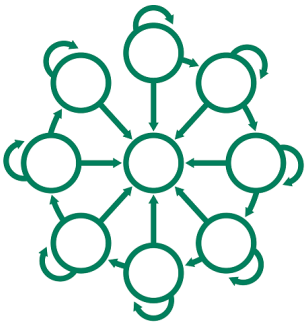


DECISION MODELLING FOR HEALTH ECONOMIC EVALUATION

Value of Informaton

Advanced Course Module 4

Jack Williams & Nichola Naylor

May/June 2021 Course

Overview

The purpose of this exercise is to show how the model you have already developed can be adapted to calculate the expected value of perfect information (EVPI). As a first step, the overall EVPI for the model is calculated in order to summarise the overall importance of uncertainty for decision-making. Secondly, the partial EVPI for individual parameters (or groups of parameters) can be calculated in order to examine the contribution of different parameters to the overall uncertainty.

The step-by-step guide below will take you through the following stages for calculating expected value of information for the overall decision problem and for the parameters of the decision model. There are 2 parts to this exercise:

Part 1: The following two scripts relate to this part of this exercise:

- "A4.3.2a_Value_of_Information_Part 1_Template.R"
- "A4.3.3a_Value_of_Information_Part 1_Solutions.R"

Additionally, the solution file from the previous exercise, and the file with ggplot functions, will be sourced:

- "A3.3.3a_Presenting_Simulation_Results_Part1_Solutions.R"
- "ggplot_CEA_functions.R"

The exercise will involve completing the following tasks:

1. Calculating per patient EVPI from simulation output
2. Calculating the effective population and population EVPI
3. Plotting EVPI as a function of the willingness to pay threshold

Part 2: We will be using three different scripts here

- "A4.3.2b_Value_of_Information_Part 2_Template.R"
- "A4.3.3b_Value_of_Information_Part 2_Solutions.R"
- "A4.3.3b_VoI_Model_Script"

The last template and solutions scripts will run the main model code from another pre-filled script ('A4.3.3b_VoI_Model_Script'). This allows you to call in the model function, and explore the key concepts of this module in a separate script.

In this part of the exercise you will:

1. Look at the adaptations to the model, to see how the model function has been developed to allow for parameters being provided as arguments in the function.
2. Work through an Expected Value of Partially Perfect Information (EVPPI) manually, to ensure that you understand the process involved in each step.
3. Look through the EVPPI analyses coded for each parameter (or group of parameters), and ensure you understand how the appropriate parameters are being selected.
4. Plot EVPPI results across a range of willingness to pay thresholds.

Across the whole exercise we will be using the data inputs and packages discussed in previous exercises. No new data and/or packages will be used.

PART 1: Step-by-step guide

(1) Calculating per patient EVPI from simulation output

In the "A4.3.2a..." template file you will see that we've begun the exercise by sourcing in the model we created in Exercise 3a and storing the simulation results as we did in exercise 3a for intended group of interest (in this case the function will run the model for females aged 60, unless other values for these are provided). Once sourced, you can see the same outputs we created in Module 3 are now available in the 'Environment' panel, e.g. "`CEAC`". *Note: this might take a few seconds depending on the capability of the device you are working on.*

Recall that, under a decision theoretic approach, it is the expected outcome that should guide the decision. That is, we will use the new prosthesis if it has an expected net benefit that is greater than that of the standard prosthesis. However, due to uncertainty, sometimes we will make the wrong decision and the losses associated with the wrong decision are the net-benefits forgone.

So we first need to estimate the EVPI for individual patients.

(i) Within the 'Estimating EVPI for individuals' section, first define a willingness-to-pay (WTP) threshold of your choice, to assess the EVPI (we show an example using £10,000 per QALY gained within the solutions).

(ii) Estimate the net monetary benefit (NMB) for SP0 and NP1 for each run using columns from the `simulation.results` data.frame, and your defined WTP. Once you have estimated the NMB for each model simulation (`nmb.SP0` and `nmb.NP1`), ensure that you have these in a data.frame (one column for each treatment) - `nmb.table`. [*We've coded this for you already, but do make sure you understand the calculations*]

(iii) Using the `apply()` function, estimate the the mean NMB for both treatments, across all simulations. This is the average NMB under current information. We can save this as `av.current`.

Note: We will be using the `apply()` *functions in several parts of this module. You can check the help file to make sure you understand how to use the function, but it essentially requires three arguments. First, a data frame, second, whether to look across rows or columns, and last, what function you wish to perform.*

(iv) Calculate the maximum NMB out of the two treatments under current knowledge, and call this (`max.current`), by running the relevant line of code we've written out for you.

