Practicals

Health Economics in R

University of Oxford 24th November 2022
Nathan Green (UCL)

Contents

Preface

All the material needed to do these practical exercises is provided for you at the course website or in a zip file. This has one folder for each practical session / chapter of this document, and contains program code and data for BUGS and R.

Some of the practical sessions have a file containing solutions to the exercises, in the corresponding folder. For some of the exercises, solutions are not necessary, since they simply consist of stepping through a script that has been provided.

0.1 Acknowledgements

This material is part taken from the annual Summer School in Bayesian Health Economics. Lectures and practical were created by Gianluca Baio, Howard Thom, Anna Heath, Nicky Best, Chris Jackson and others. Thanks to all those who have contributed to this work.

1

A (too short) introduction to R

This is a very brief introduction to R (which can be downloaded from the website www.r-project.org) and its capabilities. It will be extremely focussed on the characteristics that are instrumental to do health economic evaluations using a combination of R, BUGS and some useful packages (such as BCEA). Thus it is by no means exhaustive!

When you open the R terminal, you are presented with the possibility of typing commands. You may want to open a text editor (e.g. the simple one built into the R Windows interface) in which you can type directly these commands, and save them to a script for future use. Another possibility is to use R from within a more sophisticated "integrated development environment", such as RStudio (www.rstudio.com), which has many more features than the basic Windows interface.

In any case, R is a very powerful tool; more importantly, it is free and you can find a wealth of documentation on the internet. R has a set of built-in commands, which you can use for basic operations. However, there are also many add-on packages containing sets of functions designed to perform specific statistical tasks. These packages can be installed to *your* R by typing the command

```
> install.packages("package_name")
```

(assuming you have an internet connection and noticing that the symbol > indicates the beginning of a line of code in R). This command only needs to be executed one time. Once a package is installed in your local library (a collection of packages) you can make it available to the current R session by typing the command

```
> library(package_name)
```

For these practicals, you will need to install and load the packages:

- BCEA, which can be used to post-process the results of a (Bayesian) model to perform a health economic evaluation.
- R2OpenBUGS, which can be used to interface R and BUGS.

You do this by typing in your R terminal the commands

```
> install.packages("BCEA")
> install.packages("R2OpenBUGS")
> library(BCEA)
> library(R2OpenBUGS)
```

Both BCEA and R2OpenBUGS will automatically load other packages that they *depend on* — this means that in order to work, they need to access functions that are part of other packages.

If you wish so, you can use JAGS in the practicals. To this end (and assuming you have actually installed the current version of JAGS to your computer), you will need to also install the package R2jags, which you can do by typing in your R terminal

```
> install.packages("R2jags")
```

Notice that if you decide to use JAGS instead of BUGS, you will need to slightly modify some of the commands — we describe this in more details later in this manual.

Once a package is loaded to your R workspace, you can type the command help(package_name), which will open a window displaying a description of the package. For example help(BCEA) provides some basic information (including details of the current version). You can use the command help also on specific functions within the package, e.g. typing help(bcea) describes in detail how to use the bcea function (notice that in this case the package name is typeset in uppercase, while the function is lowercase!).

The very basic commands that are required to do a typical R session working with BUGS and BCEA will be given and described later or in the scripts that we refer to in the practicals.

2

Decision trees

This is a gentle introduction to implementing decision trees for health economics in R. This is meant as an early practical to familarise with using R and some of the basic concepts.

2.1 Running and analysing deterministic results

First run the simple depression model stored in file `practical.R` in the decision tree folder. Ensure you understand each line of this file. Recall that the model equations are

```
costs <- c.treat+p.rec*(1-p.rel)*c.rec+p.rec*p.rel*c.rel+(1-p.rec)*c.norec
effects <- p.rec*(1-p.rel)*q.rec+p.rec*p.rel*q.rel+(1-p.rec)*q.norec
```

Look at the values of the matrix parameters (e.g. `p.rel`, `c.treat`) going into this equation.

- What is the Net Benefit?
- Calculate the incremental costs and effects.
- What is the ICER?

2.2 Running and analysing probabilistic results

First run the probabilistic depression model stored in file `practical_probs.R` in the decision tree folder. Ensure you understand each line of this file.

