bayesassurance: An R Package for Calculating Sample Size and Bayesian Assurance

by Jane Pan and Sudipto Banerjee

Abstract In this paper, we present bayesassurance, an R package designed for computing Bayesian assurance criteria which can be used to determine sample size in Bayesian inference setting. The functions included in the R package offer a two-stage framework using design priors to specify the population from which the data will be collected and analysis priors to fit a Bayesian model. We also demonstrate that frequentist sample size calculations are exactly reproduced as special cases of evaluating Bayesian assurance functions using appropriately specified priors.

1 Introduction

Power is an important feature of statistical tests that refers to the probability of correctly identifying the occurrence of an event given the event is actually present. Specifically, in the frequentist setting, the power of a test describes the probability that the test will correctly reject the null hypothesis, H_0, given that the alternative hypothesis, H_1, is true. Conversely, $\beta = 1 -$ power represents the probability that a test will fail to identify a true effect. For any given test, power is a function of the underlying effect size, the significance level, α, and the sample size, n . While the underlying effect size is rarely under the experimenter's control, the significance level and the sample size are. Hence, it is important to conduct power analysis to determine appropriate assignments of α and n for an experiment to have a good chance of detecting an effect if one exists. Power curves serve as a useful tool in quantifying the degree of assurance held towards meeting a study's analysis objective across a range of sample sizes. Formulating sample size determination as a decision problem so that power is an increasing function of sample size, offers the investigator a visual aid in helping deduce the minimum sample size needed to achieve a desired power.

The analogue of power in the Bayesian setting is *assurance*, which is based upon the probability of meeting a desired analysis objective. As an example, our analysis objective could be to ascertain if the difference between two population quantities, θ_1 and θ_2, exceeds a certain threshold θ_0, i.e., $\theta_1 - \theta_2 > \theta_0$. We decide that our analysis objective is met if the posterior probability of the above event given data we have observed exceeds ω, i.e., $P(\theta_1 - \theta_2 > \theta_0 \mid y) > \omega$ with y being the realized data. Assurance is defined as the probability of the analysis objective being met under an assumed distribution of the data, i.e., $\delta = P_y \{y : P(\theta > \theta_0 \mid y) > \omega\}$, where y denotes the observed data. Several publications adopt a similar approach, where they determine sample size based upon some criteria of analysis or model performance (Rahme et al., 2000; Gelfand and Wang, 2002; O'Hagan and Stevens, 2001). Other proposed solutions of the sample size problem introduce frameworks that prioritize conditions specific to the problem at hand, e.g. the use of Bayesian average errors to simultaneously control for Type I and Type II errors (Reyes and Ghosh, 2013), the use of posterior credible interval lengths to evaluate sample size estimates (Joseph et al., 1997), and the use of survival regression models to target Bayesian meta-experimental designs (Reyes and Ghosh, 2013).

There are several R packages for Bayesian sample size determination using specified analytic criteria. The **SampleSizeMeans** package contains a series of functions used for determining appropriate sample sizes based on various Bayesian criteria for estimating means or differences between means of normal variables (Joseph and Belisle, 2012). Criteria considered include the Average Length Criterion, the Average Coverage Criterion, and the Modified Worst Outcome Criterion (Joseph et al., 1995; Joseph and Belisle, 1997). A supplementary package, **SampleSizeProportions**, addresses study designs for estimation of binomial proportions using the same set of criteria (Joseph and Belisle, 2009). Our package, **bayesassurance** calculates Bayesian assurance and sample sizes for analysis objectives using normal and binomial models. We devise a two stage framework using possibly different sets of prior distributions in the design and analysis stages. The prior in the design stage represents a model that generates the data. The prior in the analysis stage represents the analyst's beliefs about the data. These two prior distributions are possibly different because the analyst does not usually have enough prior information on the processes generating the data. We primarily demonstrate sample size determination using conjugate Bayesian linear regression models as a prototype for this article, although the R package offers functions for calculating Bayesian assurance for binary or binomial models as well. In the current article, we focus on the flexibility of Bayesian linear regression models and demonstrate determination of unequal sample sizes for two samples and longitudinal study

Function	Type	Description
`pwr_freq`	closed-form solution	Computes the frequentist power of the specified hypothesis test (either one or two-sided z-tests).
`assurance_nd_na`	closed-form solution	Computes the exact Bayesian assurance of attaining a specified analysis objective.
`bayes_sim`	simulation	Approximates the Bayesian assurance of attaining a specified condition for a balanced study design through Monte Carlo sampling.
`bayes_sim_unbalanced`	simulation	Approximates the Bayesian assurance of attaining a specified condition for an unbalanced study design through Monte Carlo sampling.
`bayes_sim_unknownvar`	simulation	Similar to `bayes_sim` but approximates the assurance assuming unknown variance.
`bayes_adcock`	simulation	Approximates the probability (assurance) that the absolute difference between the true population parameter and the sample estimate falls within a margin of error no greater than a pre-specified precision level, d (Adcock, 1997).
`bayes_sim_betabin`	simulation	Approximates the probability (assurance) that there exists a difference between two independent proportions (Pham-Gia, 1997).
`bayes_goal_func`	simulation	Approximates the rate of correct classification using a utility-based approach within a linear hypothesis testing setting (Inoue et al., 2005).
`pwr_curve`	visual tool	Constructs a plot with the power and assurance curves overlayed on top of each other for comparison.
`gen_Xn`	design tool	Constructs design matrix using given sample size(s). Used for power and sample size analysis in the Bayesian setting.
`gen_Xn_longitudinal`	design tool	Constructs design matrix using inputs that correspond to a balanced longitudinal study design.

Table 1: Overview of the functions available for use within the package.

designs.

The **bayesassurance** package is available on CRAN (Pan and Banerjee, 2022) and contains a collection of functions that can be divided into three categories based on design and usage. These include closed-form solutions, simulation-based solutions, and visualization and/or design purposes. All available functions are presented in Table 1. Fully worked-out examples and tutorials can be found on our Github page at https://github.com/jpan928/bayesassurance_rpackage. This article describes the basic underlying framework leading to analytically tractable expressions for Bayesian assurance. We will also illustrate the relationship between Bayesian and frequentist sample size determination. In addition, we briefly explore simulation-based assurance methods, outlining the statistical distribution theory associated with each method, followed by examples worked out in R that users will be able to replicate. Finally, we offer some useful graphical features and design matrix generators offered by the package. In the following sections, we provide a detailed overview for each of the available functions grouped by category followed with worked out examples in R.

2 Closed-form Solution of Assurance

Bayesian assurance evaluates the tenability of attaining a specified outcome through the implementation of prior and posterior distributions. The `assurance_nd_na` function computes the exact assurance using a closed-form solution. Classical frequentist (Neyman-Pearson) inference proceeds as follows. Prior to an experiment, the analyst determines the sample size, as well as significance level (α), which defines the maximum frequency of false positives (type I errors) they are willing to tolerate, if the null hypothesis is true. Then, once the data has been collected, a statistical test is conducted which yields the p-value or the probability of seeing the observed data under the null hypothesis. If the p-value is lower than α, the null hypothesis is rejected. In contrast, in Bayesian inference refrains from "rejecting" or "failing to reject the null hypothesis". Instead, we look at the tenability of a hypothesis based upon realized data. As an example, suppose we seek to evaluate the tenability of $H : \theta > \theta_0$ given data from a Gaussian population with mean θ_0 and known variance σ^2. We assign two sets of priors for θ, one at the *design stage* and the other at the *analysis stage*. These two stages are the primary components that make up the skeleton of our generalized solution in the Bayesian setting and will be revisited in later sections. The analysis objective specifies the condition that needs to be satisfied. It defines a positive outcome, which serves as an overarching criteria that characterizes the study. Our analysis objective is to ascertain if $P(\theta > \theta_0 \mid \bar{y}) > 1 - \alpha$, where $\bar{y}$ is the data average and α is a specified threshold. Assuming the prior $\theta \sim N\left(\mu, \frac{\sigma^2}{n_a}\right)$, the posterior distribution of θ is

$$N\left(\theta \,\middle|\, \mu, \frac{\sigma^2}{n_a}\right) \times N\left(\bar{y} \,\middle|\, \theta, \frac{\sigma^2}{n}\right) \propto N\left(\theta \,\middle|\, \frac{n_a}{n+n_a}\mu + \frac{n}{n+n_a}\bar{y}, \frac{\sigma^2}{n+n_a}\right) , \tag{1}$$

where n denotes the sample size of the data, $\bar{y}$ denotes the mean of the data, and n_a is specified by the data analyst to quantify the prior degree of belief on θ.

