

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

SDC

<http://BendixCarstensen.com/Epi>

Friday 4th October, 2013

Version 2

Compiled Friday 4th October, 2013, 16:09
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

1	Using <code>simLexis</code>	1
1.1	Introduction	1
1.2	<code>simLexis</code> in practice	1
1.2.1	Input for the simulation	2
1.3	Setting up a <code>Lexis</code> object	2
1.4	Analysis of rates	4
1.4.1	The mortality rates	6
	Proportionality of mortality rates	6
	How the mortality rates look	8
1.5	Input to the <code>simLexis</code> function	9
1.5.1	The transition object	9
1.5.2	The initial cohort	11
1.6	Simulation of the follow-up	12
1.6.1	Using other models for simulation	12
	Proportional hazards Poisson model	12
	Proportional hazards Cox model	13
1.7	Reporting the simulation results	14
1.7.1	Comparing predictions from different models	18
2	Simulation of transitions in multisate models	20
2.1	Theory	20
2.2	Components of <code>simLexis</code>	21
2.2.1	<code>simX</code>	24
2.2.2	<code>sim1</code>	25
2.2.3	<code>lint</code>	25
2.2.4	<code>get.next</code>	26
2.2.5	<code>chop.lex</code>	26
2.3	Probabilities from simulated <code>Lexis</code> objects	27
2.3.1	<code>nState</code>	27
2.3.2	<code>pState</code> , <code>plot.pState</code> and <code>lines.pState</code>	27
	References	29

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 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 fraction of the persons in the particular state at the given time. 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 contain 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 typically be an object of the same structure as the originally 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 largely a commented walk-through 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. This is provided by *simLexis*.

1.2.1 Input for the simulation

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:

1. A *preLexis* 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).

Hence, if the transition object is called `Tr` then `TrAB` (or `Tr[["A"]][["B"]]`) will represent the transition intensity from state *A* to the state *B*.

The entries in the transition object can be either *glm* objects, representing Poisson models for the transitions, *coxph* objects representing an intensity model along one time scale, or simply a function that takes a *preLexis* object as input returns an estimated intensity for each row.

In addition to these two input items, there will be a couple of tuning parameters.

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.

1.3 Setting up a Lexis object

We start by loading the *Epi* package:

```
> options( width=90 )
> library( Epi )
> print( sessionInfo(), l=F )
```

```
R version 3.0.2 (2013-09-25)
Platform: i386-w64-mingw32/i386 (32-bit)
```

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

```
other attached packages:
[1] Epi_1.1.57      foreign_0.8-55
```

```
loaded via a namespace (and not attached):
[1] tools_3.0.2
```

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 ...
 $ 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 ...
$ 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 )

```

1.4 Analysis of rates

In the `Lexis` object (which is just 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:

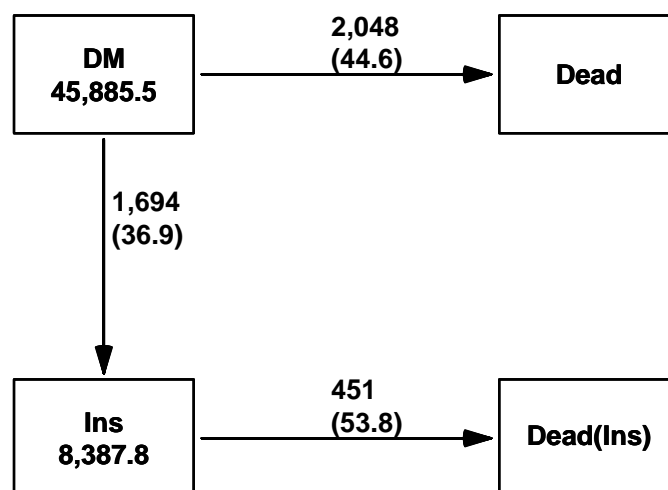


Figure 1.1: *Data overview for the dmi dataset. Numbers in the boxes are person-years, and numbers on the arrows are no. of transitions and rates (transition intensities) per 1000 PY.*

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

```
[1] 115370      15
```

```
> print( subset( Si, lex.id==97 )[,1:10], digits=6 )
```

	lex.id	Per	Age	DMdur	t.Ins	lex.dur	lex.Cst	lex.Xst	sex	dobth
1105	97	1997.55	58.9268	0.00000	NA	0.5000000	DM	DM	F	1938.62
1106	97	1998.05	59.4268	0.50000	NA	0.5000000	DM	DM	F	1938.62
1107	97	1998.55	59.9268	1.00000	NA	0.5000000	DM	DM	F	1938.62
1108	97	1999.05	60.4268	1.50000	NA	0.5000000	DM	DM	F	1938.62
1109	97	1999.55	60.9268	2.00000	NA	0.1793292	DM	Ins	F	1938.62
1110	97	1999.72	61.1061	2.17933	0.000000	0.3206708	Ins	Ins	F	1938.62
1111	97	2000.05	61.4268	2.50000	0.320671	0.5000000	Ins	Ins	F	1938.62
1112	97	2000.55	61.9268	3.00000	0.820671	0.0116359	Ins	Dead(Ins)	F	1938.62

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).

The approach that is often used in epidemiology with attaching one parameter to each intervals is not feasible when more than one time scale is used, because intervals are not classified the same way on all timescales.

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 transition out of states "DM" and "Ins". For age, 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, whereas for the timescales DMdur and tIns where we have observation from a well-defined 0, we put knots at 0 and place the remaining knots so that the number of events is the same between them and beyond the last:

```
> 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"),
+               c(0,quantile( DMdur+lex.dur, probs=(1:(nk-1))/nk ) ) ) )
```

