Appendices

A Expressing EVSI as a Function of Opportunity Losses Instead of Benefits

EVSI is typically defined as[? ?] :

$$\text{EVSI} = \mathbb{E}_X \max_d \mathbb{E}_\theta[B(d,\theta)|X] - \max_d \mathbb{E}_\theta[B(d,\theta)], \tag{A.1}$$

where θ is the parameters of interest, X is a proposed data collection, $B(d,\theta)$ is the benefit from the strategy d and the parameter values θ. Thus, the EVSI is the difference between the highest benefit given new data X and the highest expected benefit with current information. Because, X is not known, the first term is averaged across all possible values of X.

In this paper, we modify the definition of EVSI given by Equation (A.1) and express EVSI as a function of the opportunity loss from choosing a sub-optimal decision rather than the benefits.

The second term in Equation A.1 is simply the average benefit of the optimal strategy d^*, where $d^* = \arg\max_d \mathbb{E}_\theta[B(d,\theta)]$, therefore, we can express (A.1) as

$$\text{EVSI} = \mathbb{E}_X \max_d \mathbb{E}_\theta[\text{B}(d,\theta)|X] - \mathbb{E}_\theta \text{B}(d^*,\theta).$$

According to the law of conditional expectation, this is equivalent to

$$\text{EVSI}_{\theta_I} = \mathbb{E}_X \max_d \mathbb{E}_\theta[\text{B}(d,\theta)|X] - \mathbb{E}_X\mathbb{E}_\theta[\text{B}(d^*,\theta)|X]$$

or

$$\text{EVSI}_{\theta_I} = \mathbb{E}_X \left[\max_d \mathbb{E}_\theta[\text{B}(d,\theta)|X] - \mathbb{E}_\theta[\text{B}(d^*,\theta)|X]\right].$$

Because the second term inside the squared brackets is not dependent on d, we further simplify this equation as

$$\text{EVSI} = \mathbb{E}_X \max_d \mathbb{E}_\theta\left[\{\text{B}(d,\theta) - \text{B}(d^*,\theta)\}|X\right].$$

Since $\text{L}(d,\theta) = \text{B}(d,\theta) - \text{B}(d^*,\theta)$ is the expected opportunity loss from choosing the optimal intervention d^* among all possible decisions $d \in D$, the set of decisions, given the parameter values θ, Equation (A.1) can be expressed as a function of the opportunity loss as

$$\mathrm{EVSI}_{\theta_I} = \mathbb{E}_X \max_d \mathbb{E}_\theta[L(d,\theta)|X]. \tag{A.2}$$

Thus, EVSI computes the maximum expected expected loss given new data X. Again, because X is unknown the average is taken over all the possible values of X .

B R Code to Calculate EVPPI and EVSI

The two files provided below are used to apply our Gaussian approximation for EVSI. The first file (`EVSI_GA_Appendix.R`) details each of the steps that are summarized in Box 1 of the main text. This file also provides the R code for computing EVSI for several scenarios using this approach for a single parameter, multiple parameters, and balanced and unbalanced designs. The second file (`GA_functions.R`) provides the `predict.ga` function that calculates the conditional loss $\tilde{L}_d^i$ by computing the preposterior for each of the basis functions of the GAM model. This code can also be downloaded from `https://github.com/feralaes/VOI-Gaussian-Approximation`. The version used of the package `mgcv` was `1.8-17`.

