Expressing Uncertainty with Probability: practical exercises

Christopher Jackson

This PDF document contains the practical exercises for Session 1 of the Bayesian Statistics course, "Expressing Uncertainty with Probability".

How to use the material

Full worked solutions to the exercises are provided at the back of the document. Feel free to consult these as you are going along, if you are stuck.

The document and embedded code are also provided as an R Markdown document in `intro_practical.Rmd`. This allows the R code blocks to be run conveniently from RStudio.

Warning: If you copy-and-paste code from the practicals into your own documents, then this should be copied from the R Markdown document rather from the PDF, otherwise some special characters will fail to copy properly.

Software prerequisites

- Practical 1 will be entirely done using base R commands.
- Practical 2 will need an extra R package, and installation instructions will be given.
- Practical 3 in this lecture, and most of the practicals in the rest of the course, will need the JAGS software to be installed from `https://sourceforge.net/projects/mcmc-jags/`. The `rjags` R package will also be needed.

Note on R usage

Many of these questions use standard functions in R for dealing with probability distributions. These functions have a common naming convention, with names beginning with `d`, `p`, `q` or `r`. See the help pages of these functions for more information about how to use them (e.g. `help(dnorm)`).

- `d`: the probability density function (e.g. `dnorm`, `dgamma`, `dbeta`, for the Normal, Gamma and Beta distributions)
- `p`: the cumulative distribution function (e.g. `pnorm`, `pgamma`, `pbeta`), which returns the probability that a random variable X is less than a particular value.
- `q`: the quantile function (e.g. `qnorm`, `qgamma`, `qbeta`), which returns the value c of X for which $P(X < c)$ equals a particular probability.
- `r`: random sampling functions (e.g. `rnorm`, `rgamma`, `rbeta`).

1 Practical session (1): probability judgements

Learning objectives

- Properties of probability distributions (mean, quantiles, density, etc.) and their implementation in R
- Define a Beta distribution for an uncertain quantity given elicited judgements

1. In the slide named "Heuristics to obtain Beta distributions from beliefs" it was stated that a 95% credible interval for a normal distribution had a width of 4σ. The number 4 stated here is actually an approximation to one significant figure. Use the function `qnorm` in R to obtain a more precise value for this constant, assuming an equal-tailed credible interval. Save this as an R object to use in part (2).
2. You guess that the chance of surviving a disease is around 10%, and you are 95% certain that it is between 2% and 20%. Obtain a Beta distribution which roughly represents this belief. Check that it is appropriate by obtaining the actual mean, median and 95% (equal-tailed) credible interval of this distribution.
3. Suppose instead that you are only 50% certain that the survival probability is between 2% and 20%. Obtain the Beta distribution in this case using a similar technique, and check the appropriate credible interval of this distribution against your original judgement.
4. [Visualisation] Plot the density of the distributions obtained in parts (1) and (2) in R, using the function `dbeta`. The `ppoints` function may be helpful for creating a grid of points between zero and 1. What is the most striking difference between the shapes of these distributions?
5. [R programming] Write an R function to obtain the parameters of the Beta distribution for a given guess at the mean, credible interval and the probability amount covered by this interval, and check that it reproduces the parameters obtained in the previous examples.

2 Practical session (2): predictions and decisions

Learning objectives

- Predicting an outcome from a Binomial where the probability is uncertain and summarising the uncertainty in R
- Basic Monte Carlo simulation from a probability distribution. How many samples to run
- Simple decision-making under uncertainty, by combining two uncertain sources of information
- Predicting an outcome from a Normal distribution with uncertain mean and variance
- Deriving a Gamma distribution from elicited mean and SD

This practical requires the functions (`dbb`,`pbb`,`qbb` and `rbb`) to compute properties of Beta-Binomial distributions. These functions can be installed by loading their source code from the file `betabin.R` that is provided with the course material. These functions are fairly simple - examining their code should help to understand the beta binomial distribution.

