Phase-type inference on competing risks models with covariates, using MCMC methods

Christoffer Haug Laache

Master of Science in Physics and Mathematics
Submission date: June 2014
Supervisor: Bo Henry Lindqvist, MATH

Norwegian University of Science and Technology
Department of Mathematical Sciences

Preface

The work with this text has been the product of my Master's thesis at the Norwegian University of Science and Technology (NTNU). It concludes my Master of Science in Applied Physics and Mathematics, with specialization in Industrial mathematics. The work has been carried out at the Department of Mathematical Sciences.

During this work i have received excellent supervision from Bo Lindqvist. His contributions have been invaluable.

This thesis is essentially an extension of my Master's project produced in the fall of 2013, and for completeness, much of the theory in that project has been included here.

Christoffer Haug Laache Trondheim, June 2014

Sammendrag

Målet med denne mastergradsoppgaven har vært å modellere konkurrerende risikoer innen levetidsanalyse og å utføre kovariatanalyser på datasett av denne typen. Dette har blitt gjort ved å tilpasse en Phase-type fordeling med flere absorberende tilstander for å modellere dødsårsakene, og å deretter innføre kovariater i denne modellen gjennom de absorberende Markov-intensitetene, slik at modellpåvirkningen fra kovariatene kan estimeres.

Det har blitt tatt utgangspunkt i en metode utviklet av Bladt, Gonzalez og Lauritzen for å tilpasse Phase-type fordelinger til vanlige levetidsdata. Denne metoden er en MCMC-algoritme, og fremstiller levetidsdataene som Markov-realiseringer for å kunne estimere parameterene i Phase-type modellen.
Ved å utvide antall absorberende tilstander i denne Phase-type modellen kan man på en naturlig måte modellere en situasjon med konkurrerende risikoer.

Den utviklede metoden fungerer godt på mange forskjellige typer datasett, og produserer gode estimater av blant annet sub-distribusjoner, sub-hasardrater og regresjonskoeffisientene til kovariatene. Det ser ut til at koeffisientestimatene ligner mye på estimater fra Cox-regresjon.

Abstract

The aim with this Master's thesis has been to develop a method of fitting a Phase-type model to a competing risks data set with covariates, and to approximate an underlying model such that important functionals and quantities can be estimated. To do this, the method proposed by Bladt et al. of fitting a Phase-type model to survival data, has been generalized to the competing risks setting, and this generalized method has been further extended to include covariates.

The part of the theory which involves extending the method by Bladt et al. to competing risks was mainly produced in the Master's project in the fall of 2014, and is presented in the theory part. The method is a MCMC algorithm which updates the Phase-type parameters in a Gibbs-sampler.

The results for the model without covariates show that the model is able to produce estimates for the sub-distribution functions, sub-density functions and cause-specific hazard rates in a satisfying way.

Before developing the new method in the theory part, three existing competing risks regression models have been presented. This is the Fine & Gray model, Cox regression and the model developed by Scheike and Zhang. These three models have also been used for comparison with the Phase-type model in the presentation of the results.

New theory has been developed in the sense that covariates have been introduced in the existing model. This has been done by using covariate regression in the absorbing intensities of the Phase-type model.

The results show that the model is suitable for a variety of different data sets and underlying distributions. The method manages to produce good estimates for the sub-distribution functions, sub-density functions, the covariate regression coefficients, and in many cases also for the cause-specific hazard rates. The estimates of the covariate regression coefficients are similar to the Cox regression coefficients.

Contents

1 Introduction

Survival analysis is an important branch of statistics which revolves around modeling failure times and the risks and probabilities connected to these. A failure can be defined as any suitable event, so survival analysis can be applied in a large number of industries and research fields. Two examples on important fields of applications are medicine and reliability analysis. In medicine one is mainly interested in effects of medical treatments or mortality of patients, and in reliability analysis one is often interested in duration and reliability of different systems or system components, in addition to measuring how factors might influence these quantities.

A very important aspect of survival analysis is estimating covariate influence on survival models. Covariates are non-stochastic variables which might or might not influence a survival model. There are an infinite number of possible covariates for any given model, so this is an important and general area to study in almost any application field of survival analysis.

Another important subject is the situation when there can be many different types of failures, and where the event of any failure type excludes the events of any of the others. This situation is suitably called a competing risks case, because the different failure types can be seen as events that compete in occurring first. Competing risks is a more complex situation than ordinary survival analysis, and thus it is here often more difficult to make inferences on theoretical functions and parameters.

It is well known that so called Phase-type distributions can be used to model ordinary survival distributions. A Phase-type distribution is essentially known as a survival distribution based on a Markov chain, where there exists an absorbing state corresponding to a case of failure. When this state is reached a failure has occurred, and the time until absorption represents the time until failure, so this time can be viewed as an ordinary failure time. What is less known is that general Phase-type distributions can have several absorbing states, and so these distributions are suitable for modeling competing risks cases. In fact, the theory on Phase-type distributions seems to simplify many difficult subjects surrounding competing risks.

A setting which can be used to make inferences on competing risks models, is the Bayesian framework. In this framework, estimates are not just based on information from the data, but also prior information. This framework allows many flexible estimation methods, and perhaps the most famous ones are the Markov Chain Monte Carlo methods, also called MCMC methods. These methods can be viewed as Markov chains, where the limiting distributions are distributions for the parameters to be estimated, and samples of the parameters are drawn from these to be used in producing parameter estimates. The greatest strength of MCMC methods is that they are flexible, and can be made to produce a wide range of parameter and functional estimates.

In this Master's thesis, MCMC methods are used to make inferences on Phase-type distributions which correspond to competing risks data sets. The main goal is to be able to estimate covariate influences on the theoretical competing risks models for these data sets. This is an area where there is a lot of potential for improvement. Methods of modeling covariate influences in the competing risks case do exist, but they all have their shortcomings, and can estimate only specific parameters and functions. The aim of this Master's thesis is therefore to create a MCMC method which can produce estimates of many general and important competing risks parameters and functions. It is of particular interest to be able to estimate cause-specific hazard rates when covariates influence the models, since there exists no efficient methods of doing this.

The work in this Master's thesis has been both theoretical and numerical, but the most time has been spent implementing the MCMC method in the statistical programming language *R*.

The theory developed in this text is largely an extension of the theory presented in the paper by Bladt, Gonzalez and Lauritzen [6], on developing a MCMC method to fit Phase-type models to survival data. The extension is generalizing the method in this paper to the competing risks case, by allowing more than one absorbing state in the Phase-type model. In addition, covariates are introduced into the existing model, through the absorbing intensities.

The results in this Master's thesis have been generated by a custom made MCMC method. The implementation has been done entirely in the programming language *R*, and the most important code is presented in section 7.3 in the appendix.

For completeness, the theory part of the Master's project [12], produced in the fall of 2014, has been included. This constitutes all the sub-chapters from 2.1 to 3.3 in this text. Sub-chapter 4.2.1, in the results part has also been taken partially from this project. The rest of the text has been produced in the period set for this Master's thesis, which is spring 2014.

2 Theory

2.1 Markov processes

A stochastic process can be represented as a collection of state variables, $X(t)$, dependent on a time variable, $t > 0$. Essentially, $X(t)$ is a stochastic variable that has a development in time, and we generally refer to the different values of $X(t)$ as the state of the system, at time t. The state space, Ω, can be finite or infinite. $X(t)$ could for example be the amount of CO_2 in the atmosphere, the different stages of a disease, or the amount of failures in a system.

A finite stochastic process that fulfills what we call the Markov property is a Markov process, also called a Markov chain. For a continuous-time process, the Markov property is defined by

$$P\left(X(s+t) = j | X(t) = i, X(t_n) = i_n, ..., X(t_0) = i_0\right)$$

$$= P\left(X\left(t+s\right) = j | X(t) = i\right), \tag{1}$$

when $s > 0$, $t > t_n > ... > t_0$ and j,i,i_0,...$i_n \in \Omega$.

This means that the history of the process, beyond the present state, is irrelevant for the probability distribution of the next state.

A time-homogeneous Markov process is a process where the probability of going from one state to another only depends on the length of the given time interval, and not the values of time themselves. The probability is therefore not dependent on time, but on time differences. This gives that

$$P\left(X(s+t) = j | X(s) = i\right) = P\left(X(t) = j | X(0) = i\right), \tag{2}$$

for any s,t, i and j.

Consider a time homogeneous Markov process. At time s, the state of the process is i. The probability of reaching state j at time $t+s$ is denoted as

$$P_{ij}(t) = P\left(X(t+s) = j | X(s) = i\right) = P\left(X(t) = j | X(0) = i\right). \tag{3}$$

Since the state space is finite, the probabilities of going from one state to any of the states in the state space, when the time difference is t, can be represented in a transition probability matrix,

$$P(t) = \begin{bmatrix} P_{11}(t) & P_{12}(t) & \cdots & P_{1N}(t) \\ P_{21}(t) & P_{22}(t) & \cdots & P_{2N}(t) \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ P_{N1}(t) & P_{N2}(t) & \cdots & P_{NN}(t) \end{bmatrix}. \tag{4}$$

Of course, the process must be in some state at any given time, so

$$\sum_{j=1}^{N} P_{ij}(t) = 1, \tag{5}$$

for all possible initial states, i.

An absorbing state is defined as a state in which there cannot be any transitions to other states, so the process is stuck in the absorbing state. This means that

$$P_{ij}(t) = 0 \tag{6}$$

and

$$P_{ii}(t) = 1, \tag{7}$$

when i is absorbing, $j \neq i$, and t can take any positive value.

In this text, all absorbing states will be the states with the highest indexes, such that for a process with a total of $K + m$ states and m absorbing states, the states $\{K + 1, ..., K + m\}$ are the absorbing states.

Denote the time in state i before transition to any other state as T_i.
Assume that the process has been in state i for s time units, such that

$$T_i > s.$$

The probability of staying in state i for another t time units will not be affected by this, because of the Markov property. This is because the history of the process, beyond the fact that

$$X(s) = i,$$

will not be relevant for the state distribution at any time after s.

A result of this is that

$$P\left(T_i > s + t | T_i > s\right) = P\left(T_i > t\right) \tag{8}$$

This shows that the probability distribution of T_i has no memory, and therefore it must be the exponential distribution [17] (page 294).

Important quantities in Markov theory are the transition intensities. Assume that the system is in state i. Then the transition intensity to a state j ($j \neq i$) is defined by

$$a_{ij} = \lim_{\Delta t \to 0} \frac{P\left(X(t + \Delta t) = j | X(t) = i\right)}{\Delta t} = \lim_{\Delta t \to 0} \frac{P_{ij}\left(\Delta t\right)}{\Delta t} = P'_{ij}(0) \tag{9}$$

The transition intensity from i to itself is

$$a_{ii} = P'_{ii}(0) = \left(1 - \sum_{j \neq i} P_{ij}(0)\right)' = -\sum_{j \neq i} a_{ij} \tag{10}$$

The intensities are usually represented in the transition intensity matrix.

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2N} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ a_{N1} & a_{N2} & \cdots & a_{NN} \end{bmatrix} \tag{11}$$

Because of property (10), the sum of each row in this must be 0. It is also easy to see that for an absorbing state, the whole row is zero.

For a Markov chain there might exist a distribution which the chain converges to when $t \to \infty$, and it is called the *limiting distribution.* For a certain type of Markov chains, which are called *ergodic* (Gamerman and Lopes [10] page 124), this distribution always exists. Formally, each component of this distribution is given by:

$$P_j = \lim_{t\to\infty} P_{ij}(t) \tag{12}$$

It is well known that this distribution can be found by simultaneously solving the equations

$$[P_1, P_2, ..., P_N] \cdot A = P \cdot A = 0$$

and

$$\sum_{j=1}^{N} P_j = 1. \tag{13}$$

At last, it can be proven [17] (Page 410) that the transition probability matrix, $P(t)$, can be represented as

$$P(t) = e^{At} = \sum_{i=0}^{\infty} \frac{t^i}{i!} A^i \tag{14}$$

A way to sample a complete Markov chain realization is to sample the time used in each state visit, and for each state visit sample the transition to the next state with a discrete distribution. It is well known that the time used when visiting i is exponential with rate equal to $\sum_{j\neq i} a_{ij} = -a_{ii}$. In addition, we can view the state transition from state i to state j as a draw from a discrete distribution which has probability

$$\mathbb{P}\left(\text{transition from i to j}\right) = \frac{a_{ij}}{\sum_{j\neq i} a_{ij}} = \frac{a_{ij}}{-a_{ii}} \tag{15}$$

Thus we can first draw the visiting time for a state visit in state i from the exponential distribution with rate $-a_{ii}$, and then draw the next visiting state from the discrete distribution with probabilities for each j $(j \neq i)$ determined by (15).

2.2 Competing risks

Consider a unit which has a specified failure time, T, and where the failure time has a certain type, C. Here, $C \in \{1, 2, ..., m\}$.
An observation of (T, C) gives rise to a statistical case called *competing risks*.

There are a number of ways to approach this situation. The most used is by considering latent failure times, $(T_1, T_2, ..., T_k)$, where each T_j represents the failure time of case $C = j$.
The observed failure is the smallest of these failure times, such that $T = \min_j (T_j)$ and $C = \text{argmin}_j (T_j)$.

This approach is useful but has its limitations. It turns out that the joint distribution of the T_j is not linked in a one-to-one relationship with the observation (T, C).
In fact, there can be several different joint distributions of T_j which correspond to the distribution of (T, C).
This is known as the identifiability problem. An assumption that goes around this problem is independence of the latent risks, T_j. In this case, there is only one joint distribution that corresponds to the distribution of (T, C).

