Simulation of multistate models with multiple timescales: `simLexis` in the `Epi` package

SDC
http://BendixCarstensen.com/Epi
Monday 16th September, 2013
Version 2

Compiled Monday 16th September, 2013, 11:50
from: `C:/stat/R/BxC/Examples/sim-Lexis.tex`

Bendix Carstensen Steno Diabetes Center, Gentofte, Denmark
& Department of Biostatistics, University of Copenhagen
bxc@steno.dk
http://BendixCarstensen.com

Contents

Chapter 1

Using `simLexis`

1.1 Introduction

This vignette explains the machinery behind simulation of life histories through multistate models implemented in `simLexis`. In `simLexis` transition rates are allowed to depend on multiple time scales, including timescales defined as time since entry to a particular state (duration). This therefore also covers the case where time *at* entry into a state is an explanatory variable for the rates, since time at entry is merely time minus duration. Thus, the set-up here goes beyond Markov- and semi-Markov-models, and brings simulation based estimation of state-occupancy probabilities into the realm of realistic multistate models.

The basic idea is to simulate a new `Lexis` object [3, 1] as defined in the `Epi` package for R, based on 1) a multistate model defined by its states and the transition rates between them and 2) an initial population of individuals.

Thus the output will be a `Lexis` object describing the transitions of a predefined set of persons through a multistate model. Therefore, if persons starting are defined to be identical at start, then calculation of the probability of being in a particular state at a given time boils down to a simple enumeration of the persons in a particular state. Bar of course the (binomial) simulation error, but this can be brought down by simulation a sufficiently large number of persons.

An observed `Lexis` object with follow-up of persons through a number of states will normally be the basis for estimation of transition rates between states, and thus will contains all information about covariates determining the occurrence rates, in particular the *timescales* [2]. Hence, the natural input to simulation from an estimated multistate model will be an object of the same structure as the original observed. Since transitions and times are what is simulated, any values of `lex.Xst` and `lex.dur` in the input object will of course be ignored.

This first chapter of this vignette shows by an example how to use the function `simLexis` and display the results. The subsequent chapter discusses in more detail how the simulation machinery is implemented and is not needed for the practical use of `simLexis`.

1.2 `simLexis` in practice

This section is merely a commented walk-trough of the example from the help-page of `simLexis`, with a larger number of simulated persons in order to minimize the pure

simulation variation. When we want to simulate transition times through a multistate model where transition rates may depend on time since entry to the current or a previous state, it is essential that we have a machinery to keep track of the transition time on *all* time scales, as well as a mechanism that can initiate a new time scale to 0 when a transition occurs to a state where we shall use time since entry as determinant of exit rates from that state.

1.2.1 Simulation in a multistate model

Input for simulation of a single trajectory through a multistate model is a representation of the *current status* of a person, so it can basically be represented by a `Lexis` object where `lex.dur` and `lex.Xst` are ignored, since there is no follow-up (yet). The object that we supply to the simulation function must therefore contain information about all timescales of interest and which one of these that are defined as time since entry into a new state. This information is assumed to be in the attributes `time.scale` and `time.since` respectively. We shall call such an object as `preLexis` object.

Thus there are two main arguments to a function to simulate from a multistate model which is represented in a (pre-)`Lexis` object:

1. A (pre-)`Lexis` object representing the initial states and covariates of the population to be simulated. This has to have the same structure as the original `Lexis` object representing the multistate model. Except that values for `lex.Xst` and `lex.dur` are not required (since these are the quantities that will be simulated).

2. A transition object, representing the transition intensities between states. This is a list of lists of intensity representations. As an intensity representation we mean a function that for given a `preLexis` object produces estimates of the transition intensities at a set of supplied times since entry.

   The names of the elements (which are lists) of the transition object will be names of the *transient* states, that is the states *from* which a transition can occur. The names of the elements of each of these lists are the names of states *to* which transitions can occur (which may be either transient or absorbing states).

   If the transition object is called `Tr` then `TR$From1$To2` (or `Tr[["From1"]][["To2"]]`) will represent the transition intensity from state "From1" to the state "To2".

   Currently, these entries in the transition object must be `glm` objects, representing Poisson models for the transitions. These will normally be estimated using a time-split data-set, but it is really immaterial how they came about.

In addition to these two input items, there will be a couple of tuning parameters, which we will seek to give sensible defaults.

The output of the function will simply be a `Lexis` object with simulated transitions between states. This will be the basis for deriving sensible statistics from the `Lexis` object — see next section.

We start by loading the `Epi` package:

```
> library( Epi )
> print( sessionInfo(), L=F )
```

```
R version 3.0.1 (2013-05-16)
Platform: i386-w64-mingw32/i386 (32-bit)

locale:
[1] LC_COLLATE=Danish_Denmark.1252  LC_CTYPE=Danish_Denmark.1252
[3] LC_MONETARY=Danish_Denmark.1252 LC_NUMERIC=C
[5] LC_TIME=Danish_Denmark.1252

attached base packages:
[1] utils     datasets  graphics  grDevices stats     methods   base

other attached packages:
[1] Epi_1.1.56     foreign_0.8-53

loaded via a namespace (and not attached):
[1] tools_3.0.1
```

1.3 Setting up a Lexis object

As an example we will use the DMlate dataset from the Epi package; it is a dataset simulated to resemble a random sample of 10,000 patients from the Danish National Diabetes Register. First we load the diabetes data and set up a simple illness-death model:

```
> data(DMlate)
> dml <- Lexis( entry = list(Per=dodm, Age=dodm-dobth, DMdur=0 ),
+               exit = list(Per=dox),
+         exit.status = factor(!is.na(dodth),labels=c("DM","Dead")),
+               data = DMlate )


NOTE: entry.status has been set to "DM" for all.
```

This is just data for a simple survival model with states "DM" and "Dead". Now we cut the follow-up at insulin start, which for the majority of patients (T2D) is a clinical indicator of deterioration of disease regulation. We therefore also introduce a new timescale, and split the non-precursor states, so that we can address the question of ever having been on insulin:

```
> dmi <- cutLexis( dml, cut = dml$doins,
+                       pre = "DM",
+                 new.state = "Ins",
+                 new.scale = "t.Ins",
+              split.states = TRUE )
> summary( dmi )


Transitions:
     To
From    DM  Ins Dead Dead(Ins)  Records:  Events: Risk time:  Persons:
  DM  6157 1694 2048         0      9899     3742   45885.49      9899
  Ins    0 1340    0       451      1791      451    8387.77      1791
  Sum 6157 3034 2048       451     11690     4193   54273.27      9996


> str(dmi)
```

```
Classes 'Lexis' and 'data.frame':       11690 obs. of  15 variables:
 $ Per    : num  1999 2003 2005 2009 2009 ...
 $ Age    : num  58.7 64.1 86.3 44 75.8 ...
 $ DMdur  : num  0 0 0 0 0 0 0 0 0 0 ...
 $ t.Ins  : num  NA NA NA NA NA NA NA NA NA NA ...
 $ lex.dur: num  11.08 6.689 5.446 0.736 1.344 ...
 $ lex.Cst: Factor w/ 4 levels "DM","Ins","Dead",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ lex.Xst: Factor w/ 4 levels "DM","Ins","Dead",..: 1 1 1 1 1 3 1 1 3 1 ...
 $ lex.id : int  1 2 3 4 5 6 7 8 9 10 ...
 $ sex    : Factor w/ 2 levels "M","F": 2 1 2 2 1 2 1 1 2 1 ...
 $ dobth  : num  1940 1939 1918 1965 1933 ...
 $ dodm   : num  1999 2003 2005 2009 2009 ...
 $ dodth  : num  NA NA NA NA NA ...
 $ dooad  : num  NA 2007 NA NA NA ...
 $ doins  : num  NA NA NA NA NA NA NA NA NA NA ...
 $ dox    : num  2010 2010 2010 2010 2010 ...
 - attr(*, "time.scales")= chr  "Per" "Age" "DMdur" "t.Ins"
 - attr(*, "time.since")= chr  "" "" "" "Ins"
 - attr(*, "breaks")=List of 4
  ..$ Per  : NULL
  ..$ Age  : NULL
  ..$ DMdur: NULL
  ..$ t.Ins: NULL
```

We can show how many person-years we have and show the number of transitions and transition rates (per 1000), using the `boxes.Lexis` function to display the states and the number of transitions between them:

```
> boxes( dmi, boxpos=list(x=c(20,20,80,80),
+                         y=c(80,20,80,20)),scale.R=1000 )
```

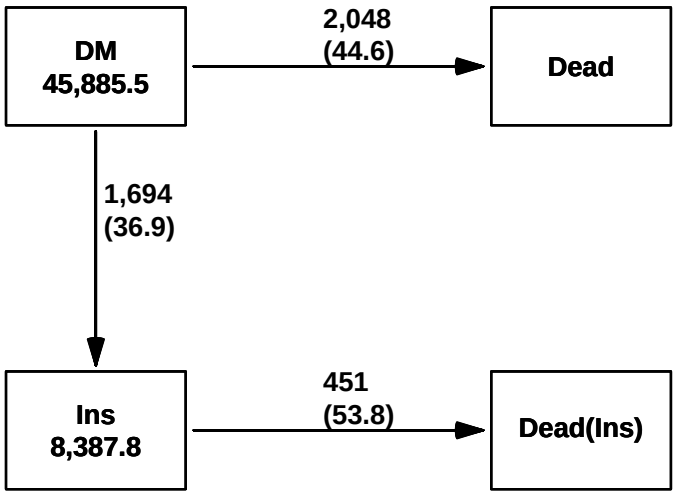

Figure 1.1: *Data overview for the* `dmi` *dataset. Number in the boxes are person-years, and numbers on the arrows are no. of transitions and rates (transition intensities) per 1000 PY.*

1.4 Analysis of rates

In the `Lexis` object (a data frame) each person is represented by one record for each transient state he occupies, thus in this case either 1 or 2 records, those who have a recorded time both without and with insulin have two records.

