Practical 9. Markov models — SOLUTIONS

Lecture 09 PDF version

2. Individual level data on event history and Markov models

In this case, we consider individual level data, e.g. derived from a randomised study, in which patients have been:

1. Randomised to either standard of care ($t = 0$) or an innovative immuno-oncologic drug;
2. Followed up for a period of time during which their "event history" has been recorded. In particular, we know whether or not the individuals have "*progressed*" to a more serious stage of the cancer (and in that case, the time since enrollement at which this has been confirmed), as well as whether or not the individual has "*died*" (again with the exact time of death being recorded).

As shown in class, the data look like this.

Patient	Treatment	Progression?	Death?	Progression time	Death time
1	1	1	0	31.99	32.00
2	1	1	0	30.55	30.60
...	...	...	...	...	...
10	1	1	1	0.17	0.46
11	1	1	1	1.27	1.57
...	...	...	...	...	...

The time is recorded in months; from the modelling point of view, this is not a problem; however, for the sake of running the Markov model for a "lifetime horizon", it may be more efficient to convert the times in, say, years — this means that the values are rescaled (by 12, in this case) and so the overall number of cycles becomes smaller (so, instead of running the Markov model for 120 months, which implies a relatively large computational burden), we can do it for 10 years.

As shown in class, this dataset has the advantage of using the nature of the data — for each individual we know whether and when they experience the two events of interest (progression and death). This is not possible when we use digitised data on PFS and OS separately. However to run the analysis, we need to convert the data to a suitable format (often referred to as "multistate"). For example, we can use the tidyverse and re-arrange the data in a very efficient way.

```
# 'tidyverse' code to re-arrange the data in "multistate" format
msmdata=
  # Transition Pre to Post
  data %>% mutate(                                                    # use the original data a
  id=patid,                                                           # patient ID
  from=1,                                                             # starting state (1="Pre-
  to=2,                                                               # arriving state (2="Prog
  trans=1,                                                            # transition ID (1="Pre-p
  Tstart=0,                                                           # entry time
  Tstop=prog_t,                                                       # exit time (time at whic
  time=Tstop-Tstart,                                                  # observed time
  status=case_when(                                                   # event indicator
    prog==1~1,                                                        #   if progressed then 1
    TRUE~0                                                            #   otherwise 0 (censored
  ),
  treat=treat                                                         # treatment arm
) %>% select(id,from,to,trans,Tstart,Tstop,time,status,treat) %>%     # selects only the releva
  bind_rows(                                                          # stack these new rows be
  # Transition Pre to Death
    data %>% mutate(                                                  # use the original data a
      id=patid,                                                       # patient ID
      from=1,                                                         # starting state (1="Pre-
      to=3,                                                           # arriving state (3="Deat
      trans=2,                                                        # transition ID (2="Pre-p
      Tstart=0,                                                       # entry time
      Tstop=death_t,                                                  # exit time (time at whic
      time=Tstop-Tstart,                                              # observed time
      status=case_when(                                               # event indicator
        (death==1 & prog_t==death_t)~1,                               #   if death then 1
        TRUE~0                                                        #   otherwise 0 (censored
      ),
      treat=treat                                                     # treatment arm
    ) %>% select(id,from,to,trans,Tstart,Tstop,time,status,treat)     # selects only the releva
  ) %>%
  bind_rows(                                                          # stack these new rows be
  # Transition Post to Death
    data %>% filter(prog==1) %>% mutate(                              # use the original data,
      id=patid,                                                       # patient ID
      from=2,                                                         # starting state (2="Prog
      to=3,                                                           # arriving state (3="Deat
      trans=3,                                                        # transition ID (3="Progr
      Tstart=prog_t,                                                  # entry time. NB: this ti
      Tstop=death_t,                                                  # exit time (time at whic
      time=Tstop-Tstart,                                              # observed time
      status=case_when(                                               # event indicator
        death==1~1,                                                   #   if death then 1
        TRUE~0                                                        #   otherwise 0 (censored
      ),
      treat=treat                                                     # treatment arm
    ) %>% select(id,from,to,trans,Tstart,Tstop,time,status,treat)     # selects only the releva
  ) %>% arrange(id,trans)

# Visualise the data
msmdata
```

```
## # A tibble: 1,868 × 9
##       id  from    to trans Tstart Tstop     time status treat
##    <int> <dbl> <dbl> <dbl>  <dbl> <dbl>    <dbl>  <dbl> <int>
##  1     1     1     2     1    0    32.0  32.0          1     1
##  2     1     1     3     2    0    32    32            0     1
##  3     1     2     3     3   32.0 32     0.00920      0     1
##  4     2     1     2     1    0    30.6  30.6          1     1
##  5     2     1     3     2    0    30.6  30.6          0     1
##  6     2     2     3     3   30.6 30.6   0.0476       0     1
##  7     3     1     2     1    0    27.9  27.9          1     1
##  8     3     1     3     2    0    28    28            0     1
##  9     3     2     3     3   27.9 28     0.0944       0     1
## 10     4     1     2     1    0     2.88  2.88         1     0
## # … with 1,858 more rows
```