Alternative installation method As an alternative to loading the `betabin.R` file, `dbb`,`pbb`,`qbb` and `rbb` are also provided in the R package `TailRank`. To install this, do

```
install.packages("BiocManager")
BiocManager::install("Biobase")
install.packages("TailRank")
```

followed by loading it into your working session with

```
library(TailRank)
```

---

1. [Beta-Binomial prediction] We have a population of 100 people with the disease referred to in Practical 1. We are uncertain about the probability of surviving this disease, and describe this uncertainty using the Beta distribution from question 2 of Practical 1. We want to predict the number of people y in this population who will survive the disease.
   (a) Write down the mean of the predictive distribution for y (no computation needed).
   (b) Use the function `pbb` to calculate $P(y > 20)$. (Note `pbb(x, )` gives $P(y \leq x)$)
   (c) Use the function `qbb` to calculate a 95% equal-tailed credible interval for y. How does the credible interval for the predicted proportion of survivors among 100 people compare to the original credible interval for the survival probability specified in (1)? Explain any difference.
2. Use Monte Carlo simulation to verify the answers from part (1). Ensure that there are enough samples that the probability computed in (1b) is accurate to within around ± 0.001, as measured by a Monte Carlo standard error of 0.0005 or less.

   Hint: To get the Monte Carlo standard error, note that the Monte Carlo estimate of a probability is calculated as the mean of a vector of sampled binary outcomes, each taking the value 0 or 1.
3. Suppose that we are deciding whether a new treatment should be provided to patients. We would prefer the new treatment if is expected to give a higher survival probability than the treatment used for the population in questions 1-2 (assuming the costs of the treatments are the same). Our evidence about the survival probability p_{new} with the new treatment can be represented by a Normal(1,1) distribution on the *log odds ratio* $\log(O_{new}/O_{old})$, where $O_d = p_d/(1-p_d)$ is the *odds* of survival with treatment d, and p_d is the survival probability ($d = old$ or $d = new$). Determine the expected survival probability under the new treatment, and a 95% credible interval for this probability.
4. This question extends the example in the lectures of a normal distribution with an uncertain mean. A pollution measurement has a normal distribution with mean μ and standard deviation $\sigma = 8$. The mean has a normal prior distribution with mean $m_\mu = 120$ and standard deviation $s_\mu = 10$.
   (a) Suppose the maximum legal level for the pollutant is 145 μ_g/m^3. Calculate the prior probability that the measurement X will exceed this limit.

(b) Instead of a fixed measurement error of $\sigma = 8$, suppose we are uncertain about the extent of measurement error. Our prior mean is $\sigma = 8$, and our uncertainty about σ is expressed by a prior standard deviation of 2. Derive an appropriate Gamma prior distribution for σ, and verify that the 95% credible interval for this distribution is roughly centered around the mean with a width of roughly four prior standard deviations.

> *Note:* The Gamma distribution is a distribution for positive-valued quantities. It can be parameterised in different ways. In the R function 'dgamma' and related functions, it is defined by a shape parameter a and a rate parameter b, and has probability density $f(x|a,b) = \frac{b^a}{\Gamma(a)} x^{a-1} \exp(-bx)$. The mean is $m = a/b$ and the variance is $v = a/b^2$.

(c) Use simulation to determine the probability that X exceeds the legal limit under this prior. Compare the standard deviation of the prior predictive distribution to the one where $\sigma = 8$ was fixed. The R functions `rnorm` and `rgamma` can be used for simulation.

> *Note:* if the gamma distribution is placed on $1/\sigma^2$ rather than σ, then the prior predictive distribution has the analytic form of a t distribution, though this is not considered further here.

3 Practical session (3): introduction to JAGS

This practical is a worked example of how to run the JAGS software, via the `rjags` R package. Simply run the code on your computer. Examine each object that you create, and ensure that the explanations make sense.

There are multiple R packages available for controlling JAGS, but `rjags` is the "official" one, in the sense that it is developed by the author of JAGS and released alongside JAGS itself.

We will use JAGS to predict a quantity from a beta-binomial distribution (a binomial distribution with an uncertain proportion) as we did using R in Practical 2.

If you have not already done so, download and install the JAGS software from

`https://sourceforge.net/projects/mcmc-jags/`

Windows and Mac installers are available, and as far as we know, it is also packaged for common Linux distributions (or can be installed from source).

After you have installed JAGS, install the `rjags` R package, in the usual way that R packages are installed, e.g.

```
install.packages("rjags")
```

Then calling

```
library(rjags)
```

should then say something like `Linked to JAGS 4.3.1` if JAGS has been installed correctly. If that doesn't work, make sure you have installed the *most recent* versions of both JAGS and rjags.