(v) Next, to consider the decision making under perfect information, estimate the maximum NMB across treatments, for each model simulation performed (i.e. the NMB for the favoured treatment, at that WTP) and save this as `max.nmb`. This will be a vector of values, the same length as the number of simulations (in our example this would mean a vector of 1000 values). The average of all these maximum NMB values can be calculated using the `mean()` function, and saved as `av.perfect`. This represents the average NMB across all simulations, if the correct treatment option was chosen every time.

(vi) Finally, you can calculate the difference between the NMB from the preferred treatment across all simulations, versus the NMB if the best treatment was chosen across every simulation. This is the EVPI for each individual, and you can save this as `EVPI.indiv`. We've given some further hints in the template to help you through this, but remember that these results are all specific to the WTP defined above (and if you have chosen a very high WTP threshold, the EVPI may be 0!).

(2) Calculating the effective population

Of course, there will be more than one patient eligible for a particular treatment. Therefore, the individual patient EVPI must be inflated by the effective population, which is a function of the eligible patients per annum and the expected lifetime of the technology.

For the hip replacement example, we are going to assume an effective technology life of ten years with 40,000 new patients eligible for treatment each year. With a discount rate of 6% used in this case.

(i) Using the above information, set the population, number of years and discount rate in the template script within the 'ESTIMATING POPULATION EVPI' section.

(ii) We have now defined a discounted population sequence for you (the annual population, discounted each year, for the number of years the technology is assumed to apply for). Next, estimate the effective population by summing across this (i.e. summing across `population.seq`).

(iii) Estimate the population level EVPI by multiplying the effective population (`effective.population`) by the `EVPI.indiv` we calculated earlier on in the exercise.

(3) Calculating the population EVPI, using a function

Since EVPI is calculated using net-benefit and net-benefit is a function of the unknown ceiling ratio it is important to estimate EVPI as a function of the ceiling ratio. We therefore need to wrap up what we've done in step 1 into a function, and run it across different WTP values.

(i) Define a new `WTP.values` vector running from 0 to 50,000 in increments of 100.

(ii) Based on what we've done so far, fill in the blanks left in the function `est.EVPI.pop()`. *(Hint: use the calculations in step 1 to help you)*

(iii) Create a data frame (`EVPI.results`) to capture the WTP values and subsequent population EVPI results (the latter can be set to NA first).

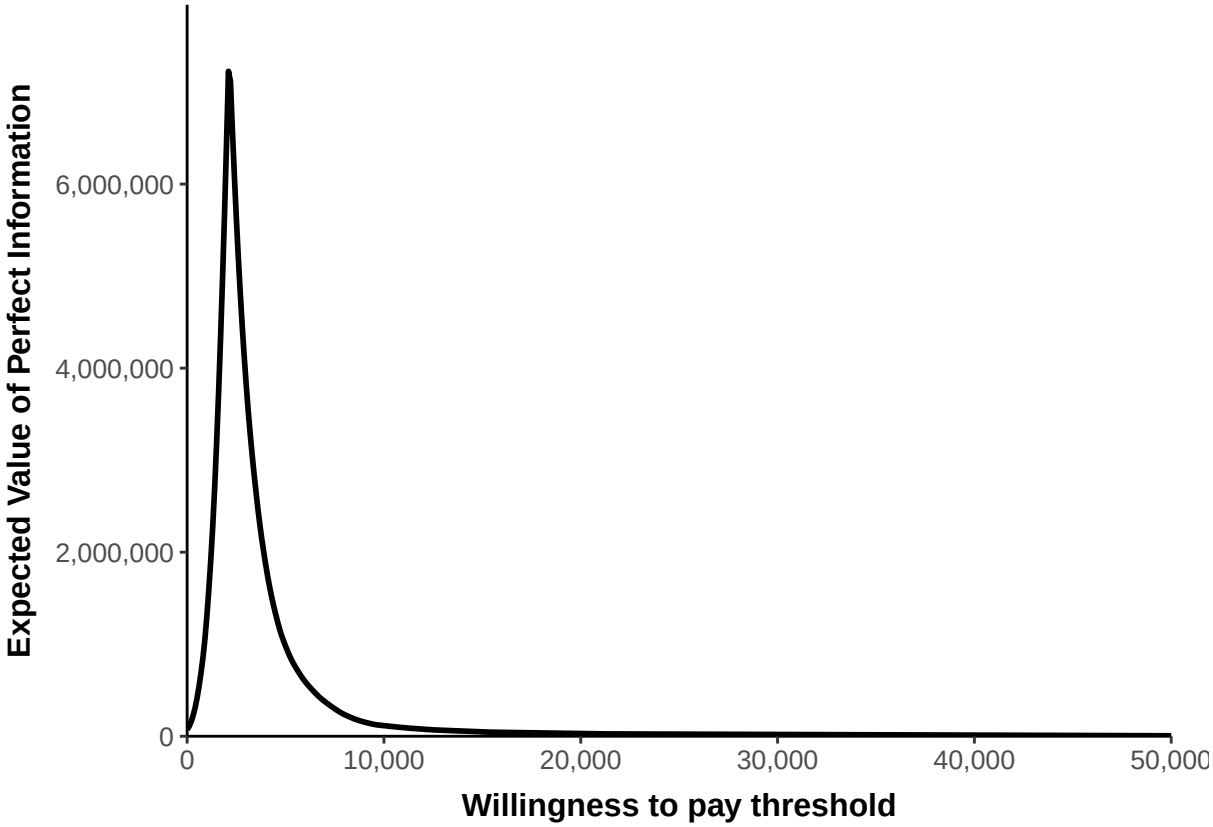
(iv) Fill in the rest of the for loop we've started to run the `est.EVPI.pop()` along different `WTP.values`. Remember that this function requires three arguments to be provided. The top of your `EVPI.results` should look similar to the below (although remember these won't be the exact same due to the sampling methods used!):

```
##   WTP      EVPI
## 1   0  71449.74
## 2 100 107349.45
## 3 200 155827.01
## 4 300 222245.51
## 5 400 304223.67
## 6 500 408855.58
```

