Example of fitting panel data with a latent continuous time Markov chain

We are using simulated data from a two-state reversible disease model with disease state space $\{A, B\}$. The sojourn times in states A and B are generated, respectively, according to Weibull(shape=1.5 and scale=1) and Weibull(shape=.75, scale=10) distributions. There are 200 independent individuals in the sample. For each individual, there are two baseline covariates: a binary variable X (half of the sample have X=0, and half have X=1) and a continous variable Y (generated independently from Normal(0,1)). The covariate X affects occupancy times, such that sojourns in state A are multiplied by 1.5 and in state, B, by .75. The state initially occupied by an individual is governed by an initial probability distribution that is related to the covariate Y. Those with Y=0 occupy state B with probability .094. An increase in Y by one unit increases the odds of occupying state B by 2.55. Disease states A and B are observed with measurement error. The probability of observing B given A is .028 in those with X=0, and the odds of missification increase by a factor of 1.306 in individuals with X=1. The probability of observing B given A is .05, and this probability is not affected by covariates.

There are 200 independent individuals in the data set, each observed (approximately) at $0, 1, 2, \ldots, 10$. Thus the data consist of panel observations with possible misclassification error. Data are stored in a list, with an entry for every subject. The list elements include the entries "obs.data" and "obs.times" correpsponding the observed data at each of the discrete observation times. Observations of state A are coded by 1 and B, by 2.

```
> the.data[[1]]$obs.data

 [1] 2 2 2 2 2 1 2 2 2 2 2

> the.data[[1]]$obs.times

 [1] 0.000000 1.488500 2.112423 2.969151 3.860416 4.960169 5.622864 7.234300
 [9] 8.186855 9.080619 9.672877
```

This data-generating model does not have an absorbing state, or any known times to absorption. However, if the disease model had an absorbing state (such as death) and individuals had known absorption times, it is necessary to specify this an the individual's data entry. Suppose subject had a known time of absorption occuring at the 10th observation time. We specify this by setting

```
> the.data[[1]]$exact.times.ranks=10.
```

The baseline covariate data is in a data frame called cov.data. Each subject is identified by an id in the same order as the.data list.

```
> cov.data[1:10,]

   id X          Y
1   1 1 -0.5982105
2   2 1  1.5467532
3   3 1  0.7701050
4   4 1  0.2851026
5   5 1  2.2890916
6   6 1  1.6181584
7   7 1  0.3472898
8   8 1  0.2726347
9   9 1 -0.4469496
10 10 1 -1.1877430
```

1 Specifying the model

We will fit the data with latent CTMC model that has a total of 4 latent states, 2 per each disease state. The latent state space is $S = \{1, 2, 3, 4\}$ where states $\{1, 2\}$ correspond to disease state A and states $\{3, 4\}$ to disease state B. The latent CTMC has transition intensity matrix Λ. The model is characterized by Coxian phase type sojourn distributions for times spent in the disease states, and the latent structure has transitions implied by figure 1. The Λ matrix that corresponds to this structure is given by

$$\Lambda = \begin{bmatrix} -(\lambda_{12} + \lambda_{13}) & \lambda_{12} & \lambda_{13} & 0 \\ 0 & -\lambda_{23} & \lambda_{23} & 0 \\ \lambda_{31} & 0 & -(\lambda_{31} + \lambda_{34}) & \lambda_{34} \\ \lambda_{41} & 0 & 0 & -\lambda_{41} \end{bmatrix}.$$

We incorporate covariates in the transition matrix model as follows. Since the covariate X scales multiplicatively the sojourn time distributions in A and B, we can assume that the covariate effect of X is the same for $\lambda_{12}, \lambda_{13}, \lambda_{23}$ and likewise for $\lambda_{34}, \lambda_{31}, \lambda_{41}$. The model is specified in terms of log-rates and has parameters $\{r_1, \ldots, r_8\}$ as follows:

$$\begin{aligned} log(\lambda_{12}) &= r_1 + r_7 X \\ log(\lambda_{13}) &= r_2 + r_7 X \\ log(\lambda_{23}) &= r_3 + r_7 X \\ log(\lambda_{34}) &= r_4 + r_8 X \\ log(\lambda_{31}) &= r_5 + r_8 X \\ log(\lambda_{41}) &= r_6 + r_8 X. \end{aligned}$$

Specifying the rate matrix model requires that we have a list called rates.setup. The list as several elements, which need to be set by the user before fitting the EM. We specify the number of latent states

```
> rates.setup$num.states
```

```
[1] 4
```

We specify the non-zero transitions between the latent states with a matrix

```
> rates.setup$transition.codes
```

```
  ni nj
1  1  2
2  1  3
3  2  3
4  3  4
5  3  1
6  4  1
```

