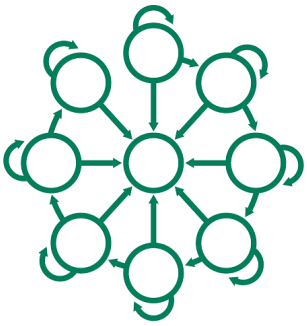


DECISION MODELLING FOR HEALTH ECONOMIC EVALUATION

Presenting Simulation Results

Advanced Course Module 3

Nichola Naylor & Jack Williams

May/June 2021 Course1

Overview

There are 2 parts to this exercise, and the purpose of these is to show how R can be used to run simulations drawing from the distributions chosen for the previously constructed probabilistic model of THR (Advanced Course Module 2 Exercise) using functions to record the results.

PART 1: We will be using the "A3.3.2a" template and "A3.3.3a" solution files. The step-by-step guide covers five main areas:

1. Creating the model function
2. Running the simulations and storing the results
3. Analysing results: the CE plane
4. Another function for calculating CEACs
5. Multiple curves: different patient sub-groups

This will require the following data files to be located in the same working-directory folder:

- hazardfunction.csv
- cov55.csv
- life-table.csv

And will require the following packages to be installed:

- ggplot2
- reshape2

The two packages mentioned above are needed as we have added some additional code at the bottom of the template and solution files that use the `reshape` and `ggplot2` packages to show you other potential ways to plot the simulation results. 'ggplot' is a graphics package that allows you to make very good graphs and plots, but requires the data to be structured in a specific way (long format), and tends to require more detailed code (which is how it allows for highly detailed plots). For more information on ggplot2, see the data visualisation cheat sheet available through RStudio: https://www.rstudio.com/resources/cheatsheets/

- Please also ensure you have the "ggplot_CEA_functions.R" script stored in the same folder as your template/solution files, with the working directory set to that same folder. Or alternatively make sure you set an appropriate file path to load the script.

PART 2: We will be using the "A3.3.2b" template and "A3.3.3b" solution files. The purpose of this exercise is to introduce a further prosthesis into the THR model in order to illustrate the approach to calculating multiple CEACs for multiple, mutually exclusive treatment options. The step-by-step guide covers two main tasks:

1. Adding in a third prosthesis option
2. Multiple curves for mutually exclusive treatment options.

All of R packages listed above for Part 1 are also needed for Part 2. Additionally the following data files are needed:

- hazardfunction_NP2.csv : Note this is different to the "hazardfunction.csv" listed for Part 1, as this now gives data on an additional treatment pathway (NP2).
- cov55_NP2.csv : Note this is different to the "cov55.csv" listed for Part 1, as this now gives data on an additional treatment pathway (NP2).
- life-tables.csv : this is the same as used in Part 1.

PART 1: Step-by-step guide

(1) Running the model function

In the template file for this exercise you will find first the same section as in Module 2 ("Deterministic Parameters"), but then a new section labelled 'THR Model Function'. The aim of this function is to wrap the main parts of the Markov model into a function, so that we can repeat and record the results of the probabilistic analysis constructed in Module 2.

As part of the previous module, we added an additional resource 'THR Model with simulation' (file name "A2.3.4_Additional_Information_THR_Model_with_simulation.R") which just created a function that re-ran the whole of the R script to run the model 1,000 times. Whilst this is a very easy way to write code to re-run the model many times, many parts of the script will be unnecessarily repeated. For example, the discount rates, and the mean and stadard error inputs for parameters will be assigned the same value in every moodel iteration, which takes time. This becomes a particular issue when the model is very complex, or you need to run a large number of simulations, running through many subgroups, or performing value of information analyses (which you learn about in the next module).

In this exercise we aimed to structure the model function to be slightly more efficient (i.e. run through the code at a faster speed) by trying to avoid repeating some of the unnecesary (deterministic) parts of the code multiple times.