Congratulations! You've completed your first EVPI in R.

Take a look at the results you've got. You can plot these results using the predefined ggplot functions given. *These don't need adapting from the code already given within the template file, just run the code and check out the plots!*

The population EVPI results should look similar to the plot below. Notice the peak in EVPI – where does this occur?


Expected Value of Perfect Information
6,000,000
4,000,000
2,000,000
0
0
10,000
20,000
30,000
40,000
50,000
Willingness to pay threshold

PART 2: Step-by-step guide

The overall EVPI for the model is a useful upper limit on returns to future research. However, of crucial importance is which particular parameters (or groups of related parameters) are most important in terms of having the greatest value of information. Partial EVPI - also known as Expected Value of Partially Perfect Information (EVPPI)- approaches are designed to look at just that. They work by looking at the value of information of the remaining parameters of the model if we assume perfect information for the parameter of interest.

In this practical session we want to look at the EVPPI of six groups of parameters (where parameters are related, such as utilities or survival parameters, it makes sense to consider their partial EVPI together):

- Relative risk of revision for new prosthesis 1 compared to standard
- Operative mortality following primary and revision surgeries
- Re-revision risk
- Survival parameters (the lamda and gamma parameters derived from the hazard function outputs)
- Costs (in this case the only probabilistic cost is cost of revision surgery)
- Utilities (utility values from successful and revision surgeries)

In this practical you will first look at the the new model function `model.THR.voi()` that is loaded in from a separate script. The adapted function is described in detail in the next section, but essentially it allows us to pass specific parameters values into the model (rather than these values being sampled within the model). This gives control over which values are selected and used in the model. This is needed for the inner and outer loops of the EVPPI calculations.

(1) Loading the appropriate model script and function

Begin by loading up the template file for Part 2 of this exercise ("A4.3.2b_Value_of_Information_Part 2_Template.R"). First, you can run the 'A4.3.3b_VoI_Model_Script' using the `source()` function. This is an adapted version of the model, which includes a function to run the Markov model - which we have labelled `model.THR.voi()`. Take a look at this script and the function. We have annotated the script to highlight changes that have been made. You will notice that in this version of the model, rather than probabilistic parameters being sampled and assigned within the model function (i.e. sampling one each time the model function is run), all of these parameters are now sampled and stored outside of the model function. This tends to be a quicker (more efficient) way of running model simulations, but it does mean some additional complexity.