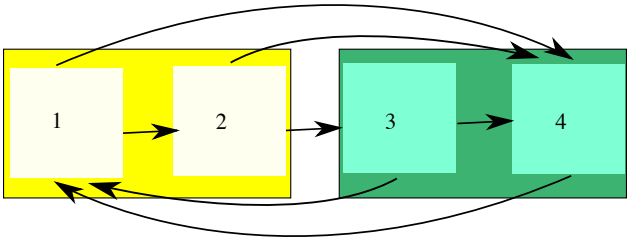


Figure 1: Latent continuous time model for two state disease process. Latent states $\{1, 2\}$ map to disease state A and $\{3, 4\}$ to disease state 2. The transitions between latent states in each disease state have Coxian phase type structure, in at each transition, it is only possible to go forward or exit to the other disease state.

We specify the design matrix for the model with the matrix

```
> rates.setup$design.matrix
```

```
  r1 r2 r3 r4 r5 r6 r7 r8
1  1 NA NA NA NA NA  X
2 NA  1 NA NA NA NA  X
3 NA NA  1 NA NA NA  X
4 NA NA NA  1 NA NA     X
5 NA NA NA NA  1 NA     X
6 NA NA NA NA NA  1     X
```

Each row in the design matrix corresponds to a single $i \rightarrow j$ transition in the order specified by the `rates.setup$transition.codes`. Intercepts are represented by 1 entries. Named covariates are specified with the name of the covariate (here, X). The setup allows us to easily specify that parameters are common across multiple transitions, e.g. r_7 and r_8.

We specify intial values for $r_1, \ldots r_8$ with

```
> rates.setup$param.values
```

```
[1] -0.2145146  0.6586720 -0.2011180  0.4489015  0.1117152 -0.1633532 -0.5802509
[8]  0.2348132
```

We also need to specify for each parameter whether it is fixed or will be estimated. if it is fixed, its value is that designated in `rates.setup$param.values`. If we set param.types=0, then we are going to estimate it, if 1, it is fixed. In this example we will estimating all parameters.

```
> rates.setup$param.types
```

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

1.1 Other apsects of rates.setup

We have now specified the model. However, to run the EM, we need to add a few more components to rates.setup. First, we want to generate an an array with a separate design matrix for each individual with entries filled in with that individual's covariate values. First we need to designate which of the entries in `rates.setup$design.matrix` that vary across individuals (rather than intercepts).

```
> rates.setup$variable.predictors
```

```
  i j
1 1 7
2 2 7
3 3 7
4 4 8
5 5 8
6 6 8
```

Then we need to create the covariate array.

```
> rates.setup$covariate.array=get.covariate.array(design.matrix,cov.data,rates.setup$variable.predictors
```

```
[1] 1 1
[1] 2 2
[1] 3 3
[1] 4 4
[1] 5 5
[1] 6 6
```

```
> rates.setup$covariate.array[,,1]
```

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    0    0    0    0    0
[2,]    0    1    0    0    0    0
[3,]    0    0    1    0    0    0
[4,]    0    0    0    1    0    0
[5,]    0    0    0    0    1    0
[6,]    0    0    0    0    0    1
[7,]    1    1    1    0    0    0
[8,]    0    0    0    1    1    1
```

Next we need to create a version of the covariate array that is limited to the parameters that we are estimating, not those that are fixed. In the case that we are estimating all parameters, the deriv.array and the covariate.array are the same.

```
> rates.setup$deriv.array=get.deriv.array(rates.setup$covariate.array, rates.setup$param.types)
> rates.setup$deriv.array[,,1]
```

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    0    0    0    0    0
[2,]    0    1    0    0    0    0
[3,]    0    0    1    0    0    0
[4,]    0    0    0    1    0    0
[5,]    0    0    0    0    1    0
[6,]    0    0    0    0    0    1
[7,]    1    1    1    0    0    0
[8,]    0    0    0    1    1    1
```

1.2 Fixed rate matrix

For certain models, we may want to specify a fixed rate matrix, with no unknown entries. In this case, we have the option to specify by setting the list item

```
> rates.setup$fixed.rates
```

```
NULL
```

to be equal to the fixed rate matrix for all individuals. This option may be most useful in the debugging stage. Note that all of the entries in the fixed rate matrix are on the intensity, not log-intensity scale.