Since this is the same model structure as you created in Modules 1 and 2, we haven't asked you to re-write the model code, but do read through the `model.THR()` function to check you understand what parts of the model are now within the function, and how "age" and "male" are set/used. Once you have looked through the function, we recommend that you use the arrow next to the the function code to close it (i.e. to stop showing the code), to help view the rest of the script, since the `model.THR()` code is around 200 lines long! You can run the model a few times to see the differences in the results generated across probabilistic runs. This may still take a fair few seconds to run so be patient on first running. You should have similar results to the below (remember actual numbers will differ due to random draws):

```
##     cost.SP0    qalys.SP0     cost.NP1    qalys.NP1     inc.cost    inc.qalys
## 515.20534620  13.55697189 621.91613248  13.58218548 106.71078628   0.02521359
```

Top tip for your future models: If you want to debug your function, after defining the deterministic parameters, manually set male and age (or equivalents), and work through chunks of the function code (treating it not as a function) to see where the error occurs

It should be noted that you can have functions stored in a separate R script within your working directory and run this into your simulation analysis script through using the `source()` function [check `?source` for more information]. However, for the simplicity of teaching this course this was not done here.

(2) Running the simulations

Create the following first:

(i) A `sim.runs` variable that stores the number of runs we want (you can run 1,000 in this case)

(ii) A data.frame with 6 columns (for costs and outcomes of standard treatment, for costs and the outcomes of new treatment, and incremental costs and outcomes - name the columns appropriately) that has the same number of rows as specified in `sim.runs`. Values can be set to NA for now, but set the class to numeric (this can be done through using `as.numeric(NA)`). You should also use the `rep()` function. We've laid out the structure for you to fill. The first 6 rows of `sim.runs` is shown below:

```
##   cost.SP0 qalys.SP0 cost.NP1 qalys.NP1 inc.cost inc.qalys
## 1       NA        NA       NA        NA       NA        NA
## 2       NA        NA       NA        NA       NA        NA
## 3       NA        NA       NA        NA       NA        NA
## 4       NA        NA       NA        NA       NA        NA
## 5       NA        NA       NA        NA       NA        NA
## 6       NA        NA       NA        NA       NA        NA
```

Then run the simulations in a loop, and replace the NA values with the results from 1,000 runs of the `model.THR()` function. Each vector returned by the function can be stored as a row in the data frame.

(3) **Creating the Cost-Effectiveness Plane**

Now that we have stored the results of 1000 probabilistic trials of the model, the results can be analysed:

(i) Using a simple `plot()` function plot incremental QALYs against incremental costs, with incremental qalys on the x-axis and incremental cost on the y-axis.

Next, we have used the `ggplot2` package to create more readable plots.

(ii) Load in the "`ggplot_CEA_functions.R`" script using the `source()` function, within the "A0.2_R_Starting Material_for_Advanced_Course" folder and run the `plot.ce.plane()` function we've written out for you.

Notice the slightly strange shape of the 'cloud' of points. This is due to the non-linearity relating input and output parameters inherent in the model structure. As a consequence, the point estimates from the deterministic model (evaluated at the means of the distributions) are not the same as the expectations across the output parameters (which can be estimated by averaging the cost and QALY value estimates for each arm and estimating the ICER).

(iii) Try calculating the expected average values of costs and outcomes estimated from the THR probabilistic model, and calculate the mean PSA ICER from these. How does this compare to the deterministic ICER of £2,918 per QALY gained ?

(4) **Creating the Cost-Effectiveness Acceptability Curve**

Once probabilistic results are obtained as above, the analysis of the results can proceed much like the analysis of bootstrap replications of real data in a statistical analysis (Briggs & Fenn 1998). This includes the potential for calculating confidence intervals for the ICER (using, for example, simple percentile methods) or cost-effectiveness acceptability curves (CEACs). We focus on the calculation of CEACs as the appropriate way of presenting uncertainty in CEA as a method that avoids the problems that can arise when uncertainty covers all four quadrants of the CE plane.

The calculation of CEACs will involves creating another function (`p.CE`). However, some preparation of the section is required first. In particular, the presentation mentioned that two alternative approaches to the use of the net benefit statistic could be used to generate acceptability curves. This exercise will involve calculating both in order to convince you that the methods are equivalent.

In the section 'Plotting the cost-effectiveness acceptability curve' we will first demonstrate the concept of estimating the average net monetary benefit of introducing the new prosthesis, relative to standard treatment.

- First define the willingness to pay (WTP) ceiling ratio (i.e. the maximum acceptable willingness to pay for health gain) to be £100,000.

