A Bayesian's journey to a better research workflow

Konstantinos Vamvourellis[‡][*], Marianne Corvellec[§]

https://youtu.be/piQvcVala9I

Abstract—This work began when the two authors met at a software development meeting. Konstantinos was building Bayesian models in his research and wanted to learn how to better manage his research process. Marianne was working on data analysis workflows in industry and wanted to learn more about Bayesian statistics. In this paper, the authors present a Bayesian scientific research workflow for statistical analysis. Drawing on a case study in clinical trials, they demonstrate lessons that other scientists, not necessarily Bayesian, could find useful in their own work. Notably, they can be used to improve productivity and reproducibility in any computational research project.

Index Terms—Bayesian statistics, life sciences, clinical trials, probabilistic programming, Stan, PyStan

Introduction

We present a research workflow for Bayesian statistical analysis. We demonstrate lessons we learned from our own computational research that other scientists, not necessarily Bayesian, could find useful when they manage their work. To illustrate these lessons, we use a specific case study in clinical trial modeling.

Clinical trial data are presented to experts and clinicians to assess the efficacy and safety of a given drug. The analysis of trial data is based on statistical summaries of the data, including averages, standard deviations, and significance levels. However, dependencies between the treatment effects are the subject of clinical judgment and are rarely included in the statistical summaries.

We propose a Bayesian approach to model clinical trial data. We use latent variables to account for the whole joint distribution of the treatment effects, including effects of different types. As a result, we can find the predictive distribution of the treatment effects on a new patient accounting for uncertainty in all the parameters, including correlation between the effects.

The analysis is implemented in PyStan, the Python interface to Stan, which is the state-of-the-art, free and open-source Bayesian inference engine. Stan and the researchers behind it provide users with guidance that make Bayesian inference easier to use. We discuss aspects of this ecosystem in the second-to-last section.

Although this case study is by no means the ideal introductory example of computational modeling, it provides us with a real-world problem from which we can share practical lessons. We believe this paper can be of help to a number of different audiences. Firstly, it can help non-Bayesian statisticians, or beginning Bayesians, get a a sense of how to apply Bayesian statistics to their work. Secondly, it can provide computational scientists with advice on building a reproducible and efficient research workflow. And, thirdly, it can spark discussions among advanced Bayesians about the complexities of Bayesian workflows and how to build better models.

* Corresponding author: k.vamvourellis@lse.ac.uk
‡ London School of Economics and Political Science
§ Institute for Globally Distributed Open Research and Education (IGDORE)

A Bayesian Workflow

We present a Bayesian workflow for statistical modeling. We recognize that the research process is too complex to summarize in a recipe-style list. However, we find that there are a few building blocks that are common to every Bayesian statistical analysis. In this paper, we focus on these and break them down to a few basic steps. To avoid over-simplification, we hint to possible connections with more advanced aspects where appropriate. We believe that following a workflow, such as suggested here, can help researchers avoid mistakes and increase productivity. It also helps make research projects more reproducible, as we discuss in the last section.

We propose a simple workflow made of the following steps:

1) Scope the problem;
2) Specify the likelihood and priors;
3) Generate synthetic data that resemble the true data to a reasonable degree;
4) Fit the model to the synthetic data;
    a. Check that the true values are recovered;
    b. Check the model fit;
5) Fit the model to the real data.

An advanced workflow, which is beyond the scope of this paper, could be extended to include the following steps:

6) Check the predictive accuracy of the model;
7) Evaluate the model fit;
8) Select best model among different candidates (model selection);
9) Perform a sensitivity analysis.

In what follows, we will use $M(\theta)$ to denote the model as a function of its parameter θ (θ is either a scalar or a vector representing a set of parameters). Data usually consist of observable outcomes[1] y and covariates[2] x, if any. We will distinguish between the two when necessary; otherwise, we will denote all data together by D. We use $p(\cdot)$ to denote either probability distributions or probability densities, even though it is not rigorous notation.

1) Scope the problem

The main goal of this workflow is to achieve successful Bayesian inference. That is, correctly retrieving samples from the posterior distribution of the parameter values, which are typically unknown before the analysis, using the information contained in the data. The major difference of the Bayesian approach relative to frequentist, is that it modifies the likelihood function (to be introduced later) into a proper distribution over the parameters, called the posterior distribution. The posterior distribution $p(\theta|D)$ forms the basis of the Bayesian approach from which we derive all quantities of interest.

Why do we need statistical inference in the first place? We need it to answer our questions about the world. Usually, our questions refer to an implicit or explicit parameter θ in a statistical model, such as:

- What values of θ are most consistent with the data?
- Do the data support a certain condition (e.g., for θ a scalar, $\theta > 0$)?
- How can we predict the future outcome of an experiment?

To proceed, we need to define a model. Choosing a model is usually tied to the exact research questions we are interested in. We can choose to start with a postulated data generation process and then decide how to interpret the parameters in relation to the research question. Alternatively, it is equally valid to start from the research question and design the model so that its parameters are directly connected to the specific questions we wish to answer. In the next section, we illustrate with an example how to design a model to answer a specific research question.

Note that the question of prediction depends directly on inferring successfully the parameter values. We shall come back to this at the end of this section.

2) Specify the likelihood and priors

Once we have defined the scope of the problem, we need to specify the design of the model which is captured in the *likelihood* function $f(D|\theta, M)$. Usually, argument M is dropped for notational simplicity, the model being chosen and assumed known[3]. Note, however, that when the model includes covariates, the more accurate expression is $f(y|\theta, x)$. This function ties together the ingredients of statistical inference and allows information to flow from the data D to the parameters θ. With Bayes' rule, $p(\theta|D) = p(D|\theta)p(\theta)/p(D)$, we can calculate the posterior distribution.