Since the probabilistic parameters are no longer being generated within the model function, we have to adapt the function to take these probabilistic values as arguments in the function. This means that these parameters are provided as inputs for the model function. Notice how the `model.THR.voi()` function requires data which contain our sampled probabilistic values (see in particular the 'PROBABILISTIC PARAMETERS' section. The model function will then use the parameters provided to it to generate the transition matrices and the Markov trace, and estimate the costs and QALYs for the two treatments. The remainder of the model function is the same as you have seen previously, except in some areas we have tried to speed up some of the code by avoiding loops. *(For those interested in more efficient coding, check how the transition matrices are now calculated using vectorisation rather than using a loop. This vectorisation works because the length of time varying parameters (as vectors) is the same as the number of matrices in the array, so R knows that the first value is for the first matrix, and so on). We have included as additional material for this module "Vectorisation_example.pdf" for further information related to this.*

To ensure you understand the data provided to the model function, consider the following. The model samples 10,000 utility values for each parameter:

```r
## This provides the dimensions of the data frame
dim(state.utilities.df) # (10000 rows and 5 columns)
```

```
## [1] 10000     5
```

```r
head(state.utilities.df) ## Now let's take a look at the first 6 rows
```

```
##   uprimary uSuccessP uRevision uSuccessR udeath
## 1        0 0.8458858 0.3506298 0.7457785      0
## 2        0 0.8636910 0.2791140 0.7331976      0
## 3        0 0.8467843 0.2936025 0.7195536      0
## 4        0 0.8395898 0.2924053 0.7210959      0
## 5        0 0.8819635 0.2972028 0.7410597      0
## 6        0 0.9078584 0.3492896 0.7872781      0
```

```r
# Now we will select one row of sampled values from data.frame,
#  in this case the first row:
state.utilities.df[1,]
```

```
##   uprimary uSuccessP uRevision uSuccessR udeath
## 1        0 0.8458858 0.3506298 0.7457785      0
```

This data can then be passed to the model. You can also check the other parameters that are provided to the `model.THR.voi()` function too. You can check the bottom of the code in the 'A4.3.3b_VoI_Model_Script' R script and see an example of the model function being performed (the code is also provided below).

```r
i = 1
model.THR.voi(RR.vec[i], omr.df[i,],  tp.rrr.vec[i],
              survival.df[i,],c.revision.vec[i],
              state.utilities.df[i,])
```

Check each the of the data input passed through the model. Notice that if you re-run this code (with i = 1 each time), the same parameter values are provided to the model, and therefore the model results will be the same since the input data remain the same.

In order to consider the results of the model using many different samples of data, we can select the appropriate samples of probabilistic values and run the model each time, in a loop. You don't need to replicate this code, it is provided below as an example, but if you wanted to performed a PSA then the following code would perform and save the results of 10,000 probabilistic analyses (this is just an example, but you could run this code to check if you wanted). For each value of `i` given in the loop, a new selection of sampled data for each parameter is provided to the model, which then produces a different set of model results:

```r
psa.results.test <- matrix(0, nrow = sim.runs, ncol = 4)

for(i in 1:sim.runs) {
  psa.results.test[i,] <- model.THR.voi(RR.vec[i], omr.df[i,],  tp.rrr.vec[i],
                                        survival.df[i,],c.revision.vec[i],
                                        state.utilities.df[i,])
}
```

Selecting the appropriate data within each model simulation becomes very important for the EVPPI loops, since each loop will take different sampled values depending on the parameter, or groups of parameters, for which we have perfect information.

(2) **EVPPI Example walkthrough**

Before we perform the EVPPI analyses, we will show a walkthrough of the calculations, using the inner and outer loops and selecting the appropriate parameter values. Once you understand the calculations being performed, the next sections will set up the full EVPPI calculations, with additional complexity. Go to the 'EVPPI example walkthrough' section of the code in the template file to start this part of the exercise:

(i) First, create the `evppi.results.SP0.test` matrix. We will just use one willingness to pay threshold (£2,200 for this first example), so this matrix will only use one column for each treatment (note that in the full EVPPI analyses we will consider a range of WTP values). You can use this matrix to create a replicate, for the NP1 treatment (`evppi.results.NP1.test`).

(ii) Next, create a matrix to store the results of the inner loop `inner.results` (i.e. the probabilistic analysis for all other parameters, after the partial parameter has been sampled).

(iii) Define a WTP value (we've set this to £2,200 this value is chosen to give a high EVPI for this part of the exercise!)