- Use this value to calculate the average net monetary benefit for the standard and new prostheses respectively based on the average results you obtained in estimating the expected average incremental costs and effects of new treatment (`PSA.inc.cost` and `PSA.inc.qalys`). *Note that the average incremental NMB calculated will be dependent on the WTP threshold defined above.*

Based on this incremental NMB estimate, would you accept or reject the new intervention at this WTP threshold (in terms of it's net monetary benefit impact)? What is the decision rule you use for this?

Next, we need to estimate how often the new treatment would be deemed cost-effective over the 1,000 model simulations, with any given WTP threshold. There are many ways this can be coded but the aim is to output 1 numeric value of how often (i.e. the probability) that the new treatment would be deemed cost-effective over the standard treatment (at any given WTP threshold). Once we have a mechanism to calculate this, then we can perform this calculation over a range of WTP thresholds, to see how the numeric value changes across WTP thresholds.

- Based on the above concepts, write a function that calculates for each simulation run the net monetary benefit and whether that is considered cost-effective or not (based on the decision rule with net monetary benefit) and then calculates the average number of times the new treatment was cost-effective. In the function, allow the willingness-to-pay (WTP) input of the function - you can do this by specifying inputs within the brackets (e.g. `p.CE <- function(WTP, simulation.results) { }`).
- Then, generate the cost-effectiveness acceptability curve table by using a for loop and the above function to estimate the probability of cost-effectiveness for different WTP values ranging from 0 to £50,000 (by increments of £10). This can be done by first creating a data.frame with 2 columns; with the WTP values, and the probability of cost-effectiveness. *(Hint: you can fill the first column with WTP values already, and leave the second column with numeric, missing values in the first instance, and then replace them with outputs of `p.CE()` using a for loop).*
- Finally, use the `plot()` function to simply plot WTP values against probability of cost-effectiveness *(Hint: use "type="l")*. We have also provided a pre-defined function to plot the CEAC using ggplot, and we have named this function `plot.ceac()` (from"`ggplot_CEA_functions.R`" script which should already be loaded into your work environment, by running `source("ggplot_CEA_functions.R")`).

(5) **Multiple curves: different patient sub-groups**

Up to this point we have been working with a model where the patient characteristics have been set for a female aged 60 years. In this part of the exercise, we want you to repeatedly obtain results for Men and Women aged 40, 60 and 80 years and record the results and present them as multiple acceptability curves.

- Create an array that can store the PSA simulation results for each subgroup defined above. Think about the output of `model.THR()` in terms of size and shape of this array. *(Hint: think about the structure and dimensions of the 'simulations.results' data frame, and repeat for each subgroup)*
- Run the function on each of the subgroups, storing the results within the array, by completing the for loop we've started for you. *Note: match the subgroup to the order listed in `subgroups.names`.*
- You may want to have larger increments in the WTP values this time to help with spee (e.g. redefine `WTP.values` with larger increments than 10)
- Next, create a data.frame called 'CEAC.subgroups', with the number of rows equal to the WTP values and the number of columns equal to the number of subgroups + 1, to record the probability of cost-effectiveness results by running the code. *(Note that you can also use a matrix instead of a data.frame, however it is generally easier to generate plots using a data frames, particularly if using ggplot).*

- Now replace the NA values with outputs from p.CE by completing the for loop we started for you. *Note: the first column is the WTP values, but we've given some hints in the annotations of the template script for the next step of how to complete the loop.*
- You can now generate a simple plot in base R, using the `plot()` and `lines()` functions. We have provided the code for the plot for you. Note that in base R plots, the 'ylim' argument indicates the high and low values of the y-axis (in this case from 0 to 1), the 'col' argument is for colour, and the 'lty' argument indicates the linetype.

