# Practical 10. Missing data - SOLUTIONS

*Wednesday, 22 June 2022*

Lecture 10 PDF version

## Bivariate Normal model

The data are stored in a list format because the variables are non-balanced, in terms of sample size. In other words, the data for arm $t = 1$ contain fewer points than for $t = 2$. It would be possible to format this dataset using a `data.frame`, but this would mean having a single column for each variable (stacking together the two treatment arms) and adding a treatment indicator.

In any case, we can visualise the relevant information using usual R commands, e.g. `names` to check the naming of the variables and the `$` operator to access them.

```r
# Reads in the data list
data=readRDS("missing_data.rds")

# Checks the name of the variables
names(data)
```

```
[1] "c" "e" "n" "u"
```

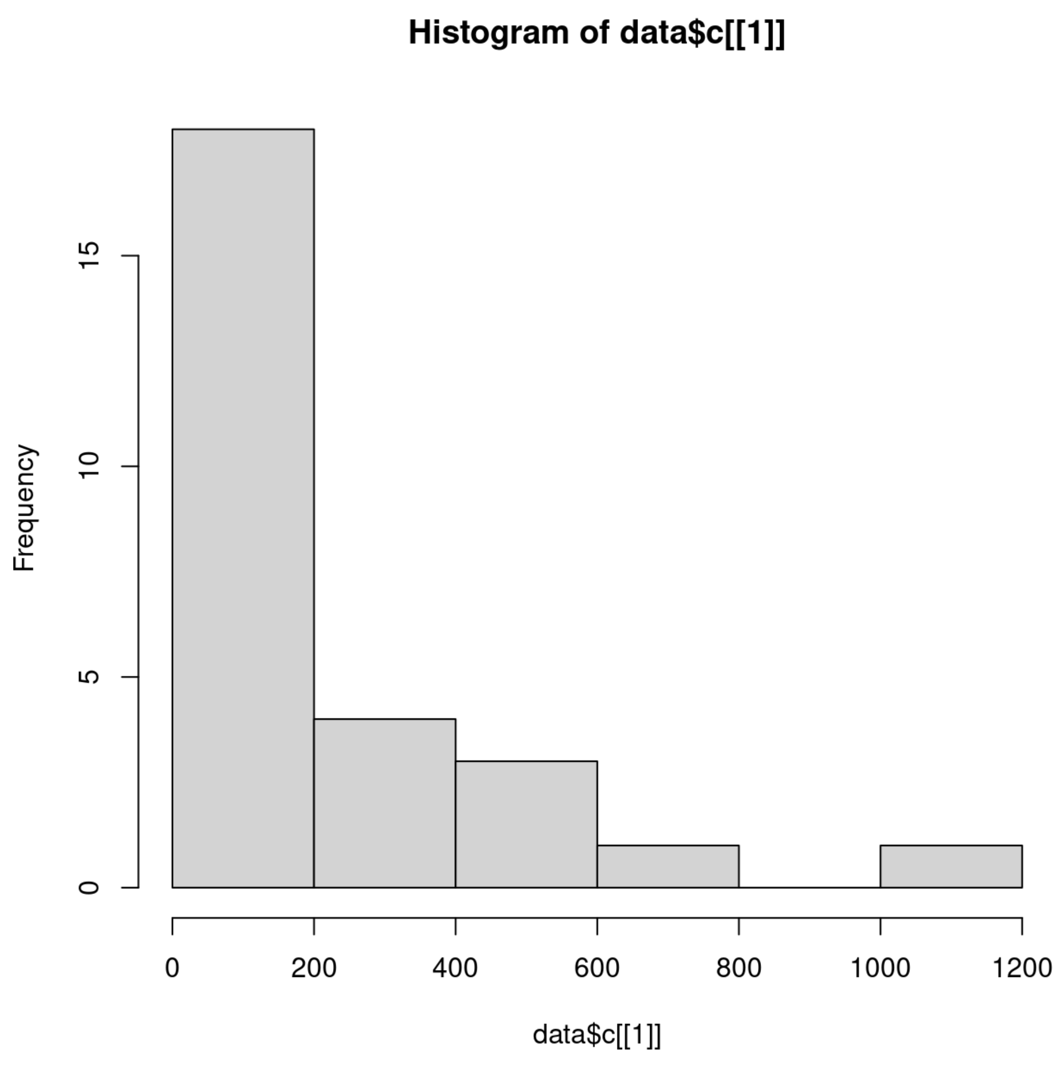
```r
# And shows some values
data$c
```

```
[[1]]
 [1]     NA    0.1     NA     NA     NA    0.1     NA     NA     NA  516.0
[11]     NA     NA     NA     NA  404.0     NA     NA  116.0  145.0  436.0
[21]  782.0     NA  145.0     NA     NA     NA    2.0     NA     NA    0.1
[31]     NA     NA     NA     NA  193.0    0.1     NA     NA   64.0     NA
[41]     NA     NA     NA     NA     NA 1039.0     NA    2.0     NA     NA
[51]     NA     NA    0.1     NA  246.0     NA     NA     NA  141.0  400.0
[61]  116.0    0.1     NA  360.0  281.0     NA    0.1     NA     NA     NA
[71]  107.0     NA     NA  123.0     NA

[[2]]
 [1]    NA    NA    NA 183.0    NA    NA    NA    NA   0.1 107.0    NA    NA
[13]    NA    NA 516.0    NA    NA    NA    NA    NA    NA    NA    NA    NA
[25]    NA   0.1    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
[37]   0.1    NA    NA 194.0  60.0 227.0    NA    NA    NA 266.0    NA    NA
[49]   0.1 169.0 346.0    NA    NA   0.1    NA    NA    NA 366.0    NA    NA
[61]    NA 380.5    NA    NA    NA    NA    NA    NA 268.0 123.0    NA    NA
[73]    NA 389.5    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
```

As is possible to see, the element `data$c` is itself a list. The data are clearly affected by many missing values (recorded in R as `NA`). We can also inspect the distribution of the variables using histograms, e.g. using the following commands.

```r
# Produces a histogram of the cost distribution in arm t=1
hist(data$c[[1]])
```