We can now run three separate survival analyses for each of the three transitions, for instance using `survHE` (but you don't have to do this — in case you need it, notice that `survHE` is installed in the Binder remote server). This step is **much** more complicated than shown here. For example, we would need to test several parametric models (as discussed in Lecture 6). Then we would need to validate the different models — on the basis of their fit to the observed data, but more importantly in relation to the **extrapolation** over times that have not been observed.

In this case, we can simplify things and assume that, for all three models (PFS, OS and OS after progression, which we indicate respectively as `m_12`, `m_13` and `m_23` — the reason for this terminology will be obvious later), we select a Gompertz distribution as the best one. In `survHE`, the models would be fitted using the following commands.

```
# Loads survHE
library(survHE)
```

```
## Loading required package: flexsurv
```

```
## Loading required package: survival
```

```
##
## Attaching package: 'survHE'
```

```
## The following objects are masked _by_ '.GlobalEnv':
##
##     data, msmdata
```

```
# Runs survival models on the specific subsets to obtain estimate of the various transition probabi
m_12=fit.models(Surv(time,status)~as.factor(treat),             # model 'formula': defines the tim
                data=msmdata %>% filter(trans==1),              # subsets the msmdata by filtering
                distr="gom",                                    # selects the Gompertz model
                method="hmc",                                   # instructs R to use HMC/Bayesian
                priors=list(gom=list(a_alpha=1.5,b_alpha=1.5))) # specifies the informative prior

m_13=fit.models(Surv(time,status)~as.factor(treat),             # model 'formula': defines the tim
                data=msmdata %>% filter(trans==2),              # subsets the msmdata by filtering
                distr="gom",                                    # selects the Gompertz model
                method="hmc",                                   # instructs R to use HMC/Bayesian
                priors=list(gom=list(a_alpha=1.5,b_alpha=1.5))) # specifies the informative prior

m_23=fit.models(Surv(time,status)~as.factor(treat),             # model 'formula': defines the tim
                data=msmdata %>% filter(trans==3),              # subsets the msmdata by filtering
                distr="gom",                                    # selects the Gompertz model
                method="hmc",                                   # instructs R to use HMC/Bayesian
                priors=list(gom=list(a_alpha=1.5,b_alpha=1.5))) # specifies the informative prior
```

When we fit these models, we obtain the estimates for the model parameters (which in the case of the Gompertz are the shape α and the rate μ). These can be used to reconstruct the full survival curves for an arbitrary time interval — for example in the case of the Gompertz model, for any given time t, the survival curve is

$$S(t) = 1 - \exp\left(-\frac{\mu}{\alpha}\exp(\alpha t) - 1\right).$$

In `survHE` the function `make.surv` can be used to construct `nsim` simulations of the survival curves for any specific interval of time `t=...`. Once these are obtained, we can use them to compute the approximated transition probabilities using the formula

$$\lambda_{s's j} \approx 1 - \frac{S_{t+k}}{S_t}$$

for a given pair of times t and $t+k$.

The script provided contains a few functions that code up this process. We can load up the functions contained in the script `survHE_utils.R`; these essentially use `dplyr` and `ggplot2` to manipulate the output of the models using the following steps.