We can also use ggplot here, however, this will require that the 'CEAC.subgroups' data.frame in reshaped from wide format to long format. We have given some examples of how this can be done below *(note the data here are examples for data shaping purposes, not linking to our output results for this exercise)*:

Wide format example:

```
##   WTP Male_40 Male_60 Female_40 Female_60
## 1   0   0.992   0.260     0.820     0.018
## 2  50   0.994   0.280     0.841     0.019
## 3 100   0.996   0.301     0.866     0.020
## 4 150   0.997   0.314     0.895     0.024
```

Long format example:

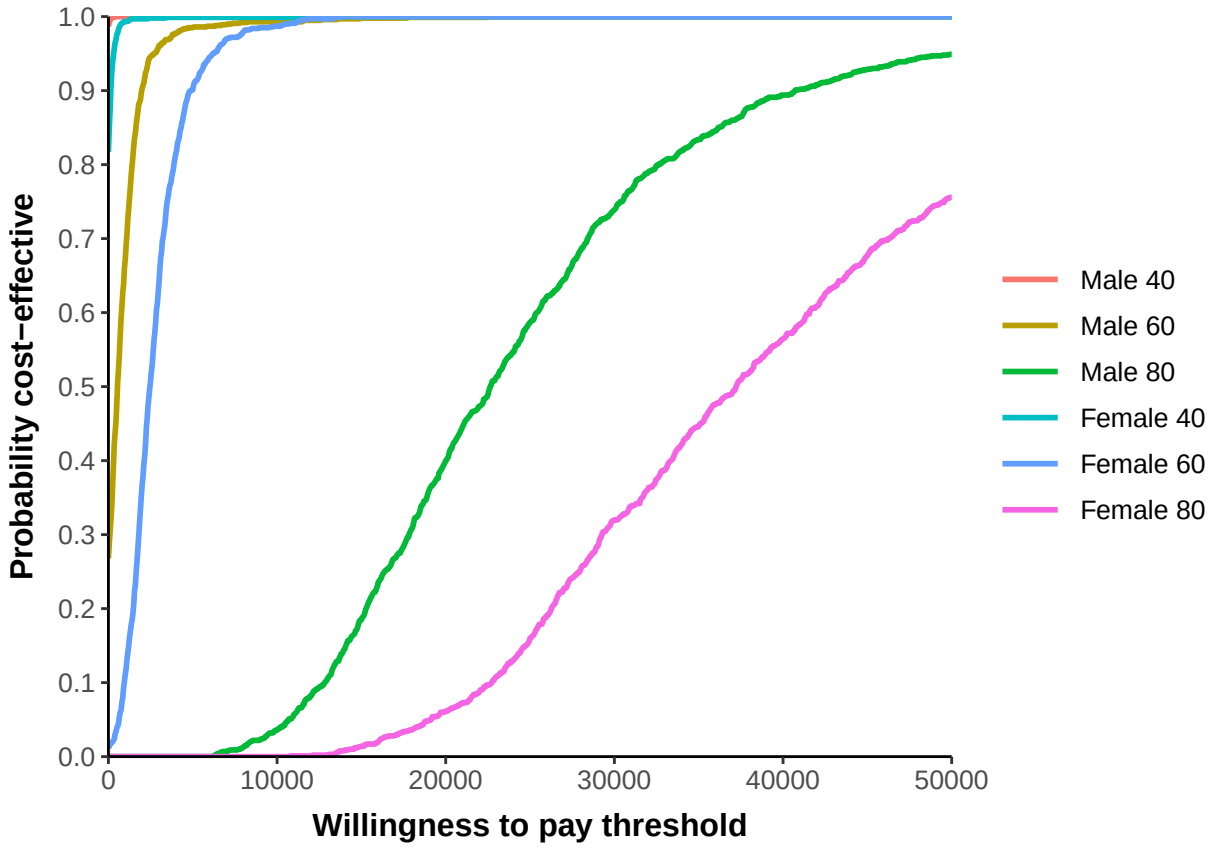
```
##   WTP subgroup   pCE
## 1   0  Male_40 0.992
## 2  50  Male_40 0.994
## 3 100  Male_40 0.996
## 4 150  Male_40 0.997
## 5   0  Male_60 0.260
## 6 100  Male_60 0.280
## 7 150  Male_60 0.301
```

We can do this using the `melt()` function within the 'reshape2' package (which should have been loaded in at the start of this exercise using `library(reshape2)`, if not go back to the beginning of this script and load this in). Type in `?melt.data.frame` into the console to see how to use this function. Have a go at filling out `melt()`. For those familiar with the 'tidyr' and 'dplyr' packages, the `gather()` function can also be used to reshape from wide to long format. For those interested, the following code would also convert the code from wide to long format *(note that this alternative is not in the template or solutions files)*:

```
CEAC.subgroups %>% gather(group, pCE, 2:7)
```

Finally, you can plot the resulting CEACs for the different patient characteristics using the pre-defined ggplot function `plot.ceac.all()`.

