Simulation in multistate models with multiple timescales

SDC
http://BendixCarstensen.com/Epi
Saturday 23rd February, 2013
Version 1.1

Compiled Saturday 23rd February, 2013, 10:57
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 Introduction

This paper explains the machinery behind simulation of life histories through multistate models where transition rates are allowed to depend on multiple time scales, including timescales defined as time since entry to a particular state (duration). This also covers the case where time *at* entry into a state is an explanatory variable for the rates, since time at entry merely is the difference between time and duration.

2 Simulation setup for Poisson models based on Lexis objects

For the sake of the argument we first take a small example. In order to keep track of the transitions we will set up a list of the glm objects that models the transitions. It is the assumption that they all are modelled using the relevant subsets of the base `Lexis` object. So it means that the prediction of rates and hence the calculation of cumulative rates relies on using a Lexis object.

We shall use the `DMlate` dataset from the `Epi` package to illustrate the construction of the machinery. We set up a Lexis object using `example(DMlate)`:

```
> library( Epi )
> sessionInfo()
```

```
R version 2.15.2 (2012-10-26)
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_1.1.45     foreign_0.8-51

loaded via a namespace (and not attached):
[1] tools_2.15.2
```

```
> example( DMlate )
```

```
DMlate> data(DMlate)

DMlate> str(DMlate)
'data.frame':       10000 obs. of  7 variables:
 $ 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 ...

DMlate> dml <- Lexis( entry=list(Per=dodm, Age=dodm-dobth, DMdur=0 ),
```

```
DMlate+                 exit=list(Per=dox),
DMlate+          exit.status=factor(!is.na(dodth),labels=c("DM","Dead")),
DMlate+                 data=DMlate )
NOTE: entry.status has been set to "DM" for all.

DMlate> # Split follow-up at insulin, introduce a new timescale,
DMlate> # and split non-precursor states
DMlate> system.time(
DMlate+ dmi <- cutLexis( dml, cut = dml$doins,
DMlate+                       pre = "DM",
DMlate+                 new.state = "Ins",
DMlate+                 new.scale = "t.Ins",
DMlate+              split.states = TRUE ) )
   user  system elapsed
   2.62    0.02    2.64

DMlate> 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
```

```
> attributes(dmi)[-2]
```

```
$names
 [1] "Per"     "Age"     "DMdur"   "t.Ins"   "lex.dur" "lex.Cst" "lex.Xst"
 [8] "lex.id"  "sex"     "dobth"   "dodm"    "dodth"   "dooad"   "doins"
[15] "dox"

$class
[1] "Lexis"      "data.frame"

$time.scales
[1] "Per"   "Age"   "DMdur" "t.Ins"

$time.since
[1] ""      ""      ""      "Ins"

$breaks
$breaks$Per
NULL

$breaks$Age
NULL

$breaks$DMdur
NULL

$breaks$t.Ins
NULL
```

```
> timeScales( dmi )
```

```
[1] "Per"   "Age"   "DMdur" "t.Ins"
```

We shall later need in indication of which of the timescales that appear as “time since entry” to some state (well, we are using the already updated version 1.1.45 of Epi, so it is already there):

```
> attr( dmi, "time.since" ) <- c("","","","Ins")
```

We will now model the three different transitions, shown in figure 1

```
> boxes.Lexis( dmi, boxpos=list( x=c(20,20,80,80),
+                              y=c(80,20,80,20) ), scale.R=1000, pos.arr=c(0.5,0.3,0.3) )
```

The point is now to define a structure that represents the (in this case 3) Poisson models for the transitions.

Before we fit the models, we first split the data, in order to be able to include effects of the time-scales:

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

```
Transitions:
     To
From      DM   Ins Dead Dead(Ins)  Records:  Events: Risk time:  Persons:
  DM   93297  1694 2048         0     97039     3742   45885.49      9899
  Ins      0 17880    0       451     18331      451    8387.77      1791
  Sum 93297 19574 2048       451    115370     4193   54273.27      9996
```