Look at the values of the matrix parameters (e.g. `p.rel`, `c.treat`) going into this equation using the `colMeans()` function. This takes a mean of the columns matrices; for example, `colMeans(p.rel)` will give the mean probability of relapse on each of the three treatment options. Look at the mean of vectors (e.g. `c.rec, c.rel`) using the `mean()` function. A quick way to check if a data structure is a matrix or vector is to use `dim()`, the dimensions of a matrix, as this will be `NULL` for a vector.

- Can you tell which treatment has the highest average probability of recovery or lowest probability of relapse?
- Of cost of no recovery, relapse, and recovery, which has the highest mean?
- Of QALY associated with no recovery, relapse, and recovery, which has the highest mean?
  Now that you understand the inputs to the costs and effects, use the `colMeans()` function to find the treatment with lowest costs and highest effects. The net benefit at £20,000 is defined as
  `net.benefit <- 20000*effects - costs`
  Note that this multiplies 20000 by all the elements of the effects matrix and subtracts the corresponding elements of the costs matrix.
- Which intervention has the highest mean net benefit and should be recommended for treatment of depression?

2.3 Using BCEA to compare depression treatment strategies.

We will now use the `BCEA` package to analyse the effects and costs matrices in`depression_psa.RData`. This will contrast the difficulty of simply comparing mean costs, effects, and net benefits (exercise 1) with a fully Bayesian and probabilistic interpretation of the results. First load the `BCEA` package using

```
library(BCEA)
```

If BCEA has not yet been installed you'll need to call `install.packages("BCEA")` first.

- First use the `bcea()` function to generate a bcea object summarising the costs and effects Use the options `ref=1` to specify that "no treatment" is the reference and interventions=t.names to specify the appropriate names of the interventions.
- Apply `summary()` to the object created by `bcea()` in part (a) above. Use the option `wtp=20000` so that the willingness-to-pay for the net benefit is £20,000 (default in BCEA is £25,000) This gives comparisons of CBT and antidepressants to no treatment. The "EIB" is expected incremental benefit at the `wtp=20000`, the "CEAC" (cost-effectiveness acceptability curve) is the probability that the reference of "no treatment" has highest net benefit (most cost-effective) at the specified willingness-to-pay, and the ICER is the incremental cost-effectiveness ratio. The last of these can be compared with the standard willingness-to-pay threshold of £20,000. On these measures, how do CBT and antidepressants compare to no treatment?
- Now apply `bcea()` and `summary()` to compare the CBT and antidepressants option. To do this, first use `bcea()` but with `ref=2`, giving comparisons relative to CBT. Now use `summary()` to get the EIB and CEAC of antidepressants relative to CBT. Which option would be recommended at a willingness-to-pay threshold of £20,000? Note that the ICER is difficult to interpret due to negative incremental costs, so only focus on EIB and CEAC.
- As there are three decision options, it may be better to compare them simultaneously, rather than doing the pairwise comparisons of (b) and (c). Pass the `bcea` object created in part (a) to the `mulit.ce()` function and store the result. Now use `ceac.plot()` on the output of `multi.ce()`. This gives the probability that each of the three options has the highest net benefit for a range of willingness-to-pay thresholds. Which treatment has the highest probability of being most cost-effective at the £20,000 threshold?

3

A decision tree deterministic sensitivity analysis

This section demonstrates a part of a full deterministic sensitivity analysis over all input parameters of the decision tree model. We will focus on the 2nd treatment for simplicity but the same step can be carried-out for the other treatments and outputs used in post processing steps such as tornado plots.

3.1 Set-up

First, open the script stored in file `practical_sa.R` in the decision tree folder. Ensure you understand each line of this file. This practical is similar to the first practical above except now rather than explictly defining the parameter values inside of the R script such as the costs or probabilities for each treatment, we now read then in from an external file. This allows us more flexibility and separates the input data part of the analysis from the actual computation part. This is especially beneficial when these get large and harder to manage as a single object.

So, read in the set of input values contained in `det_sa_inputs.csv`. You can then view it insdie of R. Alternatively, you could open this file externally, in something like Microsoft Excel and view and edit there, which is what I did originally. This is good if you're working with collaborators or people who are too comfortable with using R.

`det_sa_inputs.csv` has one row for each model run, i.e. set of input values, and each column corresponds to one of the parameters. You can arrange these however you like but in this case I have put the scenario with all mean values first and then changed each parameter one at a time from left to right - maximum value first and then minimum value. Thus, this create a matrix where the off-diagonal values are fixed at the means.

