

Applying Bayesian Hierarchical Models to N-of-1 Trials

Tuomo Kareoja

August 28, 2020

Contents

Introduction	3
1 What Are N-of-1 Trials?	4
2 Statistical Modeling of N-of-1 Trials	8
2.1 Basic Models	9
2.2 Incorporating time-trends into the model	9
2.3 The Problem of Autocorrelation	10
2.4 Non-continuous Measurements	12
3 Bayesian Estimation	15
3.1 Principles of Bayesian Inference	15
3.1.1 Challenges of Bayesian Inference	19
3.2 Why Bayesian Inference Fits N-of-1 Studies	22
3.2.1 Flexibility of Experimental Design	23
3.2.2 Richness and Communicability of the Estimates	24
4 Combining Information From Several N-of-1 Trials With Hierarchical Models	26
5 Example of a Hierarchical Bayesian Analysis Using Simulated Data	30
5.1 Defining the Experiment	30
5.2 Simulating the Data	31
5.3 Analyzing a Single Trial	33
5.3.1 Defining the Model	33
5.3.2 Implementing the Model in PyMC3	35
5.3.3 Final Results	45
5.4 Analyzing Multiple Trials With Hierarchical Models	45
5.4.1 Defining the Model	45
5.4.2 Implementing the Model in PyMC3	46

5.4.3	Final Results	53
6	Conclusion	56

Introduction

In clinical practice, N-of-1 trials are multiple crossover trials conducted on a single patient, where treatment periods are formed into multiple blocks each containing at least one period of each treatment under consideration [1]. By comparing the measurements taken during different treatment regimes over multiple blocks, the most suitable treatment option can be chosen for the particular patient studied.

In the following pages, we will walk through a concise explanation of the experimental design of N-of-1 trials and how we can statistically model them, given the kinds of challenges their design poses. We then follow up with how to apply Bayesian inference with these models and estimate the effectiveness of different treatments, highlighting how Bayesian methods can give needed flexibility when running N-of-1 trials by not relying on hypothesis testing tied to a prespecified study design. We then consider the case when there are multiple similar N-of-1 trials and show how it is possible to pool the information from these with hierarchical Bayesian methods, without losing sight of the goal of N-of-1 trials: to find the best treatment for each individual patient. Finally, we end by with a complete example of analyzing multiple N-of-1 trials with hierarchical Bayesian methods with Python and PyMC3-package using simulated data.

Chapter 1

What Are N-of-1 Trials?

In the assessment of any medical treatment the “gold standard” is a randomized controlled trial (RCT), where subjects are randomized to two or more groups that are given different treatments or no treatment at all. The measurements from these groups are then compared and a result derived about which treatment is most effective on average. This design takes into account unknown factors that might make some patients more susceptible to certain treatments by forming the groups randomly and thus, on average, distributing these patients evenly between groups. By comparing the groups against each other and not just to the same patients at the beginning and end of the study, it also takes into account time-related effects like the natural progression of a disease. Despite its unquestionable value in finding general effects, this method can run into problems when we try to apply its results to individual patients in clinical practice.

Knowing the best treatment on average might not help much in finding the right treatment for a particular patient if the variability in treatment effectiveness between patients is high. Although we can use covariates like age, gender, or a certain gene variant to explain the variance between patients in RCT:s, there can still be lots of unexplained between-patient variance left. Some of this variance is of course caused by random factors like measurement error, but a significant part might be caused by real differences between the patients [9, 10, 11, 12]. In other words, there might be individual factors that explain the variability of the efficacy of different treatments between patients that might either be too specific to be considered in an RCT study or unknown and thus impossible to analyze in this kind of experimental design.

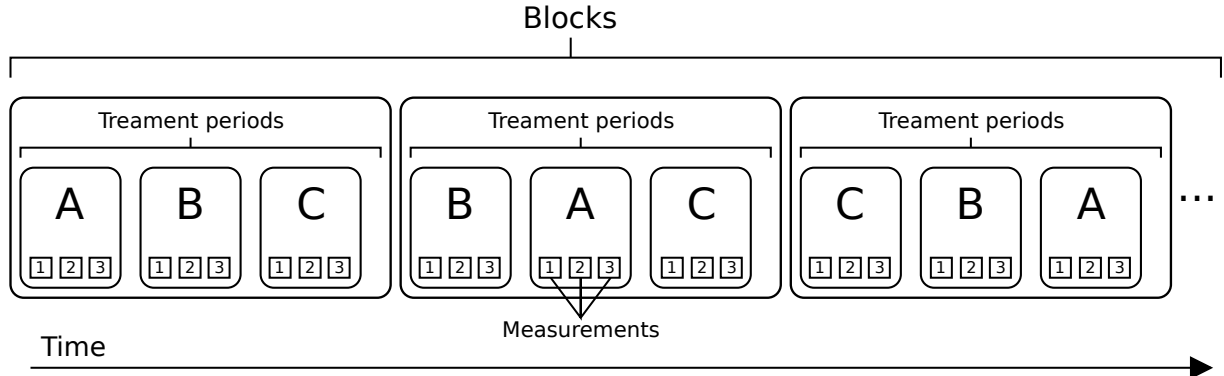
Another problem with RCTs is the peculiarity of their participants. It is a common practice to accept patients to RCTs only if they don’t suffer from any medical issue besides the one that is being studied. This lack of comorbidity makes it easier to get clear results by removing confounding factors, but at the same time this lowers the external validity of the results, because in the real world patients often suffer from multiple medical

issues simultaneously. Because of this, it might be possible that even when the scientific literature provides clear results about the relative effectiveness of different treatments, these results don't generalize that well to the kinds of patients the clinical practitioner sees daily in her office. [9]

N-of-1 trials can be used to patch the holes in the knowledge that RCTs cannot fill by changing the focus of the study from group averages to the individual patients. In N-of-1 studies, instead of comparing groups with different treatments, the comparisons are made across time with a single patient using different treatments. In this design multiple treatments are tried sequentially in periods that are formed into blocks each of which contains a treatment period of each treatment option under consideration at least once. Measurements are taken throughout the study and, depending on the ease of measuring the outcome of interest, these could be taken just once per treatment period, or even multiple times per day. Each block includes a period of each treatment under consideration in random or balanced order to take into account time-related effects. A simple example would be an ABBA design that includes two blocks within which the two treatments A and B are assigned in a balanced order.

Depending on the treatments considered, there might also be a so-called "washout" period between treatments, where the patient does not receive any treatment, and her state is allowed to return to baseline. This method is used to prevent treatment interactions that could make it difficult to analyze the results or could be dangerous to the patient. If the treatments being studied allow it, N-of-1 trial can also use the double-blind method, where both the patient and the person administering the treatment don't know which treatment is currently used, and placebo treatment, where one of the treatments only resembles treatment, but does not contain any active ingredient (pharmacological or otherwise).

Below is a schema of a more complex N-of-1 trial with three treatments A, B and C, random assignment of treatments within blocks, three measurements within each treatment period and washout periods between treatments:



The stated aim of N-of-1 trials is quite different from RCT:s: where the latter tries to generalize results to population and find which treatment is best in general, the N-of-1 trial tries to only generalize to the patient in question. This means that whereas comorbidity and other factors that cause systematic variation between the patients in treatment outcomes are a problem in RCT:s, these are not an issue in N-of-1 trials because there is no need to generalize beyond the one patient. There is also often no actual need to know what is causing a certain treatment to work better or worse, as long as it is not because of measurement errors or time-related effects.

Use of N-of-1 trials is appropriate in situations where there are multiple treatment options but there is no prior knowledge of which of these would be best, when there is known to be considerable variability between patients in treatment efficacy, or when there is reason to doubt that the results from scientific literature generalize to the patient in question [1]. This would seem to make n-of-1 trials applicable to many situations, but there are also multiple factors restricting their use [1].

Firstly, N-of-1 trials can only be used to study illnesses that are chronic, progress slowly, and are at least somewhat stable. Also the treatment options available must have noticeable treatment responses within a short timeframe. Running the trials needs time to complete and fast-changing illness or slowly effecting treatment make it either impossible to distinguish true effects from the natural progression of the disease or make the length of the trials impractically long. N-of-1 trials are also unsuitable for testing of preventative treatments because the effects of these treatments are often impossible to assess without comparisons to other patients who are not receiving the treatment.

Secondly, running a N-of-1 trial is costly because of added expenses of training the medical staff in the method, running the trial with all its measurements and analyzing the data. This means that it can be hard to find cases where using this method is cost-effective and studies trying to make these kind calculations have given mixed results

[2].

These limitations have kept the use of N-of-1 trials rare in clinical use, even though they could potentially both increase the life quality of the patients and lower the healthcare costs by finding the most suitable medications to patients that might end up potentially using them for years. This state of affairs might be changing fast though. Aging and environmental factors are changing the worlds' disease burden so that a growing proportion of it is constituted by chronic diseases [5], of which common ones like non-acute cardiovascular diseases and diabetes are excellent candidates for N-of-1 trials. Also the cost of administering N-of-1 trials are dropping with the advent of cheap and reliable health sensors like smartwatches and connected blood pressure monitors. For example in diabetic patients it is now possible to get real-time readings of blood insulin levels with minimal effort from the patient [6]. These factors mean that the popularity of this method could potentially rise significantly in the future.

Chapter 2

Statistical Modeling of N-of-1 Trials

Even though the data created by the N-of-1 trials resembles traditional time series data with autocorrelation between observations and repeated measurements from the same study unit, there are additional complexities from the structure of repeating treatment periods. Trying to take into account all the peculiarities of the study design could end up with model too complicated to the small amount of data generated by a single study, so one must consider carefully what factors actually *need* to be incorporated into the model.

Simplest model that we could employ is to just count the number of blocks where a treatment is considered “better” than others. The precise definition of “better” doesn’t matter here. This way we arrive to a simple binomial model where the number of “successes” X is the number of blocks where a treatment is considered the “best” follows binomial distribution and the probability of each treatment option of having k successes is given by:

$$(2.1) \quad P(X = k) = \binom{n}{k} p^k (1 - p)^{(n-k)},$$

where k = number of blocks, where the treatments is considered the “best”, n = total number of blocks and p = the probability of being considered the “best”.

This type of model is rudimentary at best because it fails to consider the magnitude of the differences between treatment effects and does not take into account the actual number of measurements within each treatment period. To take these factors into account, more complex models are in order.

2.1 Basic Models

Before going further we make the assumption that the measurements are continuous as this is probably the most common use case. Lets first look at a model where we assume that there are no time-trends and no autocorrelation between measurements. Let y_{mbpt} represent the outcome measured while on treatment m within treatment block b within treatment period p at time t . The treatment periods are indexed within each block and time is indexed within each treatment period:

$$(2.2) \quad y_{mbpt} = \mu_m + \gamma_b + \delta_{p(b)} + \epsilon_{t(p(b))},$$

where $\gamma_b \sim N(0, \sigma_\gamma^2)$, $\delta_{p(b)} \sim N(0, \sigma_\delta^2)$, and $\epsilon_{t(p(b))} \sim N(0, \sigma_\epsilon^2)$

This model assumes all treatment effects μ_m to be constant. Within the normally distributed terms γ_b represents random block effects, $\delta_{p(b)}$ random period effects and $\epsilon_{t(p(b))}$ random within period errors. We could choose one of the blocks as a reference and set $\gamma_1 = 0$ and assume that within each block the between period effects follow the same pattern, e.g. the difference between treatment period one and two is the same within each block.

The random effects between blocks and treatment periods could represent for example the random variations in the motivation of the patient and possible changes in treating personnel within each block and treatment period. The random within period errors represents the measurement error of single measurements within treatment periods. The relative size of these terms is important for effective design of the trial, because they determine if it is more be beneficial for the statistical power of the study to add more measurements, treatment periods or blocks.

If the measurements within blocks and within periods do not correlate, the model 2.2 can be simplified by dropping γ_b and $\delta_{p(b)}$:

$$(2.3) \quad y_{mbpt} = \mu_m + \epsilon_{t(p(b))},$$

where $\epsilon_{t(p(b))} \sim N(0, \sigma_\epsilon^2)$.

This simple model is a natural fit in scenarios where there is just one measurement within each treatment period.

2.2 Incorporating time-trends into the model

As the symptoms of the patient might not be completely stable (e.g. because symptoms get worse with the progression of the disease) fitting some kind of time-trend to the model

is usually advisable. We can modify the model 2.3 from previous chapter to include a linear time-trend by adding an intercept and slope of the time-trend. In this case the model can be expressed more concisely just in terms of the measurement y_t taken at time t , where time is indexed from the start of the study:

$$(2.4) \quad y_t = \beta_0 + \beta_1 t + \mu_t + \epsilon_t,$$

where $\epsilon_t \sim N(0, \sigma^2)$.

Here β_0 is the intercept, β_1 the slope of the time trend, μ_t the effect of the treatment given during time t and ϵ_t is the residual error at time t . More complex time-trends can be introduced by modifying the slope, for example by adding the term $\beta_2 t^2$ to introduce a quadratic trend.