3.1 Defining and supplying a model

Running JAGS requires a *model* specified in the JAGS language. Simple models can easily be stored as *character strings* in R. The model from Practical 2 can be specified as follows (with the approximate Beta parameters hard-coded here for convenience)

```
mod <- "model {
  pop_size <- 100
  theta ~ dbeta(4.168, 37.515)
  cases_pred ~ dbinom(theta, pop_size)
}
"
```

The model is supplied to JAGS by calling the `jags.model` function from `rjags`.

If there are any observed data, then this is supplied in the `data` argument. Here there are no data and we are simply making predictions from a given model with parameters given "prior" distributions. So we set `data=NULL`. In later sessions, we will be including observed data here so that we can learn from the data, and use JAGS to update the priors to posteriors.

The model code can be supplied to `jags.model` in one of two ways. Firstly, by using `textConnection` on the character string containing the model, as follows:

```
library(rjags)
mod_jag <- jags.model(textConnection(mod), data=NULL)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
```

```
##    Observed stochastic nodes: 0
##    Unobserved stochastic nodes: 2
##    Total graph size: 5
##
## Initializing model
```

An alternative way is to store the model code in a file, and specify the path to the file `jags.model`. This is demonstrated here by using the `cat` function to save the string `mod` containing the model to a temporary file. The path to this file is given in the string `mod_file`.

```
mod_file <- tempfile()
mod_file
```

```
## [1] "C:\\Users\\Chris\\AppData\\Local\\Temp\\RtmpAr0OQT\\file40885a9a7b96"
```

```
cat(mod, file=mod_file)
```

This file path is then supplied to `jags.model`.

```
mod_jag <- jags.model(mod_file, data=NULL)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 0
##    Unobserved stochastic nodes: 2
##    Total graph size: 5
##
## Initializing model
```

In your own projects, you probably wouldn't use a temporary file, instead you might prefer to store the model file in the same folder as your other analysis code.

3.2 Sampling from a model

We can now draw samples from the distributions of any uncertain quantity in the model, using the function `coda.samples`. We tell JAGS to draw 10000 samples of random values from

(a) the prior distribution of `theta`,

(b) the prior predictive distribution of `cases_pred`, the number of cases in a population of `pop_size` = 100 people.

```
pars <- c("theta", "cases_pred")
nsim <- 10000
sam_c <- coda.samples(mod_jag, pars, n.iter=nsim)
```

The `coda.samples` function returns an R list object, with one component per "chain". Chains are different sequences of samples starting from different initial values – these concepts are not relevant in this example, where all simulation draws are independent, but will become relevant in the next session when doing MCMC sampling to learn from data.

Here we ran one chain (the default) and we extract the component for this chain into the object `sam_m`. This is a matrix with rows for samples, and columns for different variables.

```
sam_m <- sam_c[[1]]
```

3.3 Summarising samples from a model

Here we extract the vector of sampled values from the prior predictive distribution of `cases_pred`, and estimate the number of cases C for which $P(cases\_pred \leq C) = 0.025$ and for which $P(cases\_pred \leq C) = 0.975$, giving an equal-tailed 95% credible interval.

```
cases_pred <- sam_m[,"cases_pred"]
quantile(cases_pred, c(0.025, 0.975))
```

```
##  2.5% 97.5%
##     2    23
```

A prior credible interval for `theta` can be obtained in the same way

```
quantile(sam_m[,"theta"], c(0.025, 0.975))
```

```
##       2.5%      97.5%
## 0.02963032 0.20412659
```

Make sure that the results agree with those from Practical 2 – what is the reason for any difference, and how could we make the agreement closer?[1]

Posterior probabilities (e.g. that `cases_pred` is less than a certain value C) are computed by counting the number of sampled values which are $< C$. This can easily be done in R as follows, say for $C = 20$:

```
mean(cases_pred < 20)
```

```
## [1] 0.9389
```

Recall Practical 2 for an explanation of how this R command works.

In other teaching materials about the BUGS language (e.g. the BUGS Book) you may see use of the `step` function in BUGS, which is used to test whether a random number is less than a reference value C, hence to compute posterior probabilities. This isn't used in this course. We think it's slightly clearer to use JAGS/BUGS solely for computing the random samples, and use R for summarising those samples.