In order to be able to fit Poisson models with occurrence rates varying by the different time-scales, we split the follow-up in 6-month intervals for modeling:

```
> Si <- splitLexis( dmi, 0:30/2, "DMdur" )
> dim( Si )
```

```
[1] 115370     15
```

Note that when we split the follow-up each person's follow up now consists of many records, each with the *current* values of the timescales at the start of the interval represented by the record. In the modelling we must necessarily assume that the rates are constant within each 6-month interval, but the *size* of these rates we model as smooth functions of the time scales (that is the values at the beginning of each interval), and *not* with a separate parameter for each interval.

We shall use natural splines (restricted cubic splines) for the analysis of rates, and hence we must allocate knots for the splines. This is done for each of the time-scales, and separately for the states "DM" and "Ins". We place the knots so that the number of events is the same between each pair of knots, but only half of this beyond each of the boundary knots:

```
> nk <- 5
> ( ai.kn <- with( subset(Si,lex.Xst=="Ins"),
+                  quantile( Age+lex.dur  , probs=(1:nk-0.5)/nk ) ) )
```

```
     10%      30%      50%      70%      90%
23.75455 45.27279 56.62919 65.47851 77.50000
```

```
> ( ad.kn <- with( subset(Si,lex.Xst=="Dead"),
+                  quantile( Age+lex.dur  , probs=(1:nk-0.5)/nk ) ) )
```

```
     10%      30%      50%      70%      90%
61.91951 72.52731 78.43121 83.32348 90.15195
```

```
> ( di.kn <- with( subset(Si,lex.Xst=="Ins"),
+                  quantile( DMdur+lex.dur, probs=(1:nk-0.5)/nk ) ) )
```

```
 10%  30%  50%  70%  90%
 1.0  3.5  5.5  8.0 11.0
```

```
> ( dd.kn <- with( subset(Si,lex.Xst=="Dead"),
+                  quantile( DMdur+lex.dur, probs=(1:nk-0.5)/nk ) ) )
```

```
      10%       30%       50%       70%       90%
0.2715948 1.4182067 3.0759754 5.1526352 8.5979466
```

```
> ( ti.kn <- with( subset(Si,lex.Xst=="Dead(Ins)"),
+                  quantile( t.Ins+lex.dur, probs=(1:nk-0.5)/nk ) ) )
```

```
      10%       30%       50%       70%       90%
0.1341547 0.6324435 1.7768652 3.5619439 6.8008214
```

We now fit Poisson models to transition rates, using the wrapper `Ns` from the `Epi` package to simplify the specification of the rates:

```
> library( splines )
> DM.Ins <- glm( (lex.Xst=="Ins") ~ Ns( Age  , knots=ai.kn ) +
+                                   Ns( DMdur, knots=di.kn ) +
+                                   I(Per-2000) + sex,
+                family=poisson, offset=log(lex.dur),
+                data = subset(Si,lex.Cst=="DM") )
> DM.Dead <- glm( (lex.Xst=="Dead") ~ Ns( Age  , knots=ad.kn ) +
+                                     Ns( DMdur, knots=dd.kn ) +
+                                     I(Per-2000) + sex,
+                family=poisson, offset=log(lex.dur),
+                data = subset(Si,lex.Cst=="DM") )
> Ins.Dead <- glm( (lex.Xst=="Dead(Ins)") ~ Ns( Age  , knots=ad.kn ) +
+                                           Ns( DMdur, knots=dd.kn ) +
+                                           Ns( t.Ins, knots=ti.kn ) +
+                                           I(Per-2000) + sex,
+                family=poisson, offset=log(lex.dur),
+                data = subset(Si,lex.Cst=="Ins") )
```

1.4.1 Proportionality of mortality rates

Note that we have made separate models for the three transitions, there is no assumption of proportionality between the mortality rates from `DM` and `Ins`.

However, there is nothing that prevents us from testing this assumption; we can just fit a model for the mortality rates in the entire data frame `Si`, and compare the deviance from this with the sum of the deviances from the separate models:

```
> with( Si, table(lex.Cst) )
```

```
lex.Cst
       DM       Ins      Dead Dead(Ins)
    97039     18331         0         0
```

```
> All.Dead <- glm( (lex.Xst %in% c("Dead(Ins)","Dead")) ~
+                                           Ns( Age  , knots=ad.kn ) +
+                                           Ns( DMdur, knots=dd.kn ) +
+                                           lex.Cst +
+                                           I(Per-2000) + sex,
+                family=poisson, offset=log(lex.dur),
+                data = Si )
> round( ci.exp( All.Dead ), 3 )
```

```
                          exp(Est.)   2.5%  97.5%
(Intercept)                   0.041  0.037  0.046
Ns(Age, knots = ad.kn)1       4.119  3.478  4.878
Ns(Age, knots = ad.kn)2       4.653  4.055  5.340
Ns(Age, knots = ad.kn)3      15.461 13.575 17.608
Ns(Age, knots = ad.kn)4       7.528  6.710  8.446
Ns(DMdur, knots = dd.kn)1     0.685  0.573  0.818
```

```
Ns(DMdur, knots = dd.kn)2     0.845  0.734  0.972
Ns(DMdur, knots = dd.kn)3     0.388  0.309  0.487
Ns(DMdur, knots = dd.kn)4     0.958  0.851  1.079
lex.CstIns                    2.164  1.943  2.411
I(Per - 2000)                 0.965  0.954  0.977
sexF                          0.665  0.614  0.720
```

From these parameter values we would in a simple setting just claim that start of insulin-treatment was associated with a slightly more than doubling of mortality.

We can compare the fit of this model with the fit of the separate models for the two mortality rates, by adding up the deviances and d.f. from these:

```
> what <- c("null.deviance","df.null","deviance","df.residual")
> ( rD <- unlist(  DM.Dead[what] ) )
```

```
null.deviance       df.null      deviance   df.residual
     19957.95      97038.00      17853.93      97028.00
```

```
> ( rI <- unlist( Ins.Dead[what] ) )
```

```
null.deviance       df.null      deviance   df.residual
     4329.880     18330.000      3680.391     18316.000
```

```
> ( rA <- unlist( All.Dead[what] ) )
```

```
null.deviance       df.null      deviance   df.residual
     24300.15     115369.00      21612.94     115358.00
```

```
> round( c( dd <- rA-(rI+rD), "pVal"=1-pchisq(dd[3],dd[4]) ), 3 )
```

```
null.deviance       df.null      deviance   df.residual pVal.deviance
       12.314         1.000        78.616        14.000         0.000
```

Thus we see there is a substantial non-proportionality of mortality rates from the two states; we shall explore this quantitatively in more detail. Note that the reason that there is a difference in the null deviances (and a difference of 1 in the null d.f.) is that the null deviance of `All.Dead` refer to a model with a single intercept, that is a model with constant and *identical* mortality rates from the states "DM" and "Ins", whereas the null models for `DM.Dead` and `Ins.Dead` have *different* mortality rates from the states "DM" and "Ins".

1.4.2 How the mortality rates look

If we want to see how the mortality rates are modelled in `DM.Dead` and `Ins.Dead` in relation to `All.Dead`, we make a prediction of rates for say men diagnosed in different ages and going on insulin at different times after this. So we consider men diagnosed in ages 40, 50, 60 and 70, and who either never enter insulin treatment or do it 1, 3 or 5 years after diagnosis of DM.

So what we do is to create a prediction data frame where we have observation times from diagnosis and 12 years on (longer would not make sense as this is the extent of the data).

We start by setting up an array to hold the predicted mortality rates, classified by diabetes duration, age at onset, time of insulin onset, and of course type of model. What we want to do is to plot the age-specific mortality rates for persons not on insulin, and for person starting insulin at different times after DM. The mortality curves start at the age where the person gets diabetes and continues 12 years.

```
> pr.rates <- NArray( list( DMdur = seq(0,12,0.1),
+                           DMage = 4:7*10,
+                           r.Ins = c(NA,1,3,5),
+                           model = c("DM/Ins","All"),
+                            what = c("rate","lo","hi") ) )
> str( pr.rates )
```

```
 logi [1:121, 1:4, 1:4, 1:2, 1:3] NA NA NA NA NA NA ...
 - attr(*, "dimnames")=List of 5
  ..$ DMdur: chr [1:121] "0" "0.1" "0.2" "0.3" ...
  ..$ DMage: chr [1:4] "40" "50" "60" "70"
  ..$ r.Ins: chr [1:4] NA "1" "3" "5"
  ..$ model: chr [1:2] "DM/Ins" "All"
  ..$ what : chr [1:3] "rate" "lo" "hi"
```

For convenience we define a function that takes the predicted (log)-rates with s.e.s and converts them to true rates with c.i.:

```
> ci.pred <-
+ function( mod, newdata )
+ {
+ zz <- predict( mod, newdata=newdata, se.fit=TRUE, type="link" )
+ exp( cbind( zz$fit, zz$se.fit ) %*% ci.mat() )
+ }
```

Then we can set up the prediction data frame and modify it in loops over ages at onset and insulin onset. Note that we set `lex.dur` to 1000 in the prediction frame, so that we obtain rates in units of events per 1000 PY.

```
> nd <- data.frame( DMdur = as.numeric( dimnames(pr.rates)[[1]] ),
+                  lex.Cst = factor( 1, levels=1:4,
+                                    labels=levels(Si$lex.Cst) ),
+                      sex = factor( 1, levels=1:2, labels=c("M","F")),
+                  lex.dur = 1000 )
> for( ia in dimnames(pr.rates)[[2]] )
+    {
+ dnew <- transform( nd, Age = as.numeric(ia)+DMdur,
+                        Per = 1998+DMdur )
+ pr.rates[,ia,1,"DM/Ins",] <- ci.pred(  DM.Dead, newdata = dnew )
+ pr.rates[,ia,1,"All"   ,] <- ci.pred( All.Dead, newdata = dnew )
+ for( ii in dimnames(pr.rates)[[3]][-1] )
+    {
+ dnew = transform( dnew, lex.Cst = factor( 2, levels=1:4,
+                                           labels=levels(Si$lex.Cst) ),
```

```
+                          t.Ins = ifelse( (DMdur-as.numeric(ii)) >= 0,
+                                          DMdur-as.numeric(ii), NA ) )
+ pr.rates[,ia, ii ,"DM/Ins",] <- ci.pred( Ins.Dead, newdata = dnew )
+ pr.rates[,ia, ii ,"All"   ,] <- ci.pred( All.Dead, newdata = dnew )
+     }
+     }
```