Another effect dependent on time to take into consideration are period effects. There might be some part of the trial that falls within a period that we presume to have its own effect. An example of this kind of effect is if we study asthma medications and part of the trial falls within the pollen season. A simple way to model this is to use a dummy variable that takes the value 1 within the period and 0 outside it. Extending the model 2.4 with a period of constant effect β_2 we end up with:

$$(2.5) \quad y_t = \beta_0 + \beta_1 t + \beta_2 Z_t + \mu_t + \epsilon_t,$$

where $\epsilon_t \sim N(0, \sigma^2)$ and dummy variable $Z_t = 1$ when $t \in (t_{\text{period start}} \dots t_{\text{period end}})$ and 0 otherwise.

Lastly, to take into account that treatment effects themselves can vary with time, for example because treatment works better during periods of greater disease severity, we can add a time-by-treatment interaction effect into the model. For example in the case where we expect that the illness gets steadily (linearly) worse with time, but the treatments compensate this by being similarly more effective, we can extend the simple linear time-trend model 2.4 by adding a second time related treatment term $\gamma_t t$:

$$(2.6) \quad y_t = \beta_0 + \beta_1 t + \mu_t + \gamma_t t + \epsilon_t,$$

where $\epsilon_t \sim N(0, \sigma^2)$

2.3 The Problem of Autocorrelation

A common occurrence in time-series data is the autocorrelation between measurements so that there is some similarity between observations defined by a function of the time

lag between them. Often a good first response to this problem is to add a time-trend to the model like we did previously. This often removes a substantial proportion of the autocorrelation that could be caused for example by the natural progression of the disease or seasonal variations in its symptoms [3]. Unfortunately, in N-of-1 trials the carryover effects from previous treatments and slow onset of treatment effects can lead to very complex autocorrelation patterns that are hard to remove with just a simple time-trend.

Carryover effects refer to the lingering effects of the treatment even after it has been stopped. This can make the treatment effects in the next treatment period with a different treatment seem larger than they actually are or smaller in the unfortunate and hopefully rare case where the previous treatment was actually harmful. Carryover effects also encompass the effects of interactions between sequential treatments, which could even be dangerous depending on the nature of the treatments. On the other hand treatment effects that manifest slowly can often give the opposite effect of carryover effects by making the treatments look less effective than they really are during the first measurements of each treatment period.[3]

To deal with these more devious sources of autocorrelation in our model we can take two routes: moving average models and autoregressive models. In moving average models the error is represented as depending on the previous error(s), by expressing the measurement at time t as function of one or more previous errors:

$$(2.7) \quad y_t = \mu_t + \epsilon_t + \rho\epsilon_{(t-1)},$$

where μ_t is the effect of the treatment given at time t , ρ is the correlation between consecutive errors and all errors $\epsilon \sim N(0, \sigma^2)$. Instead of making the measurement dependent on just the previous error the model can be adjusted to include more a complex lag by adding more error terms with bigger lag and including more ρ terms to denote the correlations between them ($\rho_1\epsilon_{t-1} + \dots + \rho_x\epsilon_{t-x}$).

In autoregressive models approach we express the autocorrelation in the measurements themselves so that the measurement at time t is a function of the measurement at $t - 1$:

$$(2.8) \quad y_t = \rho y_{t-1} + \mu_t + \epsilon_t,$$

where μ_t is the effect of the treatment given at time t , ρ is the correlation between consecutive measurements and $\epsilon_t \sim N(0, \sigma^2)$ is the error term. Like with moving average model we could also include more than just one lag by adding more lagged terms and correlation terms.

Although it can make more intuitive sense to make the whole observation dependent on the previous observation, it is important to recognize that in this case the treatment

effects μ_t must be interpreted differently as they are in this case conditioned on previous measurements.

Moving average models and autoregressive models are often combined by using both methods of taking the autocorrelation into account simultaneously (ARMA-models), but as amount of data produced in N-of-1 studies is often so small, taking this approach might not be advisable as it might make the model unnecessarily complex.

Although we can try to solve problems created by the carryover effect and the slow manifestation of treatment effect with modeling, a better way could be to take measures to mitigate the effects in the study design itself. By having a long enough washout period between different treatments, we can minimize the carryover effect. If there are no harmful interactions between the treatment the next treatment can be started within the washout period so that we minimize the problem of slow treatment effects. If there are interactions to be taken into consideration, the first few measurements at the beginning of each treatment period could also just be dropped. By doing this we are of course throwing away data, but we must remember that if the measurements at the beginning of the treatment period are mangled, this will also mangle our parameter estimates. Trying to take these effects into account in our model will probably not eliminate these effects completely and will increase the complexity of our model.

2.4 Non-continuous Measurements

Up to this point we have assumed that the measurement used are continuous, but we could of course have measurements that are binary, counts of events or categorical. With these kinds of measurement the models need to be reformatted so that they don't presume normal distributions. Despite this the models still don't have to differ much from the models presented above and the principals covered before can be applied.

To modify previously presented models to work when measurements are not continuous, we need to formulate them as generalized linear models. We do this by keeping the right-hand sides in the same form but expressing the left-hand side in terms of a link function of the mean of the probability distribution of the outcomes. So instead of expressing the model in terms of individual observations y_t , we express it as the expected value of these measurements conditional on the data $E(Y|D)$ that we feed to the link function. The link function allows us to model measurements with arbitrary distributions, as now the link function can linearly depend on the parameters of the model, rather than needing the measurements themselves to do so. We need to do this to prevent the models from giving impossible predictions, e.g. negative counts or probabilities.

Let's work with an example of a binary outcome measurements. In this case the measurements y at time t follow the Bernoulli distribution $y_t \sim \text{Bernoulli}(p)$ where

the the expected value of the distribution is the probability p of observing the event in measurement y_t . In this case the suitable link function is the logit function $\text{logit}(p) = \log_e(\frac{p}{1-p})$. With this information we can now formulate a simple model with a linear time trend:

$$(2.9) \quad \log_e\left(\frac{p_t}{1-p_t}\right) = \beta_0 + \beta_1 t + \mu_t,$$

where p_t is the the probability of observing the event at time t and $\log_e(\frac{p_t}{1-p_t})$ are the log-odds of this event, β_0 is the intercept, β_1 the slope of the time trend and μ_t the effect of the treatment given during time t .

By first exponentiating and then using simple algebraic manipulation we can express the model in terms of the probability p_t :

$$(2.10) \quad \frac{p_t}{1-p_t} = e^{\beta_0 + \beta_1 t + \mu_t}$$

$$(2.11) \quad p_t = \frac{e^{\beta_0 + \beta_1 t + \mu_t}}{e^{\beta_0 + \beta_1 t + \mu_t} + 1} = \frac{1}{1 + e^{-(\beta_0 + \beta_1 t + \mu_t)}}$$

We notice that there are no error terms in this model. This is because we are not modeling individual observations, but the expected value of these values (probability of observing the event under treatment used at time t in this case). Even though there is random variations in the individual observations, when we talk about their expected value, there is just a single value with no random errors. Apart from this difference we can see that we find the same model from the denominator that we used when modeling a continuous measurement with a linear time-trend in model 2.4, but without the aforementioned error term. This means that to build the models described before, we would just insert the right side of the equations into the denominator, apart from the error term.

With numbers of events as measurements, instead of expressing the model in terms of time, we can express it in terms of periods p that are of equal length and indexed from the beginning of the experiment. The number of events during period p follows a Poisson distribution $y_p \sim \text{Poisson}(\lambda)$ and the expected value of the distribution is the rate λ_p of the events during period p . For link function we use the natural log that is we have $\log_e(\lambda_p)$ on the left side of the equation. Once again using the same simple model with linear time trend we end up with a model:

$$(2.12) \quad \log_e(\lambda_p) = \beta_0 + \beta_1 r + \mu_p,$$

where λ_p is the the rate of events between measurements during period p , β_0 is the intercept, β_1 the slope of the time trend and μ_p the effect of the treatment given during period p . By simply exponentiating both sides we can express the model in terms of rate of events:

$$(2.13) \quad \lambda_t = e^{\beta_0 + \beta_1 t + \mu_t},$$

where we now have the familiar linear model formula in the exponent on the right side of the equation.

The case of categorical measurements is more complex as there are multiple possible link functions depending on which way we want to model the measurements, so we don't go through all of them here. The one that is probably the most relevant is the case when categorical measurements are ordinal, that is they have a natural ordering (like in a Likert-scale). In this case we can use the cumulative logit as the link function. So if we assume that our ordinal measurement has J categorical choices ordered from 1 to J , we can model the cumulative probability of getting a response y at time t that is at least as "severe", by putting this to the logit function and using the same basic model as before:

$$(2.14) \quad \log_e \left(\frac{P(y_i \leq j)}{1 - P(y_i \leq j)} \right) = \beta_0 + \beta_1 t + \mu_t,$$

where β_0 is the intercept, β_1 the slope of the time trend and μ_t the effect of the treatment given during time t . By exponentiating and algebraic manipulation we end up with similar model as with binary outcomes where we once again find the familiar linear model formula in the exponent:

$$(2.15) \quad \frac{P(y_i \leq j)}{1 - P(y_i \leq j)} = e^{\beta_0 + \beta_1 t + \mu_t}$$

$$(2.16) \quad P(y_i \leq j) = \frac{e^{\beta_0 + \beta_1 t + \mu_t}}{e^{\beta_0 + \beta_1 t + \mu_t} + 1} = \frac{1}{1 + e^{-(\beta_0 + \beta_1 t + \mu_t)}}$$

Chapter 3

Bayesian Estimation

Now that we have some models defined, we need to move into the next part of the analysis and give estimates to the parameters in these models. There are two broad ways to approach this task by using two different definitions of probability. The first is frequentist inference, where we consider the “true” parameters of our model fixed, but unknown, and randomness only applies to the process of creating our data. The second way is Bayesian inference, where we consider the data to be fixed and instead of thinking about the parameters as fixed parts of nature, we conceptualize them as probability distributions that express our internal uncertainty about their true values.

Although both of these inference methods work for all the models covered before, there are several factors that favor the use of Bayesian inference in N-of-1 studies that are related to their design and use context. We will return to these later when we have first gone through the principles of Bayesian inference.

3.1 Principles of Bayesian Inference

Bayesian inference is based on the Bayes’ Theorem that states the probability of an event conditional on another event:

$$(3.1) \quad P(A|B) = \frac{P(B|A)P(A)}{P(B)},$$

where A and B are events and $P(B) \neq 0$. $P(A|B)$ is the conditional probability of event A happening given that event B happened and is called the posterior. $P(A)$ is our initial probability estimate for the event A called the prior. The quotient $\frac{P(B|A)}{P(B)}$ represent how much information event B gives about event A happening. If this number is greater

than 1, then event B happening makes event A more likely and if it less than 1 is less likely. If the quotient is 1, event B gives no information about the probability of event A . Breaking the quotient down further $P(B|A)$, called the likelihood, is the reverse of the posterior and tells us how believable it is to see the event B given that event A happened. Finally $P(B)$ in the denominator is called marginal likelihood and tells us the probability of observing event B with or without event A .

Instead of events, in our case we want to formulate the theorem with parameters Θ and the data D , so that we get can estimate the posterior probability (or likelihood in case of continuous parameter values) of our parameters having certain values:

$$(3.2) \quad P(\Theta|D) = \frac{P(D|\Theta)P(\Theta)}{P(D)},$$

where $P(D) \neq 0$. We could make it more explicit what the marginal likelihood $P(D)$ stands for by writing it as $\sum_{\Theta} P(D|\Theta)P(\Theta)$ in the case when the parameter value takes discrete values and $\int_{\Theta} P(D|\Theta)P(\Theta)d\Theta$, if they are continuous. That is, the possibility of observing the data with all different combinations of the possible values of the parameters, taking into account our prior belief in the probability of these value combinations.

To tie this formula into the models we introduced previously we can demonstrate how this applies to a simple model with only the treatment effects μ of the given treatment and a random error ϵ at time t :

$$(3.3) \quad y_t = \mu_t + \epsilon_t,$$

where $\epsilon_t \sim N(0, \sigma^2)$. Time t is indexed from the beginning of the study. In this case when we want to estimate the parameters of the effects of the treatments μ_t . For simplicity lets assume that only one treatment was given so μ_t is constant. Let's refer to this treatment effect simply as μ . According to our model single observations follow a normal distribution defined by constant μ and the error term ϵ_t . As our model assumes that the observations y_t are independent (no autocorrelation included in the model) individual observations are defined as $y_t \sim N(\mu, \sigma^2)$ and the likelihood term in the bayesian formula is a simple product of the probability density functions of a normal distribution with mean μ and variance σ^2 that the observations are independent:

$$(3.4) \quad P(Y|\mu, \sigma) = \prod_Y \frac{1}{\sqrt{2\pi\sigma^2}} \exp \left\{ -\frac{(y_t - \mu)^2}{2\sigma^2} \right\},$$

where Y represents all the observations.

To avoid the complexity of dealing with multiple unknown variables we assume that σ is known. We will later come back to the case of multiple unknown variables. Now we have to define only one prior distribution as there is just one unknown parameter. We choose a normal distribution for the prior of μ that is defined by mean λ_0 and variance δ_0^2 , whose values we define ourself based on our previous knowledge and beliefs:

$$(3.5) \quad P(\mu) = \frac{1}{\sqrt{2\pi\delta_0^2}} \exp \left\{ \frac{-(\mu - \lambda_0)^2}{2\delta_0^2} \right\}$$

We could of course choose whatever distribution best matches our previous knowledge and uncertainty about the parameter, but using a normal distribution for prior when the likelihood also follows a normal distribution, we get some nice algebraic properties that we will soon see.