```

                20%      40%      60%      80%
0.000000 2.000000 4.500000 6.811225 9.500000

```

```

> ( dd.kn <- with( subset(Si,lex.Xst=="Dead"),
+                 c(0,quantile( DMdur+lex.dur, probs=(1:(nk-1))/nk ) ) ) )

```

```

                20%      40%      60%      80%
0.0000000 0.7687885 2.1327858 4.0465435 6.5232033

```

```

> ( ti.kn <- with( subset(Si,lex.Xst=="Dead(Ins)"),
+                 c(0,quantile( t.Ins+lex.dur, probs=(1:(nk-1))/nk ) ) ) )

```

```

                20%      40%      60%      80%
0.0000000 0.3093771 1.1307324 2.5489391 4.9117043

```

We then 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 The mortality rates

This section discusses in some detail how to extract and display the mortality rates from the models fitted. But it is not necessary for understanding how to use *simLexis* in practice.

Proportionality of mortality rates

Note that we have fitted 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)	0.049	0.043	0.056
Ns(Age, knots = ad.kn)1	4.120	3.479	4.879
Ns(Age, knots = ad.kn)2	4.652	4.054	5.338
Ns(Age, knots = ad.kn)3	15.460	13.575	17.608
Ns(Age, knots = ad.kn)4	7.529	6.711	8.447
Ns(DMdur, knots = dd.kn)1	0.520	0.429	0.629
Ns(DMdur, knots = dd.kn)2	0.707	0.622	0.803
Ns(DMdur, knots = dd.kn)3	0.319	0.238	0.428
Ns(DMdur, knots = dd.kn)4	0.829	0.742	0.926
lex.CstIns	2.168	1.946	2.414
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	17849.90	97028.00

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

null.deviance	df.null	deviance	df.residual
4329.880	18330.000	3674.067	18316.000

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

null.deviance	df.null	deviance	df.residual
24300.15	115369.00	21608.79	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	84.822	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 constant but *different* mortality rates from the states “DM” and “Ins”. This is however irrelevant for the comparison of the *residual* deviances.

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 persons starting insulin at different times after DM. The mortality curves start at the age where the person gets diabetes and continues 12 years; for persons on insulin they start at the age when they initiate insulin.

```
> 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("red","limegreen"),each=3) )
```

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 hypothetical 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:

```
> 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 models for all transitions between them; the object *Tr**\$A**\$B* is the model for the transition intensity from state A to state B.

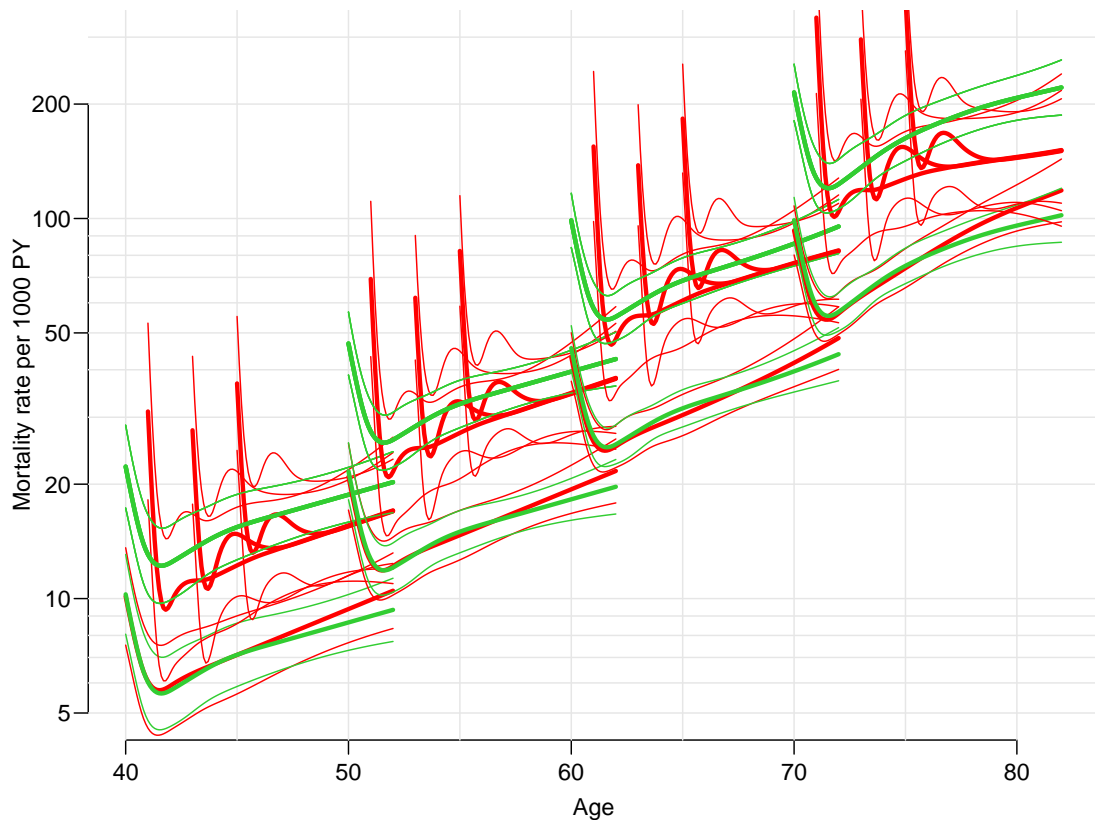


Figure 1.2: *Estimated mortality rates for male diabetes patients with no insulin (lower sets of curves) and insulin (upper curves), with DM onset in age 40, 50, 60 and 70. The red curves are from the models with separate age effects for persons with and without insulin, and a separate effect of insulin 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).*

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 when you select columns):

```
> 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":
```

```
> 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; we set up one man and one woman:

```
> 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 `t.range` argument gives the times range where the integrated intensities (cumulative rates) are evaluated and where 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 models for the transitions.

```
> system.time( simL <- simLexis( Tr,
+                               ini,
+                               t.range = 12,
+                               N = 5000 ) )
```

```
      user system elapsed
95.95    15.40    111.71
```

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.