So for each age at DM onset we make a plot of the mortality as function of current age both for those with no insulin treatment at those that start 1, 3 and 5 years after, thus 4 curves (with c.i.). These curves are replicated with a different color for the simplified model.

```
> par( mar=c(3,3,1,1), mgp=c(3,1,0)/1.6, las=1 )
> plot( NA, xlim=c(40,82), ylim=c(5,300), bty="n",
+       log="y", xlab="Age", ylab="Mortality rate per 1000 PY" )
> abline( v=seq(40,80,5), h=outer(1:9,10^(0:2),"*"), col=gray(0.9) )
> for( aa in 4:7*10 ) for( ii in 1:4 )
+    matlines( aa+as.numeric(dimnames(pr.rates)[[1]]),
+              cbind( pr.rates[,paste(aa),ii,"DM/Ins",],
+                     pr.rates[,paste(aa),ii,"All"   ,] ),
+              type="l", lty=1, lwd=c(3,1,1),
+              col=rep(c(gray(0.2),"limegreen"),each=3) )
```

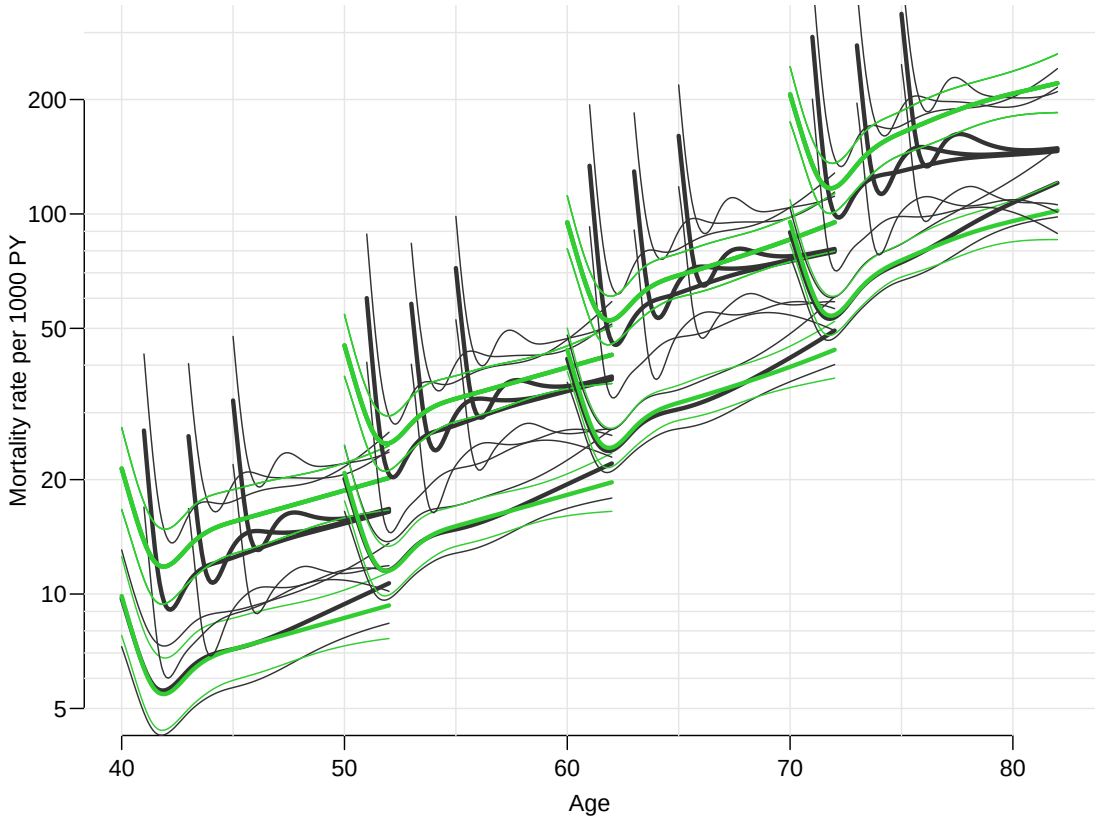

Figure 1.2: *Estimated mortality rates for male diabetes patients with no insulin (lower sets of curves) and insulin (upper curves), with DM onset in age 40, 50, 60 and 70. The black curves are from the models with separate age effects for persons with and without insulin, and a separate effect of insulin duration. The green curves are from the model with common age-effects and only a time-dependent insulin effect (the classical time-dependent variable approach).*

From figure 1.2 we see that there is a substantial insulin-duration effect which is not accommodated by the simple model with only one time-dependent variable to describe the insulin effect. Note that the simple model (green curves) for those on insulin does not depend in insulin duration, and hence the mortality curves for those on insulin are just parallel to the mortality curves for those not on insulin, regardless of diabetes duration (or age) at the time of insulin initiation. Thus the effect of insulin initiation is under estimated for short duration of insulin and overestimated for long duration of insulin.

This is the major discrepancy between the two models, and illustrates the importance of being able to accommodate different time scales, but there is also a declining overall insulin effect by age which is also overlooked by the proportional hazards approach.

1.5 Input to the `simLexis` function

In order to simulate from the multistate model with the estimated transition rates, and get the follow-up of a cohort, we must supply *both* the transition rates and the structure of the model *as well as* the initial cohort status to `simLexis`.

1.5.1 The transition object

We first put the models into an object representing the transitions; note this is a list of lists, the latter having `glm` objects as elements[1]:

```
> Tr <- list( "DM" = list( "Ins"       = DM.Ins,
+                          "Dead"      = DM.Dead  ),
+             "Ins" = list( "Dead(Ins)" = Ins.Dead ) )
```

Now we have the description of the rates and of the structure of the model. The `Tr` object defines the states and all transitions between them; the object `Tr$A$B` contains the model for the transition intensity from state `A` to state `B`.

1.5.2 The initial cohort

We now define an initial `Lexis` object of persons with all relevant covariates defined. Note that we use `subset` to get a pre`Lexis` object, this conserves the `time.scale` and `time.since` attributes which is needed for the simulation (the usual "`[`" operator does not preserve these attributes):

```
> str( Si[NULL,1:9] )
```

```
Classes 'Lexis' and 'data.frame':       0 obs. of  9 variables:
 $ lex.id : int
 $ Per    : num
 $ Age    : num
 $ DMdur  : num
 $ t.Ins  : num
 $ lex.dur: num
 $ lex.Cst: Factor w/ 4 levels "DM","Ins","Dead",..:
 $ lex.Xst: Factor w/ 4 levels "DM","Ins","Dead",..:
 $ sex    : Factor w/ 2 levels "M","F":
```

[1] In future implementations the natural thing would be to have a function that returns cumulative rates given a (pre)`Lexis` object as input

```
> ini <- subset(Si,select=1:9)[NULL,]
> str( ini )


Classes 'Lexis' and 'data.frame':       0 obs. of  9 variables:
 $ lex.id : int
 $ Per    : num
 $ Age    : num
 $ DMdur  : num
 $ t.Ins  : num
 $ lex.dur: num
 $ lex.Cst: Factor w/ 4 levels "DM","Ins","Dead",..:
 $ lex.Xst: Factor w/ 4 levels "DM","Ins","Dead",..:
 $ sex    : Factor w/ 2 levels "M","F":
 - attr(*, "breaks")=List of 4
  ..$ Per  : NULL
  ..$ Age  : NULL
  ..$ DMdur: num  0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 ...
  ..$ t.Ins: NULL
 - attr(*, "time.scales")= chr  "Per" "Age" "DMdur" "t.Ins"
 - attr(*, "time.since")= chr  "" "" "" "Ins"
```

We now have an empty `Lexis` object with attributes reflecting the timescales in multistate model we want to simulate, so we must now enter some data to represent the persons whose follow-up we want to simulate through the model:

```
> ini[1:2,"lex.id"] <- 1:2
> ini[1:2,"lex.Cst"] <- "DM"
> ini[1:2,"Per"] <- 1995
> ini[1:2,"Age"] <- 60
> ini[1:2,"DMdur"] <- 5
> ini[1:2,"sex"] <- c("M","F")
> ini

  lex.id  Per Age DMdur t.Ins lex.dur lex.Cst lex.Xst sex
1      1 1995  60     5    NA      NA      DM    <NA>   M
2      2 1995  60     5    NA      NA      DM    <NA>   F
```

1.6 Simulation of the follow-up

Now we simulate 5000 of each of these persons using the estimated model. The `time.pts` argument gives the times at which the integrated intensities (cumulative rates) are evaluated and between which linear interpolation is used when simulating transition times. Note that this must be given in the same units as `lex.dur` in the `Lexis` object used for fitting the Poisson models for the transitions.

```
> system.time( simL <- simLexis( Tr,
+                                ini,
+                           time.pts = seq(0,20,0.2),
+                                  N = 5000 ) )

   user  system elapsed
  92.72   15.66  108.67


> summary( simL, by="sex" )
```

```
$M

Transitions:
     To
From    DM  Ins Dead Dead(Ins)  Records:  Events: Risk time:  Persons:
  DM   382 2380 2238         0      5000     4618   44206.37      5000
  Ins    0 1059    0      1321      2380     1321   19732.85      2380
  Sum  382 3439 2238      1321      7380     5939   63939.22      5000

$F

Transitions:
     To
From    DM  Ins Dead Dead(Ins)  Records:  Events: Risk time:  Persons:
  DM   863 2219 1918         0      5000     4137   53986.55      5000
  Ins    0 1366    0       853      2219      853   20249.19      2219
  Sum  863 3585 1918       853      7219     4990   74235.74      5000
```

The result is a `Lexis` object — a data frame representing the simulated follow-up of 10,000 persons (5000 identical men and 5000 identical women) according to the rates we estimated from the original dataset.