3.2 Looping through scenarios

The main part of the script is the for loop over scenarios. The cost and effect equations should look familiar from the previous practical but now the values use in these equations are taken from the `inputs` matrix.

We use a small trick of using the `with()` function to wrap around the equations. This allows us to reference the column of the matrix without having to write `scenario$p.rec` etc every time, which make the code easier to read and less errror prone.

3.3 Post-processing

For post processing, we calculate the Net Benefit for all of the calculated scenarios and append this to the result object.

We also plot a simple tornado plot using the `geom_pointrange()` fro the `ggplot2` package. This is a quick way of viewing the output but for publications we would have to do a bit more cosmetic work.

- Change some of the input values in the .csv file and see how this changes the output.
- Repeat the analysis for the 3rd treatment.

4

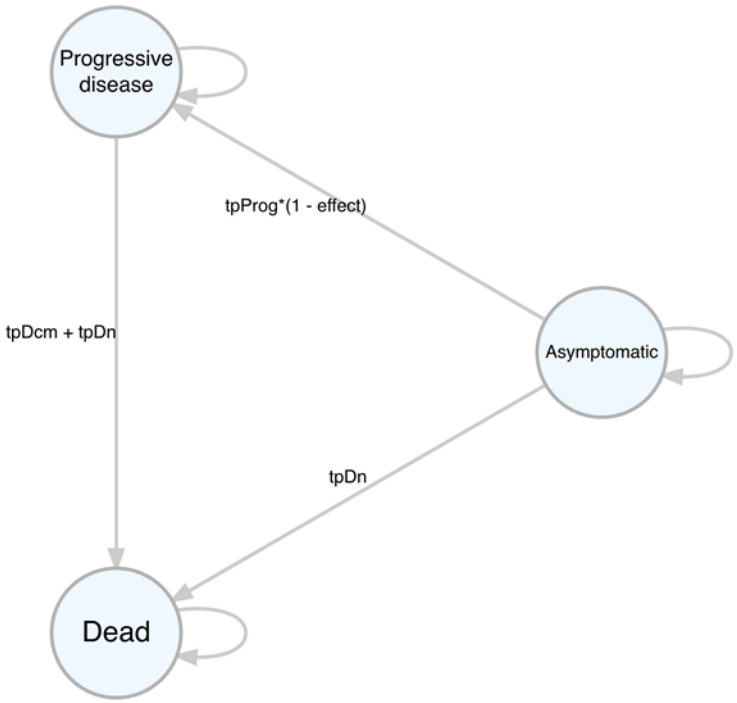
Introduction to Markov models

This is rather long practical to implement a complete Markov model for cost-effectiveness analysis. Extensions to the basic model will be made as we go. The R code is available in a separate file `practical.R` and the full code in `practical_solutions.R` so you don't have to cut and paste everything.

4.1 Three-state model

This model is take from an example used in Briggs A, Sculpher M. Introducing Markov models for economic evaluation. PharmacoEconomics 1998; 13(4): 397-409. and Briggs AH. Handling uncertainty in cost-effectiveness models. PharmacoEconomics (2000). The model is freely available as an Excel spreadsheet which we provide. In the following we will ask you to use the spreadsheet to complete the R code provided.

Consider a three-state model is used to describe how patients transition between health states with a chronic disease. Patients begin in an asymptomatic health state, which means that the patient has developed the chronic disease but has so symptoms. Patients can stay in the asymptomatic health state or move to a progressive health state which means that the patient is experiencing symptoms of the disease. Patients can move from the asymptomatic health state to the dead health state at the same rate as the general population without the disease. Patients can stay in the progressive health state or move from the progressive health state to dead, but at an increased risk of death. The dead state is an absorbing state as a patient cannot change from being dead.



- Take a look at the original Excel spreadsheet and understand how it implements the model and what the particular characteristics are. What are the initial values? How does it incorporate age-dependent transitions?

Open the R script "practical.R" in the markov model folder. Look over the code and try and understand what the commands are doing. Next we will step through the script.

