Eps ~ scaled_inv_chi_square(nu0Eps[1], sqrt(s0Eps2[1]));
  } else {
    sigma2Eps ~ scaled_inv_chi_square(priorNuEpsilon, priorScaleEpsilon);
  }

  if(nRdm > 0){
    if(sigmaRdmHierarch == 1){
      vector[nRdm] currentSigma2Rdm;
      spos = 1;
      for(k in 1:K){
        currentSigma2Rdm = segment(sigma2Rdm, spos, nRdm);
        currentSigma2Rdm ~ scaled_inv_chi_square(nu0Rdm, sqrt(s0Rdm2));
        spos += nRdm;
      }

    } else {
      sigma2Rdm ~ scaled_inv_chi_square(priorNuRdm, priorScaleRdm);
    }
  }


  // hyper parameters priors => no priors !!


}

generated quantities {

    //other ways to express sigma2 parameters
    real sigmaEps[K];
    real tauEps[K];
    vector [nRdm*K] sigmaRdm;
    vector [nRdm*K] tauRdm;
    vector [modelMiss*gammaHierarch?(MGammaMax+NMAR):0] tau0gm;
    vector [modelMiss*gammaHierarch?(MGammaMax+NMAR):0] sigma0gm;

    // contrasts
    vector [NContrastSum] contrast;


    sigmaEps = sqrt(sigma2Eps);
    tauEps = inv(sigma2Eps);
    if(nRdm>0){
      sigmaRdm = sqrt(sigma2Rdm);
      tauRdm = inv(sigma2Rdm);
    }
    if(modelMiss * gammaHierarch){
      tau0gm = inv(sigma20gm);
      sigma0gm = sqrt(sigma20gm);
```

```
  }

  {
    int aPos;
    int ctrPos;
    int ctrCoefPos;
    aPos = 1;
    ctrCoefPos = 1;
    ctrPos = 1;
    for(k in 1:K){
      if(nContrast[k]>0){
        for(c in 1:nContrast[k]){
          contrast[ctrPos] =
            sum(alpha[aPos:(aPos+MAlpha[k]-1)] .*
                contrastCoefs[ctrCoefPos:(ctrCoefPos+MAlpha[k]-1)]);
          ctrCoefPos += MAlpha[k];
          ctrPos += 1;
        }
      }

      aPos += MAlpha[k];
    }
  }

}
```

G.2 Code of Stan Laplace approximation

In the following, one provides the code of the Stan model script that is used for Laplace approximation with parameters split (Section 4.4.3).

```
// saved as PosteriorApproximation2.stan

data {
  int<lower=1> K;                    // number of proteins
  int<lower=0> nRdm;                 // number of random effects
  int<lower=1> NObs[K];              // number of observations per protein
  int<lower=1> NMis[K];              // number of missing data per protein
  int<lower=1> MAlpha[K];            // number of alpha's (fixed effects coefs) per protein
  int<lower=0> MBeta[K];             // number of beta's (random effects coefs) per protein
  int<lower=1> MBetaPerRdmEffect[nRdm*K]; // same as before, per random effect
  int<lower=1> MGamma[K];            // number of gamma coeffs (missingness model)
  int<lower=1> NObsSum;              // total number of observations (=sum(NObs))
  int<lower=1> NMisSum;              // total number of missing data (=sum(NMis))
  int<lower=1> XObsLen;              // dimension of the flat vector containing all model matrices
                                     // for the fixed effects of the observed responses
  int<lower=1> XMisLen;              // dimension of the flat vector containing all model matrices
                                     // for the fixed effects of the missing responses
  int<lower=0> ZObsLen;              // dimension of the flat vector containing all model matrices
                                     // for the random effects of the observed responses
  int<lower=0> ZMisLen;              // dimension of the flat vector containing all model matrices
                                     // for the random effects of the missing responses
  int<lower=1> TObsLen;              // dimension of the flat vector containing all model matrices
                                     // for the fixed effects of the missing indicators equal 0
  int<lower=1> TMisLen;              // dimension of the flat vector containing all model matrices
                                     // for the fixed effects of the missing indicators equal 1

  row_vector[XObsLen] XObs;          // concatenation of model matrices in a flat row vector format,
                                     // for the fixed effects of the observed responses
  row_vector[XMisLen] XMis;          // concatenation of model matrices in a flat row vector format,
                                     // for the fixed effects of the missing responses
  vector[NObsSum] yObs;              // observed responses (log-2 intensities of peptides)
  row_vector[ZObsLen] ZObs;          // concatenation of model matrices in a flat row vector format,
                                     // for the random effects of the observed responses
  row_vector[ZMisLen] ZMis;          // concatenation of model matrices in a flat row vector format,
                                     // for the random effects of the missing responses
  row_vector[TObsLen] TObs;          // concatenation of model matrices in a flat row vector format,
                                     // for the fixed effects of the missing indicators equal 0
  row_vector[TMisLen] TMis;          // concatenation of model matrices in a flat row vector format,
                                     // for the fixed effects of the missing indicators equal 1