If needed, see `help(jags.model)` for how to set the seed in `rjags` so that repeated runs are reproducible – note that `rjags` uses a non-standard method for doing this, and the `set.seed` function typically used in R for this will not work. Though forcing simulations to be reproducible by setting the seed should not be used as a substitute for understanding how much Monte Carlo error there may be in the estimates.

[1] Monte Carlo error. Reduce this by drawing more samples.

Solutions: Practical 1

1. We can obtain this constant most easily using the standard normal distribution

```
n_sds <- qnorm(0.975) - qnorm(0.025)
n_sds
```

```
## [1] 3.919928
```

Note that for any value of the SD σ, the 95% credible interval of the normal distribution has a width of `n_sds*SD` (since if X has a $N(\mu, \sigma^2)$ distribution, then $(X - \mu)/\sigma$ has a standard $N(0, 1)$ distribution).

```
n_sds <- (qnorm(0.975, sd=1.234) - qnorm(0.025, sd=1.234))/1.234
n_sds
```

```
## [1] 3.919928
```

2. Using the method described in the lectures, where we assume the best guess is the mean μ, convert the credible interval width to a standard deviation σ, and then obtain a and b in terms of μ and σ.

```
upper <- 0.2
lower <- 0.02
mu <- 0.1
sigma <- (upper - lower) / n_sds
(a <- (mu*(1 - mu)/sigma^2 - 1)*mu)
```

```
## [1] 4.168288
```

```
(b <- (mu*(1 - mu)/sigma^2 - 1)*(1 - mu))
```

```
## [1] 37.51459
```

```
a / (a+b)
```

```
## [1] 0.1
```

```
qbeta(c(0.025, 0.5, 0.975), a, b)
```

```
## [1] 0.02964344 0.09361607 0.20623407
```

3. We use the same method as in parts 1-2, but changing the "number of standard deviations" `n_sds` to cover a 50% credible interval instead of a 95% interval.

```
n_sds <- qnorm(0.75) - qnorm(0.25)
sigma <- (upper - lower) / n_sds
(a50 <- (mu*(1 - mu)/sigma^2 - 1)*mu)
```

```
## [1] 0.4054849
```

```
(b50 <- (mu*(1 - mu)/sigma^2 - 1)*(1 - mu))
```

```
## [1] 3.649364
```

```
a50/(a50+b50)
```

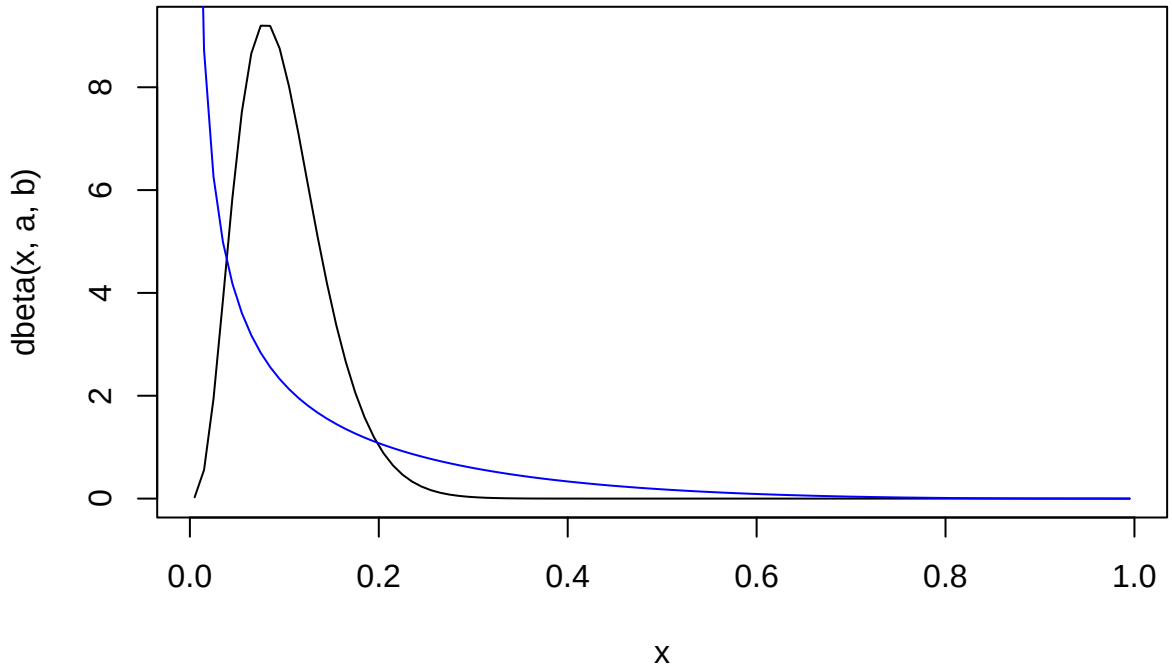
```
## [1] 0.1
```

```
qbeta(c(0.25, 0.5, 0.75), a50, b50)
```

```
## [1] 0.007347167 0.043441782 0.141495341
```