The design objective is to find the sample size needed to ensure that the analysis objective is met $100\delta\%$ of the time, where δ denotes the assurance. At the design stage, we specify a model for the underlying population from which the data is generated. Our belief about this population is quantified using a *design prior* (we borrow this terminology from O'Hagan and Stevens, 2001), say $\theta \sim N\left(\mu, \frac{\sigma^2}{n_d}\right)$, where n_d is specified by the user to quantify the degree of belief (or amount of confidence) on the population parameters. Bayesian assurance is given by

$$\delta = P_{\bar{y}}\left\{\bar{y} : P(\theta > \theta_0 \mid \bar{y}) > 1 - \alpha\right\} , \tag{2}$$

where $P_{\bar{y}}(\cdot)$ denotes the marginal distribution of $\bar{y}$ obtained from the design priors

$$\int N\left(\theta \,\middle|\, \mu, \frac{\sigma^2}{n_d}\right) \times N\left(\bar{y} \,\middle|\, \theta, \frac{\sigma^2}{n}\right) d\theta = N\left(\bar{y} \,\middle|\, \mu, \left(\frac{1}{n} + \frac{1}{n_d}\right)\sigma^2\right) .$$

Applying this to (2) we obtain

$$\delta(\Delta, n) = \Phi\left(\sqrt{\frac{nn_d}{n+n_d}}\left[\frac{n+n_a}{n}\frac{\Delta}{\sigma} + Z_\alpha \frac{n+n_a}{n}\right]\right) , \tag{3}$$

where $\Phi(\cdot)$ is the standard normal CDF with Z_α being its α-th quantile and $\Delta = \mu - \theta_0$.

We remark that the above expression assumes that σ^2 is known. This is a customary assumption made about the population in sample size calculations and is usually based upon pilot studies or historic data. Nevertheless, in theory we can relax this assumption and assign a prior distribution to σ^2. Conjugate priors include the inverse-Gamma family of distributions and analogous formulas to (3) may be derived in terms of distribution functions of the (non-central) t-distribution. Nevertheless, we do not pursue this development here but discuss the case of unknown σ^2 later in the linear regression setting. We now describe the function to compute (3). Table 2 lists the set of parameters used in `assurance_nd_na`, with `alpha` taking a default value of 0.05.

The following code loads the **bayesassurance** package and assigns arbitrary parameters to `assurance_nd_na` prior to executing the function.

```
R> library(bayesassurance)

R> n <- seq(100, 250, 10)
R> n_a <- 10
R> n_d <- 10
R> theta_0 <- 0.15
R> theta_1 <- 0.25
```

`assurance_nd_na`: **Parameters**	
Variable	Description
`n`	sample size (either scalar or vector)
`n_a`	precision parameter within the analysis stage that quantifies the degree of belief carried towards parameter θ
`n_d`	precision parameter within the design stage that quantifies the degree of belief of the population from which we are generating samples from
`theta_0`	initial parameter value provided by the client
`theta_1`	prior mean of θ assigned in the analysis and design stage
`sigsq`	known variance
`alt`	specifies alternative test case, where `alt = "greater"` tests if $\theta_1 > \theta_0$, `alt = "less"` tests if $\theta_1 < \theta_0$, and `alt = "two.sided"` performs a two-sided test for $\theta_1 \neq \theta_0$. By default, `alt = "greater"`
`alpha`	significance level

Table 2: Parameter specifications needed to run `assurance_nd_na`.

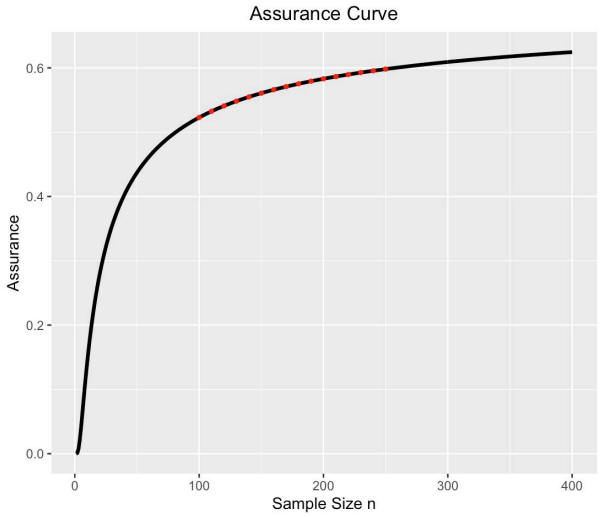


Figure 1: Resulting assurance plot with specific points passed in marked in red.

```
R> sigsq <- 0.30

R> out <- assurance_nd_na(n = n, n_a = n_a,  n_d = n_d,
        theta_0 = theta_0, theta_1 = theta_1, sigsq = sigsq,
        alt = "greater", alpha = 0.05)

R> head(out$assurance_table)
R> out$assurance_plot
```

```
     n Assurance
1  100 0.5228078
2  110 0.5324414
3  120 0.5408288
4  130 0.5482139
5  140 0.5547789
6  150 0.5606632
```

Running this block of code will return a table of assurance values and a graphical display of the assurance curve, shown in Figure 1. The first six rows of the table are reported in the outputs.

We make a few remarks pertaining to the functions in **bayesassurance**. First, we are passing a

single value of sample size or a vector of sample sizes for n. We save the results as a variable out. n can be either a scalar or vector. If n is a scalar, this tells the function that we only want to determine the assurance for one particular sample size. When this is the case, out will return a single assurance value with no plot. Alternatively, if a vector of sample sizes is passed in as n, as is the case of the example above, assurance is computed for a list of sample sizes, and the function will produce both a table and an assurance curve showing the results. As long as n is of length two or greater, assurance_nd_na will plot the assurance curve with red points denoting user-specified assurance values and black points denoting all other points outside those specified points. Figure 1 presents the assurance curve resulting from the example. The graph is created using **ggplot2** Wickham (2016) which is imported into **bayesassurance**. Simply typing out$assurance_table and out$assurance_plot will display the table and plot respectively in this particular set of examples.

2.1 Special case: convergence with the frequentist setting

Depending on how we define the parameters in assurance_nd_na, we can demonstrate a relationship between the Bayesian and classical power analysis. In particular, the classical, or frequentist, power function is a special case of Bayesian solution when letting $n_d \to \infty$ and $n_a = 0$ in (3). Therefore, assigning a weak analysis prior and a strong design prior yields

$$\Phi\Big(\sqrt{n}\frac{\Delta}{\sigma} + Z_\alpha\Big), \tag{4}$$

which is equivalent to the frequentist power expression that takes the form

$$1-\beta = P\Big(\bar{y} > \theta_0 + \frac{\sigma}{\sqrt{n}}Z_{1-\alpha}\Big) = \Phi\Big(\sqrt{n}\frac{\Delta}{\sigma} + Z_\alpha\Big).$$

The following code chunk demonstrates this special case in R using the assurance_nd_na function:

```
R> library(bayesassurance)

R> n <- seq(10, 250, 5)
R> n_a <- 1e-8
R> n_d <- 1e+8
R> theta_0 <- 0.15
R> theta_1 <- 0.25
R> sigsq <- 0.104

R> out <- assurance_nd_na(n = n, n_a = n_a,  n_d = n_d,
        theta_0 = theta_0, theta_1 = theta_1, sigsq = sigsq,
        alt = "greater", alpha = 0.05)

R> head(out$assurance_table)
R> out$assurance_plot
```

```
     n Assurance
1   10 0.2532578
2   15 0.3285602
3   20 0.3981637
4   25 0.4623880
5   30 0.5213579
6   35 0.5752063
```

The **bayesassurance** package includes a pwr_freq function that determines the statistical power of testing the difference between two means given a set of fixed parameter values that yield a closed-form solution of power and sample size. Continuing with the one-sided case, the solution is

$$1-\beta = P\left(\bar{y} > \theta_0 + \frac{\sigma}{\sqrt{n}}Z_{1-\alpha}\right) = \Phi\left(\sqrt{n}\frac{\Delta}{\sigma} + Z_\alpha\right), \tag{5}$$

where $\Delta = \theta_1 - \theta_0$ is the critical difference and Φ denotes the cumulative distribution function of the standard normal. Note this formula is equivalent to the special case of the assurance definition expressed in Equation (4). Table 3 includes the set of parameters needed to run this function.

pwr_freq: **Parameters**	
Variable	Description
n	sample size (either scalar or vector)
theta_0	value specified in the null hypothesis; to be provided by the user
theta_1	alternative value to test against the null value; this is the assumed effect size in the population
alt	specifies alternative test case, where alt = "greater" tests if $\theta_1 > \theta_0$, alt = "less" tests if $\theta_1 < \theta_0$, and alt = "two.sided" performs a two-sided test for $\theta_1 \neq \theta_0$. By default, alt = "greater"
sigsq	known variance
alpha	significance level

Table 3: Parameter specifications needed to run pwr_freq.