The second ingredient of Bayesian inference is the prior distribution $p(\theta)$. Priors are inescapably part of the Bayesian approach and, hence, have to be considered carefully. The goal of Bayesian inference is to combine the prior information on the parameters (the prior distribution), with the evidence contained in the data (the likelihood), to derive the posterior distribution $p(\theta|D)$. It is difficult to predict how sensitive the final results will be to a change in the priors. However, it is important to note that the impact of priors progressively diminishes as the number of observations increases.

The ideal scenario for applying the Bayesian approach is when prior knowledge is available, in which case the prior distribution can and should capture that knowledge. But, sometimes, we might want to avoid expressing prior knowledge, especially when such knowledge is not available. How are we supposed to choose priors then? Constructing default priors is an active area of research that is beyond the scope of this work. Here, we provide a high-level overview and refer the interested reader to various sources for further reading.

Priors which express very little or no prior knowledge are called vague or *uninformative priors*. Such priors are deliberately constructed in a way which minimizes their impact on the resulting inference, relative to the information brought in by the likelihood. In fact, Bayesian inference technically works even when the prior is not a proper distribution but a function that assumes all values are equally likely, referred to as *improper prior*. However, it is generally advisable to avoid improper priors, especially in settings beyond just inference, such as the more advanced workflow of steps 6)–9). If no prior knowledge is available, a normal distribution with large variance is still a better default prior than a uniform distribution. It is important to note that improper or even vague priors are not appropriate for model selection.

Additional considerations can impact the choice of priors, especially when chosen together with the likelihood. From a computational perspective, the most convenient priors are called *conjugate priors*, because they mimic the structure of the likelihood function and lead to a closed-form posterior distribution. Priors can have additional benefits when used with a certain goal in mind. For example, priors can be used to guard against overfitting by pulling the parameters away from improbable values, or help with feature selection (e.g., see horse-shoe priors).

Bayesian critics often see priors as a weakness, whereas in reality they are an opportunity. Notably, priors give us the opportunity to employ our knowledge to guide the inference in the absence of evidence from the data. Also, it is important to remember that, in a scientific research context, we rarely have absolutely no prior knowledge and we typically do not consider any parameter value to be equally likely.

3) Generate synthetic data

Once we have agreed on a generative process, i.e., a model M, we can use it to simulate data D'. To do that, we choose reasonable parameter values θ_0 and use M to generate data based on these values. Alternatively, instead of coming up with reasonable parameter values, we can sample these values from the prior distribution $\theta_0 \sim p(\theta)$. The synthetic data D' can then be interpreted as our prior distribution of the data. Hence, by inspecting the synthetic data, we can reflect back on our choices for the likelihood and priors. However, if we do use our priors to generate parameter values, we should make sure that our priors are not uninformative, which would likely produce unreasonable synthetic data.

Note how the model M is a hypothesized process and comes with necessary assumptions and simplifications. It is highly unlikely that the real world would follow exactly M. That being said, if M is close enough to the real generative process, it can still be very useful to help us understand something about the world. As the phrase goes, "all models are wrong, but some models are useful."

4) Fit the model to the synthetic data

1. Depending on their field, readers may want to think 'dependent variables' or 'labels'.

2. Depending on their field, readers may want to think 'independent variables' or 'features'.

3. This is a good time to highlight that the choice of the model is a constant assumption in everything we do from now on. In research projects, it is common to work with a few different models in parallel.

If simulating data using our generative process M is the forward direction, statistical inference is the reverse direction by which we find what parameter values could have produced such data, under M.

The most popular statistical inference algorithm is maximum likelihood estimation (MLE), which finds the parameter values that maximize the likelihood given the observed data. To reiterate, under the Bayesian approach, we treat the parameters θ as random variables and express our prior knowledge about θ with the prior probability distribution $p(\theta)$. Bayesian inference is the process of updating our beliefs about θ in light of the data D. The updating process uses Bayes' rule and results in the conditional distribution $p(\theta|D)$, the posterior distribution. Bayesian inference is generally a hard problem. In most cases, we cannot derive the mathematical form of the posterior distribution; instead, we settle for an algorithm that returns samples from the posterior distribution.

When we fit the model to synthetic data, we want to check two things: the correctness of the inference algorithm and the quality of our model.

a. Much like in software testing, we want to check if the inference process works by starting simple and advance progressively to the real challenge. By fitting the model to synthetic data generated from the same model, we effectively rule out issues of mismatch between our model and the real data. Testing the inference algorithm under these ideal conditions allows us to perfect the inference algorithm in a controlled environment, before trying it on the real data. In our experience, this step brings to the surface many bugs in the code as well as issues about the model in general. It offers an added benefit, later on, when we critique the fit of our model M to the real data D. Having confidence in the correctness of our inference process allows us to attribute any mismatch issues to the choice of the model, as opposed to the inference algorithm.

By fitting the model to synthetic data, we recover samples from the posterior distribution of the model parameters. There are various model fit tests to choose from. At a minimum, we need to check that the true parameter values θ_0 are within the range implied by the posterior distributions[4]. Success at this stage is not a sufficient guarantee that the model will fit well to the real data, but it is a necessary condition for proceeding further.

b. Fitting the model to synthetic data is the first opportunity to critique the model M and, if necessary, calibrate it to better suit our needs. This is a good time to catch any issues that affect the quality of the model irrespective of how well it captures reality. For example, an issue that comes up often is non-identifiability, the situation where the likelihood and the data is specified in a way such that there is not enough information to identify the correct parameter values, no matter how big the sample size is. It is also a good time to check if small variations to the model (such as replacing a normal with a heavier-tail distribution) fit our needs better. For instance, calibrating a model to make inferences about the center of a distribution, such as the mean, is relatively easy. On the other hand, we might need to do more extensive calibration if we are interested in the tail behavior of the distribution, such as maximum values. If we do choose to use a different model M', we need to go back to step 2) and start again.