2 Initial distribution model

The model for the initial distribution of the latent CTMC is specified by a multinomial logit model. For our model, the initial distribution is $(\pi_1, ..\pi_4)$. According to the Coxian specification, an individual allows starts in the first latent state of the corresponding disease state. So, for our model, an individual either starts in latent state 1 or latent state 3. The probability of the initial state occupancy is related to the covariate Y. With state 1 as the reference state, the model is

$$\log \frac{\pi_3}{\pi_1} = b1 + b2Y.$$

To specify the initial distribution model, we have an `init.setup` list. We have an entry for the number of states in the model.

```
> init.setup$num.states
```

```
[1] 4
```

We list the reference state with the entry

```
> init.setup$ref
```

```
[1] 1
```

We then specify all of the states, other than the reference state, that can be occupied at the initial time point. States that cannot be occupied are not listed.

```
> init.setup$states
```

```
[1] 3
```

We then specify the design matrix for the initial distribution in a way that is analogous to the rate setup.

```
> init.setup$design.matrix
```

```
  b1 b2
1  1  Y
```

We specify which of the parameters are fixed versus unknown. The code is 0 for unknown, 1 for fixed.

```
> init.setup$param.types
```

```
[1] 0 0
```

We specify the initial values of the parameters, or their value if they are fixed in the model.

```
> init.setup$param.values
```

```
[1] -2.265745  0.937000
```

To run the EM we also need to get `init.setupu$covariate.array` and `\init.setup$deriv.array` for the initial distribution. We need first to specify the entries of the design matrix that correspond to covariates.

```
> init.setup$variable.predictors
```

```
   col_names
    i j
  1 1 2
```

Then we get the `init.setup$covariate.array` and `init.setup$deriv.array`.

```
> init.setup$covariate.array=get.covariate.array(init.setup$design.matrix,cov.data,variable.entries=init
```

```
[1] 1 1
```

```
> init.setup$deriv.array=get.deriv.array(init.setup$covariate.array, init.setup$param.types)
```

Alternatively, we can specify the initial distiribution as a fixed value for everyone. For example, assume that we knew that the individual started in latent state 1. Then we would specify

```
> init.setup$fixed.dist=c(1,0,0,0)
```

3 Setting up the model for the Emission distribution

The emission matrix $E = \{E[i,j]\}$ has entries $E[i,j] = P(O(t) = j|X(t) = i$. The missclassification probably of state B given A is related to the covariate X. The emission distributions are specified according to mulitnomial logistic regression models. That is,

$$log(\frac{E[1,2]}{E[1,1]}) = g_1 + g_2X$$
$$log(\frac{E[2,2]}{E[2,1]}) = g_1 + g_2X$$
$$log(\frac{E[3,1]}{E[3,2]}) = g_3$$
$$log(\frac{E[4,1]}{E[4,2]}) = g_3$$

The rows are repated since we assume that latent states corresponding to a given disease state yield the same misclassification probability. The emission matrix model is specified with the emission.setup. First we specify the entries of the matrix with non-zero emission probability.

```
> emission.setup$emission.states
```

```
  i j
1 1 2
2 2 2
3 3 1
4 4 1
```

Then we specify the reference emisssion probabilities that for the multinomial logit models.

```
> emission.setup$ref.states
```

```
  i j
1 1 1
2 2 1
3 3 2
4 4 2
```

The design.matrix encodes the parameterization of the model. The rows correspond to rows of emission.states.

```
> emission.setup$design.matrix
```

```
  g1 g2 g3
1  1  X NA
2  1  X NA
3 NA     1
4 NA     1
```

We set set the initial values of the parameters, or the values of the parameters if they are fixed.

```
> emission.setup$param.values
```

```
[1] -3.547151  1.306000 -2.944439
```

We specify which of the parameters will be estimated (0) and which are fixed (1).

```
> emission.setup$fixed.params
```

```
NULL
```

We get the covariate.array and the deriv.array.

```
> emission.setup$covariate.array=get.covariate.array(emission.setup$design.matrix,cov.data=cov.data,emis

[1] 1 1
[1] 2 1
[1] 3 3
[1] 4 3

> emission.setup$deriv.array=get.deriv.array(emission.setup$covariate.array, emission.setup$param.types)
```

We also have the option of specifying entries in the emission matrix that are observed without error. For example, if $E[1,1] = 1$ we set

```
> emission.setup$exact.states=matrix(c(1,1),nrow=1,dimnames=list(1,c("i","j")))
> emission.setup$exact.states

  i j
1 1 1
```

Finally, we can alternatively specify if a known emission matrix that is the same for all individuals, we can specify this by setting

```
> emission.setup$fixed.dist

NULL
```

as a matrix equal to the emission distribution. This option is useful if there is no misclassification error, as entries of the emission matrix are either 0 or 1.