As a simple example, consider the following code segment that directly runs pwr_freq through specifying the above parameters and loading in **bayesassurance**:

```
R> library(bayesassurance)
R> pwr_freq(n = 20, theta_0 = 0.15, theta_1 = 0.35, sigsq = 0.30,
               alt = "greater", alpha = 0.05)
```

```
"Power: 0.495"
```

Running this returns the associated power printed as a statement rather than a table as we are only passing in one sample size, n = 20, to undergo evaluation. Now consider the next code example.

```
R> library(bayesassurance)
R> n <- seq(10, 250, 5)
R> out <- pwr_freq(n = n, theta_0 = 0.15, theta_1 = 0.25, sigsq = 0.104,
        alt = "greater", alpha = 0.05)

R> head(out$pwr_table)
R> out$pwr_plot
```

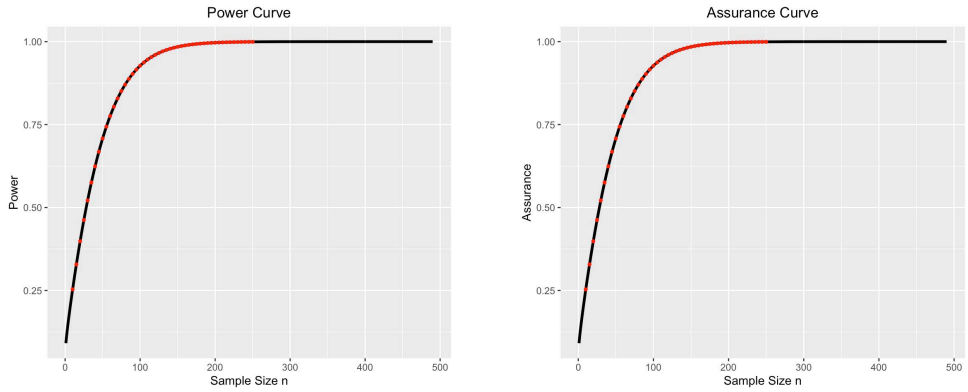
```
     n     Power
1   10 0.2532578
2   15 0.3285602
3   20 0.3981637
4   25 0.4623880
5   30 0.5213579
6   35 0.5752063
```

This code produces identical results as the assurance values obtained in the example using the assurance_nd_na function, where we assigned a weak analysis prior and a strong design prior. This demonstrates that under these conditions, Bayesian assurance converges to frequentist power. Figure 2 provides a side-by-side comparison of the resulting power and assurance curves, portraying identical plots in this particular setting.

3 Simulation-based functions using conjugate linear models

Henceforth, we focus on computing assurance through simulation-based means and highlight the common scenario of performing sample size analysis where closed-form solutions are unavailable. In the following sections, we extend upon the two-stage design structure and discuss how the design and analysis objectives are constructed based on the sample size criteria.

We seek to evaluate the tenability of a well-defined analysis objective using our simulation-based functions. The functions take an iterative approach alternating between generating a dataset in the



(a) Power curve **(b)** Assurance curve

Figure 2: Power and assurance curves are identical when the design priors are strong and the analysis priors are weak.

design stage and evaluating whether or not the dataset satisfies the analysis objective. The assurance equates to the proportion of datasets that meet the objective.

We start in the analysis stage. Consider a set of n observations denoted by $y = (y_1, y_2, \cdots, y_n)^\top$ that are to be collected together with p controlled explanatory variables, $x_1, x_2, \cdots, x_p$. Specifically,

$$y = X_n\beta + \epsilon_n,$$

where X_n is an $n \times p$ design matrix whose ith row $x_i^\top$, and $\epsilon_n \sim N(0, \sigma^2 V_n)$, where V_n is a known $n \times n$ correlation matrix. We assume X_n has linearly independent columns. A conjugate Bayesian linear regression model specifies the joint distribution of the parameters $\{\beta, \sigma^2\}$ and y as

$$IG(\sigma^2|a_\sigma, b_\sigma) \times N(\beta|\mu_\beta^{(a)}, \sigma^2 V_\beta^{(a)}) \times N(y|X_n\beta, \sigma^2 V_n),$$

where superscripts (a) indicate parameters in the analysis stage. The objective is to find the assurance of the realized data favoring $H : u^\top\beta > C$, where u is a $p \times 1$ vector of fixed contrasts and C is a known constant. Inference proceeds from the posterior distribution given by

$$p(\beta, \sigma^2|y) = IG(\sigma^2|a_\sigma^*, b_\sigma^*) \times N(\beta|M_n m_n, \sigma^2 M_n), \tag{6}$$

where

$$M_n^{-1} = V_\beta^{-1(a)} + X_n^\top V_n^{-1} X_n; \quad m_n = V_\beta^{-1(a)}\mu_\beta^{(a)} + X_n^\top V_n^{-1} y$$
$$a_\sigma^* = a_\sigma + \frac{n}{2}; \quad b_\sigma^* = b_\sigma + \frac{1}{2}\left\{\mu_\beta^\top V_\beta^{-1}\mu_\beta + y_n^\top V_n^{-1} y - m_n^\top M_n m_n\right\}.$$

The posterior distribution helps shape our analysis objective.

If σ^2 is known and fixed, then the posterior distribution of β is $p(\beta|\sigma^2, y) = N(\beta|M_n m_n, \sigma^2 M_n)$ shown in the Equation (6). We use the posterior components of β to evaluate $H : u^\top\beta > C$, where standardization leads to

$$\frac{u^\top\beta - u^\top M_n m_n}{\sigma\sqrt{u^\top M_n u}}\,\Big|\,\sigma^2, y \sim N(0,1)\,. \tag{7}$$

Hence, to assess the tenability of $H : u^\top\beta > C$, we decide in favor of H if the observed data belongs in the set

$$A_\alpha(u, \beta, C) = \left\{y : P\left(u^\top\beta \le C|y\right) < \alpha\right\} = \left\{y : \Phi\left(\frac{C - u^\top M_n m_n}{\sigma\sqrt{u^\top M_n u}}\right) < \alpha\right\}.$$

This defines our analysis objective, which we will monitor within each sample iteration. Sample generation is taken to account for in the design stage discussed in the next section.

In the design stage, the goal is to seek a sample size n such that the analysis objective is met at least $100\delta\%$ of the time, where δ is the assurance. This step requires determining the marginal distribution of y, which is assigned a separate set of priors to quantify our belief about the population from which

the sample will be taken. Hence, the marginal of y under the design priors will be derived from

$$y = X_n\beta + \epsilon_n; \quad \epsilon_n \sim N(0, \sigma^2 V_n)\ ; \quad \beta = \mu_\beta^{(d)} + \omega; \quad \omega \sim N(0, \sigma^2 V_\beta^{(d)})\ , \tag{8}$$

where $\beta \sim N(\mu_\beta^{(d)}, \sigma^2 V_\beta^{(d)})$ is the design prior on β and (d) denotes parameters in the design stage. Substituting the equation for β into the equation for y gives $y = X\mu_\beta^{(d)} + (X\omega + \epsilon_n)$ and, hence,

$$y \sim N\left(X\mu_\beta^{(d)}, \sigma^2 V_n^*\right) \quad \text{and} \quad V_n^* = \left(XV_\beta^{(d)}X^\top + V_n\right).$$

To summarize our simulation strategy for estimating the Bayesian assurance, we fix sample size n and generate a sequence of J datasets $y^{(1)}, y^{(2)}, ..., y^{(J)}$. A Monte Carlo estimate of the Bayesian assurance is computed as

$$\hat{\delta}(n) = \frac{1}{J}\sum_{j=1}^{J} \mathbb{I}\left(\left\{y^{(j)} : \Phi\left(\frac{C - u^\top M_n^{(j)} m_n^{(j)}}{\sigma\sqrt{u^\top M_n^{(j)} u}}\right) < \alpha\right\}\right)\ ,$$

where $\mathbb{I}(\cdot)$ is the indicator function of the event in its argument, $M_n^{(j)}$ and $m_n^{(j)}$ are the values of M_n and m_n computed from $y^{(j)}$.

3.1 Assurance computation with known variance

The simulation-based function, `bayes_sim`, determines the assurance within the context of conjugate Bayesian linear regression models assuming known variance, σ^2. The execution of `bayes_sim` is straightforward. An important feature is that users are not required to provide their own design matrix, X_n, when executing `bayes_sim`. When `Xn = NULL`, the function automatically constructs an appropriate design matrix based on the specified sample size(s), using the built-in `gen_Xn` function.

Later sections discuss design matrix generators in greater detail. Setting `Xn = NULL` facilitates calculation of assurances across a vector of sample sizes, where the function sequentially updates the design matrix for each unique sample size undergoing evaluation. Table 4 lists the required set of parameters.