(iv) Remember, from looking through the model script above, that we need to pass the appropriate parameters into the model function. Familiarise yourself with the input parameters sampled in the '`A4.3.3b_VoI_Model_Script`' script that you ran earlier. We have provided these in the script for you, and have also provided the code to show you how these parameters can be selected and given to the model (see below). **It is very important that you understand how this data is being selected and passed to the model function for you to understand the rest of the exercise**

In the below example, we have selected the first row (or value) for each of the 1000 samples, for each parameter. In the code, you should check the results using another set of sampled parameters.

```
model.THR.voi(RR.vec[1], omr.df[1,],  tp.rrr.vec[1],
              survival.df[1,],c.revision.vec[1],
              state.utilities.df[1,])
```

```
##  cost.SP0 qalys.SP0  cost.NP1 qalys.NP1
## 485.96386  14.54137 598.70893  14.59520
```

One you understand how the parameters are passed into the model, we can start to look at how the EVPPI loop works. This task may seem repetitive, but the aim is to get a full appreciation of the approach before running the full code later, which will do the same thing but many more times, and across a range of willingness to pay values.

We will use the letter 'a' to indicate the outer loop number, and the letter 'b' to indicate the inner loop number. We will use the relative risk parameter first and evaluate the value of partial perfect information for this parameter (the sampled parameters are stored in the `RR.vec` vector). However, since we are uncertain as to the true value of the relative risk, therefore the first step is to select a value from the samples of `RR.vec`.

(i) First, define 'a' as being equal to 1 (i.e. the first outer loop simulation). Next, select the value of `RR.vec` using the outer loop as the indicator as to which value should be used. Use 'a' to select the first RR.vec value. You do not need to assign this yet, but be aware that it will go into the model function, within the inner loop. *(Note: the use of 'a' and 'b' in selecting the appropriate parameters is important to ensure the correct data are provided within EVPPI simulations)*

(ii) Note that the value for the relative risk will be constant within the inner loop (this is now the 'perfect information for the parameter'. Next, we can run 100 model simulations within the inner loop, and ensure that all other probabilistic parameters in the model are sampled. To do this, all other parameters

inside the inner loop will be selected using 'b' so that they will change for each inner loop, whilst the relative risk value remains fixed (in each inner loop, because a is fixed, and equal to 1). To ensure that the relative risk remains fixed, whilst the other parameters are sampled inside the loop, the following code should be used to run the model.

```
for(b in 1:100){
  inner.results[b,] <-  model.THR.voi(RR.vec[a], omr.df[b,],  tp.rrr.vec[b],
                                      survival.df[b,],c.revision.vec[b],
                                      state.utilities.df[b,])
}
```

The results of one model run will look like this:

```
model.THR.voi(RR.vec[a], omr.df[b,],  tp.rrr.vec[b],
              survival.df[b,],c.revision.vec[b],
              state.utilities.df[b,])
```

```
##  cost.SP0 qalys.SP0  cost.NP1 qalys.NP1
## 514.42941  14.81918 618.77811  14.86146
```

You will see that each model simulation is then saved in the `inner.results` matrix. This shows the uncertainty in decision making whilst the relative risk parameter is known (i.e. perfect information).

(iii) Next, calculate the mean NMB across the 100 simulations of the inner loop for both treatments (note that this uses the WTP value defined above). Then save these in the evppi.results matrix for both treatments.

(iv) Now you need to repeat steps (i) to (iii), but selecting a new relative risk parameter in the outer loop. Change the value of 'a' to 2, and then rerun the code.

(v) Do this 10 times, so that you have considered 10 samples of the parameter of interest for the EVPPI analysis (the relative risk parameter in this case)

(vi) Once you have done this, you should be able to view the NMB values for each outer loop. *(Hint: the NMB should be the same for the comparator across model runs, since the relative risk parameter selected does not influence the standard prosthesis)*

Next, you will calculate the value of information in the same way as we did for the EVPI in the previous section. First though, run the code to add our results into a data.frame (`evppi.df`).

(vii) We first calculate the average NMB across the outer loops for both treatments. Whichever treatment has the maximum NMB, on average across the simulations, is preferred, so we take the maximum value NMB (from the two treatments). We do this by first creating a data.frame of the NMB for both treatments, then use the `apply()` function. The code is provided for you. This is saved as `current.info`. We then use a similar process to define `perfect.info`, but this time calculating for the scenario of perfect information (note that this is based on perfect information of the relative risk parameter only), we want to take the maximum NMB of either treatment across each model simulation. This means that for every value of the relative risk parameter, the correct treatment (with the highest NMB) would be selected. Then take the mean value of these, to give the average NMB across the simulations, when choosing the correct treatment at each simulation. *Check the ?apply documentation for a reminder of what goes into the apply() function, and if you are confused about having the mean and maximum within and outside of the apply functions for the different calculations, then try to separate these out to understand what is happening, or refer back to Part 1 of this exercise to see what we did.*

(viii) The EVPPI is the difference between these two values. You can subtract the current NMB from the NMB with perfect information for the parameter to calculate EVPPI.