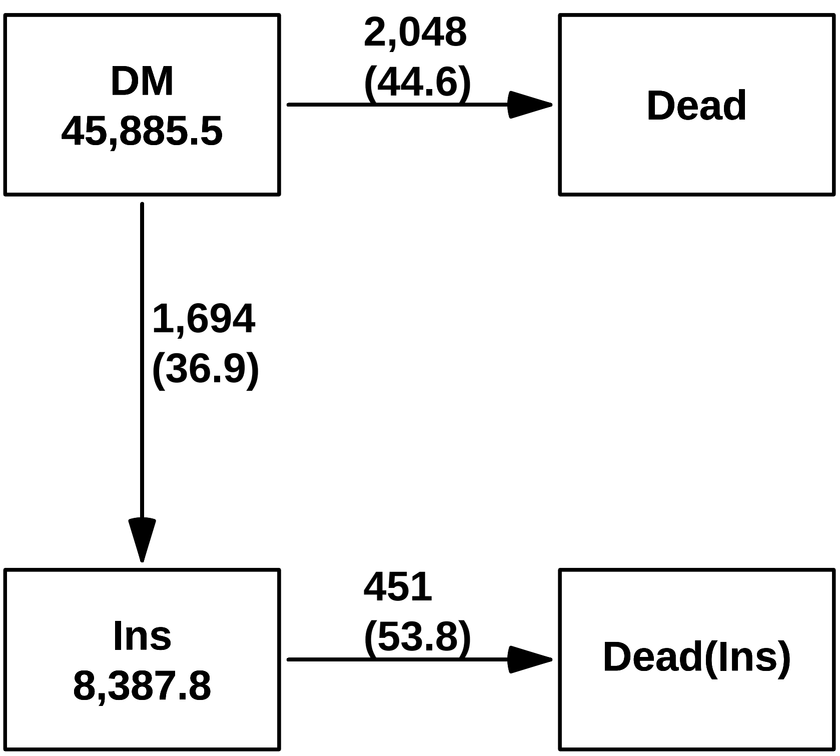

Figure 1: *No. of transitions between states, and average transitions rates per 1000 PY.*

```
> attr( Si, "time.since" ) <- c("","","","Ins")
> attributes( Si )[-2]
```

```
$names
 [1] "lex.id"  "Per"     "Age"     "DMdur"   "t.Ins"   "lex.dur" "lex.Cst"
 [8] "lex.Xst" "sex"     "dobth"   "dodm"    "dodth"   "dooad"   "doins"
[15] "dox"

$breaks
$breaks$Per
NULL

$breaks$Age
NULL

$breaks$DMdur
 [1]  0.0  0.5  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0  5.5  6.0  6.5  7.0
[16]  7.5  8.0  8.5  9.0  9.5 10.0 10.5 11.0 11.5 12.0 12.5 13.0 13.5 14.0 14.5
[31] 15.0

$breaks$t.Ins
NULL


$time.scales
[1] "Per"   "Age"   "DMdur" "t.Ins"

$time.since
[1] ""    ""    ""    "Ins"

$class
[1] "Lexis"      "data.frame"
```

Then we define the number of knots we will use for modelling of age, DM-duration and insulin-duration — period will just be modelled linearly:

```
> nk <- 4
> ( ai.kn <- with( subset(Si,lex.Xst=="Ins"),
+                  quantile( Age+lex.dur, probs=(1:nk-0.5)/nk ) ) )
```

```
   12.5%    37.5%    62.5%    87.5%
28.00642 50.05600 62.12076 75.69020
```

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

```
   12.5%    37.5%    62.5%    87.5%
63.61875 74.98700 81.38501 89.26831
```

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

```
12.5% 37.5% 62.5% 87.5%
  1.5   4.0   7.0  10.5
```

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

```
    12.5%     37.5%     62.5%     87.5% 
0.3778234 1.9582478 4.3370979 8.0232717 
```

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

```
    12.5%     37.5%     62.5%     87.5% 
0.1759069 1.0095825 2.7939767 6.3579740 
```