You should find that the curves can be quite different for different patient characteristics (your final graph for Part 1 should look similar to the below). For example, the new prosthesis seems to be more cost-effective generally for men (due to the fact that men have higher failure rates on average and so the absolute benefits of reducing failure risk are greater) and that the new prosthesis is not so cost-effective for elderly patients (where death is an important competing risk for prosthesis failure).


Probability cost−effective
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
10000
20000
30000
40000
50000
Willingness to pay threshold
Male 40
Male 60
Male 80
Female 40
Female 60
Female 80

PART 2: Step-by-step guide

The purpose of this exercise is to introduce a further prosthesis into the THR model in order to illustrate the approach to calculating multiple CEACs for multiple treatment options.

The template for this exercise is 'A3.3.2b_Presenting_Simulation_Results_Part2_Template.R.R'.

This step-by-step guide covers three main tasks:

1. Adding in a third prosthesis option to the function
2. Running a new round of simulations and storing these results
3. Creating multiple curves for mutually exclusive treatment options

(1) Adding in a third prosthesis option

Although in the vast majority of trials just two treatment alternatives are compared, in modelling it is much more common to look at multiple treatment options. If you open A3.3.2b_Presenting_Simulation_Results_Part2_Template.R you will find that the read in for "cov55.csv" has been replaced by "cov55_NP2.csv", and "hazardfunction.csv" by "hazardfunction_NP2.csv". Take a moment to examine the new `hazards` and `cov.55` data, which contains additional information concerning this prosthesis in terms of its effectiveness and correlation with other parameters.

The aim in this section is to update the model so that the third prosthesis option is fully integrated. This involves updating the parameter, analysis and simulation sections before re-running the probabilistic sensitivity analysis.

We start by defining the deterministic parameters that don't update for each run, as we have done previously. The main difference here is we need to add the the new treatment pathway (NP2) to the Cholesky decomposition matrix. You can see that the new data on the new prosthesis (NP2) has been taken from the new hazards data frame (see `mn.NP2`).

Once you have assigned values to the parameters outside of the model function by running the code we have written, you will need to set up a new function `model.THR.3b()` to run the model. We've outlined the bulk of `model.THR.3b()`, but there are few things you need to fill in before it will work:

- Relative risk of revision for new prosthesis 2 compared to standard: `r.NP2` and `RR.NP2` (within the hazard function section of the function) and `revision.risk.np2` (within the Markov model section, referring to the time dependent risk of revision for NP2). *Hint: use the Ctrl+F aka "Find" function within the template script if you can't find these variables, however it is useful to read through the whole script line by line, even if we've filled this in for you.*
- Create transition and trace matrices for NP2 (`tm.NP2` and `trace.NP2`), using the code provided for NP1 as a guide.
- Estimate the discounted cost (`disc.cost.NP2`) and QALY (`disc.QALYs.NP2`) outcomes from NP2, using the code provided for NP1 as a guide.
- Finally, you need to make sure the function returns the results for all three comparators now. Update the `output` matrix to ensure that this includes estimated discounted cost and qalys for NP2.

Hint: It might be worth testing each stage of the model by defining `age` (e.g. `age <- 60`) and `male` (e.g. `male <- 0`) and work through each section of the function before running the whole thing. If you do this, the first 6 rows of the updated revision.risk.NP2 column in tdtps should look like the following:

```
## [1] 3.790277e-05 6.698251e-05 8.535086e-05 1.000037e-04 1.125353e-04
## [6] 1.236476e-04
```