  // general model settings
  int<lower=0, upper=1> t_likelihood;      // use student likelihood (robust version),
                                           // otherwise normal likelihood
  int<lower=1> t_fixedDF;                  // the fixed df parameter for the student likelihood
  int<lower=0, upper=1> modelMiss;         // model missingness (1) or not (0)
  int<lower=0, upper=1> NMAR;              // include NMAR component, yes (1) or no (0)
  vector[K] refIntensity;                  // ref intensity set in the NMAR term
                                           // of the missingness model
  int<lower=0, upper=1> sigmaHierarch;     // hierarchical set-up for sigma (1) or not (0)
  int<lower=0, upper=1> sigmaRdmHierarch;  // hierarchical set-up for sigma of random effects (1)
                                           // or not (0)
  int<lower=0, upper=1> gammaHierarch;     // hierarchical set-up for gamma (1) or not (0)

  // priors for intensity model
  real priorIntensityMean;             // prior distribution for intercept : mean
  real priorIntensitySd;               // prior distribution for intercept : std dev
  real priorFoldChangeMean;            // prior distribution for fold changes : mean
  real priorFoldChangeSd;              // prior distribution for fold changes : std dev
  real priorNuEpsilon;                 // prior inv chi2 distribution for error variances :
                                       // nu parameter
  real priorScaleEpsilon;              // prior inv chi2 distribution for error variances :
                                       // scale parameter
  real priorNuRdm;                     // prior inv chi2 distribution for random effects variances :
                                       // nu parameter
  real priorScaleRdm;                  // prior inv chi2 distribution for random effects variances :
                                       // scale parameter

  // priors for missingness model
  real priorGammaInterceptMean;        // prior normal distribution for intercept
                                       // in missingness model : mean
  real priorGammaInterceptSd;          // prior normal distribution for intercept
                                       // in missingness model : std dev
  real priorGammaTermsMean[max(MGamma[1]-1,1)];// prior normal distribution for the other MAR coefficients
                                       // in missingness model : mean
  real priorGammaTermsSd[max(MGamma[1]-1,1)];  // prior normal distribution for the other MAR coefficients
                                       // in missingness model : std dev
  real priorGammaYMean;                // prior normal distribution for NMAR coefficient
                                       // in missingness model : mean
  real priorGammaYSd;                  // prior normal distribution for NMAR coefficient
                                       // in missingness model : std dev
```

```
  // contrasts
  int<lower=0> nContrast[K];           // nb of contrasts per protein
  int<lower=0> contrastCoefLen;        // dimension of the flat vector containing all contrast coefficients
  vector[contrastCoefLen] contrastCoefs;// concatenation of all constrast coefficient vectors

}


transformed data {
  int NMAlphaObs[K];
  int NMBetaObs[K];
  int NMGammaObs[K];
  int NMAlphaMis[modelMiss?K:0];
  int NMBetaMis[modelMiss?K:0];
  int NMGammaMis[modelMiss?K:0];
  int MAlphaBeta[K];
  int MMAlphaBeta[K];
  int MAlphaSum;
  int MBetaSum;
  int MGammaSum;
  int MGammaMax;
  int MAlphaBetaSum;
  int MMAlphaBetaSum;
  int NContrastSum;

  // store N*M for all k
  for(k in 1:K){
    NMAlphaObs[k] = NObs[k] * MAlpha[k];
    NMBetaObs[k] = NObs[k] * MBeta[k];
    NMGammaObs[k] = NObs[k] * MGamma[k];
    MAlphaBeta[k] = MAlpha[k]+MBeta[k];
    MMAlphaBeta[k] = (MAlpha[k]+MBeta[k]) * (MAlpha[k]+MBeta[k]);
    if(modelMiss){
      NMAlphaMis[k] = NMis[k] * MAlpha[k];
      NMBetaMis[k] = NMis[k] * MBeta[k];
      NMGammaMis[k] = NMis[k] * MGamma[k];
    }
  }
  MAlphaSum = sum(MAlpha);
  MBetaSum = sum(MBeta);
  MGammaSum = sum(MGamma);
  MGammaMax = max(MGamma);
  MAlphaBetaSum = sum(MAlphaBeta);
  MMAlphaBetaSum = sum(MMAlphaBeta);
  NContrastSum = sum(nContrast);
}

parameters {
  // parameters of intensity model
  // alpha, beta are here fixed, depending on the value of the other parameters
  // variance parameters are parametrized in log()
  vector[K] logSigma;                                  // log of error std dev
  vector[nRdm*K] logSigmaRdm;                          // log of std dev of random effects
  vector[t_likelihood*NObsSum] logBObs;                // log of b auxiliary variables
                                                       // (for student t likelihood)
  vector[t_likelihood*modelMiss*NMisSum] logBMis; // log of b auxiliary variables
                                                       // (for student t likelihood)

  // hyperparameters of hierarchical intensity model
  real<lower=1> nu0Eps[sigmaHierarch?1:0];             // degree of freedom hyperparameter
                                                       // for error variances
  real<lower=0> s0Eps2[sigmaHierarch?1:0];             // scale hyperparameter for error variances
  real<lower=1> nu0Rdm[nRdm*sigmaRdmHierarch];         // DoF for variances of rdm effects
  real<lower=0> s0Rdm2[nRdm*sigmaRdmHierarch];         // scale for variances of rdm effects