Instead of 0.02 to 0.2, the actual theoretical 50% interval of this distribution is 0.01 to 0.14. This Beta distribution is a less accurate representation of our belief than the one that we derived in part (2).

4.
```
x <- ppoints(100)
plot(x, dbeta(x, a, b), type="l")
lines(x, dbeta(x, a50, b50), col="blue")
```



The distribution in (2) has a *mode* (maximum probability density) between 0 and 1 (close to the mean of 0.1), whereas the distribution in (3) has a mode of zero. We might not have wanted a distribution where the mode is so different from the mean and median. We may want to revise our judgements so that the elicited distribution more closely represents our beliefs. We could search for a Beta with a given mode and/or median instead of a given mean, and/or provide credible intervals with different widths.

See, e.g. the Wikipedia page about the Beta distribution for a formula for the mode of the Beta for different ranges of the parameters a and b. Note that a and b both need to be > 1 for the distribution to have a single peak inside $(0, 1)$.

5. There is no single correct way to write this function, as long as it does what it is designed to do. Ideally it should be written clearly enough that someone else (e.g. you in a year's time!) can understand what each part does. I might write it in the following way:

```
beta_dist <- function(mu, ci, ci_width=0.95) {
  tail_prob <- (1 - ci_width)/2
  n_sds <- qnorm(1 - tail_prob) - qnorm(tail_prob)
  sigma <- (upper - lower) / n_sds
  n <- mu*(1 - mu)/sigma^2 - 1
  a <- n*mu
  b <- n*(1 - mu)
  c(a=a, b=b)
}

beta_dist(0.1, c(0.02, 0.2))
```

```
##         a         b
##  4.168288 37.514588
```

```r
beta_dist(0.1, c(0.02, 0.2), ci_width=0.5)
```

```
##         a         b
## 0.4054849 3.6493642
```

Note the programming style of

- giving names to things that have meanings (e.g. `tail_prob`, `n_sds`)
- defining quantities that are used more than once (e.g. `n` here. This quantity is actually the "effective sample size" — as explained in the lecture).

Solutions: Practical 2

1. (a) Recall that the Beta distribution in question 2 of Practical 1 was defined by a mean based on a prior guess of 10%. Therefore the mean of the prior predictive distribution for the number of survivors among 100 people is simply 10.

   (b)

```
1 - pbb(20, 100, a, b)
```

```
## [1] 0.04536761
```

   (c)

```
qbb(c(0.025, 0.975), 100, a, b)
```

```
## [1]  2 23
```

   The credible interval for the proportion of survivors out of 100 people is 2/100 to 23/100. This is similar to the prior credible interval of 0.02 to 20 for the survival probability. The proportion of survivors is an *estimate* of this probability based on a sample of 100 people and exactly the same information as the prior credible interval. Therefore we expect it to be similar. Any difference is from the finite sample size of 100, and the discrete support of the the beta-binomial distribution (therefore its quantiles must be integers).

2. We draw a large number of values of survival probabilities from the Beta prior, and then draw a vector of Binomially-distributed outcomes generated from each sampled probability in turn.

```
R <- 200000
theta <- rbeta(R, a, b)
y_pred <- rbinom(R, 100, theta)
```

   The proportion of those sampled Binomial outcomes that are greater than 20 can be calculated as follows.

```
mean(y_pred > 20)
```

```
## [1] 0.045265
```

   This works in R, because `y_pred < 20` is actually a vector of "logical" values `TRUE` or `FALSE`. The `i`th element is `TRUE` if the `i`th element of `y_pred` is less than 20. The function `mean` in R computes the mean of a vector of values. Logical values `TRUE` and `FALSE` are treated the same as the numeric values 1 and 0. So the following are equivalent:

```
mean(c(18, 19, 21) > 20)
```

```
## [1] 0.3333333
```

```
mean(c(FALSE, FALSE, TRUE))
```

```
## [1] 0.3333333
```

```
mean(c(0, 0, 1))
```

```
## [1] 0.3333333
```