Example 1: scalar parameter

This example computes the tenability of $H : u^\top\beta > C$ in the case when β is a scalar. In the following code block, we assign a set of values for the parameters of `bayes_sim` and saves the outputs as `assur_vals`. The first ten rows of the table is shown.

```
R> library(bayesassurance)
R> n <- seq(100, 300, 10)
R> assur_vals <- bayes_sim(n, p = 1, u = 1,
      C = 0.15, Xn = NULL, Vbeta_d = 0, Vbeta_a_inv = 0,
      Vn = NULL, sigsq = 0.265, mu_beta_d = 0.25, mu_beta_a = 0,
      alt = "greater", alpha = 0.05, mc_iter = 5000)

R> head(assur_vals$assurance_table)
R> assur_vals$assurance_plot
```

```
   Observations per Group (n) Assurance
1                        100    0.6162
2                        110    0.6612
3                        120    0.6886
4                        130    0.7148
5                        140    0.7390
6                        150    0.7746
```

Each unique value passed into `n` corresponds to a separate balanced study design containing that particular sample size for each of the p groups undergoing assessment. In this example, setting `p = 1`, `u = 1` and `C = 0.15` implies that we are evaluating the tenability of $H : \beta > 0.15$, where β is a scalar. Furthermore, `Vbeta_d` and `Vbeta_a_inv` are scalars to align with the dimension of β. A weak analysis prior (`Vbeta_a_inv = 0`) and a strong design prior (`Vbeta_d = 0`) produces the classical power analysis.

`bayes_sim`: **Parameters**	
Variable	Description
`n`	Sample size (either vector or scalar). If vector, each value corresponds to a separate study design.
`p`	Number of explanatory variables being considered. Also denotes the column dimension of design matrix `Xn`. If `Xn = NULL`, `p` must be specified for the function to assign a default design matrix for `Xn`.
`u`	a scalar or vector included in the expression to be evaluated, e.g. $u^\top \beta > C$, where β is an unknown parameter that is to be estimated.
`C`	constant to be compared to
`Xn`	design matrix characterizing the observations given by the normal linear regression model $y_n = X_n\beta + \epsilon_n$, where $\epsilon_n \sim N(0, \sigma^2 V_n)$. See above description for details. Default `Xn` is an $np \times p$ matrix comprised of $n \times 1$ ones vectors that run across the diagonal of the matrix.
`Vbeta_d`	correlation matrix that characterizes prior information on β in the design stage, i.e. $\beta \sim N(\mu_\beta^{(d)}, \sigma^2 V_\beta^{(d)})$.
`Vbeta_a_inv`	inverse-correlation matrix that characterizes prior information on β in the analysis stage, i.e. $\beta \sim N(\mu_\beta^{(a)}, \sigma^2 V_\beta^{(a)})$. The inverse is passed in for computation efficiency, i.e. $V_\beta^{-1(a)}$.
`Vn`	an $n \times n$ correlation matrix for the marginal distribution of the sample data y_n. Takes on an identity matrix when set to `NULL`.
`sigsq`	a known and fixed constant preceding all correlation matrices `Vn`, `Vbeta_d` and `Vbeta_a_inv`.
`mu_beta_d`	design stage mean, $\mu_\beta^{(d)}$
`mu_beta_a`	analysis stage mean, $\mu_\beta^{(a)}$
`alt`	specifies alternative test case, where `alt = "greater"` tests if $u^\top \beta > C$, `alt = "less"` tests if $u^\top \beta < C$, and `alt = "two.sided"` performs a two-sided test for $u^\top \beta \neq C$. By default, `alt = "greater"`.
`alpha`	significance level
`mc_iter`	number of MC samples evaluated under the analysis objective

Table 4: Parameter specifications needed to run `bayes_sim`.

We revisit this example in a later section when reviewing features that allow users to simultaneously visualize the Bayesian and frequentist power analyses. Finally, `Xn` and `Vn` are set to `NULL`, which means they will take on the default settings described above.

Example 2: linear contrasts

In this example, we assume β is a vector of unknown components rather than a scalar. We use the real-world example discussed in O'Hagan and Stevens (2001). Specifically, we consider a randomized clinical trial that compares the cost-effectiveness of two treatments. Cost-effectiveness is evaluated using a net monetary benefit measure expressed as

$$\xi = K(\mu_2 - \mu_1) - (\gamma_2 - \gamma_1),$$

where μ_1 and μ_2 denote the efficacy of treatments 1 and 2, and γ_1 and γ_2 denote the costs, respectively. Hence, $\mu_2 - \mu_1$ and $\gamma_2 - \gamma_1$ correspond to the true differences in treatment efficacy and costs. The threshold unit cost, K, represents the maximum price that a health care provider is willing to pay for a

unit increase in efficacy.

In this setting,we seek the tenability of $H : \xi > 0$, which, if true, indicates that treatment 2 is more cost-effective than treatment 1. To comply with the conjugate linear model framework outlined in (6), we set $u = (-K, 1, K, -1)^\top$, $\beta = (\mu_1, \gamma_1, \mu_2, \gamma_2)^\top$, and $C = 0$, giving us an equivalent form of $\xi > 0$ expressed as $u^\top \beta > 0$. All other inputs of this application were directly pulled from O'Hagan and Stevens (2001). The following code sets up the inputs to be passed into `bayes_sim`.

```
R> n <- 285
R> p <- 4
R> K <- 20000 # threshold unit cost
R> C <- 0
R> u <- as.matrix(c(-K, 1, K, -1))
R> sigsq <- 4.04^2

## Assign mean parameters to analysis and design stage priors
R> mu_beta_d <- as.matrix(c(5, 6000, 6.5, 7200))
R> mu_beta_a <- as.matrix(rep(0, p))

## Assign correlation matrices (specified in paper)
## to analysis and design stage priors
R> Vbeta_a_inv <- matrix(rep(0, p^2), nrow = p, ncol = p)
R> Vbeta_d <- (1 / sigsq) * matrix(c(4, 0, 3, 0, 0, 10^7, 0,
    0, 3, 0, 4, 0, 0, 0, 0, 10^7), nrow = 4, ncol = 4)

R> tau1 <- tau2 <- 8700
R> sig <- sqrt(sigsq)
R> Vn <- matrix(0, nrow = n*p, ncol = n*p)
R> Vn[1:n, 1:n] <- diag(n)
R> Vn[(2*n - (n-1)):(2*n), (2*n - (n-1)):(2*n)] <- (tau1 / sig)^2 * diag(n)
R> Vn[(3*n - (n-1)):(3*n), (3*n - (n-1)):(3*n)] <- diag(n)
R> Vn[(4*n - (n-1)):(4*n), (4*n - (n-1)):(4*n)] <- (tau2 / sig)^2 * diag(n)
```

The inputs specified above should result in an assurance of approximately 0.70 according to O'Hagan and Stevens (2001). The `bayes_sim` returns a similar value, demonstrating that sampling from the posterior yields results similar to those reported in the paper.

```
R> library(bayesassurance)

R> assur_vals <- bayes_sim(n = 285, p = 4,  u = as.matrix(c(-K, 1, K, -1)),
   C = 0, Xn = NULL, Vbeta_d = Vbeta_d,Vbeta_a_inv = Vbeta_a_inv,
   Vn = Vn, sigsq = 4.04^2, mu_beta_d = as.matrix(c(5, 6000, 6.5, 7200)),
   mu_beta_a = as.matrix(rep(0, p)),  alt = "greater", alpha = 0.05, mc_iter = 10000)

R> assur_vals


## [1] "Assurance: 0.722"
```

3.2 Assurance computation in the longitudinal setting

We demonstrate an additional feature embedded in the function tailored to longitudinal data. In this setting, n no longer refers to the number of subjects but rather the number of repeated measures reported for each subject, assuming a balanced study design. Referring back to the linear regression model discussed in the general framework, we can construct a longitudinal model that utilizes this same linear regression form, where $y_n = X_n\beta + \epsilon_n$.

Consider a group of subjects in a balanced longitudinal study with the same number of repeated measures at equally-spaced time points. In the base case, where time is treated as a linear term, subjects can be characterized as

$$y_{ij} = \alpha_i + \beta_i t_{ij} + \epsilon_i,$$

where y_{ij} denotes the j^{th} observation on subject i at time t_{ij}, α_i and β_i denote the intercept and slope terms for subject i, respectively, and $\epsilon_i \sim N(0, \sigma_i^2)$ is an error term.