  // * parameters of missingness model *

  // coefficients for predictors in probit regression of missingness (MAR)
  vector[modelMiss?MGammaSum:0] gamma;
  // coefficients for Y predictor in probit regression of missingness (NMAR)
  vector[modelMiss*NMAR?K:0] gammY;
  // missing responses
  vector[modelMiss?NMisSum:0] yMis;

  // * hyperparameters of hierarchical missingness model *

  // mean hyperparameters for gamma parameters
  vector [modelMiss*gammaHierarch?(MGammaMax+NMAR):0] mu0gm;
  // scale hyperparameters for gamma parameters
  vector<lower=0> [modelMiss*gammaHierarch?(MGammaMax+NMAR):0] sigma20gm;
}

transformed parameters{

  vector[MAlphaBetaSum] AllMuqAlBet;
  vector[MMAlphaBetaSum] AllInvSigqAlBet;

  {
    int yobspos;
    int zobspos;
```

```
int xobspos;
int tobspos;
int ymispos;
int zmispos;
int xmispos;
int tmispos;
int apos;
int bpos;
int spos;

int muAlphaBetapos;
int invSigmaAlphaBetapos;

yobspos = 1;
xobspos = 1;
zobspos = 1;
tobspos = 1;
ymispos = 1;
xmispos = 1;
zmispos = 1;
tmispos = 1;
apos = 1;
bpos = 1;
spos = 1;

muAlphaBetapos = 1;
invSigmaAlphaBetapos = 1;


for(k in 1:K){
  vector[NObs[k]] YO;
  matrix[NObs[k], MAlpha[k]] XO;
  matrix[NObs[k], MBeta[k]] ZO;
  matrix[modelMiss?NObs[k]:0, modelMiss?MGamma[k]:0] TO;

  vector[modelMiss?NMis[k]:0] YM;
  matrix[modelMiss?NMis[k]:0, MBeta[k]] ZM;
  matrix[modelMiss?NMis[k]:0, modelMiss?MAlpha[k]:0] XM;
  matrix[modelMiss?NMis[k]:0, modelMiss?MGamma[k]:0] TM;

  matrix[modelMiss?(NObs[k]+NMis[k]):NObs[k], MAlpha[k] + MBeta[k]] W;
  matrix[NObs[k], MAlpha[k] + MBeta[k]] WO;
  matrix[modelMiss?NMis[k]:0, MAlpha[k] + MBeta[k]] WM;

  vector[NObs[k]] invBO;
  vector[modelMiss?NMis[k]:0] invBM;
  matrix[modelMiss?(NObs[k]+NMis[k]):NObs[k], modelMiss?(NObs[k]+NMis[k]):NObs[k]] D;
  matrix[NObs[k], NObs[k]] DO;
  matrix[modelMiss?NMis[k]:0, modelMiss?NMis[k]:0] DM;

  vector[modelMiss?NObs[k]:0] aO;
  vector[modelMiss?NMis[k]:0] aM;
  vector[modelMiss?(NObs[k]+NMis[k]):0] a;

  real invSigma2Eps;
  real sigma2Eps;

  vector[nRdm] invSigma2Rdm;
  vector[nRdm] currentLogSigmaRdm;
  vector[nRdm] sigma2Rdm;


  vector[MAlpha[k]] currentAlpha;
  vector[MBeta[k]] currentBeta;
  vector[modelMiss?MGamma[k]:0] currentGamma;
  real currentGammY;

  vector[MAlpha[k]] alphaInvVarPriors;
  matrix[MAlpha[k]+MBeta[k], MAlpha[k]+MBeta[k]] invSigmaAlphaBeta;

  matrix[MAlphaBeta[k], MAlphaBeta[k]] invSigmaqAlphaBeta;

  vector[MAlpha[k]] alphaMeanPriors;
  vector[MAlpha[k] + MBeta[k]] muAlphaBeta;
  vector[MAlphaBeta[k]] muqAlphaBeta;

  // initialize vectors and matrices
  YO = segment(yObs, yobspos, NObs[k]);
  XO = to_matrix( segment(XObs, xobspos, NMAlphaObs[k]), NObs[k], MAlpha[k]);

  if(modelMiss){
    YM = segment(yMis, ymispos, NMis[k]);
    XM = to_matrix( segment(XMis, xmispos, NMAlphaMis[k]), NMis[k], MAlpha[k]);
    TO = to_matrix( segment(TObs, tobspos, NMGammaObs[k]), NObs[k], MGamma[k]);
    TM = to_matrix( segment(TMis, tmispos, NMGammaMis[k]), NMis[k], MGamma[k]);
  }

  if(nRdm > 0){
    ZO = to_matrix( segment(ZObs, zobspos, NMBetaObs[k]), NObs[k], MBeta[k]);
    WO = append_col(XO, ZO);
```