Now the only missing term from our bayesian equation is the marginal likelihood $P(Y)$. When we express the term more precisely as the probability of getting the data we observed over possible values of our unknown parameter $\int P(Y|\mu)P(\mu)d\mu$, we can see that as we integrate over all the possible values of μ , the term is not dependent on the specific values of μ . This means that the marginal likelihood is just a constant and as such does not affect the shape of the posterior distribution, So the shape of posterior distribution is defined by just by the likelihood and the prior $P(\mu|Y) \propto P(Y|\mu, \sigma^2)P(\mu)$. With some arithmetic manipulation we can see that the posterior probability is actually a normal distribution with mean $\lambda_1 = \delta_1^2 (\lambda_0\delta_0^{-2} + \sum_Y y\sigma^{-2})$ and standard deviation $\delta_1^2 = \frac{1}{\delta_0^{-2} + n\sigma^{-2}}$:

(3.6)

$$P(\mu|Y) \propto P(Y|\mu, \sigma)P(\mu)$$

$$\begin{aligned} &= \prod_{t=1}^n \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{(y_t - \mu)^2}{2\sigma^2}\right\} \frac{1}{\sqrt{2\pi\delta_0^2}} \exp\left\{-\frac{(\mu - \lambda_0)^2}{2\delta_0^2}\right\} \\ &= \frac{1}{(2\pi)^{\frac{n+1}{2}} \sqrt{\delta_0^2 \sigma^{2n}}} \exp\left\{-\frac{\mu^2 + 2\mu\lambda_0 - \lambda_0^2}{2\delta_0^2} - \sum_{t=1}^n \frac{y_t^2 - 2\mu y_t + \mu^2}{2\sigma^2}\right\} \end{aligned}$$

Dropping the constant terms

$$\begin{aligned} &\propto \exp\left\{-\frac{\mu^2(\sigma^2 + t\delta_0^2) + 2\mu(\lambda_0\sigma^2 + \delta_0^2 y_1 + \dots + \delta_0^2 y_n) - (\lambda_0^2\sigma^2 + \delta_0^2 y_1^2 + \dots + \delta_0^2 y_n^2)}{2\delta_0^2\sigma^2}\right\} \\ &\propto \exp\left\{-\frac{\mu^2 + 2\mu\frac{\lambda_0\sigma^2 + \sum_{t=1}^n \delta_0^2 y_t}{\sigma^2 + n\delta_0^2} - \left(\frac{\lambda_0\sigma^2 + \sum_{t=1}^n \delta_0^2 y_t}{\sigma^2 + n\delta_0^2}\right)^2}{2\frac{\delta_0^2\sigma^2}{\sigma^2 + n\delta_0^2}}\right\} \times \exp\left\{-\frac{\lambda_0\sigma^2 + \sum_{t=1}^n \delta_0^2 y_t}{2\delta_0^2\sigma^2}\right\} \\ &\propto \exp\left\{-\frac{\left(\mu - \frac{\lambda_0\sigma^2 + \sum_{t=1}^n \delta_0^2 y_t}{\sigma^2 + n\delta_0^2}\right)^2}{2\frac{\delta_0^2\sigma^2}{\sigma^2 + n\delta_0^2}}\right\} \\ &= \exp\left\{-\frac{\left(\mu - \frac{\lambda_0\delta_0^{-2} + \sum_{t=1}^n y_t\sigma^2}{\delta_0^{-2} + n\sigma_0^{-2}}\right)^2}{2\frac{1}{\sigma^{-2} + n\delta_0^{-2}}}\right\} \end{aligned}$$

Recognizing that we have the "right" part of normal distribution and adding the constant part back in

$$\begin{aligned} &\propto \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{(\mu - \lambda_1)^2}{2\delta_1^2}\right\}, \text{ where} \\ \delta_1^2 &= \frac{1}{\delta_0^{-2} + n\delta_0^{-2}} \quad \text{and} \quad \lambda_1 = \frac{\lambda_0\delta_0^{-2} + \sum_{t=1}^n y_t\sigma^2}{\delta_0^{-2} + n\sigma_0^{-2}} = \delta_1^2 \left(\lambda_0\delta_0^{-2} + \sum_{t=1}^n y_t\sigma^{-2}\right) \end{aligned}$$

So we ended up with a posterior distribution that has the shape of a normal distribution. The trick where we ignored the marginal likelihood can be used anywhere, but the fact that we could arithmetically solve that the posterior follows a well-defined distribution is not a general occurrence. Usually we need to go beyond arithmetic methods and solve the problem algorithmically.

3.1.1 Challenges of Bayesian Inference

Although the Bayes formula is conceptually simple, but when actually utilizing it we run into two big difficulties. First is the common case when the probability distributions of the prior and/or the likelihood or their product are not well-defined distributions with known properties. This can make the integral in the marginal likelihood impossible to solve analytically like we did above. Even with discrete distributions similar problems emerge if the number of possible parameter value combinations is very large (this can happen easily when there are multiple parameters) as this makes just calculating the posterior probability for each individual value unfeasible. The second problem is how we should define our priors. If we have prior knowledge, we need to be able to formulate our beliefs in a precise mathematic form and when we do not have prior knowledge we still need to define a prior, but in this case it has to be defined so that it has minimal effects on the posterior distribution.

Dealing with the Marginal Likelihood

As mentioned before, the marginal likelihood can often be impossible or unfeasible to calculate analytically even in the case parameters with discrete values where we don't need to solve a complex integral. A common solution to this, that has been historically the only option, is to rely on so-called "conjugate priors". These are priors that follow distribution which multiplied by suitable likelihood function produce a posterior distribution that has the same form as the prior distribution. A good example of this is our example above (3.6) using a normal distribution for the prior when the likelihood is also normally distributed. If the likelihood function belongs to the exponential family of distributions, there exists at least one corresponding conjugate prior distribution (often also in the exponential family). For likelihood functions outside the exponential family there are some cases where a conjugate prior exists, but these are rare.

The conjugate function method makes the denominator analytically solvable, but it is in practice very limiting. First, we need to restrict ourselves when modeling to only use models that create a suitable likelihood function with an existing conjugate prior. Second, it limits our options in defining our prior beliefs as we have to be able to express them with the conjugate function. To have a more general solution we need to abandon the search for an analytical solution and tackle the problem algorithmically. We start this by noticing that as the marginal likelihood is calculated over all possible parameter values, it is not dependent on particular values of the parameter and is thus constant across them. Therefore we can drop it from the Bayes' formula (3.2) and state that the posterior distribution follows the shape of the distribution of likelihood times prior:

$$(3.7) \quad P(\Theta|D) \propto P(D|\Theta)P(\Theta)$$

Even though we can't calculate the exact posterior probability any set value of values for our parameters because we don't know the correct denominator, we now have distribution whose shape is identical to the posterior. If we could take samples from this distribution we could approximate the posterior distribution with the probability distribution of the values of the samples. The general method to get this kind of sample is the following:

1. Pick a set of values for our parameters as our starting position in the parameter space. The starting location has to be a plausible set of values. We can check this As we can know that $P(D|\Theta)P(\Theta)$ has the same shape as our posterior and so if the value of this function is zero when we pluck our set of parameter values in it, these values are impossible and should not be use as a starting point.
2. Pick a second set of values for our parameters with some random method.
3. Calculate the probability of moving from our current position in the parameter space to the second set of parameter values with the following formula:

$$(3.8) \quad p_{move \ to \ new \ location} = \min \left(\frac{P(D|\Theta = proposed \ set)P(\Theta = proposed \ set)}{P(D|\Theta = current \ set)P(\Theta = current \ set)}, 1 \right)$$

In other words, if the value calculated with the proposed parameter value set is higher than the value for the current set we always move to the new proposed location in the parameter space and if the value is lower we move there with a probability defined by the ratio of the two values.

4. Generate a random number between 0 and 1 and move to the proposed location in the parameter space if the number is lower than the probability that we calculated in the previous step. Otherwise we stay in the same place.
5. Mark down the values of our current location in the parameter space.
6. Repeat from step 2.

This method takes us on a random walk on the distribution with the steps defined by the likelihood times prior, but as this is distribution has an identical shape with the posterior function we indirectly end up with a representative sample of values from the posterior distribution. So in actuality we have taken a random walk in the posterior probability distribution, where we spend more steps in parts of the distribution where the values are more likely. To end up with an estimate of the posterior distribution we can now just divide the parameter space into parts, calculate how many times we visited each of these parts and divide this by the number of steps taken in total. We now have a proper probability distribution that approximates the true posterior distribution. If we want a more precise estimate, we can just take more steps and divide the parameter space into smaller and smaller parts.

This algorithm is a general description of a group of algorithms called Metropolis-Hastings algorithms. All of these methods follow the principles above, but they differ in the way they decide where to move next. Although with even very simple rules of picking a random proposed value the described will end up exploring the whole posterior distribution eventually, more clever algorithms can make this process much more efficient. By proposing steps that are more likely in posterior distribution a clever algorithm can lower the chance of proposing steps that have a high probability of being rejected and so assure efficient moving through the parameter space with less steps.

Metropolis-Hastings algorithms themselves are a part of a more broader group of algorithms called Markov Chain Monte Carlo methods (MCMC) that all are methods of sampling from a probability distribution. The development of these methods has been pushed forward by the need to have the sampling be as efficient as possible, as even with the computational power of a modern computer, more complex models can run into unfeasibly long computational times. On top of choosing the right algorithm, there are lots of other things related to the computation to consider when implementing Bayesian inference, but we will return to these later with the example of practical implementation of this kind of calculation with Python and PyMC3 package.

How to Define the Priors?

For the Bayesian formula to work we have to define a prior and this can be trickier than it seems at first glance. Ideally the prior would codify our prior beliefs and knowledge [8, 7], but trying to actually codify all we know is a fool's errand, because of how complex this is and we are anyway working with a model that is a simplified representation of reality. Even if we aim for a more limited goal, codifying what we know (or think we know) to mathematic formulas can be a tricky business. But it gets more difficult when we consider that we are doing the study not just to convince ourselves but also other people and we should try to capture the beliefs of our target audience. Unless this target audience is

easily available for questions and is highly mathematically minded, this leads to multiple layers of uncertainty as we try guess what others believe and then codify these beliefs with mathematical precision.

To avoid this hurdle the other option is to try to define some non-informative priors that “let the data speak for itself”, but this approach also has problems. Firstly, defining priors that have no effects on the posterior can be mathematically very difficult when we move beyond simple models [8] and if we would need to invest a lot of time finding these priors we should instead probably spend this time to define proper informed priors. Second, non-informative priors do not always seem so non-informative, when we actually know something about the issues. Consider a case where are trying to predict if a patient has a rare disease and we assign a uniform prior from 0 to 1 as this seems non-informative. This prior will have the effect of moving the posteriors towards the possibility of the patient actually having the disease. So if we definitely know something about the phenomenon, a non-informative prior can work against these beliefs.

Considered more broadly, chasing after truly non-informative prior is also somewhat of a fool’s errand if the data actually has “something to say”. If the data is informative, then lots of different priors that are somewhat weak should lead to almost identical posterior distributions. On the other hand if the data is weak and we have a weak prior, why are we even doing the analysis if we know that it will give us almost no information?

As a compromise between having strong priors based on previous beliefs and knowledge and letting the data stand on its own, Gelman et al. suggest using weakly informative priors [8]. These can be defined in two different ways. First we can start with non-informative prior (when it is easily available) and then add minimum information constraining the posterior results to reality (e.g. object cannot move faster than light or the height of an adult human must be between 50 and 300 cm). The other way is to start with strong informative priors and then broaden these to take into account the uncertainty of how applicable our priors knowledge is to the question at hand.

Because of how complex defining priors can get, this problem can act as an additional limiter on how complex Bayesian models we can build. The more complex model, the more priors we have to define and the more prior information and beliefs we could codify and the more difficult this problem gets.

3.2 Why Bayesian Inference Fits N-of-1 Studies

As mentioned at the beginning of the previous chapter, both frequentist and Bayesian inference can be used to estimate the parameter values of the models described. The choice between these two methods is not just a technical one as both come with different conceptions about the nature of probability and assumptions related to this conception

and these have serious implications on how we should interpret the results and run the studies. Unfortunately for frequentist inference, its assumptions fit the reality the N-of-1 trials poorly, and its estimates, and especially their uncertainty, can be hard to communicate to a lay audience (doctors and patients). Because of these issues, Bayesian inference can usually be recommended over frequentist inference when analyzing and communicating the results of N-of-1 studies, even though it comes with its own complexities that we previously went over.

3.2.1 Flexibility of Experimental Design

The central tenet of frequentist inference is the assumption that the experiment could be in theory repeated with statistically independent results. Probability is defined in relation to these hypothetical repetitions as a proportion of them that would have some event of interest (e.g. over 5 tails with 10 coin tosses). To repeat the same experiment its design needs to be defined precisely and this definition needs to be made before the study has begun because the changes made to the design during the experiment might be influenced by its results.