B.1 File 1: `EVSI_GA_Appendix.R`

To calculate EVSI using our GA approach on the example PSA dataset, make sure that the two R files and the `psa_demo_GA.csv` dataset are located in the same folder, which should be specified as your working directory by typing: `setwd("your_working_directory_here")`

```
### Load PSA
psa <- read.csv("psa_demo_GA.csv")
head(psa)

### Create following matrices from your PSA
## Net monetary benefit (NMB) matrix
nmb <- psa[, 5:7]
head(nmb)
## Matrix of model parameter inputs values theta
theta <- psa[, 1:4]
head(theta)
## Number of simulations
n.sim        <- nrow(nmb)
## Number of strategies
n.strategies <- ncol(nmb)
```

```
### Load required packages and functions
## For column and row stats
library(matrixStats)
## To fit spline models
library(mgcv) # mgcv version 1.8-17
## Functions to calculate the conditional loss by computing the
## preposterior of each of the basis functions of the GAM model
source("Rcode/Appendix_Final/GA_functions.R")

### Find optimal strategy (d*) based on the highest expected NMB
d.star <- which.max(colMeans(nmb))
d.star

### Define the Loss matrix
loss <- nmb - nmb[, d.star]

### EVPI
evpi <- mean(rowMaxs(as.matrix(loss)))
evpi

#========================#
#### Single parameter ####
#========================#
### Generate linear metamodel of one parameter for each opportunity loss
## Selected parameter for EVPPI & EVSI
sel.param <- 3
lmm1 <- gam(as.formula(paste("loss[, 1] ~ s(", colnames(theta)[sel.param], ")")),
            data = theta)
lmm2 <- gam(as.formula(paste("loss[, 2] ~ s(", colnames(theta)[sel.param], ")")),
            data = theta)
```

```
lmm3 <- gam(as.formula(paste("loss[, 3] ~ s(", colnames(theta)[sel.param], ")")),
            data = theta)


#### Compute EVPPI on one parameter ####
## Compute estimated losses
loss.hat <- cbind(lmm1$fitted, lmm2$fitted, lmm3$fitted)

### Apply EVPPI equation
evppi <- mean(rowMaxs(loss.hat))
evppi

#### Compute EVSI on one parameter ####
## Initial sample size
n0 <- 10
## Additional sample size
n <- 100

### Compute expected conditional loss for each strategy
Ltilde1 <- predict.ga(lmm1, n = n, n0 = n0)
Ltilde2 <- predict.ga(lmm2, n = n, n0 = n0)
Ltilde3 <- predict.ga(lmm3, n = n, n0 = n0)
## Combine losses into one matrix
loss.tilde <- cbind(Ltilde1, Ltilde2, Ltilde3)

### Apply EVSI equation
evsi <- mean(rowMaxs(loss.tilde))
evsi

#======================#
#### Two parameters ####
```

```
#======================#
### Generate linear metamodel of two parameters for each opportunity loss
## Select parameters for EVPPI & EVSI
sel.param1 <- 1
sel.param2 <- 3
sel.params <- c(sel.param1, sel.param2)
### Estimate linear metamodel of two parameters
lmm1 <- gam(as.formula(paste("loss[, 1] ~ s(",
                             paste(colnames(theta[, sel.params]), collapse= ") + s("),
                             ") + ti(",
                             paste(colnames(theta[, sel.params]), collapse= ", "),
                             ")")), data = theta)
lmm2 <- gam(as.formula(paste("loss[, 2] ~ s(",
                             paste(colnames(theta[, sel.params]), collapse= ") + s("),
                             ") + ti(",
                             paste(colnames(theta[, sel.params]), collapse= ", "),
                             ")")), data = theta)
lmm3 <- gam(as.formula(paste("loss[, 3] ~ s(",
                             paste(colnames(theta[, sel.params]), collapse= ") + s("),
                             ") + ti(",
                             paste(colnames(theta[, sel.params]), collapse= ", "),
                             ")")), data = theta)

#### Compute EVPPI on two parameters ####
## Compute estimated losses
loss.hat <- cbind(lmm1$fitted, lmm2$fitted, lmm3$fitted)

### Apply EVPPI equation
evppi <- mean(rowMaxs(loss.hat))
evppi
```

```
#### Compute EVSI on two parameters Case 1: Same n0 and n ####
## Initial sample size of each parameter
n0 <- c(10, 10)
## Additional sample size
n  <- c(100, 100)

### Compute expected conditional loss for each strategy
Ltilde1 <- predict.ga(lmm1, n = n, n0 = n0)
Ltilde2 <- predict.ga(lmm2, n = n, n0 = n0)
Ltilde3 <- predict.ga(lmm3, n = n, n0 = n0)
## Combine losses into one matrix
loss.tilde <- cbind(Ltilde1, Ltilde2, Ltilde3)

### Apply EVSI equation
evsi <- mean(rowMaxs(loss.tilde))
evsi

#### Compute EVSI on two parameters Case 2: Different n0, same n ####
## Initial sample size of each parameter
n0 <- c(10, 50)
## Additional sample size
n  <- c(100, 100)

### Compute expected conditional loss for each strategy
Ltilde1 <- predict.ga(lmm1, n = n, n0 = n0)
Ltilde2 <- predict.ga(lmm2, n = n, n0 = n0)
Ltilde3 <- predict.ga(lmm3, n = n, n0 = n0)
## Combine losses into one matrix
loss.tilde <- cbind(Ltilde1, Ltilde2, Ltilde3)

### Apply EVSI equation
```

```
evsi <- mean(rowMaxs(loss.tilde))
evsi

#### Compute EVSI on two parameters Case 3: Different n0 and n ####
## Initial sample size of each parameter
n0 <- c(10, 50)
## Additional sample size
n  <- c(100, 1000)

### Compute expected conditional loss for each strategy
Ltilde1 <- predict.ga(lmm1, n = n, n0 = n0)
Ltilde2 <- predict.ga(lmm2, n = n, n0 = n0)
Ltilde3 <- predict.ga(lmm3, n = n, n0 = n0)
## Combine losses into one matrix
loss.tilde <- cbind(Ltilde1, Ltilde2, Ltilde3)

### Apply EVSI equation
evsi <- mean(rowMaxs(loss.tilde))
evsi
```