Here follows a set of useful definitions and relations in the field of competing risks. These results have mainly been taken from Braarud ([8], 2012) and Lindqvist ([14], 2006).

The sub-distribution function is given by

$$F_j(t) = P(T \leq t, C = j), \tag{16}$$

where $t \geq 0$ and $j \in \{1, 2, ..., k\}$.

This gives the definition of the sub-density function,

$$f_j(t) = F_j'(t). \tag{17}$$

The sub-survival function is given by

$$R_j(t) = P(T > t, C = j) = \pi_j - F_j(t), \tag{18}$$

where π_j is called the marginal distribution of C, and is defined by

$$\pi_j = P(C = j) = \int_0^\infty f_j(t)dt = F_j(\infty). \tag{19}$$

Note that then we have

$$F_j(t) + R_j(t) = \pi_j. \tag{20}$$

The marginal distribution function of T is

$$F(t) = \sum_{j=1}^{k} F_j(t). \tag{21}$$

The sub-hazard functions, also called the cause-specific hazard rates, are given by

$$\lambda_j(t) = \lim_{\Delta t \to 0} \frac{P(T \leq t + \Delta t, C = j | T > t)}{\Delta t} = \frac{f_j(t)}{R(t)}. \tag{22}$$

Taking the derivative of (21), we see that the marginal density of T is also a sum of sub-densities, and thus the hazard function of T must be given by a sum of sub-hazard functions, according to (22);

$$\lambda(t) = \sum_{j=1}^{k} \lambda_j(t). \tag{23}$$

Integrating the sub-hazard functions gives the cumulative sub-hazard functions,

$$\Lambda_j(t) = \int_0^t \lambda_j(u)du. \tag{24}$$

2.3 Phase type distributions

In a continuous-time Markov chain with one absorbing state, the time until absorption has a distribution which is called a *Phase-type* distribution. In this model the absorbing state corresponds to a failure, and all the transient states can be viewed as different states on the way to failure. In medicine, an example of a model that can be set up this way is the different stages of the development of cancer in a patient. The transient states might here be interpreted as different stages of cancer, while the absorbing state is death.

Even so, the distributions are actually quite general, so there isn't really a need for a physical interpretation of a situation which is to be modeled. In fact, any general failure time can be approximated by a phase type distribution.

An extension to a competing risks setting can easily be made. Now there are several absorbing states, where each state corresponds to a type-specific failure. The time until failure is just the time until one of the absorbing states has been reached, and the type of failure simply corresponds to the type of absorbing state which has been reached.

To model the Markov chain of a Phase-type distribution with K transient states and m absorbing states, consider a $(K+m) \times (K+m)$ intensity matrix of the chain,

$$A = \begin{bmatrix} Q & L \\ 0_1 & 0_2 \end{bmatrix}. \tag{25}$$

Here, Q is the $K \times K$ matrix of intensities for the transient states, L is the $K \times m$ matrix of intensities from the transient and into the absorbing states, while 0_1 and 0_2 are respectively $m \times K$ and $m \times m$ matrices of zeros.

(14) gives that the transition probability matrix can be represented by

$$P(t) = e^{At} = \sum_{i=0}^{\infty} \frac{t^i}{i!} A^i, \tag{26}$$

It is quite straightforward to see that

$$A^i = \begin{bmatrix} Q^i & Q^{i-1}L \\ 0 & 0 \end{bmatrix},$$

and by definition of the matrix exponential,

$$A^0 = I_{(K+m)\times(K+m)}.$$

This can be used in (26) to get the following

$$P(t) = \sum_{i=0}^{\infty} A^i \frac{t^i}{i!} = I + \sum_{i=1}^{\infty} A^i \frac{t^i}{i!}. \tag{27}$$

A sum of matrices is an operation working componentwise, so the sums above can also be represented blockwise. The different blocks in A^i can therefore be evaluated separately in (26) and (27). $P(t)$ is denoted as

$$P(t) = \begin{bmatrix} P_1(t) & P_2(t) \\ 0 & I \end{bmatrix}. \tag{28}$$

By using (27), the upper right block of $P(t)$ can be represented as

$$P_2(t) = 0 + \sum_{i=1}^{\infty} \frac{t^i}{i!} Q^{i-1} L = \sum_{i=1}^{\infty} \frac{t^i}{i!} Q^{i-1} L,$$

because the part of I in this block is only zeros.

This further gives that

$$P_2(t) = \sum_{i=1}^{\infty} \frac{t^i}{i!} Q^{i-1} L = Q^{-1} \left(\sum_{i=0}^{\infty} \frac{t^i}{i!} Q^i - I \right) L = Q^{-1} \left(e^{Qt} - I \right) L. \tag{29}$$

Using (26) and the block representation of A^i, $P_1(t)$ can be represented as

$$P_1(t) = \sum_{i=0}^{\infty} Q^i \frac{t^i}{i!} = e^{Qt}. \tag{30}$$

Using both (29) and (30), the final transition probability matrix becomes

$$P(t) = \begin{bmatrix} e^{Qt} & Q^{-1} \left(e^{Qt} - I \right) L \\ 0 & I \end{bmatrix}. \tag{31}$$

Define $\mathbf{p}$ as the initial distribution in a Phase type model with $K+m$ states in total, K transient and m absorbing. The matrix product of $\mathbf{p}$ and $P(t)$,

$$\mathbf{p}P(t),$$

will then be a vector of probabilities for being in the different states at time t. Notice that the probability of being in an absorbing state j at time t is the same as $Pr(T \leq t, C = j)$, where T is the time to reach an absorbing state. Thus, the sub-distribution function of type j failure is just the $(K+j)$-th coordinate of this vector, such that

$$F_j(t) = (\mathbf{p}P(t))_{K+j}$$

A way to extract this component is to use the $((K+m) \times 1)$-vector

$$v_{K+j}^T = \begin{bmatrix} 0 \\ 0 \\ . \\ . \\ 1 \\ . \\ 0 \\ 0 \end{bmatrix},$$

where the unity value is placed in the $(K+j)$-th row.

The sub-distribution function then becomes

$$F_j(t) = \mathbf{p}P(t)v_{K+j}^T. \tag{32}$$

A simplification can be made concerning the dimensions of this expression. Because we are only interested in the absorbing states, the probabilities of going from the transient states to the absorbing states are the only ones we need to define the sub-distribution functions. Thus the absorbing part of the matrix $P(t)$ is the only part of $P(t)$ needed. This was defined in (29). Using only this part of $P(t)$ in (32), redefining $\mathbf{p}$ to contain only transient states, and also redefining v to contain only absorbing states, will be sufficient to represent the sub-distribution function.

The adjusted expression now becomes

$$F_j(t) = \mathbf{p}Q^{-1}\left(e^{Qt} - I\right)Lv_j^T. \tag{33}$$

Here, $\mathbf{p}$ is a $(1 \times K)$-vector, and v_j is a $(1 \times m)$-vector.

To find the sub-hazard function, given in (22), the sub-density function must be determined. It is found in a straightforward way:

$$f_j(t) = F_j^{'}(t) = \mathbf{p}\Big(Q^{-1}\left(e^{Qt} - I\right)L\Big)^{'}v_j^T = \mathbf{p}Q^{-1}Qe^{Qt}Lv_j^T = \mathbf{p}e^{Qt}Lv_j^T. \tag{34}$$

Using that $R(t) = 1 - F(t)$, it now follows from (21) and (22) that

$$\lambda_j(t) = \frac{f_j(t)}{R(t)} = \frac{f_j(t)}{1 - \sum_{j=1}^m F_j(t)} = \frac{pe^{Qt}Lv_j^T}{1 - pQ^{-1}\left(e^{Qt} - I\right)L\mathbf{1}}.$$

Since the sums of the rows of A are 0, $A\mathbf{1} = Q\mathbf{1} + L\mathbf{1} = 0$, such that,

$$Q\mathbf{1} = -L\mathbf{1}.$$

We then have that

$$\lambda_j(t) = \frac{pe^{Qt}Lv_j^T}{1 - pQ^{-1}\left(e^{Qt} - I\right)(-Q\mathbf{1})} = \frac{pe^{Qt}Lv_j^T}{1 - 1 + pe^{Qt}\mathbf{1}} = \frac{pe^{Qt}Lv_j^T}{pe^{Qt}\mathbf{1}}. \tag{35}$$

By first finding $F_j(t)$, $R_j(t)$ can also be found using (18):

$$R_j(t) = \pi_j - F_j(t). \tag{36}$$

Using (19), π_j is given as

$$\pi_j = F_j(\infty),$$

which, by using (33) can be expressed as

$$\pi_j = \lim_{t\to\infty} \mathbf{p}Q^{-1}\left(e^{Qt} - I\right) Lv_j^T. \tag{37}$$

Because e^{Qt} is the probability matrix for the transient part, according to (31), and we must be in an absorbing state when $t \to \infty$,

$$\lim_{t\to\infty} e^{Qt} = \mathbf{0},$$

which is just the zero matrix.

This gives that

$$\pi_j = -\mathbf{p}Q^{-1}Lv_j^T, \tag{38}$$

according to (37).

2.4 Markov Chain Monte Carlo Methods

2.4.1 Bayesian framework

MCMC methods are based on a Bayesian framework. In this framework, even the parameter values are random variables, so they have a joint probability distribution.

By using Bayes theorem, any probability can be decomposed into

$$P(B) = \frac{P(B|A)\,P(A)}{P(A|B)},$$

where A and B are two random events. This further gives that

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)} \propto P(A|B)P(B)$$

If event A is the event of obtaining a sample x, and event B is the event of obtaining the parameter values θ, we are essentially talking about densities, and the situation becomes

$$f(\theta|x) = \frac{f(x|\theta)f(\theta)}{f(x)} \propto f(x|\theta)f(\theta), \tag{39}$$

where $f(\theta)$ is the joint probability density to the components of the vector θ, and $f(x|\,\theta)$ is simply the likelihood function of the sample. $f(\theta|x)$ is called the posterior density, and $f(\theta)$ is called the prior density. The prior density contains our information of the parameters before any data is collected, and the posterior density contains our information after data is collected. So essentially, the sample x gives us information about θ.

MCMC methods exploit this relationship between the data and the parameter values, to be able to simulate the parameter values from the posterior distribution corresponding to the posterior density above.

2.4.2 Monte Carlo integrals

It is well known that a consistent and unbiased estimator for the integral

$$I = \int_{-\infty}^{\infty} t(\theta)\pi(\theta)d\theta, \tag{40}$$

is the estimator

$$\hat{I} = \frac{1}{n}\sum_{i=1}^{n} t(\theta_i), \tag{41}$$

when the data sample, $(\theta_1, ..., \theta_n)$, is sampled from $\pi(\theta)$, and the samples are independent of each other.

Because $\pi(\theta)$ can be very difficult to sample from, or because we might just have determined an expression proportional to this distribution, without having the normalizing constant, it is very common to use approximations or algorithms which simulate $\pi(\theta)$, in order to generate (41). This is especially true when considering posterior distributions, which often can be quite complex, and are often not completely determined. A very common way to generate samples from posterior distributions is by generating Markov chains that have the posterior distribution as a limiting distribution, given in (12). Although samples gathered from the states of a Markov chain will be dependent of each other, it can be shown that ergodic Markov chains produce limiting samples where this dependency plays an

insignificant role, such that (41) can still be used. This result is called the Ergodic theorem (Gamerman and Lopes [10] page 125).
There are two methods that are mostly used to generate such Markov chains. They are called the Metropolis-Hastings method and the Gibbs method.

2.4.3 Gibbs sampling

As given in Gamerman and Lopes [10] (page 142 and 143), a way to generate an ergodic Markov chain with a limiting distribution equal to the posterior, is by sampling iteratively from the full conditional distributions of the parameters. We represent the parameter values in vector form, $\theta = [\theta_1,\ \theta_2, \dots,\ \theta_n]$. The full conditionals are given as:

$$p\left(\theta_j \middle| \, \theta_{-j}, x\right),$$

where θ_{-j} are all the components of θ except component j, and x is the data.

For each step in the iterative procedure, the parameter values are sampled from the full conditional distributions. The distributions use estimates of the parameter values, instead of true values. The sampled values are used in the succeeding steps, so that the parameter values evolve.
For a given step, b, the iterative updating of the parameter values, within this step, might evolve as the following

$$\begin{aligned} \left(\hat{\theta}_2^b, ..., \hat{\theta}_n^b\right) &\rightarrow \left(\hat{\theta}_1^{b+1}\right) \\ \left(\hat{\theta}_1^{b+1}, \hat{\theta}_3^b, ..., \hat{\theta}_n^b\right) &\rightarrow \left(\hat{\theta}_2^{b+1}\right) \\ \vdots \qquad & \qquad \vdots \\ \left(\hat{\theta}_1^{b+1}, \hat{\theta}_2^{b+1}, ..., \hat{\theta}_{n-1}^{b+1}\right) &\rightarrow \left(\hat{\theta}_n^{b+1}\right) \end{aligned}$$

Here, the arrows symbolize using the approximate full conditionals to produce the parameter estimates pointed to, by using the parameter estimates pointed from.

This means that for each step, every parameter value is estimated by using the estimates from the previous step or, if possible, the estimates already made at the current step. The first estimates are produced using the prior distribution of the parameter values.
The order of the parameters being updated within a step is theoretically not important, but can have effects concerning the efficiency of the algorithm.

A key point to see here, is that the parameter values evolve in a Markovian manner. At each step, the estimates are only based on the information from the previous step or the current. This means that this type of stepwise estimation constitutes a Markov chain, and it is proven that the chain has properties which gives it a limiting distribution equal to the posterior distribution of the parameters (Gamerman and Lopes [10] page 147 and 148).