In a simple case with two subjects, we can individually express the observations as

$$\begin{aligned} y_{11} &= \alpha_1 + \beta_1 t_{11} + \epsilon_1 \\ &\vdots \\ y_{1n} &= \alpha_1 + \beta_1 t_{1n} + \epsilon_1 \\ y_{21} &= \alpha_2 + \beta_2 t_{21} + \epsilon_2 \\ &\vdots \\ y_{2n} &= \alpha_2 + \beta_2 t_{2n} + \epsilon_2, \end{aligned}$$

assuming that each subject contains n observations. The model can also be expressed using matrices:

$$\underbrace{\begin{pmatrix} y_{11} \\ \vdots \\ y_{1n} \\ y_{21} \\ \vdots \\ y_{2n} \end{pmatrix}}_{y_n} = \underbrace{\begin{pmatrix} 1 & 0 & t_{11} & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 0 & t_{1n} & 0 \\ 0 & 1 & 0 & t_{21} \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 1 & 0 & t_{2n} \end{pmatrix}}_{X_n} \underbrace{\begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \beta_1 \\ \beta_2 \end{pmatrix}}_{\beta} + \underbrace{\begin{pmatrix} \epsilon_1 \\ \vdots \\ \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_2 \end{pmatrix}}_{\epsilon_n} \tag{9}$$

bringing us back to the linear model structure. If higher degrees are to be considered for the time variable, such as the inclusion of a quadratic term, the model would be altered to include additional covariate terms that can accommodate these changes. In the two-subject case, incorporating a quadratic term for the time variable in (9) will result in the model being modified as follows:

$$\underbrace{\begin{pmatrix} y_{11} \\ \vdots \\ y_{1n} \\ y_{21} \\ \vdots \\ y_{2n} \end{pmatrix}}_{y_n} = \underbrace{\begin{pmatrix} 1 & 0 & t_{11} & 0 & t_{11}^2 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 0 & t_{1n} & 0 & t_{1n}^2 & 0 \\ 0 & 1 & 0 & t_{21} & 0 & t_{21}^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 1 & 0 & t_{2n} & 0 & t_{2n}^2 \end{pmatrix}}_{X_n} \underbrace{\begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \beta_1 \\ \beta_2 \\ \phi_1 \\ \phi_2 \end{pmatrix}}_{\beta} + \underbrace{\begin{pmatrix} \epsilon_1 \\ \vdots \\ \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_2 \end{pmatrix}}_{\epsilon_n}$$

In general, for m subjects who each have n repeated measures, a one-unit increase in the degree of the time-based covariate will result in m additional columns being added to the design matrix X_n and m additional rows added to the β vector.

When working in the longitudinal setting, additional parameters need to be specified in the `bayes_sim` function, which can be found in Table 5. By default, `longitudinal = FALSE` and `ids`, `from`, and `to` are set to `NULL` when working within the standard conjugate linear model. When `longitudinal = TRUE`, `n` takes on a different meaning as its value(s) correspond to the number of repeated measures for each subject rather than the total number of subjects in each group. When `longitudinal = TRUE` and `Xn = NULL`, `bayes_sim` implicitly relies on a design matrix generator, `gen_Xn_longitudinal`, that is specific to the longitudinal setting to construct appropriate design matrices.

Example 3: longitudinal setting

The following example uses similar parameter settings as the cost-effectiveness example we had previously discussed in Example 2, now with longitudinal specifications. We assume two subjects and want to test whether the growth rate of subject 1 is different from that of subject 2. Figure 3 displays the estimated assurance points given the specifications.

Assigning an appropriate linear contrast lets us evaluate the tenability of an outcome. Let us consider the tenability of $u^\top \beta \neq C$ in this next example that uses simulated data, where $u = (1, -1, 1, -1)^\top$ and $C = 0$. There are 120 arbitrary timepoints. The number of repeated measurements per subject to be tested includes values 10 through 100 in increments of 5. This indicates that we are evaluating the assurance for 19 study designs in total. $n = 10$ divides the specified time interval into 10 evenly-spaced timepoints between 0 and 120.

For a more complicated study design comprised of more than two subjects that are divided into two treatment groups, consider testing if the mean growth rate is higher in the first treatment group

bayes_sim: **Additional Parameters used in the Longitudinal Setting**	
Variable	Description
longitudinal	logical that indicates the simulation will be based in a longitudinal setting. If Xn = NULL, the function will construct a design matrix using inputs that correspond to a balanced longitudinal study design.
ids	vector of unique subject ids.
from	start time of repeated measures for each subject
to	end time of repeated measures for each subject
num_repeated_measures	desired length of the repeated measures sequence.This should be a non-negative number, will be rounded up otherwise if fractional.
num_repeated_measures	desired length of the repeated measures sequence.This should be a non-negative number, will be rounded up otherwise if fractional.
poly_degree	degree of polynomial in longitudinal model, set to 1 by default.

Table 5: Additional parameter specifications needed to run bayes_sim in the longitudinal setting.

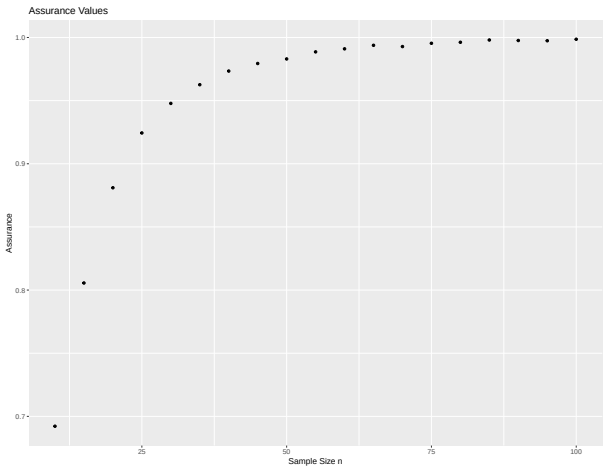


Figure 3: Estimated assurance points for longitudinal example.

than that of the second, e.g. if we have three subjects per treatment group, the linear contrast would be set as $u = (0,0,0,0,0,0,1/3,1/3,1/3,-1/3,-1/3,-1/3)^\top$.

```
R> n <- seq(10, 100, 5)
R> ids <- c(1,2)
R> Vbeta_a_inv <- matrix(rep(0, 16), nrow = 4, ncol = 4)
R> sigsq <- 100
R> Vbeta_d <- (1 / sigsq) * matrix(c(4, 0, 3, 0, 0, 6, 0, 0, 3, 0, 4, 0, 0, 0, 0, 6),
                                  nrow = 4, ncol = 4)

R> assur_out <- bayes_sim(n = n, p = NULL, u = c(1, -1, 1, -1), C = 0, Xn = NULL,
                          Vbeta_d = Vbeta_d, Vbeta_a_inv = Vbeta_a_inv,
                          Vn = NULL, sigsq = 100,
                          mu_beta_d = as.matrix(c(5, 6.5, 62, 84)),
                          mu_beta_a = as.matrix(rep(0, 4)), mc_iter = 5000,
                          alt = "two.sided", alpha = 0.05, longitudinal = TRUE, ids = ids,
                          from = 10, to = 120)

R> head(assur_out$assurance_table)
R> assur_out$assurance_plot

  Observations per Group (n) Assurance
1                         10    0.6922
```

bayes_sim_unbalanced: **Parameters for Unbalanced Study Designs**	
Variable	Description
n1	first sample size (either vector or scalar).
n2	second sample size (either vector or scalar).
repeats	an integer denoting the number of c(n1,n2) pairs we we are considering. For example, if repeats = 2, this means we will compute the assurance corresponding to the sample size set of c(n1,n2,n1,n2). By default, repeats = 1. See Example 5.
surface_plot	logical parameter that indicates whether a contour plot is to be constructed. When set to TRUE, and n1 and n2 are vectors, a contour plot (i.e. heat map) showcasing assurances obtained for all unique combinations of n1 and n2 is produced.

Table 6: Parameter specifications needed to run bayes_sim_unbalanced.

```
2                    15    0.8056
3                    20    0.8810
4                    25    0.9244
5                    30    0.9478
6                    35    0.9626
```

3.3 Assurance computation for unbalanced study designs

The bayes_sim_unbalanced function operates similarly to bayes_sim but estimates the assurance of attaining $u^\top \beta > C$ specifically in unbalanced design settings. Users provide two sets of sample sizes of equal length, whose corresponding pairs are considered for each study design case. The bayes_sim_unbalanced function provides a higher degree of flexibility for designing unbalanced studies and offers a more advanced visualization feature. Users have the option of viewing assurance as a 3-D contour plot and assess how the assurance behaves across varying combinations of the two sets of sample sizes that run along the x and y axes.

The bayes_sim_unbalanced function is similar to bayes_sim in terms of parameter specifications with a few exceptions. Parameters unique to bayes_sim_unbalanced are summarized in Table 6. Here, Xn = NULL, Vn = NULL, repeats = 1 and surface_plot = TRUE by default.

As in bayes_sim, it is recommended that users set Xn = NULL to facilitate the automatic construction of appropriate design matrices that best aligns with the conjugate linear model. Recall that every unique sample size (or sample size pair) passed in corresponds to a separate study that requires a separate design matrix. Should users choose to provide their own design matrix, it is advised that they evaluate the assurance for one study design at a time, in which a single design matrix is passed into Xn along with scalar values assigned for the sample size parameter(s).