Note that in this walkthrough, the EVPPI is estimated per individual (since the model using a cohort size of 1). Also, the results shown are only evaluating the EVPPI at one WTP value. Later on, you will see that we can evaluate the EVPPI across a range of WTP values.

(3) **Loading the parameters and model functions**

Now we will walk through the code to perform EVPPI analyses for six sets of model parameters, and evaluate the EVPPI across a range of WTP thresholds. Before you start, there are parameters that need to be defined before you can complete the EVPPI analyses. Go to the 'EVPPI Model runs' section of the code to begin.

(i) Define the number of inner and outer loops to run *(Note that the number here will need to be equal to or lower than the number of samples in the model script).* You can set these to 100 each to allow the simulations to be run fairly quickly, although this will depend on your computer!

However – and this is very important – accurate estimation of partial EVPIs takes many more runs than this – and we would recommend at least 1000 inner and 1000 outer loops. This is one of the advantages of using R, as doing this across each parameter in Excel would take a very long time (if it even manages to finish!). This is also one of the reasons that we should be aware of how we are coding our model in R, since a model with lots of loops and inefficient code will take a lot longer to run. Whilst this is less important when running a PSA or EVPI analysis, it will begin to have a big impact when using the nested loops used in EVPPI analyses (for example, 1000 inner and outer loops means 1 million simulations being performed).

(ii) Now create a vector of WTP values, so that we can evaluate the EVPPI across a range of thresholds. Also (by running the code already written) we can create the matrices to evaluate the inner loop results, and the overall `evppi.results`, to store results. You can see that the structure of the `inner.results` matrix remains the same as used in the walkthrough above. However, for the `evppi.results` matrix, the number of rows is equal to the outer loop number, but now there is a column for each WTP value being considered. In the next steps we will see how the results can be evaluated and stored here. *(Note: in previous sessions (Module 3) we created functions that evaluate one WTP value at a time, whereas now we will provide all the WTP values into the function and calculate them all at once)*

(iii) Calculate the effective population that will benefit from the intervention over an appropriate time horizon. We have provided the information from Part 1 of this exercise for you to run here.

(iv) Next, run the code that provides the functions to estimate NMB for the inner loop results - `nmb.function`. This is essentially a function that estimates the mean NMB across all inner loop simulations (as you did in the initial part of this exercise after running the inner loops), except now it does this for a range of WTP values. The mean NMB values across all WTP thresholds are saved for each treatment.

(v) Finally, run the code to create the `gen.evppi.results()` function. You can see from the function that this estimates the NMB for current information and perfect parameter information (as you did in the walkthrough of EVPPI results), except the function has been adapted to do this across all WTP values.

(4) **Running the code for 100 inner and outer loops, for 6 parameter groups of interest**

Now that you understand the process behind calculating partial EVPIs, and have created the matrices and functions to save the results appropriately, take a moment to examine the EVPPI loops coded for you, for each of the six groups of parameters. The code is similar in structure to the walkthrough you did earlier,

albeit with more simulations, and with more WTP thresholds considered. Notice that for each EVPPI analysis, the parameter of interest will be selected in the outer loop (using 'a'), and the other parameters are included in the inner loop (sampled using 'b'). We've labelled each section of the code accordingly, to help guide you through.

Looking at the model code below, you should be able to recognise which EVPPI analysis this code is from:

```
model.THR.voi(RR.vec[b], omr.df[b,],tp.rrr.vec[b],
                                        survival.df[b,],
                                        c.revision.vec[a],
                                        state.utilities.df[b,])
```

The above code is evaluating the EVPPI with perfect information with the cost of revision. What if you wanted to evaluate the value of partial information for two parameters that are arguments in the model function? If you wanted to, you should be able to work out how to write the above code, but including two parameters in the outer loop, whilst all other remain in the inner loop.

Once you've had a look at all the EVPPI loops and have understood them, run the code to perform them all (i.e. run all the sections beginning with with 'EVPPI loops -'). We have added a progress bar for you, for each of the six groups, so that you can see how long each EVPPI simulation takes.