1.7 Reporting the simulation results

We can now tabulate the number of persons in each state at a predefined set of times on a given time scale. Note that in order for this to be sensible, the `from` argument would normally be equal to the starting time for the simulated individuals.

```
> system.time(
+ nSt <- nState( subset(simL,sex=="M"),
+                at=seq(0,15,0.2), from=1995, time.scale="Per" ) )


   user  system elapsed
   2.06    0.02    2.08


> nSt[1:10,]


        State
when       DM Ins Dead Dead(Ins)
  1995   5000   0    0         0
  1995.2 4940  33   27         0
  1995.4 4874  73   52         1
  1995.6 4802 113   84         1
  1995.8 4725 160  110         5
  1996   4642 208  144         6
  1996.2 4541 269  177        13
  1996.4 4468 311  205        16
  1996.6 4391 362  226        21
  1996.8 4332 390  249        29
```

We see that as time goes by, the 5000 men slowly move away from the starting state (`DM`).

Based on this table (`nSt` is a table) we can now compute the fractions in each state, or, rather more relevant, the cumulative fraction across the states in some specified order, so that a plot of the stacked probabilities can be made:

```
> pp <- pState( nSt, perm=c(1,2,4,3) )
> head( pp )
```

```
when         DM    Ins Dead(Ins) Dead
  1995   1.0000 1.0000    1.0000    1
  1995.2 0.9880 0.9946    0.9946    1
  1995.4 0.9748 0.9894    0.9896    1
  1995.6 0.9604 0.9830    0.9832    1
  1995.8 0.9450 0.9770    0.9780    1
  1996   0.9284 0.9700    0.9712    1
```

```
> par( mfrow=c(1,2), mar=c(3,3,1,1), mgp=c(3,1,0)/1.6 )
> plot( pp )
> plot( pp, border="black", col="transparent", lwd=3 )
> text( rep(as.numeric(rownames(pp)[nrow(pp)-1]),ncol(pp)),
+       pp[nrow(pp),]-diff(c(0,pp[nrow(pp),]))/3,
+       colnames( pp ), adj=1 )
```

A more useful set-up of the graph would include a more through annotation and sensible choice of colors:

```
> clr <- c("limegreen","orange")
> # expand with a lighter version of the two chosen colors
> clx <- c( clr, rgb( t( col2rgb( clr[2:1] ) + rep(255,3) ) / 2, max=255 ) )
> par( mfrow=c(1,2), las=1, mar=c(3,3,4,2), mgp=c(3,1,0)/1.6 )
> # Men
> plot( pp, col=clx )
> lines( as.numeric(rownames(pp)), pp[,2], lwd=3 )
> mtext( "60 year old male, diagnosed 1990, aged 55", side=3, line=2.5, adj=0, col=gray(0.6) )
> mtext( "Survival curve", side=3, line=1.5, adj=0 )
> mtext( "DM, no insulin   DM, Insulin", side=3, line=0.5, adj=0, col=clr[1] )
```

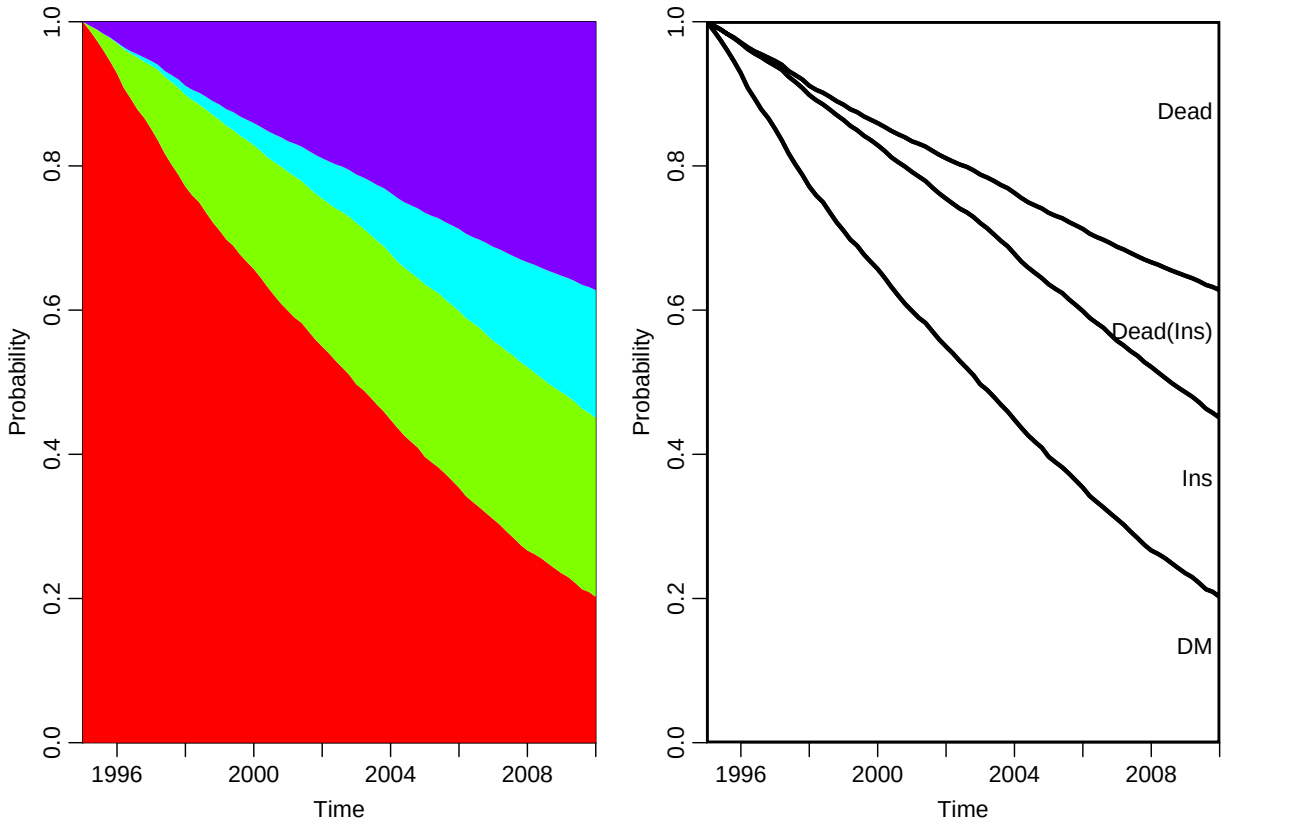

Figure 1.3: *Default layout of the* `plot.pState` *graph (left), and a version with the state probabilites as lines and annotation of states.*

```
> mtext( "DM, no insulin", side=3, line=0.5, adj=0, col=clr[2] )
> axis( side=4 )
> # Women
> pf <- pState( nState( subset(simL,sex=="F"),
+                       at=seq(0,15,0.2),
+                       from=1995,
+                       time.scale="Per" ),
+               perm=c(1,2,4,3) )
> plot( pf, col=clx )
> lines( as.numeric(rownames(pf)), pf[,2], lwd=3 )
> mtext( "60 year old female, diagnosed 1990, aged 55", side=3, line=2.5, adj=0, col=gray(0.6) )
> mtext( "Survival curve", side=3, line=1.5, adj=0 )
> mtext( "DM, no insulin   DM, Insulin", side=3, line=0.5, adj=0, col=clr[1] )
> mtext( "DM, no insulin", side=3, line=0.5, adj=0, col=clr[2] )
> axis( side=4 )
```

If we instead wanted to show the results on the age-scale, we would just as for that when constructing the probabilities; otherwise the code is pretty much the same as before:

```
> par( mfrow=c(1,2), las=1, mar=c(3,3,4,2), mgp=c(3,1,0)/1.6 )
> # Men
> pm <- pState( nState( subset(simL,sex=="M"),
+                       at=seq(0,15,0.2),
+                       from=60,
+                       time.scale="Age" ),
+               perm=c(1,2,4,3) )
> plot( pm, col=clx, xlab="Age" )
> lines( as.numeric(rownames(pm)), pm[,2], lwd=3 )
```