Lets take for example an experiment where we are trying to find out if a coin is fair. For this purpose, we decide to flip the coin 20 times and count the number of heads. Then we can model how “typical” the result we got would be for a fair coin. We do this by calculating the distribution of heads from multiple 20 coin toss trials assuming that the coin is fair. Then we can compare our actual result against this distribution. If the result would be typical within this distribution, then we can say that the null hypothesis stands and the coin is fair. The problem comes if we did not actually decide to throw the coin 20 times beforehand, but first threw it 10 times, looked at the results, thought that we should get some more data, and then threw the rest 10. Now what kind of distribution should we compare our results against? The length of the trials seems to now hinge in some undefined way from the results of the first 10 coin tosses that made us decide to keep tossing the coin some more.

This demand for predefined experiment designs can be especially problematic in N-of-1 studies used in a clinical setting. There might be cases when some treatment seems clearly better in the middle of an experiment and the patient (or the clinician) wants to stop the experiment. As the subjects are real people, it would be unethical to force them to continue the study just for the sake of statistical rigor. There might also be a situation where a couple of treatments seem promising, but the rest seem completely useless, and we would want to change the design of the experiment on the fly to focus just on the promising treatments. These changes to the predefined experimental design will brake frequentist assumptions and make the inferences made with this method very difficult.

The possibility to stop the study or change the experimental design can be incorporated

to frequentist inference if we predefine the rules of when to stop the experiment and under what circumstances the design should be altered and how. Then we need to take these rules into account when performing the estimation calculations which adds complexity but is still possible. The problem is that it is questionable if we can actually define and stick to these rules because of the human element: we would need to convince the patient that she should stop the experiment only in these predefined circumstances. This seems quite unrealistic, as if there are any unforeseen problems with the treatments, it is more than likely that the patient will want to stop whatever rules she sign to follow beforehand.

With Bayesian inference doing on the fly modifications of the study design do not cause such problems as with frequentist inference, as there is no assumption about repeating the experiment. Instead of hypothesizing about future experiment, we just take into account our prior information and the data created by the current experiment. The changes to the experimental design modify what data we get out of it, but don't brake any assumptions of Bayesian inference. This adaptability is a big benefit when applying N-of-1 designs in a clinical setting, as the unpredictable nature of the patients and treatment effects and all the other practical considerations that could force changes to our experiment design don't break the assumptions of the inference method.

3.2.2 Richness and Communicability of the Estimates

In N-of-1 studies, much consideration has to be given to the communication of the results, both to the patient and the clinician administering the experiment. We would like to portray not just the estimates of the effects of the treatments, but also the uncertainty associated with these estimates. In frequentist inference there are two options to communicate these facts. The first one is maximum likelihood estimates combined with confidence intervals. Maximum likelihood estimates are just point estimates of the most likely parameter values and don't communicate any uncertainty. Confidence intervals of these parameter values communicate uncertainty, but don't actually mark the most likely values of the estimate. A 95 % confidence interval, is often interpreted wrongly as including the true value of the parameter with 95 % change, but the real meaning is that if we repeated the experiment indefinitely 95 % of the 95 % confidence intervals would include the true parameter value. This is because in frequentist interpretation of probability, true parameter values are not random but fixed, so a single confidence interval either does or does not include the true value. This confusing definition makes confidence intervals a poor choice of presenting the uncertainty of the estimates to a lay audience as they might create lots of misinterpretations and even when interpreted correctly are hard to grasp.

The second frequentist communication tool are p-values of hypothesis tests. We might define a null hypothesis that the parameters of the effects of different treatments are equal and calculate the likelihood to obtain the data or a more extreme version of it if

this is indeed the case. This would give us a more easily communicable way to portray the probability to obtain the data if the null hypothesis is true. If the p-value is low we could say with confidence that there is real differences between the effectiveness of the treatments. The problem is that this actually gives us very little useful information, as we would also like to know how big the differences in the treatments' effectiveness could realistically be.

A Bayesian approach to inference gives us richer and more easily communicable information about the estimates because it gives us a distribution as a result in the form of the posterior distribution. With this we can easily point out the plausible values with the tops of this distribution and the uncertainty with how flat the distribution is. The usual misinterpretation of the confidence intervals as containing the true value of the parameter with a certain probability is actually the right interpretation for Bayesian credible intervals, that contain certain percentage of the probability mass of the posterior distribution and so have a defined chance of containing the true value of the parameter. Bayesian method itself also seems to fit really well with the ways that clinicians conceptualize their practice of arriving to diagnosis for a patient as updating their beliefs (i.e. priors) with the new information gained from new observation such as test results [4].

Chapter 4

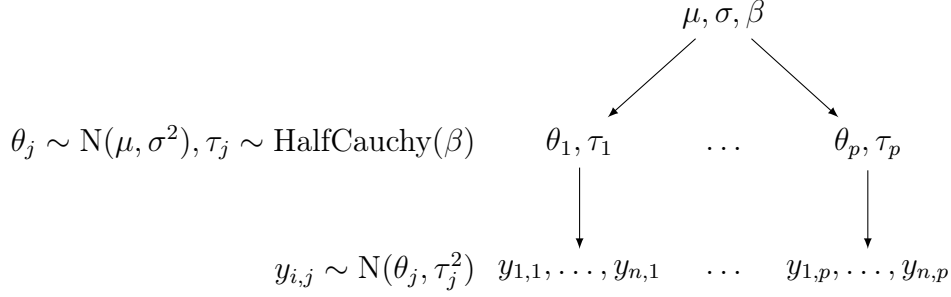
Combining Information From Several N-of-1 Trials With Hierarchical Models

Let's say we have our N-of-1 trial set up nicely and we have a good model for analyzing the results. After running the first trials we continue to perform the same kind of setup with multiple patients as the treatments we are considering seem to have lots of individual variation in their effectiveness. Running these experiments separately is fine, but one might wonder could these separate trials be somehow combined. Of course, we could pool the data and run a group-based analysis about what treatments seems to be the best in general, but there is a better way to keeps true to the goal of N-of-1 experiments of finding the best treatment for each *individual patient*.

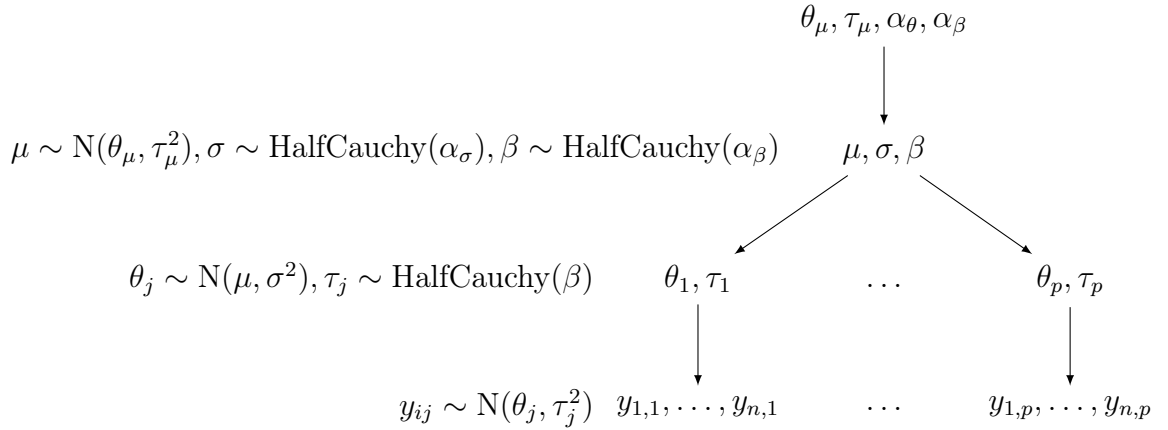
In the Bayesian framework we can do this by using hierarchical models where each model parameter is imagined to come from "higher" distributions that is controlled by its own parameters. Individual parameters lower-level parameters represent the individual patients, the distribution where these parameters come from represents the population which the patients are drawn from and for this higher-level distribution we give our priors to. Lets take a simple example where we have p patients with n measurements and, to make things simple just one treatment option. We can model this by a simple two-parameter model:

$$(4.1) \quad y_{ij} = \theta_j + \epsilon_{ij}, i \in (1 \dots n), j \in (1 \dots p)$$

where θ_j represents the effect of the treatment for patient j and $\epsilon_{ij} \sim N(0, \tau_p^2)$ is the error term. As a Bayesian model we could represent the model conceptually like this:



On top are the prior parameters that represent our existing knowledge and are shared by all patients. Next are actual prior distributions defined by these parameters from which the individual patient level patient parameters are drawn. At the bottom we describe the actual data generating process where values of individual observations are drawn from normal distributions defined by the patient level parameters. To broaden this model from just being a bunch of patient-specific models (with shared priors) we can combine them by adding a population level that links patients together:



Now instead of drawing the patient level parameters from priors, these are drawn from the population level distribution. The parameters defining the population level distribution are drawn from the prior distribution. So instead of letting the patient level parameters being depend directly from the prior parameters, we have introduced an additional layer between these.

With all the parameters that our model now has, the question now becomes, how we apply the Bayesian formula in this case? Luckily this happens to be very simple as we just have to remember two rules of probability: the chain rule, that tells us that the probability of A and B is the probability of B on condition A times the probability of A ,

and the rule of independence that states that the probability of A on condition B is just the probability of A if A and B are independent.

$$(4.2) \quad P(A \cap B) = P(B|A)P(A)$$

$$(4.3) \quad P(A|B) = P(A), \text{ if } P(A \cap B) = P(A)P(B)$$

Now lets see what happens when we apply write are our model in the Bayes formula 3.2 and apply these rules. For simplicity I assume that we only have two patients and leave out the prior parameters defining the shape of the population distribution.

$$\begin{aligned}
\text{posterior} &= \frac{P(\theta_1, \tau_1, \theta_2, \tau_2, \mu, \sigma, \beta|Y)}{P(Y)} \\
&\propto P(Y|\theta_1, \tau_1, \theta_2, \tau_2, \mu, \sigma, \beta)P(\theta_1, \tau_1, \theta_2, \tau_2, \mu, \sigma, \beta) \\
&\text{(using rule of independence as data Y depends only on parameters } \theta \text{ and } \tau) \\
&= P(Y|\theta_1, \tau_1, \theta_2, \tau_2)P(\theta_1, \tau_1, \theta_2, \tau_2, \mu, \sigma, \beta) \\
&\text{(using rule of independence to separate the two patients)} \\
&= P(Y_1|\theta_1, \tau_1)P(Y_2|\theta_2, \tau_2)P(\theta_1, \tau_1, \mu, \sigma, \beta)P(\theta_2, \tau_2, \mu, \sigma, \beta) \\
&\text{(using the chain rule separate the patient and population parameters)} \\
&= P(Y_1|\theta_1, \tau_1)P(Y_2|\theta_2, \tau_2)P(\theta_1, \tau_1|\mu, \sigma, \beta)P(\theta_2, \tau_2|\mu, \sigma, \beta)P(\mu, \sigma, \beta)^2 \\
&\text{(population parameters are independent of each other)} \\
&= P(Y_1|\theta_1, \tau_1)P(Y_2|\theta_2, \tau_2)P(\theta_1, \tau_1|\mu, \sigma, \beta)P(\theta_2, \tau_2|\mu, \sigma, \beta)P(\mu)^2P(\sigma)^2P(\beta)^2 \\
&\text{(\theta and } \tau \text{ are independent on each other and are dependent on different population level parameters)} \\
&= P(Y_1|\theta_1, \tau_1)P(Y_2|\theta_2, \tau_2)P(\theta_1|\mu, \sigma)P(\tau_1|\beta)P(\theta_2|\mu, \sigma)P(\tau_2|\beta)P(\mu)^2P(\sigma)^2P(\beta)^2
\end{aligned}$$

We can see that the measurements of each patient Y_1, Y_2 depend only on the patient level parameters $\theta_1, \theta_2, \tau_1, \tau_2$, so that if these values are set, then the measurements are independent of all other parameters. Similarly, the patient level parameters depend conditionally only on the population level parameters μ, σ, β . So we have now successfully expressed our hierarchical model in the Bayes formula. We could continue by writing the formula out by adding in the prior terms and replacing the abstract probability markings

with the actual probability density functions given by our chosen distributions, but this gets messy and, in this case, no nice analytical solution can be found anyway, so we will leave it at this.

The elegance of this method comes from the fact that, even though we now have population-level parameters we still keep the individual patient-level parameters that can vary between the patients. As the population-level parameters are not fixed, the patient-level data will inform us about the population-level parameters and this in turn affects the patient-level parameters. These kinds of dependencies among parameters are useful in several respects. First, the dependencies are meaningful for the given application, e.g. we can model the treatment effect for a single patient as an instance from a broader population distribution of this effectiveness. Now the model will give us information also about the general population level and not just about the single patient. Second, because of the dependencies across parameters, all the data can now jointly inform all the parameter estimates. The reduction of variance in the estimators that this effect causes, is generally referred by the term “shrinkage”. In general, shrinkage in hierarchical models causes lower-level parameters to shift toward the modes of the higher-level distribution. If the higher-level distribution has multiple modes, then the low-level parameter values cluster more tightly around those multiple modes, which might pull some low-level parameter estimates apart instead of together. The greatest thing is, that as we don’t explicitly set the parameter values of the higher-level distributions, the amount of shrinkage is informed by the data so that similar observed data points from lower-level distributions lead to “tighter” estimates for the higher-level distributions and in this in turn leads to greater shrinkage (e.g. posterior estimates of the patient level parameters being more tightly clustered together).