```
    if(modelMiss){
      ZM = to_matrix( segment(ZMis, zmispos, NMBetaMis[k]), NMis[k], MBeta[k]);
      WM = append_col(XM, ZM);
      W = append_row(WO, WM);
    }
    else{
      W = append_col(XO, ZO);
    }
  } else {
    WO = XO;
    if(modelMiss){
      WM = XM;
      W = append_row(WO, WM);
    } else {
      W = WO;
    }
  }

  invSigma2Eps = exp(-2*logSigma[k]);
  if(t_likelihood){
    invBO = exp(-segment(logBObs, yobspos, NObs[k]));
    if(modelMiss){
      invBM = exp(-segment(logBMis, ymispos, NMis[k]));
    }
  } else {
    invBO = rep_vector(1., NObs[k]);
    if(modelMiss){
      invBM = rep_vector(1., NMis[k]);
    }
  }
  DO = diag_matrix( invBO );
  if(modelMiss){
    DM = diag_matrix( invBM );
    D = diag_matrix(append_row(invBO, invBM));
  } else {
    D = DO;
  }

  if(nRdm > 0){
    currentLogSigmaRdm = segment(logSigmaRdm, spos, nRdm);
    invSigma2Rdm = exp(-2*currentLogSigmaRdm);
    sigma2Rdm = inv(invSigma2Rdm);
  }

  // reorder priors for alpha and beta => compute related matrices and vectors
  if(MAlpha[k] > 1){
    alphaInvVarPriors = append_row(1/(priorIntensitySd*priorIntensitySd),
    rep_vector(1/(priorFoldChangeSd*priorFoldChangeSd), MAlpha[k]-1));
    alphaMeanPriors = append_row(priorIntensityMean,
    rep_vector(priorFoldChangeMean, MAlpha[k]-1));
  } else {
    alphaInvVarPriors = rep_vector(1/(priorIntensitySd*priorIntensitySd), 1);
    alphaMeanPriors = rep_vector(priorIntensityMean, 1);
  }
  if(nRdm > 0){
    vector[MBeta[k]] betaInvVarPriors;
    int ipos;
    ipos = 1;
    for(r in 1:nRdm){
      for(i in ipos:(ipos+MBetaPerRdmEffect[spos+r-1]-1)){
        betaInvVarPriors[i] = invSigma2Rdm[r];
      }
      ipos += MBetaPerRdmEffect[spos+r-1];
    }
    invSigmaAlphaBeta = diag_matrix(append_row(alphaInvVarPriors, betaInvVarPriors));
    muAlphaBeta = append_row(alphaMeanPriors, rep_vector(0., MBeta[k]));
  } else {
    invSigmaAlphaBeta = diag_matrix(alphaInvVarPriors);
    muAlphaBeta = alphaMeanPriors;
  }

  // STEP 1 :
  // precompute fixed values for mu and sigma of (alpha, beta, YMis)
  // and calculate logPost term linked to determinant()
  invSigmaqAlphaBeta = invSigma2Eps * W'* D * W + invSigmaAlphaBeta;
      muqAlphaBeta = invSigma2Eps * W'* D * append_row(YO, YM) + invSigmaAlphaBeta * muAlphaBeta;
  muqAlphaBeta = invSigmaqAlphaBeta \ muqAlphaBeta; // A \ b = inverse(A) * b

  // store mu and Sigma for alpha, beta, YMis
  AllMuqAlBet[muAlphaBetapos:(muAlphaBetapos+MAlphaBeta[k]-1)] =
    muqAlphaBeta;
  AllInvSigqAlBet[invSigmaAlphaBetapos:(invSigmaAlphaBetapos+MMAlphaBeta[k]-1)] =
    to_vector(invSigmaqAlphaBeta);

  // update postion indexes
  apos += MAlpha[k];
  bpos += MBeta[k];
  yobspos += NObs[k];
  xobspos += NMAlphaObs[k];
  zobspos += NMBetaObs[k];
```

```
      spos += nRdm;

      if(modelMiss){
        ymispos += NMis[k];
        xmispos += NMAlphaMis[k];
        zmispos += NMBetaMis[k];
        tobspos += NMGammaObs[k];
        tmispos += NMGammaMis[k];
      }

      muAlphaBetapos += MAlphaBeta[k];
      invSigmaAlphaBetapos += MMAlphaBeta[k];
    }
  }
}

model {
  int yobspos;
  int zobspos;
  int xobspos;
  int tobspos;
  int ymispos;
  int zmispos;
  int xmispos;
  int tmispos;
  int apos;
  int bpos;
  int cpos;
  int spos;
  int muAlphaBetapos;
  int invSigmaAlphaBetapos;



  yobspos = 1;
  xobspos = 1;
  zobspos = 1;
  tobspos = 1;
  ymispos = 1;
  xmispos = 1;
  zmispos = 1;
  tmispos = 1;
  apos = 1;
  bpos = 1;
  cpos = 1;
  spos = 1;

  muAlphaBetapos = 1;
  invSigmaAlphaBetapos = 1;


  for(k in 1:K){
    vector[NObs[k]] YO;
    matrix[NObs[k], MAlpha[k]] XO;
    matrix[NObs[k], MBeta[k]] ZO;
    matrix[modelMiss?NObs[k]:0, modelMiss?MGamma[k]:0] TO;