Now we can model the three transitions; note that we use `lex.dur` as the offset argument, because the subsequent simulation machinery will rely on this. Similarly, when an intermediate state is used as a mere hazard multiplier, we must use `lex.Cst` as argument. The latter is not illustrated here, as we model the two mortality rates separately:

```
> library( splines )
> source("c:/stat/r/bxc/library.sources/useful/r/Ns.R")
> Ns
```

```
function (x, df = NULL, knots = NULL, intercept = FALSE, Boundary.knots = NULL) 
{
    if (is.null(Boundary.knots)) {
        if (!is.null(knots)) {
            knots <- sort(unique(knots))
            ok <- c(1, length(knots))
            Boundary.knots <- knots[ok]
            knots <- knots[-ok]
        }
    }
    ns(x, df = df, knots = knots, intercept = intercept, Boundary.knots = Boundary.knots)
}
```

```
> 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=td.kn ) + I(Per-2000) + sex,
+               family=poisson, offset=log(lex.dur),
+               data = subset(Si,lex.Cst=="Ins") )
```

We can now place these three glms 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)"
```

Thus in this case `Tr$DM$Ins` is the `glm` object that models the transition from “DM” to “Ins”.

Now we want to simulate transitions according to this model for a (group of) persons. In order to do this 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 subsequently 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
```

```
> str( ini )
```

```
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.dur: num  NA NA
 $ lex.Cst: Factor w/ 4 levels "DM","Ins","Dead",..: 1 1
 $ lex.Xst: Factor w/ 4 levels "DM","Ins","Dead",..: NA NA
 $ sex    : Factor w/ 2 levels "M","F": 1 2
 - 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"
```

Eventually, a data frame like this and an object like `Tr` will be the input to a prediction function for a multistate model based on a `Lexis` object.

Now we predict the cumulative incidence based on these persons using the value of `lex.Cst` to select the relevant element of `Tr`: The prediction of the survival function is in `np` points, starting at 0, so 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<- 50      # How long into the future should we predict
> ni  <- 25       # 20 intervals would be more realistical
> int <- tmax/ni
> pt  <- 0:ni*int
> 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 averages over the intervals, that is the mean of the intensities at the two endpoints, so we define a small function to do that kind of calculation:

```
> 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 incidence in an array classified by transition, time for FU and person, only assuming that all persons are in the same current state.

So we want to 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)
> cbind( pt, nd )
```

```
     pt lex.id  Per Age DMdur t.Ins lex.dur lex.Cst lex.Xst sex
1     0      1 2000  50     1    NA      NA      DM    <NA>   M
1.1   2      1 2002  52     3    NA      NA      DM    <NA>   M
1.2   4      1 2004  54     5    NA      NA      DM    <NA>   M
1.3   6      1 2006  56     7    NA      NA      DM    <NA>   M
1.4   8      1 2008  58     9    NA      NA      DM    <NA>   M
1.5  10      1 2010  60    11    NA      NA      DM    <NA>   M
1.6  12      1 2012  62    13    NA      NA      DM    <NA>   M
1.7  14      1 2014  64    15    NA      NA      DM    <NA>   M
1.8  16      1 2016  66    17    NA      NA      DM    <NA>   M
1.9  18      1 2018  68    19    NA      NA      DM    <NA>   M
1.10 20      1 2020  70    21    NA      NA      DM    <NA>   M
1.11 22      1 2022  72    23    NA      NA      DM    <NA>   M
1.12 24      1 2024  74    25    NA      NA      DM    <NA>   M
1.13 26      1 2026  76    27    NA      NA      DM    <NA>   M
1.14 28      1 2028  78    29    NA      NA      DM    <NA>   M
1.15 30      1 2030  80    31    NA      NA      DM    <NA>   M
1.16 32      1 2032  82    33    NA      NA      DM    <NA>   M
1.17 34      1 2034  84    35    NA      NA      DM    <NA>   M
1.18 36      1 2036  86    37    NA      NA      DM    <NA>   M
1.19 38      1 2038  88    39    NA      NA      DM    <NA>   M
1.20 40      1 2040  90    41    NA      NA      DM    <NA>   M
1.21 42      1 2042  92    43    NA      NA      DM    <NA>   M
1.22 44      1 2044  94    45    NA      NA      DM    <NA>   M
1.23 46      1 2046  96    47    NA      NA      DM    <NA>   M
1.24 48      1 2048  98    49    NA      NA      DM    <NA>   M
1.25 50      1 2050 100    51    NA      NA      DM    <NA>   M
2     0      2 2000  50     1    NA      NA      DM    <NA>   F
2.1   2      2 2002  52     3    NA      NA      DM    <NA>   F
2.2   4      2 2004  54     5    NA      NA      DM    <NA>   F
```

```
2.3    6       2 2006  56     7    NA      NA      DM    <NA>   F
2.4    8       2 2008  58     9    NA      NA      DM    <NA>   F
2.5   10       2 2010  60    11    NA      NA      DM    <NA>   F
2.6   12       2 2012  62    13    NA      NA      DM    <NA>   F
2.7   14       2 2014  64    15    NA      NA      DM    <NA>   F
2.8   16       2 2016  66    17    NA      NA      DM    <NA>   F
2.9   18       2 2018  68    19    NA      NA      DM    <NA>   F
2.10  20       2 2020  70    21    NA      NA      DM    <NA>   F
2.11  22       2 2022  72    23    NA      NA      DM    <NA>   F
2.12  24       2 2024  74    25    NA      NA      DM    <NA>   F
2.13  26       2 2026  76    27    NA      NA      DM    <NA>   F
2.14  28       2 2028  78    29    NA      NA      DM    <NA>   F
2.15  30       2 2030  80    31    NA      NA      DM    <NA>   F
2.16  32       2 2032  82    33    NA      NA      DM    <NA>   F
2.17  34       2 2034  84    35    NA      NA      DM    <NA>   F
2.18  36       2 2036  86    37    NA      NA      DM    <NA>   F
2.19  38       2 2038  88    39    NA      NA      DM    <NA>   F
2.20  40       2 2040  90    41    NA      NA      DM    <NA>   F
2.21  42       2 2042  92    43    NA      NA      DM    <NA>   F
2.22  44       2 2044  94    45    NA      NA      DM    <NA>   F
2.23  46       2 2046  96    47    NA      NA      DM    <NA>   F
2.24  48       2 2048  98    49    NA      NA      DM    <NA>   F
2.25  50       2 2050 100    51    NA      NA      DM    <NA>   F
```