In general terms, it is important to make some preliminary analysis to evaluate the impact of missingness. For instance, if you had a large dataset with only a handful of missing values, then perhaps you could simply do a "Complete Case Analysis", which probably would not have a massive impact on your overall results. But if missingness is very prevalent, then this would imply the need for more sophisticated analyses, as well as the fact that the results would be by necessity to be taken with a rather large pinch of salt… We can summarise the proportion of missing data using the following command.

```
lapply(1:2,function(x) table(is.na(data$e[[x]]))/sum(table(is.na(data$e[[x]]))))
```

```
[[1]]

FALSE  TRUE
 0.36  0.64

[[2]]

    FALSE      TRUE
0.2261905 0.7738095
```

In this case, we can use the `R` command `table` to tabulate the data (on the effectiveness variable, named `e`) depending on whether they are `NA` (= missing) or not. We use the `lapply` function to perform the operation on both the elements of the list `data$e`. The code

```
table(is.na(data$e[[x]]))/sum(table(is.na(data$e[[x]])))
```

can be used to compute the proportion (instead of the absolute number) of missingness in each arm. As is possible to see, in this particular case, missingness is a big problem, as in the two arms there are respectively 64.00% and 77.38% of missing values.

We can then proceed to the modelling part. Firstly, we fit a bivariate Normal model for costs and effects; while this is not ideal because it clearly fails to accommodate the marked skewness in the data (as evidenced by the histogram above), it is a good starting point and at least allows us to account for potential correlation between costs and effects (and their missing mechanisms).

The model code is not too dissimilar to those used in Practical 4 (on individual level data), but it does require some modifications, to account for the missing data. We firstly assume a marginal Normal distribution for the effectiveness

$$e_i \sim \text{Normal}\left(\phi_{eit}, \psi_{et}\right),$$

where the individual mean response (QALYs) is modelled as a linear regression as a function of the baseline utility

$$\phi_{eit} = \alpha_0 + \alpha_1(u_{it} - \mu_{ut}).$$

In the BUGS code, we need to be careful in defining the Normal distribution, which requires the *precision* instead of the variance. For this reason, we code for example

```
eff1[i] ~ dnorm(phi.e1[i],tau.e[1])
```

where `tau.e[1]` is the precision, i.e. $\tau_{e1} = \psi_{et}^{-1}$. In addition, the linear predictor simply translates the regression for $\phi_{eit}$ — we notice here that we center the covariate $u_{it}$ for simplicity in the interpretation and ease of convergence (in this way, $\alpha_0$ is the overall mean QALY in the population).

The next thing to understand is that, because also $u_{it}$ is affected by missing values, we need to model it explicitly, even if it is used as covariate in the model. Thus, we need to include a suitable `BUGS} statement to define a probability distribution to represent it. We choose for simplicity the same form as the QALYs $e_{it}$ and thus specify a Normal distribution depending on a mean $\mu_{ut}$ and a *precision* $\tau_{ut}$.

Finally, we can model the conditional distribution of the costs, given the effectiveness variable. The assumption is a model

$$c_{it} \sim \text{Normal}(\phi_{cit}, \psi_{ct}),$$

where the mean is specified as a linear predictor

$$\phi_{cit} = \beta_0 + \beta_1(e_{it} - \mu_{et}).$$

Because we are again centering the covariate included in this model, we can simply characterise the population mean costs and benefits as $\mu_{ct} = \beta_0$ and $\mu_{et} = \alpha_0$, which we can then use in the economic analysis.

The `BUGS} code is replicated for the two treatment arms and it then specifies the prior distributions. The model parameters are $\boldsymbol\theta = (\boldsymbol\alpha, \boldsymbol\beta, \boldsymbol\sigma_e, \boldsymbol\sigma_c, \boldsymbol\mu_u, \boldsymbol\sigma_u)$, where:

- $\boldsymbol\alpha = (\alpha_0, \alpha_1) \overset{iid}{\sim} \text{Normal}(0, v)$, with $v$ a large variance (e.g. a precision of 0.00001);
- $\boldsymbol\beta = (\beta_0, \beta_1) \overset{iid}{\sim} \text{Normal}(0, v)$, with $v$ a large variance (e.g. a precision of 0.00001);
- $\boldsymbol\sigma_e = (\sigma_{e1}, \sigma_{e2})$ are the standard deviations for the effectiveness measures in the two treatment arms. We specify vague priors on the log-standard deviation scale, i.e. $\log\sigma_{et} \sim \text{Uniform}(-5, 10)$. This of course induces a prior on $\sigma_{et}$ and then on $\sigma^2_{et}$ and then on $\tau_{et}$;
- $\boldsymbol\sigma_c = (\sigma_{c1}, \sigma_{c2})$ are the standard deviations for the cost measures in the two treatment arms. We specify vague priors on the log-standard deviation scale, i.e. $\log\sigma_{ct} \sim \text{Uniform}(-5, 10)$. This of course induces a prior on $\sigma_{ct}$ and then on $\sigma^2_{et}$ and then on $\tau_{ct}$;
- $\boldsymbol\mu_u = (\mu_{u1}, \mu_{u2}) \overset{iid}{\sim} \text{Normal}(0, v)$, with $v$ a large variance (e.g. a precision of 0.00001);
- $\boldsymbol\sigma_u = (\sigma_{u1}, \sigma_{u2})$ are the standard deviations for the baseline utility in the two treatment arms. We specify vague priors on the log-standard deviation scale, i.e. $\log\sigma_{ut} \sim \text{Uniform}(-5, 10)$. This of course induces a prior on $\sigma_{ut}$ and then on $\sigma^2_{ut}$ and then on $\tau_{ut}$.

The BUGS code maps these assumptions directly and also adds some lines to derive the analytic form of the *conditional* variance and precision for the costs. These are useful, but are not necessarily fundamental parameters.