    vector[modelMiss?NMis[k]:0] YM;
    matrix[modelMiss?NMis[k]:0, MBeta[k]] ZM;
    matrix[modelMiss?NMis[k]:0, modelMiss?MAlpha[k]:0] XM;
    matrix[modelMiss?NMis[k]:0, modelMiss?MGamma[k]:0] TM;

    vector[NObs[k]] invBO;
    vector[modelMiss?NMis[k]:0] invBM;

    vector[NObs[k]] BO;
    vector[modelMiss?NMis[k]:0] BM;

    real invSigma2Eps;
    real sigma2Eps;
    vector[NObs[k]] sigmaEpsO;
    vector[modelMiss?NMis[k]:0] sigmaEpsM;
    vector[nRdm] invSigma2Rdm;
    vector[nRdm] currentLogSigmaRdm;
    vector[nRdm] sigma2Rdm;


    vector[MAlpha[k]] currentAlpha;
    vector[MBeta[k]] currentBeta;
    vector[modelMiss?MGamma[k]:0] currentGamma;
    real currentGammY;

    matrix[MAlphaBeta[k], MAlphaBeta[k]] invSigmaqAlphaBeta;

    vector[MAlphaBeta[k]] muqAlphaBeta;

    // initialize vectors and matrices
    YO = segment(yObs, yobspos, NObs[k]);
    XO = to_matrix( segment(XObs, xobspos, NMAlphaObs[k]), NObs[k], MAlpha[k]);
```

```
if(modelMiss){
  YM = segment(yMis, ymispos, NMis[k]);
  XM = to_matrix( segment(XMis, xmispos, NMAlphaMis[k]), NMis[k], MAlpha[k]);
  TO = to_matrix( segment(TObs, tobspos, NMGammaObs[k]), NObs[k], MGamma[k]);
  TM = to_matrix( segment(TMis, tmispos, NMGammaMis[k]), NMis[k], MGamma[k]);
}

if(nRdm > 0){
  ZO = to_matrix( segment(ZObs, zobspos, NMBetaObs[k]), NObs[k], MBeta[k]);
  if(modelMiss){
    ZM = to_matrix( segment(ZMis, zmispos, NMBetaMis[k]), NMis[k], MBeta[k]);
  }
}

invSigma2Eps = exp(-2*logSigma[k]);
if(t_likelihood){
  invBO = exp(-segment(logBObs, yobspos, NObs[k]));
  BO = inv(invBO);
  if(modelMiss){
    invBM = exp(-segment(logBMis, ymispos, NMis[k]));
    BM = inv(invBM);
  }
} else {
  invBO = rep_vector(1., NObs[k]);
  if(modelMiss){
    invBM = rep_vector(1., NMis[k]);
  }
}

sigmaEpsO = sqrt(inv(invBO * invSigma2Eps));
if(modelMiss){
  sigmaEpsM = sqrt(inv(invBM * invSigma2Eps));
}
sigma2Eps = inv(invSigma2Eps);

if(nRdm > 0){
  currentLogSigmaRdm = segment(logSigmaRdm, spos, nRdm);
  invSigma2Rdm = exp(-2*currentLogSigmaRdm);
  sigma2Rdm = inv(invSigma2Rdm);
}

if(modelMiss){
  currentGamma = segment(gamma, cpos, MGamma[k]);
  if(NMAR){
    currentGammY = gammY[k];
  } else {
    currentGammY = 0.;
  }
}

muqAlphaBeta = AllMuqAlBet[muAlphaBetapos:(muAlphaBetapos+MAlphaBeta[k]-1)];
invSigmaqAlphaBeta = to_matrix(
    AllInvSigqAlBet[invSigmaAlphaBetapos:(invSigmaAlphaBetapos+MMAlphaBeta[k]-1)],
    MAlphaBeta[k], MAlphaBeta[k]);

currentAlpha = segment(muqAlphaBeta, 1, MAlpha[k]);
if(nRdm > 0){
  currentBeta = segment(muqAlphaBeta, MAlpha[k]+1, MBeta[k]);
}

target += -0.5 * log(determinant(invSigmaqAlphaBeta));

// STEP 2 : now calculate the remaining terms of the log posterior

// likelihood of observed responses


if(nRdm > 0){


  int rpos = 1;

  YO ~ normal( XO * currentAlpha + ZO * currentBeta, sigmaEpsO );

  // likelihood for current Beta
  for(r in 1:nRdm){
    int MCol = MBetaPerRdmEffect[(k-1)*nRdm+r];
    segment(currentBeta, rpos, MCol) ~
        normal(0, sqrt(sigma2Rdm[r]));
    rpos += MCol;
  }

} else {
  YO ~ normal( XO * currentAlpha, sigmaEpsO );
}



if (modelMiss == 1){
```

```
    int Obs[NObs[k]];
    int Mis[NMis[k]];

    Obs = rep_array(1, NObs[k]);
    Mis = rep_array(0, NMis[k]);

    // missing responses distributions

    if(nRdm > 0){
      YM ~ normal( XM * currentAlpha + ZM * currentBeta, sigmaEpsM);