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 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 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 0.05220453 0.01238514
1.1      1 0.01690560 0.01036758
1.2      1 0.01763702 0.01312218
1.3      1 0.04514729 0.01512384
1.4      1 0.05646396 0.01656640
1.5      1 0.04227533 0.01810780
```

So now `inc` contains the estimated rates at specific time points of follow-up, so now 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 — theory

Suppose that the rates out of the current state are λ_1, λ_2 and λ_3, and the corresponding cumulative rates are Λ_1, Λ_2 and Λ_3. If we want to simulate an exit time and an exit state (that is either 1 or 2). 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 $S(t_u) = u \Leftrightarrow \sum_j \Lambda_j(t_u) = -\log(u)$.

(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 variate u_1, u_2 and u_3.

   (b) Solve the equations $\Lambda(t_i) = -\log(u_i)$ 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 $k \in \{1, 2, 3\}$, 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, and the first one wins.

The formal argument goes as follows: [...]

2.2 Simulation of transition times — implementation

We shall use the latter approach here.

This is done for a single person using split and then applying a function that returns the time and the state

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

```
     lex.id          Ins       Dead
1         1 5.220453e-02 0.01238514
1.1       1 1.690560e-02 0.01036758
1.2       1 1.763702e-02 0.01312218
1.3       1 4.514729e-02 0.01512384
1.4       1 5.646396e-02 0.01656640
1.5       1 4.227533e-02 0.01810780
1.6       1 3.007524e-02 0.01979262
1.7       1 2.124267e-02 0.02163423
1.8       1 1.489624e-02 0.02365509
1.9       1 1.038359e-02 0.02590161
1.10      1 7.203742e-03 0.02843595
1.11      1 4.980168e-03 0.03133772
1.12      1 3.435113e-03 0.03470927
1.13      1 2.366927e-03 0.03867982
1.14      1 1.630703e-03 0.04333417
1.15      1 1.123478e-03 0.04867521
1.16      1 7.740244e-04 0.05466543
1.17      1 5.332668e-04 0.06128462
1.18      1 3.673961e-04 0.06860836
1.19      1 2.531188e-04 0.07674006
1.20      1 1.743871e-04 0.08580600
1.21      1 1.201446e-04 0.09594080
1.22      1 8.277405e-05 0.10727264
1.23      1 5.702747e-05 0.11994292
1.24      1 3.928928e-05 0.13410973
1.25      1 2.706849e-05 0.14994981
```