2.4.4 Metropolis-Hastings sampling

As given in Gamerman and Lopes [10] (chapter 6.2), another way to generate an ergodic Markov chain with a limiting distribution equal to the posterior, is by producing a reversible Markov chain of iterative estimates, which is specifically tailored with respect to this limiting distribution.

A way to do this is by proposing new updates, and accepting these updates with a certain acceptance probability. If $\theta = [\theta_1,\ \theta_2, \dots, \theta_n]$ are the present values of the parameters, and $\Phi = [\Phi_1,\ \Phi_2, \dots, \Phi_n]$ are the proposed values, then the probability of accepting the proposed parameter values is set to

$$\alpha(\theta, \Phi) = \min\left(1,\ \frac{\frac{\pi(\Phi)}{q(\theta,\Phi)}}{\frac{\pi(\theta)}{q(\Phi,\theta)}}\right) = \min\left(1,\ \frac{\pi(\Phi)q(\Phi,\theta)}{\pi(\theta)q(\theta,\Phi)}\right), \tag{42}$$

where $\pi(\theta)$ is the posterior distribution, and $q(\theta, \Phi)$ is the proposal distribution. If the proposed values are accepted, they replace the existing ones.

For an iteration, b, and an existing parameter estimate, $\hat{\theta}^b$, we thus have the following situation

$$\hat{\theta}^{b+1} = \begin{cases} \Phi & \text{with probability } \alpha \\ \hat{\theta}^b & \text{with probability } (1-\alpha) \end{cases} .$$

The initial parameter vector, θ^0, is drawn from the prior distribution.

Again, we see that the iterative estimates are updated in a Markovian manner, and because of the choice of α, it can be shown that this Markov chain has a limiting distribution equal to the posterior distribution (Gamerman an Lopes [10] page 195).

3 Constructing the basic algorithm for the case without covariates

The basic algorithm (without covariates) presented in this text is essentially a Gibbs sampler, but uses a Metropolis-Hastings method (also called MH-method) within each Gibbs step. It is designed to estimate the initial distribution, **p**, and the intensity matrix, A, in a Phase type representation for the distribution of a certain set of competing risks data, $\mathbb{x} = [(x_1, c_1), ..., (x_\mathbb{N}, c_\mathbb{N})]$, where x_i is the failure time of data point i, and c_i is the type of failure. For each Gibbs step, the MH-method is run a finite number of times for each data point, (x_i, c_i), to simulate corresponding Markov chain realizations, y_i. These realizations are then used as the data in the approximate posterior distribution of A and **p**, which is used to estimate A and **p**. A number of samples of A and **p** are taken, after convergence, and these can be used to estimate a large variety of competing risks functions and quantities.

As mentioned in the competing risks section, x_i can be viewed as the absorption time in a Phase-type representation, and c_i can be viewed as the absorption state.

The Phase-type representation corresponding to **p** and A can have any chosen number of transient states, and this is specified at the beginning of the algorithm. The choice of dimensions will often affect the quality of the estimates for the theoretical quantities, meaning that a large number of transient states will result in estimates that are closer to the underlying model corresponding to the data.

The basic algorithm developed in this chapter is a generalization of the algorithm developed by Bladt, Gonzalez and Lauritzen [6]. This algorithm was constructed for ordinary lifetime distributions, with one type of failure. The basic algorithm has been constructed for a competing risks setting, where there can be several types of failures. The generalization has been made by increasing the number of absorbing states in the Phase-type representation of the underlying lifetime distribution.

The complete algorithm can roughly be described as the following:

1 Sample prior estimates of **p** and A.

2 Run a Gibbs sampler to iteratively estimate **p** and A

  – For each Gibbs-step we first simulate likelihood information on the form of Markov chain realizations corresponding to each data point (x_i, c_i), by using a finite number of iterations from a MH-method.

3 After a finite number of Gibbs-steps, a representative sample of both posterior $\mathbf{p}$'s and A's has been produced. This can either be used to produce Monte Carlo estimates for $\mathbf{p}$ and A, which then are used to estimate preferred functions, or the functions can be calculated for every sample entry, such that Monte Carlo estimates of the functions themselves can be produced.

3.1 Estimating Markov chain realizations

A fact that complicates the whole situation of estimating the Phase-type parameters for the underlying distribution of the data, is that the data, $\mathbb{x}$, does not give full information on how the data points could have been developed as Markov chains. Information on the Markov chain needed to represent the distribution of the data as a Phase-type distribution is only given implicitly through the absorption times and the absorption states. So the data are not complete Markov chain realizations, and thus we have to simulate these realizations before estimating the Markov chain parameters, A and $\mathbf{p}$ by using Gibbs sampling.

A fact worth mentioning is that when a Phase-type distribution is used to model a lifetime, the lifetime only corresponds to a Markov chain realization up to absorption. This means that we are only interested in the time until absorption, which is completely defined by how the transient states are traversed in the Markov chain model, and which state becomes the final absorbing state.

The complete data sample of Markov chain realizations is denoted as $Y = [y_1, ..., y_{\mathrm{N}}]$, where each y_i is defined as

$$y_i = (J^{(i)}, s^{(i)}).$$

Here, $J^{(i)}$ is the vector of visited states, including the final absorbing state, $j_c^{(i)}$,

$$J^{(i)} = \left[j_0^{(i)}, j_1^{(i)}, j_2^{(i)}, ..., j_n^{(i)}, j_c^{(i)}\right],$$

and $s^{(i)}$ is the vector of sojourn times,

$$s^{(i)} = \left[s_0^{(i)}, s_1^{(i)}, s_2^{(i)}, ..., s_n^{(i)}\right].$$

$j_n^{(i)}$ is the final state before absorption, and $s_n^{(i)}$ is the sojourn time in this final transient state.

When trying to simulate realizations from a Phase-type Markov chain, we only need to sample realizations up to absorption, given the absorption times, $X = x_i$, and absorption states, $C = c_i$. Because the absorbing state is given in the data by c_i, this means that we only need to sample transient parts of the Markov chain realizations from a given Markov chain model. The probability of obtaining a particular transient part of a Markov chain realization, given a distinct data point, is expressed as

$$\begin{aligned}&\mathbb{P}\left(\{J_t,\ t < x_i\} = \{j_t,\ t < x_i\} \middle|\, X = x_i, C = c_i\right)\\ &\quad\equiv \mathbb{P}_{(x_i,c_i)}\left(\{J_t,\ t < x_i\} = \{j_t,\ t < x_i\}\right). \qquad (43)\end{aligned}$$

Here, a complete Markov chain realization is expressed as $\{J_t\}$, where J_t is the state at time t, and $\{J_t,\ t < x_i\}$ is the corresponding transient part.

If sampling from $\mathbb{P}_{(x_i,c_i)}\left(\{J_t,\ t < x_i\} = \{j_t,\ t < x_i\}\right)$ would be easy, then we could sample directly from this distribution and simulate the Markov chain realizations we need to estimate the Phase-Type parameters. Unfortunately, sampling directly from this distribution is difficult. A way to go around this problem is to sample from a less difficult proposal distribution, and accept the proposal if it is close enough to a sample from $\mathbb{P}_{(x_i,c_i)}\left(\{J_t,\ t < x_i\} = \{j_t,\ t < x_i\}\right)$. This can be implemented by making a MH-method with a suitable proposal distribution and acceptance probability. We can obtain approximated samples from $\mathbb{P}_{(x_i,c_i)}\left(\{J_t,\ t < x_i\} = \{j_t,\ t < x_i\}\right)$ if the MH-method is a Markov chain with this distribution as limiting distribution. In theory, convergence of the complete algorithm will be reached by using only one MH-iteration for each Gibbs-step. This is because the complete algorithm can be viewed as two separate MCMC algorithms, and when both are converging the whole algorithm does so too. To improve mixing, sometimes it is necessary to use a small finite number of MH-steps for each Gibbs-step, instead of one.

A much simpler distribution to sample from is

$$\begin{aligned}&\mathbb{P}\left(\{J_t,\ t < x_i\} = \{j_t,\ t < x_i\} \middle|\, X \geq x_i, C = c_i\right)\\ &\quad\equiv \mathbb{P}^*_{(x_i,c_i)}\left(\{J_t,\ t < x_i\} = \{j_t,\ t < x_i\}\right). \qquad (44)\end{aligned}$$

This is because $X \geq x_i$, so absorption here doesn't have to happen exactly at time x_i, but can also happen later.

This is the proposal distribution which will be used in the MH-method. An easy way to sample from this distribution is to start with sampling a Markov chain realization with any given time to absorption, X, by using the method described at the end of the Markov process chapter, reject this sample if $X < x_i$ or $C \neq c_i$, and accept otherwise. This *rejection procedure* is repeated until we get an accepted sample. Keep in mind that we are just interested in the part up to time x_i, $\{J_t,\ t < x_i\}$, and not the complete Markov chain realization up to absorption, $\{J_t,\ t < X\}$. Even so, this can easily be extracted from $\{J_t,\ t < X\}$ after sampling. The procedure can be summed up as

(45)

1 For data point (x_i, c_i), simulate a Markov chain realization, $\{j_t\}$, with the parameters A and $\mathbf{p}$, by simulating the change of states with the discrete distribution $\left(\frac{a_{k1}}{-a_{kk}}, \frac{a_{k2}}{-a_{kk}}, ..., \frac{a_{kK}}{-a_{kk}}\right)$ and the time in each state visit with the exponential distribution, exponential$(-a_{kk})$, where a visited state is state k. At absorption, the simulation is complete.

2 If the failure time, X, of the realization is such that $X \geq x_i$ and the last absorbing state of the realization, C, is such that $C = c_i$, accept the realization. If this is not the case, move back to 1.

3.1.1 The acceptance probability

By now, having defined a proposal distribution, the main goal is to derive the acceptance probability of this MH-method. This is by no means an easy task, and some quite rigorous calculations and observations must be made. They are all based on exactly the same principles as in the paper by Bladt, Gonzalez and Lauritzen [6], but with an extension to having several absorbing states.

First, lets define some very useful results.

According to (26) the distribution of the state at time t, J_t, is given by

$$\underline{\mathbf{p}}P(t) = \underline{\mathbf{p}}e^{At},$$

where A is the $((K+m) \times (K+m))$ intensity matrix, and $\underline{\mathbf{p}}$ is the $(1 \times (K+m))$ initial distribution.

The probability of being in state j at time t is thus

$$q_j(t) = \underline{\mathbf{p}} e^{At} \underline{v_j}^T,$$

where $\underline{v_j}$ is the $(1 \times (K+m))$ unity vector with unity value in coordinate j.

When j is transient this expression can be transformed to

$$q_j(t) = \mathbf{p} e^{Qt} v_j^T, \tag{46}$$

according to (31).
Here $\mathbf{p}$ is the $(1 \times K)$ initial distribution, and v_j is the $(1 \times K)$ unity vector.

We can now define the sub-density of a data point, (x_i, c_i). $q_j(x_i)$ can be viewed as the probability of being in state j exactly prior to x_i, and the transition rate a_{jc_i} can be viewed as the probability of instantaneously moving from state j to the absorbing state c_i. This makes the joint probability of being in j exactly prior to absorption, and also being absorbed into state c_i at time x_i, equal to $q_j(x_i)a_{jc_i}$. Since the chain must be in some transient state exactly prior to absorption, the sub-density of c_i is

$$f_{c_i}(x_i) = \sum_j q_j(x_i) a_{jc_i},$$

where the sum runs over all the transient states.

By using simple theory on conditional distributions,

$$P(A|B) = \frac{P(A \cap B)}{P(B)}, \tag{47}$$

it is now easy to see that the conditional probability of being in j precisely before absorbtion is given by,

$$\tilde{\pi}_j^i = \mathbb{P}\left(J_{x_i^-} = j \middle| X = x_i, C = c_i\right) = \mathbb{P}_{(x_i,c_i)}(J_{x_i^-} = j) = \frac{q_j(x_i)a_{jc_i}}{f_{c_i}(x_i)}, \tag{48}$$

where x_i^- is the moment exactly prior to absorption.

The distribution, $\mathbb{P}\left(J_{x_i^-} = j \middle| X \geq x_i, C = c_i\right)$, is also derived by using (47).
Because

$$\mathbb{P}(X \geq x_i, C = c_i) = \int_{x_i}^{\infty} f_{c_i}(u)du,$$

the conditional probability of being in state j exactly prior to x_i, given $X \geq x_i$ and $C = c_i$,
is

$$\pi_j^{*i} = \mathbb{P}\left(J_{x_i^-} = j \middle| X \geq x_i, C = c_i\right) = \mathbb{P}^*_{(x_i,c_i)}(J_{x_i^-} = j)$$

$$= \frac{\mathbb{P}\left((J_{x_i^-} = j)\ \cap\ (C = c_i)\ \cap (X \geq x_i)\right)}{\int_{x_i}^{\infty} f_{c_i}(u)du}. \tag{49}$$

Using (47) again, we get

$$\mathbb{P}\left((J_{x_i^-} = j)\ \cap\ (C = c_i)\ \cap\ (X \geq x_i)\right)$$

$$= \mathbb{P}\left(C = c_i \mid\ (J_{x_i^-} = j)\ \cap\ (X \geq x_i)\right) \cdot \mathbb{P}\left((J_{x_i^-} = j)\ \cap\ (X \geq x_i)\right). \tag{50}$$

Because j is a transient state the absorption obviously happens after or exactly at x_i, such that

$$\left((J_{x_i^-} = j)\ \cap\ (X \geq x_i)\right) = (J_{x_i^-} = j).$$

This simplifies (50), giving that

$$\mathbb{P}\left(C = c_i \mid\ (J_{x_i^-} = j)\ \cap\ (X \geq x_i)\right)$$

$$= \mathbb{P}\left(C = c_i \mid\ J_{x_i^-} = j\right) = \int_0^{\infty} f_{c_i}^j(u)du = F_{c_i}^j(\infty). \tag{51}$$

Here, $f_{c_i}^j(u)$ is the sub-density when the initial state is j, such that the initial distribution is the $(1 \times K)$ unity-vector, v_j.