Chapter 5

Example of a Hierarchical Bayesian Analysis Using Simulated Data

To put everything we learned into practice, let's go over an example Bayesian analysis of multiple N-of-1 trials, where first analyze the trials separately and then combine them using a hierarchical model. For the analysis we use Python and PyMC3 library [13] built for probabilistic programming. We will not go over each part of the code, but focus only on the critical parts where we see how the models and inference we went over in the previous chapters look like when implemented in practice. For those who want to see the full code it is available from a GitHub repository [14] with instructions how to run it.

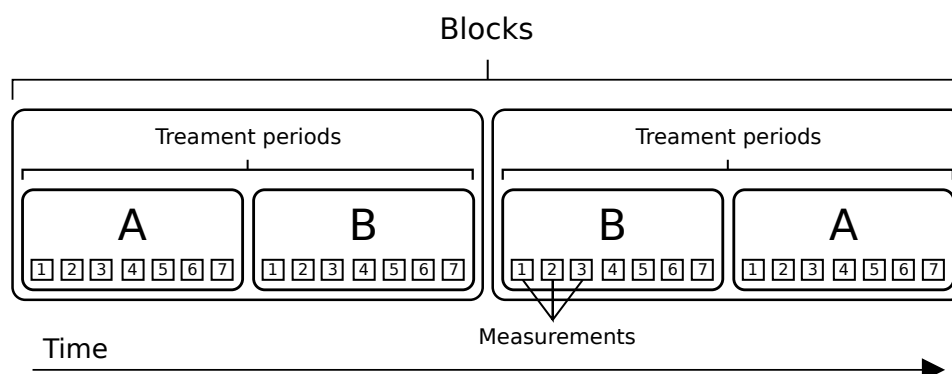
As we now jump from theoretical and analytical considerations to practical consideration when doing this kind of analysis, we will introduce some new concepts related to how PyMC3 is actually performing the calculations and what do we need to pay attention to make sure that what we get out is actually reliable. We will see how important visualizations are in interpreting the output and what new metrics to pay attention to diagnose if our model is performing adequately.

5.1 Defining the Experiment

Let's first describe our experiment design, because this is critical for understanding everything that comes later. Imagine that we have 6 patients that all suffer from the same medical condition to which there are two treatment options available (A) and (B). We know from the literature that there is clear patient-level variation in which treatment works better so we decide to test this with N-of-1 trials. We know that both of the treatments act very fast and that there are no known problems with interaction effects between them. The state of the condition can also be easily measured at home so no visits

to a hospital are needed to take the measurements. Higher measurements mean worse condition so we would like to find the treatment that gives the lowest measurements. The condition is usually steadily worsening with or without the treatment as the treatment is more for managing the disease and not curing it. Based on this information we come up with the following experiment design:

- The trial will run for 4 weeks with 2 weeks on each treatment
- Treatment periods will be one week long
- The week-long treatment periods will be organized into two blocks, each of which will run for 2 weeks and contain 1 week on each treatment
- The sequence of the treatments in the first block will be randomized and then the second block will have the treatments in reverse order creating a balanced design (ABBA or BAAB)
- Measurements will be taken by the patient at the end of each day so we end up with 28 measurements per patient with 14 for each of the treatments that separate by roughly equal time periods
- There will be no washout period when switching between the treatments



5.2 Simulating the Data

Next we simulate the data that we could get from our defined experiment with Python. We do this in a fairly simple way that makes it easier to see how the models we build are

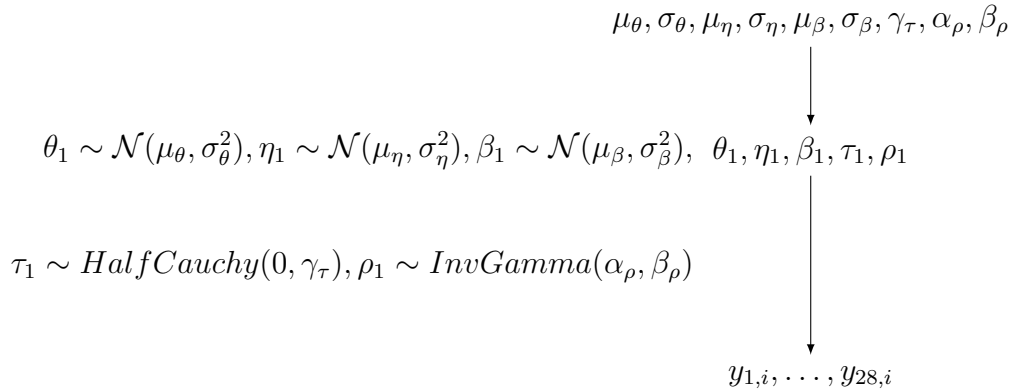
actually doing. We don't go over the actual code that creates the data, but just look at the mathematical model used. Code for creating the data can be found in the repository [14]. For simulating each observation t for patient i we use the following model:

$$(5.1) \quad y_{ti} = Z_t \theta_i + W_t \eta_i + t \beta_i + \epsilon_t + \rho_i \epsilon_{t-1},$$

where θ_i and μ_i are the treatment effects with Z_{ti} and W_{ti} being indicator variables that are 1 when observation t is within a period where treatment was applied and 0 otherwise. β_i is the trend representing the natural progress of the condition. $\epsilon_{ti} \sim N(0, \tau_i^2)$ is the error term at t and ρ_i is the correlation between the consecutive errors.

This data generating model has both a trend and a lag-1-autocorrelation. The trend we are going to be modeling later, but also modeling autocorrelation would make the model too complex for the limited amount observations we will be dealing with. This way the example tries to be more realistic by having the data generation function to be of such complexity that it cannot be fully modeled and some decisions have to be made on how to try to approximate the process with a simpler model.

“Above” the part creating the individual observations we have a population-level parameters that define the distribution from which the patient-level parameters are drawn from:



We set the population-level parameters to the following values:

population treatment A mean: $\mu_\theta = 10$
 population treatment A standard deviation: $\sigma_\theta = 0.2$
 population treatment B mean: $\mu_\eta = 9.7$
 population treatment B standard deviation: $\sigma_\eta = 0.3$
 population trend mean: $\mu_\beta = 0.02$
 population trend standard error: $\sigma_\beta = 0.01$
 population measurement error scale: $\gamma_\tau = 0.3$
 population error autocorrelation alpha: $\alpha_\rho = 100$
 population error autocorrelation beta: $\beta_\rho = 200$

There are some important things to note about these values. First, we defined the treatment A to usually have higher mean than treatment B meaning that the treatment B should perform better (lower measurement values) for most patients. Second, the mean of the trend β is positive and its variance is small. This means that almost all patient should have a steadily worsening condition (measurements get higher as the experiment progresses).

Lets now simulate the data with these models and parameter values and see what we get:

Here we see the measurement at each time index for each patient in separate lines. We have highlighted patient 1.

We can clearly see the for the highlighted patient one treatment is working better than the other one as we can easily pick out the ABBA treatment assignment from the picture. We can also see that for this patient the t

Let's next look at the parameters themselves

5.3 Analyzing a Single Trial

We start with the case of analyzing just a single patient.

Normally before we start creating any models we would first start with an extensive exploration of the data, but as teaching preliminary analysis is not the focus here and we created the data ourself and so know exactly what kind of distribution it follows, we will skip this here. For analyzing any real-world data this part would be *mandatory*.

5.3.1 Defining the Model

We start with a very simple model with just the treatment parameters and an error term:

Figure 5.1: Timeline of Measurements for Each Patient

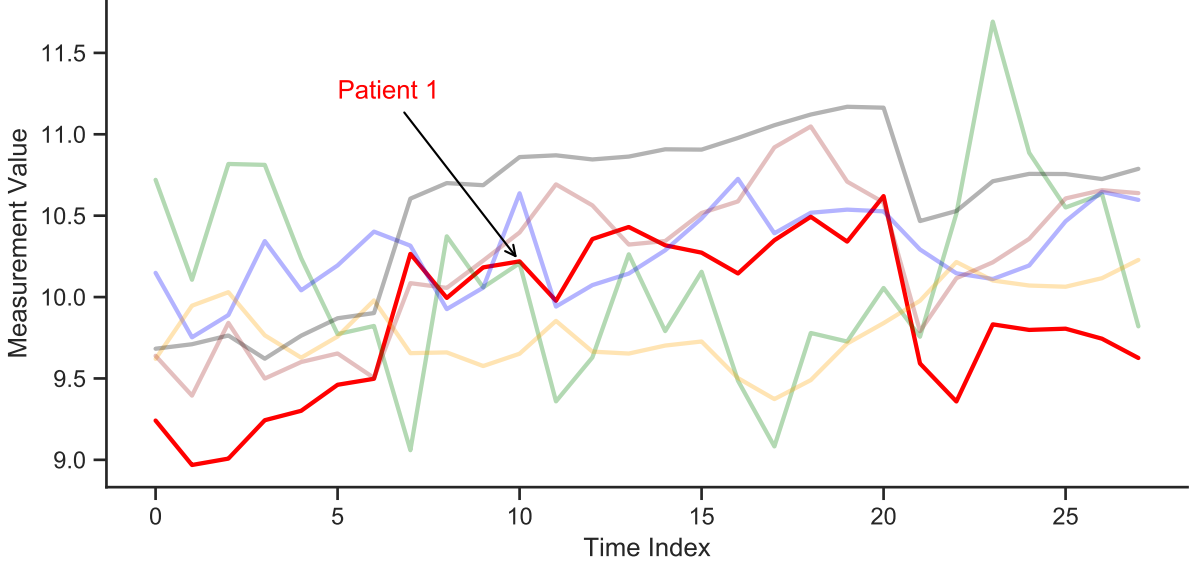
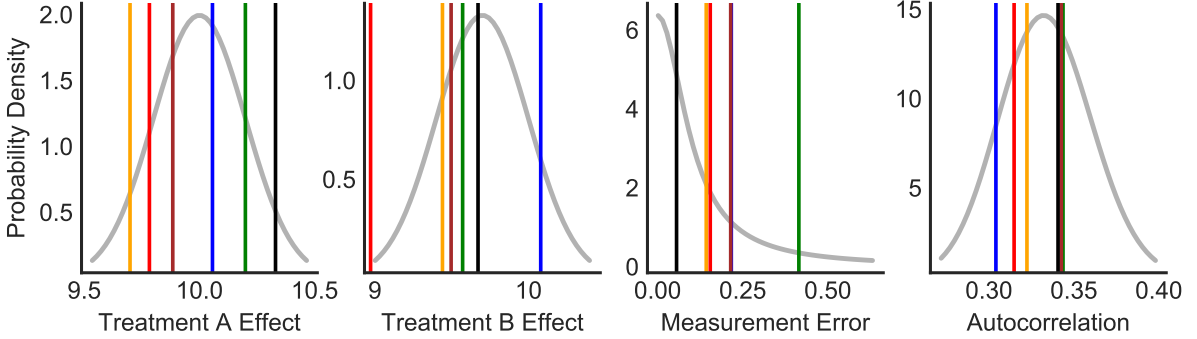


Figure 5.2: Population Parameter Distribution and the Individual Patient Parameters



$$(5.2) \quad y_t = Z_t\theta + W_t\eta + t\beta + \epsilon_t$$

where θ and η are the treatment effects with Z_t and W_t being indicator variables that is 1 when observation t is within a period where treatment was applied and 0 otherwise. $\epsilon_t \sim N(0, \tau^2)$ is the error term at t .

Our model assumes that the measurement values are distributed normally with variance τ^2 and a mean of either θ or η depending on which treatment is applied at the

observation $y_t \sim N(Z_t\theta + W_t\eta, \tau^2)$ and so our likelihood function is a bit of special normal distribution with a varying mean. This will cause use some headache later as this is definitely a not a text book example.

Next we go to choose the prior distribution for our model. We choose normal distributions for the treatment effect parameters as it makes sense to think that the effectiveness would depends on multiple independent biological and social factors (very reasonable assumption!) then the central limit theorem tells us the that the effect should be normally distributed. For the variance of the error term we choose a half-Cauchy distribution.

$$\begin{aligned}
 (5.3) \quad & \theta \sim N(\mu_\theta, \sigma_\theta) \\
 & \eta \sim N(\mu_\eta, \sigma_\eta) \\
 & \beta \sim N(\mu_\beta, \sigma_\beta) \\
 & \tau \sim HalfCauchy(0, \gamma_\tau)
 \end{aligned}$$

Lets define next what kind of parameters we should give to our prior distributions.

$$\begin{aligned}
 (5.4) \quad & \text{treatment A mean prior: } \mu_\theta = 10 \\
 & \text{treatment A standard deviation prior: } \sigma_\theta = 0.2 \\
 & \text{treatment B mean prior: } \mu_\eta = 10 \\
 & \text{treatment B standard deviation prior: } \sigma_\eta = 0.3 \\
 & \text{measurement error scale prior: } \gamma_\tau = 0.1
 \end{aligned}$$

5.3.2 Implementing the Model in PyMC3

PyMC3 is Python package that is built for probabilistic programming. Although it is not limited to Bayesian methods, that is its main focus. The package work in principal so that you give it all the parts of the Bayesian formula: the likelihood, prior distribution and the values defining the prior distributions. Then it automatically chooses a Markov Chain Monte Carlo method (MCMC) that it thinks can most efficiently algorithmically approximate the posterior distribution. After it has solved to the model, it has multiple inbuilt model checking methods that the user can use to get a sense of how well the approximation is working. This is very nice, as this takes out a huge chunk of work from us, namely choosing and implementing the approximation method ourselves. But now that we have entered the territory of algorithmic solutions, we have introduced a new hurdle, namely checking that the algorithm is actually works for our defined model and gives us good approximations. On top of this we of course have all the usual stuff related

to model accuracy and checking that our model itself is a good approximation of the reality.