```
> 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 35.24372
```

```
> list( min(tt), colnames(ci)[tt==min(tt)] )
```

```
[[1]]
[1] NA

[[2]]
[1] NA NA
```

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) ) )
+ }
```

So we get

```
> sim1( subset(inc,lex.id==1), pt )
```

```
  lex.id  lex.dur lex.Xst
1      1 36.11108    Dead
```

```
> sim1( subset(inc,lex.id==2), pt )
```

```
  lex.id  lex.dur lex.Xst
1      2 11.41325    Dead
```

If we want to assemble this, we must 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 28.84452    Dead
2      2 50.00000    <NA>
```

This is the 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  28.8 50
 $ lex.Xst: Factor w/ 4 levels "DM","Ins","Dead",..: 3 NA
 - 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 28.84452    Dead
2      2 2000  50     1    NA      DM   F 50.00000    <NA>
```

Then we can split this data frame, by taking those who exited to a transient state and those who did not.

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

```
> tr.states <- names( Tr )
> # The final rows
> 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
> # Rows requiring another simulation
> ini.nxt <- subset( ini.upd,  lex.Xst %in% tr.states )
> # This should be automatic
> attr( ini.nxt, "time.since" ) <- attr( ini, "time.since" )
> ini.nxt[,timeScales(ini.nxt)] <- ini.nxt[,timeScales(ini.nxt)] + ini.nxt$lex.dur
> for( i in 1:length(wh<-attr(ini.nxt,"time.since")) )
+    if( wh[i] != "" & sum(ini.nxt$lex.Xst==wh[i])>0 )
+        ini.nxt[ini.nxt$lex.Xst==wh[i],timeScales(ini.nxt)[i]] <- 0
> ini.nxt$lex.Cst <- ini.nxt$lex.Xst
> ini.nxt$lex.dur <- 1
> ini.nxt
```

3 Putting it all together in a function

There are two main arguments to a function to simulate from a multistate model which is represented in a `Lexis` object:

1. A `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.

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 given a `Lexis` object produces estimates of the transition intensities at the time points given in the supplied `Lexis` object.

   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 of the stats *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".

Alternatively the entries can be `glm` objects, in which case we just substitute by `function(nd) exp(predict(glm.obj,newdata=nd))`.

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.

3.1 Components of `simLexis`

The function `simLexis` need a `Lexis` object as input. This defines the initial state(s) and times of the start. Since the purpose is to simulate a history through the estimated multistate model, the variables `lex.Xst` and `lex.dur` are ignored.

Note 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:

```
> attr( ini, "time.since" )
```

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

Lexis objects created with Epi version 1.1.45 or later will have this attribute set for time scales created with `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 a multistate model. It is assumed (and not checked) that timescales enter in the model via the timescales of the `Lexis` object and also the variable `lex.dur` enters in the offset of 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` and `split` construction to do simulations for different initial states:

```
> 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?
        type = "glm-mult"
         )
{
# Expand the input data frame using N
if( length(N)>1 )
  {
  if( length(N)!=nrow(init) ) stop( "N has ", length(N) , " elements, but\n",
                                "init has ", nrow(init), " rows; must be the same.\n" )
  else init <- init[rep(1:nrow(init),N),]
  }
else if( N>1 ) init <- init[rep(1:nrow(init),each=N),]

# Make sure that each line represents one person
init$lex.id <- 1:nrow(init)

# 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>
```