We can now run the model using OpenBUGS.

```
# Loads the package
library(R2OpenBUGS)

# Defines:
# 1. model file
filein="Normal_Normal.txt"

# 2. data list
datalist=list(N1=data$n[[1]],eff1=data$e[[1]],cost1=data$c[[1]],u1=data$u[[1]],
          N2=data$n[[2]],eff2=data$e[[2]],cost2=data$c[[2]],u2=data$u[[2]])

# 3. parameters to monitor
params<-c("mu.e","sd.e","alpha0","alpha1","beta0","beta1","Delta_e",
          "Delta_c","mu.c","sd.c","eff1","eff2","cost1","cost2")

# 4. number of iterations
n.iter<-10000

# 5. sets up initial values for crucial parameters
inits=function(){
   list(alpha0=rnorm(2),alpha1=rnorm(2),beta0=rnorm(2),beta1=rnorm(2),mu.u=rnorm(2))
}

# 6. runs the model
NN=bugs(data=datalist,inits=inits,parameters.to.save=params,
         model.file=filein,n.chains=2,n.iter=n.iter,n.thin=1,DIC=TRUE)
```

All is fairly straightforward; we first point R to the `.txt` file including the model code, then we define a datalist in the format that matches the name of the variables in the model code, then define the vector of parameters to monintor and select a suitable number of iterations for the MCMC procedure.

In this case, it is important to generate suitable initial values. For instance, if you did not create the function `inits} and let`BUGS} initialise the chains, the programme would return an error and cannot start the procedure. We set up the initial values for $\alpha, \beta, \mu_u$ from Normal(0,1) distributions, which ensures that the algorithm can sample reasonable starting points and run.

We can check that the model has reached convergence and that the estimates are reasonable. For instance, we could use the `print` method and apply it to the object NN, where we have stored the BUGS output. However, we have monitored many parameters and so this may be complicated to see. In cases such as this, it is probably more effective to use the element `$summary" to visualise the data, for instance using the following code

```
head(NN$summary[,c("mean","sd","2.5%","97.5%","Rhat","n.eff")])
```

```
                 mean         sd      2.5%     97.5%     Rhat n.eff
mu.e[1]   0.87119481 0.02232442 0.8268950 0.914805 1.000952 10000
mu.e[2]   0.91179054 0.02240187 0.8679975 0.956400 1.001053  9700
sd.e[1]   0.07969351 0.01191242 0.0605200 0.107200 1.000956 10000
sd.e[2]   0.08954983 0.01636157 0.0641200 0.127800 1.000912 10000
alpha0[1] 0.87119481 0.02232442 0.8268950 0.914805 1.000952 10000
alpha0[2] 0.91179054 0.02240187 0.8679975 0.956400 1.001053  9700
```

which would simply show the first few rows of the overall summary table for a selected set of columns (those displaying the mean, sd, 2.5- and 97.5% quantiles, as well as the convergence statistics). We could also specify the parameters for which we want to see the summary statistics, for instance using the following code

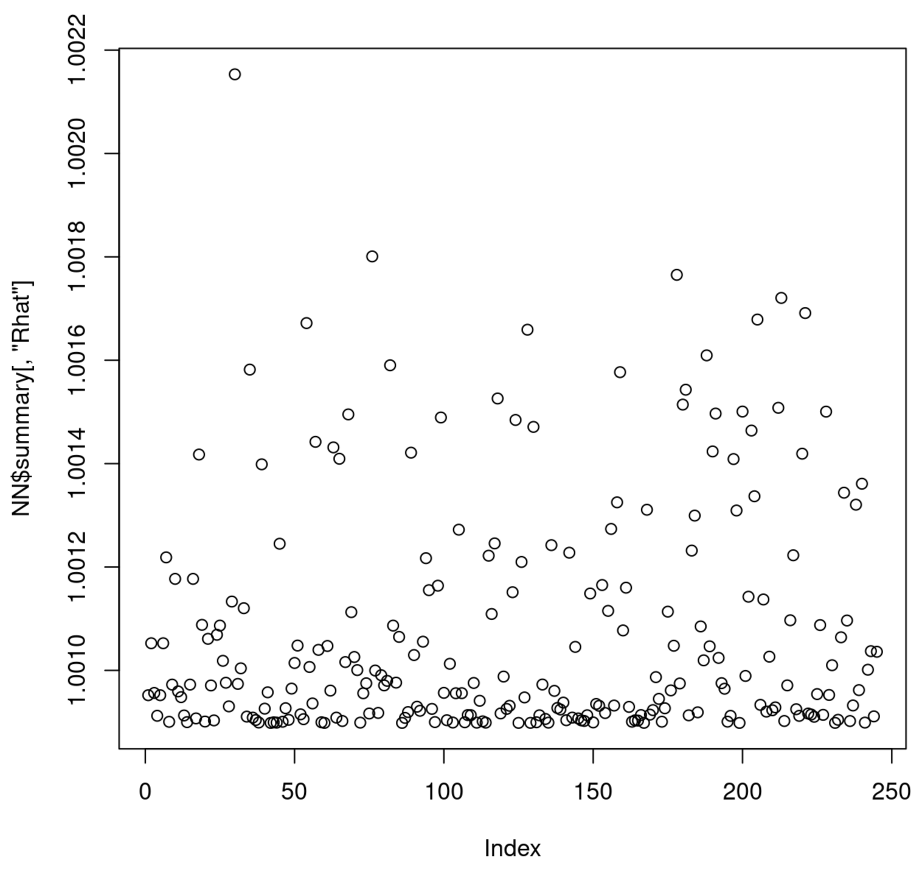
```
NN$summary[grep("alpha",rownames(NN$summary)),
           c("mean","sd","2.5%","97.5%","Rhat","n.eff")]
```

```
               mean         sd      2.5%    97.5%     Rhat n.eff
alpha0[1] 0.8711948 0.02232442 0.8268950 0.914805 1.000952 10000
alpha0[2] 0.9117905 0.02240187 0.8679975 0.956400 1.001053  9700
alpha1[1] 0.7912292 0.15647962 0.4809875 1.091025 1.001219  4700
alpha1[2] 0.2751874 0.08561333 0.1061975 0.443105 1.000901 10000
```

to show the selected summary statistics for all the nodes whose name contains the keyword `alpha` (this is done using the `grep` function — see `help(grep)` in your `R` terminal, for more details).