Model evaluation is an essential part of a good workflow. It is a complex task that can be used with both synthetic and real data, providing possibly different insights each time. We do not have space to go into more details in this paper but we provide pointers in the further reading section.

5. Fit the model to the real data

This is the moment we have been waiting for: We are ready to fit our model to the real data and get the final results. Usually, we focus our attention on a specific quantity of interest that is derived from the posterior samples (see further reading for pointers). If we are satisfied with the fit of the model, we are done. In most cases, though, at this stage we are expected to evaluate the model again, this time focusing on how well it captures reality. This step is highly application-specific and requires a combination of statistical expertise and subject-matter expertise (we refer the interested reader to sources later). We note that it is important to build confidence in the power of our inference algorithm before proceeding on to interpreting results. This helps us separate, to the extent possible, inference issues from model issues. At this stage, it is likely that we will come up with a slightly updated model M'. We then have to go back and start again from the beginning.

Posterior Predictive Checks and Model Evaluation

In this subsection, we would like to touch briefly on two topics for more advanced workflows, prediction and model evaluation. The Bayesian posterior predictive distribution is given by the following formula:

$$\begin{aligned} p(\tilde{y}|D) &= \int p(\tilde{y},\theta|D)d\theta \\ &= \int p(\tilde{y}|\theta)p(\theta|D)d\theta \end{aligned}$$

In practice, we approximate the integral using samples from the posterior distributions. Posterior predictive checks, evaluating the predictive accuracy of a model, can also be used to evaluate a model. To do this, we check how well it predicts unknown observable data $\tilde{y}$, where unknown means that the model was not fit to $\tilde{y}$[5].

Further reading

For a concise overview of statistical modeling and inference, including a high-level comparison with the frequentist approach, see [Woo15]. For a more extended treatment of the Bayesian approach, see [Rob07]. For an accessible Bayesian modeling primer, especially for beginner Bayesians, see [McE15] and [MR06]. For a complete treatment of Bayesian data analysis, including many workflow-related discussions, see [GCS+13][6].

A Case Study in Clinical Trial Data Analysis

We propose a Bayesian model to extract insights from clinical trial datasets. We are interested in understanding the effect of a treatment on the patients. Our goal is to use the data to predict the effect of the treatment on a new patient. We apply our method on artificially created data, for illustration purposes only.

4. A common test is to construct an interval that includes 95% of the most likely values, called highest posterior density interval, and check that it covers the true parameter values θ_0 that were used to generate the synthetic data. We should tolerate a few misses, since 95% intervals will not cover the true values 5% of the time, even if the algorithm is perfectly calibrated.

5. To check the predictive accuracy of the model, we need to measure our predictions $\tilde{y}$ against real data. To do this, we usually hold out a small random sample of the original data and deliberately restrain from fitting the model to that sample.

6. And for an example implementation of a complete workflow with PyStan, see https://github.com/betanalpha/jupyter_case_studies/tree/master/pystan_workflow.

Subject ID	Group Type	Hemoglobin Level	Dyspepsia	Nausea
123	Control	3.42	1	0
213	Treatment	4.41	1	0
431	Control	1.12	0	0
224	Control	-0.11	1	0
233	Treatment	2.42	1	1

TABLE 1: *Toy clinical trial data.*

1) Scope the problem

Regulators focus on a few key effects when deciding whether a drug is fit for market. In our case we will assume, for simplicity, that there are three effects, where two are binary variables and the other is a continuous variable.

Our dataset is organized as a table, with one patient (subject) per row and one effect per column. For example, if our clinical trial dataset records three effects per subject, 'Hemoglobin Levels' (continuous), 'Nausea' (yes/no), and 'Dyspepsia' (yes/no), the dataset looks like Table 1.

The fact that the effects are of mixed data types, binary and continuous, makes it harder to model their interdependencies. To address this challenge, we use a latent variable structure. Then, the expected value of the latent variables will correspond to the average effect of the treatment. Similarly, the correlations between the latent variables will correspond to the correlations between effects. Knowing the distribution of the latent variables will give us a way to predict what the effect on a new patient will be, conditioned on the observed data.

2) Specify the model, likelihood, and priors

a. Model: Let Y be a $N \times K$ matrix where each column represents an effect and each row refers to an individual subject. This matrix contains our observations, it is our clinical trial dataset. We distinguish between treatment and control subjects by considering separately Y^T (resp. Y^C), the subset of Y containing only treatment (resp. control) subjects. Since the model for Y^T and Y^C is identical, for convenience, we suppress the notation into Y in the remainder of this section.

We consider the following general latent variable framework. We assume subjects are independent and wish to model the dependencies between the effects. The idea is to bring all columns to a common scale $(-\infty, \infty)$. The continuous effects are observed directly and are already on this scale. For the binary effects, we apply appropriate transformations on their parameters via user-specified link functions $h_j(\cdot)$, in order to bring them to the $(-\infty, \infty)$ scale. Let us consider the i-th subject. Then, if the j-th effect is measured on the binary scale, the model is

$$\begin{aligned} Y_{ij} &\sim \text{Bernoulli}(\eta_{ij}) \\ h_j(\eta_{ij}) &= Z_{ij}, \end{aligned}$$

where the link function $h_j(\cdot)$ can be the logit, probit, or any other bijection from $[0,1]$ to the real line. Continuous data are assumed to be observed directly and accurately (without measurement error), and modeled as follows:

$$Y_{ij} = Z_{ij} \quad \text{for } i = 1, \ldots, N.$$

In order to complete the model, we need to define the $N \times K$ matrix Z. Here, we use a K-variate normal distribution $N_K(\cdot)$ on each $Z_{i\cdot}$ row, such that

$$Z_{i\cdot} \sim N_K(\mu, \Sigma),$$

where Σ is a $K \times K$ covariance matrix, μ is a row K-dimensional vector, and $Z_{i\cdot}$ are independent for all i.