4 Running the EM

Now we run the EM algorithm to get parameter estimates. Arguments to the EM include the setup objects for the rate matrix, initial distribution, and emission distribution. We also need to specify the the number of sujbects, the number of latent states, number of observed states, the tolerance for convergence, and the maximum number of EM iterations. If one of the states in the latent model corresponds to an absorbing state, we specify that here. If there is no absorbing state, we set the value to "null". The object returned by the EM algorithm is a list that contains the parameter estimates, the LL at each interation, the runtime, and other information.

```
fit.4state=EM(rates.setup=rates.setup,
   init.setup=init.setup,
   emission.setup=emission.setup,
   the.data=the.data,
   num.subjects=length(the.data),
   num.states=4,
   num.obs.states=2,
   tol=1e-7,
   absorb.state=NULL,
   maxiter=500)

> fit.4state$LL

[1] -927.942
```

```
> fit.4state$param

         r          r          r          r          r          r          r
 0.7646372 -1.6795936  0.8461781 -2.6481639 -1.6541670 -3.8273887 -0.3550389
         r          e          e          e          i          i
-0.2508152 -4.2244855  1.8024719 -3.1790706 -2.3668581  0.7376466

> fit.4state$time

   user  system elapsed
225.753   8.273 234.345
```

Note that the rate parameters are listed first, followed by emission parameters, followed by initial distribution parameters. Next we need to get the information/variance of the parameter estimates.

5 Variance of parameter estimates

```
var4state=get_variance(the.data=the.data,
                       num.subjects=length(the.data),
                       num.states=4,
                       num.obs.states=2,
                       rate.param.values=fit.4state$params[1:8],
                       emission.param.values=fit.4state$params[9:11],
                       init.param.values=fit.4state$params[12:13],
                       absorb.state=NULL,
                       rates.setup=rates.setup,
                       emission.setup=emission.setup,
                       init.setup=init.setup)
```

The function actually provides the information of the estimates; to get the variance, we need to take the inverse. Diagonal entries of the variance matrix corresond to rate, emission, and initial distribution parameters according to the order they were specified in the repsective design matrices. Any parameter that was set to be fixed will lack an entry in the information matrix. The variance of the parameter estiamtes enables us to get 95% confidence intervals of the parmaeter esitmates based on normal approximation of their sampling distribution. Sometimes the information matrix will not be of full rank. This usually happens as a result of over-parameterization in the model, or lack of identifiability of model parameters. Those concerned should consider simplifying the model or setting certain elements to be fixed. Even in the absence of a non-invertible information matrix, it is usually still possible to get estimates for point-wise standard errors of disease process functionals based on the peudo-inverse of the information matrix and the delta method formulae.

6 Estimating hazard and survival functions

The disease process can be described in terms of the distriution of sojourn times in states A and B, as well as the hazard rates for leaving these states. We will esitmate these quantities for X=0 and X=1. We can obtain delta-method based standard errors using the covariance matrix for the parameters. First we obtain the covariance matrix that corresponds to the rate parameters with a pseudo inverse function from the package "corpcor."

```
> #Get functional estimates for covariates
> library(corpcor)
> out=fit.4state
> covar=pseudoinverse(var4state$information$out)[1:8,1:8]
> covar
```

```
             [,1]        [,2]        [,3]        [,4]        [,5]        [,6]
[1,]  0.703099216 -3.51160182 -0.13885808  0.07436775  0.01895133  0.10640330
[2,] -3.511601821 28.79437915 -0.65876131 -0.25605253  0.08971177 -0.79970037
[3,] -0.138858079 -0.65876131  0.24657047  0.04533291  0.02520268  0.02645218
[4,]  0.074367751 -0.25605253  0.04533291  2.07592928  0.24853752  5.27140466
[5,]  0.018951331  0.08971177  0.02520268  0.24853752  0.13836661  0.16898495
[6,]  0.106403295 -0.79970037  0.02645218  5.27140466  0.16898495 20.86490291
[7,] -0.006104228 -0.07402601 -0.01212376 -0.02100850 -0.02196070  0.04369474
[8,] -0.017924074 -0.05485319 -0.02121112 -0.10065230 -0.08958420  0.17906204
             [,7]        [,8]
[1,] -0.006104228 -0.01792407
[2,] -0.074026014 -0.05485319
[3,] -0.012123757 -0.02121112
[4,] -0.021008496 -0.10065230
[5,] -0.021960703 -0.08958420
[6,]  0.043694738  0.17906204
[7,]  0.026513584  0.03779318
[8,]  0.037793180  0.14185119
```

Then we point estimates for the rate matrix for all of the individuals in the dataset. The first and last entry correspond to individuals with X=1 and X=0.

```
> rates.list=get.rate.matrix.list(out$params[1:8],rates.setup)
> rates.list[[1]]

            [,1]      [,2]       [,3]        [,4]
[1,] -1.63694108  1.506213  0.1307285  0.00000000
[2,]  0.00000000 -1.634177  1.6341768  0.00000000
[3,]  0.14882529  0.000000 -0.2039047  0.05507942
[4,]  0.01693786  0.000000  0.0000000 -0.01693786

> rates.list[[200]]

            [,1]      [,2]       [,3]        [,4]
[1,] -2.33466456  2.148215  0.1864497  0.00000000
[2,]  0.00000000 -2.330722  2.3307221  0.00000000
[3,]  0.19125130  0.000000 -0.2620324  0.07078106
[4,]  0.02176638  0.000000  0.0000000 -0.02176638
```