      // NOTE : currentBeta already has a prior defined (in likelihood section of observed data)
    } else {
      YM ~ normal( XM * currentAlpha, sigmaEpsM);
    }

    // likelihood of non missingness of observed responses

    { vector[NObs[k]] temp;
      if(NMAR){
        temp = Phi_approx( TO * currentGamma + (YO - refIntensity[k]) * gammY[k] );
      } else {
        temp = Phi_approx( TO * currentGamma );
      }
      for(n in 1:NObs[k]){
        Obs[n] ~ bernoulli(fmin(fmax(temp[n], 0.000001), 0.999999));
      }
    }


    // likelihood of missingness of missing responses
    { vector[NMis[k]] temp;
      if(NMAR){
        temp = Phi_approx( TM * currentGamma + (YM - refIntensity[k]) * gammY[k] );
      } else {
        temp = Phi_approx( TM * currentGamma );
      }
      for(n in 1:NMis[k]){
        Mis[n] ~ bernoulli(fmin(fmax(temp[n], 0.000001), 0.999999));
      }
    }


    // priors for gamma and gammaY
    if(gammaHierarch == 1){
      segment(gamma, cpos, MGamma[k]) ~ normal( mu0gm[1:MGamma[k]], sqrt(sigma20gm[1:MGamma[k]]) );

      // NMAR coefficient
      if(NMAR){
        gammY[k] ~ normal(mu0gm[MGamma[k]+1], sqrt(sigma20gm[MGamma[k]+1]));
      }
    } else {
      // intercept
      gamma[cpos] ~ normal(priorGammaInterceptMean, priorGammaInterceptSd);
      // other MAR coefficients
      if(MGamma[k] > 1){
        segment(gamma, cpos+1, MGamma[k]-1) ~ normal( priorGammaTermsMean, priorGammaTermsSd);
      }
      // NMAR coefficient
      if(NMAR){
        gammY[k] ~ normal(priorGammaYMean, priorGammaYSd);
      }
    }

  } // end if(modelMiss)

  // priors for sigma
  if(sigmaHierarch == 1){
    sigma2Eps ~ scaled_inv_chi_square(nu0Eps[1], sqrt(s0Eps2[1]));
  } else {
    sigma2Eps ~ scaled_inv_chi_square(priorNuEpsilon, priorScaleEpsilon);
  }

  if(nRdm > 0){
    if(sigmaRdmHierarch == 1){
      vector[nRdm] currentSigma2Rdm;
      spos = 1;
      sigma2Rdm ~ scaled_inv_chi_square(nu0Rdm, sqrt(s0Rdm2));
    } else {
      sigma2Rdm ~ scaled_inv_chi_square(priorNuRdm, priorScaleRdm);
    }
  }

  // priors for b
  if(t_likelihood){
    BO ~ inv_gamma(t_fixedDF/2., t_fixedDF/2.);
    if(modelMiss){
      BM ~ inv_gamma(t_fixedDF/2., t_fixedDF/2.);
    }
```

```
    }

    // STEP 3 : add additional terms due to change of variables :
    // log(sigmaEps), log(sigmaRdm), log(b)
    target += log(sigma2Eps);
    if(nRdm > 0){
      target += sum(log(sigma2Rdm));
    }

    if(t_likelihood){
      target += - sum(log(invBO));
      if(modelMiss){
        target += - sum(log(invBM));
      }
    }


    // STEP 4 : update position indicators
    apos += MAlpha[k];
    bpos += MBeta[k];
    cpos += MGamma[k];
    yobspos += NObs[k];
    xobspos += NMAlphaObs[k];
    zobspos += NMBetaObs[k];
    spos += nRdm;

    if(modelMiss){
      ymispos += NMis[k];
      xmispos += NMAlphaMis[k];
      zmispos += NMBetaMis[k];
      tobspos += NMGammaObs[k];
      tmispos += NMGammaMis[k];
    }

    muAlphaBetapos += MAlphaBeta[k];
    invSigmaAlphaBetapos += MMAlphaBeta[k];
  } // end loop on proteins

  // hyper parameters priors => no priors !!

}

generated quantities {

  vector[MAlphaSum] alpha;                              // coefficients for fixed effects
  vector[nRdm>0?MBetaSum:0] beta;                       // coefficients for random effects

  vector[K] sigmaEps;
  vector[K] tauEps;
  vector[K] sigma2Eps;
  vector [nRdm*K] sigmaRdm;
  vector [nRdm*K] tauRdm;
  vector [nRdm*K] sigma2Rdm;
  vector [t_likelihood*NObsSum] invBObs;
  vector [t_likelihood*modelMiss*NMisSum] invBMis;

  // contrasts
  vector [NContrastSum] contrast;