Saved outputs from executing the function include

1. assurance_table: table of sample size and corresponding assurance values
2. contourplot: contour map of assurance values if surface.plot = TRUE
3. mc_samples: number of Monte Carlo samples that were generated for evaluation

Example 4: unbalanced assurance computation with surface plot

The following code provides a basic example of how bayes_sim_unbalanced is executed. It is important to check that the parameters passed in are appropriate in dimensions, e.g. mu_beta_a and mu_beta_d should each contain the same length as that of u, and the length of u should be equal to the row and column dimensions of Vbeta_d and Vbeta_a_inv.

A table of assurance values is printed simply by calling assur_out$assurance_table, which contains the exact assurance values corresponding to each sample size pair. The contour plot, shown in Figure 4, is displayed using assur_out$contourplot, and offers a visual depiction of how the assurance varies across unique combinations of n1 and n2. Areas with lighter shades denote higher assurance levels. The next example implements the function in a real-world setting that offers more sensible results.

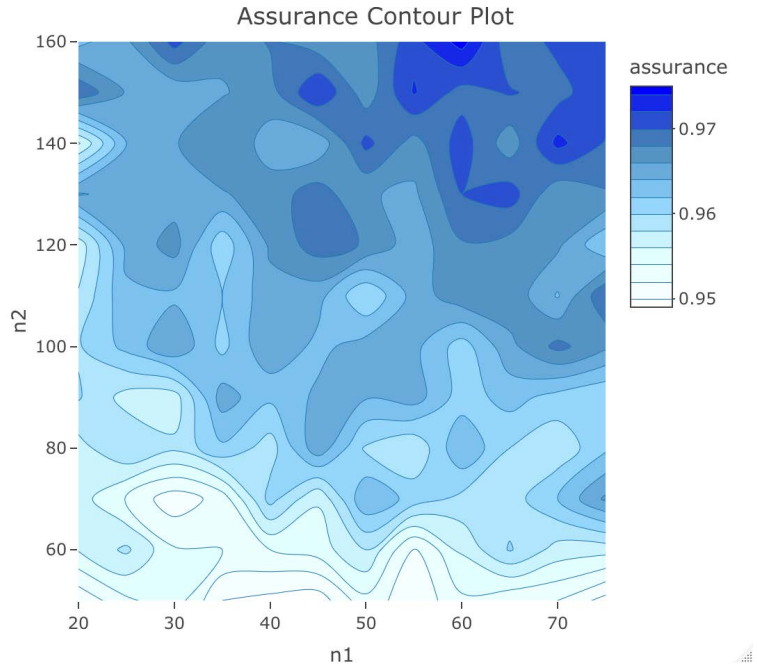


Figure 4: Contour map of assurance values with varying sample sizes `n1` and `n2`.

```
R> library(bayesassurance)

R> n1 <- seq(20, 75, 5)
R> n2 <- seq(50, 160, 10)


R> assur_out <- bayes_sim_unbalanced(n1 = n1, n2 = n2, repeats = 1, u = c(1, -1),
          C = 0, Xn = NULL, Vbeta_d = matrix(c(50, 0, 0, 10),nrow = 2, ncol = 2),
          Vbeta_a_inv = matrix(rep(0, 4), nrow = 2, ncol = 2),
          Vn = NULL, sigsq = 100,  mu_beta_d = c(1.17, 1.25),
          mu_beta_a = c(0, 0), alt = "two.sided", alpha = 0.05, mc_iter = 5000,
          surface_plot = TRUE)

R> head(assur_out$assurance_table)
R> assur_out$contourplot


  n1  n2 Assurance
1 20  50    0.9504
2 25  60    0.9584
3 30  70    0.9508
4 35  80    0.9616
5 40  90    0.9624
6 45 100    0.9634
```

Example 5: cost-effectiveness application

We revisit the cost-effectiveness problem discussed in Example 2. In addition to providing a 3-D graphical display of the assurance, this example also demonstrates how the `repeats` parameter is applied.

Recall from Example 2 that two distinct sets of efficacy and cost measures are used to compare the cost-effectiveness of treatments 1 and 2. The efficacy and costs are denoted by μ_i and γ_i for $i = 1, 2$ treatments. Hence, the parameter we want to estimate contains four elements tied to the unknown efficacy and costs of treatments 1 and 2, i.e. $\beta = (\mu_1, \gamma_1, \mu_2, \gamma_2)^\top$. It was previously assumed that the treatments contain an equal number of observations, suggesting that the sample sizes across each of the four explanatory variables are also equal. Using `bayes_sim_unbalanced` offers the added flexibility of constructing an unbalanced study design between treatments 1 and 2. Since the two treatments each

contain two components to be measured, we use the `repeats` parameter to indicate that we want two sets of sample sizes, `c(n1,n2)`, passed in, i.e. `c(n1,n2,n1,n2)`. It then becomes clear that our study design consists of n_1 observations for the efficacy and cost of treatment 1, and n_2 observations for those of treatment 2. Figure 5 displays a contour plot with a noticeable increasing trend of assurance values across larger sets of sample sizes.

```
R> library(bayesassurance)
R> n1 <- c(4, 5, 15, 25, 30, 100, 200)
R> n2 <- c(8, 10, 20, 40, 50, 200, 250)

R> mu_beta_d <- as.matrix(c(5, 6000, 6.5, 7200))
R> mu_beta_a <- as.matrix(rep(0, 4))
R> K = 20000 # threshold unit cost
R> C <- 0
R> u <- as.matrix(c(-K, 1, K, -1))
R> sigsq <- 4.04^2
R> Vbeta_a_inv <- matrix(rep(0, 16), nrow = 4, ncol = 4)
R> Vbeta_d <- (1 / sigsq) * matrix(c(4, 0, 3, 0, 0, 10^7, 0, 0,
3, 0, 4, 0, 0, 0, 0, 10^7),nrow = 4, ncol = 4)

R> assur_out <- bayes_sim_unbalanced(n1 = n1, n2 = n2, repeats = 2,
                u = as.matrix(c(-K, 1, K, -1)), C = 0, Xn = NULL,
                Vbeta_d = Vbeta_d, Vbeta_a_inv = Vbeta_a_inv,
                Vn = NULL, sigsq = 4.04^2,
                mu_beta_d = as.matrix(c(5, 6000, 6.5, 7200)),
                mu_beta_a = as.matrix(rep(0, 4)),
                alt = "greater", alpha = 0.05, mc_iter = 5000,
                surface_plot = TRUE)

R> assur_out$assurance_table
R> assur_out$contourplot

   n1  n2 Assurance
1   4   8    0.1614
2   5  10    0.1724
3  15  20    0.3162
4  25  40    0.3942
5  30  50    0.4440
6 100 200    0.6184
7 200 250    0.7022
```

4 Bayesian assurance using other conditions

The **bayesassurance** R package contains several other assurance functions characterized by analysis stage objectives that are dependent on fixed precision levels (Adcock, 1997) and posterior credible intervals (Pham-Gia, 1997). These functions are denoted respectively as `bayes_adcock` and `bayes_sim_betabin`. The package also includes a `bayes_goal_func` function framed under a utility-based setting (see, e.g., Raiffa and Schlaifer, 1961; Berger, 1985; Lindley, 1997; Müller and Parmigiani, 1995; Parmigiani, 2002; Inoue et al., 2005) that determines sample size in relation to the rate of correct classification (Inoue et al., 2005). Since the simulation-based assurance functions all follow a similar format, for the sake of brevity, we will not include detailed descriptions of them in this article. Vignettes outlining detailed descriptions and walkthrough tutorials can be found on our Github page (https://github.com/jpan928/bayesassurance_rpackage), which contains examples that users can easily follow along and reproduce on their own machines.

5 Visualization Features and Useful Tools

5.1 Overlapping power and assurance curves

To facilitate an understanding of the relationship held between Bayesian and frequentist power analysis, the `pwr_curves` function produces a single plot displaying both power and assurance points.

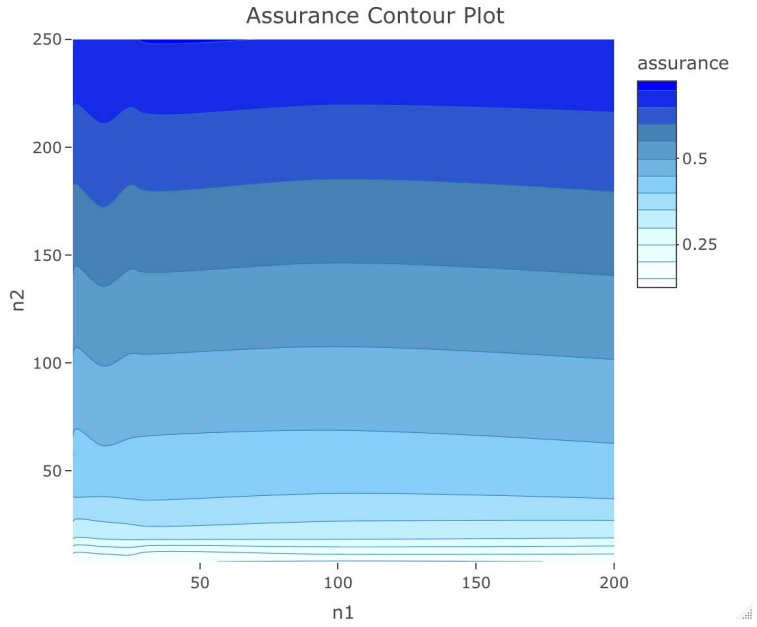


Figure 5: Contour map of assurance values in cost-effectiveness application.