Another way to simply check convergence when the number of model parameters is very large is to plot the output for both $\hat{R}$ and $n_{eff}$ for all parameters. We can do this simply using the following command

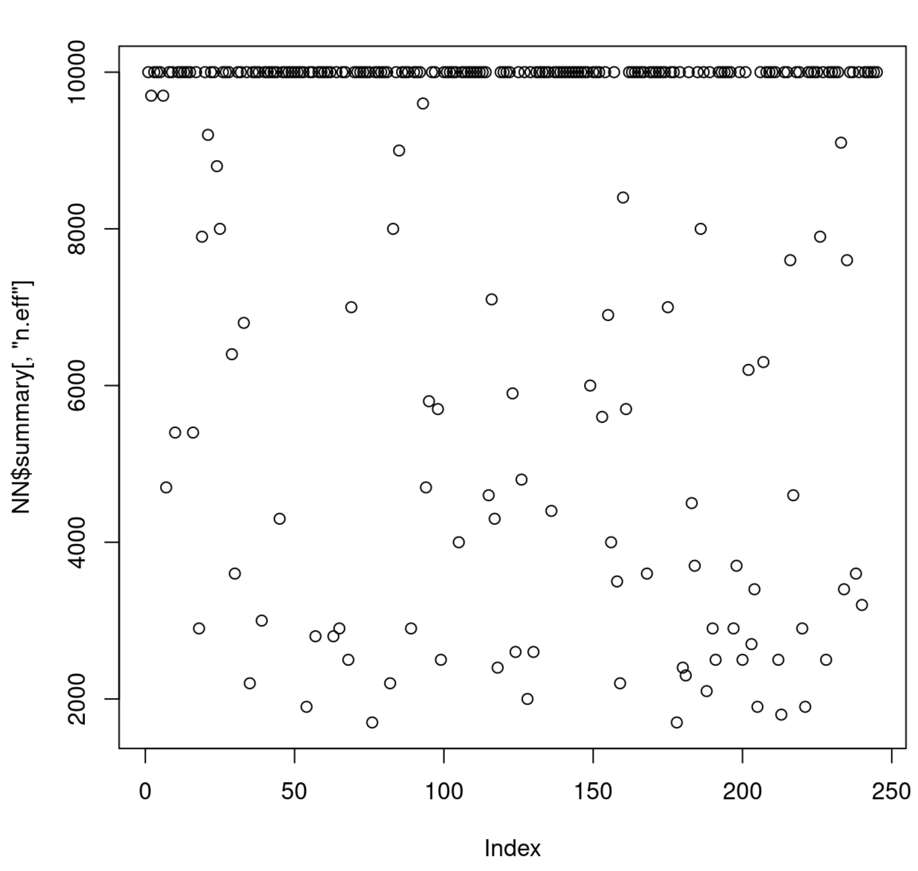
```
plot(NN$summary[,"Rhat"])
```



to check that the value of $\hat{R}$ is below the arbitrary threshold of 1.1 for all nodes. If this is the case, then the model has reached convergence; if not, we can investigate further and check which node has not converged yet. Of course the graph can be annotated and made prettier, but this crude version can still be very helpful.

Another similar graphical representation can be made to check the number of effective samples obtained using the MCMC procedure.

```
plot(NN$summary[,"n.eff"])
```



As is possible to see here, most of the nodes have a value that is close to the nominal sample size, indicating virtually no issues with autocorrelation.

Once we are satisfied about the output of the Bayesian model, we can feed it to BCEA to perform the economic analysis.

```
# Extracts the simulated values for the population mean effectiveness and costs
e=cbind(NN$sims.list$mu.e[,1],NN$sims.list$mu.e[,2])
c=cbind(NN$sims.list$mu.c[,1],NN$sims.list$mu.c[,2])

# Post-processes the results using BCEA
library(BCEA)
```

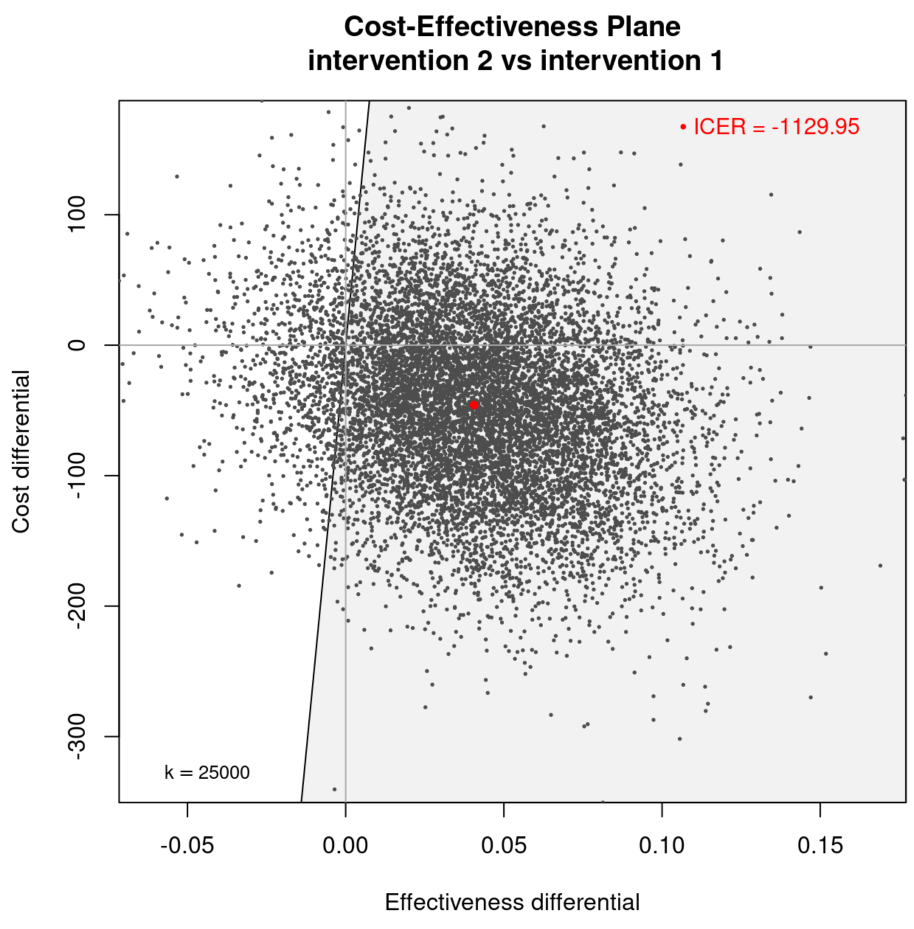
```

Attaching package: 'BCEA'
```

```
The following object is masked from 'package:graphics':

    contour
```