We will assume that a cohort of 1000 patients receives a drug and compares to a cohort of 1000 patients that does not receive a drug to assess how patients move throughout the health states over time with a chronic disease. To set up the model in R some definitions are needed first. We outline the R code used to define the number of treatments (prefixed with `t_`) and their names (prefixed with `n_`), the number of states (prefixed with `s_`) and their names. The number of cycles starting at 1 (not 0 as in spreadsheet models) is specified, as well as the initial age of patients in the cohort beginning at 55.

```
t_names <- c("without_drug", "with_drug")
n_treatments <- length(t_names)

s_names  <- c("Asymptomatic_disease", "Progressive_disease", "Dead")
n_states <- length(s_names)

n_pop <- 1000

n_cycles <- 46
Initial_age <- 55
```

The unit costs and unit utilities associated with each of the health states as well as the cost of the drug need to be defined. The utility of being in the dead health state is 0 so does not need to be defined here. Costs begin with a `c` and utilities with a `u`. The discount rate for costs and outcomes are also included at a rate of 6%.

- Initialise the variable using the values in the spreadsheet.

```
cAsymp <- ?
cDeath <- ?
cDrug <- ?
cProg <- ?
uAsymp <- ?
uProg <- ?
oDr <- ?
cDr <- ?
tpDcm <- ?

# transition matrix variables
tpProg <- 0.01
tpDcm <- 0.15
tpDn <- 0.0138
effect <- 0.5
```

This process of defining treatment names, states and cycles is similar to that in MS Excel. One addition in R is that space for when calculations are performed is needed. It is like creating the cells in MS Excel. By creating matrices, empty space for costs and utilities in each health state for patients with or without the drug is specified. The structure of the matrices for the cost of transition to a health state, and cost and QALYs accrued being in health state are the same. We describe the cost of transitioning to a state, in this case only for the dead state. The first argument defines a vector of values for each health state and for each cohort, similar to a column of values in a parameters sheet in Excel. The argument `byrow = TRUE` makes sure that all the states for the first cohort are defined first and then all the states for the second cohort. The `trans_c_matrix` creates empty space and assigned the cost of £1000 for transitioning to the Dead state. The first line is for the cohort who are without a drug, and the second line for the cohort that are with a drug.

```
# cost of staying in state
state_c_matrix <-
  matrix(c(cAsymp, cProg, 0,
           cAsymp + cDrug, cProg, 0),
```

```
          byrow = TRUE,
          nrow = n_treatments,
          dimnames = list(t_names,
                          s_names))

# qaly when staying in state
state_q_matrix <-
  matrix(c(uAsymp, uProg, 0,
           uAsymp, uProg, 0),
         byrow = TRUE,
         nrow = n_treatments,
         dimnames = list(t_names,
                         s_names))

# cost of moving to a state
# same for both treatments
trans_c_matrix <-
  matrix(c(0, 0, 0,
           0, 0, cDeath,
           0, 0, 0),
         byrow = TRUE,
         nrow = n_states,
         dimnames = list(from = s_names,
                         to = s_names))
```

Space is also needed to define the transition probabilities between health states. A matrix is created as above but with another dimension for the movements between health states. The resulting matrix shows transitions between each health state dependent on the cohort being with or without a drug. We then insert specific values.

- The following arrays are 3-dimensional. Why do you think the dimensions have the order they do? Print the array to screen for a clue.

```
# Transition probabilities
p_matrix <- array(data = 0,
                  dim = c(n_states, n_states, n_treatments),
                  dimnames = list(from = s_names,
                                  to = s_names,
                                  t_names))

## assume doesn't depend on cycle
p_matrix["Asymptomatic_disease", "Progressive_disease", "without_drug"] <- tpProg
p_matrix["Asymptomatic_disease", "Dead", "without_drug"] <- tpDn
p_matrix["Asymptomatic_disease", "Asymptomatic_disease", "without_drug"] <- 1 - tpProg - tpDn
p_matrix["Progressive_disease", "Dead", "without_drug"] <- tpDcm + tpDn
p_matrix["Progressive_disease", "Progressive_disease", "without_drug"] <- 1 - tpDcm - tpDn
p_matrix["Dead", "Dead", "without_drug"] <- 1

# Matrix containing transition probabilities for with_drug
p_matrix["Asymptomatic_disease", "Progressive_disease", "with_drug"] <- tpProg*(1 - effect)
p_matrix["Asymptomatic_disease", "Dead", "with_drug"] <- tpDn
p_matrix["Asymptomatic_disease", "Asymptomatic_disease", "with_drug"] <- 1 - tpProg*(1 - effect) - tpDn
p_matrix["Progressive_disease", "Dead", "with_drug"] <- tpDcm + tpDn
p_matrix["Progressive_disease", "Progressive_disease", "with_drug"] <- 1 - tpDcm - tpDn
p_matrix["Dead", "Dead", "with_drug"] <- 1


# Store population output for each cycle

# state populations
pop <- array(data = NA,
             dim = c(n_states, n_cycles, n_treatments),
```

```
            dimnames = list(state = s_names,
                            cycle = NULL,
                            treatment = t_names))
```