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

\$M

Transitions:

	From	To	DM	Ins	Dead	Dead(Ins)	Records:	Events:	Risk time:	Persons:
	DM		1540	1938	1522	0	5000	3460	36824.50	5000
	Ins		0	1304	0	634	1938	634	10633.77	1938
	Sum		1540	3242	1522	634	6938	4094	47458.27	5000

\$F

Transitions:

	From	To	DM	Ins	Dead	Dead(Ins)	Records:	Events:	Risk time:	Persons:
	DM		2176	1623	1201	0	5000	2824	42222.93	5000
	Ins		0	1299	0	324	1623	324	9132.40	1623
	Sum		2176	2922	1201	324	6623	3148	51355.33	5000

1.6.1 Using other models for simulation

Proportional hazards Poisson model

We fitted a proportional mortality model `All.Dead` (which fitted worse than the other two), this is a model for *both* the transition from “DM” to “Death” *and* from “Ins” to “Dead(Ins)”, assuming that they are proportional. But it can easily be used in the simulation set-up, because the state is embedded in the model via the term `lex.Cst`, which is updated during the simulation.

Simulation using this instead just requires that we supply a different transition object:

```
> Tr.p <- list( "DM" = list( "Ins"      = DM.Ins,
+                           "Dead"     = All.Dead ),
+              "Ins" = list( "Dead(Ins)" = All.Dead ) )
> system.time( simP <- simLexis( Tr.p,
+                               ini,
+                               t.range = 12,
+                               N = 5000 ) )
```

```

      user  system elapsed
    91.51   13.37   105.12

> summary( simP, by="sex" )

$M
Transitions:
  To
From   DM  Ins Dead Dead(Ins) Records: Events: Risk time: Persons:
DM    1647 1976 1377         0     5000     3353   37720.90     5000
Ins     0 1134   0      842     1976      842   9634.54     1976
Sum   1647 3110 1377      842     6976     4195  47355.44     5000

$F
Transitions:
  To
From   DM  Ins Dead Dead(Ins) Records: Events: Risk time: Persons:
DM    2313 1679 1008         0     5000     2687   42683.39     5000
Ins     0 1184   0      495     1679      495   8764.84     1679
Sum   2313 2863 1008      495     6679     3182  51448.24     5000

```

Proportional hazards Cox model

A third possibility would be to replace the two-time scale proportional mortality model by a one-time-scale Cox-model, using diabetes duration as time scale:

```

> library( survival )
> Cox.Dead <- coxph( Surv( DMdur, DMdur+lex.dur,
+                          lex.Xst %in% c("Dead(Ins)","Dead")) ~
+                          Ns( Age-DMdur, knots=ad.kn ) +
+                          I(lex.Cst=="Ins") +
+                          I(Per-2000) + sex,
+                          data = Si )
> round( ci.exp( Cox.Dead ), 3 )

```

	exp(Est.)	2.5%	97.5%
Ns(Age - DMdur, knots = ad.kn)1	4.172	3.535	4.923
Ns(Age - DMdur, knots = ad.kn)2	4.503	3.825	5.301
Ns(Age - DMdur, knots = ad.kn)3	16.076	14.086	18.347
Ns(Age - DMdur, knots = ad.kn)4	7.478	6.500	8.604
I(lex.Cst == "Ins")TRUE	2.170	1.948	2.418
I(Per - 2000)	0.966	0.954	0.977
sexF	0.667	0.616	0.723

```

> round( ci.exp( All.Dead ), 3 )

```

	exp(Est.)	2.5%	97.5%
(Intercept)	0.049	0.043	0.056
Ns(Age, knots = ad.kn)1	4.120	3.479	4.879
Ns(Age, knots = ad.kn)2	4.652	4.054	5.338
Ns(Age, knots = ad.kn)3	15.460	13.575	17.608
Ns(Age, knots = ad.kn)4	7.529	6.711	8.447
Ns(DMdur, knots = dd.kn)1	0.520	0.429	0.629
Ns(DMdur, knots = dd.kn)2	0.707	0.622	0.803
Ns(DMdur, knots = dd.kn)3	0.319	0.238	0.428
Ns(DMdur, knots = dd.kn)4	0.829	0.742	0.926
lex.CstIns	2.168	1.946	2.414
I(Per - 2000)	0.965	0.954	0.977
sexF	0.665	0.614	0.720

Note that in order for this model to be usable for simulation, it is necessary that we use the components of the `Lexis` object to specify the survival. Each record in the dataframe `Si` represents follow up from `DMdur` to `DMdur+lex.dur`, so the model is a Cox model with diabetes duration as underlying timescale and age at diagnosis, `Age-DMdur`, as covariate.

Also note that we used `I(lex.Cst=="Ins")` instead of just `lex.Cst`, because `coxph` assigns design matrix columns to all levels of `lex.Cst`, also those not present in data, which would give NAs among the parameter estimates and NAs as mortality outcomes.