Lets just jump in and implement our model in PyMC3 and then walk trough what the code is actually doing line by line¹.

```
import pymc3 as pm
import pandas as pd

measurements_df = pd.read_csv("patient_measurements.csv")

with pm.Model() as single_patient_no_trend_model:

    # separate priors for each treatment
    treatment1_prior = pm.Normal("treatment1", mu=10, sigma=10)
    treatment2_prior = pm.Normal("treatment2", mu=10, sigma=10)
    # common variance parameter defining the error
    gamma_prior = pm.HalfCauchy("gamma", beta=10)

    # measurements are created from both priors, with a indicator setting the
    # values to 0 if the particular treatment is not applied at the observation
    measurement_means = (
        treatment1_prior
        * measurements_df[patient == 0]["treatment1"]
        + treatment2_prior
        * measurements_df[patient == 0]["treatment2"]
    )

    # likelihood is normal distribution with the same amount of dimensions
    # as the patient has measurements and and the mean is defined by either
    # the treatment 1 prior or treatment 2 prior with the same sigma
    likelihood = pm.Normal(
        "y",
        measurement_means,
        sigma=gamma_prior,
        observed=measurements_df[patient == 0]["measurement"],
```

¹The code snippets that we will be showing are a slight simplification of the actual code that was used to calculate the results. The simplification does not mean that the code shown is not runnable, but parts of it that are related to technical niceties (e.g. inter operational system compatibility, saving the plots ...) are taken out to focus on what is important for Bayesian calculations

```

)

# running the model
trace = pm.sample(800, tune=500, cores=3)

# Checking diagnostic describing how the MCMC method performed
pm.traceplot(trace, ["treatment1", "treatment2", "gamma"])
plt.show()
pm.summary(trace)

# checking the posterior distribution
pm.plot_posterior(trace)
plt.show()

# posterior sampling
with single_patient_no_trend_model as model:
    post_pred = pm.sample_posterior_predictive(trace, samples=500)
    predictions = post_pred["y"]

draw_posterior_checks(
    predictions=predictions,
    measurements_df=measurements_df[patient == 0],
    parameters_df=parameters_df[parameters_df["patient"] == 0],
)

```

Loading in the Package and the Data

```

import pymc3 as pm
import pandas

measurements_df = pd.read_csv("patient_measurements.csv")

```

The first thing we of course have to do is to load the package and get our data. There is not much to comment about loading the package, but lets show how our data looks like:

patient	measurement	period	block	treatment1	treatment2	measurement
0	0	0	0	0	1	9.241493
0	1	0	0	0	1	8.968412
0	2	0	0	0	1	9.007580
0	3	0	0	0	1	9.243449
0	4	0	0	0	1	9.301344
0	5	0	0	0	1	9.461129
0	6	0	0	0	1	9.498025
0	7	1	0	1	0	10.265324
0	8	1	0	1	0	9.994795
0	9	1	0	1	0	10.182349

All columns except the last one are index columns. As is normal in Python indices are zero-indexed. First indexes the patient (0-5), second the measurement (0-27), third the treatment period (0-3), fourth the block (0-1) and fifth and sixth which treatment was applied during the particular measurement (0-1). Last column is the actual measurement. PyMC3 does not require any particular format from the data and almost any format is possible as long as you can feed the right values into the right places in the model definition. Also using the Pandas package to handle the data as a dataframe is not necessary, but is very convenient.

Initializing the Model

```
with pm.Model() as single_patient_no_trend_model:
```

PyMC3 models are usually defined within the Python context manager (with statements). This is not necessary, but makes the model definition much cleaner as when the model is defined in the context manager, all subsequent parts defined within this context are automatically associated with this particular model. This gets rid of the need to manually define for each part, which model we want to associate it with.

Defining the Priors

```
treatment1_prior = pm.Normal("treatment1", mu=10, sigma=10)
treatment2_prior = pm.Normal("treatment2", mu=10, sigma=10)
gamma_prior = pm.HalfCauchy("gamma", beta=10)
```

Now we start getting to the actual modeling. We start by defining our prior distributions. For both treatments we use a normal distribution with the same mean (assume that we got the mean from the existing literature) and a very large variance. For the prior for the variance of the error term γ we use a Half-Cauchy distribution with also very large variance. The location term does not need to be defined as it is by default zero, just like we want it. These priors could be considered to weakly informative priors as they contain some actual information (namely the mean of the treatment effects), but are very wide and don't make strong assumptions.

As the distributions that we are using are all widely used distributions with well-known properties we can find functions implementing them already built-in PyMC3. It is also possible to define completely custom distributions, but unfortunately this is quite complex, as you don't just have to worry about the math part, but also all you have to implement all the methods that PyMC3 relies on when making the calculations².

Defining the Likelihood

```
measurement_means = (
    treatment1_prior
    * measurements_df[patient == 0] ["treatment1"]
    + treatment2_prior
    * measurements_df[patient == 0] ["treatment2"]
)

likelihood = pm.Normal(
    "y",
    measurement_means,
    sigma=gamma_prior,
    observed=measurements_df[patient == 0] ["measurement"],
)
```

Defining the likelihood is the most complex part our model definition because of the complexity introduced by the varying treatments. Because of this, the likelihood cannot be implemented as a simple normal distribution. We solve this problem by instead using

²This complexity is a reason why the author would recommend using the Stan package for very complex models with lots of weird distributions. Although Stan requires to you to write model definition with its own language, this hurdle is worth in some cases for the greater flexibility as you just worry about the math and there is no need to consider implementing any helper functions

a multivariate normal distribution with as many dimensions as the patient has measurements. For each of the dimensions one of the two treatment effects priors is used for the mean and the gamma distribution is used for variance in all cases. We construct the vector of treatment means by multiplying the treatment priors by the treatment indices (1 when treatment given, 0 when not). So instead of considering the observations separately we think of them as a single multidimensional sample from multivariate normal distribution. This is not the only possible way to solve the problem, but happens to be the most convenient one here. If we would have some dependencies between the measurements in our model (e.g. autocorrelation), this approach would not work. In this case we would have to implement the likelihood distribution ourselves.

Running the Model

```
trace = pm.sample(draw=800, tune=500, cores=3)
```

Now that we have defined our priors and our likelihood we can move on to running the model. Notice that priors and the likelihood is all we need to have defined. We don't have to worry about the "bottom part" of the Bayes formula 3.1. PyMC3 will now estimate the posterior with an automatically chosen Markov Chain Monte Carlo (MCMC) method that moves through the posterior from point to point. We could also manually choose what algorithm to use, but unless we see problems when we evaluate the estimation efficiency, we should go with the automatic selection. We also don't need to define the point in the posterior distribution where the chain starts as this is also automatically chosen. This we could also define manually, but unless we run into problems, we should not need to.

The three parameters that we have to manually define are as follow:

- draw = how many moves the MCMC chain should calculate
- tune = how many moves the MCMC chain should take before it starts to take the observations into account
- cores = how many chains we should calculate in parallel

The total steps taken by each chain is draw + tune. The reason for the tuning step is that the starting position of the chain is probably biased. We would want the starting position to be a sample from the posterior so that the probability of it being in a certain point would be defined by the posterior. We can do this by letting the chain first take a number of steps and as it moves through the posterior its location will start to reflect the actual posterior. If we would not have the tuning steps it would be possible that the

chain starts in a weird position and the first steps it takes don't reflect the posterior as it might have to make lots of moves in a certain direction to get to the representative parts of the distribution. The more draws and tuning steps we take the better the estimation will be, but the more time the calculation will take. Finding the right number of draw and tuning steps involves usually some trial and error. It is hard to know how many steps you need before you try to run the model. Luckily PyMC3 will automatically alert us if it thinks that we need more tuning steps or draws and we can also diagnose this from metrics and plots that we will look at after we have run the model.

As modern CPUs have usually multiple cores, we can speed up the calculation by running multiple chains in parallel. Each chain will be independent of each other and each will take same amount of draws and tuning steps. As the chain can run in parallel, having multiple chains running at once does not make the other calculations any slower. This can dramatically lower the time that our estimation takes. The total number of steps that the posterior will be calculated with is the combined draws (but tuning steps) of the chains. So in our case we will use 3 cores and have 3 chains (running this on a 4 core CPU so we will leave 1 core free for other tasks), in total these chains will make $(800 + 500) * 3 = 3900$ steps, of which $800 * 3 = 2400$ steps will be used for the estimation.

Checking Algorithm Diagnostics

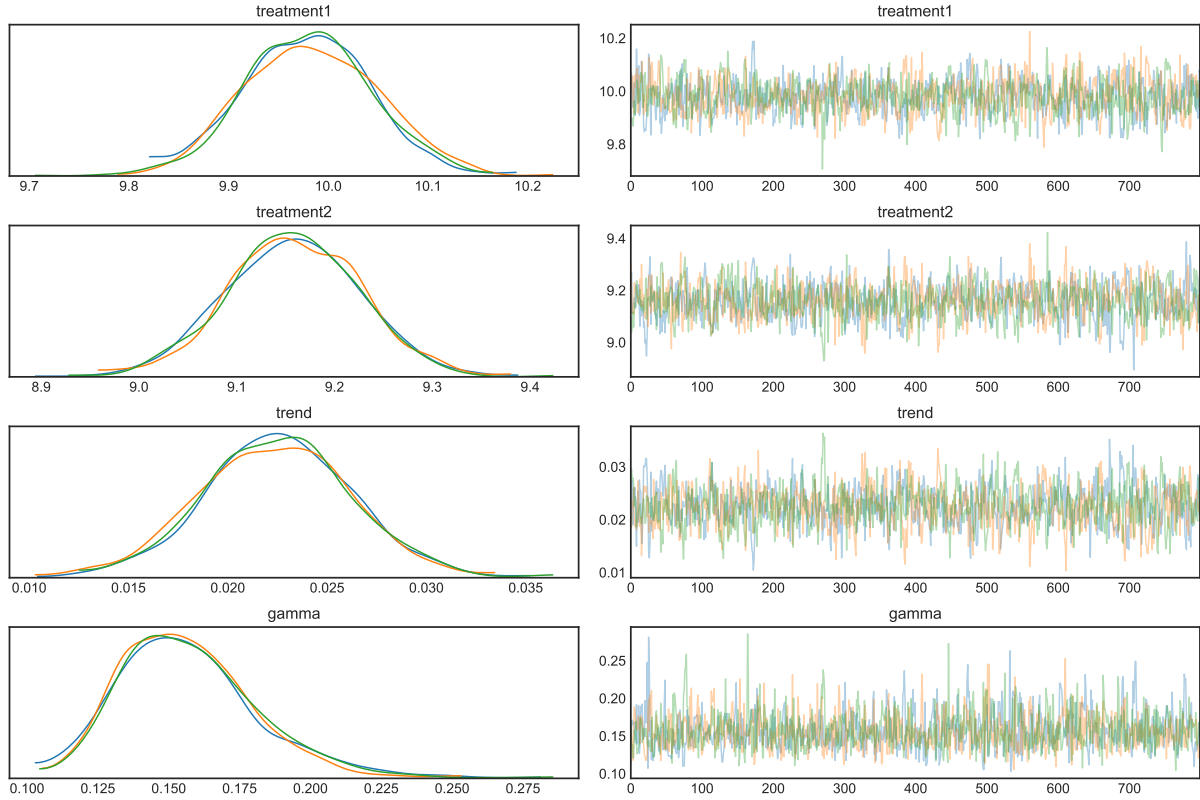
```
# Checking diagnostic describing how the MCMC method performed
pm.traceplot(trace, ["treatment1", "treatment2", "gamma"])
plt.show()
pm.summary(trace)
```

Now that we have run the model we need to evaluate if the algorithm did good job with the estimation. We can evaluate this by looking at diagnostic plots and from diagnostic metrics. Lets first look at the traceplots:

On the left we can see the steps taken for each chain in our three parameters. These are just the steps used for the posterior estimation and don't include the tuning steps. Each chain is portrayed with a different color. What we want to see here is that the timeseries looks like noise with no obvious patterns. If we would see these, it would mean that the chains had problems moving trough the posterior and ended up taking biased samples. These effects should be most pronounced at the beginning steps and if we would see these, we should increase the number of tuning steps.