The transition probability matrix is defined upfront in this scenario but later we will see how to very it during run time.

- Set the starting state populations. What are the indices? What are the assigned values?

```
# _arrived_ state populations
trans <- array(data = NA,
               dim = c(n_states, n_cycles, n_treatments),
               dimnames = list(state = s_names,
                               cycle = NULL,
                               treatment = t_names))

trans[, cycle = 1, ] <- 0
```

4.2 Output variables

A population matrix (`pop`) and transition matrix (`trans`) are created with an additional dimension so there is blank space for each health state, with or without a drug, for each of the 46 cycles. Below shows this process for a generic array (`cycle_empty_array`).

```
# Sum costs and QALYs for each cycle at a time for each drug

cycle_empty_array <-
  array(NA,
        dim = c(n_treatments, n_cycles),
        dimnames = list(treatment = t_names,
                        cycle = NULL))

cycle_state_costs <- cycle_trans_costs <- cycle_empty_array
cycle_costs <- cycle_QALYs <- cycle_empty_array
LE <- LYs <- cycle_empty_array    # life-expectancy; life-years
cycle_QALE <- cycle_empty_array   # qaly-adjusted life-years

total_costs <- setNames(c(NA, NA), t_names)
total_QALYs <- setNames(c(NA, NA), t_names)
```

4.3 Running the model

To run the model and combine all the information and code from the previous sections, an algorithm will be created. A loop is first created over treatments and then a second loop repeats from cycle number 2 to cycle 46 (`n_cycles`), as cycle 1 was already defined above, equivalent to cycle 0 rows in Excel models. Recall the matrix multiplication operator `%*%`, used here to calculate the state population at the next time step (`pop`) and the number of individuals who transition between states (`trans`).

```
for (i in 1:n_treatments) {

  age <- Initial_age

  for (j in 2:n_cycles) {

    pop[, cycle = j, treatment = i] <-
      pop[, cycle = j - 1, treatment = i] %*% p_matrix[, , treatment = i]

    trans[, cycle = j, treatment = i] <-
     pop[, cycle = j - 1, treatment = i] %*% (trans_c_matrix * p_matrix[, , treatment = i])
```

```
    age <- age + 1
  }

  cycle_state_costs[i, ] <-
   (state_c_matrix[treatment = i, ] %*% pop[, , treatment = i]) * 1/(1 + cDr)^(1:n_cycles - 1)

  # discounting at _previous_ cycle
  cycle_trans_costs[i, ] <-
    (c(1,1,1) %*% trans[, , treatment = i]) * 1/(1 + cDr)^(1:n_cycles - 2)

  cycle_costs[i, ] <- cycle_state_costs[i, ] + cycle_trans_costs[i, ]

  LE[i, ] <- c(1,1,0) %*% pop[, , treatment = i]

  LYs[i, ] <- LE[i, ] * 1/(1 + oDr)^(1:n_cycles - 1)

  cycle_QALE[i, ] <-
    state_q_matrix[treatment = i, ] %*% pop[, , treatment = i]

  cycle_QALYs[i, ] <- cycle_QALE[i, ] * 1/(1 + oDr)^(1:n_cycles - 1)

  total_costs[i] <- sum(cycle_costs[treatment = i, -1])
  total_QALYs[i] <- sum(cycle_QALYs[treatment = i, -1])
}
```

The discount rate is also incorporated into the model here easily as each cycle's costs and QALYs will depend on the cycle number. These repeated steps are performed for each of the two treatments (1:n treatments) and the total costs and QALYs over the lifetime of the model can then be calculated for each treatment.

4.4 Plot results

Displaying the results after running the model is easy in R. . These results will therefore assume that the strategy where the cohort are without a drug is the standard of care or base case analysis. Swapping with and without the drug will change this around.