The hazard and CDF functions have other arguments, which we set here.

```
> num.transitions=dim(rates.setup$transition.codes)[1]
> num.params=8
> transitions=rates.setup$transition.codes
> num.states=4
```

In general get the hazard and CDFs of sojourn times, we can't just use the rate matrices as is. Consider the first passage time to state B starting in state A. We treat B as an absorbing state. This process is governed by a rate matrix that has rows set to zero for latent states in B.

```
> ######################################################################
> #A to B CDF
> #coviarate X=1
> rate.firstpassage.AB=rates.list[[1]]
> rate.firstpassage.AB[3:4,]=0
> rate.firstpassage.AB
```

```
          [,1]      [,2]      [,3] [,4]
[1,] -1.636941  1.506213 0.1307285    0
[2,]  0.000000 -1.634177 1.6341768    0
[3,]  0.000000  0.000000 0.0000000    0
[4,]  0.000000  0.000000 0.0000000    0
```

First we will get the CDF of sojourn time in state A/first passage time to state B. The alpha argument specifies the initial distribution (we assume we are starting in latent state 1). the "states" argument indicates the latent end state of interest. We will start the calculatin at time=.01 and end at time=10.

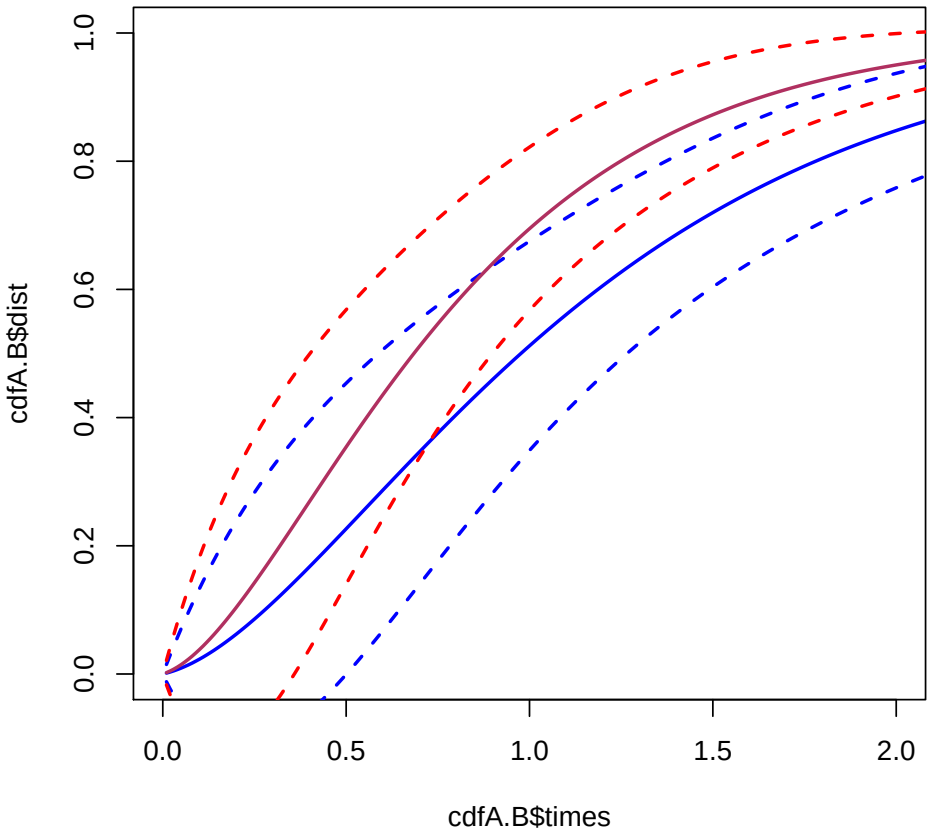
```
> cdfA.B=sub_dist_times(start=.01,end=10,states=3,alpha=c(1,0,0,0),rate.firstpassage.AB)
```

Next we get the standard errors for the cdf and 95% pointwise CIs.

```
> param.deriv=rates.setup$deriv.array[,,1]
> se=se.dist.times(start=.01,end=10,covar[1:8,1:8],rate.firstpassage.AB,states=3,
+                  alpha=c(1,0,0,0), param.deriv,num.transitions,num.params,transitions,
+                  num.states,length.out=1000)
> ci_high1=1.96*se+cdfA.B$dist
> ci_low1=-1.96*se+cdfA.B$dist
```