B.2 File 2: `GA_functions.R`

```
predict.ga <- function(object, n, n0, verbose = T){
  #### Function to compute the preposterior for each of the
  #### basis functions of the GAM model.
  #### Inputs:
  #### - object: gam object
  #### - n: scalar or vector of new sample size to compute evsi on
  #### - n0: scalar or vector of effective prior sample size
  #### - verbose: Prints the variance reduction factor for each parameter

  ### Name of parameters
  names.data <- colnames(object$model)
  ### Create dataframe with parameter values
  data <- data.frame(object$model[,-1])
  ## Name columns of dataframe
  colnames(data) <- names.data[-1]

  ### Number of parameters
  n.params <- ncol(data)

  ### Sanity checks
  if(!(length(n)==1 | length(n)==n.params)){
    stop("Variable 'n' should be either a scalar or a vector
         the same size as the number of parameters")
  }
  if(!(length(n0)==1 | length(n0)==n.params)){
    stop("Variable 'n0' should be either a scalar or a vector
         the same size as the number of parameters")
  }

  ### Make n & n0 consistent with the number of parameters
```

```
if(length(n) == 1){
  n <- rep(n, n.params)
}
if(length(n0) == 1){
  n0 <- rep(n0, n.params)
}

### Compute variance reduction factor
v.ga <- sqrt(n/(n+n0))
if (verbose){
  print(paste("Variance reduction factor =", round(v.ga, 3)))
}

### Number of smoothers
n.smooth <- length(object$smooth)
### Number of total basis functions
n.colX <- length(object$coefficients)
### Number of observations
n.rowX <- nrow(object$model)

### Initialize matrix for preposterior of total basis functions
X <- matrix(NA, n.rowX, n.colX)
X[, 1] <- 1

for (k in 1:n.smooth) { # k <- 1
  klab <- substr(object$smooth[[k]]$label, 1, 1)
  if (klab == "s"){
    Xfrag <- Predict.smooth.ga(object$smooth[[k]], data, v.ga[k])
  } else {
    Xfrag <- Predict.matrix.tensor.smooth.ga(object$smooth[[k]], data, v.ga)
  }
```

```
    X[, object$smooth[[k]]$first.para:object$smooth[[k]]$last.para] <- Xfrag
  }

  ### Coefficients of GAM model
  Beta <- coef(object)

  ### Compute conditional Loss
  Ltilde <- X %*% Beta

  return(Ltilde)
}

Predict.smooth.ga <- function (object, data, v.ga = 1) {
  #### Function to compute the preposterior for each of the
  #### basis functions of a smooth for one parameter

  ### Produce basis functions for one parameter
  X <- PredictMat(object, data) # mgcv version 1.8-17
  ## Number of observations
  n.obs <- nrow(X)

  ### Apply variance reduction to compute the preposterior
  ### for each of the basis functions
  ## Vector of ones
  ones <- matrix(1, n.obs, 1)
  ## Compute phi on each of the basis function
  X.ga <- v.ga*X + (1-v.ga)*(ones %*% colMeans(X))

  return(X.ga)
}
```

```
Predict.matrix.tensor.smooth.ga <- function (object,
                                             data,
                                             v.ga = rep(1, ncol(data))){
  #### Function to compute the preposterior for each of the
  #### basis functions for one or more parameters and calculates
  #### the tensor product if more than one parameter is selected
  #### (Heavily based on function Predict.matrix.tensor.smooth from
  #### mgcv package)

  m <- length(object$margin)
  X <- list()
  for (i in 1:m) { # i <- 1
    term <- object$margin[[i]]$term
    dat <- list()
    for (j in 1:length(term)) { # j <- 1
      dat[[term[j]]] <- data[[term[j]]]
    }
    X[[i]] <- if (!is.null(object$mc[i])) # before: object$mc[i]
      PredictMat(object$margin[[i]], dat, n = length(dat[[1]])) # mgcv version 1.8-17
    else Predict.matrix(object$margin[[i]], dat)
    n.obs <- nrow(X[[i]])
  } # end for 'i'
  mxp <- length(object$XP)
  if (mxp > 0)
    for (i in 1:mxp) if (!is.null(object$XP[[i]]))
      X[[i]] <- X[[i]] %*% object$XP[[i]]

  ### Apply variance reduction to compute the preposterior
  ### for each of the basis functions
  ## Vector of ones
  ones <- matrix(1, n.obs, 1)
  ## Initialize and fill list with preposterior of basis functions
  ## for each parameter
  X.ga <- list()
  for (i in 1:m) { # i <- 1
    X.ga[[i]] <- v.ga[i]*X[[i]] + (1-v.ga[i])*(ones %*% colMeans(X[[i]]))
  }

  ### Compute tensor product
  T.ga <- tensor.prod.model.matrix(X.ga) # mgcv version 1.8-17

  return(T.ga)
}
```

C Indirect methods for calculating n_0

Because the direct method is only possible for a limited set of prior-likelihood combinations, we propose a couple of indirect methods that can be applied in many cases, including situations in which the prior and data likelihoods are not conjugate. In Box 2 of the main text we provide a summary of this approaches and here we provide a more detailed step-by-step approach in addition to the associated R codes for a beta-binomial example.