1. Compute the survival curves using the `make.surv` function in `survHE`
2. Use the approximation formula to translate these into the probabilities λ_{12} (transition from Pre-progression to Progressed), λ_{13} (transition from Pre-progressed to Death) and λ_{23} (transition from Progressed to Death).
3. Run the specialised function `three_state_mm` that:

- first completes the transition matrix by retrieving the remaining transition probabilities ($\lambda_{11} = 1 - \lambda_{12} - \lambda_{13}$ and $\lambda_{22} = 1 - \lambda_{23}$;
- then move people around according to the Markov matrix algebra.

4. Run the specialised function `markov_trace` to manipulate the resulting state occupancy tibble and then use `ggplot2` to plot the Markov trace.

```
# Sources the specialised functions
source("survHE_utils.R")

# Then run the Markov model using the specialised function 'three_state_mm'
mm=three_state_mm(
  m_12,m_13,m_23,          # these are the three objects containing the parameters estimates
  t=seq(0,130),            # specifies that the Markov model needs to be run for discrete times from
  nsim=1,                  # only uses 1 simulation from the distribution of the model parameters (t
  start=c(1000,0,0)        # initial population: 1000 in pre-progression, 0 in progressed and 0 in d
)
```

We can visualise the resulting object

```
mm
```

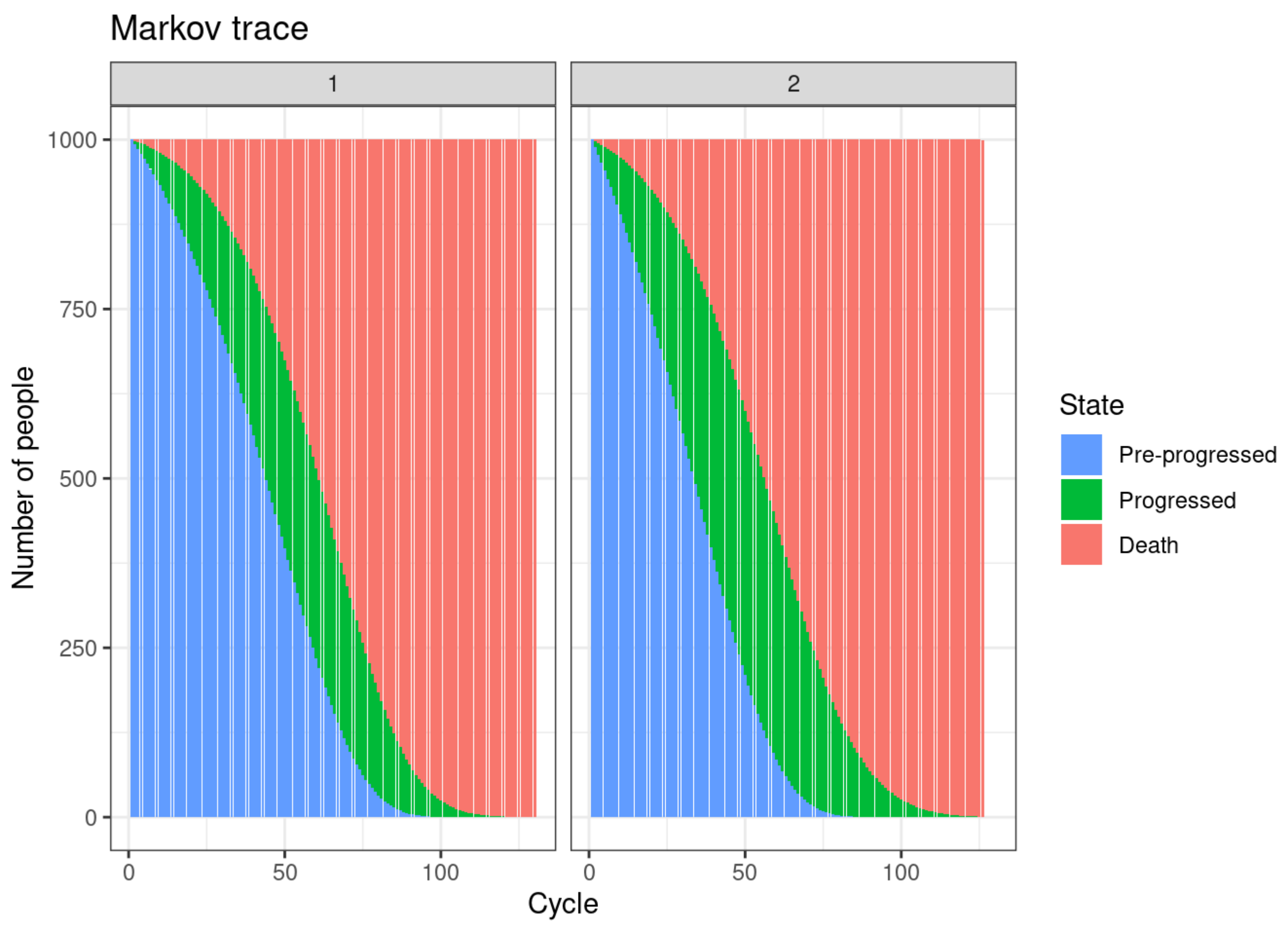
```
## $m
## # A tibble: 260 × 11
##    treat     t `Pre-progressed` Progressed Death sim_idx lambda_11 lambda_12
##    <int> <int>            <dbl>      <dbl> <dbl>   <int>     <dbl>     <dbl>
##  1     1     1            1000         0     0         1     0.993   0.00506
##  2     1     2             993.        5.06  1.67      1     0.993   0.00527
##  3     1     3             986.       10.2   3.45      1     0.993   0.00550
##  4     1     4             979.       15.4   5.33      1     0.993   0.00574
##  5     1     5             972.       20.7   7.32      1     0.992   0.00599
##  6     1     6             964.       26.1   9.43      1     0.992   0.00625
##  7     1     7             957.       31.6  11.7       1     0.992   0.00652
##  8     1     8             949.       37.2  14.0       1     0.991   0.00680
##  9     1     9             941.       42.8  16.5       1     0.991   0.00710
## 10     1    10             932.       48.6  19.1       1     0.991   0.00740
## # … with 250 more rows, and 3 more variables: lambda_13 <dbl>, lambda_22 <dbl>,
## #   lambda_23 <dbl>
##
## $running_time
## Time difference of 0.1350291 secs
##
## $base_case
## NULL
```