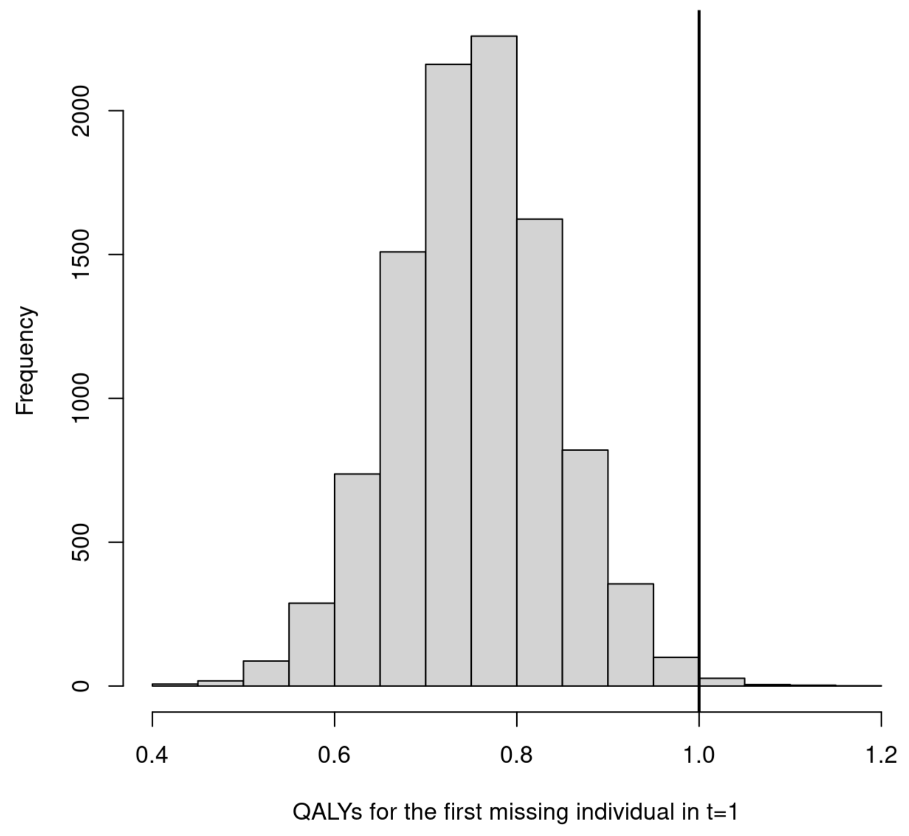
```
CEA_NN=bcea(e=e,c=c,ref = 2)
# Shows the C/E plane
ceplane.plot(CEA_NN)
```



Notice that we first create suitable matrices with the simulations for the two main variables $(\mu_{et}, \mu_{ct})$; we extract these simulations from the object NN — in particular, they are stored in the two lists `$sims.list$mu.e` and `$sims.list$mu.c`, respectively. Each of these two objects has dimension (10000, 2), i.e. 10000 rows (simulations) and 2 columns (treatment arms). We use the R function `cbind` to “bind” together these values, so that the resulting objects `e` and `c` are matrices, as required by BCEA.

Because we have monitored the variables `eff1,eff2,cost1,cost2` in the object NN, we can also check what the model is doing in terms of imputing the missing values. As suggested in the script, the easiest way doing so is by plotting the estimated posterior distribution, which we do using the following commands.

```
hist(NN$sims.list$eff1[,1],xlab="QALYs for the first missing individual in t=1",main="")
abline(v=1,lwd=2)
```