The indirect method involves computing n_0 by generating new data from the prior and the likelihood function.

C.1 Detailed Algorithms for computing n_0

C.1.1 Using a summary statistic S

This approach requires simulating data from the prior and then generate data from the likelihood. In cases where the likelihood is non-Gaussian, a summary statistic is needed. However, calculating a summary statistic may not always be trivial. For such cases, we propose an indirect MCMC approach that does not require computing the summary statistic, which is detailed in the next subsection. The steps to estimate $n0$ using the summary statistic approach are

1. Draw m random samples from the prior distribution of θ_I.
2. For each random sample $\theta_I^{(i)}$, where $i = \{1, ..., m\}$, draw n samples of x from the distribution of the data likelihood. n can be any positive integer, the larger the n, the more accurate n_0 is.
3. Compute a summary statistic $S^{(i)}$ from the n samples for each $\theta_I^{(i)}$ prior value.
4. Estimate n_0 using Equation (**??**):

$$\hat{n}_0 = n\left(\frac{\mathrm{Var}(S)}{\mathrm{Var}(\theta_I)} - 1\right),$$

   where all the variables are as previously described. In appendix C.2, we provide the R code to compute n_0 using a GA for Experiments 1 and 2 described in Table (1), and an additional gamma-Poisson experiment.

C.1.2 Using MCMC

Computing $n0$ using Markov-chain Monte-Carlo (MCMC) does not require calculating a summary statistic. However, in addition of generating new data from the likelihood function, it is necessary to obtain the preposterior distribution as follows:

1. Draw m random samples from the prior distribution of θ_I. For each random sample $\theta_I^{(i)}$, where $i = \{1, ..., m\}$, draw n samples from the distribution of the data likelihood. n can be any positive integer, the larger the n, the more accurate n_0 is.

2. Take q samples from the posterior distribution of $\theta|x^{(i)}$. Then, we calculated the posterior mean for each value of $\theta_I^{(i)}$, such that $\phi^{(i)} = \frac{1}{q}\sum_{k=1}^{q} \theta^{(k)}|x^{(i)}$.

3. From Equation (**??**), n_0 can be estimated as

$$\hat{n}_0 = n\left(\frac{\text{Var}(\theta_I)}{\text{Var}_X(\phi)} - 1\right), \tag{C.1}$$

   where all the variables are as previously described. In appendix C.2, we provide the R code to compute n_0 uisng MCMC for Experiments 1 and 3 described in Table (1).

C.2 R Code to estimate n_0

Using a summary statistic S

Beta-binomial The R code below illustrates how to estimate n_0 from a probability parameter that follows a beta distribution $p \sim \text{beta}(a = 12, b = 8)$, and a binomial likelihood for the proposed experimental data collection $x \sim \text{binomial}(p, n = 50)$, where n is the new sample size. First, draw m samples of p from the prior. Second, for each simulated $p^{(i)}$ draw one $x^{(i)}$ value from the likelihood, where $i = 1, \dots, m$. Third, compute the summary statistic $S^{(i)} = \frac{x^{(i)}}{n}$. Finally, estimate n_0 using Equation (**??**). The predicted n_0 from this example was 21.93 which was close to the true $n_0 = a + b = 20$.

```
#### beta-binomial ####
### 1. Obtain samples of theta from the prior distribution in R
```

```
a <- 12 # alpha parameter for the beta prior distribution theta~beta(a,b)
b <- 8  # beta parameter for the beta distribution
nSample <- 1000 # Number of samples from the prior
## Draw sample
theta <- rbeta(nSample, a, b)
## Compute variance of theta
var.theta <- var(theta)

### 2. Obtain experimental data from the data likelihood in R
S <- numeric(nSample) # Initialize summary statistic S vector
n <- 50 # Additional data collections
## Generate data and compute summary statistic S
for (i in 1:nSample){
  k <- rbinom(1, n, theta[i])
  S[i] <- k/n
}
## Compute variance of S
var.S <- var(S)

## Estimate n0
n0.hat <- n*(var.S/var.theta-1)
n0.hat
```