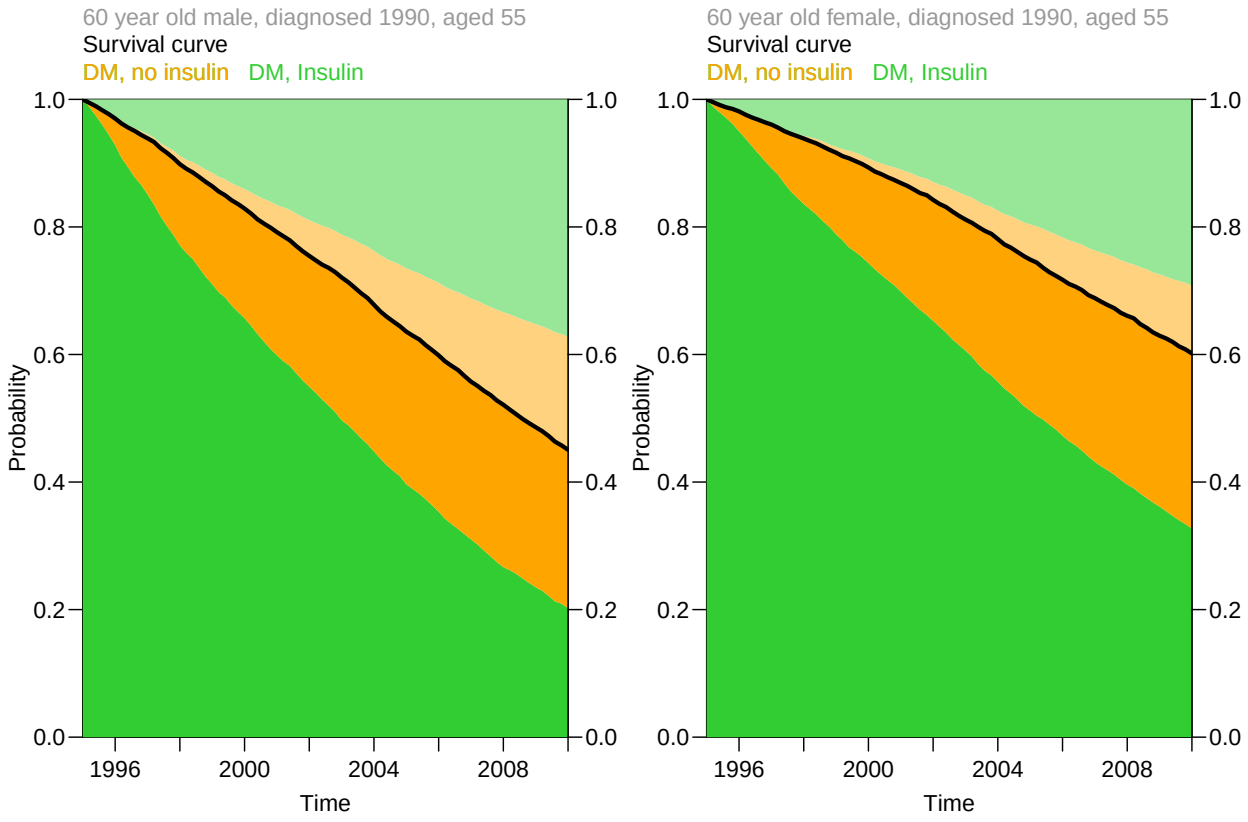

Figure 1.4: `plot.pState` *graphs where persons ever on insulin are given in orange and persons never on insulin in green, and the overall survival (dead over the line) as a black line.*

```
> mtext( "60 year old male, diagnosed 1990, aged 55", side=3, line=2.5, adj=0, col=gray(0.6) )
> mtext( "Survival curve", side=3, line=1.5, adj=0 )
> mtext( "DM, no insulin   DM, Insulin", side=3, line=0.5, adj=0, col=clr[1] )
> mtext( "DM, no insulin", side=3, line=0.5, adj=0, col=clr[2] )
> axis( side=4 )
> # Women
> pf <- pState( nState( subset(simL,sex=="F"),
+                       at=seq(0,15,0.2),
+                       from=60,
+                       time.scale="Age" ),
+               perm=c(1,2,4,3) )
> plot( pf, col=clx, xlab="Age" )
> lines( as.numeric(rownames(pf)), pf[,2], lwd=3 )
> mtext( "60 year old female, diagnosed 1990, aged 55", side=3, line=2.5, adj=0, col=gray(0.6) )
> mtext( "Survival curve", side=3, line=1.5, adj=0 )
> mtext( "DM, no insulin   DM, Insulin", side=3, line=0.5, adj=0, col=clr[1] )
> mtext( "DM, no insulin", side=3, line=0.5, adj=0, col=clr[2] )
> axis( side=4 )
```

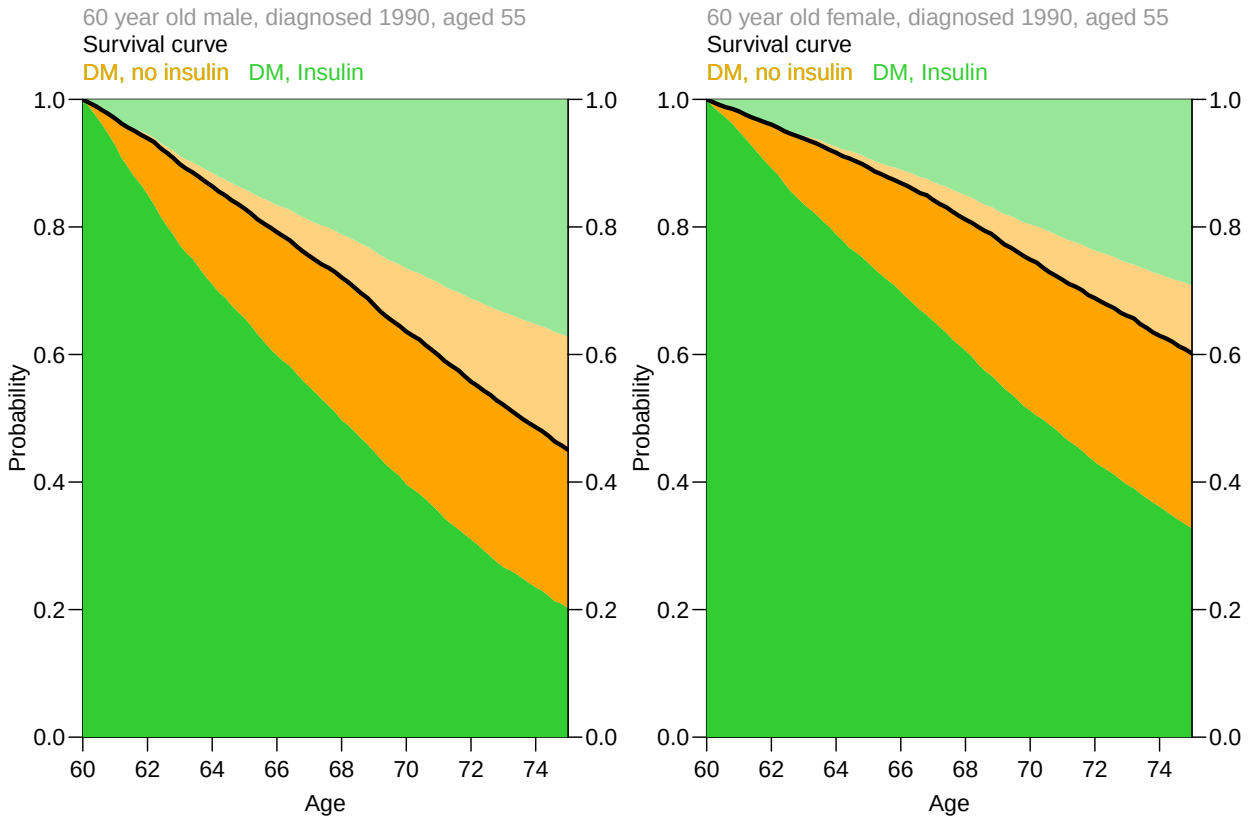

Figure 1.5: `plot.pState` *graphs where persons ever on insulin are given in orange and persons never on insulin in green, and the overall survival (dead over the line) as a black line.*

Chapter 2

How `simLexis` is coded

This section explains in more detail how the simulation machinery is implemented and how it exploits the structure of the `Lexis` objects.

For the sake of the argument we use the example with the `DMlate` described above, and use the models for the transition rates fitted to the time-split data set.

We now put these three `glm`s in a `Tr` list of list of glm objects, designed to represent the possible transitions in the multistate model. The list has names equal to the states *from* which transitions occur (the transient states), and the sublists have names equal to the states *to* which the transitions occur.

```
> Tr <- list( "DM" = list( "Ins"       = DM.Ins,
+                          "Dead"      = DM.Dead  ),
+             "Ins" = list( "Dead(Ins)" = Ins.Dead ) )
> lapply( Tr, names )
```

```
$DM
[1] "Ins"  "Dead"

$Ins
[1] "Dead(Ins)"
```

If we want to simulate transitions according to this model for a (group of) persons, we must define all relevant covariates, among which are the time scales, `lex.Cst`, whereas `lex.dur` and `lex.Xst` will be the target of the simulation. It will be a Lexis object because we want to keep track of the timescales — this is the major point that makes it possible to simulate from processes where the rates depend on multiple time scales.

For a start we could make a data frame with only 1 person in it; but we set up `N` identical persons, because we ultimately will be simulating transitions and -times for a data frame of different persons:

```
> N <- 2
> ini <- subset(Si,select=1:9)[NULL,]
> ini[1:N,"lex.id"] <- 1:N
> ini[1:N,"lex.dur"] <- NA
> ini[1:N,"lex.Cst"] <- "DM"
> ini[1:N,"lex.Xst"] <- NA
> ini[1:N,"Per"] <- 2000
> ini[1:N,"Age"] <- 50
> ini[1:N,"DMdur"] <- 1
> ini[1:N,"sex"] <- c("M","F")
> attr( ini, "time.since" ) <- c("","","","Ins")
> ini
```

```
  lex.id  Per Age DMdur t.Ins lex.dur lex.Cst lex.Xst sex
1      1 2000  50     1    NA      NA      DM    <NA>   M
2      2 2000  50     1    NA      NA      DM    <NA>   F
```

The core of the simulation is to simulate transition times in the estimated model, and to this end we must predict the cumulative incidence for each person: We use the value of `lex.Cst` to select the relevant element of `Tr`, and based on this, compute the prediction of the cumulative intensity in `np` points, starting at 0, across `ni` intervals, at an equidistance of `int`. Note that `int` are assumed given in the same units as those in which the person-risk time were supplied to the offset when fitting the `glm`s to the original `Lexis` object.

```
> tmax<- 20          # How long into the future should we predict
> ni  <- 100         # Number of intervals to use
> int <- tmax/ni     # The lenght of the intervals
> pt  <- 0:ni*int    # The prediction points
> np  <- length(pt)
```

We will need the intensities calculated at these time points, but for the calculation of the cumulative rates we need the cumulative sum using the average rates in each of the intervals, which we choose to define as the mean of the intensities at the two endpoints, so we define a function to compute cumulative rates this way; the argument `x` represents the estimated rates at the points `pt`:

```
> cummid <- function( x, pt=1:length(x) ) cumsum( c(0, (x[-1]-diff(x)/2)*diff(pt) ) )
```

What we will do is to use the `Tr` object to make predictions of the cumulative rates in an array classified by transition, time for FU and person, the only assumption being assuming that all persons are in the same current state.