We see that the effect of insulin and the other covariates are pretty much the same as in the two-time-scale model. We can simulate from this model too; there is no restrictions on what type of model can be used for different transitions

```
> Tr.c <- list( "DM" = list( "Ins"      = Tr$DM$Ins,
+                           "Dead"     = Cox.Dead ),
+              "Ins" = list( "Dead(Ins)" = Cox.Dead ) )
> system.time( simC <- simLexis( Tr.c,
+                               ini,
+                               t.range = 12,
+                               N = 5000 ) )
```

```
      user system elapsed
87.10    13.98   101.48
```

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

\$M

Transitions:

	To								
From	DM	Ins	Dead	Dead(Ins)	Records:	Events:	Risk time:	Persons:	
DM	1768	2021	1211	0	5000	3232	37872.98	5000	
Ins	0	1349	0	672	2021	672	9796.82	2021	
Sum	1768	3370	1211	672	7021	3904	47669.80	5000	

\$F

Transitions:

	To								
From	DM	Ins	Dead	Dead(Ins)	Records:	Events:	Risk time:	Persons:	
DM	2420	1658	922	0	5000	2580	42897.55	5000	
Ins	0	1256	0	402	1658	402	8676.76	1658	
Sum	2420	2914	922	402	6658	2982	51574.30	5000	

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(0,11,0.2), from=1995, time.scale="Per" ) )
```

```
      user system elapsed
1.41     0.00     1.40
```



```
> nSt[1:10,]
```

	State			
when	DM	Ins	Dead	Dead(Ins)
1995	5000	0	0	0
1995.2	4916	57	27	0
1995.4	4837	112	50	1
1995.6	4756	162	81	1
1995.8	4677	207	113	3
1996	4583	264	150	3
1996.2	4516	292	184	8
1996.4	4440	336	214	10
1996.6	4368	371	249	12
1996.8	4289	420	277	14

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, using either the default rather colorful layout, or a more minimalistic version:

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

when	DM	Ins	Dead(Ins)	Dead
1995	1.0000	1.0000	1.0000	1
1995.2	0.9832	0.9946	0.9946	1
1995.4	0.9674	0.9898	0.9900	1
1995.6	0.9512	0.9836	0.9838	1
1995.8	0.9354	0.9768	0.9774	1
1996	0.9166	0.9694	0.9700	1

```
> par( mfrow=c(1,2), mar=c(3,3,1,1), mgp=c(3,1,0)/1.6 )
> plot( pM )
> plot( pM, border="black", col="transparent", lwd=3 )
> text( rep(as.numeric(rownames(pM)[nrow(pM)-1])),ncol(pM)),
+       pM[nrow(pM),]-diff(c(0,pM[nrow(pM),]))/5,
+       colnames( pM ), 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( pM, col=clx )
> lines( as.numeric(rownames(pM)), pM[,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] )
> 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(0,11,0.2),
+                         from=1995,
+                         time.scale="Per" ),
+               perm=c(1,2,4,3) )
```

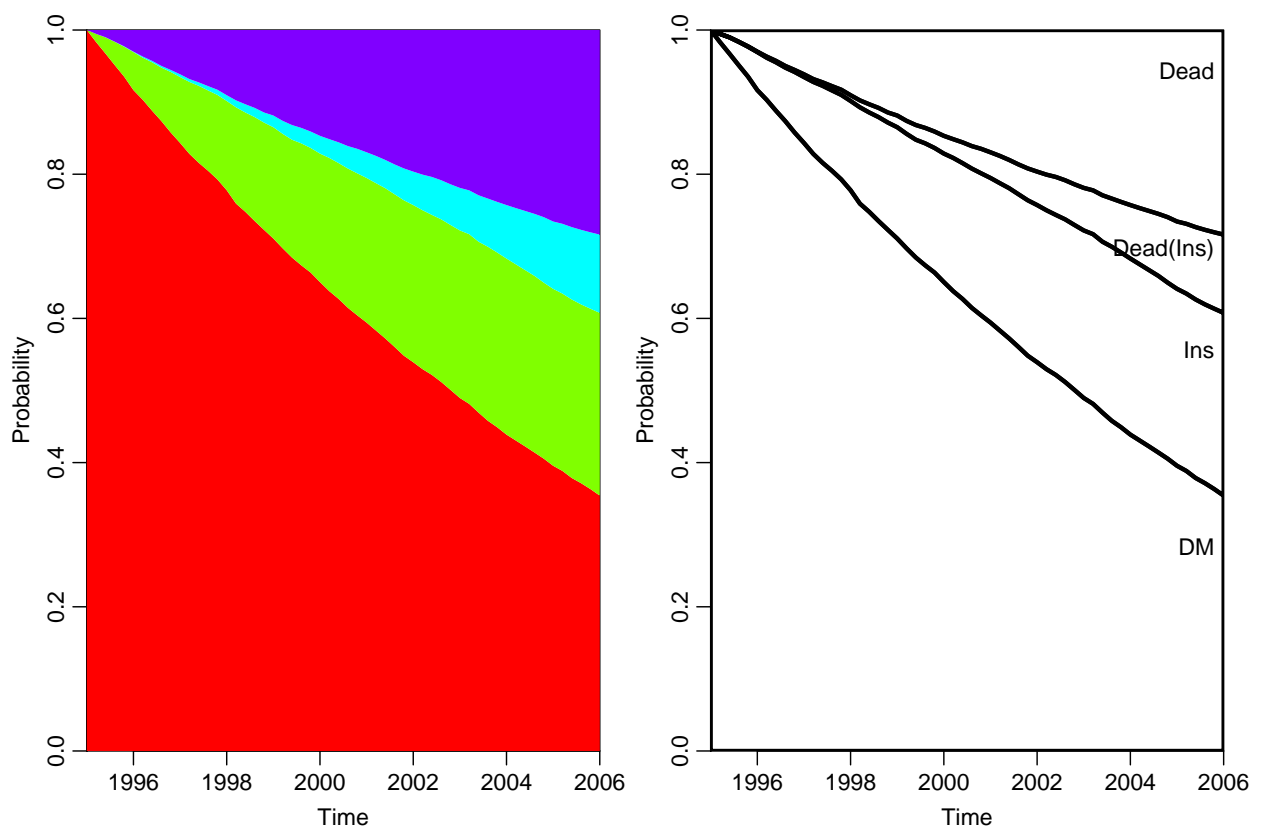


Figure 1.3: Default layout of the `plot.pState` graph (left), and a version with the state probabilities as lines and annotation of states.