We will get the same CDF and confidence intervals for individuals with X=1 and plot the CDF along with the confidence intervals. Finally, we plot the the CDF along with the confidence intervals.

```
> plot(cdfA.B$times,cdfA.B$dist,type="l",col="blue",lty=1,lwd=2,ylim=c(0,1),xlim=c(0,2))
> lines(cdfA.B$times,ci_low1,type="l",lwd=2,col="blue",lty=2)
> lines(cdfA.B$times,ci_high1,type="l",lwd=2,col="blue",lty=2)
> #covariate X=0
> param.deriv=rates.setup$deriv.array[,,200]
> rate.firstpassage.AB=rates.list[[200]]
> rate.firstpassage.AB[3:4,]=0
> cdfA.B=sub_dist_times(start=.01,end=10,states=3,alpha=c(1,0,0,0),rate.firstpassage.AB)
> se2=se.dist.times(start=.01,end=10,covar[1:8,1:8],rate.firstpassage.AB,states=3,
+                   alpha=c(1,0,0,0), param.deriv,num.transitions,num.params,
+                   transitions,num.states,length.out=1000)
> ci_high2=1.96*se2+cdfA.B$dist
> ci_low2=-1.96*se2+cdfA.B$dist
> lines(cdfA.B$times,cdfA.B$dist,type="l",col="maroon",lty=1,lwd=2,ylim=c(0,1),xlim=c(0,2))
> lines(cdfA.B$times,ci_low2,type="l",lwd=2,col="red",lty=2)
> lines(cdfA.B$times,ci_high2,type="l",lwd=2,col="red",lty=2)
```

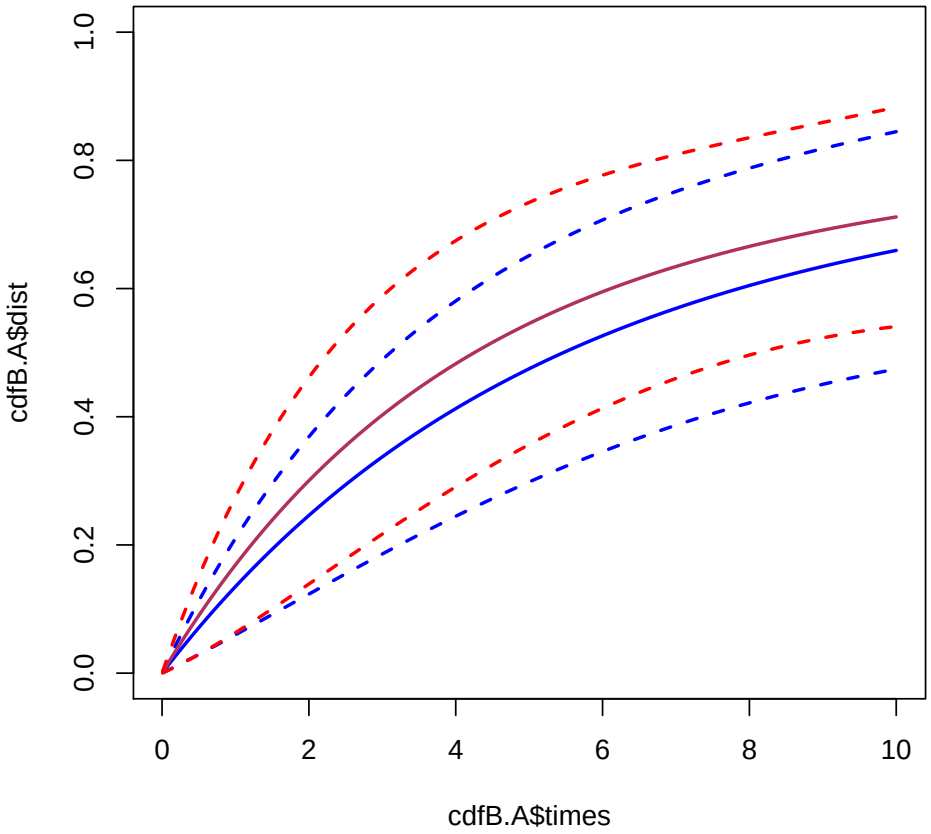


We note that the red lines are shifted left relative to the blue lines. This makes sense given that the sojourn times in state A were scaled up by 1.5 for individuals with X=1 (red).