So we set up a prediction data frame which basically is the `Lexis` object of the starters with each row repeated `np` times, but where the timescales are updated by adding `pt`.

```
> nd <- ini[rep(1:nrow(ini),each=np),]
> nd[,timeScales(ini)] <- nd[,timeScales(ini)] + rep(pt,np)
> head( cbind( pt, nd ), 20 )
```

```
      pt lex.id    Per  Age DMdur t.Ins lex.dur lex.Cst lex.Xst sex
1    0.0      1 2000.0 50.0   1.0    NA      NA      DM    <NA>   M
1.1  0.2      1 2000.2 50.2   1.2    NA      NA      DM    <NA>   M
1.2  0.4      1 2000.4 50.4   1.4    NA      NA      DM    <NA>   M
1.3  0.6      1 2000.6 50.6   1.6    NA      NA      DM    <NA>   M
1.4  0.8      1 2000.8 50.8   1.8    NA      NA      DM    <NA>   M
1.5  1.0      1 2001.0 51.0   2.0    NA      NA      DM    <NA>   M
1.6  1.2      1 2001.2 51.2   2.2    NA      NA      DM    <NA>   M
1.7  1.4      1 2001.4 51.4   2.4    NA      NA      DM    <NA>   M
1.8  1.6      1 2001.6 51.6   2.6    NA      NA      DM    <NA>   M
1.9  1.8      1 2001.8 51.8   2.8    NA      NA      DM    <NA>   M
1.10 2.0      1 2002.0 52.0   3.0    NA      NA      DM    <NA>   M
1.11 2.2      1 2002.2 52.2   3.2    NA      NA      DM    <NA>   M
1.12 2.4      1 2002.4 52.4   3.4    NA      NA      DM    <NA>   M
1.13 2.6      1 2002.6 52.6   3.6    NA      NA      DM    <NA>   M
1.14 2.8      1 2002.8 52.8   3.8    NA      NA      DM    <NA>   M
1.15 3.0      1 2003.0 53.0   4.0    NA      NA      DM    <NA>   M
1.16 3.2      1 2003.2 53.2   4.2    NA      NA      DM    <NA>   M
1.17 3.4      1 2003.4 53.4   4.4    NA      NA      DM    <NA>   M
1.18 3.6      1 2003.6 53.6   4.6    NA      NA      DM    <NA>   M
1.19 3.8      1 2003.8 53.8   4.8    NA      NA      DM    <NA>   M
```

But if we want to predict using `nd` as the `newdata=` argument we should insert a value for `lex.dur` that will make the predictions from the `glm`s into actual (log) rates. If that is going to work we need an assumption that the units in which time points are given, are the same as the units in which the risk time was given to the `glm` as offset:

```
> nd[,"lex.dur"] <- 1
> # This is where we assume the state for all persons is the same:
> inc <- data.frame( lex.id=nd$lex.id,
+                    exp( sapply( Tr[[nd[1,"lex.Cst"]]],
+                                 predict.glm,
+                                 newdata=nd ) ) )
> head( inc )
```

```
    lex.id        Ins       Dead
1        1 0.05228017 0.01182956
1.1      1 0.04418061 0.01113970
1.2      1 0.03743490 0.01065319
1.3      1 0.03188624 0.01037278
1.4      1 0.02737428 0.01026745
1.5      1 0.02374795 0.01030256
```

Now `inc` contains the estimated rates (out of the current box) at specific time points of follow-up, so we need to derive the cumulative incidences within each person, and from that derive a transition time and a transition for each person.

2.1 Simulation of transition times

2.1.1 The theory

Suppose that the rate functions for the transitions out of the current state to , say, 3 different states are λ_1, λ_2 and λ_3, and the corresponding cumulative rates are Λ_1, Λ_2 and Λ_3, and we want to simulate an exit time and an exit state (that is either 1, 2 or 3). This can be done in two slightly different ways:

1. First time, then state:
   (a) Compute the survival function, $S(t) = \exp\big(-\Lambda_1(t) - \Lambda_2(t) - \Lambda_3(t)\big)$
   (b) Simulate a random U(0,1) variate, u, say.
   (c) The simulated exit time is then the solution t_u to the equation $S(t_u) = u \quad \Leftrightarrow \quad \sum_j \Lambda_j(t_u) = -\log(u)$.
   (d) A simulated transition at t_u is then found by simulating from the multinomial distribution with probabilities $p_i = \lambda_i(t_u) / \sum_j \lambda_j(t_u)$.
2. Separate cumulative incidences
   (a) Simulate 3 independent U(0,1) random variates u_1, u_2 and u_3.
   (b) Solve the equations $\Lambda_i(t_i) = -\log(u_i), i = 1, 2, 3$ and get (t_1, t_2, t_3).
   (c) The simulated survival time is then $\min(t_1, t_2, t_3)$, and the simulated transition is to the state corresponding to this, that is $k \in \{1, 2, 3\}$, where $t_k = \min(t_1, t_2, t_3)$

The intuitive argument is that with three possible transition there are 3 independent processes running, but only the first transition is observed. The latter approach is used in this implementation.

The formal argument for the equality of the two approaches goes as follows:

1. Equality of the transition times:

   (a) In the first approach we simulate from a distribution with cumulative rate $\Lambda_1(t) + \Lambda_2(t) + \Lambda_3(t)$, hence from a distribution with survival function:

   $$\begin{aligned} S(t) &= \exp\bigl(-(\Lambda_1(t) + \Lambda_2(t) + \Lambda_3(t))\bigr) \\ &= \exp\bigl(-\Lambda_1(t)\bigr) \times \exp\bigl(-\Lambda_2(t)\bigr) \times \exp\bigl(-\Lambda_3(t)\bigr) \end{aligned}$$

   (b) In the second approach we choose the smallest of three survival times, each with survival function $\exp(-\Lambda_i), i = 1, 2, 3$. Now, the survival function for the minimum of three survival times is:

   $$\begin{aligned} S_{\min}(t) &= \mathrm{P}\left\{\min(t_1, t_2, t_3) > t\right\} \\ &= \mathrm{P}\left\{t_1 > t\right\} \times \mathrm{P}\left\{t_2 > t\right\} \times \mathrm{P}\left\{t_3 > t\right\} \\ &= \exp\bigl(-\Lambda_1(t)\bigr) \times \exp\bigl(-\Lambda_2(t)\bigr) \times \exp\bigl(-\Lambda_3(t)\bigr) \end{aligned}$$

   which is the same survival function as derived above.

2. Type of transition:

   (a) In the first instance the conditional distribution of the state to which the transition is, conditional on the transition time being t, is taken as: $\lambda_i(t)/(\lambda_1(t) + \lambda_2(t) + \lambda_3(t))$.

   (b) In the second approach we choose the transition corresponding to the the smallest of the transition times. So when we condition on the event that a transition takes place at time t, we have to shown that the conditional probability that the smallest of the three simulated transition times was actually the ith, is as above.

   But the probabilities conditional on *survival* till t, that events of type $1, 2, 3$ takes place in the interval $(t, t + \mathrm{d}t)$ are $\lambda_1(t)\,\mathrm{d}t$, $\lambda_2(t)\,\mathrm{d}t$ and $\lambda_1(t)\,\mathrm{d}t$, respectively, hence the conditional probabilities *given a transition time* in $(t, t + \mathrm{d}t)$ is exactly a above:

   $$\frac{\lambda_i(t)\,\mathrm{d}t}{\lambda_1(t)\,\mathrm{d}t + \lambda_2(t)\,\mathrm{d}t + \lambda_3(t)\,\mathrm{d}t} = \frac{\lambda_i(t)}{\lambda_1(t) + \lambda_2(t) + \lambda_3(t)}$$

2.1.2 Implementation

If we only look at the contribution from a single person to the data frame with the predicted intensities from time of entry and the designated time into the future, we can simulate a survival time by taking a uniform random variate (u, say) and pretend that it is the survival function, or equivalently that $-\log(u)$ is the cumulative rate. We then find the corresponding times for each of the transitions by linear interpolation (using `approx`), and finally choose the smallest transition time:

```
> dd <- subset( inc, lex.id==1 )
> head( dd, 20 )
```

```
     lex.id        Ins       Dead
1         1 0.05228017 0.01182956
1.1       1 0.04418061 0.01113970
1.2       1 0.03743490 0.01065319
1.3       1 0.03188624 0.01037278
1.4       1 0.02737428 0.01026745
1.5       1 0.02374795 0.01030256
1.6       1 0.02087299 0.01044975
1.7       1 0.01863587 0.01068337
1.8       1 0.01694541 0.01097779
1.9       1 0.01573344 0.01130548
1.10      1 0.01495530 0.01163573
1.11      1 0.01459142 0.01193575
1.12      1 0.01465085 0.01219214
1.13      1 0.01517573 0.01240773
1.14      1 0.01619434 0.01258660
1.15      1 0.01771144 0.01273353
1.16      1 0.01974696 0.01285392
1.17      1 0.02232446 0.01295357
1.18      1 0.02545510 0.01303863
1.19      1 0.02911790 0.01311548
```

```
> ci <- apply( dd[,-1,drop=FALSE], 2, cummid, pt )
> tt <- uu <- -log( runif(ncol(ci)) )
> for( i in 1:ncol(ci) ) tt[i] <- approx(ci[,i],pt,uu[i])$y
> tt
```

```
[1]       NA 15.88534
```

```
> list( min(tt,na.rm=TRUE), colnames(ci)[tt==min(tt,na.rm=TRUE)] )
```

```
[[1]]
[1] 15.88534

[[2]]
[1] NA     "Dead"
```