Also,

$$\mathbb{P}\left((J_{x_i^-}=j)\ \cap\ (X\geq x_i)\right)=\mathbb{P}\left(J_{x_i^-}=j\right)=q_j(x_i). \tag{52}$$

Using (51) and (52) we finally get that

$$\pi_j^{*i}=\frac{F_{c_i}^j(\infty)q_j(x_i)}{\int_{x_i}^{\infty}f_{c_i}(u)du}. \tag{53}$$

One last observation can now be made before calculating the importance weights,

$$\frac{\mathbb{P}_{(x_i,c_i)}\left(\{J_t,\ t<x_i\}=\{j_t,\ t<x_i\}\right)}{\mathbb{P}^*_{(x_i,c_i)}\left(\{J_t,\ t<x_i\}=\{j_t,\ t<x_i\}\right)},$$

needed to derive the acceptance probabilities in the MH-method. These weights constitute the fractions in the numerator and denominator of the second expression in (42).

The observation is the following:

Because $\{J_t\}$ is a Markov chain, the Markov property ensures that whatever happens before the time x_i^- is independent of the absorption, (X,C), given that we know $J_{x_i^-}$. This can be interpreted as; if we know the state at time x_i^-, whatever happens before x_i^- does not provide more information on the outcome of the absorption, (X,C), which might happen after time x_i^-. The independence goes both ways, so (X,C) will also not provide any additional information on the history beyond x_i^-, given $J_{x_i^-}$.
This is a direct consequence of the Markov property, given in (1).

A consequence of this observation is that

$$\mathbb{P}_{(x_i,c_i)}\left(\{J_t,\ t<x_i\}=\{j_t,\ t<x_i\}|\ J_{x_i^-}=j_{x_i^-}\right)$$

$$=\mathbb{P}^*_{(x_i,c_i)}\left(\{J_t,\ t<x_i\}=\{j_t,\ t<x_i\}|\ J_{x_i^-}=j_{x_i^-}\right) \tag{54}$$

This can be directly used in the expression for the importance weights, which now can be seen to be

$$\frac{\mathbb{P}_{(x_i,c_i)}\left(\{J_t,\ t < x_i\} = \{j_t,\ t < x_i\}\right)}{\mathbb{P}^*_{(x_i,c_i)}\left(\{J_t,\ t < x_i\} = \{j_t,\ t < x_i\}\right)}$$

$$= \frac{\mathbb{P}_{(x_i,c_i)}\left(\{J_t,\ t < x_i\} = \{j_t,\ t < x_i\} \middle|\, J_{x_i^-} = j_{x_i^-}\right)\mathbb{P}_{(x_i,c_i)}(J_{x_i^-} = j_{x_i^-})}{\mathbb{P}^*_{(x_i,c_i)}\left(\{J_t,\ t < x_i\} = \{j_t,\ t < x_i\} \middle|\, J_{x_i^-} = j_{x_i^-}\right)\mathbb{P}^*_{(x_i,c_i)}(J_{x_i^-} = j_{x_i^-})}$$

$$= \frac{\mathbb{P}_{(x_i,c_i)}(J_{x_i^-} = j_{x_i^-})}{\mathbb{P}^*_{(x_i,c_i)}(J_{x_i^-} = j_{x_i^-})} = \frac{\tilde{\pi}^i_{j_{x_i^-}}}{\pi^{*i}_{j_{x_i^-}}}. \tag{55}$$

If $\{j_t,\ t < x_i\}$ is an existing Markov chain realization, and $\{j'_t,\ t < x_i\}$ is a proposal realization, sampled from $\mathbb{P}^*_{(x_i,c_i)}\left(\{J_t,\ t < x_i\} = \{j'_t,\ t < x_i\}\right)$, then according to (42) and (55), α can now be expressed as

$$\alpha = \frac{\frac{\tilde{\pi}^i_{j'_{x_i-}}}{\pi^{*i}_{j'_{x_i-}}}}{\frac{\tilde{\pi}^i_{j_{x_i-}}}{\pi^{*i}_{j_{x_i-}}}} = \frac{\tilde{\pi}^i_{j'_{x_i-}}\ \pi^{*i}_{j_{x_i-}}}{\pi^{*i}_{j'_{x_i-}}\ \tilde{\pi}^i_{j_{x_i-}}}.$$

By using (48) and (53), and the notation $a_{jc} = a(j,c)$, this expression becomes

$$\alpha = \frac{\frac{q_{j'_{x_i-}}\left(x_i\right)a(j'_{x_i-},c_i)}{f_{c_i}(x_i)}\ \frac{F^{j_{x_i-}}_{c_i}(\infty)q_{j_{x_i-}}\left(x_i\right)}{\int_{x_i}^{\infty} f_{c_i}(u)du}}{\frac{F^{j'_{x_i-}}_{c_i}(\infty)q_{j'_{x_i-}}\left(x_i\right)}{\int_{x_i}^{\infty} f_{c_i}(u)du}\ \frac{q_{j_{x_i-}}\left(x_i\right)a(j_{x_i-},c_i)}{f_{c_i}(x_i)}}$$

$$= \frac{a(j'_{x_i-},c_i)F^{j_{x_i-}}_{c_i}(\infty)}{a(j_{x_i-},c_i)F^{j'_{x_i-}}_{c_i}(\infty)}.$$

Using (38), this can be expressed as

$$\alpha = \frac{a(j'_{x_i-},c_i)\left(\underline{v}_{j_{x_i-}}Q^{-1}Lv^T_{c_i}\right)}{a(j_{x_i-},c_i)\left(\underline{v}_{j'_{x_i-}}Q^{-1}Lv^T_{c_i}\right)}, \tag{56}$$

which is the final expression, used directly in the algorithm. Here, $\underline{v}_k$ is a unity vector in the transient state space, with unity value in component k, and v_k is a similar unity vector but for the absorbing state space.

It is interesting to compare this expression with the acceptance probability in the paper by Bladt et al. [6] (page 288). The acceptance probability is there on the form

$$\alpha^* = \frac{a(j'_{x_i-}, c_i)}{a(j_{x_i-}, c_i)},$$

where c_i can only take one value, since in this paper only one absorbing state is allowed.

We see that, apart from being multiplied by a fraction of marginal distribution expressions,

$$\frac{\left(-\underline{v}_{j_{x_i-}} Q^{-1} L v_{c_i}^T\right)}{\left(-\underline{v}_{j'_{x_i-}} Q^{-1} L v_{c_i}^T\right)},$$

the expression in (56) looks completely similar to α^*.

This shows that by extending the method by Bladt et al. [6] to the competing risks case, we introduce another aspect to the method.
We now also have to take into consideration the probabilities of being absorbed into a particular absorbing state, c_i, from given last transient states, j_{x-} or j'_{x-}, in corresponding Markov chain realizations.
We might consider the method in the paper by Bladt et al. [6] to be a special case of this method, because there only one absorbing state is possible, so the marginal distribution expressions,

$$-\underline{v}_{j_{x_i-}} Q^{-1} L v_{c_i}^T \quad \text{and} \quad -\underline{v}_{j'_{x_i-}} Q^{-1} L v_{c_i}^T$$

both become equal to 1. Thus the fraction in (56) also becomes equal to 1.

3.1.2 Complete Metropolis-Hastings method

The Metropolis-Hastings algorithm can now be summed up as the following:

For each data point, (x_i, c_i),

1 If this is the first Gibbs step in the complete algorithm, draw an initial sample, $\{j_t,\ t < x_i\}$, from $\mathbb{P}^*_{(x_i,c_i)}\left(\{J_t,\ t < x_i\} = \{j_t,\ t < x_i\}\right)$, by using the *rejection procedure* explained in (45). If this is not the first Gibbs step, define the initial sample, $\{j_t,\ t < x_i\}$, as the Markov realization corresponding to (x_i, c_i) from the previous Gibbs step.

2 Draw a sample, $\{j'_t,\ t < x_i\}$, from $\mathbb{P}^*_{(x_i,c_i)}\left(\{J_t,\ t < x_i\} = \{j'_t,\ t < x_i\}\right)$, by using the *rejection procedure*.

3 Draw $U \sim \text{Uniform}[0,1]$.

4 If $U \leq \min\left(1,\ \frac{-a(j'_{x_i-},c_i)\left(\underline{v}_{j_{x_i-}}Q^{-1}Lv^T_{c_i}\right)}{-a(j_{x_i-},c_i)\left(\underline{v}_{j'_{x_i-}}Q^{-1}Lv^T_{c_i}\right)}\right)$ then replace $\{j_t,\ t < x_i\}$ with $\{j'_t,\ t < x_i\}$.

5 Return to step 2

3.2 Gibbs step

The MH-method is supposed to be run a finite number of times for each Gibbs-step in the complete algorithm. In each Gibbs step, approximate draws from the posterior distribution of A and $\mathbf{p}$ are made.

The Metropolis-Hastings algorithm provides simulated information on the underlying Markov chain realizations corresponding to the data points (x_i, c_i). Information from these realizations are used in the likelihood-part of the posterior distribution.

With K transient states, m absorbing states, and the $((K+m) \times (K+m))$-matrix A consisting of the components a_{ij}, for arbitrary states i and j, the prior is chosen to be on the following form

$$\Phi(\mathbf{p}, A) \propto \left(\prod_{i=1}^{K} \mathbf{p}_i^{\beta_i - 1}\right)\left(\prod_{i=1}^{K}\prod_{j=1, j\neq i}^{K+m} a_{ij}^{\nu_{ij}-1} e^{-a_{ij}\zeta_i}\right), \tag{57}$$

where β_i, ν_{ij} and ζ_i are hyper-parameters corresponding to the states i and j.

Because A is the intensity matrix of a Markov chain, all the components on the rows of absorbing states are equal to 0, so only components on the transient rows are sampled. Also, the diagonal components, a_{ii}, are just negative sums of the other row components, so they are not sampled.

A convenient fact with this prior distribution is that we immediately observe that $\mathbf{p}$ can be viewed as an independent Dirichlet distributed vector variable, if all the other parameters are held constant, and each a_{ij} can be viewed as independent gamma distributed variables, again if all other parameters are held constant. This is because we recognize the kernels of these distributions in the prior expression. The distributions are given as:

$a_{ij} \sim \text{Gamma}(1/\zeta_i, \nu_{ij})$

$\mathbf{p} \sim \text{Dirichlet}(\beta)$

Here, β is a vector with components β_i.

These distributions are in general easy to sample from.

The posterior expression is more complicated, but on the same form. Now the likelihood-information comes into play. This information is information gathered from all the Markov chain realizations corresponding to the data points, and is on the following form:

B_i is the number of times any of the Markov chain realizations are initiated in state i.

Z_i is the total time all the Markov chain realizations have spent in state i.

N_{ij} is the number of times a transition from state i to state j has been made in any of the Markov chain realizations.

This can be derived by looking at one single realization. A realization is defined completely by the states that have been visited, the order they have been visited in, and the sojourn times, meaning the time spent in the different states visited. In any Markov chain, the sojourn times are exponentially distributed, with rates equal to the absolute values of the diagonal intensities of the states visited.

In addition, a well known result is:
given that there is a transition from state i, the change to state j has a probability equal to

$$\rho_{ij} = \frac{a_{ij}}{-a_{ii}}. \tag{58}$$

At last, there is always an initial state, and the probability of a particular initial state, j_0, is given by $\mathbf{p}_{j_0}$, component j_0 of the initial distribution.

By now considering a realization, y, which has n transient state visits, and is defined by the vector of visited states,

$$J = [j_0, j_1, j_2, ..., j_n, j_c],$$

and the vector of sojourn times,

$$S = [s_0, s_1, s_2, ..., s_n],$$

the likelihood can be defined as

$$\begin{aligned} g(y|A,\mathbf{p}) &= \mathbf{p}_{j_0}(-a_{j_0j_0})\rho_{j_0j_1}e^{a_{j_0j_0}s_0}\cdots(-a_{j_nj_n})\rho_{j_nj_c}e^{a_{j_nj_n}s_n} \\ &= \mathbf{p}_{j_0}(-a_{j_0j_0})\frac{a_{j_0j_1}}{-a_{j_0j_0}}e^{a_{j_0j_0}s_0}\cdots(-a_{j_nj_n})\frac{a_{j_nj_c}}{-a_{j_nj_n}}e^{a_{j_nj_n}s_n} \\ &= \mathbf{p}_{j_0}a_{j_0j_1}e^{a_{j_0j_0}s_0}\cdots a_{j_nj_c}e^{a_{j_nj_n}s_n} \\ &= \mathbf{p}_{j_0}a_{j_0j_1}\exp\left(-s_0\sum_{j\neq j_0}a_{j_0j}\right)\cdots a_{j_nj_c}\exp\left(-s_n\sum_{j\neq j_n}a_{j_nj}\right) \\ &= \mathbf{p}_{j_0}\prod_{i=1}^{K}\prod_{j=1,j\neq i}^{K+m}a_{ij}^{n_{ij}}e^{-a_{ij}z_i}, \end{aligned} \tag{59}$$

where z_i is the sum of the sojourn times in state i, and n_{ij} is the number of transitions from i to j.

It is important to see that j_n is defined as the final state before absorption, and thus, transition from j_n will always go to the absorbing state, c.