This construction calls the function `simX`, which 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 nd by chunks of starting state, lex.Cst
# Necessary because different states have different (sets of) exit
# rates. Therefore, this simulates for a set of persons from
# the same starting state.
np <- length( time.pts )
nr <- nrow( nd )
if( nr==0 ) return( NULL )
cst <- unique( nd$lex.Cst )
if( length(cst)>1 ) stop( "More than one lex.Cst present.\n" )
# Expand each person by the timepoints
nx <- nd[rep(1:nrow(nd),each=np),]
nx[,timeScales(init)] <- nx[,timeScales(init)] + rep(time.pts,nr)
nx$lex.dur <- 1
# Make a dataframe with predicted rates for each of the transitions
# out of this state for these times
rt <- data.frame( lex.id=nx$lex.id )
for( i in 1:length(Tr[[cst]]) ) rt <- cbind( rt, exp(predict(Tr[[cst]][[i]],newdata=nx)) )
names( rt )[-1] <- names( Tr[[cst]] )
# Then we find the transition time and exit state for each person:
xx <- match( c("lex.dur","lex.Xst"), names(nd) )
if( any( !is.na(xx) ) ) nd <- nd[,-xx[!is.na(xx)]]
merge( nd, do.call( "rbind", lapply( split(rt,rt$lex.id), sim1, init, time.pts ) ), by="lex.id" )
}
<environment: namespace:Epi>
```

This is fed, person by person, to `sim1` — again via a `do.call` - `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 dataframe
# rt with columns lex.id and timescales. Each row in rt is 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>
```

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 )
{
# Procduces an initial Lexis object for the next simulation for those
# who have ended up in a transient state
# Note that this exploits the existance of the "time.since" attribute
# for Lexis objects and assumes that a charcter 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>
```

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>
```

4 Functions for deriving statistics from simulated `Lexis` objects

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

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 the 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 )
```

```
{
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 correponding 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>
```

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

```
> pState


function( nSt, perm=1:ncol(nSt) )
{
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 = col,
       xlab = "Time",
       ylab = "Probability", ... )
{
# 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>
```

5 How it actually works

This is just a walk-trough of the example from the help-page of `simLexis`, with somewhat more realistic parameters supplied.

First we take 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.
```

Split follow-up at insulin, introduce a new timescale, and split non-precursor states, so that we can address the question of ever been on insulin:

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

Then we split the follow in 1-year intervals for modelling

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

Define knots for the analysis of rates

```
> nk <- 4
> ( ai.kn <- with( subset(Si,lex.Xst=="Ins"),
+                  quantile( Age+lex.dur, probs=(1:nk-0.5)/nk ) ) )
```

```
   12.5%    37.5%    62.5%    87.5%
28.00642 50.05600 62.12076 75.69020
```

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

```
   12.5%    37.5%    62.5%    87.5%
63.61875 74.98700 81.38501 89.26831
```

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

```
12.5% 37.5% 62.5% 87.5%
  1.5   4.0   7.0  10.5
```

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

```
    12.5%     37.5%     62.5%     87.5%
0.3778234 1.9582478 4.3370979 8.0232717
```

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

    12.5%     37.5%     62.5%     87.5%
0.1759069 1.0095825 2.7939767 6.3579740
```

Fit Poisson models to transition rates

```
> library( splines )
> DM.Ins <- glm( (lex.Xst=="Ins") ~ ns( Age  , knots=ai.kn[2:(nk-1)], Bo=ai.kn[c(1,nk)] ) +
+                                   ns( DMdur, knots=di.kn[2:(nk-1)], Bo=di.kn[c(1,nk)] ) +
+                                   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[2:(nk-1)], Bo=ad.kn[c(1,nk)] ) +
+                                     ns( DMdur, knots=dd.kn[2:(nk-1)], Bo=dd.kn[c(1,nk)] ) +
+                                     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[2:(nk-1)], Bo=ad.kn[c(1,nk)] ) +
+                                           ns( DMdur, knots=dd.kn[2:(nk-1)], Bo=dd.kn[c(1,nk)] ) +
+                                           ns( t.Ins, knots=td.kn[2:(nk-1)], Bo=td.kn[c(1,nk)] ) +
+                                           I(Per-2000) + sex,
+                family=poisson, offset=log(lex.dur),
+                data = subset(Si,lex.Cst=="Ins") )
```

Stuff 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 all states and all transitions between them.

We now define an initial object of persons in a given state with all relevant covariates defined. These will be the persons that we simulate through the model:

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

Simulate 5000 of each sex 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.

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

   user  system elapsed
  84.25   12.68   97.17
```

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

```
$M

Transitions:
     To
From   DM  Ins Dead Dead(Ins)  Records:  Events: Risk time:  Persons:
  DM  938 1798 2264         0      5000     4062   47692.66      5000
  Ins   0  728    0      1070      1798     1070   17535.79      1798
  Sum 938 2526 2264      1070      6798     5132   65228.44      5000

$F

Transitions:
     To
From    DM  Ins Dead Dead(Ins)  Records:  Events: Risk time:  Persons:
  DM  1606 1480 1914         0      5000     3394   58233.12      5000
  Ins    0  818    0       662      1480      662   16033.00      1480
  Sum 1606 2298 1914       662      6480     4056   74266.12      5000
```