In the model above, the vector $\mu = (\mu_1, \ldots, \mu_K)$ represents the average treatment effect in the common scale. In our example, the first effect (Hemoglobin Level) is continuous and hence its latent value directly observed. Regarding the remaining two effects (Dyspepsia and Nausea), their latent values can only be inferred via their binary observations. Note that the variance of the non-observed latent variables is non-identifiable [CG98], [TDM12], so we need to fix it to a known constant (here we use 1) to fully specify the model. We do this by decomposing the covariance into correlation and variance: $\Sigma = DRD$, where R is the correlation matrix and D is a diagonal matrix of variances $D_{jj} = \sigma_j^2$ for the j-th effect.

b. Likelihood: The likelihood function can be expressed as

$$\begin{aligned} f(Y|Z,\mu,\Sigma) &= f(Y|Z) \cdot p(Z|\mu,\Sigma) \\ &= [\prod_{j \in J_b} \prod_{i=1}^{N} h_j^{-1}(Z_{ij})^{Y_{ij}} (1 - h_j^{-1}(Z_{ij}))^{(1-Y_{ij})}] \cdot p(Z|\mu,\Sigma) \\ &= [\prod_{j \in J_b} \prod_{i=1}^{N} \eta_{ij}^{Y_{ij}} (1 - \eta_{ij})^{(1-Y_{ij})}] \cdot N(Z|\mu,\Sigma), \end{aligned}$$

where J_b is the index of effects that are binary and $N(Z|\mu,\Sigma)$ is the probability density function (pdf) of the multivariate normal distribution.

c. Priors: In this case study, the priors should come from previous studies of the treatment in question or from clinical judgment. If there was no such option, then it would be up to us to decide on an appropriate prior. We use the following priors for demonstration purposes:

$$\begin{aligned} \mu_i &\sim N(0,10) \\ R &\sim \text{LKJ}(2) \\ \sigma_j &\sim \text{Cauchy}(0,2) \text{ for } j \notin J_b \\ Z_{ij} &\sim N(0,1) \text{ for } j \in J_b. \end{aligned}$$

This will become more transparent in the next section, when we come back to the choice of priors[7]. Let us note that our data contain a lot of information, so the final outcome will be relatively insensitive to the priors.

3) Generate synthetic data

To generate synthetic data, given some values for the parameters (μ, Σ) we only need to follow the recipe given by the model. To fix the parameter values we could sample from the priors we chose, or just choose some reasonable values. Here we picked $\mu = (0.3, 0.5, 0.7)$, $\sigma = (1.3, 1, 1)$, and $R(1,2) = -0.5$, $R(1,3) = -0.3$, $R(2,3) = 0.7$. Then, as the model dictates, we use these values to generate samples of underlying latent variables $Z_{i\cdot} \sim N(\mu, \Sigma)$[8]. Each $Z_{i\cdot}$ corresponds to a subject, here we choose to generate 200 subjects. The observed synthetic data Y_{ij} are defined to be equal to Z_{ij} for the effects that are continuous. For

7. On the LKJ distribution, see https://www.sciencedirect.com/science/article/pii/S0047259X09000876.

the binary effects, we sample Bernoulli variables with probability equal to the inverse logit of the corresponding Z_{ij} value.

Recall that a Bayesian model with proper informative priors, such as the ones we use in this model, can also be used directly to sample synthetic data. As explained in the previous section, we can sample all the parameters according to the prior distributions. The synthetic data can then be interpreted as our prior distribution on the data.

4) Fit the model to the synthetic data

The Stan program encoding this model is the following:

```
data {
  int<lower=0> N;
  int<lower=0> K;
  int<lower=0> Kb;
  int<lower=0> Kc;
  int<lower=0, upper=1> yb[N, Kb];
  vector[Kc] yc[N];
}

transformed data {
  matrix[Kc, Kc] I = diag_matrix(rep_vector(1, Kc));
}

parameters {
  vector[Kb] zb[N];
  cholesky_factor_corr[K] L_R;
  vector<lower=0>[Kc] sigma;
  vector[K] mu;
}

transformed parameters {
  matrix[N, Kb] z;
  vector[Kc] mu_c = head(mu, Kc);
  vector[Kb] mu_b = tail(mu, Kb); {
    matrix[Kc, Kc] L_inv = \
    mdivide_left_tri_low(diag_pre_multiply(sigma, \
    L_R[1:Kc, 1:Kc]), I);
     for (n in 1:N) {
       vector[Kc] resid = L_inv * (yc[n] - mu_c);
       z[n,] = transpose(mu_b + tail(L_R * \
       append_row(resid, zb[n]), Kb));
     }
  }
}

model {
  mu ~ normal(0, 10);
  L_R ~ lkj_corr_cholesky(2);
  sigma~cauchy(0, 2.5);
  yc ~ multi_normal_cholesky(mu_c, \
  diag_pre_multiply(sigma, L_R[1:Kc, 1:Kc]));
  for (n in 1:N) zb[n] ~ normal(0, 1);
  for (k in 1:Kb) yb[, k] ~ bernoulli_logit(z[, k]);
}

generated quantities {
  matrix[K, K] R = \
  multiply_lower_tri_self_transpose(L_R);
  vector[K] full_sigma = append_row(sigma, \
                                     rep_vector(1, Kb));
  matrix[K, K] Sigma = \
  multiply_lower_tri_self_transpose(\
  diag_pre_multiply(full_sigma, L_R));
}
```

Model Fit Checks

Figures 1, 2, and 3, we plot the posterior samples on top of the true values (vertical black lines). We check visually that the intervals containing 95% of samples (around their respective means) cover the true values we used to generate the synthetic data.