Because there are many sampled realizations, $Y = [y_1, ..., y_{\mathbb{N}}]$, the complete likelihood will be products of these types of expressions, on the following form

$$f(Y|A,\mathbf{p}) = \left(\prod_{i=1}^{K}\mathbf{p}_i^{B_i}\right)\left(\prod_{i=1}^{K}\prod_{j=1,j\neq i}^{K+m}a_{ij}^{N_{ij}}e^{-a_{ij}Z_i}\right), \tag{60}$$

where B_i is the total number of initiations in state i, $Z_i = z_i^{(1)} + z_i^{(2)} + \dots$ is the sum of z_i for all the realizations, and $N_{ij} = n_{ij}^{(1)} + n_{ij}^{(2)} \dots$ is the sum of n_{ij}.

It is easy to see that B_i, N_{ij} and Z_i are the same as given in the definitions earlier.

Thus (60) gives the likelihood distribution, and the posterior distribution is just the product of the likelihood and the prior distribution, which is represented as

$$\pi(A, \mathbf{p}|Y) \propto \left(\left(\prod_{i=1}^{K} \mathbf{p}_i^{\beta_i - 1} \right) \left(\prod_{i=1}^{K} \prod_{j=1, j\neq i}^{K+m} a_{ij}^{\nu_{ij}-1} e^{-a_{ij}\zeta_i} \right) \right)$$

$$\cdot \left(\left(\prod_{i=1}^{K} \mathbf{p}_i^{B_i} \right) \left(\prod_{i=1}^{K} \prod_{j=1, j\neq i}^{K+m} a_{ij}^{N_{ij}} e^{-a_{ij} Z_i} \right) \right)$$

$$= \left(\prod_{i=1}^{K} \mathbf{p}_i^{\beta_i + B_i - 1} \right) \left(\prod_{i=1}^{K} \prod_{j=1, j\neq i}^{K+m} a_{ij}^{\nu_{ij} + N_{ij} - 1} e^{-a_{ij}(Z_i + \zeta_i)} \right). \tag{61}$$

In just the same way as for the prior, we see that this distribution consists of independent gamma distributions and a Dirichlet distribution.

We now get that:

$a_{ij} \sim \text{Gamma}(1/\left(\zeta_i + Z_i\right), \nu_{ij} + N_{ij})$

$\mathbf{p} \sim \text{Dirichlet}(\beta + B)$

Here, B is the vector with components B_i.

These distributions are as easy to sample from as the prior distributions.

Once the likelihood information is gathered, in the MH-step, and the parameters are sampled from the posterior distribution, the new parameter values are used in the next iteration to produce better estimates.

The complete algorithm can now be described as:

1 Specify hyper-parameters, β_i, ν_{ij} and ζ_i, where $i = 1, ..., K$ and $j = 1, ..., K + m$, and draw prior samples from the prior distribution.

2 Generate a Markov chain realization, y_k, corresponding to each of the $\mathbb{N}$ data points, (x_k, c_k), with finite steps of the MH-method.

3 Extract information;
$B = (B_1, B_2, ..., B_K),\ N = \{N_{ij} \text{ for } i \neq j,\ i = 1, ..., K,\ j = 1, ..., K + m\}$ and $Z = (Z_1, Z_2, ..., Z_K)$ from the Markov chain realizations.

4 Draw new values for the intensity parameters, a_{ij},
where $i \neq j,\ i = 1, ..., K,$ and $j = 1, ..., K + m$,
using $\text{Gamma}\left(1/\left(\zeta_i + Z_i\right), \nu_{ij} + N_{ij}\right)$.
Draw new values for the initial distribution, $\mathbf{p} = (p_1, ..., p_K)$,
using $\text{Dirichlet}(\beta + B)$.

5 Return to step 2

This algorithm is used to sample estimates of A and $\mathbf{p}$, which can then be used to estimate functions of the underlying competing risks distribution, like the sub-densities, sub-distributions, probabilities, quantiles, etc. There are mainly two possibilities here. One can for example first sample A and $\mathbf{p}$, and then use these samples to find the posterior mean of A and $\mathbf{p}$, which can later be used to estimate the chosen functions. Another option is to sample functional values directly at each iteration, and calculate the posterior mean of the function for each time-point, t.

It is important to note that a distribution can be represented in Phase-type form in many different ways, so the estimates of A and $\mathbf{p}$ can look very different from a distinct Phase-type approximation of the underlying distribution. Even so, the samples from the algorithm converges towards samples of Phase-type parameters which correspond to a true approximation of the underlying distribution.

How many samples needed to achieve convergence is highly individual for each data set and each underlying distribution, not to mention the choice of hyper-parameters, which defines the prior information. In many situations the convergence might in practice never be reached, so the choice of hyper-parameters can be very important. For many data sets it might be hard to find hyper-parameters which give reasonably quick mixing, if they exist at all. This problem leads to a variant of the algorithm above, where the hyper-parameters are randomized so the mixing is improved.

3.3 Randomized hyper-parameters

To make the algorithm more flexible with respect to the choice of prior distribution, the following conditional prior distributions are proposed:

$\mathbf{p} \sim \text{Dirichlet}(\beta)$

$\theta_i \sim \text{Gamma}\left(1/d,\ c\right)$

$\zeta_i \sim \text{Gamma}\left(1/\theta_i,\ \tau_i\right)$

$a_{ij} \sim \text{Gamma}\left(1/\zeta_i,\ \nu_{ij}\right)$

Here, c, d and τ_i are new hyper-parameters, and we see that ζ_i is now a random variable. Its distribution depends on the new variable, θ_i, which is a random variable with a distribution completely dependent on the hyper-parameters, c and d. By making ζ_i a random variable, we will see below that it has to be estimated for each iteration, which makes the algorithm more flexible.

The posterior expression now becomes

$$\pi(A, \mathbf{p}|Y) \propto \tag{62}$$

$$\left(\left(\prod_{i=1}^{K} \mathbf{p}_i^{\beta_i - 1}\right)\left(\prod_{i=1}^{K}\prod_{j=1,j\neq i}^{K+m} \zeta_i^{\nu_{ij}} a_{ij}^{\nu_{ij}-1} e^{-a_{ij}\zeta_i}\right)\left(\prod_{i=1}^{K} \theta_i^{\tau_i}\zeta_i^{\tau_i - 1} e^{-\theta_i\zeta_i}\right)\left(\prod_{i=1}^{K} \theta_i^{c-1} e^{-d\theta_i}\right)\right)$$

$$\cdot\left(\left(\prod_{i=1}^{K} \mathbf{p}_i^{B_i}\right)\left(\prod_{i=1}^{K}\prod_{j=1,j\neq i}^{K+m} a_{ij}^{N_{ij}} e^{-a_{ij}Z_i}\right)\right)$$

$$= \left(\prod_{i=1}^{K} \mathbf{p}_i^{\beta_i + B_i - 1}\right)\left(\prod_{i=1}^{K}\prod_{j=1,j\neq i}^{K+m} \zeta_i^{\nu_{ij}} a_{ij}^{\nu_{ij}+N_{ij}-1} e^{-a_{ij}(Z_i+\zeta_i)}\right)\left(\prod_{i=1}^{K} \theta_i^{\tau_i + c - 1}\zeta_i^{\tau_i - 1} e^{-\theta_i\zeta_i - d\theta_i}\right).$$

If each random variable in this expression is considered separately, while at the same time, all the other variables in the expression are considered constant, it is possible to extract the kernels of the full conditional distributions. By using these kernels we can decide the full conditional distributions of all the random variables.

These can be seen to be:

$\mathbf{p} \sim \text{Dirichlet}(\beta + B)$

$\theta_i \sim \text{Gamma}\left(1/\left(\zeta_i + d\right),\ \tau_i + c\right)$

$\zeta_i \sim \text{Gamma}\left(1/\left(\theta_i - a_{ii}\right),\ \sum_{j=0,j\neq i}^{K+m} \nu_{ij} + \tau_i\right)$

$a_{ij} \sim \text{Gamma}\left(1/\left(\zeta_i + Z_i\right),\ N_{ij} + \nu_{ij}\right)$

These are now the distributions we draw samples from at each iteration.

3.4 Censoring

So far, the method has been developed to handle complete data sets. An extension of the algorithm to also handle right censored data sets is easily accomplished. This is because the distribution needed to sample a Markov realization corresponding to a right censored data point is easy to sample from. It is given by

$$\mathbb{P}\left(\{J_t\} = \{j_t\} \middle| X > x_i\right) \equiv \mathbb{P}^*_{(x_i)}\left(\{J_t\} = \{j_t\}\right) \tag{63}$$

Similar to (44), this can be sampled from by sampling Markov chain realizations from the complete distribution, $\mathbb{P}\left(\{J_t\} = \{j_t\}\right)$, repeatedly until a sampled realization has an absorption time beyond x_i, completely analogous to the *rejection procedure* explained in (45). This realization will then be accepted as a sample from (63). Sampling Markov chain realizations from $\mathbb{P}\left(\{J_t\} = \{j_t\}\right)$ is done by the procedure mentioned at the end of the Markov process chapter.

It is important to note that a draw from this distribution is done directly, not by using an iterative method like a MH-method. This is because the MH-step in the non-censored case is used to make the Markov realizations eventually become draws from (43), by using (44) as a proposal distribution. For the censored case, the target distribution to sample from is (63), which is easily sampled from using the *rejection procedure*. The complete algorithm is adjusted such that it uses this sampling procedure for every right censored data point, and the MH-method for every complete data point.

4 Introducing covariates

In statistics one is very often interested in modeling the way certain non-stochastic variables, or covariates, affect the outcome of a response model. This can in general be done by including values of these covariates in the data, implementing the covariates in the response model, and then use some statistical method to estimate the effect of the covariates on the model. In lifetime analysis a lot of attention has been devoted to finding good methods for modeling how covariates affect the failure time distributions, and important quantities connected to these.

As is explained by Ansell & Phillips [4] (page 58), two important types of covariate variables are factors and variates.

4.0.1 Factors

A factor is a variable which can take one of a finite number of integer values, called levels. The values of the levels do not need to have a numeric interpretation, but often correspond to a certain condition or state in which a subject under study is in at the time of the data sampling. The *main effect* of a factor level is a value which is included in the response model, and determines how a factor level influence the model. For level j, the effect can be denoted as α_j. Different levels have different main effects, and the factor influence the model through the size of these values. The different main effects of the levels are often introduced in the model using dummy binary covariates, which take either value 0 or value 1. If the factor has a levels, $a-1$ binary covariates are used, and with $X_2, ..., X_a$ as the binary covariates, the following coding of levels is favorable

Factor level	X_2	X_3	$\dots$	X_a
1	0	0	$\dots$	0
2	1	0	$\dots$	0
3	0	1	$\dots$	0
$\dots$	$\dots$	$\dots$	$\dots$	.
a	0	0	$\dots$	1

Table 1

This is mainly because it is now easy to incorporate the *main effects* in the model by using the binary covariates. If the main effect of level 1 is defined as $\alpha_1 = 0$, all the other main effects can be introduced as the coefficients of the corresponding binary covariates in a linear expression on the form

$$\alpha_2 X_2 + \alpha_3 X_3 + ... + \alpha_a X_a \tag{64}$$

Now, all that is needed to introduce the factor in the model is to implement this expression into some part of the model. The situation when all the binary covariates are 0 is now defined as factor level 1, and the other factor levels are defined as the situations when the corresponding binary covariates are 1 and all the other covariates are 0.

4.0.2 Variates

A variate is a continous variable which influence the model through a coefficient, β. A variate is represented in the model multiplied with this coefficient, which determines how much influence the variate value has on the model. The expression

$$\beta w \tag{65}$$

is introduced in some part of the model, where w is the variate value and β is the effect of the variate.

4.0.3 Cox regression

Cox suggested introducing covariates in a lifetime analysis setting by implementing what is called a proportional hazards model [4] (pages 68-80). The main principle in this model is that the hazard rates for different covariate values are set to be proportional to each other, making the ratio of hazard rates for different sets of covariates equal to a constant. In this model the covariates are either binary dummy variables, used to determine factor levels, or variates. The hazard rate for a set of covariates, $w = w_1, w_2, ...w_W$, can be written as

$$h_w(t) = \Psi h_0(t), \tag{66}$$

where Ψ is a constant and $h_0(t)$ is a function depending only on t, not the covariates. Because $h_0(t)$ only depends on t, it is the same for every set of covariates, and the ratio of the hazard rates for two different sets of covariates, $w^{(i)}$ and $w^{(j)}$, becomes

$$\frac{h_{w^{(i)}}(t)}{h_{w^{(j)}}(t)} = \frac{\Psi_i h_0(t)}{\Psi_j h_0(t)} = \frac{\Psi_i}{\Psi_j},$$

which is constant with respect to t.

The function, $h_0(t)$ is called the baseline hazard rate, and is the hazard rate for the specific set of covariate values in which every covariate is set to 0.

The constant part of the hazard rate for a set of covariates, Ψ, can be expressed in many ways, but the most common form is

$$\Psi = e^{\beta_1 w_1 + \beta_2 w_2 + \beta_3 w_3 + \ldots + \beta_W w_W} = \exp\left(\beta w^T\right).$$

This form is realistic, because hazard rates can not be negative, and the exponential function can have values from 0 to ∞.