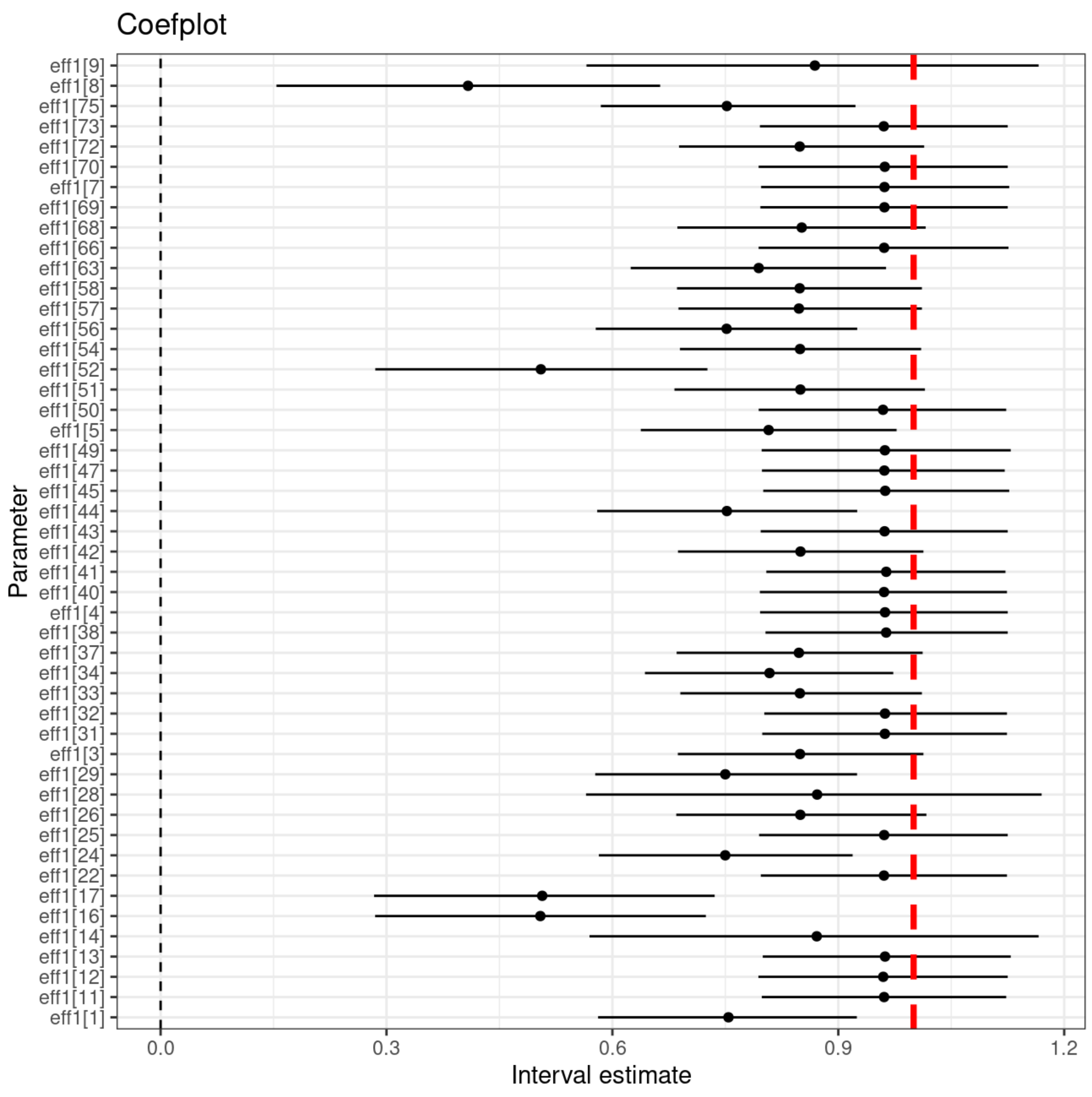


```
sum(NN$sims.list$eff1[,1]>1)/NN$n.sims
```

```
[1] 0.0036
```

The graph shows the posterior for `eff1`, the measure of effectiveness in treatment arm $t = 1$. As is possible to see, about 0.360% of the simulated values above the theoretical upper limit of 1 — which is of course not reasonable. Another way of visualising this feature of the model output is by using the specialised plotting function `coefplot` (which is available from the `R` package `bmhe` — which you would need to install), to produce a graph such as the following.

```
bmhe::coefplot(NN,parameter="eff1") + geom_vline(xintercept=1,linetype=2,size=1.3,col="red")
```



For each of the 48 individuals affected by missing QALYs in $t = 1$, this shows a description of the posterior distribution (imputed values). The dots represent the means, while the lines extend over the 95% credible interval. As is possible to see, there are quite a few individuals whose imputed values distributions expands above 1, although the means are all below this threshold.

## Beta-Gamma model

The Beta-Gamma model is theoretically more appropriate, because it correctly recognises the natural range of the two variables — the interval [0-1] for the QALYs and the open positive real line $(0, \infty)$ for the costs. Thus, it seems a more suitable modelling structure for this problem.

In general terms, the model code is not much more complicated than the one for the bivariate Normal — the main complication is that the regression models for the individual average QALYs and costs cannot be constructed as linear predictor. Instead, we need to rescale the average QALYs (defined in [0-1]) onto $(-\infty, \infty)$, which we do using a logistic regression

$$\text{logit}(\phi_{eit}) = \alpha_0 + \alpha_1(u_{it} - \mu_{ut}).$$

This implies that the overall average effectiveness measure $\mu_{et}$ is just a rescaled version of the intercept $\alpha_0$ (given that we are centering the covariate $u_{it}$). In other words, we need to take the inverse logit transformation and define

$$\mu_{et} = \frac{\exp(\alpha_0)}{1 + \exp(\alpha_0)}.$$

Similarly, we need to rescale the conditional regression for the average costs $\phi_{cit}$ as a function of the centered effectiveness, which we can do using a log-linear regression

$$\log(\phi_{cit}) = \beta_0 + \beta_1(e_{it} - \mu_{et})$$

and again the overall average cost is obtained by rescaling the intercept $\beta_0$ as

$$\mu_{ct} = \exp(\beta_0).$$

In fact, the main complication in the BUGS script is in the parameterisation of the Beta and Gamma distributions. BUGS requires that the Beta distribution is defined in terms of two scale parameters and thus we code $e_{it} \sim \text{Beta}(a_{ti}, b_{ti})$ for each intervention arm and individuals. However, we need to actually model the individual mean $\phi_{eit}$ and so we need to link this with $(a_{ti}, b_{ti})$. We can use the properties of the Beta distribution and derive that

$$a_{it} = \phi_{eit}\left[\frac{\phi_{eit}(1-\phi_{eit})}{\psi_{et}} - 1\right] \qquad \text{and} \qquad b_{it} = (1-\phi_{eit})\left[\frac{\phi_{eit}(1-\phi_{eit})}{\psi_{et}} - 1\right],$$

where $\psi_{et}$ is the variance for the Beta distribution (which is coded as `ss.e` in the model file). By modelling the mean $\phi_{eit}$ (through the priors induced on the parameters $\boldsymbol\alpha$) and standard deviation $\sigma_{et}$, we then imply a prior on $(a_{it}, b_{it})$.

A further complication is given by the fact that, because of the mathematical properties of the Beta distribution, $\sigma_{et}$ is constrained in its range by the value taken by the overall mean $\mu_{et}$; for this reason, we define the prior as Uniform in the range $(0,).