Recall that the primary difference held between `pwr_freq` and `assurance_nd_na` is the need to specify additional precision parameters, n_a and n_d, in `assurance_nd_na`. Knowing that power and sample size analysis in the frequentist setting is essentially a special case of the Bayesian assurance with precision parameters tailored to weak analysis priors and strong design priors, the `pwr_curves` function serves as a visualization tool in seeing how varying precision levels affect assurance values and how these assurance values compare to those we would expect under classical/frequentist power analysis (strong design priors, weak analysis priors).

The `pwr_curves` function takes the combined set of parameters presented in `pwr_freq` and `assurance_nd_na`, which includes `n`, `n_a`, `n_d`, `theta_0`, `theta_1`, `sigsq`, and `alpha`. For further customization, users have the option to include a third set of points in their plot along with the power and assurance curves. These additional points would correspond to the simulated assurance results obtained using `bayes_sim`. Optional parameters to implement this include

1. `bayes_sim`: logical that indicates whether the user wishes to include simulated assurance results obtained from `bayes_sim`. Default setting is `FALSE`.
2. `mc_iter`: specifies the number of MC samples to evaluate given `bayes_sim = TRUE`.

The following code segment runs the `pwr_curves` function using a weak analysis stage prior (`n_a` is set to be small) and a strong design stage prior (`n_d` is set to be large). Implementing this produces a plot where the assurance points lay perfectly on top of the power curve as shown in Figure 6. The simulated assurance points obtained from `bayes_sim` are also plotted as we set `bayes_sim = TRUE`. These points are highlighted in blue, which lie very close in proximity to those of the exact assurance points highlighted in red. We can also view individual tables of the three sets of points by directly calling them from the saved outputs, e.g. `out$power_table` shows the individual frequentist power values for each sample size. The output we provide shows the first ten rows.

```
R> library(bayesassurance)

R> out <- pwr_curve(n = seq(10, 200, 10), n_a = 1e-8, n_d = 1e+8,
        sigsq = 0.104, theta_0 = 0.15,theta_1 = 0.25, alt = "greater", alpha = 0.05,
        bayes_sim = TRUE, mc_iter = 5000)

R> head(out$power_table)
R> head(out$assurance_table)
R> out$plot


     n     Power
1   10 0.2532578
2   20 0.3981637
```

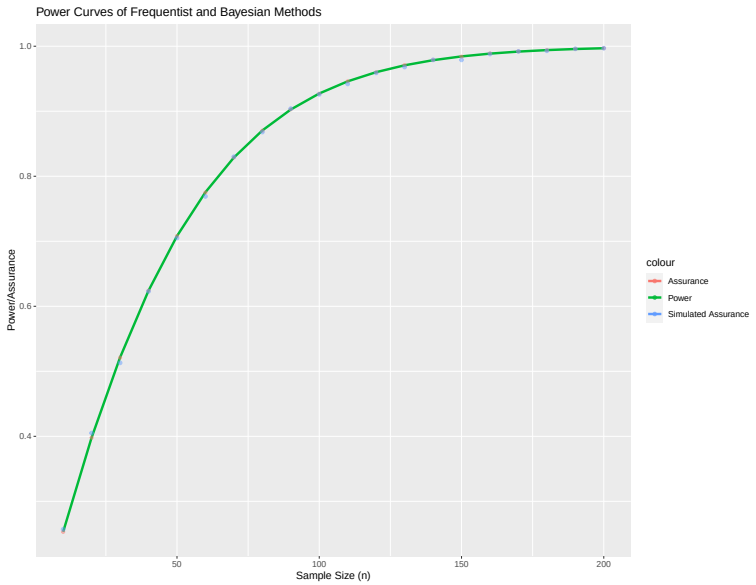


Figure 6: Power curve with exact and simulated assurance points for weak analysis prior and strong design prior.

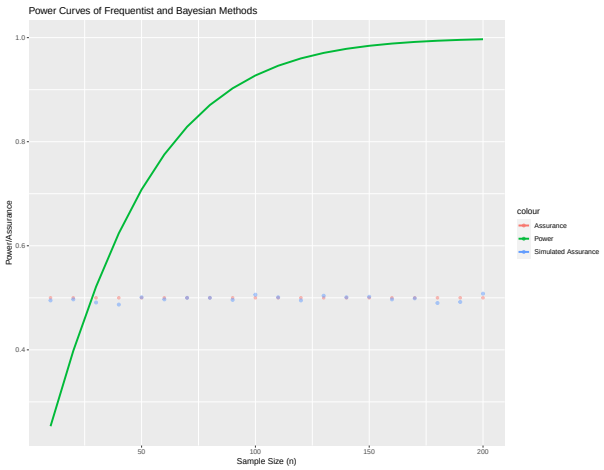


Figure 7: Power curve with exact and simulated assurance points for weak analysis and design priors.

```
3   30 0.5213579
4   40 0.6241155
5   50 0.7080824
6   60 0.7754956
```

```
     n Assurance
1   10 0.2532578
2   20 0.3981637
3   30 0.5213579
4   40 0.6241155
5   50 0.7080824
6   60 0.7754956
```

The next code segment considers the scenario in which both analysis and design stage priors are weak (`n_a` and `n_d` are set to be small). This special case shows how the assurance behaves when vague priors are assigned. Substituting 0 in for both n_a and n_d in Equation (3) results in a constant assurance of $\Phi(0) = 0.5$ regardless of the sample size and critical difference. Figure 7 illustrates these results, where we have the regular power curve and the flat set of assurance points at 0.5 for both exact and simulated cases.

```
R> library(bayesassurance)
```

```
R> pwr_curve(n = seq(10, 200, 10), n_a = 1e-8, n_d = 1e-8,
sigsq = 0.104, theta_0 = 0.15,theta_1 = 0.25, alt = "greater", alpha = 0.05,
bayes_sim = TRUE, mc_iter = 5000)
```

5.2 Design matrix generators

In the last few sections, we go over design matrix generators that are used inside functions within the **bayesassurance** package when the `Xn` parameter is set to `NULL`. We include these functions in case users wish to see how design matrices are constructed under this particular setting.

Standard design matrix generator

The standard design matrix generator, `gen_Xn`, is relevant to a majority of the simulation-based assurance functions discussed throughout the paper. It should be noted that all simulation-based functions available in this package do not require users to specify their own design matrix X_n. Users have the option of leaving `Xn = NULL`, which prompts the function to construct a default design matrix using `gen_Xn` that complies with the general linear model $y_n = X_n\beta + \epsilon, \epsilon \sim N(0, \sigma^2 V_n)$. The function runs in the background while `bayes_sim` is used.

When called directly, the `gen_Xn` function takes in a single parameter, `n`, which can either be a scalar or vector. The length of `n` corresponds to the number of groups being assessed in the study design as well as the column dimension of the design matrix, denoted as `p`. Therefore, in general, the resulting design matrix is of dimension $n \times p$. If a scalar value is specified for `n`, the resulting design matrix carries a dimension of $n \times 1$.