One of the main reasons why this model is easy to work with is that a likelihood function not depending on any other underlying assumptions can be derived. It also turns out to be independent of the baseline hazard rates, such that β can be estimated directly from the data set, without any baseline assumptions. This likelihood function is called the partial likelihood and it is defined by

$$L(\beta) = \prod_{i=1}^{n_0} \frac{\exp\left(\beta S_{[i]}^T\right)}{\left(\sum_{j \in R_i} \exp\left(\beta w_j^T\right)\right)^{m_i}} \qquad \text{(Ansell \& Phillips [4], page 70).} \tag{67}$$

Here, $1, ..., n_0$ is the set of data indices for the ordered set of failure times, m_i is the number of failure times equal to the i'th smallest failure time, $t_{[i]}$, $S_{[i]}$ is the sum of covariate vectors for all the failure times equal to $t_{[i]}$, and R_i is the set of subjects at risk at time $t_{[i]}$.

It is evident from this expression that the right censored data points will contribute as subjects at risk, before the censoring times are passed, and is therefore indirectly influencing the partial likelihood. When censoring times are passed, the censored data points are no longer contributing to the risk sets, R_i. Censorings are thus accounted for in this expression.

By solving the first derivatives of this function equated to zero, one is able to estimate β. The negative information matrix can be found by evaluating the second derivatives, and using this, the standard errors for the estimates can be found [4] (page 70).

If a factor with a levels is present in the model, then it is represented by using binary covariates which are used directly in the proportional hazards expression. For any covariate number, r, the covariates $(w_r, w_{r+1}, ..., w_{r+a-1})$ are the $a-1$ binary covariates, they are coded as in Table 1, where

$$(X_2, X_3, ..., X_a) = (w_r, w_{r+1}, ..., w_{r+a-1}).$$

The β-estimates for the binary covariates can thus be regarded as estimates for the *main effects* of the factor levels $(2, ..., a)$. Factor level 1 is defined to have *main effect* equal to 0.

With variates, the situation is straightforward. If a covariate, w_r, represents a variate, it is the variate value, and the β-estimates are the variate coefficients.

For this model, the baseline case is defined as the case when all covariates are equal to zero at the same time, which includes both variates and binary covariates.

So far, Cox regression is only defined for lifetime models with one failure type. Generalizing it to competing risks, by using it on the cause-specific hazard rates in (22), is not possible without making some assumptions on the way to handle the different types of failures. For a particular type of failure, j, the cause-specific hazard rate can be modeled with the proportional hazards model, but the other types of failures must be accounted for. A way to do this is to treat them as censored observations, and use ordinary Cox regression on the cause-specific hazard rate for failure j. If this model is chosen, separate Cox regression analyses are performed for each failure type, j. The estimated β-values are now called the cause specific β-values, and are denoted as β_j.

4.0.4 The Fine & Gray method

Cox regression is a very common way to introduce covariates in lifetime distributions, and works fine in many situations. An alternative is to use the method developed by Fine & Gray [9]. This method has been developed because it is a known fact that covariates sometimes tend to influence the cause-specific hazard rates very differently than the sub-distributions, given by (16) (Fine & Gray [9], page 496). As Fine & Gray notes, sometimes there might be a large covariate effect on the cause-specific hazard rate, but no effect on the corresponding sub-distribution, and vice versa. This makes it sometimes impossible to measure covariate effects on the sub-distributions, using ordinary proportional hazards methods on the cause-specific hazard rates, and one is in this situation forced to measure the covariate effects on the sub-distributions differently. Fine & Gray proposes defining a function called the sub-distribution hazard, given by

$$\breve{\lambda}_j = \lim_{\Delta t \to 0} \frac{1}{\Delta t} Pr\{t \leq T \leq t + \Delta t, C = j | T \geq t \cup (T \leq t \cap C \neq j), \mathbf{w}\}$$

$$= \frac{\frac{dF_j(t;\mathbf{w})}{dt}}{1 - F_j(t;\mathbf{w})} = \frac{-d \log\{1 - F_j(t;\mathbf{w})\}}{dt}. \tag{68}$$

Here, j is the specified failure type, and $\mathbf{w}$ is the set of covariates.

They view this expression as the hazard rate of the random variable

$$T^{\star} = \begin{cases} T & \text{if } C = j \\ \infty & \text{if } C \neq j \end{cases} \tag{69}$$

which defines the failure time of a type j failure as the original failure time given in the data set, T, and the failure time of any other type of failure as ∞.

Put simply, the Fine & Gray method wants to model the covariate effects through the sub-distribution functions, rather than the cause-specific hazard rates, and therefore covariate regression on a transformation of the sub-distribution functions is performed. This transformation is interpreted as the hazard rate of the artificial random variable, $T^{\star}$, and a proportional hazards method is used on this to perform the regression.

Fine & Gray admits in their paper that the method theoretically has its problems, because the risk set in the partial likelihood used to perform the regression becomes unnaturally large (Fine and Gray [9], page 497). This is because, in this model, when the partial likelihood is estimated for a particular failure time, t, and failure type, j, failures that have already occurred from other failure types must represent subjects that are still at risk. This is seen from (69), which defines failure times of these failures as ∞. That already failed subjects contribute to the number of subjects at risk is clearly an unnatural assumption.

An important aspect with this method appears when censorings are introduced. The definition of $T^{\star}$ greatly complicates many types of censorings, and Fine & Gray works around these complications partly by using what they call a *weighted score function* (Fine & Gray [9], pages 500-501). This function is mathematically difficult to derive, and will not be discussed further in this text, but it's complexity indicates that censorings in this method generally is a problematic subject.

In *R* this method can be applied using the function *crr* in the package *cmprsk*.

4.0.5 Other methods

Scheike and Zhang have developed methods which are similar to the FG (Fine & Gray) method, and in addition allow time-varying effects for the covariates [18]. Just as for the FG method, the sub-distribution functions are used for modeling covariate effects. There are generally two transformations of the sub-distribution functions which are considered (Scheike and Zhang [18], pages 2-3):

Proportional models on the form

$$h\{F_j(t;w,z)\} = c\log\log\{1-F_j(t;w,z)\} = \alpha(t)w^T + \gamma z^T, \tag{70}$$

where j is the failure type, w and z are different parts of the covariate vector, and $\alpha(t)$ and γ are time-varying and constant regression coefficient vectors respectively.

Additive models on the form

$$h\{F_j(t;w,z)\} = -\log\{1-F_j(t;w,z)\} = \alpha(t)w^T + (\gamma z^T)t. \tag{71}$$

Binomial regression is used to estimate the timevarying and constant effects, $\alpha(t)$ and γ, in both of the model types.

In *R* these models can be applied using the function *comp.risk* in the the package *timereg*.

Probably because these models are also based on using transformations of the sub-distribution functions, experience shows that they often give quite similar results as the FG method.

4.0.6 Phase-type models with covariates

So far, we have considered covariates which have been implemented in models by using either the cause-specific hazard rates (Cox regression) or through the sub-distributions (FG and other methods). It is also possible to introduce covariates in the Phase-type model used in this text. There are many ways to do this, because there are a lot of parameters and functionals related to this Phase-type model which can be modeled as covariate dependent expressions.

Cox regression goes through the cause-specific hazard rates, and the FG method goes through a transformation of the sub-distribution functions, both using non-parametric partial likelihood expressions. Different from these regression models, the Phase-type model is parametric, so one can use model parameters for covariate regression instead of indirectly and non-parametrically going through any of the functionals like the hazard rates or the sub-distribution functions. So which parameters should be used? This question has no good answer, because it depends on how one wants to model the covariate effects. In this text a proportional model has been used on the transition intensities for various parts of the transition matrix, A.

One possibility is using the transient part of the transition matrix, Q, and model each non-diagonal transition like

$$a_{ij} = q_{ij}\exp(\eta_{ij} w^T), \text{ when } i \text{ and } j \text{ are transient, and } i \neq j. \tag{72}$$

Here, w is a covariate vector, q_{ij} is the baseline intensity, η_{ij} is the covariate coefficient vector, and the transition is between the states i and j.

Another possibility is using the absorbing part of the intensity matrix, L, and instead model the absorbing intensities on this form, such that,

$$a_{ij} = l_{ij}\exp(\eta_{ij} w^T), \text{ when } i \text{ is transient and } j \text{ is absorbing.} \tag{73}$$

Here, l_{ij} is the baseline intensity.

In both of these models, the parameters not expressed in (72) and (73) are defined just as before.

Model (72) represents a situation where the transient transition intensities are affected directly by the covariates. In a *coxian* model, where there can be no feedback (moving from a higher to a lower state), modeling the covariates this way makes the covariates directly affect the total time spent in the different transient states, and therefore the speed of the transient transitions, which greatly affects the time to failure. This might be representative to a situation where there are different stages before failure, with different risk levels, like a mortal cancer patient going through the different stages of the disease before death. Here, the covariates will directly influence the speed in which the patient goes through the stages. A new drug which slows down the rate of the disease symptoms, and thus the movement speed through the stages, might be naturally modeled as a covariate in this coxian model.

In model (73) the risk of absorption at each transient state is directly affected by the covariates, but what happens in the transient part before failure is not directly affected. This model might be viewed as a type of parametric Phase-type analog to the Cox regression model. Because the hazards of failure (absorption intensities) at each transient state are directly influenced by the covariates, and the specific transition intensities are proportional to themselves at different covariate levels, just like the hazard rate in Cox regression. It is still worth mentioning that in this model regression is performed on the absorbing intensities, not the cause-specific hazard rates, represented by the expression in (22). If regression would be performed on these functionals, model (73) would just be a parametric version of the Cox regression model for competing risks, discussed at the end of the Cox regression sub-chapter. Thus (73) is theoretically different than the model used in Cox regression.

It is of course possible to use a combination of both the models above, such that the model becomes more flexible and can handle covariate influence more generally. A model like this would be able to handle countless ways of covariate influence on the underlying Phase-type model, and it should be able to adapt to a lot of different data sets. Still, such a general model has its cost, and that is efficiency. The amount of η-values in the vectors, η_{ij}, to be estimated would increase almost quadratically with the number of states used in the Phase-type fit. This can be computationally demanding, and in practice prevent an estimation from producing good estimates within reasonable time.

Two possible models have been presented here, and they are just a few examples on how covariate regression can be implemented in the Phase-type model. They all rely on the idea of regression with proportional parameter values, similar to proportional hazards methods. Other principles of regression could be applied on the parameters involved, and the initial distribution, $\mathbf{p}$, could also be expressed with the use of covariates, so there are other regression models available for Phase-type representations of competing risks processes.

4.0.7 Introducing covariates in the existing model

The model which has been used to produce the results in this text is model (73) with a constant vector, η_j, for each absorbing state, j, such that

$$\eta_{ij} = \eta_j \text{ for all transient states } i.$$

This has fewer η-values to be estimated compared to the more general models presented, and it therefore manage to produce good estimates within reasonable time for most data sets evaluated in this text. Because of its similarity with Cox regression, it has been expected to produce similar η-estimates. When implementing this model, the main extension of the Phase-type model without covariates, developed in the previous chapters, is to introduce the covariate-dependent expressions on the form of (73) as the new transition rates for the absorbing part of the intensity matrix. This transformation of the absorbing transition intensities will affect both the prior distribution and the likelihood expression.

In the rest of this text, η_j will represent the cause-specific covariate coefficient vector, with length W, for any absorbing state, j. η will represent the matrix with all of these vectors used as row vectors. Also, A will represent the matrix with the transient transition intensities, a_{ij}, and L will represent the matrix of baseline absorption intensities, l_{ij}.

(73) is just as before a Phase-type model, with an intensity matrix, and an initial distribution. The only difference now is that the Phase-type parameters are affected by the covariate values, and will therefore vary with these. Because (73) is a Phase-type model, the Metropolis-Hastings algorithm developed to sample Markov chain realizations corresponding to the data points, (x_k, c_k, w_k), works just as fine for this model as it did for the model without covariates. The only difference is that when applying this algorithm to sample the Markov chain realizations one has to use Phase-type parameters which are depending on the data points due to varying covariate values in the data.

The algorithm developed to estimate the parameters in the chosen covariate model is just an extended version of the algorithm without covariates. The new algorithm will consist of an initial part where prior parameter values are sampled, a Metropolis-Hastings part where Markov chain realizations are produced from the parameter estimates and the data, and a Gibbs-step where new parameter estimates are produced. The difference between the old and the new algorithm is that the covariate values make the estimation of parameters more difficult. Especially, the estimation of η_j complicates the situation a lot.

The outline for the development of the new covariate dependent algorithm is to first derive the likelihood expression and the prior distributions for the covariate model based on (73), and based on these derive the conditional posterior distributions. These are, as before, used to sample new parameter estimates in the Gibbs-step.

Using (59), and modeling the absorbing intensities in this expression according to (73), the likelihood expression for a Markov chain realization, y, corresponding to a data point, (x, c, w), is on the following form

$$g(y|A, L, \eta, \mathbf{p}) =$$

$$\mathbf{p}_{j_0}\left(\prod_{i=1}^{K}\prod_{j=1,j\neq i}^{K} a_{ij}^{n_{ij}} e^{-a_{ij}z_i}\right)\left(\prod_{i=1}^{K}\prod_{j=K+1,j\neq i}^{K+m}\left(l_{ij}\exp\left(\eta_j w^T\right)\right)^{n_{ij}} e^{-l_{ij}\exp\left(\eta_j w^T\right)z_i}\right)$$

$$= \mathbf{p}_{j_0}\left(\prod_{i=1}^{K}\prod_{j=1,j\neq i}^{K} a_{ij}^{n_{ij}} e^{-a_{ij}z_i}\right)\left(\prod_{i=1}^{K}\prod_{j=K+1,j\neq i}^{K+m} l_{ij}^{n_{ij}}\exp\left(n_{ij}\eta_j w^T\right) e^{-l_{ij}\exp\left(\eta_j w^T\right)z_i}\right)$$

$$= \mathbf{p}_{j_0}\left(\prod_{i=1}^{K}\prod_{j=1,j\neq i}^{K} a_{ij}^{n_{ij}} e^{-a_{ij}z_i}\right)\left(\prod_{i=1}^{K}\prod_{j=K+1,j\neq i}^{K+m} l_{ij}^{n_{ij}} e^{-l_{ij}\exp\left(\eta_j w^T\right)z_i}\right)$$

$$\cdot\exp\left(\sum_{i=1}^{K}\sum_{j=K+1,j\neq i}^{K+m} n_{ij}\eta_j w^T\right). \tag{74}$$

Here, all quantities which are also found in (59) have the same interpretation as before, w is the covariate vector, and η_j and l_{ij} are the cause-specific regression coefficients and the baseline absorption intensities, respectively. y now also include the information on the covariates, w.