8. Both $Z_{i\cdot} \sim N_K(\mu,\Sigma)$ and $Z_{i\cdot} \sim N(\mu,\Sigma)$ hold, since the $\sim$ symbol means "is distributed as" and $N(\mu,\Sigma)$ is the pdf of $N_K(\mu,\Sigma)$.

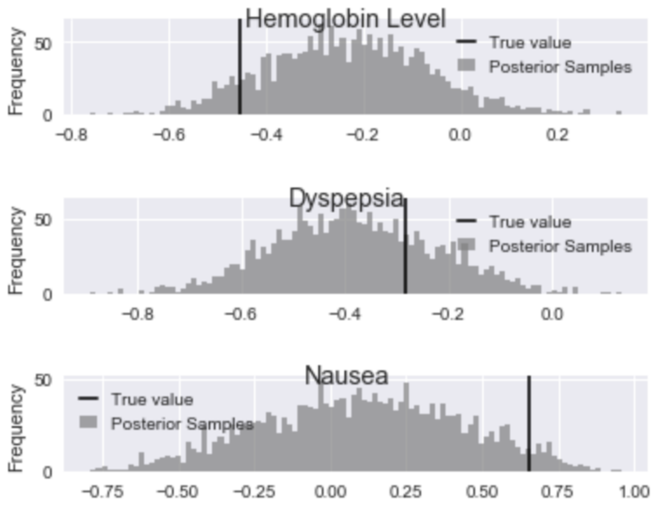

Fig. 1: *Histogram of values sampled from the posterior mean of latent variables.*

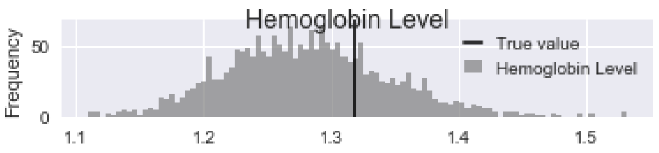

Fig. 2: *Histogram of values sampled from the posterior standard deviation for Hemoglobin Level.*

With Stan, we can also utilize the built-in checks to inspect the correctness of the inference results. One of the basic tests is the $\hat{R}$ (Rhat), which is a general summary of the convergence of the Hamiltonian Monte Carlo (HMC) chains. Another measure is the number of effective samples, denoted by n_eff. Below, we show an excerpt from Stan's summary of the fit object, displaying Rhat and n_eff, along with other metrics (mean and standard deviation), for various parameters. We shall come back to the topic of fit diagnostics in the next section.

```
Inference for Stan model:
```

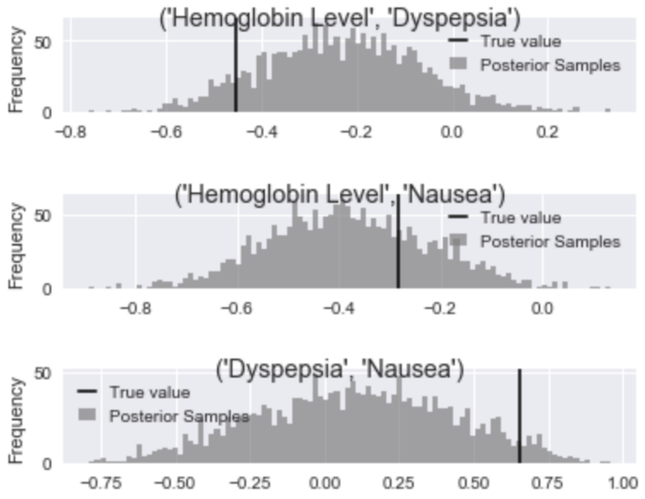

Fig. 3: *Histogram of values sampled from the posterior correlation of effects.*

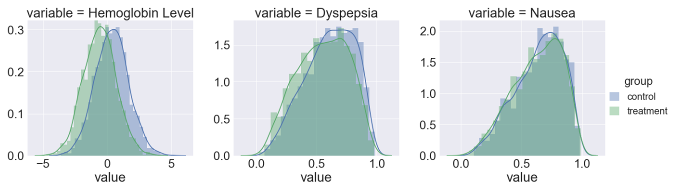

Fig. 4: *Histogram of values sampled from the posterior predictive distributions.*

```
anon_model_389cd056347577840573e8f6df0e7636.
4 chains, each with iter=1000; warmup=500; thin=1;
post-warmup draws per chain=500,
total post-warmup draws=2000.

              mean       sd   ...   n_eff   Rhat
mu[0]         0.36     0.09   ...    2000    1.0
mu[1]         0.56     0.18   ...    2000    1.0
mu[2]         0.67     0.18   ...    2000    1.0
R[0,0]         1.0      0.0   ...    2000    nan
R[1,0]       -0.24     0.16   ...    2000    1.0
R[2,0]       -0.38     0.16   ...    2000    1.0
R[0,1]       -0.24     0.16   ...    2000    1.0
R[1,1]         1.0  9.3e-17   ...    1958    nan
R[2,1]         0.1     0.32   ...     550    1.0
R[0,2]       -0.38     0.16   ...    2000    1.0
R[1,2]         0.1     0.32   ...     550    1.0
R[2,2]         1.0  7.8e-17   ...    2000    nan
sigma[0]      1.28     0.06   ...    2000    1.0
```

5. Fit the model to the real data

Once we have built confidence in our inference algorithm, we are ready to fit our model to the real data and answer the question of interest. Our goal is to use the data to predict the effect of the treatment on a new patient, i.e., the posterior predictive distribution.

