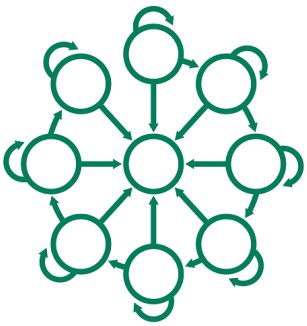

DECISION MODELLING FOR HEALTH ECONOMIC EVALUATION

Making Models Probabilistic

Advanced Course Module 2

Nichola Naylor & Jack Williams

May/June 2021 Course

Overview

The aim of this exercise is to demonstrate how a deterministic Markov model can be made probabilistic by fitting distributions to parameters. Emphasis is on choosing the correct distribution for different types of parameters and on correlating parameters using the Cholesky decomposition of a variance-covariance matrix. This Module will focus on coding the model to perform one probabilistic run (or simulation). In Module 3 we will look at storing and analysing the results of many model simulations.

The step-by-step guide below will take you through a number of stages of making the model probabilistic:

1. Setting up the 'Parameters' section for probabilistic analysis
2. Fitting Beta distributions to (constant) probability parameters
3. Fitting Gamma distributions to cost parameters
4. Fitting Beta distributions to utility parameters
5. Correlating parameters by Cholesky decomposition
6. Application to the regression model for prosthesis survival

A template for the exercise, 'A2.3.2_Making_Models_Probablistic_Template.R', is available, and builds upon the deterministic model developed in Module 1.

To work through this exercise, you will need to have the following csv files available within the same working-directory folder as the template and solution files:

- "hazardfunction.csv"
- "life-tables.csv"
- "cov55.csv"

Step by step guide

(1) Setting up the 'Parameters' section for probabilistic analysis

Before working on the code within this section, reacquaint yourself with the 'hazardfunction' data - these are the results from the regression analysis used in Module 1. You can also open the 'cov55' CSV file, which is the corresponding covariance matrix. Take a moment familiarise yourself with these data. You'll first need to read in these data files using the read.csv functions we've written at the top of the template file.

Firstly, within the 'Deterministic parameters section', we need to start by defining all parameters that are not drawn from distributions for the probabilistic analysis: patient characteristics (age & sex), discount rates and prosthesis costs. These are to be left deterministic, and the parameter values have been provided for you (these values are the same as in Module 1). Before starting on the exercise, ensure that you are totally happy with why these parameters will not form part of the probabilistic analysis.

Next, we will move to the 'Probabilistic parameters' section (Line 34), to sampling values from distributions.

(2) Fitting Beta distributions to (constant) probability parameters

The following information has been provided to you concerning operative mortality rates following primary surgery:

'The hospital records of a sample of 100 patients receiving a primary THR were examined retrospectively. Of these patients, two patients died either during or immediately following the procedure. The operative mortality for the procedure is therefore estimated to be 2%.'

1) Use this information to specify the parameters (alpha and beta) of a beta distribution for the probability of operative mortality during primary surgery. These are noted by "`a.`" and "`b.`" respectively. Your beta values for operative mortality should be equivalent to **98** (but make sure you understand how the number was derived!).

ii) Use the `rbeta()` function to generate one random draw from this distribution to define `tp.PTHR2dead`. Note that the rbeta function takes 3 arguments, first the number of draws or samples, the alpha parameter (named shape1) and the beta parameter (named shape2). If you are unsure of how the rbeta function works, check the following code to see a demonstration of five samples, before you begin assigning values in this exercise.

```
rbeta(n = 5, shape1 = 50, shape2 = 100)
```

```
## [1] 0.2398356 0.3304985 0.3374072 0.3685315 0.3911948
```

Notice that when you re-run `rbeta()` function, a different sample will be drawn each time (hence why it is probabilistic). We have shown you the code to draw 5 samples above, but for the `tp.PTHR2dead` parameter, ensure that you only assign one sampled value.

iii) Since no information is available on operative mortality following a revision procedure, use the same distribution again for this parameter (`tp.RTHR2dead`).

The following information has been provided to you concerning revision procedures among patients already having had a revision operation (i.e. the re-revision risk).

'The hospital records of a sample of 100 patients having experienced a revision procedure to replace a failed primary THR were reviewed at one year. During this time, four patients had undergone a further revision procedure.'

iv) Use this information to fit a constant transition probability for the annual re-revision rate parameter (`tp.rrr`), using the same approach as for operative mortality above. Your beta value for re-revision risk should be equivalent to **96** .