  // generate samples for alpha, beta, yMis
  {

    int apos;
    int bpos;
    int ymispos;
    int muAlphaBetapos;
    int invSigmaAlphaBetapos;

    apos = 1;
    bpos = 1;
    ymispos = 1;

    muAlphaBetapos = 1;
    invSigmaAlphaBetapos = 1;

    for(k in 1:K){
      vector[MAlphaBeta[k]] muqAlphaBeta;
      vector[MAlphaBeta[k]] alphaBeta;
      matrix[MAlphaBeta[k], MAlphaBeta[k]] invSigmaqAlphaBeta;

      muqAlphaBeta = AllMuqAlBet[muAlphaBetapos:(muAlphaBetapos+MAlphaBeta[k]-1)];

      invSigmaqAlphaBeta = to_matrix(
        AllInvSigqAlBet[invSigmaAlphaBetapos:(invSigmaAlphaBetapos+MMAlphaBeta[k]-1)],
        MAlphaBeta[k], MAlphaBeta[k]);

      alphaBeta = multi_normal_rng(muqAlphaBeta, inverse(invSigmaqAlphaBeta));
      //alphaBeta = muqAlphaBeta;
      alpha[apos:(apos+MAlpha[k]-1)] = alphaBeta[1:MAlpha[k]];
```

```
      if(nRdm > 0){
        beta[bpos:(bpos+MBeta[k]-1)] = alphaBeta[(MAlpha[k]+1):(MAlpha[k]+MBeta[k])];
      }

      apos += MAlpha[k];
      bpos += MBeta[k];
      if(modelMiss){
        ymispos += NMis[k];
      }
      muAlphaBetapos += MAlphaBeta[k];
      invSigmaAlphaBetapos += MMAlphaBeta[k];
    }
  }

  sigmaEps = exp(logSigma);
  sigma2Eps = sigmaEps .* sigmaEps;
  tauEps = inv(sigma2Eps);
  if(nRdm>0){
    sigmaRdm = exp(logSigmaRdm);
    sigma2Rdm = sigmaRdm .* sigmaRdm;
    tauRdm = inv(sigma2Rdm);
  }
  if(t_likelihood){
    invBObs = exp(-logBObs);
    if(modelMiss){
      invBMis = exp(-logBMis);
    }

  }

  {
    int aPos;
    int ctrPos;
    int ctrCoefPos;
    aPos = 1;
    ctrCoefPos = 1;
    ctrPos = 1;
    for(k in 1:K){
      if(nContrast[k]>0){
        for(c in 1:nContrast[k]){
          contrast[ctrPos] =
            sum(alpha[aPos:(aPos+MAlpha[k]-1)] .*
                contrastCoefs[ctrCoefPos:(ctrCoefPos+MAlpha[k]-1)]);
          ctrCoefPos += MAlpha[k];
          ctrPos += 1;
        }
      }