In the following example, we pass in a vector of length $p = 4$, which outputs a design matrix of column dimension 4. Each column is comprised of ones vectors with lengths that align with the sample sizes passed in for `n`. The row dimension is therefore the sum of all the entries in `n`. In this case, since the values 1, 3, 5, and 8 are being passed in to `n`, the design matrix to be constructed carries a row dimension of $1 + 3 + 5 + 8 = 17$ and a column dimension of 4.

```
R> n <- c(1,3,5,8)
R> gen_Xn(n = n)
```

```
      [,1] [,2] [,3] [,4]
 [1,]    1    0    0    0
 [2,]    0    1    0    0
 [3,]    0    1    0    0
 [4,]    0    1    0    0
 [5,]    0    0    1    0
 [6,]    0    0    1    0
 [7,]    0    0    1    0
 [8,]    0    0    1    0
 [9,]    0    0    1    0
[10,]    0    0    0    1
[11,]    0    0    0    1
[12,]    0    0    0    1
[13,]    0    0    0    1
[14,]    0    0    0    1
[15,]    0    0    0    1
[16,]    0    0    0    1
[17,]    0    0    0    1
```

The `bayes_sim` function and its related family of functions generate design matrices using `gen_Xn` in the following way. Each unique value contained in `n` that is passed into `bayes_sim` corresponds to a distinct study design and thus requires a distinct design matrix. The `gen_Xn` function interprets each i^{th} component of `n` as a separate balanced study design comprised of n_i participants within each of the p groups, where `p` is a parameter specified in `bayes_sim`. For example, if we let `Xn = NULL` and pass in `n <-2`, `p <-4` for `bayes_sim`, `gen_Xn` will process the vector `n <-c(2,2,2,2)` in the background. Hence,

we'd obtain an 8×4 matrix of the form

$$X_n = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Design matrix generator in longitudinal setting

Example 3 demonstrates how the linear model is extended to incorporate time-based covariates within the context of a longitudinal setting. For this special case, a separate function is used to generate design matrices that are appropriate for this setting. The `genXn_longitudinal` constructs its design matrices differently than `gen_Xn` and therefore requires a different set of parameter specifications. When the `longitudinal` parameter is set to `TRUE` in `bayes_sim`, the user is required to specify the following set of parameters, which are directly passed into `genXn_longitudinal`:

1. `ids`: vector of unique subject ids, usually of length 2 for study design purposes
2. `from`: start time of repeated measures for each subject
3. `to`: end time of repeated measures for each subject
4. `num_repeated_measures`: desired length of the repeated measures sequence. Should be a non-negative number, will be rounded up if fractional.
5. `poly_degree`: degree of polynomial in longitudinal model, set to 1 by default.

Referring back to the model that was constructed for the case involving two subjects, we observe in Equation (9) that the design matrix contains vectors of ones within the first half of its column dimension and lists the timepoints for each subject in the second half. Constructing this design matrix requires several components. The user needs to specify subject IDs that are capable of uniquely identifying each individual in the study. Next, the user needs to specify the start and end time as well as the number of repeated measures reported for each subject. The number of repeated measures denotes the number of evenly-spaced timepoints that take place in between the start and end time. Since we are assuming a balanced longitudinal study design, each subject considers the same set of timepoints. Finally, if the user wishes to consider time covariates of higher degrees, such as a quadratic or cubic function, this can be altered using the `poly_degree` parameter, which takes on a default assignment of 1.

In the following code, we pass in a vector of subject IDs and specify the start and end timepoints along with the desired length of the sequence. The resulting design matrix contains vectors of ones with lengths that correspond to the number of repeated measures for each unique subject.

```
R> ids <- c(1,2,3,4)
R> gen_Xn_longitudinal(ids, from = 1, to = 10, num_repeated_measures = 4)

      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
 [1,]    1    0    0    0    1    0    0    0
 [2,]    1    0    0    0    4    0    0    0
 [3,]    1    0    0    0    7    0    0    0
 [4,]    1    0    0    0   10    0    0    0
 [5,]    0    1    0    0    0    1    0    0
 [6,]    0    1    0    0    0    4    0    0
 [7,]    0    1    0    0    0    7    0    0
 [8,]    0    1    0    0    0   10    0    0
 [9,]    0    0    1    0    0    0    1    0
[10,]    0    0    1    0    0    0    4    0
[11,]    0    0    1    0    0    0    7    0
[12,]    0    0    1    0    0    0   10    0
[13,]    0    0    0    1    0    0    0    1
[14,]    0    0    0    1    0    0    0    4
[15,]    0    0    0    1    0    0    0    7
[16,]    0    0    0    1    0    0    0   10
```

The next code block modifies the previous example to incorporate a quadratic term. Notice there are four additional columns being aggregated to the design matrix. These four columns are obtained from squaring the four columns of the linear term.

```
R> ids <- c(1,2,3,4)
R> gen_Xn_longitudinal(ids, from = 1, to = 10, num_repeated_measures = 4,
poly_degree = 2)

      1 2 3 4  1  2  3  4   1   2   3   4
 [1,] 1 0 0 0  1  0  0  0   1   0   0   0
 [2,] 1 0 0 0  4  0  0  0  16   0   0   0
 [3,] 1 0 0 0  7  0  0  0  49   0   0   0
 [4,] 1 0 0 0 10  0  0  0 100   0   0   0
 [5,] 0 1 0 0  0  1  0  0   0   1   0   0
 [6,] 0 1 0 0  0  4  0  0   0  16   0   0
 [7,] 0 1 0 0  0  7  0  0   0  49   0   0
 [8,] 0 1 0 0  0 10  0  0   0 100   0   0
 [9,] 0 0 1 0  0  0  1  0   0   0   1   0
[10,] 0 0 1 0  0  0  4  0   0   0  16   0
[11,] 0 0 1 0  0  0  7  0   0   0  49   0
[12,] 0 0 1 0  0  0 10  0   0   0 100   0
[13,] 0 0 0 1  0  0  0  1   0   0   0   1
[14,] 0 0 0 1  0  0  0  4   0   0   0  16
[15,] 0 0 0 1  0  0  0  7   0   0   0  49
[16,] 0 0 0 1  0  0  0 10   0   0   0 100
```

6 Discussion

This article introduced **bayesassurance**, a new R package for computing Bayesian assurance under various conditions using a two-stage framework. The goal of this package is to provide a convenient and user-friendly interface to statisticians and data scientists who seek sample size calculations using the assurance function in Bayesian data analysis. We have attempted to provide an organized, well-documented open-source code that can be used to address a wide range of study design problems, such as in the case of clinical trials, and demonstrate the feasibility of applying Bayesian methods to such problems.

References

C. Adcock. Sample size determination: A review. *The Statistician*, 46(2), 1997. [p139, 152]

J. O. Berger. *Statistical Decision Theory and Bayesian Analysis*. Springer New York, New York, NY, 1985. [p152]

A. E. Gelfand and F. Wang. A simulation based approach to bayesian sample size determination for performance under a given model and for separating models. *Statistical Science*, 17(2), 2002. [p138]

L. Y. Inoue, D. A. Berry, and G. Parmigiani. Relationship between bayesian and frequentist sample size determination. *The American Statistician*, 59(1), 2005. [p139, 152]

L. Joseph and P. Belisle. Bayesian sample size determination for normal means and differences between normal means. *The Statistician*, 46(2), 1997. [p138]

L. Joseph and P. Belisle. *Package SampleSizeProportions*, 2009. URL https://cran.r-project.org/web/packages/SampleSizeProportions/SampleSizeProportions.pdf. R package version 1.0. [p138]

L. Joseph and P. Belisle. *Package SampleSizeMeans*, 2012. URL https://cran.r-project.org/web/packages/SampleSizeMeans/SampleSizeMeans.pdf. R package version 1.1. [p138]

L. Joseph, D. B. Wolfson, and R. du Berger. Sample size calculations for binomial proportions via highest posterior density intervals. *The Statistician*, 44(2), 1995. [p138]

L. Joseph, R. D. Berger, and P. Belisle. Bayesian and mixed bayesian/likelihood criteria for sample size determination. *Statistics in Medicine*, 16(7), 1997. [p138]

D. V. Lindley. The choice of sample size. *The Statistician*, 46(2), 1997. [p152]

P. Müller and G. Parmigiani. Optimal design via curve fitting of monte carlo experiments. *Journal of the American Statistical Association*, 90(432), 1995. [p152]

A. O'Hagan and J. W. Stevens. Bayesian assessment of sample size for clinical trials of cost-effectiveness. *Medical Decision Making*, 21(3), 2001. [p138, 140, 146, 147]

J. Pan and S. Banerjee. *Package bayesassurance*, 2022. URL https://cran.r-project.org/web/packages/bayesassurance/bayesassurance.pdf. R package version 0.1.0. [p139]

G. Parmigiani. *Modeling in Medical Decision Making: A Bayesian Approach*. Wiley, Hoboken, NJ, 2002. [p152]

T. Pham-Gia. On bayesian analysis, bayesian decision theory and the sample size problem. *The Statistician*, 46(2), 1997. [p139, 152]

E. Rahme, Lawrence, and T. W. Gyorkos. Bayesian sample size determination for estimating binomial parameters from data subject to misclassification. *Journal of Royal Statistical Society*, 49(1), 2000. [p138]

H. Raiffa and R. Schlaifer. *Applied Statistical Decision Theory*. Harvard University Graduate School of Business Administration (Division of Research), Massachusetts, 1961. [p152]

E. M. Reyes and S. K. Ghosh. Bayesian average error based approach to sample size calculations for hypothesis testing. *Biopharm*, 23(3), 2013. [p138]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL https://ggplot2.tidyverse.org. [p142]

Jane Pan
University of California, Los Angeles
650 Charles E Young Dr S, Los Angeles, CA 90095
USA
jpan1@ucla.edu

Sudipto Banerjee
University of California, Los Angeles
650 Charles E Young Dr S, Los Angeles, CA 90095
USA
sudipto@ucla.edu