As for the Gamma distribution, we need to follow a similar strategy: the "original scale" parameters (which are required by BUGS) are the shape and rate say $(\zeta_{it}, \lambda_{it})$; however, these can be related to the mean and variance using the mathematical properties of the Gamma distribution as

$$\text{shape}_{it} = \zeta_{it} = \frac{\phi^2_{cit}}{\sigma^2_{ct}} \qquad \text{and} \qquad \text{rate}_{it} = \lambda_{it} = \frac{\phi_{cit}}{\sigma^2_{ct}}.$$

Again, priors on $\boldsymbol\beta$ and $\sigma_{ct}$ (which in the model code is indicated as `sd.c[t]` and is given a truncated $t$ distribution, to provide a continuous distribution on the positive real line) induce suitable priors for the shape and rate.

The interesting point of this model is that because of its relative complexity, OpenBUGS struggles to determine suitable initial values from which to start the simulation. In fact, we have tested several configurations, providing initial values for all the relevant model parameters and still have failed to run the model. Conversely, JAGS is able to start and successfully estimate this model and data. The main reason for this crucial difference is probabaly to do with the specific version of the sampler used by the two pieces of software.

In other words, to be able to run the Beta-Gamma model, it is necessary to install JAGS (which is available and easy to install from https://sourceforge.net/projects/mcmc-jags/files/. Then we need to install the R package R2jags (which is the equivalent to R2OpenBUGS). Fortunately, the differences between JAGS and OpenBUGS are virtually none. Thus, we can simply run the model following the script and using the commands below:

```r
# Installs and R2jags
install.packages("R2jags")

# Then loads the package
library(R2jags)
```

(this is of course only necessary once). When the package is installed, we just need to load it into our workspace and then proceed with the commands.

```
# 1. specifies the model file
filein="Beta_Gamma.txt"

# 2. data list
datalist=list(N1=data$n[[1]],eff1=data$e[[1]],cost1=data$c[[1]],u1=data$u[[1]],
              N2=data$n[[2]],eff2=data$e[[2]],cost2=data$c[[2]],u2=data$u[[2]])

# 3. parameters to monitor
params<-c("mu.e","sd.e","alpha0","alpha1","beta0","beta1","
          Delta_e","Delta_c","mu.c","sd.c","eff1","eff2","cost1","cost2")

# 4. number of iterations
n.iter<-10000

# 5. reset the inits function
inits=NULL
```

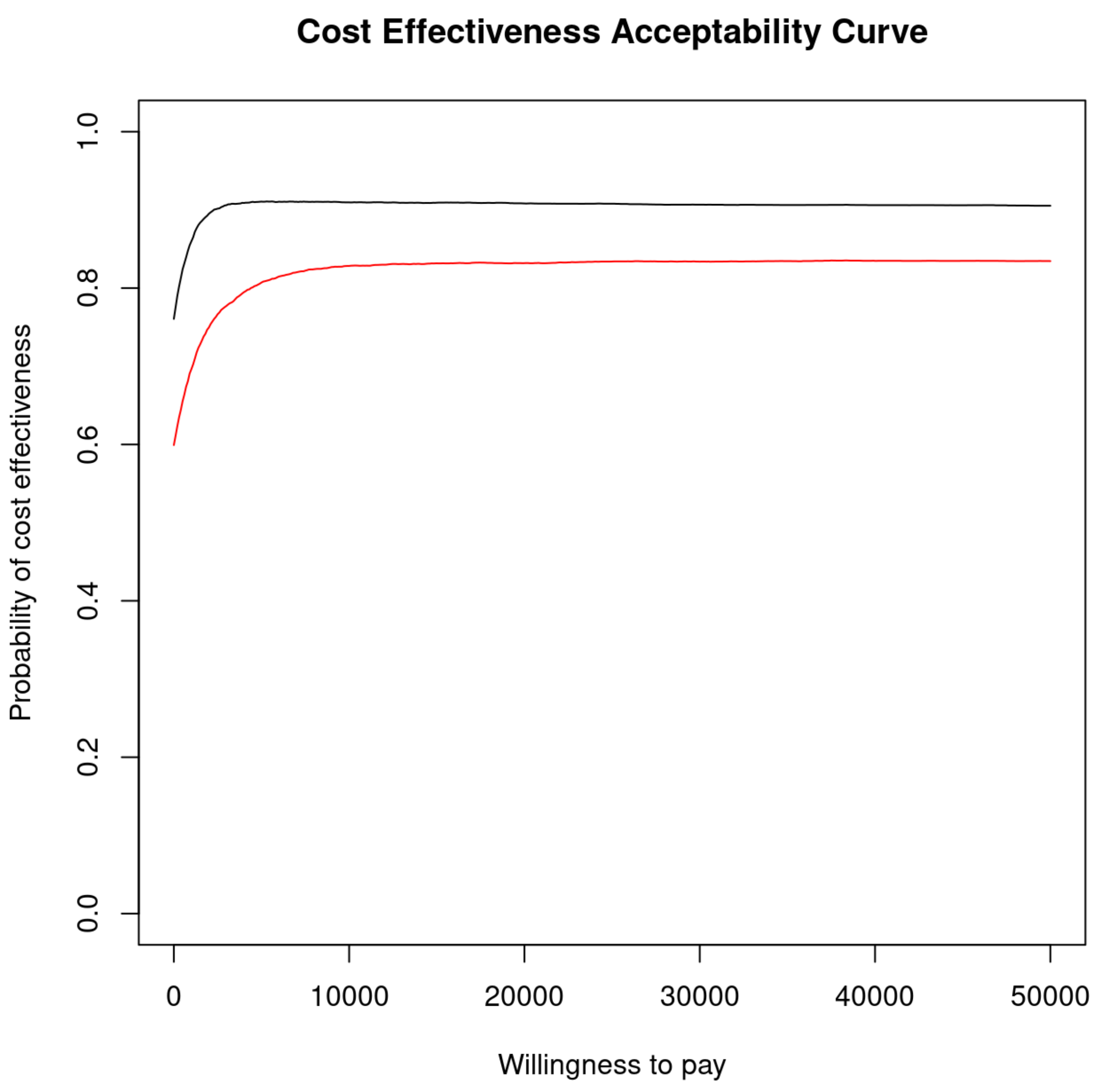
Finally, we're ready to run the main `jags` function to run the model:

`JAGS` compiles and initialises the model successfully and it does not even require us to specify the initial values (we still can do so, if we wanted to have full control, of course).