Gamma-exponential The R code below illustrates how to estimate n_0 from a rate λ that follows a gamma distribution, $\lambda \sim \text{gamma}(a = 20, b = 0.1)$, and an exponential likelihood for the proposed experimental data collection $x \sim \text{Exp}(\lambda)$. First, draw m samples of λ from the prior. Second, for each simulated $\lambda^{(i)}$ we draw $x_j^{(i)}$ values from the likelihood, where $i = 1, \ldots, m$ and $j = 1, \ldots, n$. Third, compute the summary statistic $S^{(i)} = \frac{1}{\frac{1}{n}\sum_{j=1}^{n} x_j^{(i)}}$. Finally, estimate n_0 using Equation (**??**). The predicted n_0 from this example was 25 which was close to the true $n_0 = a = 20$.

```
#### Gamma-exponential ####
### 1. Obtain samples of theta from the prior distribution in R
```

```
a <- 20 # alpha parameter for the gamma prior distribution theta~gamma(a,b)
b <- 0.1 # beta parameter for the gamma distribution
nSample <- 10000 # Number of samples from the prior
## Draw sample
theta <- rgamma(nSample, shape = a, scale = b)
## Compute variance of theta
var.theta <- var(theta)

### 2. Obtain experimental data from the data likelihood in R
S <- numeric(nSample) # Initialize summary statistic S vector
n <- 100 # Additional data collections
## Generate data and compute summary statistic S
for (i in 1:nSample){
  k <- rexp(n, theta[i])
  S[i] <- 1/mean(k)
}
## Compute variance of S
var.S <- var(S)

## Estimate n0
n0.hat <- n*(var.S/var.theta-1)
n0.hat
```

Gamma-Poisson In addition to estimate n_0 on Experiments 1 and 2 using the summary statistic method, we also demonstrate the approach with a gamma-Poisson experiment. The R code below illustrates how to estimate n_0 from a rate λ that follows a gamma distribution, $\lambda \sim \text{gamma}(a = 50, b = 0.01)$, and a Poisson likelihood for the proposed experimental data collection $x \sim \text{Poisson}(\lambda)$. First, draw m samples of λ from the prior. Second, for each simulated $\lambda^{(i)}$ we draw $x_j^{(i)}$ values from the likelihood, where $i = 1, \ldots, m$ and $j = 1, \ldots, n$. Third, compute the summary statistic $S^{(i)} = \frac{1}{n}\sum_{j=1}^{n} x_j^{(i)}$. Finally, estimate n_0 using Equation (**??**). The predicted n_0 from this example was 104 which was close to the true $n_0 = 1/b = 100$.

```
#### Gamma-Poisson ####
### 1. Obtain samples of theta from the prior distribution in R
a <- 50 # alpha parameter for the gamma prior distribution theta~gamma(a,b)
b <- 0.01  # beta parameter for the gamma distribution
nSample <- 10000 # Number of samples from the prior
## Draw sample
theta <- rgamma(nSample, shape = a, scale = b)
## Compute variance of theta
var.theta <- var(theta)

### 2. Obtain experimental data from the data likelihood in R
S <- numeric(nSample) # Initialize summary statistic S vector
n <- 50 # Additional data collections
## Generate data and compute summary statistic S
for (i in 1:nSample){
  k <- rpois(n, theta[i])
  S[i] <- mean(k)
}
## Compute variance of S
var.S <- var(S)

## Estimate n0
n0.hat <- n*(var.S/var.theta-1)
n0.hat
```

C.2.1 Using MCMC

Below we provide two examples to compute n_0 using MCMC. The first example involves a beta prior and a binomial data likelihood, and the second example involves a normal prior and a Weibull data likelihood.

Beta-binomial The R code below illustrates how to estimate n_0 from a probability parameter that follows a beta distribution $p \sim \text{beta}(a = 12, b = 8)$, and a binomial likelihood for the proposed experimental data collection $x \sim \text{binomial}(p, n = 50)$, where n is the new sample size. The code below can be modified for various data collection exercises. The code first generates samples for x, then passes these samples to JAGS to obtain the posterior distribution of $p|x$. Then it calculates the posterior mean, and computes n_0 per Equation (C.1). The predicted n_0 from this example was 21.17 which was close to the true $n_0 = a + b = 20$.

```
library(R2jags)
library(matrixStats)
## 1. Obtain experimental data from the data likelihood in R
a <- 12 # alpha parameter for the beta prior distribution p~beta(a,b)
b <- 8 # beta parameter for the beta distribution
n <- 50 # additional data collections
nSample <- 2000
x <- matrix(0, nSample, 1) # initialize the new data matrix
p <- rbeta(nSample, a, b) # prior distribution of p
for (i in 1:nSample){ # For each prior value, conduct an experiment of size n
  x[i] <- rbinom(1, n, p[i]) # likelihood: x|p
}
## 2. Obtain the posterior distribution & compute the posterior means
test.model <- function() {
  for (i in 1:nSample){
    p[i] ~ dbeta(a, b) #prior distribution of p
    x[i,1] ~ dbin(p[i], n) # likelihood: x|p
  }
}
linedata <- list(x = x, n = n, nSample = nSample, a = a, b = b)
test.sim <- try(jags(data = linedata, inits = NULL, model.file = test.model,
                     parameters.to.save = c("p"),
                     n.chains = 1, n.iter = 11000, n.burnin = 1000, n.thin = 1,
                     progress.bar = "text"))
```

```
post.p <- test.sim$BUGSoutput$sims.list$p #get the posterior distribution sample
prepost <- colMeans(post.p) #obtain the preposterior distribution from the posterior mean of p
var.prior <- var(p) #prior variance
var.prepost <- var(prepost) #preposterior variance
n0.hat <- n * (var.prior / var.prepost - 1) #predicted prior sample size
n0.hat #21.17 compared to a+b = 12+8 = 20.
```