This is then packed into a function that takes a data frame with predicted incidence rates along `pt` as input and delivers the time and transition as output. However, we really want everyone to have a simulated transition time or censoring time. Basically we only simulate transition times up to the maximal value of `pt`, if we simulate a value that is beyond this we, set the follow-up time to `max(pt)` and treat it as a censoring.

```
> sim1 <-
+ function( dd, pt )
+ {
+ ci <- apply( dd[,-1,drop=FALSE], 2, cummid, pt )
+ tt <- uu <- -log( runif(ncol(ci)) )
+ for( i in 1:ncol(ci) ) tt[i] <- approx(ci[,i],pt,uu[i],rule=2)$y
+ data.frame( lex.id  = dd[1,1],
+              lex.dur = min(tt,na.rm=TRUE),
+              lex.Xst = factor( if( min(tt)<max(pt) ) colnames(ci)[tt==min(tt)]
+                                else NA, levels=levels(ini$lex.Cst) ) )
+ }
```

To make this work sequentially for all persons, we pack it in a `do.call`:

```
> ( rr <- do.call( "rbind", lapply( split(inc,inc$lex.id), sim1, pt ) ) )
```

```
  lex.id  lex.dur lex.Xst
1      1 16.98680    Dead
2      2 10.52879     Ins
```

The result of this is then used to update the initial data frame:

```
> xx <- match( c("lex.dur","lex.Xst"), names(ini) )
> ini.upd <- merge( ini[,-xx], rr )
> attr( ini.upd, "time.scales" ) <- attr( ini, "time.scales" )
> str( ini.upd )
```

```
Classes 'Lexis' and 'data.frame':        2 obs. of  9 variables:
 $ lex.id : int  1 2
 $ Per    : num  2000 2000
 $ Age    : num  50 50
 $ DMdur  : num  1 1
 $ t.Ins  : num  NA NA
 $ lex.Cst: Factor w/ 4 levels "DM","Ins","Dead",..: 1 1
 $ sex    : Factor w/ 2 levels "M","F": 1 2
 $ lex.dur: num  17 10.5
 $ lex.Xst: Factor w/ 4 levels "DM","Ins","Dead",..: 3 2
 - attr(*, "time.scales")= chr  "Per" "Age" "DMdur" "t.Ins"
```

```
> ini.upd
```

```
  lex.id  Per Age DMdur t.Ins lex.Cst sex  lex.dur lex.Xst
1      1 2000  50     1    NA      DM   M 16.98680    Dead
2      2 2000  50     1    NA      DM   F 10.52879     Ins
```

Then we can split this data frame, into those who exited to a transient state, who subsequently must have simulated a further transition, and those who exited to an absorbing state and who so to speak have reached their final destination.

We can derive the transient states from the `Tr` object; it is simply the names of the `Tr` object:

```
> ( tr.states <- names( Tr ) )
```

```
[1] "DM"  "Ins"
```

The finalized rows (*i.e.* persons) of the object are those who exited to an absorbing state (non-transient):

```
> ini.fin <- subset( ini.upd, !lex.Xst %in% tr.states )
> ini.fin[is.na(ini.fin$lex.Xst),"lex.Xst"] <- ini.fin[is.na(ini.fin$lex.Xst),"lex.Cst"]
> ini.fin
```

```
  lex.id  Per Age DMdur t.Ins lex.Cst sex lex.dur lex.Xst
1      1 2000  50     1    NA      DM   M 16.9868    Dead
```

Rows requiring another simulation are those who have been simulated to exit to a transient state:

```
> ( ini.nxt <- subset( ini.upd,  lex.Xst %in% tr.states ) )
```

```
  lex.id  Per Age DMdur t.Ins lex.Cst sex  lex.dur lex.Xst
2      2 2000  50     1    NA      DM   F 10.52879     Ins
```

The input is then the previous rows with the timescales updated by `lex.dur`, the time since entry to the current state updated, and the current state, `lex.Cst` set to the state to which the exit was:

```
> attr( ini.nxt, "time.since" ) <- attr( ini, "time.since" )
> ini.nxt[,timeScales(ini.nxt)] <- ini.nxt[,timeScales(ini.nxt)] + ini.nxt$lex.dur
> # Initialize the timescale starting at the state where we just entered
> for( i in 1:length(wh<-attr(ini.nxt,"time.since")) )
+    if( wh[i] != "" & sum(ini.nxt$lex.Xst==wh[i])>0 )
+        ini.nxt[ini.nxt$lex.Xst==wh[i],timeScales(ini.nxt)[i]] <- 0
> ini.nxt$lex.Cst <- ini.nxt$lex.Xst
> if( nrow(ini.nxt)>0 ) ini.nxt$lex.dur <- 1
> ini.nxt
```

```
  lex.id      Per      Age    DMdur t.Ins lex.Cst sex lex.dur lex.Xst
2      2 2010.529 60.52879 11.52879     0     Ins   F       1     Ins
```

Note here that we have updated the value of `t.Ins` to 0, as this record represents a person that has just started insulin. This record is then input to a new simulation with the possibility of using the `t.Ins` timescale in the definition of the mortality rates.

2.2 Components of `simLexis`

This section explains the actually existing code for `simLexis`, as it is in the current version of `Epi`.

```
> print( sessionInfo(), L=F )
```

```
R version 3.0.1 (2013-05-16)
Platform: i386-w64-mingw32/i386 (32-bit)

locale:
[1] LC_COLLATE=Danish_Denmark.1252  LC_CTYPE=Danish_Denmark.1252
[3] LC_MONETARY=Danish_Denmark.1252 LC_NUMERIC=C
[5] LC_TIME=Danish_Denmark.1252

attached base packages:
[1] splines   utils     datasets  graphics  grDevices stats     methods
[8] base

other attached packages:
[1] Epi_1.1.56     foreign_0.8-53

loaded via a namespace (and not attached):
[1] tools_3.0.1
```

The function `simLexis` takes a (pre-)`Lexis` object as input. This defines the initial state(s) and times of the start for a number of persons. Since the purpose is to simulate a history through the estimated multistate model, the input values of the outcome variables `lex.Xst` and `lex.dur` are ignored — the aim is to simulate values for them.

Note however that the attribute `time.since` must be present in the object. This is used for initializing timescales defined as time since entry into a particular state, it is a character vector of the same length as the `time.scales` attribute, with value equal to a state name if the corresponding time scale is defined as time since entry into that state. In this example the 4th timescale is time since entry into the "Ins" state, and hence:

```
> cbind(
+ attr( ini, "time.scale" ),
+ attr( ini, "time.since" ) )
```

```
     [,1]    [,2]
[1,] "Per"   ""
[2,] "Age"   ""
[3,] "DMdur" ""
[4,] "t.Ins" "Ins"
```

`Lexis` objects will have this attribute set for time scales created using `cutLexis`.

The other necessary argument is a transition object `Tr`, which is a list of lists. The elements of the lists should be `glm` objects derived by fitting Poisson models to a `Lexis` object representing follow-up data in a multistate model. It is assumed (but not checked) that timescales enter in the model via the timescales of the `Lexis` object. Formally, there are no assumptions about how `lex.dur` enters in the model.

The two optional arguments are `time.pts`, a numerical vector giving the times after entry at which the cumulative rates will be computed (the maximum of which will be taken as the censoring time), and `N` a scalar or numerical vector of the number of persons with a given initial state each record of the `init` object should represent.

The central part of the functions uses a `do.call` / `lapply` / `split` construction to do simulations for different initial states. This is the construction in the middle that calls `simX`. `simLexis` also calls `get.next` which is also detailed below.

```
> simLexis
```

```
function( Tr, # List of lists of glm objects
        init, # Lexis objects of persons to simulate. Must have the
              # same attributes as the original object, in particular
              # "time.scales" and "time.since".
    time.pts = 0:50/2, # Points where rates are computed in the
                       # simulation
           N = 1, # How many persons should each line in
                  # init represent?
      lex.id = 1:(N*nrow(init)), # What should be the ids of the simulated persons?
        type = "glm-mult" # Not used currently
         )
{
# Expand the input data frame using N and put in lex.id
foo <- lex.id
init <- init[rep(1:nrow(init),each=N),]
init$lex.id <- foo

# Fix attributes
if( is.null( nts <- attr(init,"time.scales") ) )
  stop( "No time.scales attribute for init" )
if( is.null( attr(init,"time.since") ) )
  {
  attr(init,"time.since") <- rep( "", nts )
  cat( "WARNING:\n
       'time.since' attribute set, which means that you assume that\n
```

```
        none of the time scale represent time entry to a state." )
  }
# Convenience constants
np <- length( time.pts )
tr.st <- names( Tr )

# The first set of sojourn times in the initial states
sf <- do.call( "rbind",
               lapply( split( init,
                              init$lex.Cst ),
                       simX,
                       init, Tr, time.pts ) )

# Then we must update those who have ended in transient states
# and keep on doing that till all are in absorbing states or censored
nxt <- get.next( sf, init, tr.st )
while( nrow(nxt) > 0 )
{
nx <- do.call( "rbind",
               lapply( split( nxt,
                              nxt$lex.Cst ),
                       simX,
                       init, Tr, time.pts ) )
sf <- rbind( sf, nx )
nxt <- get.next( nx, init, tr.st )
}

# Doctor lex.Xst for the censored, and supply attributes
sf$lex.Xst[is.na(sf$lex.Xst)] <- sf$lex.Cst[is.na(sf$lex.Xst)]

# Finally, nicely order the output by persons, then times and states
nord <- match( c( "lex.id", timeScales(sf),
                  "lex.dur",
                  "lex.Cst",
                  "lex.Xst" ), names(sf) )
noth <- setdiff( 1:ncol(sf), nord )
sf <- sf[order(sf$lex.id,sf[,timeScales(init)[1]]),c(nord,noth)]
rownames(sf) <- NULL
attr( sf, "time.scales" ) <- attr( init, "time.scales" )
attr( sf, "time.since"  ) <- attr( init, "time.since" )
chop.lex( sf, max(time.pts) )
}
<environment: namespace:Epi>
```