- The incremental cost-effectiveness ratio (ICER) requires the incremental costs and incremental QALYs. Calculate the ICER value.

  Create a simple cost-effectiveness plane as follows:

```
plot(x = q_incr/n_pop, y = c_incr/n_pop,
     xlim = c(0, 1500/n_pop),
     ylim = c(0, 12e6/n_pop),
     pch = 16, cex = 1.5,
     xlab = "QALY difference",
     ylab = "Cost difference (£)",
     frame.plot = FALSE)
```

4.5 Cycle-dependent probability matrix

Transition probabilities in a Markov model can either be time-independent, such as tpDm, or time-dependent. Next, define the transition probabilities in the full model depend on age and cycle. To begin with let us only depend on cycle and fix the age varying variable `tpDn` at a middle value. Define the transition probability from Asymptomatic to Progressive disease, previously just `tpProg`, to depend on cycle such that the new transition probability is `tpProg` $\times$ `cycle`.

To account for the time dependency, the hard-coded array in MS Excel has been replaced with a function in R named `p_matrix_cycle` which is called at each cycle iteration. By simply replacing a fixed array with a function, this will decouple the calculation of the transition matrix and the higher-level cost-effectiveness calculations. This makes changes and testing to either part easier and more reliable.

Note that, strictly speaking, in R this is an array but we have named the data structure `p_matrix` to emphasise that it provides what is known in Markov modelling as the transition probability matrix. The time dependent probabilities are those that depend on the age of the cohort. A lookup function is used to describe the transition probability from Asymp to Dead using 6 age group categories.

```
p_matrix_cycle <- function(p_matrix, cycle,
                           tpProg = 0.01,
                           tpDcm = 0.15,
                           tpDn = 0.0138
                           effect = 0.5) {

  # Matrix containing transition probabilities for without_drug
  p_matrix["Asymptomatic_disease", "Progressive_disease", "without_drug"] <- tpProg*cycle
  p_matrix["Asymptomatic_disease", "Dead", "without_drug"] <- tpDn
 p_matrix["Asymptomatic_disease", "Asymptomatic_disease", "without_drug"] <- 1 - tpProg*cycle - tpDn
  p_matrix["Progressive_disease", "Dead", "without_drug"] <- tpDcm + tpDn
 p_matrix["Progressive_disease", "Progressive_disease", "without_drug"] <- 1 - tpDcm - tpDn
  p_matrix["Dead", "Dead", "without_drug"] <- 1

  # Matrix containing transition probabilities for with_drug
 p_matrix["Asymptomatic_disease", "Progressive_disease", "with_drug"] <- tpProg*(1 - effect)*cycle
  p_matrix["Asymptomatic_disease", "Dead", "with_drug"] <- tpDn
  p_matrix["Asymptomatic_disease", "Asymptomatic_disease", "with_drug"] <-
    1 - tpProg*(1 - effect)*cycle - tpDn
  p_matrix["Progressive_disease", "Dead", "with_drug"] <- tpDcm + tpDn
  p_matrix["Progressive_disease", "Progressive_disease", "with_drug"] <- 1 - tpDcm - tpDn
  p_matrix["Dead", "Dead", "with_drug"] <- 1

  return(p_matrix)
}
```

4.6 Running the model

The model is run in the same way as the first example.

- Include a new line `p_matrix <- p_matrix_cycle(p_matrix, j - 1)` inside the loops used to run the model. Where should it go?

The cost, QALY, LE, LY and QALE at each cycle are calculated as the `p_matrix` is updated at each iteration of the loop.

4.7 Plot results

- Calculate the ICER value.

```
plot(x = q_incr/n_pop, y = c_incr/n_pop,
     xlim = c(0, 1500/n_pop),
     ylim = c(0, 12e6/n_pop),
     pch = 16, cex = 1.5,
     xlab = "QALY difference",
     ylab = "Cost difference (£)",
     frame.plot = FALSE)
```

- Draw a willingness-to-pay threshold at £30,000.
- Are the results different to previously? How?