Normal-Weibull First, draw m samples $\theta_I^{(1)}, ..., \theta_I^{(m)}$ from the prior distribution of $\theta_I \sim$ Normal($\mu = 1, \sigma^2 = 0.04$). For each value of $\theta_I^{(i)}$, $i = \{1, ..., m\}$, take $j = \{1, ..., n\}$ samples from the data likelihood, such that $x^{(i,j)} \sim Weibull(v = 1, \lambda = 1/\theta^{(i)})$. Obtain q samples from the posterior distribution of $\theta | x^{(i)}$. Then, calculate the posterior mean for each value of $\theta_I^{(i)}$, such that $\phi^{(i)} = \frac{1}{q}\sum_{k=1}^{q} \theta^{(k)} | x^{(i)}$. Finally, estimate n_0 from Equation (C.1), which was approximately 25.

```
require(R2jags)
shape <- 1 # Weibull shape parameter
n <- 100 # additional data collections
m <- 1000 #number of prior samples
q <- 2000 #number of posterior samples
n.burnin <- 200 # number of burnin posterior values
x <- matrix(0, m, n) # initialize the new data matrix
theta <- rnorm(m, 1, 0.2) # prior distribution of theta
for (i in 1:m){  # For each prior value, conduct an experiment of size n
  x[i, ] <- rweibull(n, 1, theta[i]) # likelihood: x|p
}

## 2. Obtain the posterior distribution & compute the posterior means
test.model <- function() {
  tau <- 1/pow(0.2,2)
  for (i in 1:m){
    theta[i] ~ dnorm(1, tau) #prior distribution of theta
    lambda[i] <- 1/pow(theta[i], v)
    for (j in 1:n)
    {
      x[i,j] ~ dweib(shape, lambda[i]) # likelihood: x|p
    }
  }
}
linedata <- list(x = x, n = n, m = m, shape = shape)
test.sim <- try(jags(data = linedata, inits = NULL, model.file = test.model,
                     parameters.to.save = c("theta"),
                     n.chains = 1, n.iter = q + n.burnin, n.burnin = n.burnin, n.thin = 1,
                     progress.bar = "text"))
post.p <- test.sim$BUGSoutput$sims.list$theta #get the posterior distribution sample
prepost <- colMeans(post.p) #obtain the preposterior distribution from the posterior mean of p
var.prior <- var(theta) #prior variance
var.prepost <- var(prepost) #preposterior variance
n0.hat <- n * (var.prior / var.prepost - 1) #predicted prior sample size
n0.hat #24.8
```

D Markov Model Description

The Markov Model simulated a hypothetical cohort of 30-year old patients with a genetic disorder (syndrome X) who can be in one of three health states, well, disabled and dead (Figure D.1). Syndrome X is mostly asymptomatic, but may cause a sudden flare-up that may necessitate hospitalization. There are three alternatives available for preventing progression to permanent disability: A, B and C, where C is standard care. If treatment fails, permanent disability occurs and quality of life (QoL) decreases from 1.0 to 0.8. For ease of calculations, we assumed that patients face a constant annual mortality rate of 0.044. In addition, syndrome X is assumed to be associated with a 0.5% increase in absolute mortality. Furthermore, if disabled, mortality rate is assumed to increase by an additional 1%. The cycle length is one year. The willingness to pay threshold (WTP) for this analysis is \$50K/QALY and both benefits and costs are discounted by 5% annually. The model parameters are summarized in Table D.1.

Four parameters in the model are uncertain and their corresponding sampling distributions are shown in Figure D.2. Figure D.3 shows the nonlinear relation between the parameters of the Markov model and the estimated benefits from an analysis where we varied each parameter while holding the rest of the parameters fixed at their mean values. The parameters representing the mean number of visits for intervention A and B have a linear relation with the NMB. However, the probability of failing A and B have a nonlinear relation with their respective interventions. This is because Markov models are typically linear in state payoffs, but non-linear in the state-transition probabilities. The n_0 is 10 for all parameters, which can be easily confirmed from the distribution characteristics of these parameters and their corresponding data likelihoods in Table D.1.

Tables

Table D.1: Description of Markov model parameters in case study 2.