Just as before, N_{ij} is defined as the total number of transitions from i to j, B_i is defined as the total number of initiations in state i, and Z_i is defined as the total amount of time spent in state i, for all the data points.
For data point (x_k, c_k, w_k), z_i^k is defined as the total time in state i and n_{ij}^k is defined as total number of transitions from i to j.

With the new definitions and the new likelihood expression for a single Markov chain realization, it is now possible to define the new total likelihood expression, in a similar way as for (60). It is the following:

$$f(Y|A, L, \eta, \mathbf{p}) = \left(\prod_{i=1}^{K}\mathbf{p}_i^{B_i}\right)\left(\prod_{i=1}^{K}\prod_{j=1,j\neq i}^{K} a_{ij}^{N_{ij}} e^{-a_{ij}Z_i}\right) \tag{75}$$

$$\cdot\left(\prod_{i=1}^{K}\prod_{j=K+1,j\neq i}^{K+m} l_{ij}^{N_{ij}}\exp\left(-l_{ij}\sum_{k=1}^{\mathbb{N}}\exp\left(\eta_j w_k^T\right)z_i^k\right)\right)\exp\left(\sum_{k=1}^{\mathbb{N}}\sum_{i=1}^{K}\sum_{j=K+1,j\neq i}^{K+m} n_{ij}^k\eta_j w_k^T\right),$$

which is derived by multiplying expressions like (74) for every data point, by using $w = w_k$, $n_{ij} = n_{ij}^k$ and $z_i = z_i^k$, and at the same time applying the definitions of B_i, N_{ij} and Z_i for the total sums. Here, Y is again the vector of $\mathbb{N}$ Markov chain realizations, $(y_1, ..., y_{\mathbb{N}})$, corresponding to the data points. Each Markov chain realization, y_k, includes information on the covariates.

The prior distribution is also affected by the covariates because it must account for the new parameters η_j and l_{ij}. Using randomized hyper-parameters, the kernel of the joint prior, $p(A_0, \mathbf{p}, \zeta, \theta)$, for the case without covariates is given by the first expression in (62), such that

$$p(A_0, \mathbf{p}, \zeta, \theta) \propto \left(\prod_{i=1}^{K} \mathbf{p}_i^{\beta_i - 1}\right) \left(\prod_{i=1}^{K} \prod_{j=1, j\neq i}^{K+m} \zeta_i^{\nu_{ij}} a_{ij}^{\nu_{ij}-1} e^{-a_{ij}\zeta_i}\right)$$

$$\cdot \left(\prod_{i=1}^{K} \theta_i^{\tau_i + c - 1} \zeta_i^{\tau_i - 1} e^{-\theta_i \zeta_i - d\theta_i}\right), \qquad (76)$$

where A_0 is the complete intensity matrix for this case.

For simplicity, we want to keep the prior distributions for all the parameters which are not influenced directly by the covariates just the same as before. This means that introducing the covariates will not directly affect the prior distributions of a_{ij}, $\mathbf{p_i}$, ζ_i and θ_i, for transient states i and j. Introducing the priors for η_j and l_{ij}, where i is transient and j is absorbing, gives that (76) can be extended to

$$p(A, L, \eta, \mathbf{p}, \zeta, \theta) \propto$$

$$\left(\left(\prod_{i=1}^{K} \mathbf{p}_i^{\beta_i - 1}\right) \left(\prod_{i=1}^{K} \prod_{j=1, j\neq i}^{K} \zeta_i^{\nu_{ij}} a_{ij}^{\nu_{ij}-1} e^{-a_{ij}\zeta_i}\right) \left(\prod_{i=1}^{K} \theta_i^{\tau_i + c - 1} \zeta_i^{\tau_i - 1} e^{-\theta_i \zeta_i - d\theta_i}\right)\right)$$

$$\cdot \left(\prod_{j=K+1}^{K+m} \text{prior}(\eta_j) \prod_{i=1}^{K} \text{prior}(l_{ij})\right). \qquad (77)$$

The priors for η_j and l_{ij} must be determined before going any further.

For l_{ij}, the baseline absorption intensities, the prior can be chosen on the same form as for the ordinary transition intensities, which is simply a Gamma-distribution, and the situation will be almost as simple as for these. This is because from (75) we see that the expressions containing l_{ij} constitute expressions equal to kernels of

$$\text{Gamma}\left(1/\left(\sum_{k=1}^{\mathbb{N}} \exp\left(\eta_j w_k^T\right) z_i^k\right), N_{ij}+1\right),$$

and with Gamma$(1/\zeta_i, \nu_{ij})$ as prior distributions for l_{ij} it is easy to see that the conditional posterior distributions for l_{ij}, conditioned on all the other parameters, then become Gamma-distributions, such that

$$l_{ij}|\ Y, \zeta, \theta, A, L_{(-ij)}, \eta, \mathbf{p}\ \sim$$

$$\text{Gamma}\left(1/\left(\zeta_i + \sum_{k=1}^{\mathbb{N}} \exp\left(\eta_j w_k^T\right) z_i^k\right), \nu_{ij}+N_{ij}\right). \tag{78}$$

The covariates complicate the situation a little bit, but these expressions are also quite simple to work with, and the sampling of l_{ij} is almost as easy as before. At every Gibbs-step, each l_{ij} is simply sampled from the Gamma-distribution given in (78).

The priors for l_{ij} can thus be expressed as

$$\text{prior}(l_{ij}) = \text{Gamma}\left(1/\zeta_i, \nu_{ij}\right) \tag{79}$$

Sampling η_j

The sampling of η_j is more difficult to perform. Here, it is important to note the fact that each η_j is a vector on the form of

$$\eta_j = (\eta_{j1}, \eta_{j2}, ..., \eta_{jW}),$$

where W is the number of covariates in the model.
For covariate indice r and failure j, the conditional posterior distribution for η_{jr} can be found by evaluating the full posterior expression. This means that a prior of each η_{jr} must be chosen first. The prior distributions for η_{jr} have been chosen as Normal distributions with mean equal to 0 and a chosen variance, σ. This is because η-values can have both negative and positive values, so a distribution

symmetric around 0 is natural to choose. Also, the Normal distribution has a simple density-expression to implement into a sampling algorithm for η_{jr}.

The priors for η_{jr} can thus be defined as

$$\text{prior}(\eta_{jr}) = \text{Normal}\left(0, \sigma\right) \tag{80}$$

Multiplying (75) and (77), with the new priors defined, one gets the kernel of the new posterior expression,

$$\begin{aligned}
\pi(A, L, \eta, \mathbf{p}, \theta, \zeta|Y) \propto & \left(\prod_{i=1}^{K} \mathbf{p}_i^{B_i}\right)\left(\prod_{i=1}^{K}\prod_{j=1,j\neq i}^{K} a_{ij}^{N_{ij}} e^{-a_{ij}Z_i}\right) \\
&\cdot \left(\prod_{i=1}^{K}\prod_{j=K+1,j\neq i}^{K+m} l_{ij}^{N_{ij}} \text{exp}\left(-l_{ij}\sum_{k=1}^{\mathbb{N}} \text{exp}\left(\eta_j w_k^T\right) z_i^k\right)\right) \text{exp}\left(\sum_{k=1}^{\mathbb{N}}\sum_{i=1}^{K}\sum_{j=K+1,j\neq i}^{K+m} n_{ij}^k \eta_j w_k^T\right) \\
&\cdot \left(\left(\prod_{i=1}^{K} \mathbf{p}_i^{\beta_i-1}\right)\left(\prod_{i=1}^{K}\prod_{j=1,j\neq i}^{K} \zeta_i^{\nu_{ij}} a_{ij}^{\nu_{ij}-1} e^{-a_{ij}\zeta_i}\right)\left(\prod_{i=1}^{K} \theta_i^{\tau_i}\zeta_i^{\tau_i-1}e^{-\theta_i\zeta_i}\right)\left(\prod_{i=1}^{K}\theta_i^{c-1}e^{-d\theta_i}\right)\right) \\
&\cdot \left(\prod_{j=K+1}^{K+m}\prod_{r=1}^{W} \text{prior}(\eta_{jr})\right)\left(\prod_{i=1,i\neq j}^{K} \text{prior}(l_{ij})\right) \\
= & \left(\prod_{i=1}^{K} \mathbf{p}_i^{B_i+\beta_i-1}\right)\left(\prod_{i=1}^{K}\prod_{j=1,j\neq i}^{K} a_{ij}^{N_{ij}+\nu_{ij}-1} e^{-a_{ij}(Z_i+\zeta_i)}\right) \\
&\cdot \left(\prod_{i=1}^{K}\prod_{j=K+1,j\neq i}^{K+m} l_{ij}^{N_{ij}+\nu_{ij}-1} \text{exp}\left(-l_{ij}\left(\zeta_i + \sum_{k=1}^{\mathbb{N}} \text{exp}\left(\eta_j w_k^T\right) z_i^k\right)\right)\right) \\
&\cdot \text{exp}\left(\sum_{k=1}^{\mathbb{N}}\sum_{i=1}^{K}\sum_{j=K+1,j\neq i}^{K+m} n_{ij}^k \eta_j w_k^T\right)\left(\prod_{i=1}^{K} \zeta_i^{\tau_i-1}\theta_i^{\tau_i+c-1}e^{-d\theta_i-\theta_i\zeta_i}\prod_{j=1,j\neq i}^{K}\zeta_i^{\nu_{ij}}\right) \\
&\cdot \prod_{j=K+1}^{K+m}\prod_{r=1}^{W} \frac{1}{\sigma\sqrt{2\pi}} \text{exp}\left(-\frac{\eta_{jr}^2}{2\sigma^2}\right).
\end{aligned} \tag{81}$$

To sample η-values, many options are available, but a simple glance at the posterior expression in (81) reveals that the conditional posterior distribution for any η_{jr} is complicated. It is not possible to recognize the kernel of any known distribution which is easy to sample from. This is why separate Metropolis-Hastings samplers have been chosen to sample the values of η_{jr}. These samplers will after some number of iterations start to sample good estimates of η_{jr}, which are then further used in the sampling of other parameters.

Any Metropolis-Hastings sampler must have a proposal distribution and an acceptance probability. By choosing Normal distributions which are symmetric around the existing η-estimates as proposal distributions, one gets acceptance probabilities that become independent of the proposal distribution. One can see this from (42). The symmetry makes

$$q(\theta, \Phi) = q(\Phi, \theta),$$

in this expression.

Normal distributions, symmetric around the existing estimates of η_{jr}, with any chosen standard deviation, Ω_{jr}, has been chosen as proposal distributions, such that

$$q(\eta_{jr}, \phi) = \text{Normal}\left(\eta_{jr}, \Omega_{jr}^2\right), \tag{82}$$

where ϕ is the proposed parameter value, and r is the covariate number.

Since the target distributions for these MH-methods are the conditional posterior distributions of the η-values, these have to be defined before the acceptance probabilities can be derived completely. By looking at the posterior expression in (81) it is possible to recognize the kernels of the joint densities of the η_j-vectors, such that

$$\pi\left(\eta_j \middle| Y, \zeta, \theta, A, \eta_{(-j)}, \mathbf{p}\right) \propto \left(\prod_{i=1}^{K} \exp\left(-l_{ij} \sum_{k=1}^{\mathbb{N}} \exp\left(\eta_j w_k^T\right) z_i^k\right)\right)$$

$$\cdot \exp\left(\sum_{k=1}^{\mathbb{N}} \sum_{i=1}^{K} n_{ij}^k \eta_j w_k^T\right) \left(\prod_{r=1}^{W} \exp\left(-\frac{\eta_{jr}^2}{2\sigma^2}\right)\right). \tag{83}$$

Since

$$\eta_j w_k^T = \eta_{j1} w_{k1} + \eta_{j1} w_{k1} + ... + \eta_{jW} w_{kW},$$

we have that

$$\exp\left(\sum_{k=1}^{\mathbb{N}} \sum_{i=1}^{K} n_{ij}^k \eta_j w_k^T\right) \propto \exp\left(\sum_{k=1}^{\mathbb{N}} \sum_{i=1}^{K} n_{ij}^k \eta_{jr} w_{kr}\right),$$

when the expression for η_{jr} is evaluated.

Using this and (83), we get that

$$\pi\left(\eta_{jr} \middle| Y, \zeta, \theta, A, \eta_{(-jr)}, \mathbf{p}\right) \propto \left(\prod_{i=1}^{K} \exp\left(-l_{ij} \sum_{k=1}^{\mathbb{N}} \exp\left(\eta_j w_k^T\right) z_i^k\right)\right)$$

$$\cdot \exp\left(\sum_{k=1}^{\mathbb{N}} \sum_{i=1}^{K} n_{ij}^k \eta_{jr} w_{kr}\right) \left(\exp\left(-\frac{\eta_{jr}^2}{2\sigma^2}\right)\right), \tag{84}$$

which defines the kernel of a conditional posterior distribution for η_{jr}. These distributions are the target distributions for the MH- method.