This includes three elements:

1. The tibble `m`; this includes the state occupancy as well as the value of the relevant transition probabilities for each time point in the "virtual follow up".
2. The running time; this is a function of the number of simulations used — in this case, we're only using `nsim`=1 and so the computation is very fast. Note that in general, running this `R` is more efficient than any implementation in spreadsheets.
3. The user can instruct `R` to also compute the Markov model for the "base-case" scenario, which is essentially the same as the case with `nsim`=1. So in this case, the element `base_case` is not computed and it is set to `NULL`.

Finally, we can visualise the results using the following code.

```
markov_trace(mm)
```



Note on using survHE

The code above uses the current `CRAN` version of `survHE`, which you can install by using the `R` command `install.packages('survHE')` or from the GitHub repository `remotes::install_github('giabaio/survHE')` – assuming you have installed `remotes` (check here for more details on `remotes`).

`survHE` is a bit of a complicated package – it's not like most of its functions are difficult, it's just that it is designed to "wrap up" complex packages doing the survival modelling. In particular, the two Bayesian versions are performed using very structured and heavy `R` packages (`rstan` and `INLA`), which means that its installation can be long. If you're on the Binder VM, the installation of the whole thing may break it. In that case, you can resort to doing a "cheat" and installing the frequentist module only. You can do this by using the `R` command `remotes::install_github("giabaio/survHE",ref="devel")`. Note that in this case we use the extra option `ref="devel"`, which instructs `R` to install the version contained in the GitHub branch named `devel`, which contains the code to run the frequentist version of the models, only.

In this case, you will need to slightly modify the code above, to remove the reference to the `"hmc"` method. For instance, you need to modify the call to `fit.models` to the following

```
m_12=fit.models(Surv(time,status)~as.factor(treat),            # model 'formula': defines the tim
                data=msmdata %>% filter(trans==1),             # subsets the msmdata by filtering
                distr="gom"                                    # selects the Gompertz model
)
```

(notice the removal to the `prior` argument too: this is a frequentist model, so there's no space for priors…).

NEXT
Cohort discrete Markov model