(2) **Running and Storing Simulation Results**

Now that the adaptations relating to the third option are complete the updated results can be produced. We can now store the simulation results, similar to the end of Part 1 of this exercise, work your way through the 'Running the simulations' section of the template file. Be patient, it might take a few seconds for your simulations to run. Complete the for loop we've started to replace blank rows for `simulation results`. Your simulation.results should have the same structure as the below:

```
##   cost.SP0 qalys.SP0 cost.NP1 qalys.NP1 cost.NP2 qalys.NP2
## 1 497.6273  14.28784 617.5027  14.30963 806.6046  14.31630
## 2 507.1884  14.14835 608.3530  14.18589 824.3230  14.18276
## 3 486.0976  14.45873 602.5787  14.47566 799.6667  14.47862
## 4 609.7804  14.81916 617.4793  14.91832 797.3600  14.93468
## 5 548.0864  14.31910 608.2155  14.34966 834.9179  14.34531
## 6 485.6092  15.21352 597.8175  15.26201 818.8096  15.25401
```

Note: in the solution file we have added code that shows the progress of your simulation runs, using `txtProgressBar()`. You don't need to include it in your template file for this exercise, but take a look at how we've implemented it, as we will start to use it more in the next exercise.

We can calculate the mean results across the simulations and use these values to update the point estimates for the probabilistic analysis in the 'Analysis' section. Use the `apply()` function to calculate these averages. *Hint: you want to use the `mean` function over columns, enter `?apply` into the console to see more information on this function*

(3) **Multiple curves for mutually exclusive treatment options**

Now that you've run the simulations and stored the results, we can begin to plot the simulation results graphically. First let's plot the cost-effectiveness plane. Go to the 'Plotting the cost-effectiveness plane' section, where we've added the code to firstly reshape the data, and second to generate the plot for the cost-effectiveness plane. Run the code and take a look at the results.

Note that one of the problems of looking at the three clouds of points on the CE plane is that we lose the perception of how the points are correlated between the three options.

The results from the probabilistic analysis of mutually exclusive options can also be presented as multiple cost-effectiveness acceptability curves. However, in contrast to the independent sub-group analysis of the first part of this exercise, these multiple curves must sum to one, since they represent the probability of each comparator being cost-effective at any given WTP threshold. We must therefore calculate the NMB associated with each intervention at each WTP threshold, but this time for three model comparators.

(i) First, calculate the average net monetary benefit for each option by first defining a vector of WTP values from £0 to £50,000 per QALY gained, by increments of your chosing (`WTP.values`). Then create a data.frame (`CEAC`) which has the WTP values and empty columns for standard, NP1 and NP2 columns.

(ii) Next, create a function (`pCE.3b`), to estimate net monetary benefit for each intervention, and specify WTP as an argument in the function. We've started the function, but left a few definitions blank for you to complete. The aim of the function is to estimates the probability of the new intervention being cost-effective. You will need to use the `apply()` function to first select the maximum 'nmb' value by row (call this `max.nmb`) and then again to calculate the mean number of times that each treatment pathway was found to be the most cost-effective (i.e. the mean times each column within the `CE` data.frame was found to be the strategy with the maximum NMB) - defining `probCE`.

(iii) Complete the for loop to fill in the CEAC table with the outputs of the `pCE.3b` function.

After running the for loop, you should have a CEAC with a 'head' and 'tail' similar to the below (differing values are to be expected, the main thing is to have a similar structure filled with numerics and a similar pattern - i.e. a shift from Standard being the most cost-effective to the NP1/NP2 pathways being more cost-effective as we move from lower WTP values to higher WTP values).