The speed to complete these analyses will depend on your computer (and what else your computer might be doing too), as well as the complexity of the model. In this example, each EVPPI analysis should be less than a minute, which is pretty good for 100 inner and outer loops (10,000 simulations in total). Needless to say, this where the efficiency of your code can start to make a meaningful difference to how long your simulations will take to run. *This is why we made some minor edits within the 'A4.3.3b_VoI_Model_Script.R' code earlier!*

(5) Plotting and selecting the results of interest

Now you have run and stored the results for the six different EVPPI analyses, we need to store them all in a data frame, and then to help us plot these in ggplot, reshape the data to 'long' format (instead of 'wide' format). Turn to the 'Analysis of EVPPI results' section of the template script.

Remember the results generated at this point are per patient, not for the population.

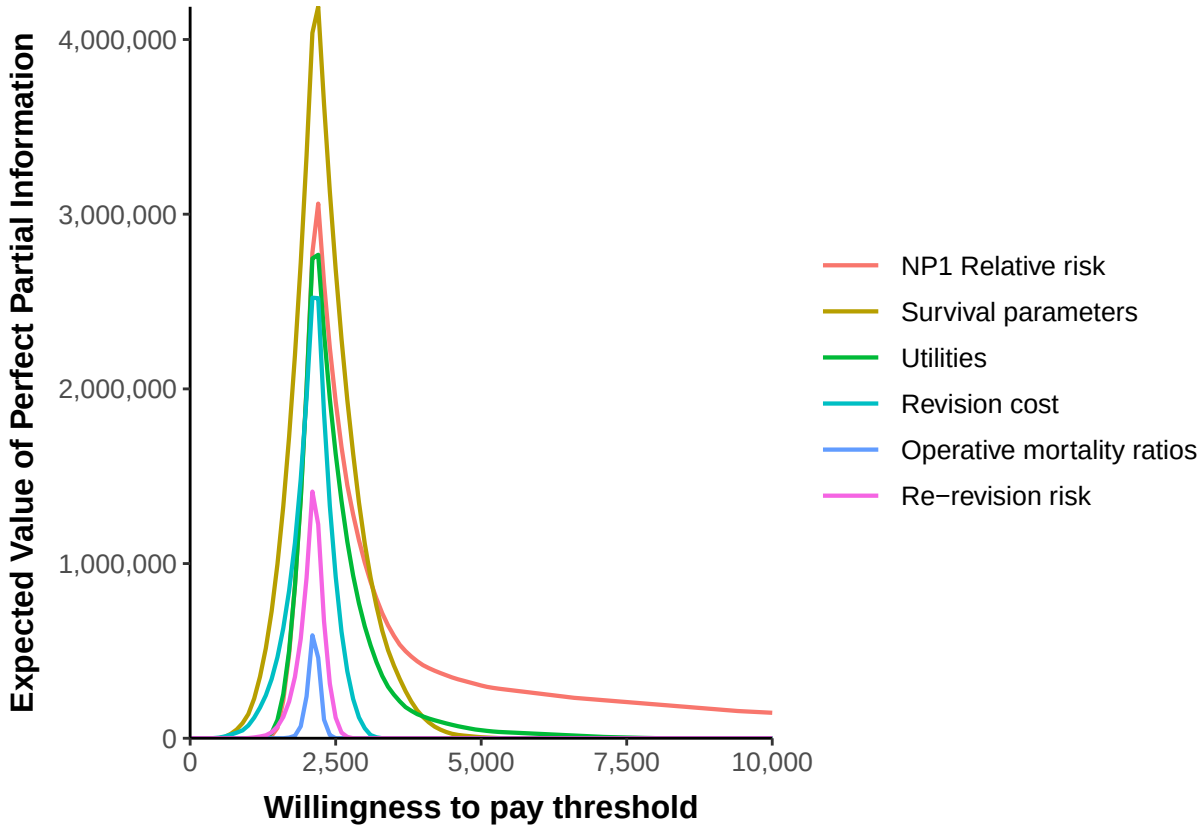
(i) Run the code to create a data.frame combining all the results (`evppi.wide.patient`).

(ii) Multiply this data.frame by the effective population, to generate EVPPI results for the whole population (`evppi.wide.pop`).

(iii) Now run the reshape function, to move the data from wide format to long format (so we can plot it in ggplot). This will be named `evppi.long.pop`.