```

> 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(0,11,0.2),
+                        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 )
> 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(0,11,0.2),

```

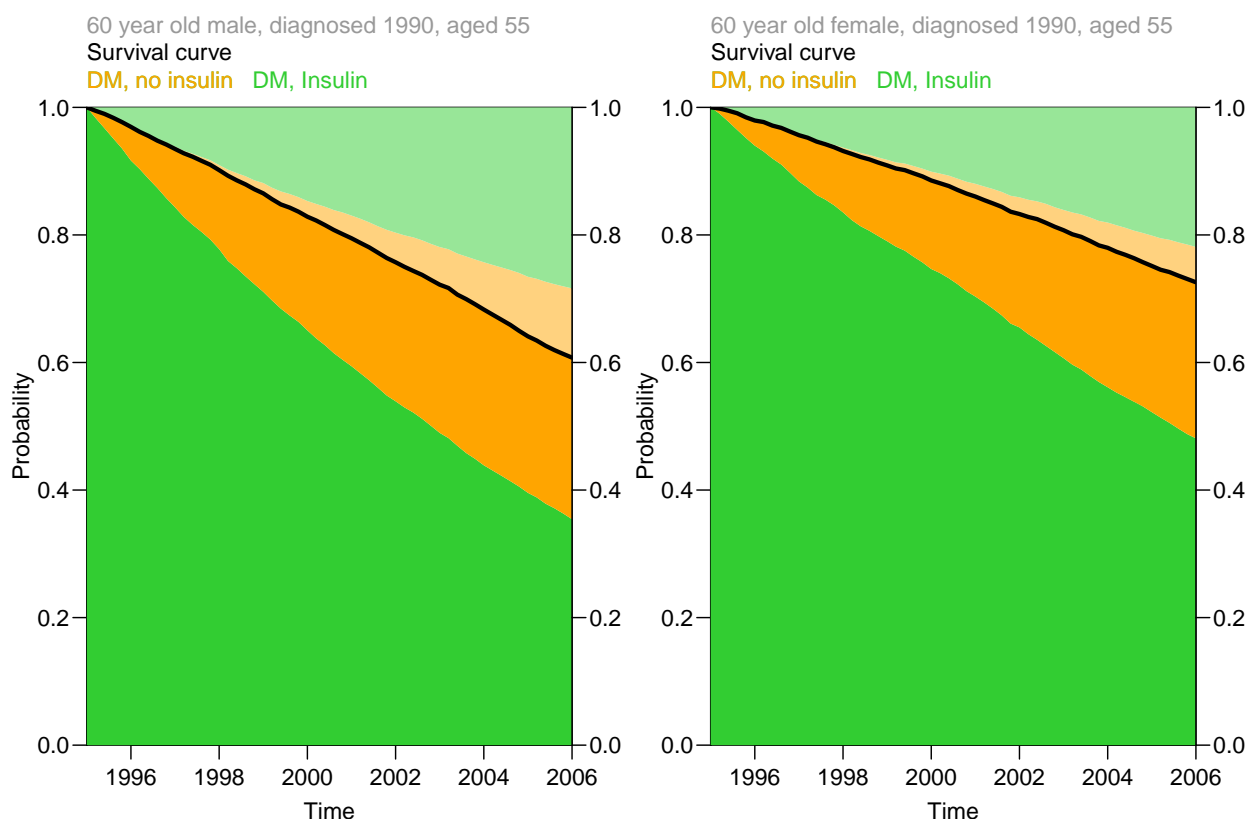


Figure 1.4: `plot.pState` graphs where persons ever on insulin are given in orange and persons never on insulin in green, and the overall survival (dead over the line) as a black line.

```

+           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 )

```

1.7.1 Comparing predictions from different models

We have actually fitted different models for the transitions, and we have simulated Lexis objects from both approaches, so we can plot these on top of each other:

```

> PrM <- pState( nState( subset(simP,sex=="M"),
+                           at=seq(0,11,0.2),
+                           from=60,
+                           time.scale="Age" ),
+               perm=c(1,2,4,3) )
> PrF <- pState( nState( subset(simP,sex=="F"),
+                           at=seq(0,11,0.2),
+                           from=60,
+                           time.scale="Age" ),
+               perm=c(1,2,4,3) )

```

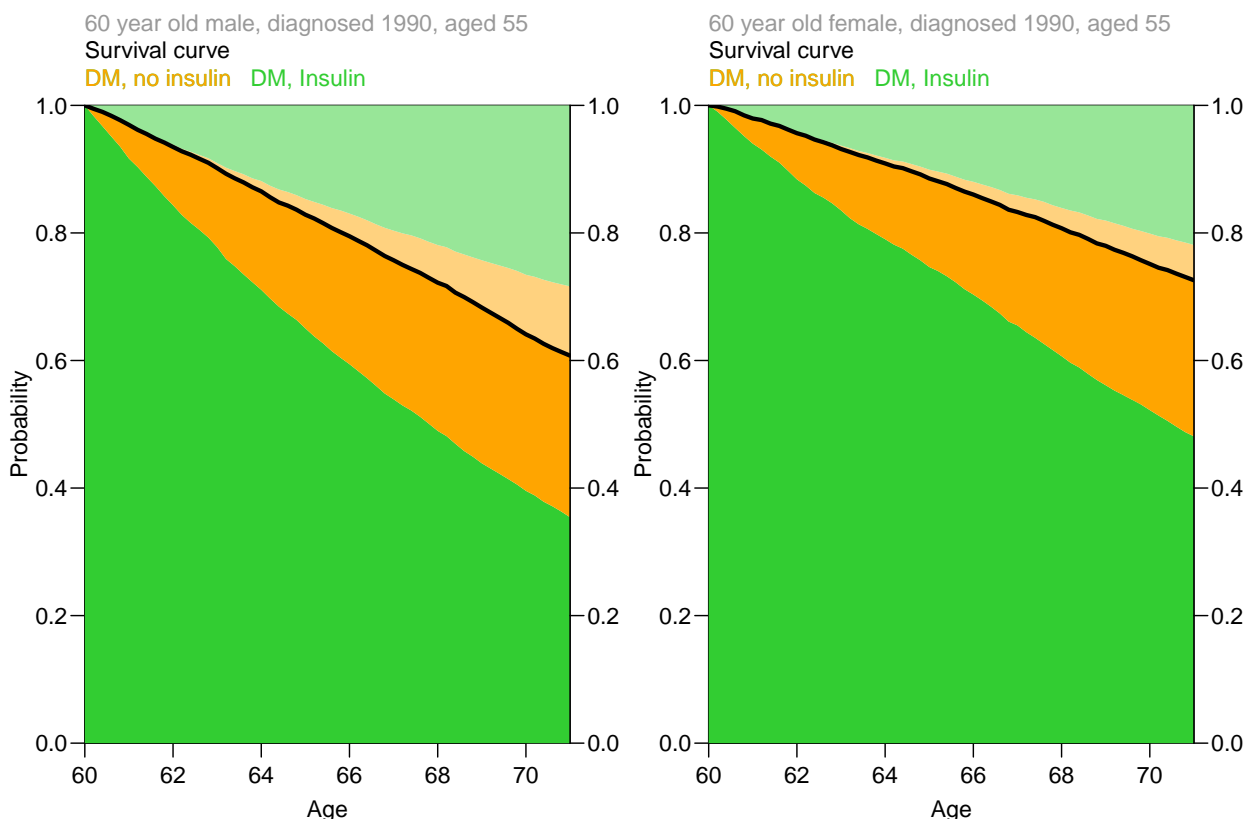


Figure 1.5: `plot.pState` graphs where persons ever on insulin are given in orange and persons never on insulin in green, and the overall survival (dead over the line) as a black line.

```

> CoxM <- pState( nState( subset(simC,sex=="M"),
+                           at=seq(0,11,0.2),
+                           from=60,
+                           time.scale="Age" ),
+                 perm=c(1,2,4,3) )
> CoxF <- pState( nState( subset(simC,sex=="F"),
+                           at=seq(0,11,0.2),
+                           from=60,
+                           time.scale="Age" ),
+                 perm=c(1,2,4,3) )
> par( mfrow=c(1,2), mar=c(3,3,1,1), mgp=c(3,1,0)/1.6 )
> plot( pM, border="black", col="transparent", lwd=3 )
> lines( PrM, border="blue", col="transparent", lwd=3 )
> lines( CoxM, border="red", col="transparent", lwd=3 )
> text( 60.5, 0.05, "M" )
> box( lwd=3 )
> plot( pF, border="black", col="transparent", lwd=3 )
> lines( PrF, border="blue", col="transparent", lwd=3 )
> lines( CoxF, border="red", col="transparent", lwd=3 )
> text( 60.5, 0.05, "F" )
> box( lwd=3 )

```

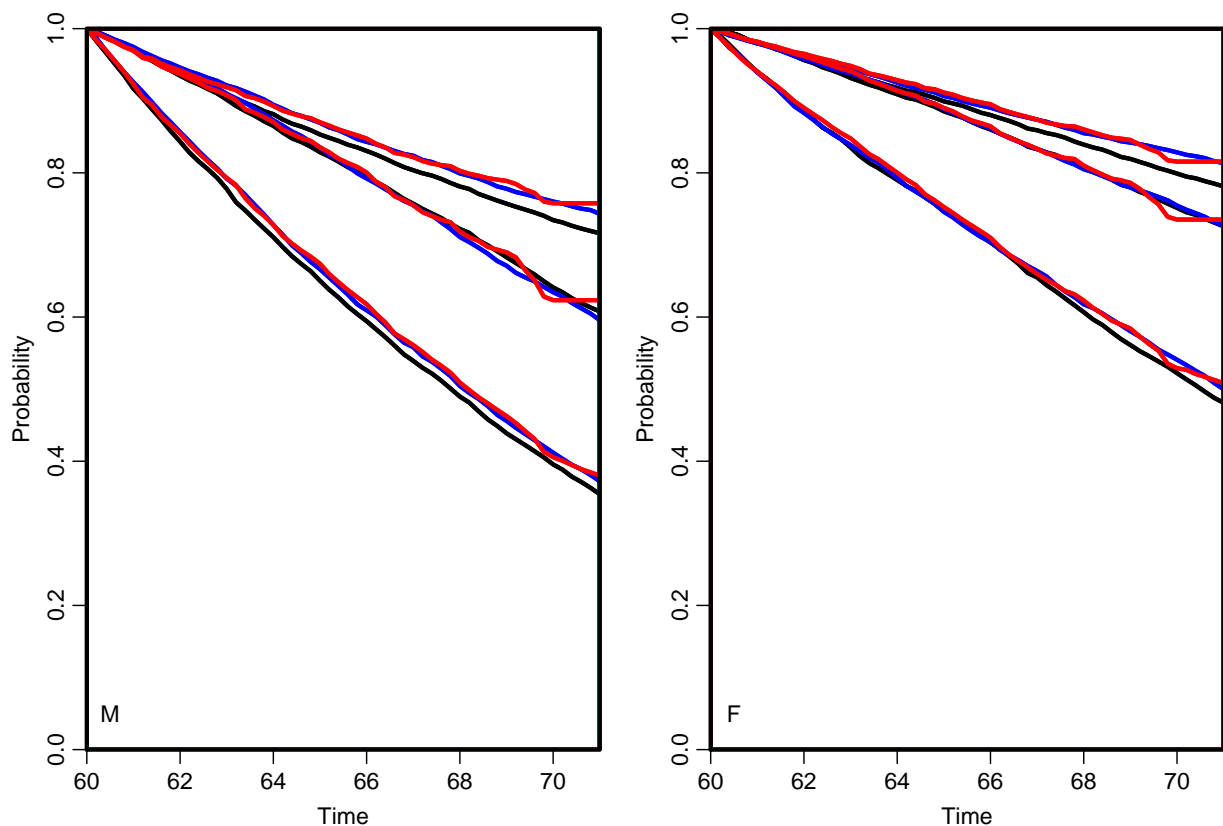


Figure 1.6: Comparison of the simulated state occupancy probabilities using separate Poisson models for the mortality rates with and without insulin (black) and using proportional hazards Poisson models (blue) and Cox-models with diabetes duration as timescale and age at diabetes diagnosis as covariate (red).

Chapter 2

Simulation of transitions in multisate models

2.1 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, $S(t) = \exp(-\Lambda_1(t) - \Lambda_2(t) - \Lambda_3(t))$
- (b) Simulate a random $U(0,1)$ variate, u , say.
- (c) The simulated exit time is then the solution t_u to the equation $S(t_u) = u \Leftrightarrow \sum_j \Lambda_j(t_u) = -\log(u)$.
- (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 the implementation in `simLexis`.

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:

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

- (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:

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

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 show that the conditional probability that the smallest of the three simulated transition times was actually the i th, is as above.

But the probabilities conditional on *survival* till t , that events of type 1, 2, 3 takes place in the interval $(t, t + dt)$ are $\lambda_1(t) dt$, $\lambda_2(t) dt$ and $\lambda_3(t) dt$, respectively, hence the conditional probabilities *given a transition time* in $(t, t + dt)$ is exactly as above:

$$\frac{\lambda_i(t) dt}{\lambda_1(t) dt + \lambda_2(t) dt + \lambda_3(t) dt} = \frac{\lambda_i(t)}{\lambda_1(t) + \lambda_2(t) + \lambda_3(t)}$$

2.2 Components of *simLexis*

This section explains the actually existing code for *simLexis*, as it is in the current version of *Epi*. The function *simLexis* takes a *preLexis* 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.

Optional arguments are `t.range`, `n.int` or `time.pts`, specifying 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 further detailed below.