Tabulate the number of persons in each state at a set of times. 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
   1.93    0.03    1.96
```

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

```
        State
when       DM Ins Dead Dead(Ins)
  1995   5000   0    0         0
  1995.2 4951  21   28         0
  1995.4 4886  58   55         1
  1995.6 4809  99   88         4
  1995.8 4742 129  123         6
  1996   4687 162  144         7
  1996.2 4632 187  173         8
  1996.4 4568 219  203        10
  1996.6 4508 251  230        11
  1996.8 4406 304  274        16
```

Show the cumulative prevalences in a different order than that of the state-level ordering and plot them

```
> 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 0.9902 0.9944    0.9944    1
  1995.4 0.9772 0.9888    0.9890    1
  1995.6 0.9618 0.9816    0.9824    1
  1995.8 0.9484 0.9742    0.9754    1
  1996   0.9374 0.9698    0.9712    1
```

```
> plot( pp )
```

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

```
> clr <- c("orange2","forestgreen")
> par( las=1 )
> plot( pp, col=clr[c(2,1,1,2)] )
> lines( as.numeric(rownames(pp)), pp[,2], lwd=3 )
> mtext( "60 year old male, diagnosed 1990", side=3, line=2.5, adj=0 )
> 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 )
```

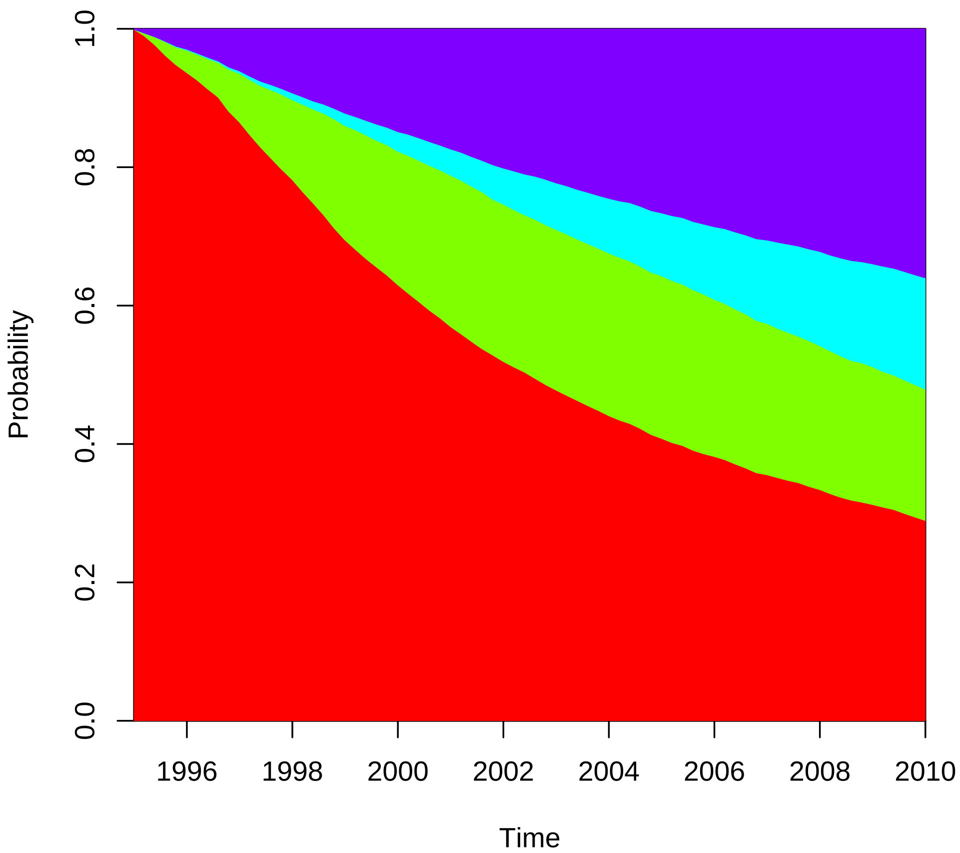

Figure 2: *Default layout of the* `plot.pState` *graph.*

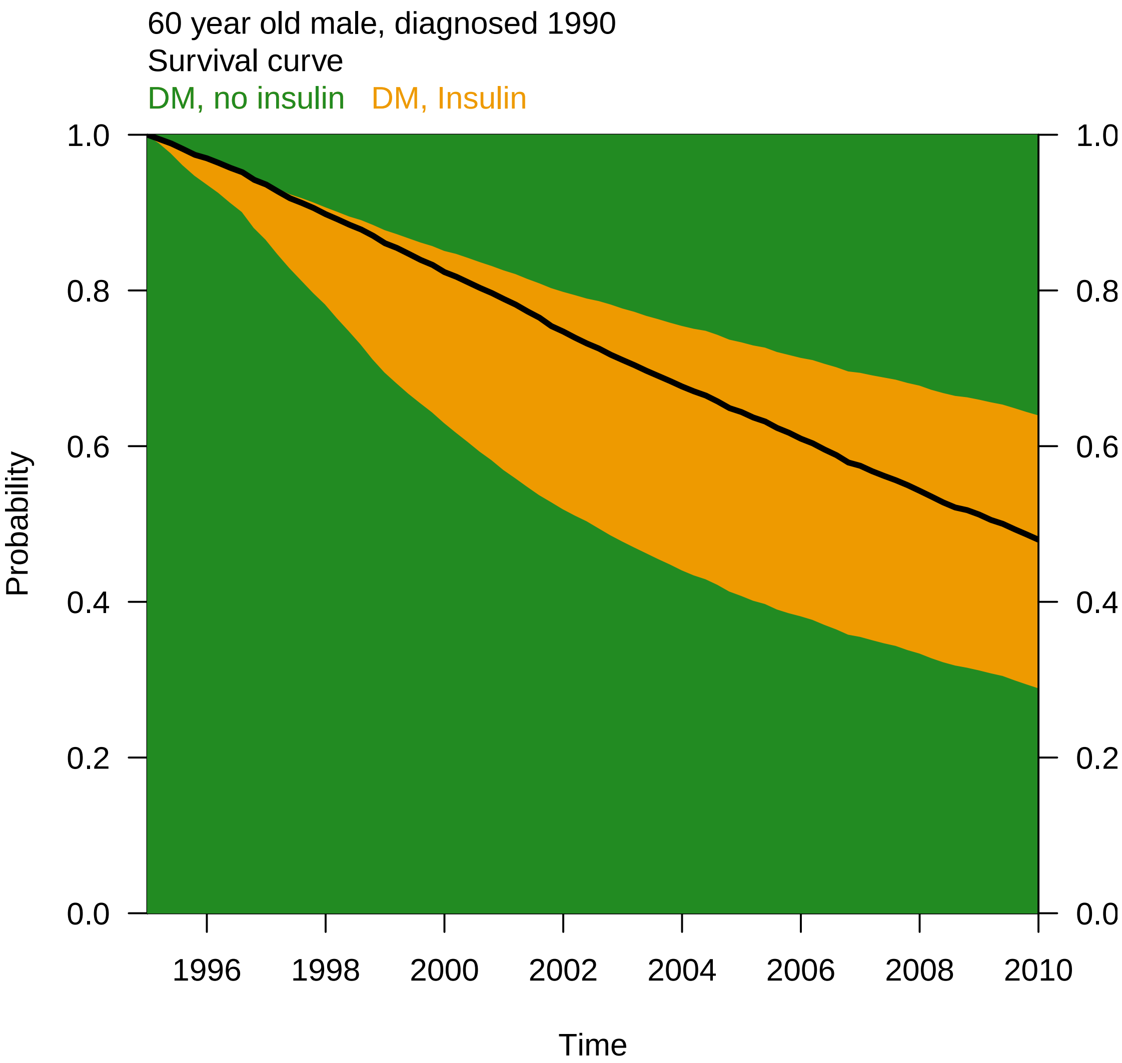

Figure 3: *A* `plot.pState` *graph where persons ever on insulin are given in orange and persons never on insulin in green, and the overall survival (dead over the line) as a black line.*