On the right we can see the distribution of the posterior values for each chain. Here we would want to see that the distributions from different chain more or less align with each other. If this is not the case it means that the chains got stuck in different parts of

Figure 5.3: Single Patient Traceplots



the posterior. This would also show on the left side plots with the different chains staying in different parts of the posterior and being distinguishable from each other. If we see this problem the solution is once again to increase the tuning steps.

	mcse_mean	mcse_sd	ess_mean	ess_sd	ess_bulk	ess_tail	r_hat
treatment1	0.001	0.001	2712.0	2712.0	2724.0	1680.0	1.0
treatment2	0.001	0.001	2349.0	2348.0	2351.0	1830.0	1.0
gamma	0.001	0.001	2148.0	2140.0	2127.0	1703.0	1.0

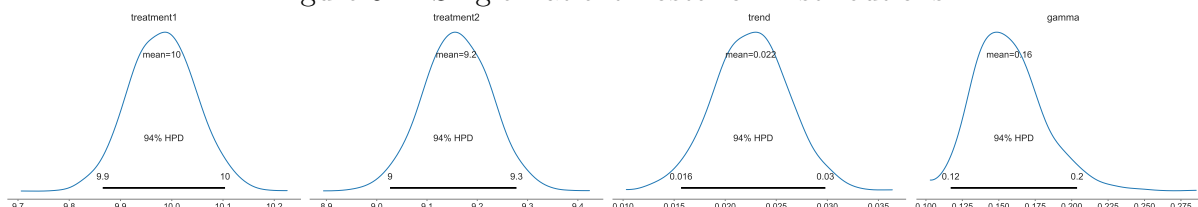
If increasing the tuning steps does not seem to help we have to consider two things: the MCMC algorithm chosen by PyMC3 is not a good match for our model or our model might not be properly defined. The first thing is try to manually choose a different algorithm than PyMC3 automatically chose (can be seen from the model computation printout) and seeing if we get better results. If this does not seem to work, we have to over our model and think if the model is in some way improperly defined.

Checking the Posterior

```
pm.plot_posterior(trace)
```

If the estimation looks fine, we can move on to checking the actual results and look at the computed posterior distributions. In this case just looking at individual posterior parameter estimates is not enough, as we are interested in the distribution of the *differences* between the two treatments.

Figure 5.4: Single Patient Posterior Distributions



Checking Model Fit with Posterior Sampling

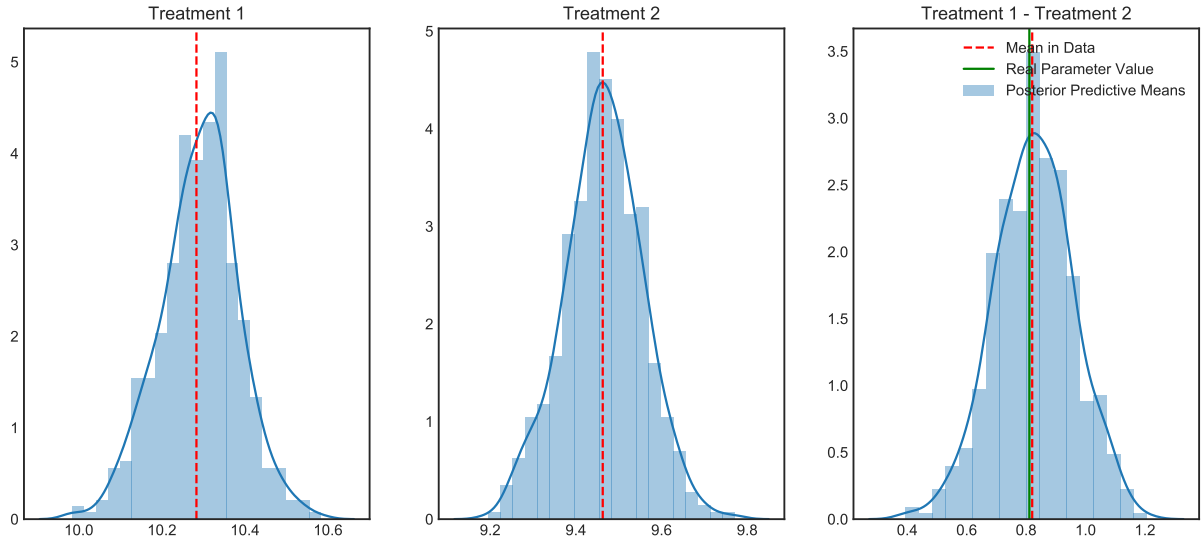
```
# posterior sampling
with single_patient_no_trend_model as model:
    post_pred = pm.sample_posterior_predictive(trace, samples=500)
    predictions = post_pred["y"]

draw_posterior_checks(
    predictions=predictions,
    measurements_df=measurements_df[patient_index == 0],
    parameters_df=parameters_df[parameters_df["patient_index"] == 0],
    plot_name="single_patient_no_trend_posterior_sampling",
)
```

After checking the trace diagnostics to see if the algorithm is doing a good estimating the posterior there is still one more important diagnostic to make: we need to see if the estimated model actually fits the data nicely. We do this by generating samples from the posterior distributions and for each of these samples create simulated dataset based on the chosen parameter values. For this purpose we have written a special function called “draw posteriors”. We don’t go over the code of this function, but what it does is to

use the sampled posterior parameters with model we specified and generate a dataset that simulates data from a new experiment, where the data generating process works exactly like our specified model. It does this for every parameter sample we picked (in this case 500 samples) and then calculates the mean treatment effectiveness for both of our two treatments (taking the measurements done under this treatment) for each simulated new experiment and then plots these means as distribution against the mean treatment effectiveness calculated from the original data. It also does for the difference of the treatment effectiveness and in this case also compares it to the actual difference in the treatment effectiveness that we picked for the patient. Normally we would not know this true value, but in this case we do as we generated the data ourselves. This makes it possible to see how accurate the model is and how uncertain are the predictions.

Figure 5.5: Single Patient Posterior Sample



We see that the distribution of mean treatment effectiveness calculated from the generated datasets matches nicely with the actual mean of the data. This means that our model is not biased. Also when we look at the difference in the treatment effectiveness, we see that our model captures the true value quite nicely.

5.3.3 Final Results

5.4 Analyzing Multiple Trials With Hierarchical Models

Lets next combine all the trials to understand general patterns and see if what kind of an effect the population parameters have on the estimates we got from our single patient model.

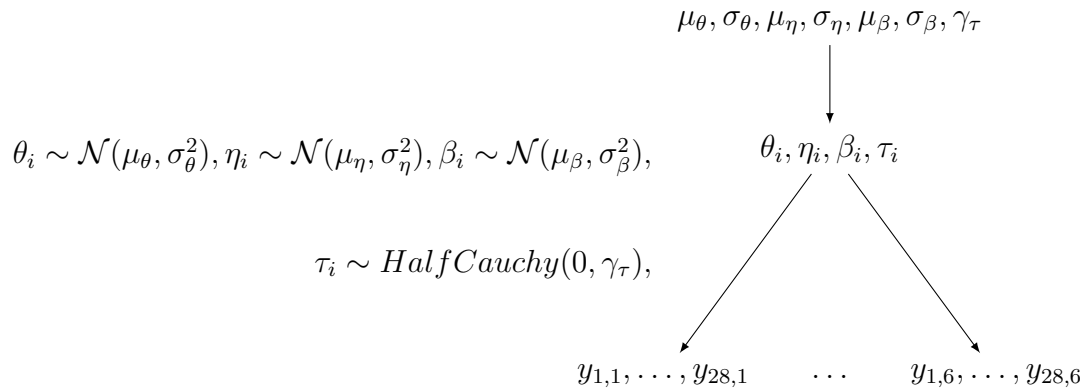
5.4.1 Defining the Model

For the single patients we use the same model as before, but now we index all our measurements not just by time t , but also by patient i :

$$(5.5) \quad y_{ti} = Z_{ti}\theta_i + W_{ti}\eta_i + t\beta_i + \epsilon_{ti}$$

where θ_i and η_i are the treatment effects with Z_{ti} and W_{ti} being indicator variables that is 1 when observation t is within a period where treatment was applied and 0 otherwise. $\epsilon_{ti} \sim N(0, \tau_i^2)$ is the error term at t .

On top of the individual patient model we now add the population-level parameters, which the patient-level parameters are drawn from.



Last we need to define prior distributions for the parameters defining the population distribution:

$$\begin{aligned}
\mu_\theta &\sim \mathcal{N}(10, 10) \\
\sigma_\theta &\sim \text{HalfCauchy}(0, 10) \\
\mu_\eta &\sim \mathcal{N}(10, 10) \\
\sigma_\eta &\sim \text{HalfCauchy}(0, 10) \\
\mu_\beta &\sim \mathcal{N}(0.01, 0.3) \\
\sigma_\beta &\sim \text{HalfCauchy}(0, 2) \\
\gamma_\tau &\sim \text{HalfCauchy}(0, 10)
\end{aligned}$$

5.4.2 Implementing the Model in PyMC3

As we are now already familiar with basics of PyMC3, we can now skip explaining some parts of the code.

There is no need to load the packages and the data as they have already been loaded in.

```

with pm.Model() as hierarchical_model:

    # population priors
    pop_treatment1_mean_prior = pm.Normal(
        "pop_treatment1_mean", mu=10, sigma=10
    )
    pop_treatment1_sd_prior = pm.HalfCauchy("pop_treatment1_sd", beta=10)

    pop_treatment2_mean_prior = pm.Normal(
        "pop_treatment2_mean", mu=10, sigma=10
    )
    pop_treatment2_sd_prior = pm.HalfCauchy("pop_treatment2_sd", beta=10)

    pop_trend_mean_prior = pm.Normal("pop_trend_mean", mu=0.1, sigma=0.3)
    pop_trend_sd_prior = pm.HalfCauchy("pop_trend_sd", beta=2)

    pop_measurement_error_beta_prior = pm.HalfCauchy(
        "pop_measurement_error_beta", beta=10
    )

```

```

# separate parameter for each patient
pat_treatment1 = pm.Normal(
    "treatment1",
    mu=pop_treatment1_mean_prior,
    sigma=pop_treatment1_sd_prior,
    shape=patients_n,
)
pat_treatment2 = pm.Normal(
    "treatment2",
    mu=pop_treatment2_mean_prior,
    sigma=pop_treatment2_sd_prior,
    shape=patients_n,
)
pat_sigma = pm.HalfCauchy(
    "sigma", beta=pop_measurement_error_beta_prior, shape=patients_n,
)
pat_trend = pm.Normal(
    "trend",
    mu=pop_trend_mean_prior,
    sigma=pop_trend_sd_prior,
    shape=patients_n,
)

measurement_means = (
    pat_treatment1[patient_idx] * measurements_df["treatment1"]
    + pat_treatment2[patient_idx] * measurements_df["treatment2"]
    + pat_trend[patient_idx] * measurements_df["measurement_index"]
)

likelihood = pm.Normal(
    "y",
    measurement_means,
    sigma=patients_sigma[patient_idx],
    observed=measurements_df["measurement"],
)

trace = pm.sample(800, tune=300, cores=3)

pm.traceplot(

```



```

        trace,
        [
            "treatment1",
            "treatment2",
            "sigma",
            "trend",
            "pop_treatment1_mean",
            "pop_treatment1_sd",
            "pop_treatment2_mean",
            "pop_treatment2_sd",
            "pop_measurement_error_beta",
            "pop_trend_mean",
            "pop_trend_sd",
        ],
    )
    plt.show()
    pm.summary(trace)

    pm.plot_posterior(trace)
    plt.show()

# posterior sampling
with hierarchical_model as model:
    post_pred = pm.sample_posterior_predictive(trace, samples=500)
    predictions = post_pred["y"]

draw_posterior_checks(
    predictions=predictions,
    measurements_df=measurements_df,
    parameters_df=parameters_df,
    plot_name="hierarchical_model_posterior_sampling",
)

```

Defining the Population Level Priors

```

with pm.Model() as hierarchical_model:

```

```

# population priors
pop_treatment1_mean_prior = pm.Normal(
    "pop_treatment1_mean", mu=10, sigma=10
)
pop_treatment1_sd_prior = pm.HalfCauchy("pop_treatment1_sd", beta=10)

pop_treatment2_mean_prior = pm.Normal(
    "pop_treatment2_mean", mu=10, sigma=10
)
pop_treatment2_sd_prior = pm.HalfCauchy("pop_treatment2_sd", beta=10)

pop_trend_mean_prior = pm.Normal("pop_trend_mean", mu=0.1, sigma=0.3)
pop_trend_sd_prior = pm.HalfCauchy("pop_trend_sd", beta=2)

pop_measurement_error_beta_prior = pm.HalfCauchy(
    "pop_measurement_error_beta", beta=10
)

```

Defining the Patient Level Parameters

```

pat_treatment1 = pm.Normal(
    "treatment1",
    mu=pop_treatment1_mean_prior,
    sigma=pop_treatment1_sd_prior,
    shape=patients_n,
)
pat_treatment2 = pm.Normal(
    "treatment2",
    mu=pop_treatment2_mean_prior,
    sigma=pop_treatment2_sd_prior,
    shape=patients_n,
)
pat_sigma = pm.HalfCauchy(
    "sigma", beta=pop_measurement_error_beta_prior, shape=patients_n,
)
pat_trend = pm.Normal(
    "trend",

```

```

mu=pop_trend_mean_prior,
sigma=pop_trend_sd_prior,
shape=patients_n,
)

```

Defining the Likelihood

```

measurement_means = (
    treatment1_prior[patient_idx] * measurements_df["treatment1"]
    + treatment2_prior[patient_idx] * measurements_df["treatment2"]
    + trend_prior[patient_idx] * measurements_df["measurement_idx"]
)

likelihood = pm.Normal(
    "y",
    measurement_means,
    sigma=patients_sigma[patient_idx],
    observed=measurements_df["measurement"],
)

```

Running the Model

As our model is now much more complex than with the single patient model with more parameters, we need to use more steps to get a reliable estimate.