In this case study, we may not share real data but, for demonstration purposes, we created two other sets of synthetic data, one representing the control group and the other the treatment group. For each posterior sample of parameters (μ_i, Σ_i), we generate a latent variable $Z_{i\cdot} \sim N(\mu_i, \Sigma_i)$. We then set $Y_{ij} = Z_{ij}$ for $j = 1$, whereas for $j = \{2, 3\}$, we sample $Y_{ij} \sim \text{Bernoulli}(\text{logit}^{-1}(Z_{ij})$. The resulting set of $Y_{i\cdot}$ is the posterior predictive distribution. We do this for the parameters learned from both groups, Y^T and Y^C separately, and plot the results in Figure 4.

Looking at the plots, we can visualize the effect of the drug on a new patient by distinguishing the effects with the treatment (green) versus without (blue). We observe that the Hemoglobin levels are likely to decrease under the treatment by about 1 unit on average. The probability of experiencing dyspepsia is slightly lower under the treatment, contrary to that of nausea which is the same in both groups. Note how the Bayesian approach results in predictive distributions rather than point estimates, by incorporating the uncertainty from the inference of the parameters.

Bayesian Inference with Stan

Stan is a powerful tool which "mitigates the challenges of programming and tuning" HMC to do statistical inference. Stan is a compiled language written in C++. It includes various useful tools and integrations which make the researcher's life easier. It can be accessed from different languages via interfaces. This case study was created with the Python interface, Pystan. Note that, at the time of writing, the most developed interfaced is the R one, called RStan. Although the underlying algorithm and speed is the same throughout the different interfaces, differences in user experience can be meaningful.

Stan requires a description of the basic ingredients of Bayesian inference (i.e., the model, likelihood, priors, and data) and returns samples from the posterior distribution of the parameters. The user specifies these ingredients in separate code blocks called *model* (lines 37–45), *parameters* (lines 14–20), and *data* (lines 1–8). Stan code is passed in via a character string or a plain-text *.stan* file, which is compiled down to C++ when the computation happens. Results are returned to the interface as objects.

Choice of priors

Stan provides many distributions to choose from, which are pre-implemented to maximize efficiency. The Stan team also provides researchers with recommendations on default priors for commonly used parameters, via the Stan manual [Tea17] and other online materials. In our case study, we chose an LKJ prior (line 39) for the correlation matrix, one of the pre-implemented distributions in Stan. The LKJ prior has certain attractive properties and is a recommended prior for correlation matrices in Stan (for reasons beyond the scope of this paper). It has only one parameter (we set it to 2) which pulls slightly the correlation terms towards 0. Another example is the half-Cauchy prior distribution for scale parameters such as standard deviation (line 40). Half-Cauchy is the recommended prior for standard deviation parameters because its support is the positive real line but it has higher dispersion than other alternatives such as the normal distribution. Note that it is easy to truncate any pre-implemented distribution. Stan accepts restrictions on parameters. For example, we restrict the parameter for standard deviation to be positive (line 18). This restriction is then respected when combined with the prior distribution defined later (line 40) to yield a constrained half-Cauchy prior.

Fit diagnostics

HMC has many parameters that need to be tuned and can have a big impact on the quality of the inference. Stan provides many automated fit diagnostics as well as options to tune manually the algorithm, if the default values do not work. For example, the Gelman–Rubin convergence statistic, $\hat{R}$, comes for free with a Stan fit; effective sample size is another good way to evaluate the fit. In most cases, $\hat{R}$ values need to be very close to 1.0 (± 0.01) for the results of the inference to be trusted, although this on its own does not guarantee a good fit. More advanced topics, such as divergent transitions, step sizes and tree depths are examined in the Stan manual, together with recommendations on how to use them.

Challenges

Stan, and HMC in general, is not perfect and can be challenged in various ways. For example multimodal posterior distribution, which are common in mixture models, are hard to explore[9].

Another common issue is that mathematically equivalent parameterizations of a model can have vastly different performance in terms of sampling efficiency[10]. Although finding the right model parameterization does not admit a simple recipe, the Stan manual [Tea17] provides recommendations to common problems. For example, we can usually improve the sampling performance for normally distributed parameters of the form $x \sim N(\mu, \sigma^2)$ if we use the non-center parameterization $x = \mu + \sigma z$ for $z \sim N(0, 1)$. In our case study, we use this trick, or rather its multivariate version, by targeting the non-centered parts of the latent variable `Z` (lines 15, 23, 31–32 and 43). Another cause of bad inference results in regression models is correlation among covariates. The way to improve the sampling efficiency of a regression model is to parameterize it using the QR decomposition[11]. We note that these issues, among others, that a researcher will encounter when using Stan stem from the difficulties of Bayesian inference, and HMC in particular [BG13], not Stan. The biggest limitation of HMC is that it only works for continuous parameters. As a result we cannot use Stan, or HMC for that matter, to do inference on discrete unknown model parameters. However, in some cases we are able to circumvent this issue[12].

Stan vs PyMC3

In this subsection, we provide a brief overview of the similarities and differences between PyStan and PyMC3, which is another state-of-the-art FLOSS[13] implementation of automatic Bayesian inference in Python. By 'automatic,' we mean that the user only needs to specify the model and the data and the software takes care of the Bayesian inference. Both PyStan and PyMC3 let users fit highly complex Bayesian models, by using HMC under the hood.

Stan and PyMC3 are the same insofar as they serve exactly the same purpose. They both are expressive languages and allow flexible model specification in code. PyMC3 leverages Theano to implement automatic differentiation, whereas Stan relies on its own algorithm. Practitioners report that PyMC3 is easier to get started with (hence, more suitable for prototyping), while Stan is more robust (hence, more suitable for production). For example, Prophet[14] is a timeseries forecasting package by Facebook implemented with Stan. Indeed, there is a rich ecosystem of packages built on top of Stan. However, most of these are available in R only. Most of RStan derived packages follow pre-existing conventions to ease the transition of researchers who want to try Bayesian modeling seamlessly. For example, R users are usually familiar with the *glm* building block for fitting generalized linear models; with the *brms* package[15] users can insert a Bayesian estimates in place of frequentist estimates with minimal changes to their scripts. This way users can easily compare the estimates of the two methods and judge whether the Bayesian approach works for them.