Next we obtain the CDF for sojourn times in state B, using a similar approach. Here we notice that the the CDF for indviduals with X=1 (red) is shifted right relative to that corresponding to X=0 (blue). This makes sense given that the sojourn times in state B were scaled by a factor of .75 for individuals with X=1. The confidence intervals are broadly overlapping, however.

```
> ################################################
> #B to A CDF
> ################################################
> #covariate  X=1
> param.deriv=rates.setup$deriv.array[,,1]
> rate.firstpassage.BA=rates.list[[1]]
> rate.firstpassage.BA[1:2,]=0
> cdfB.A=sub_dist_times(start=.01,end=10,1,alpha=c(0,0,1,0),rate.firstpassage.BA)
> se3=se.dist.times(start=.01,end=10,covar,rate.firstpassage.BA,states=1,
+                   alpha=c(0,0,1,0), param.deriv,num.transitions,num.params,
+                   transitions,num.states,length.out=1000)
> ci_high3=1.96*se3+cdfB.A$dist
> ci_low3=-1.96*se3+cdfB.A$dist
> plot(cdfB.A$times,cdfB.A$dist,type="l",col="blue",lty=1,lwd=2,ylim=c(0,1))
> lines(cdfB.A$times,ci_low3,type="l",lwd=2,col="blue",lty=2)
> lines(cdfB.A$times,ci_high3,type="l",lwd=2,col="blue",lty=2)
```

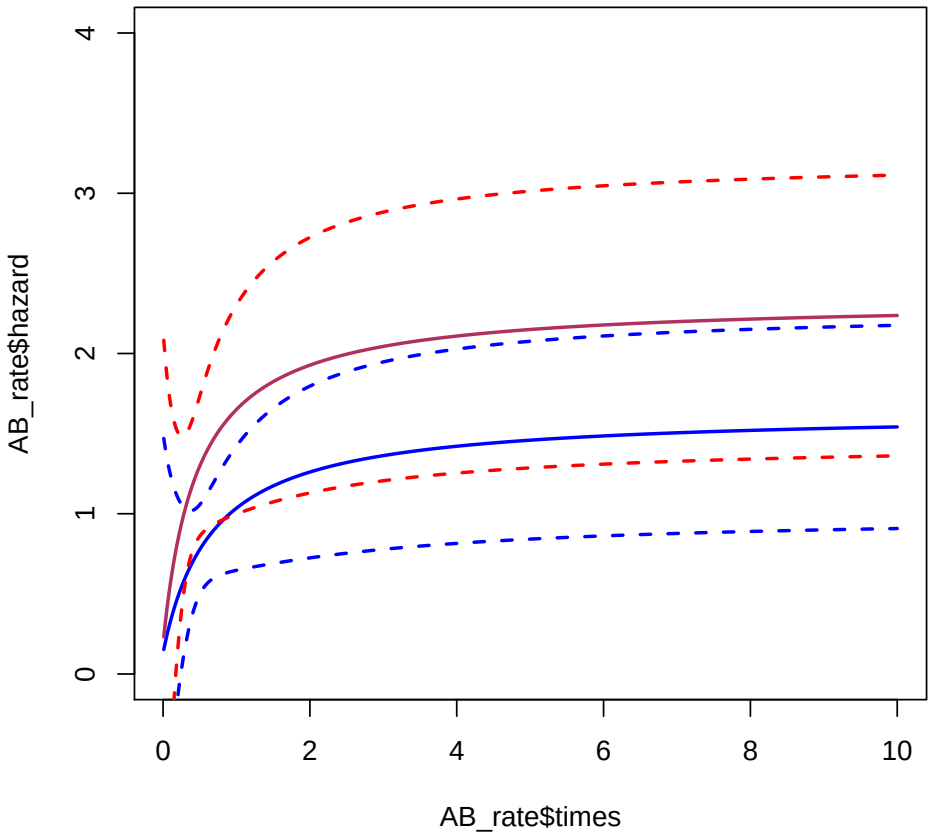
```
> #covariate X=0
> param.deriv=rates.setup$deriv.array[,,200]
> rate.firstpassage.BA=rates.list[[200]]
> rate.firstpassage.BA[1:2,]=0
> cdfB.A=sub_dist_times(start=.01,end=10,1,alpha=c(0,0,1,0),rate.firstpassage.BA)
> se4=se.dist.times(start=.01,end=10,covar,rate.firstpassage.BA,states=1,
+                   alpha=c(0,0,1,0), param.deriv,num.transitions,
+                   num.params,transitions,num.states,length.out=1000)
> ci_high4=1.96*se4+cdfB.A$dist
> ci_low4=-1.96*se4+cdfB.A$dist
> lines(cdfB.A$times,cdfB.A$dist,type="l",col="maroon",lty=1,lwd=2,ylim=c(0,1))
> lines(cdfB.A$times,ci_low4,type="l",lwd=2,col="red",lty=2)
> lines(cdfB.A$times,ci_high4,type="l",lwd=2,col="red",lty=2)
```