Also the number of tuning steps needs to be increased.

```

trace = pm.sample(800, tune=300, cores=3)

```

Checking Algorithm Diagnostics

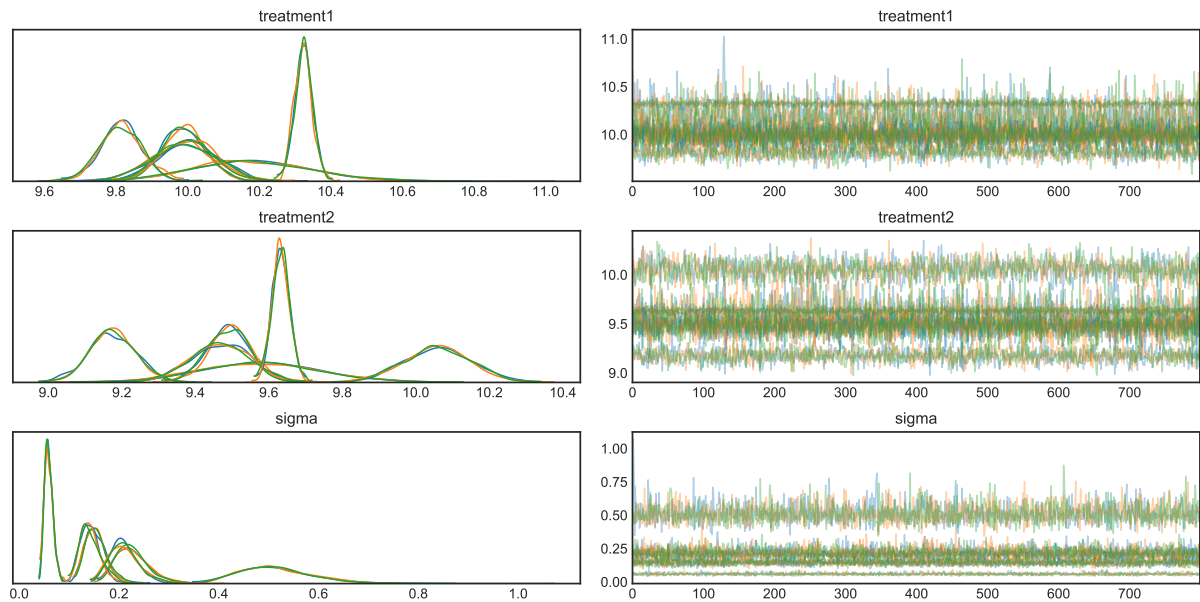
```
pm.traceplot(
    trace,
    [
        "treatment1",
        "treatment2",
        "sigma",
        "trend",
    ],
)
plt.show()

pm.traceplot(
    trace,
    [
        "pop_treatment1_mean",
        "pop_treatment1_sd",
        "pop_treatment2_mean",
        "pop_treatment2_sd",
        "pop_measurement_error_beta",
        "pop_trend_mean",
        "pop_trend_sd",
    ],
)
plt.show()

pm.summary(trace)
```

	mcse_mean	mcse_sd	ess_mean	ess_sd	ess_bulk	ess_tail	r_hat
treatment1	0.001	0.001	2712.0	2712.0	2724.0	1680.0	1.0
treatment2	0.001	0.001	2349.0	2348.0	2351.0	1830.0	1.0
gamma	0.001	0.001	2148.0	2140.0	2127.0	1703.0	1.0

Figure 5.6: Hierarchical Model Patient Level Traceplots



Checking the Posterior

```
pm.plot_posterior(trace)
plt.show()
```

Checking Model Fit with Posterior Sampling

```
# posterior sampling
with hierarchical_model as model:
    post_pred = pm.sample_posterior_predictive(trace, samples=500)
    predictions = post_pred["y"]

draw_posterior_checks(
    predictions=predictions,
    measurements_df=measurements_df,
    parameters_df=parameters_df,
    plot_name="hierarchical_model_posterior_sampling",
)
```

5.4.3 Final Results

Figure 5.7: Hierarchical Model Population Level Traceplots

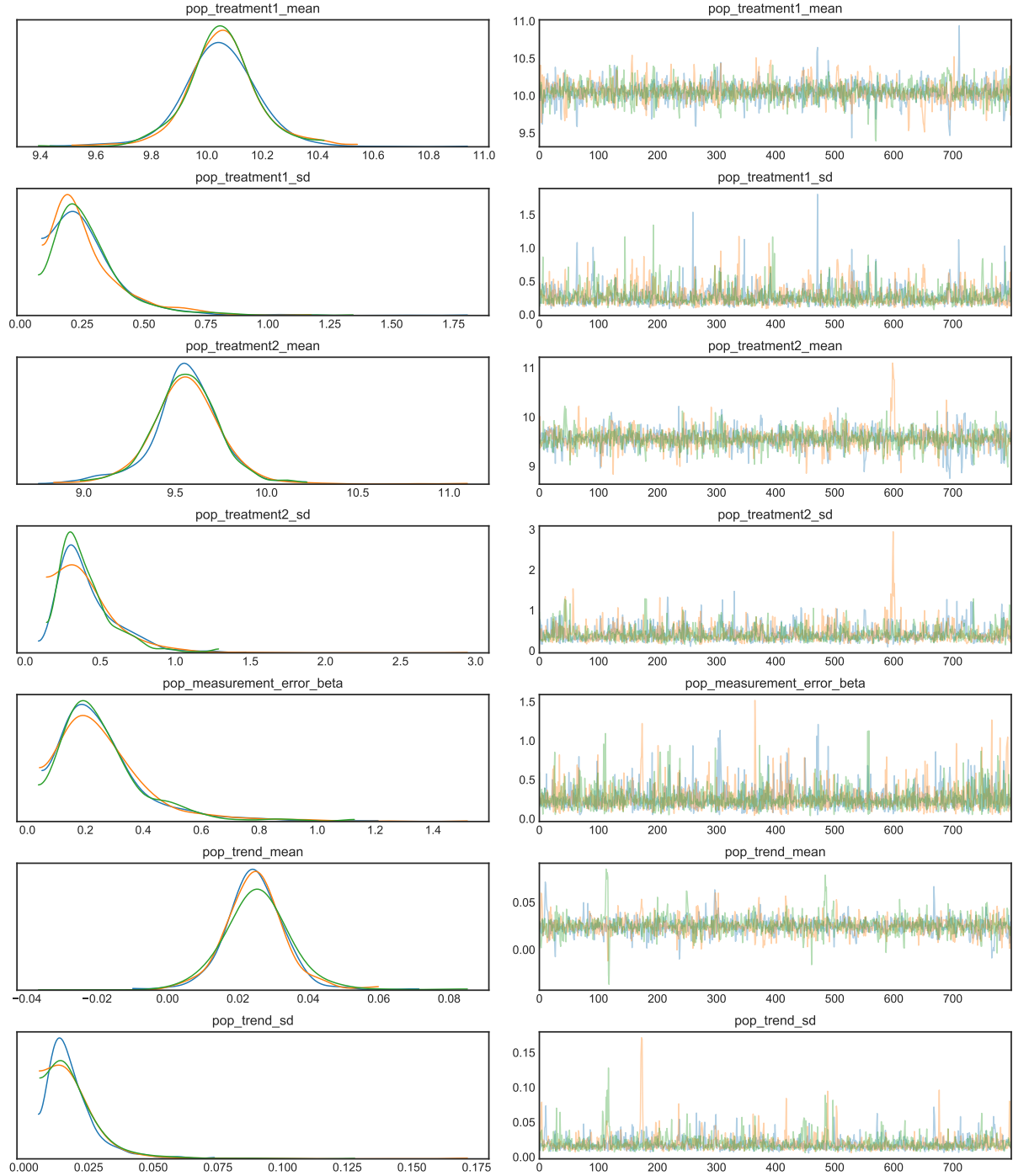
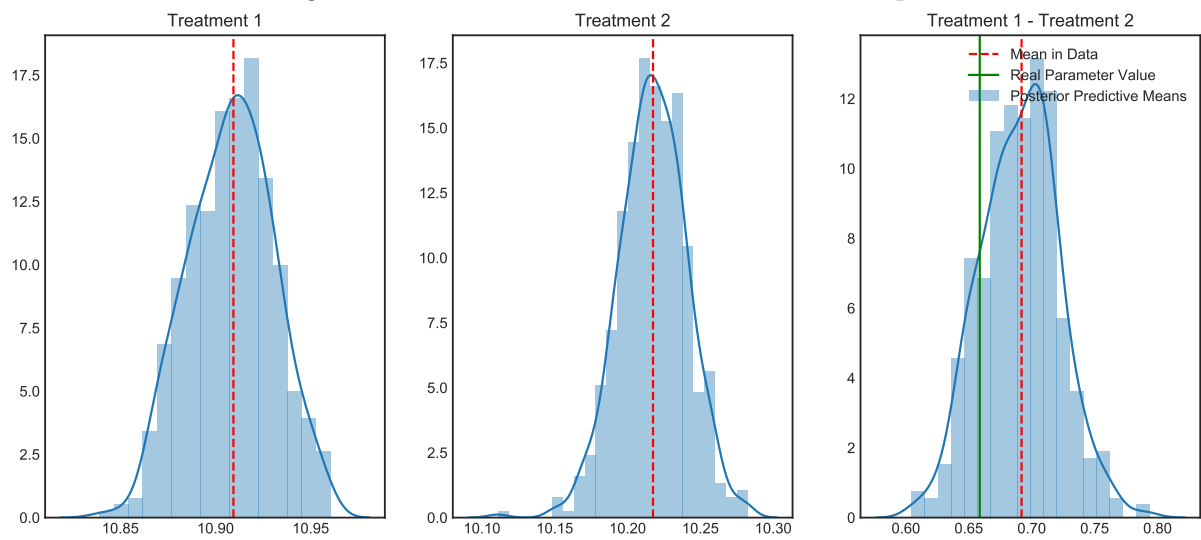


Figure 5.8: Hierarchical Model Posterior Sample



Chapter 6

Conclusion

We did not go over all the usual steps when building the models.

No model testing and now trial and error with the amount of steps.

Bibliography

- [1] Richard L. Kravitz, Naihua Duan, Sunita Vohra, Jiang Li: Introduction to to N-of-1 Trials: Indications and Barriers in Design and Implementation of N-of-1 Trials: A User’s Guide, AHRQ, 2014.
- [2] Wilson D. Pace, Elizabeth W. Staton, Eric B. Larson: Financing and Economics of Conducting N-of-1 Trials in Design and Implementation of N-of-1 Trials: A User’s Guide, AHRQ, 2014.
- [3] Christopher H. Schmid, Naihua Duan: Statistical Design and Analytic Considerations in Design and Implementation of N-of-1 Trials: A User’s Guide, AHRQ, 2014.
- [4] Christoper J. Gill, Lora Savin, Christopher H. Schmid: Why clinicians are natural Bayesians, British Medical Journal 2005;330(7499):1080-1083.
- [5] Colin D. Mathers, Dejan Loncar: Projections of global mortality and burden of disease from 2002 to 2030, PLoS medicine 2016;3.11:e442.
- [6] Irl B. Hirsch, et al.: Clinical application of emerging sensor technologies in diabetes management: consensus guidelines for continuous glucose monitoring (CGM). Diabetes Technology and Therapeutics 2018;10.4:232-246.
- [7] John K. Kruschke: Doing Bayesian Data Analysis: A Tutorial With R, JAGS, and Stan (2nd), Academic Press, 2014.
- [8] Andrew Gelman, John B. Carlin, Hal Stern, David Dunson, Aki Vehtari, Donald B. Rubin: Bayesian Data Analysis, CRC Press, 2013
- [9] Peter Rothwell: External Validity of Randomised Controlled Trials: “To Whom Do the Results of This Trial Apply?”, Lancet, Jan 1-7 2005;365(9453):82-93.15
- [10] Peter Rothwell: Treating individuals 2. Subgroup Analysis in Randomised Controlled Trials: Importance, Indications, and Interpretation, Lancet, Jan 8-14 2005;365(9454):176-186.16

- [11] Diane Warden, John Rush, Madhukar Trivedi, et al.: The STAR*D Project Results: A Comprehensive Review of Findings, *Current Psychiatry Reports*, Dec 2007;9(6):449-459
- [12] David Kent, Peter Rothwell, John Ioannidis, et al.: Assessing and Reporting Heterogeneity in Treatment Effects in Clinical Trials: A Proposal, *Trials*, 2010;11:85
- [13] John Salvatier, Thomas Wiecki, Christopher Fonnesbeck: Probabilistic Programming in Python Using PyMC3, *PeerJ Computer Science*, 2016, 2:e55
- [14] Tuomo Kareoja: <https://github.com/TuomoKareoja/hierarchical-bayes-nof1-thesis>, GitHub repository, 2020