4.8 Age-dependent probability matrix

Now extend the `p_matrix_cycle()` function to include a dependence on age as well as cycle. The additional code looks like the following:

```
p_matrix_cycle <- function(p_matrix, age, cycle,
                           tpProg = 0.01,
                           tpDcm = 0.15,
                           effect = 0.5) {

 # time-dependent age lookup table
  tpDn_lookup <-
    c("(34,44]" = 0.0017,
      "(44,54]" = 0.0044,
      "(54,64]" = 0.0138,
      "(64,74]" = 0.0379,
      "(74,84]" = 0.0912,
      "(84,100]" = 0.1958)

  # discretize age in to age groups
  age_grp <- cut(age, breaks = c(34,44,54,64,74,84,100))

  # map an age group to a probability
  tpDn <- tpDn_lookup[age_grp]

  #############
  # insert here
  # same as previous transition probs
  #############
}
```

- Use the age-dependent probability matrix function in the code for simulating the Markov model and compute the cost-effectiveness outputs. Are they different? How?

5

A Markov model probability sensitivity analysis (PSA)

The formulation in the previous section can be extended to include uncertainty about one or more of the parameters. Briggs (2000) describe this for the current model by repeating the analytical solution of the model employing different values for the underlying parameters sampled from specified ranges and distributions. Sensitivity analyses can be performed one at a time (one-way) or for multiple parameters simultaneously (multi-way). This section presents a multi-way probabilistic sensitivity analysis.

Performing a PSA analysis can be done by inputting random draws from the unit costs and QALY distributions as inputs to the existing model function. In R, there are numerous ways of implementing a PSA. Following from the R code presented in the previous sections, we can wrap this model code in a function, e.g. called `ce_markov()`, which we can then repeatedly call with different parameter values. To this we will need to pass the starting conditions: population (`start_pop`), age (`init_age`) and number of cycles (`n_cycle`) (in our case, if not defined then age and number of cycles are assigned default values). We will also need the probability transition matrix (`p_matrix`), state cost and QALY matrices (`state_c_matrix`, `state_q_matrix`, `trans_c_matrix`).

In extension to the first analysis, the unit values have distributions rather than point values. To sample from a base R distribution the function name syntax is a short form version of the distribution name preceded by and `r` (for random or realisation). For example, to sample from a normal distribution then call `rnorm()`. We could sample all of the random numbers before running the model which would allow us to save them to use again and improve run time because this would only be performed once outside of the main loop. Alternatively, we can sample the random variables at runtime, within the Markov model function. This is arguably neater and if we wish to replicate a particular run then we can set the random seed beforehand with `set.seed()`. We will demonstrate how to implement a simple version when sampling at runtime.

```
ce_markov <- function(start_pop,
                      p_matrix,
                      state_c_matrix,
                      trans_c_matrix,
                      state_q_matrix,
                      n_cycles = 46,
                      init_age = 55,
                      s_names = NULL,
                      t_names = NULL) {

  n_states <- length(start_pop)
  n_treat <- dim(p_matrix)[3]

  pop <- array(data = NA,
               dim = c(n_states, n_cycles, n_treat),
               dimnames = list(state = s_names,
                               cycle = NULL,
                               treatment = t_names))
  trans <- array(data = NA,
                 dim = c(n_states, n_cycles, n_treat),
                 dimnames = list(state = s_names,
                                 cycle = NULL,
                                 treatment = t_names))
```

```
for (i in 1:n_states) {
  pop[i, cycle = 1, ] <- start_pop[i]
}

cycle_empty_array <-
  array(NA,
        dim = c(n_treat, n_cycles),
        dimnames = list(treatment = t_names,
                        cycle = NULL))

cycle_state_costs <- cycle_trans_costs <- cycle_empty_array
cycle_costs <- cycle_QALYs <- cycle_empty_array
LE <- LYs <- cycle_empty_array    # life-expectancy; life-years
cycle_QALE <- cycle_empty_array   # qaly-adjusted life-years

total_costs <- setNames(rep(NA, n_treat), t_names)
total_QALYs <- setNames(rep(NA, n_treat), t_names)

for (i in 1:n_treat) {

  age <- init_age

  for (j in 2:n_cycles) {

    # difference from point estimate case
    # pass in functions for random sample
    # rather than fixed values
    p_matrix <- p_matrix_cycle(p_matrix, age, j - 1,
                               tpProg = tpProg(),
                               tpDcm = tpDcm(),
                               effect = effect())

    # Matrix multiplication
    pop[, cycle = j, treatment = i] <-
      pop[, cycle = j - 1, treatment = i] %*% p_matrix[, , treatment = i]

    trans[, cycle = j, treatment = i] <-
    pop[, cycle = j - 1, treatment = i] %*% (trans_c_matrix * p_matrix[, , treatment = i])

    age <- age + 1
  }

  cycle_state_costs[i, ] <-
  (state_c_matrix[treatment = i, ] %*% pop[, , treatment = i]) * 1/(1 + cDr)^(1:n_cycles - 1)

  cycle_trans_costs[i, ] <-
    (c(1,1,1) %*% trans[, , treatment = i]) * 1/(1 + cDr)^(1:n_cycles - 2)

  cycle_costs[i, ] <- cycle_state_costs[i, ] + cycle_trans_costs[i, ]

  LE[i, ] <- c(1,1,0) %*% pop[, , treatment = i]

  LYs[i, ] <- LE[i, ] * 1/(1 + oDr)^(1:n_cycles - 1)

  cycle_QALE[i, ] <-
   state_q_matrix[treatment = i, ] %*%  pop[, , treatment = i]

  cycle_QALYs[i, ] <- cycle_QALE[i, ] * 1/(1 + oDr)^(1:n_cycles - 1)

  total_costs[i] <- sum(cycle_costs[treatment = i, -1])
  total_QALYs[i] <- sum(cycle_QALYs[treatment = i, -1])
```

```
  }

  list(pop = pop,
       cycle_costs = cycle_costs,
       cycle_QALYs = cycle_QALYs,
       total_costs = total_costs,
       total_QALYs = total_QALYs)
}
```