You should now have three probabilistic parameters (`tp.PTHR2dead`, `tp.PTHR2dead` and `tp.rrr`), that should automatically generate a new value from the specified distribution when you rerun the random draw assignments for these values. You can test this by re-running this section of the code and seeing the different results.

v) Use the formula for the moments of a Beta distribution from the presentation in order to specify the mean and standard error for these distributions.

(3) **Fitting Gamma distributions to cost parameters**

Since the costs of prostheses are handled deterministically, the cost of the primary procedure is ignored (since it is assumed to be the same in each arm) and there is assumed to be no cost of monitoring successful procedures, there is only really one probabilistic cost to specify for the model – the cost of the revision procedure.

You are given the following information regarding the cost of revising a failed primary THR:

'A number of units involved in THR were reviewed and the average cost of a revision procedure was found to be £5,294 with a standard error of £1,487.'

i) Define the mean and standard error values for costs of a revision procedure.

ii) Use the method of moments approach described in the presentation to calculate the parameters (alpha and beta) of a Gamma distribution that has the corresponding mean and standard error. The formulae for alpha and beta are also provided below, where $\overline{m}$ equals the sample mean, and s equals the standard error.

$$\alpha = \frac{\overline{m}^2}{s^2}, \qquad \beta = \frac{s^2}{\overline{m}}$$

iii) Use the `rgamma()` function to generate a random draw from this distribution. *(Hint: the base R functions for distributions tend to use the same arguments: number of draws or samples, followed by the shape and/or scale parameters)*

An alternative approach would have been to fit a normal distribution for this parameter – do you think this would be valid in this case?

iv) Create a vector of costs that holds the costs for each state within the model (remember to match the order defined by `state.names`).

(4) **Fitting Beta distributions to utility parameters**

As was indicated in the presentation, there are two methods to fitting the beta distribution. Here the second approach is taken using the method of moments to fit beta distributions to the utility data.

You are given the following information on utilities for patients experiencing different states of the model. The following information has been provided to you concerning operative mortality rates following primary surgery:

'A study was instigated to explore the utility weights subjects placed on different outcomes of THR related to the states of the Markov model – the following results were calculated in terms of mean (se) by state:'

'Successful primary – 0.85 (0.03)'

'Successful revision – 0.75 (0.04)'

'Revision – 0.3 (0.03)'

i) Enter the means and standard errors for these parameters in the `mn.` and `se.` variables in the template file.

Next, you need to obtain the Beta parameters corresponding to these means and standard errors by method of moments. The formulae are provided below (and are also provided in the videos).

$$(\alpha + \beta) = \frac{\overline{m}(1 - \overline{m})}{s^2} - 1$$

$$\alpha = \overline{m}(\alpha + \beta)$$

ii) Use the `rbeta()` function in cells to generate random draws from these distributions. Do this first for utility of successful primary prosthesis, and repeat for the 2 other utility values that need specifying.

iii) Create a vector of utility values for the states of the model (matching the `state.names` order of health states).

You should now have the following vectors defined (although note the numbers for costs and utilities will not be exactly the same to yours, as these are taken from a random draw process):

```
print(state.names)
```

```
## [1] "P-THR"        "successP-THR" "R-THR"        "successR-THR" "Death"
```

```
print(seed)
```

```
## [1] 1 0 0 0 0
```

```
print(state.costs)
```

```
## [1]    0.00    0.00 7923.98    0.00    0.00
```

```
print(state.utilities)
```

```
## [1] 0.0000000 0.8832146 0.2809115 0.7185653 0.0000000
```

(5) **Application of Cholesky to the regression model for prosthesis survival**

We now implement the Cholesky decomposition matrix method in the model. Look at the 'Hazard function' section, which gives the results of a regression analysis for the hazard ratio. You have used these results before when constructing the deterministic model, but now we have added the covariance matrix for the parameters. It should be clear that, while some parameters do not have a strong relationship, there are strong relationships within this set of parameters – particularly between the regression constant and the other parameters.