Such packages can also be of use to more advanced users of Bayesian inference as they typically implement the state-of-the-art modeling choices such as default priors and expose the generated Stan code to the user. Hence, interested researchers can learn by essentially using them to generate a baseline Stan code that they can tweak further according to their needs. At the time of writing, PyStan users cannot directly benefit from the Stan ecosystem of packages without leaving Python, at least briefly, as most of the packages above are not available in Python. As a result, we think that PyMC3 seems to be a more complete solution from a Python perspective. PyMC3 is native to Python and hence more integrated into Python than PyStan. PyMC3 also offers more integrated plotting capabilities than PyStan[16].

The value of Stan, in the authors' view, should be considered beyond the mere software implementation of HMC. Stan consists of a dynamic research community that aims at making Bayesian inference more accessible and robust. This is achieved through open discussion of all Bayesian topics, many of which are areas of active research. Interested users can learn more about Bayesian inference in general, not just Stan, by reading online and participating in the discussion (see next subsection).

Further reading

The Stan manual [Tea17] is a comprehensive guide to Stan but also includes guidance for Bayesian data analysis in general. For a concise discussion on the history of Bayesian inference programs and the advantages of HMC, see [McE17]. For examples of other case studies and tutorials in Stan, see http://mc-stan.org/users/documentation/. For active discussions and advice on how to use Stan, see the Stan forum at http://discourse.mc-stan.org/.

Reproducibility

In this last section, we report on our experience of making the case study more reproducible. We consider the definition of reproducibility put forward by [KTD18]. Namely, reproducibility is "the ability of a researcher to duplicate the results of a prior study using the same materials as were used by the original investigator" [RMS18]. To achieve it, we follow the guidance of the three key practices of computational reproducibility [Kit18]:

1) Organizing the project into meaningful files and folders;
2) Documenting each processing step;
3) Chaining these steps together (into a processing *pipeline*).

We care about reproducibility for both high-level and low-level reasons. In the big picture, we want to make the work more shareable, reliable, and auditable. In the day-to-day, we want to save time, catch mistakes, and ease collaboration. We are experiencing these benefits already, having taken a few steps towards computational reproducibility. Finally, let us borrow a quote which is well-known in the reproducible research communities: "Your most important collaborator is your future self."

The case study presented earlier was not originally set up according to the three practices outlined above. Notably, it used to live in a variety of files (scripts, notebooks, figures, etc.) with no particular structure. File organization is a common source of confusion and frustration in academic research projects. So, the first step we took was to create a clear, relatively standardized directory structure. We went for the following:

```
|-- mixed-data/          <- Root (top-most) directory
                            for the project.
  |-- README.md          <- General information about
                            the project.
  |-- environment.yml    <- Spec. file for reproducing
                            the computing environment.
  |-- data/
    |-- raw/             <- The original, immutable
                            data dump.
    |-- interim/         <- Intermediate outputs.
  |-- models/
```

9. See https://github.com/betanalpha/knitr_case_studies/tree/master/identifying_mixture_models.
10. See http://mc-stan.org/users/documentation/case-studies/mle-params.html.
11. See http://mc-stan.org/users/documentation/case-studies/qr_regression.html.
12. See http://elevanth.org/blog/2018/01/29/algebra-and-missingness/.
13. FLOSS stands for "Free/Libre and Open Source Software."
14. See https://research.fb.com/prophet-forecasting-at-scale/.
15. This package makes it easy to fit models (https://github.com/paul-buerkner/brms).
16. For additional sources on PyMC3 vs Stan comparisons, see:

- https://github.com/jonsedar/pymc3_vs_pystan
- http://discourse.mc-stan.org/t/jonathan-sedar-hierarchical-bayesian-modelling-with-pymc3-and-pystan/3207
- http://andrewgelman.com/2017/05/31/compare-stan-pymc3-edward-hello-world/
- https://towardsdatascience.com/stan-vs-pymc3-vs-edward-1d45c5d6da77
- https://pydata.org/london2016/schedule/presentation/30/
- https://github.com/jonsedar/pymc3_vs_pystan

```
    |-- modelcode.stan <- Model definition.
  |-- notebooks/       <- <- Jupyter notebooks.
    |-- rosi_py.ipynb
    |-- rosi_py_files/ <- Subdirectory for temporary
                          outputs such as figures.
      |-- README.md    <- Documentation for this
                          subdirectory.
```

We have found this directory structure to be very helpful and useful in the case of an exploratory data analysis project. Additionally, there is value in reusing the same structure for other projects (given a structure that works for us): By reducing unnecessary cognitive load, this practice has made our day-to-day more productive and more enjoyable. For further inspiration, we refer the interested reader to [Tra17], [Dc] and references therein.

The second step we took was to set up the project as its own Git repository[17]. Thus, we can track changes conveniently and copy ('clone') the project on other machines safely (preserving the directory structure and, hence, relative paths)[18].

Reproducible research practitioners recommend licensing your scientific work under a license which ensures attribution and facilitates sharing [Sto09]. Raw data are not copyrightable, so it makes no sense to license them. Code should be made available under a FLOSS license. Licenses suitable for materials which are neither software nor data (i.e., papers, reports, figures), and offering both attribution and ease of sharing, are the Creative Commons Attribution (CC BY) licenses. The case study (notebook) has been licensed under CC BY since the beginning. This practice can indeed contribute to improving reproducibility, since other researchers may then reuse the materials independently, without having to ask the copyright holders for permission.