      aPos += MAlpha[k];
    }
  }

}
```

Bibliography

Ahlmann-Eltze, C. and Anders, S. (2020). "proDA: Probabilistic Dropout Analysis for Identifying Differentially Abundant Proteins in Label-Free Mass Spectrometry". In: *BioRxiv*.

Anderson, N. and Leigh Anderson, N. (1998). "Proteome and proteomics: New technologies, new concepts, and new words". In: *Electrophoresis* 19, pp. 1853–1861.

Araújo, A. M. et al. (2017). "Metabolomic approaches in the discovery of potential urinary biomarkers of drug-induced liver injury (DILI)". en. In: *Crit. Rev. Toxicol.* 47.8, pp. 633–649.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Ed. by N. Y. Springer.

Blei, D. M., Kucukelbir, A., and McAuliffe, J. D. (2017). "Variational Inference: A Review for Statisticians". In: *J. Am. Stat. Assoc.* 112.518, pp. 859–877.

Bolstad, B. M. et al. (2003). "A comparison of normalization methods for high density oligonucleotide array data based on variance and bias". en. In: *Bioinformatics* 19.2, pp. 185–193.

Carpenter, B. et al. (2017). "Stan: A probabilistic programming language". In: *Journal of statistical software* 76.1.

Clement, L. (2019). *Proteomics Data Analysis Shortcourse (statOmics UGent Course Notes)*.

Coulibaly, R. (2019). "Approche Bayésienne Variationnelle dans des Modèles Semi-Paramétriques de Régressions". MA thesis. Université Catholique de Louvain.

Croux, C. and Dehon, C. (2014). "Robust Estimation of Location and Scale". In: *Wiley StatsRef: Statistics Reference Online*. Ed. by N. Balakrishnan et al. Vol. 3. Chichester, UK: John Wiley & Sons, Ltd, p. 1045.

Dhingra, V. et al. (2005). "New frontiers in proteomics research: a perspective". en. In: *Int. J. Pharm.* 299.1-2, pp. 1–18.

Elias, J. E. and Gygi, S. P. (2010). "Target-decoy search strategy for mass spectrometry-based proteomics". en. In: *Methods Mol. Biol.* 604, pp. 55–71.

Faber, N. M. (1999). "Estimating the uncertainty in estimates of root mean square error of prediction: application to determining the size of an adequate test set in multivariate calibration". en. In: *Chemometrics Intellig. Lab. Syst.* 49.1, pp. 79–89.

Faes, C., Ormerod, J. T., and Wand, M. P. (2011). "Variational Bayesian Inference for Parametric and Nonparametric Regression With Missing Data". In: *J. Am. Stat. Assoc.* 106.495, pp. 959–971.

Frith, M. C., Pheasant, M., and Mattick, J. S. (2005). "Genomics: The amazing complexity of the human transcriptome". In: *Eur. J. Hum. Genet.* 13.8, pp. 894–897.

Gatto, L. (2019). *Mass spectrometry-based proteomics (LSTAT2340 - UCLouvain Course Notes)*.

Gelman, A. et al. (2013). *Bayesian Data Analysis, Third Edition*. en. CRC Press.

Goeminne, L. J. E., Gevaert, K., and Clement, L. (2016). "Peptide-level Robust Ridge Regression Improves Estimation, Sensitivity, and Specificity in Data-dependent Quantitative Label-free Shotgun Proteomics". en. In: *Mol. Cell. Proteomics* 15.2, pp. 657–668.

Goeminne, L. J. E., Sticker, A., et al. (2020). "MSqRob Takes the Missing Hurdle: Uniting Intensity- and Count-Based Proteomics". en. In: *Anal. Chem.* 92.9, pp. 6278–6287.

Hastings, W. K. (1970). "Monte Carlo sampling methods using Markov chains and their applications". en. In: *Biometrika* 57.1, pp. 97–109.

Hoffman, M. D. and Gelman, A. (2014). "The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo". In: *J. Mach. Learn. Res.* 15.1, pp. 1593–1623.

Holmes, C. C. and Held, L. (2006). "Bayesian auxiliary variable models for binary and multinomial regression". en. In: *Bayesian Anal.* 1.1, pp. 145–168.

Jullion, A. and Lambert, P. (2007). "Robust specification of the roughness penalty prior distribution in spatially adaptive Bayesian P-splines models". In: *Comput. Stat. Data Anal.* 51.5, pp. 2542–2558.

Kelley, C. T. (1999). "4. The BFGS Method". In: *Iterative Methods for Optimization*. Frontiers in Applied Mathematics. Society for Industrial and Applied Mathematics, pp. 71–86.

Lazar, C. et al. (2016). "Accounting for the Multiple Natures of Missing Values in Label-Free Quantitative Proteomics Data Sets to Compare Imputation Strategies". en. In: *J. Proteome Res.* 15.4, pp. 1116–1125.

Mallikarjun, V., Richardson, S. M., and Swift, J. (2020). "BayesENproteomics: Bayesian Elastic Nets for Quantification of Peptidoforms in Complex Samples". en. In: *J. Proteome Res.* 19.6, pp. 2167–2184.

Metropolis, N. et al. (1953). "Equation of state calculations by fast computing machines". In: *J. Chem. Phys.* 21.6, pp. 1087–1092.

Molenberghs, G. et al. (2014). *Handbook of Missing Data Methodology*. en. CRC Press.

Muñoz Maldonado, Y. (2009). "Mixed Models, Posterior Means and Penalized Least-Squares". en. In: *Optimality*. Institute of Mathematical Statistics, pp. 216–236.

Naumann, U. (2012). "The Art of Differentiating Computer Programs - An Introduction to Algorithmic Differentiation". In: *Software, environments, tools*.

Neal, R. M. et al. (2011). "MCMC using Hamiltonian dynamics". In: *Handbook of markov chain monte carlo* 2.11, p. 2.

Ormerod, J. T. and Wand, M. P. (2010). "Explaining Variational Approximations". In: *Am. Stat.* 64.2, pp. 140–153.

Rue, H., Martino, S., and Chopin, N. (2009). "Approximate Bayesian inference for latent Gaussian models by using integrated nested Laplace approximations". In: *J. R. Stat. Soc. Series B Stat. Methodol.* 71.2, pp. 319–392.

Shaffer, J. P. (1995). "Multiple Hypothesis Testing". In: *Annu. Rev. Psychol.* 46, pp. 561–584.

Smyth, G. K. (2004). "Linear models and empirical bayes methods for assessing differential expression in microarray experiments". en. In: *Stat. Appl. Genet. Mol. Biol.* 3, Article3.

Stan Development Team (2020a). *RStan: the R interface to Stan*. R package version 2.21.2. URL: `http://mc-stan.org/`.

— (2020b). *Stan Modelling Language Users Guide and Reference Manual*. version 2.24. URL: `http://mc-stan.org/`.

Sticker, A. et al. (2020). "Robust Summarization and Inference in Proteome-wide Label-free Quantification". en. In: *Mol. Cell. Proteomics* 19.7, pp. 1209–1219.

The, M. and Käll, L. (2019). "Integrated Identification and Quantification Error Probabilities for Shotgun Proteomics". en. In: *Mol. Cell. Proteomics* 18.3, pp. 561–570.

Tipping, M. E. and Lawrence, N. D. (2003). "A variational approach to robust Bayesian interpolation". In: *2003 IEEE XIII Workshop on Neural Networks for Signal Processing (IEEE Cat. No.03TH8718)*, pp. 229–238.

Venables, W. N. and Ripley, B. D. (2002). *Modern Applied Statistics with S*. Fourth. ISBN 0-387-95457-0. New York: Springer. URL: `http://www.stats.ox.ac.uk/pub/MASS4/`.

Wand, M. P. (2003). "Smoothing and mixed models". In: *Comput. Stat.* 18.2, pp. 223–249.