```
> simLexis
```

```
function( Tr, # List of lists of transition objects
          init, # Lexis object of persons to simulate.
          N = 1, # No. persons simulated per line in init
          lex.id,
          t.range = 20, # Range for rate computation in the simulation
          n.int = 101, # length of time intervals
          time.pts = seq(0,t.range,length.out=n.int)
        )
{
  # Expand the input data frame using N and put in lex.id
  if( time.pts[1] !=0 )
    stop( "First time point must be 0, time.pts[1:3]= ",
          time.pts[1:3] )

  # Expand init
  if( !missing(N) )
  {
    if( length(N) == 1 )
      init <- init[rep(1:nrow(init),each=N),]
    else init <- init[rep(1:nrow(init),      N),]
  }
  # and update lex.id if necessary
  if( !missing(lex.id) )
  {
    if( length(lex.id)==nrow(init) )
      init$lex.id <- lex.id
    else init$lex.id <- 1:nrow(init)
  }
  else  init$lex.id <- 1:nrow(init)

  # Check/fix attributes
  if( is.null( tS <- attr(init,"time.scales") ) )
    stop( "No time.scales attribute for init" )
```



```

if( is.null( tF <- attr(init,"time.since") ) )
{
  attr(init,"time.since") <- tF <- rep( "", tS )
  warning( "'time.since' attribute set to blanks" )
}

# Convenience constants
np <- length( time.pts )
tr.st <- names( Tr )

# Set up a NULL object to hold the follow-up records
sf <- NULL

# Take only those who start in a transient state
nxt <- init[init$lex.Cst %in% tr.st,]

# If some are not in a transient state then say so
if( nrow(nxt) < nrow(init) )
{
  tt <- table(init$lex.Cst)
  tt <- tt[tt>0]
  nt <- length(tt)
  warning("\nSome initiators start in a absorbing state\n",
    "Initiator states represented are: ",
    paste( rbind( names(tt), rep(":",nt),
      paste(tt), rep(" ",nt) ), collapse="" ), "\n",
    "Transient states are: ", paste( names( Tr ), coll=" " ) )
  if( nrow(nxt)==0 ) stop( "\nNo initiators in transient states!" )
}

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

# Doctor lex.Xst levels, fix values for the censored
sf$lex.Xst <- factor( sf$lex.Xst, levels=levels(sf$lex.Cst) )
sf$lex.Xst[is.na(sf$lex.Xst)] <- sf$lex.Cst[is.na(sf$lex.Xst)]

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

```

2.2.1 `simX`

`simX` is called by `simLexis` and 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.

Note that it is here the switch between `glm`, `coxph` and objects of class `function` is made.

The dataset on which this prediction is done has `length(time.pts)` rows per person.