Because we will want to sample more than once, rather than just once at the start of the simulation, we can wrap the random sampling statements in a function so that they are called newly every time the Markov model is run. So, using the same names as we used for the point values in the previous analysis

```
# replace point values with functions to random sample

cAsymp <- function() rnorm(1, 500, 127.55)
cDeath <- function() rnorm(1, 1000, 255.11)
cDrug  <- function() rnorm(1, 1000, 102.04)
cProg  <- function() rnorm(1, 3000, 510.21)
effect <- function() rnorm(1, 0.5, 0.051)
tpDcm  <- function() rbeta(1, 29, 167)
tpProg <- function() rbeta(1, 15, 1506)
uAsymp <- function() rbeta(1, 69, 4)
```

- What is `uProg`, taken from the spreadsheet? What distribution does it have?

Similarly, rather than using fixed `state_c_matrix`, `trans_c_matrix` and `state_q_matrix`, if we define these as functions, we can sample newly their component values each time they are called. In practice, the code looks the same as previously but now the unit values are function calls so are followed by open and closed brackets.

```
# Define cost and QALYs as functions

state_c_matrix <- function() {
  matrix(c(cAsymp(), cProg(), 0,            # without drug
           cAsymp() + cDrug(), cProg(), 0), # with drug
           byrow = TRUE,
           nrow = n_treatments,
           dimnames = list(t_names,
                           s_names))
}
```

- Do the same modification for `state_q_matrix` and `trans_c_matrix`.

5.1 Run PSA analysis

To finally obtain the PSA output, loop over `ce_markov()` remembering to record the cost and QALYs outputs each time.

```
n_trials <- 500

costs <- matrix(NA, nrow = n_trials, ncol = n_treatments,
                dimnames = list(NULL, t_names))
qalys <- matrix(NA, nrow = n_trials, ncol = n_treatments,
                dimnames = list(NULL, t_names))

for (i in 1:n_trials) {
  ce_res <- ce_markov(start_pop = c(n_pop, 0, 0),
                      p_matrix,
                      state_c_matrix(),
                      trans_c_matrix(),
                      state_q_matrix())
```

```
  costs[i, ] <- ce_res$total_costs
  qalys[i, ] <- ce_res$total_QALYs
}
```

5.2 Plot results

```
# incremental costs and QALYs of with_drug vs to without_drug
c_incr_psa <- costs[, "with_drug"] - costs[, "without_drug"]
q_incr_psa <- qalys[, "with_drug"] - qalys[, "without_drug"]
```

- Plot the cost-effectiveness plane for the PSA output. You can base this on the simple case above. Indicate the ICER. How does this compare with the Excel spreadsheet?
- Use the `BCEA` package to create the standard cost-effectiveness plots.