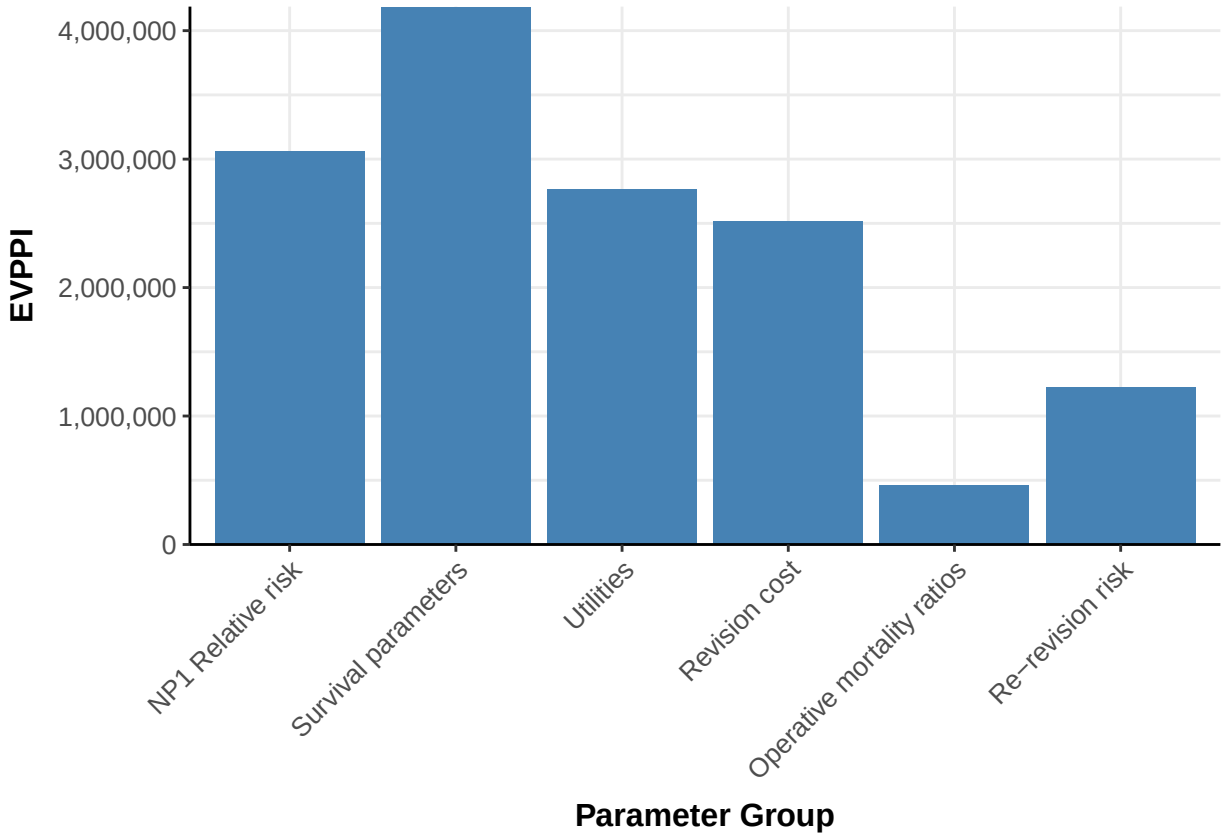
(iv) Next, it's time to visualise the results! Run the `plot.evppi()` function (remember this has been loaded in at the top of the script through the sourced `ggplot_CEA_functions.R` script) to see the EVPPI results across the WTP thresholds, for the six sets of parameters. If you want to view the plot in more detail, click the 'zoom' button above the plot in the 'Plots' pane of R. *Note that ggplot will give a warning for removed rows if not all of the data in the data.frame is plotted. This can happen when we calculate the EVPPI from willingness to pay thresholds from £0 to £50,000 (as in this example), but then only plot values up to a willingness to pay of £15,000 (to observe the data).*

The plot should look similar to this:



(v) Finally, if you want to see the specific EVPPI results at any given WTP threshold value, you can use the `subset()` function to do this. We have provided some code to do this at £2,200 (where the EVPPI is high!)

```
sub.evppi <- subset(evppi.long.pop, WTP==2200)
plot.sub.evppi(sub.evppi)
```



Congratulations! You have now calculated the EVPPI for six sets of parameters in the model, and plotted them! Although we have done this using a relatively low number of inner and outer loops (100 each), you can always go back and run these at higher numbers to ensure that the results are robust (perhaps keeping in mind how long it took to run the 100 inner and outer loops the first time!). This will also improve the quality of the plots too.