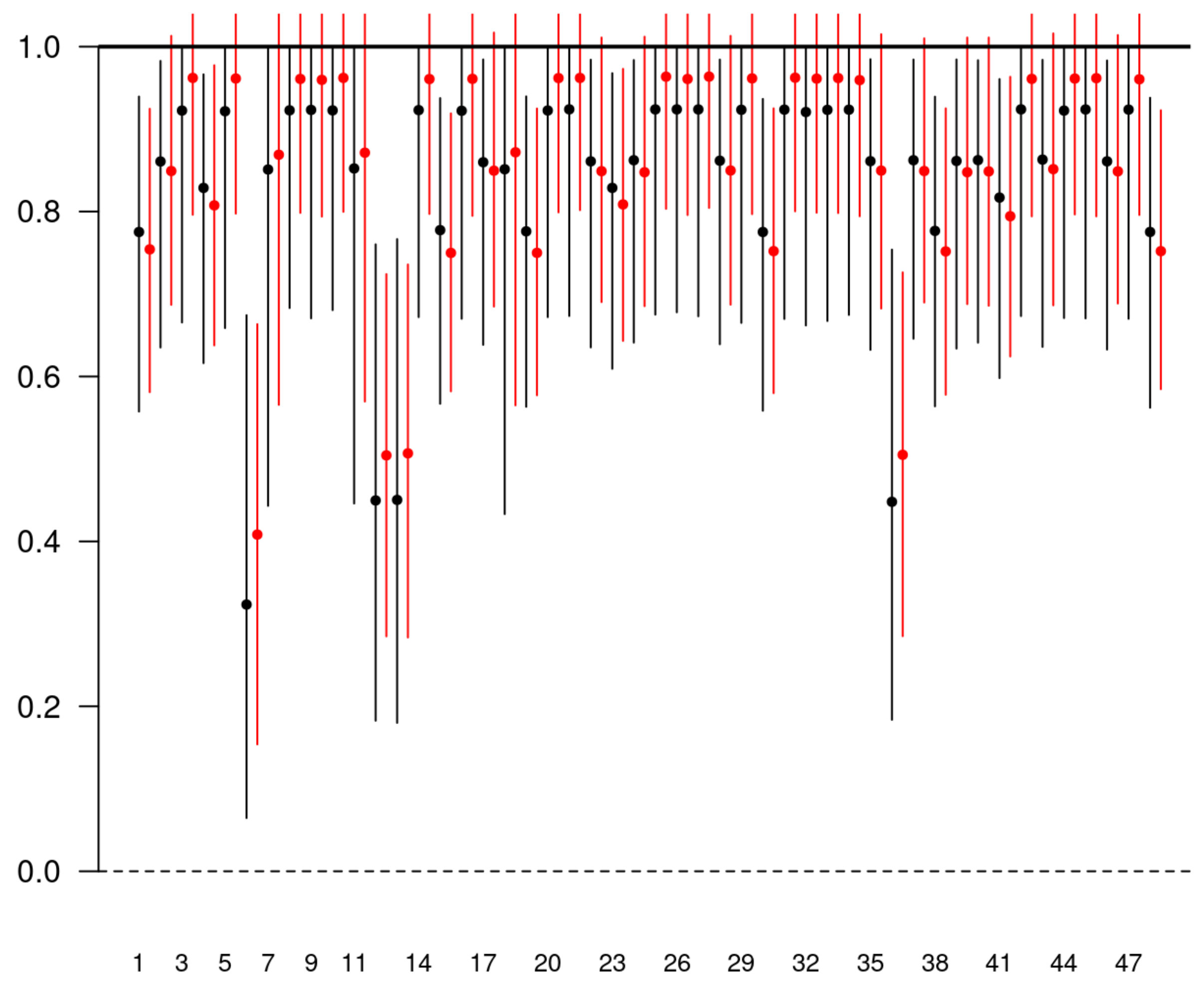
We can now replicate the analysis performed above to check and summarise convergence, as well as to feed the output of our Beta-Gamma Bayesian model to `BCEA`, to perform the economic analysis. Notice that the `JAGS` object `BG` has a slightly different structure and contains an "extra-layer" in comparison to its `OpenBUGS` counterpart `NN`}. As a result, in order to access the simulations, we need to go deeper inside the object and look into the element `$BUGSoutput$sims.list`.

```
e<-cbind(BG$BUGSoutput$sims.list$mu.e[,1],BG$BUGSoutput$sims.list$mu.e[,2])
c<-cbind(BG$BUGSoutput$sims.list$mu.c[,1],BG$BUGSoutput$sims.list$mu.c[,2])
CEA_BG<-bcea(e=e,c=c,ref = 2)

# Plots the CEAC
ceac.plot(CEA_NN)
points(CEA_BG$k,CEA_BG$ceac,t="l",col="red")
```



Interestingly, the results are different, depending on the model used for imputation of the missing data. The graph above shows the CEACs for the two model specifications and as is possible to see the Normal-Normal model somewhat overestimates the probability of cost-effectiveness. This is likely due to the higher mean QALYs associated with each individual, given the imputations exceed the theoretical range of 1. If we check the `coefplot` for the Beta-Gamma model, we now see that all the distributions are within the range [0-1], as should be.



In the graph we display the distributions for the Beta-Gamma (black) and Normal-Normal (red) models. A clear indication of the "pull" towards higher values that the bivariate Normal model shows is given by individual number 6. The red line is shifted upwards in comparison to the black one — this is because the Normal-Normal model does not know that it should not go above one (because the Normal distribution is unrestricted) and so it pulls up the estimate for this individual. This translate in slightly higher values for the QALYs, which in turn artificially increase the cost-effectiveness profile of the active intervention.