We were confronted with the issue of software portability in real life, as soon as we (the authors) started collaborating. We created an isolated Python 3 environment with *conda*, a cross-platform package and environment manager[19]. As it turned out, the conventional file `environment.yml`, which specifies package dependencies, did not suffice: We run different operating systems and some dependencies were not available for the other platform. Therefore, we included a `spec-file.txt` as a specification file for creating the *conda* environment on GNU/Linux. Admittedly, this feels only mildly satisfying and we would welcome feedback from the community.

At the moment, all the analysis takes place in one long Jupyter notebook[20]. We could break it down into smaller notebooks (and name them with number prefixes, for ordering). This way, someone new to the project could identify the various modelling and computing steps, in order, only by looking at the 'self-documenting' file structure. If we ever take the project to a production-like stage, we could further modularize the functionality of each notebook into modules (*.py* files), which would contain functions and would be organized into a project-specific Python package. This would pave the way for creating a build file[21] which would chain all operations together and generate results for our specific project. Reaching this stage is referred to as *automation*.

17. Git is a distributed version control system which is extremely popular in software development (https://git-scm.com/).

18. The *mixed-data* project is hosted remotely at https://github.com/bayesways/mixed-data.

19. See https://conda.io/docs/.

20. See https://github.com/bayesways/mixed-data/blob/d2fc4ea72466a4884dc2a5c46510129fac602f1f/notebooks/rosi_py.ipynb.

21. See https://swcarpentry.github.io/make-novice/reference#build-file.

In data analysis, the first of these operations usually consists in accessing the initial, raw dataset(s). This brings about the question of data availability. In human subject research, such as clinical trials, the raw data cannot, and should not, be made publicly available. We ackowledge the tension existing between reproducibility and privacy[22]. At the time of this writing and as mentioned in the case study section, we are showcasing the analysis only with synthetic input data.

References

[Bar18] Pablo Barberá. *The Trade-Off Between Reproducibility and Privacy in the Use of Social Media Data to Study Political Behavior*. University of California Press, Oakland, CA, 2018. URL: https://www.practicereproducibleresearch.org/case-studies/barbera.html.

[BG13] Michael Betancourt and Mark Girolami. Hamiltonian monte carlo for hierarchical models. 2013. arXiv:1312.0906v1.

[CG98] Siddhartha Chib and Edward Greenberg. Analysis of multivariate probit models. *Biometrika*, 85(2):347–361, jun 1998. doi:10.1093/biomet/85.2.347.

[Dc] DrivenData and contributors. The cookiecutter data science project. Accessed on Wed, May 23, 2018. URL: http://drivendata.github.io/cookiecutter-data-science/.

[GCS+13] Andrew Gelman, John B. Carlin, Hal S. Stern, David B. Dunson, Aki Vehtari, and Donald B. Rubin. *Bayesian Data Analysis*. CRC press, 2013.

[Kit18] Justin Kitzes. *The Basic Reproducible Workflow Template*, chapter 3. University of California Press, Oakland, CA, 2018. URL: https://www.practicereproducibleresearch.org/core-chapters/3-basic.html.

[KTD18] J. Kitzes, D. Turek, and F. Deniz, editors. *The Practice of Reproducible Research: Case Studies and Lessons from the Data-Intensive Sciences*. University of California Press, Oakland, CA, 2018. URL: https://www.practicereproducibleresearch.org/.

[McE15] Richard McElreath. *Statistical rethinking : a Bayesian course with examples in R and Stan*. 2015.

[McE17] Richard McElreath. Markov chains: Why walk when you can flow?, 2017. Accessed on Wed, May 23, 2018. URL: http://elevanth.org/blog/2017/11/28/build-a-better-markov-chain/.

[MR06] Jean-Michel Marin and Christian P. Robert. *The bayesian core: a practical approach for computational bayesian statistics*, volume 102. Springer Texts in Statistics, 2006. doi:10.1016/j.peva.2007.06.006.

[RMS18] Ariel Rokem, Ben Marwick, and Valentina Staneva. *Assessing Reproducibility*, chapter 2. University of California Press, Oakland, CA, 2018. URL: https://www.practicereproducibleresearch.org/core-chapters/2-assessment.html.

[Rob07] Christian P. Robert. *The Bayesian choice: from decision-theoretic foundations to computational implementation*. Springer Science & Business Media, 2007.

[Sto09] Victoria Stodden. Enabling reproducible research: Open licensing for scientific innovation. *International Journal of Communications Law and Policy*, 2009. doi:10.7916/D8N01H1Z.

[TDM12] Aline Talhouk, Arnaud Doucet, and Kevin Murphy. Efficient bayesian inference for multivariate probit models with sparse inverse correlation matrices. *Journal of Computational and Graphical Statistics*, 21(3):739–757, jul 2012. doi:10.1080/10618600.2012.679239.

[Tea17] Stan Development Team. Stan modeling language: User's guide and reference manual, 2017. URL: https://github.com/stan-dev/stan/releases/download/v2.17.0/stan-reference-2.17.0.pdf.

[Tra17] Dustin Tran. A research to engineering workflow, 2017. Accessed on Wed, May 23, 2018. URL: http://dustintran.com/blog/a-research-to-engineering-workflow.

[Woo15] Simon N. Wood. *Core Statistics*. Cambridge University Press, 2015. URL: https://people.maths.bris.ac.uk/~sw15190/core-statistics.pdf.

22. A case study in political science is discussed in this respect in [Bar18]. Some private communication with political scientists and various technologists have led us to throw the idea of leveraging the blockchain to improve reproducibility in human subject research: What if the raw datasets could live as private data on a public blockchain, notably removing the possibility of cherry-picking *by design*?