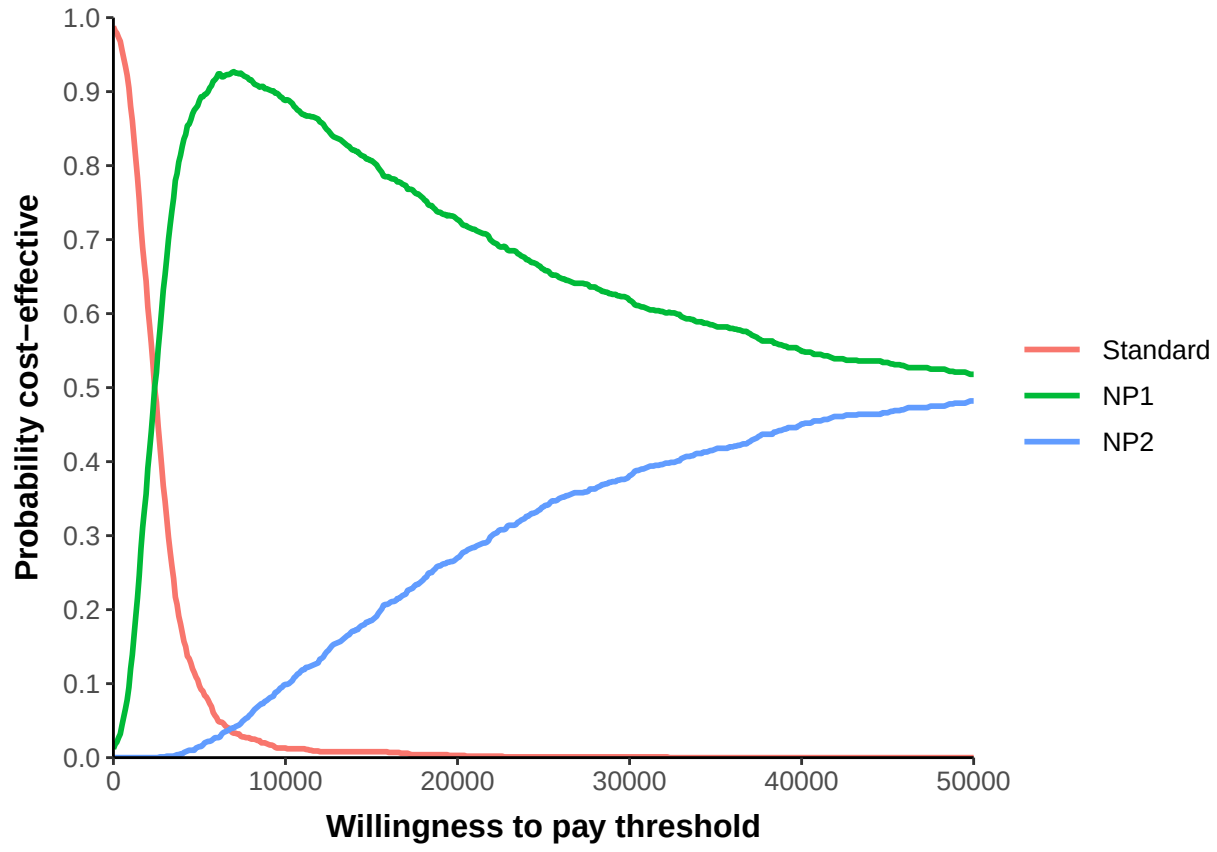
```
head(CEAC)
```

```
##   WTP Standard   NP1 NP2
## 1   0    0.988 0.012   0
## 2 100    0.982 0.018   0
## 3 200    0.979 0.021   0
## 4 300    0.973 0.027   0
## 5 400    0.968 0.032   0
## 6 500    0.956 0.044   0
```

```
tail(CEAC)
```

```
##       WTP Standard   NP1   NP2
## 496 49500        0 0.521 0.479
## 497 49600        0 0.520 0.480
## 498 49700        0 0.519 0.481
## 499 49800        0 0.518 0.482
## 500 49900        0 0.518 0.482
## 501 50000        0 0.518 0.482
```

Now run the final lines of code that uses our pre-defined `plot.ceac.all()` from the "ggplot_CEA_functions.R" script, first reshaping the data to match the format needed. The final CEAC should look like this:



As was noted in the presentation, the interpretation of multiple acceptability curves can be tricky – especially when the model is non-linear. Fenwick et al (2001) have argued for the use of a cost-effectiveness acceptability frontier (CEAF) in these situations and an example of this has been provided for you at the bottom of the R script.

The `CEAF.function()` function has been written for you, which is similar to the function used earlier in the script to estimate the probability of cost-effectiveness, except that rather than selecting the treatment with the highest probability of cost-effectiveness, the function selects the probability of cost-effectiveness for the treatment with the highest net monetary benefit (based on average net monetary benefit across all simulations run). We can also use the same plot function '`plot.ceac.all()`' to create a plot to show the results, this time for the CEAF.

Congratulations! You have now created a time-dependent, probabilistic Markov model with an additional treatment arm comparator, additionally you've presented results from such analyses graphically. In the next module we'll be building on `model.THR` to explore Value of Information Analyses.