Hazard rates are calculated with a similar approach.

```
> ########################################################################
> #AB hazard rates
> #covariate X=1
> param.deriv=rates.setup$deriv.array[,,1]
> rate.firstpassage.AB=rates.list[[1]]
> rate.firstpassage.AB[3:4,]=0
> AB_rate=hazard_times(start=.01,end=10,state_of_interest=3,at_risk_states=c(1,2),
+                      alpha=c(1,0,0,0),rate=rate.firstpassage.AB)
```

```
> se5=se.haz.times(start=.01,end=10,covar,rate.firstpassage.AB,at_risk_states=c(1,2),
+                  state_of_interest=3,alpha=c(1,0,0,0), param.deriv,num.transitions,
+                  num.params,transitions,num.states,length.out=1000)
> ci_high5=1.96*se5+AB_rate$hazard
> ci_low5=-1.96*se5+AB_rate$hazard
> plot(AB_rate$times,AB_rate$hazard,type="l",lwd=2,col="blue",ylim=c(0,4))
> lines(seq(.01,10,length.out=1000),ci_low5,type="l",lwd=2,col="blue",lty=2)
> lines(seq(.01,10,length.out=1000),ci_high5,type="l",lwd=2,col="blue",lty=2)
> #covariate X=0
> param.deriv=rates.setup$deriv.array[,,200]
> rate.firstpassage.AB=rates.list[[200]]
> rate.firstpassage.AB[3:4,]=0
> AB_rate=hazard_times(start=.01,end=10,state_of_interest=3,at_risk_states=c(1,2),
+                      alpha=c(1,0,0,0),rate=rate.firstpassage.AB)
> se6=se.haz.times(start=.01,end=10,covar,rate.firstpassage.AB,at_risk_states=c(1,2),
+                  state_of_interest=3,
+                  alpha=c(1,0,0,0), param.deriv,num.transitions,
+                  num.params,transitions,num.states,length.out=1000)
> ci_high6=1.96*se6+AB_rate$hazard
> ci_low6=-1.96*se6+AB_rate$hazard
> lines(AB_rate$times,AB_rate$hazard,type="l",lwd=2,col="maroon",ylim=c(0,4))
> lines(seq(.01,10,length.out=1000),ci_low6,type="l",lwd=2,col="red",lty=2)
> lines(seq(.01,10,length.out=1000),ci_high6,type="l",lwd=2,col="red",lty=2)
```




```
> ########################################################################
> #BA hazard rates
> #covariate X=1
> param.deriv=rates.setup$deriv.array[,,1]
> rate.firstpassage.BA=rates.list[[1]]
> rate.firstpassage.BA[1:2,]=0
> BA_rate=hazard_times(start=.01,end=10,state_of_interest=1,at_risk_states=c(3,4),
+                      alpha=c(0,0,1,0),rate=rate.firstpassage.BA)
> se7=se.haz.times(start=.01,end=10,covar,rate.firstpassage.BA,at_risk_states=c(3,4),
+                  state_of_interest=1,
+                  alpha=c(0,0,1,0), param.deriv,num.transitions,num.params,
+                  transitions,num.states,length.out=1000)
> ci_high7=1.96*se7+BA_rate$hazard
> ci_low7=-1.96*se7+BA_rate$hazard
> plot(BA_rate$times,BA_rate$hazard,type="l",lwd=2,col="blue",ylim=c(0,.4))
> lines(seq(.01,10,length.out=1000),ci_low7,type="l",lwd=2,col="blue",lty=2)
> lines(seq(.01,10,length.out=1000),ci_high7,type="l",lwd=2,col="blue",lty=2)
> #covariate X=0
> param.deriv=rates.setup$deriv.array[,,200]
> rate.firstpassage.BA=rates.list[[200]]
> rate.firstpassage.BA[1:2,]=0
> BA_rate=hazard_times(start=.01,end=10,state_of_interest=1,at_risk_states=c(3,4),
```

```
+                       alpha=c(0,0,1,0),rate=rate.firstpassage.BA)
> se8=se.haz.times(start=.01,end=10,covar,rate.firstpassage.BA,at_risk_states=c(3,4),
+                  state_of_interest=1,
+                  alpha=c(0,0,1,0), param.deriv,num.transitions,num.params,
+                  transitions,num.states,length.out=1000)
> ci_high8=1.96*se8+BA_rate$hazard
> ci_low8=-1.96*se8+BA_rate$hazard
> lines(BA_rate$times,BA_rate$hazard,type="l",lwd=2,col="maroon",ylim=c(0,.4))
> lines(seq(.01,10,length.out=1000),ci_low8,type="l",lwd=2,col="red",lty=2)
> lines(seq(.01,10,length.out=1000),ci_high8,type="l",lwd=2,col="red",lty=2)
> ##########################################################################
```