```
> Epi::simX
```

```
function( nd, Tr, time.pts, tS )
{
  # 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
  prfrm <- nd[rep(1:nr,each=np),]
  prfrm[,tS] <- prfrm[,tS] + rep(time.pts,nr)
  prfrm$lex.dur <- il <- min( diff(time.pts) )
  # Poisson-models should use the estimated rate at the midpoint of the
  # intervals:
  prfrp <- prfrm
  prfrp[,tS] <- prfrp[,tS]+il/2

  # Make a data frame with predicted rates for each of the transitions
  # out of this state for these times
  rt <- data.frame( lex.id = prfrm$lex.id )
  for( i in 1:length(Tr[[cst]]) )
  {
    if( inherits( Tr[[cst]][[i]], "glm" ) )
      rt <- cbind( rt, predict( Tr[[cst]][[i]],
                              type="response",
                              newdata=prfrp ) )
    else
      if( inherits( Tr[[cst]][[i]], "coxph" ) )
        rt <- cbind( rt, predict( Tr[[cst]][[i]],
                                type="expected",
                                newdata=prfrm ) )
      else
        if( is.function( Tr[[cst]][[i]] ) )
          rt <- cbind( rt, Tr[[cst]][[i]](prfrm) )
        else
          stop( "Invalid object supplied as transition, must be either\n",
                "glm(poisson) or coxph object fitted to a Lexis object or\n",
                "a function that takes a Lexis object as argument and returns\n",
                "average rates for each record in the same units as lex.dur.")
    }
  }
  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,
                         time.pts ) ),
       by="lex.id" )
}
<environment: namespace:Epi>
```

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

2.2.2 `sim1`

The predicted cumulative intensities are fed, person by person, to `sim1` — again via a `do.call` / `lapply` / `split` construction — and the resulting time and state is appended to the `nd` data frame. This way we have simulated *one* transition for each person:

```
> Epi:::sim1
```

```
function( rt, 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( rbind(0,rt[,-1,drop=FALSE]), 2, cumsum )[1:nrow(rt),,drop=FALSE]
  tt <- uu <- -log( runif(ncol(ci)) )
  for( i in 1:ncol(ci) ) tt[i] <- lint( ci[,i], time.pts, uu[i] )
  # 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 ) )
}
<environment: namespace:Epi>
```

2.2.3 `lint`

Note that we do not use `approx` to do linear interpolation, because this function does not do the right thing if the cumulative incidences (`ci`) are constant across a number of times. Therefore we have a custom made linear interpolator that does the interpolation exploiting the fact the `ci` is non-decreasing and `tt` is monotonely increasing:

```
> Epi:::lint
```

```
function( ci, tt, u )
{
  # Makes a linear interpolation, but does not crash if all ci values are
  # identical, but requires that both ci and tt are non-decreasing
  if( any( diff(ci)<0 ) | any( diff(tt)<0 ) ) stop("Non-increasing arguments")
  c.u <- min( c( ci[ci>u], max(ci) ) )
  c.l <- max( c( ci[ci<u], min(ci) ) )
  t.u <- min( c( tt[ci>u], max(tt) ) )
  t.l <- max( c( tt[ci<u], min(tt) ) )
  # c.u==c.l if u is outside the range of ci
  ifelse( c.u==c.l, t.l, t.l + (u-c.l)/(c.u-c.l)*(t.u-t.l) )
}
<environment: namespace:Epi>
```

2.2.4 `get.next`

We must repeat the simulation 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, tr.st, tS, tF )
{
# 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$lex.Xst %in% tr.st,]
if( nrow(nxt) == 0 ) return( nxt )
nxt[,tS] <- nxt[,tS] + nxt$lex.dur
wh <- tF
for( i in 1:length(wh) )
  if( wh[i] != "" ) nxt[nxt$lex.Xst==wh[i],tS[i]] <- 0
nxt$lex.Cst <- nxt$lex.Xst
return( nxt )
}
<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, tS, cens )
{
# A function that chops off all follow-up beyond cens since entry for
# each individual
# Entry times on 1st timescale
zz <- entry( obj, 1, by.id=TRUE )
# Merge with the revised exit times on this timescale
ww <- merge( obj, data.frame( lex.id = as.numeric(names(zz)),
                             cens = zz+cens ) )
# Only retain records with an entry time prior to the revised exit time
ww <- ww[ww[,tS[1]] < ww$cens,]
# Revise the duration according to the revised exit time
x.dur <- pmin( ww$lex.dur, ww[, "cens"]-ww[,tS[1]] )
# Change lex.Xst to lex.Cst for those with shortened follow-up
ww$lex.Xst[x.dur<ww$lex.dur] <- ww$lex.Cst[x.dur<ww$lex.dur]
# Insert the updated follow-up time
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
  tS <- 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(tS,"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[,tS] <= tab.frm$when &
                    tab.frm[,tS]+tab.frm$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[,tS]+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` and `lines.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`, which returns a matrix of class `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` and `lines` method for the resulting `pState` objects:

```
> plot.pState
```

```

function( x,
          col = rainbow(ncol(x)),
          border = "transparent",
          xlab = "Time",
          ylim = 0:1,
          ylab = "Probability", ... )
{
# Function to plot cumulative probabilities along the time scale.
matplot( as.numeric(rownames(x)), x, type="n",
          ylim=ylim, yaxs="i", xaxs="i",
          xlab=xlab, ylab=ylab, ... )
lines.pState( x,
              col = col,
              border = border, ... )
}
<environment: namespace:Epi>

```

```
> lines.pState
```

```

function( x,
          col = rainbow(ncol(x)),
          border = "transparent", ... )
{
# 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 )
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 Medicine*, e-pub ahead of print: 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.