i) Note that the covariance matrix you have been given is from the `cov55.csv` file we read in at the beginning of this exercise. To turn this into a lower triangular Cholesky decomposition of the covariance matrix we have presented 2 options; (1) in the main script you can use the `chol()` function (we've coded this for you, note that you have to transpose the cov55 matrix using `t()` before reading in and then transpose back to get the correct format of the lower triangular Cholesky decomposition, this is due to the way the function reads in and outputs data), or (2) to understand what is actually happening within this function, you can also take a look at the `cholesky_decomposition.R` script provided within this course to see the manual calculation of the cholesky decomposition matrix. Take a few moments to understand how this was obtained.

ii) Create a vector (`z`) of 5 random values drawn from the standard normal distribution using the `rnorm()` function. This should have a mean of 0 and a standard deviation of 1.

iii) Enter the solution of the `Tz` matrix calculation: that is the Cholesky decomposition matrix multiplied by the vector of standard normal variates *(Hint: use `%*%`).*

iv) Now add the estimated mean values from the regression to `Tz` to create `x`.

v) You have now created a vector of five multivariate normal parameters that are correlated according to the estimated covariance matrix. Below is a data frame showing the numbers from the three vectors you have created; `Tz`, `mn` and `x` (again, the actual numbers will be different given that Tz is based on z, which is from a random draw):

```
##                    Tz      mean          x
## lngamma -0.005627919  0.374097  0.3684691
## cons     0.127214180 -5.490935 -5.3637208
## age     -0.002649902 -0.036702 -0.0393519
## male    -0.157319856  0.768536  0.6112161
## NP1      0.831865219 -1.344474 -0.5126088
```

The final task for this section is to link the probabilistic transition probabilities to the relevant vector you have just created.

vi) Remember that we have been working on the log hazard scale for the Weibull regression. We need to make sure that the model parameters are exponentiated. Using the parameters just created assign values for `lambda` [Lambda parameter survival analysis (depends on chosen mix of above coefficients)], `gamma` [Ancillary parameter in Weibull distribution] and `rr.NP1` [Relative risk of revision for new prosthesis 1 compared to standard].

(6) Life-table transitions for background mortality

As we described in Module 1, we can still employ time dependent transitions in other model state providing the time dependency relates to time in the model rather than time in the state itself. This is the case for the background mortality rate, which depends on the age of a subject rather than which state of the model they are in.

Open the 'Life tables' section of the code, first define vectors for the cycle numbers and cohort age throughout the model. Then re-familiarise yourself with the `findInterval()` function, and how this is used to create the 'death.risk' data.frame, by selecting the appropriate mortality values for each cycle (based on the cohort age at that cycle).

(7) Building the Markov model

Using the same approach taken in Module 1 we can build the time-dependent transitions for standard and new prosthesis pathways, based on the inputs you've created above. We've ordered the model to first build the transition matrices for each treatment arm, then build the cost and effect matrices to multiply through (but the order doesn't really matter in terms of construction and is down to preference). The model structure (transition matrix and trace matrix creation and outcome expected value calculations) are the same as the model you built in Module 1. Run this section of the script before moving on to the final section of this exercise.

(8) **Estimating cost-effectiveness**

The final section of this exercise is very straightforward. Calculate the incremental cost, incremental effect and the ICER within the `output` data.frame, as you did for Module 1.

That's it! Your Markov model is now complete. Re-run the script a few times to see the different outputs. Have a play with the patient characteristics to see how these influence the results. Make sure you are clear in your own mind why patient characteristics influence the results. In the next module we'll be exploring how to store and plot the results of such analyses, so it's important you understand why the results change over different runs of this exercise.

Additional notes

We've turned this probabilistic model into a simulation model in the following R script:

- A2.3.4_Additional_Information_THR_Model_with_simulation.R"

In this example, the THR model is wrapped up in a function - so that the function can run the whole model script, the model code within the function is the same as that in this exercise. Once the model is in the function, the function can be called to easily produce the model results.

We can then create a loop to perform a large number of model simulations, and store the results of these simulations, for the probabilistic sensitivity analysis. It's worth looking through the structure of this, we'll be doing something similar in the next course exercises. As with many things in R, there will be many ways to run the probabilistic models over a number of runs. "A2.3.4_Additional_Information_THR_Model_with_simulation.R" starts off by simply wrapping everything into one function. This is perhaps the easiest, but least efficient way to run many simulations, but is an ideal starting point for this course. In later exercises we will show you alternative ways in which this can be done to increase the efficiency of the model runs by structuring our scripts and functions differently. Run the function a few times to see the different results.