   Therefore we can obtain the Monte Carlo standard error from the formula $sd(Y)/\sqrt{R}$, where Y is the sampled 0/1 binary variable whose mean is the probability that we want.

```
(mcse <- sd(y_pred > 20) / sqrt(R))
```

```
## [1] 0.0004648457
```

Predicting $R = 200000$ samples gives a Monte Carlo standard error (MCSE) of just less than 0.0005 for the probability (in theory, the estimate of MCSE itself is affected by Monte Carlo error, but this does not generally affect the conclusions)

```
quantile(y_pred, c(0.025, 0.975))
```

```
##  2.5% 97.5%
##     2    23
```

3. This requires simulation. First sample from the uncertainty distribution of survival probabilities under the old treatment, then convert these to a sample for the *odds* of survival.

```
nsim <- 100000
p_old <- rbeta(nsim, a, b)
odds_old <- p_old / (1 - p_old)
```

Generate a sample from the uncertainty distribution of the log odds ratio, and convert to a sample for the odds ratio O_{new}/O_{old}.

```
log_or <- rnorm(nsim, 1, 1)
or <- exp(log_or)
```

Then multiply the odds of survival with the old treatment by the odds ratio, to obtain the odds of survival for the new treatment, and convert the odds to the probability of survival.

```
odds_new <- odds_old * or
p_new <- odds_new / (1 + odds_new)
```

The expected survival probability under the new treatment is 0.26 (95% credible interval 0.03 to 0.71), compared to 0.1 under the old one.

```
mean(p_old)
```

```
## [1] 0.1000777
```

```
mean(p_new)
```

```
## [1] 0.2602324
```

```
quantile(p_new, c(0.025, 0.975))
```

```
##       2.5%      97.5%
## 0.02782226 0.70976940
```

(Note: we did not need this analysis to determine that the new treatment was better, since we already knew that the mean of the Normal distribution for the log odds ratio was positive. However this analysis gives us a full posterior distribution for the *absolute* risk under the new treatment, which might be used as a part of a more complicated model for determining all of the costs and consequences of each policy.)

4. (a) Using the expression for the normal prior predictive distribution given in the lectures, and using `pnorm` in R to obtain the value of the CDF of this distribution at the critical value of 145:

```
1 - pnorm(145, mean=120, sd = sqrt(10^2 + 8^2))
```

```
## [1] 0.02545889
```

(b) Simple algebraic manipulation gets us $b = m/v$, and $a = m^2/v$, where $v = s^2$. Setting $m = 8$ and $s = 2$ then gives the prior of $\sigma \sim Gamma(a = 16, b = 2)$. The credible interval roughly reflects the beliefs that we specified about the plausible ranges of σ.

```
qgamma(c(0.025, 0.975), 16, 2)
```

```
## [1]  4.572691 12.370109
```

(c) First we simulate R plausible alternative values for σ from the prior distribution.

```
R <- 100000
sigma_rep <- rgamma(R, 16, 2)
```

Then for each $\sigma^{(r)}, r = 1, \ldots, R$ we draw a sample from the prior predictive distribution $X^{(r)} \sim N(m_\mu, \sqrt{s_\mu^2 + {\sigma^{(r)}}^2})$.

```
X_rep <- rnorm(R, mean=120, sd=sqrt(10^2 + sigma_rep^2))
```

Properties of the prior predictive distribution can be computed by summarising this sample. For example, $P(X > 145)$ is computed by counting the proportion of sampled values which are greater than 145.

```
mean(X_rep > 145)
```

```
## [1] 0.0272
```

Introducing uncertainty about the measurement error σ has slightly increased the uncertainty about the predicted value X, as shown by comparing the standard deviations of the prior predictive distributions:

```
c(
  sd_fixed = sqrt(10^2 + 8^2),
  sd_uncertain = sd(X_rep)
)
```

```
##     sd_fixed sd_uncertain
##     12.80625     12.93627
```

For reassurance that these results are not affected by Monte Carlo error, you might also repeat the simulation, and/or increase R, and ensure the summaries you want are consistent to the desired accuracy.