2.2.1 simX

simX simulates transition-times and -types for a set of patients assumed to be in the same state. It is called from **simLexis** with a data frame as argument, uses the state in **lex.Cst** to select the relevant component of **Tr** and compute predicted cumulative intensities for all states reachable from this state. The dataset on which this is done has **length(time.pts)** rows per person:

```
> Epi:::simX
```

```
function( nd, init, Tr, time.pts )
{
# Simulation is done from the data frame nd, in chunks of starting
# state, lex.Cst. This is necessary because different states have
# different (sets of) exit rates. Therefore, this function simulates
# for a set of persons from the same starting state.
np <- length( time.pts )
```

```
nr <- nrow( nd )
if( nr==0 ) return( NULL )

# The 'as.character' below is necessary because indexing by a factor
# by default is by the number of the level, and we are not indexing by
# this, but by components of Tr which just happens to have names that
# are a subset of the levels of lex.Cst.
cst <- as.character( unique(nd$lex.Cst) )
if( length(cst)>1 ) stop( "More than one lex.Cst present:\n", cst, "\n" )

# Expand each person by the time points
nx <- nd[rep(1:nrow(nd),each=np),]
nx[,timeScales(init)] <- nx[,timeScales(init)] + rep(time.pts,nr)
nx$lex.dur <- 1

# Make a data frame with predicted rates for each of the transitions
# out of this state for these times
rt <- data.frame( lex.id=nx$lex.id )
for( i in 1:length(Tr[[cst]]) ) rt <- cbind( rt, exp(predict(Tr[[cst]][[i]],newdata=nx)) )
names( rt )[-1] <- names( Tr[[cst]] )

# Then find the transition time and exit state for each person:
xx <- match( c("lex.dur","lex.Xst"), names(nd) )
if( any( !is.na(xx) ) ) nd <- nd[,-xx[!is.na(xx)]]
merge( nd,
       do.call( "rbind",
                lapply( split( rt,
                               rt$lex.id ),
                        sim1,
                        init,
                        time.pts ) ),
       by="lex.id" )
}
<environment: namespace:Epi>
```

Thus `simX` calls `sim1` which simulates the transition for one person.

2.2.2 `sim1`

This is fed, person by person, to `sim1` — again via a `do.call` / `lapply` / `split` construction — and the resulting time and state is appended to the `init` object. This way we have simulated *one* transition for each person:

```
> Epi:::sim1

function( rt, init, time.pts )
{
# Simulates a single transition time and state based on the data frame
# rt with columns lex.id and timescales. The rows in rt are the id,
# followed by the set of estimated transition rates to the different
# states reachable from the current one.
ci <- apply( rt[,-1,drop=FALSE], 2, cummid, time.pts )
tt <- uu <- -log( runif(ncol(ci)) )
for( i in 1:ncol(ci) ) tt[i] <- approx( ci[,i], time.pts, uu[i], rule=2 )$y
# Note this resulting data frame has 1 row
data.frame( lex.id  = rt[1,1],
            lex.dur = min(tt,na.rm=TRUE),
            lex.Xst = factor( if( min(tt)<max(time.pts) )
                                colnames(ci)[tt==min(tt)]
                              else
                                NA,
                              levels=levels(init$lex.Cst) ) )
}
<environment: namespace:Epi>
```

2.2.3 `get.next`

We must repeat this operation on those that have a simulated entry to a transient state, and also make sure that any time scales defined as time since entry to one of these states be initialized to 0 before a call to `simX` is made for these persons. This accomplished by the function `get.next`:

```
> Epi:::get.next
```

```
function( sf, init, tr.st )
{
# Produces an initial Lexis object for the next simulation for those
# who have ended up in a transient state.
# Note that this exploits the existence of the "time.since" attribute
# for Lexis objects and assumes that a character vector naming the
# transient states is supplied as argument.
if( nrow(sf)==0 ) return( sf )
nxt <- sf[sf$lex.Xst %in% tr.st,]
if( nrow(nxt) == 0 ) return( nxt )
nxt[,timeScales(init)] <- nxt[,timeScales(init)] + nxt$lex.dur
wh <- attr( init,"time.since" )
for( i in 1:length(wh) )
   if( wh[i] != "" ) nxt[nxt$lex.Xst==wh[i],timeScales(init)[i]] <- 0
nxt$lex.Cst <- nxt$lex.Xst
return( nxt )
}
<environment: namespace:Epi>
```

2.2.4 `cummid`

Note that the `cummid` function is used by `sim1` to compute the cumulative intensities given the intensities at the time points.

```
> Epi:::cummid
```

```
function( x, time.pts=1:length(x) )
{
# Computes the cumulative area under a curve with values x at time.pts
cumsum( c(0, (x[-1]-diff(x)/2)*diff(time.pts) ) )
}
<environment: namespace:Epi>
```

2.2.5 `chop.lex`

The operation so far has censored individuals `max(time.pts)` after *each* new entry to a transient state. In order to groom the output data we use `chop.lex` to censor all persons at the same designated time after initial entry.

```
> Epi:::chop.lex
```

```
function( obj, cens )
{
# A function that chops off all follow-up beyond cens since entry for
# each individual
zz <- entry( obj, 1, by.id=TRUE )
```

```
ww <- merge( obj, data.frame( lex.id=as.numeric(names(zz)),
                                  cens=zz+cens ) )
ww <- ww[ww[,timeScales(obj)[1]] < ww$cens,]
x.dur <- pmin( ww$lex.dur, ww[,"cens"]-ww[,timeScales(obj)[1]] )
ww$lex.Xst[x.dur<ww$lex.dur] <- ww$lex.Cst[x.dur<ww$lex.dur]
ww$lex.dur <- pmin( x.dur, ww$lex.dur )
ww
}
<environment: namespace:Epi>
```

2.3 Probabilities from simulated `Lexis` objects

Once we have simulated a Lexis object we will of course want to use it for estimating probabilities, so basically we will want to enumerate the number of persons in each state at a pre-specified set of time points.

2.3.1 `nState`

Since we are dealing with multistate model with potentially multiple time scales, it is necessary to define the timescale (`time.scale`), the starting point on this timescale (`from`) and the points after this where we compute the number of occupants in each state, (`at`).

```
> nState
```

```
function ( obj,
            at,
          from,
     time.scale = 1 )
{
# Counts the number of persons in each state of the Lexis object 'obj'
# at the times 'at' from the time 'from' in the time scale
# 'time.scale'

# Determine timescales and absorbing and transient states
tmsc <- Epi:::check.time.scale(obj,time.scale)
TT <- tmat(obj)
absorb <- rownames(TT)[apply(!is.na(TT),1,sum)==0]
transient <- setdiff( rownames(TT), absorb )

# Expand each record length(at) times
tab.frm <-
    obj[rep(1:nrow(obj),each=length(at)),c(tmsc,"lex.dur","lex.Cst","lex.Xst")]

# Stick in the corresponding times on the chosen time scale
tab.frm$when <- rep( at, nrow(obj) ) + from

# For transient states keep records that includes these points in time
tab.tr <- tab.frm[tab.frm[,tmsc]                  <= tab.frm$when &
                  tab.frm[,tmsc]+tab.frm$lex.dur >  tab.frm$when,]
tab.tr$State <- tab.tr$lex.Cst

# For absorbing states keep records where follow-up ended before
tab.ab <- tab.frm[tab.frm[,tmsc]+tab.frm$lex.dur <= tab.frm$when &
                  tab.frm$lex.Xst %in% absorb,]
tab.ab$State <- tab.ab$lex.Xst

# Make a table using the combination of those in transient and
# absorbing states.
```

```
with( rbind( tab.ab, tab.tr ), table( when, State ) )
}
<environment: namespace:Epi>
```

2.3.2 pState & plot.pState

In order to plot probabilities of state-occupancy it is useful to compute cumulative probabilities across states in any given order; this is done by the function **pState**:

```
> pState
```

```
function( nSt, perm=1:ncol(nSt) )
{
# Compute cumulative proportions of persons across states in order
# designate by 'perm'
tt <- t( apply( nSt[,perm], 1, cumsum ) )
tt <- sweep( tt, 1, tt[,ncol(tt)], "/" )
class( tt ) <- c("pState","matrix")
tt
}
<environment: namespace:Epi>
```

There is also a plot method for resulting objects:

```
> plot.pState
```

```
function( x,
        col = rainbow(ncol(x)),
     border = "transparent",
       xlab = "Time",
       ylab = "Probability", ... )
{
# Function to plot cumulative probabilities along the time scale.

# Fixing the colors:
nc <- ncol(x)
col    <- rep( col   , nc )[1:nc]
border <- rep( border, nc )[1:nc]

# Just for coding convenience when plotting polygons
pSt <- cbind( 0, x )
matplot( as.numeric(rownames(pSt)), pSt, type="n",
         ylim=c(0,1), yaxs="i", xaxs="i",
         xlab=xlab, ylab=ylab, ... )
for( i in 2:ncol(pSt) )
   {
   polygon( c(    as.numeric(rownames(pSt)) ,
              rev(as.numeric(rownames(pSt))) ),
            c(    pSt[,i  ],
              rev(pSt[,i-1]) ),
            col=col[i-1], border=border[i-1], ... )
   }
}
<environment: namespace:Epi>
```

Bibliography

[1] Bendix Carstensen and Martyn Plummer. Using Lexis objects for multi-state models in R. *Journal of Statistical Software*, 38(6):1–18, 1 2011.

[2] S Iacobelli and B Carstensen. Multistate models with multiple timescales. *Statistics in Medicaine*, DOI: 10.1002/sim.5976, 2013.

[3] Martyn Plummer and Bendix Carstensen. Lexis: An R class for epidemiological studies with long-term follow-up. *Journal of Statistical Software*, 38(5):1–12, 1 2011.