Parameter	Distribution	Mean	Shape	Scale
Number of hospital visits (intervention A)	Gamma	1.0	10	0.1
Number of hospital visits (intervention B)	Gamma	2.0	20	0.1
Number of hospital visits (intervention C)	Constant	2.8		
Annual probability of failing A	Beta	0.2	2	8
Annual probability of failing B	Beta	0.3	3	7
Annual probability of failing C	Constant	0.2		
QoL weight while disabled	Constant	0.8		
Annual mortality rate	Constant	0.044		
Absolute increase in annual mortality rate of well	Constant	0.005		
Absolute increase in annual mortality rate of disabled	Constant	0.010		
Annual cost of intervention A	Constant	$10,000		
Annual cost of intervention B	Constant	$2,000		
Annual cost of intervention C	Constant	$100		
Annual cost of disabled	Constant	$5,000		
Cost per hospital visit	Constant	$5,000		

Table D.2: Number and percent of simulations being optimal, and expected net monetary benefit for each intervention

Intervention	Simulations being optimal (%)	Benefit
A	2,847 (28.5)	$445,179
B	4,586 (45.9)	$449,543
C	2,567 (25.7)	$445.305
Total	10,000	

For value of information, we measure benefits in monetary terms.

Figures

Figure D.1. A simplified diagram of the Markov model. The ovals represent the Markov health states. A patient can be in one of the three distinct states: well, disabled or dead. Only transitions in the direction of the arrows are allowed.

Figure D.2. Probabilistic distributions of the uncertain model parameters. Distribution characteristics are described in Table D.1.

Figure D.3. Examining the degree of linearity between the NMBs and the individual parameters. The NMB is reevaluated after changing the value of each parameter and holding the rest of the parameters at their mean values.

Figure D.1:

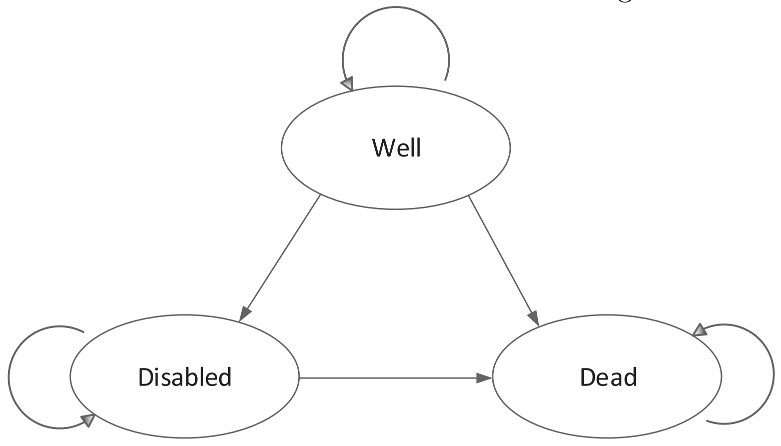
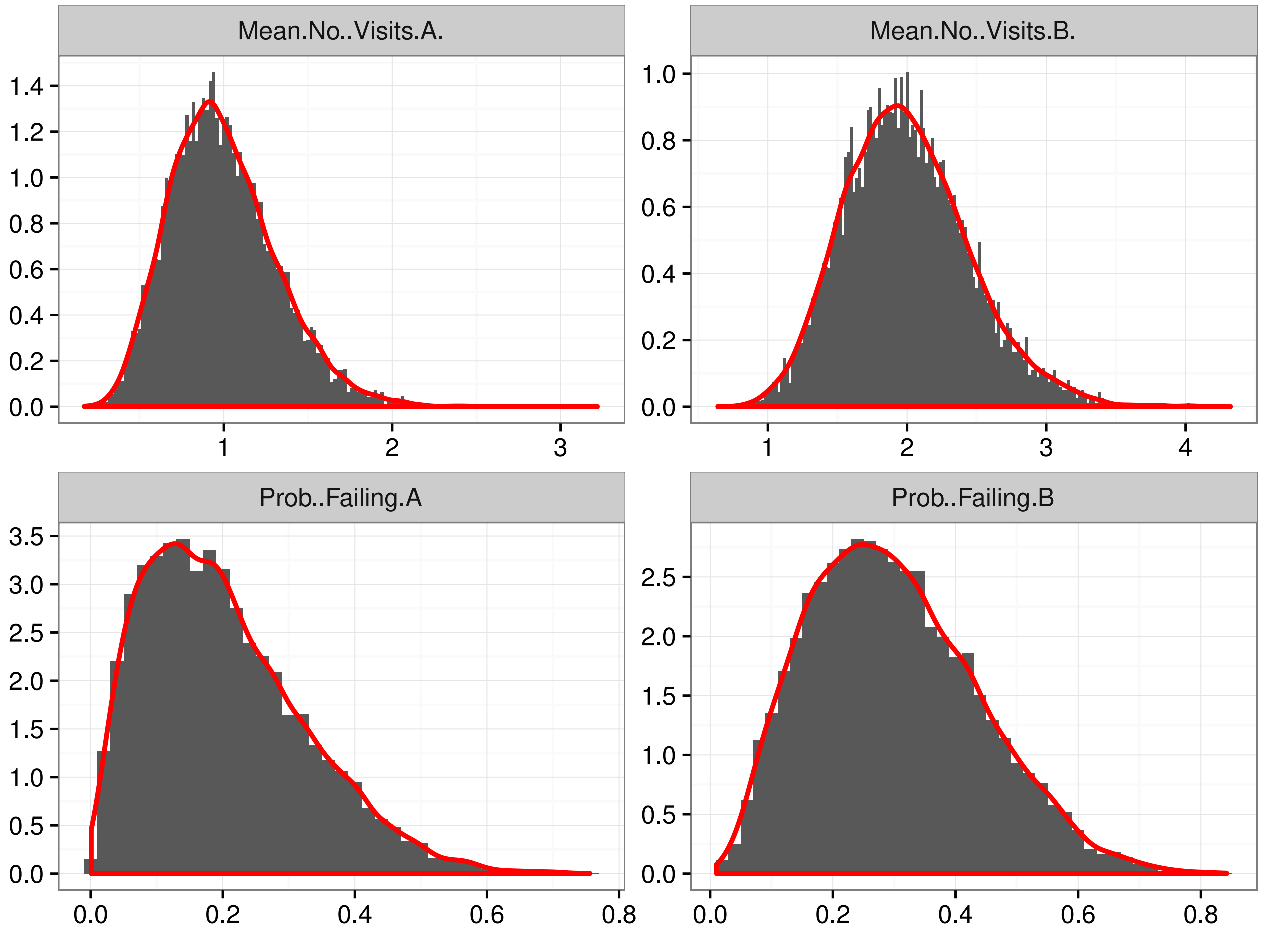
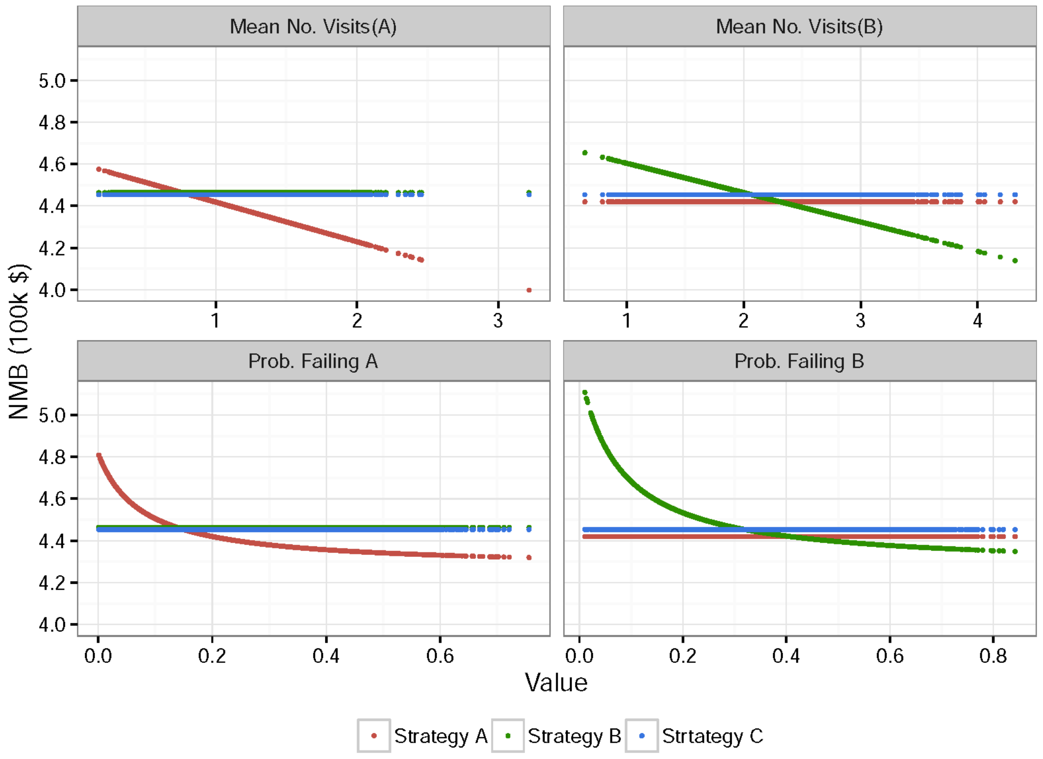


Figure D.2:



Mean.No..Visits.A.
Mean.No..Visits.B.
Prob..Failing.A
Prob..Failing.B

Figure D.3:


Mean No. Visits(A)
Mean No. Visits(B)
Prob. Failing A
Prob. Failing B
NMB (100k $)
5.0
4.8
4.6
4.4
4.2
4.0
1
2
3
4
0.0
0.2
0.4
0.6
0.8
Value
Strategy A
Strategy B
Strtategy C