Now, finding the acceptance probabilities is easy. The acceptance probabilities are defined by using (42), and the only expressions involved in these will be the expressions for the conditional posterior kernels in (84), because of the symmetrical proposal distributions. These kernels constitute $\pi(\phi)$ in (42), where ϕ is the new proposal value of η_{jr}.

Using the expression in (42), and setting $\phi = \eta_{jr}^{(n)}$ and $\theta = \eta_{jr}^{(o)}$, we now have that

$$\alpha = \min\left(1, \frac{\left(\prod_{i=1}^{K} \exp\left(-l_{ij} \sum_{k=1}^{\mathbb{N}} \exp\left(\eta_j^{(n)} w_k^T\right) z_i^k\right)\right)}{\left(\prod_{i=1}^{K} \exp\left(-l_{ij} \sum_{k=1}^{\mathbb{N}} \exp\left(\eta_j^{(o)} w_k^T\right) z_i^k\right)\right)} \cdot \frac{\exp\left(\sum_{k=1}^{\mathbb{N}} \sum_{i=1}^{K} n_{ij}^k \eta_{jr}^{(n)} w_{kr}\right) \left(\exp\left(-\frac{{\eta_{jr}^{(n)}}^2}{2\sigma^2}\right)\right)}{\exp\left(\sum_{k=1}^{\mathbb{N}} \sum_{i=1}^{K} n_{ij}^k \eta_{jr}^{(o)} w_{kr}\right) \left(\exp\left(-\frac{{\eta_{jr}^{(o)}}^2}{2\sigma^2}\right)\right)}\right) \tag{85}$$

This is the complete acceptance probability expression, used directly in the algorithm.

Other parameters

Because of the covariates, the sampling process becomes a lot more complicated, as seen above. Although the baseline absorbtion intensities can be sampled in almost the same way as the absorbtion intensities in the model without covariates, the η-values need separate Metropolis-Hastings samplers which are iterated once for every Gibbs-step in the complete algorithm, and in practice need to converge before good estimates for many of the other parameters can be sampled. This is because other parameters are also depending on the η-values, directly or indirectly. The baseline absorption intensities, l_{ij}, are depending on the covariates and the η-estimates directly. This is easily seen from (78), which shows the sampling distribution of l_{ij}.

A dependency which is a little more obscure is the dependency of ζ on the values of l_{ij}. This dependency is seen through the posterior expression, (81). For a given transient state, i, one can see that

$$\pi\left(\zeta_i \middle| Y, A, L, \eta, \zeta_{(-i)}, \theta, \mathbf{p}\right) \propto$$

$$\left(\prod_{j=1,j\neq i}^{K} e^{-a_{ij}\zeta_i}\right)\left(\prod_{j=K+1}^{K+m} e^{-l_{ij}\zeta_i}\right)\left(\zeta_i^{\tau_i-1}e^{-\theta_i\zeta_i}\prod_{j=1,j\neq i}^{K}\zeta_i^{\nu_{ij}}\right)$$

$$= \zeta_i^{\tau_i+\left(\sum_{j=1,j\neq i}^{K}\nu_{ij}\right)-1}\exp\left(-\zeta_i\left(\theta_i+\sum_{j=1,j\neq i}^{K}a_{ij}+\sum_{j=K+1}^{K+m}l_{ij}\right)\right), \qquad (86)$$

which is recognized as a Gamma-distribution.

Based on (86), the conditional posterior distribution of ζ_i must be

$$\pi\left(\zeta_i \middle| Y, A, L, \eta, \zeta_{(-i)}, \theta, \mathbf{p}\right) =$$

$$\text{Gamma}\left(1/\left(\theta_i+\sum_{j=1,j\neq i}^{K}a_{ij}+\sum_{j=K+1}^{K+m}l_{ij}\right), \tau_i+\left(\sum_{j=1,j\neq i}^{K}\nu_{ij}\right)\right). \qquad (87)$$

This expression shows that ζ_i for any transient state, i, is dependent on l_{ij}, for all absorbing states, j. Because the estimates of l_{ij} are directly dependent on estimates of η_j and the covariates, one clearly sees that ζ is depending indirectly on the η-values and the covariates.

Because ζ is used in estimates of other parameters, this indirect dependency on the η-estimates and the covariates goes further. We see from the posterior expression in (81) that the parameters a_{ij} and θ_i are also influenced by ζ, and are therefore indirectly dependent on η-estimates and the covariates. But this dependency goes entirely through ζ, so the sampling distributions are just the same as before, given ζ.

It is evident that introducing covariates into the algorithm affects almost all the parameters in the model. The η-estimates are involved in most of the other parameter estimates, and because every η_{jr} is sampled through a MH-sampler, which needs to converge before the estimates are sampled from the correct distribution, most of the other parameter estimates will thus converge after this has occurred. One can view the estimates of η_{jr} as bottlenecks which increase the convergence time for the whole algorithm. Another reason that this new algorithm will have larger convergence time than the one without covariates is simply that this algorithm has more parameters. The increase in convergence time has been expected and has also been observed in the results which are presented later in this text.

Complete algorithm

The complete algorithm for the situation with covariates can be summed up as the following:

1 Specify hyper-parameters, β_i, ν_{ij}, c, d and τ_i, where $i = 1, ..., K$ and $j = 1, ..., K + m$, and draw prior samples from the following distributions:

$\mathbf{p} \sim \text{Dirichlet}(\beta)$

$\theta_i \sim \text{Gamma}\left(1/d, c\right)$

$\zeta_i \sim \text{Gamma}\left(1/\theta_i, \tau_i\right)$

$a_{ij} \sim \text{Gamma}\left(1/\zeta_i, \nu_{ij}\right)$

$l_{ij} \sim \text{Gamma}\left(1/\zeta_i, \nu_{ij}\right)$

$\eta_{jr} \sim \text{Normal}\left(0, \sigma\right), \text{ where } r = 1, ..., W$

2 Generate a Markov chain realization, y_k, corresponding to each of the $\mathbb{N}$ data points, (x_k, c_k, w_k), with finite steps of the MH-method, using model (73) with parameters $a_{ij}, l_{ij}, \eta_{jr}$, and $\mathbf{p}$, and the data points, (x_k, c_k, w_k).

3 Extract information;
$B = (B_1, B_2, ..., B_K),\ N = \{N_{ij} \text{ for } i \neq j,\ i = 1, ..., K,\ j = 1, ..., K + m\}$ and $Z = (Z_1, Z_2, ..., Z_K)$ from all the realizations together.

Extract information;
$n^k = \{n_{ij}^k \text{ for } i \neq j,\ i = 1, ..., K,\ j = 1, ..., K + m\}$ and $z^k = (z_1^k, z_2^k, ..., z_K^k)$, from each realization.

4 Draw new parameter values from the conditional posterior distributions:

$\mathbf{p} \sim \text{Dirichlet}(\beta + B)$

$\theta_i \sim \text{Gamma}\left(1/\left(\zeta_i + d\right), \tau_i + c\right)$

$\zeta_i \sim \text{Gamma}\left(1/\left(\theta_i + \sum_{j=1,j\neq i}^{K} a_{ij} + \sum_{j=K+1}^{K+m} l_{ij}\right), \tau_i + \left(\sum_{j=1,j\neq i}^{K} \nu_{ij}\right)\right)$

$a_{ij} \sim \text{Gamma}\left(1/\left(Z_i + \zeta_i\right), \nu_{ij} + N_{ij}\right)$

$l_{ij} \sim \text{Gamma}\left(1/\left(\zeta_i + \sum_{k=1}^{\mathbb{N}} \exp\left(\eta_j w_k^T\right) z_i^k\right), \nu_{ij} + N_{ij}\right)$

$\eta_{jr} \sim$ Distribution with kernel like (84). Sampled by one iteration of MH-method defined by proposal distribution (82) and the acceptance probability (85), after convergence has been reached.

5 Return to step 2

4.1 Hyper-parameters

An important aspect of this algorithm is the choice of hyper-parameters, c, d, τ_i, σ, β_i and ν_{ij}, for $i = 1, ..., K,\ j = 1, ..., K + m$ and $i \neq j$. These constants affect the posterior estimates, as seen from (81), and in addition greatly affects the convergence. As in any Bayesian method the prior distributions will influence the estimates made, so a certain prior distribution for any of the variables to be estimated might impose estimation bias and will in addition affect the estimation variance. Thus it is of great importance to choose priors with as little influence as possible, if there exists little or no prior information. For any of the data sets analyzed in this text, it has proven difficult to gain any useful information on the underlying parameters in model (73) before running the algorithm. This is largely because there can be many different Phase-type approximations giving similar results, and they do not need to have an intuitive interpretation in any way. For some data sets, there might be an intuitive way to model the process behind the data in a Phase-type setting, but in most cases this is not easy or possible. Some general rules when choosing the hyper-parameters are very useful, such that the prior distributions will impose as little estimation influence as possible and the estimates become almost completely defined by the data.

First of all, the estimate of η_{jr}, for absorbing state j and covariate r, is based on draws from the distribution in (84). This distribution is affected by the prior of η_{jr} through the kernel of a normal distribution, which is the expression

$$\exp\left(-\frac{\eta_{jr}^2}{2\sigma^2}\right).$$

From this it is clear that increasing σ will make the value of the whole expression move closer to 1, and decreasing σ will make the value of the whole expression move further away from 1. This expression is the prior part of the conditional posterior distribution for η_{jr}, and it is thus favourable to make this part as close to one as possible, since it is multiplied with the rest of the conditional posterior distribution. This generally means that σ should be chosen as large as possible. Doing this might impose problems though, since the algorithm will most definitely experience difficulties in the first iterations if σ is unnaturally large. This is because the risk of absorption increases exponentially with the η-values, so with very large η-values the *rejection procedure* in (45) will have difficulties producing Markov chain realizations with absorption times larger than the failure times in the data points (x_k, c_k, w_k). σ^2 is the variance of the prior distributions for the η-values, so large σ corresponds to large prior estimates of the η-values. The *rejection procedure* will simply stall, not being able to accept any Markov chain realizations, and the complete algorithm will stop.
It is thus a trade-off between large values of σ, making the posterior estimates less biased, and efficiently starting the algorithm.

With the choice of c, d, τ_i and ν_{ij}, the situation is more complicated. From the complete algorithm in last section, it is seen that the conditional posterior distributions for a_{ij} and l_{ij}, for any given states i and j, are depending on the random variables ζ_i. These random variables represent much of the prior dependence in the conditional posterior distributions for a_{ij} and l_{ij}. Minimizing ζ_i will thus minimize a great part the prior dependency of a_{ij} and l_{ij}. As mentioned, making estimates less dependent on the priors is in this algorithm preferable. One way of minimizing ζ_i is by making the prior distributions for ζ_i produce small prior estimates, and also minimizing the prior variances. Not only does one in this case start the algorithm with small estimates of ζ_i, but one also create a correlation structure in the rest of the algorithm where ζ_i are kept relatively small at all times. Choosing the hyper-parameters c, d and τ_i is thus reduced to creating small values of ζ_i throughout the algorithm, by producing small prior estimates of ζ_i with small prior variances. In a Gamma-distribution, the variance is given by

$$\mathrm{Var}\left(\mathrm{Gamma(scale,\ shape)}\right) = \mathrm{shape} \times \mathrm{scale}^2,$$

and the expectation is given by

$$\mathrm{E}\left(\mathrm{Gamma(scale,\ shape)}\right) = \mathrm{shape} \times \mathrm{scale}.$$

We have that

$$\mathrm{scale} = \frac{1}{\theta_i}$$

and

$$\mathrm{shape} = \tau_i,$$

in the priors of ζ_i. Minimizing prior variances and expectations of ζ_i is then reduced to maximizing θ_i and minimizing τ_i.

Because

$$\mathrm{scale} = \frac{1}{d}$$

and

$$\mathrm{shape} = c,$$

in the priors of θ_i, it is possible to maximize the prior expectations of θ_i by maximizing the ratio between c and d. If the values of θ_i are maximized this way, and the hyper-parameters τ_i are minimized, the priors will induce as little bias on the results as possible through the values of ζ_i.

But this is only one part of the story. ν_{ij} are also influencing the estimates of a_{ij} and l_{ij}, and should also be minimized such that the data influences the estimates as much as possible relative to the priors. This is easily seen from the conditional posterior distributions for a_{ij} and l_{ij}.

In the complete algorithm it is possible to see that the initial distribution, $\mathbf{p}$, is drawn from the Dirichlet distribution, Dirichlet$(\beta + B)$. This indicates that in order to make the data influence the estimation as much as possible, the components of the vector β should be as small as possible. Choosing small values of each component, β_i, is thus the strategy which usually is recommended for obtaining good estimates. It should still be mentioned that doing so might make the algorithm become unstable, producing estimates that takes a long time to converge. By instead fixing the initial distribution, choosing one state as the possible initial state in the Phase-type fit, will sometimes stabilize the behavior of the algorithm, making it easier to obtain good estimates within reasonable time. This is performed by choosing high values of β_i for the chosen initial state, i, and excluding the other transient states by choosing β-components equal to zero.

Choosing the hyper-parameters c, d, τ_i and ν_{ij} in the ways described here will create estimates with as little bias as possible, looking away from the estimation of η-values. But as with the η-values, it is also in this situation more difficult to get the algorithm to start when doing this. The problem now is numerical instabilities in working with the resulting intensity matrices, when the matrix components become too small as a result of minimizing ζ_i and ν_{ij}.

5 Results

5.1 Convergence issues

The results presented in this text have all been sampled with different burn-in periods, step-lengths, sample sizes and hyper-parameters. This is because every data set is unique and makes the MCMC method perform differently.

For some data sets, convergence is reached quickly and small burn-in periods are needed. Other data sets are more difficult to handle, and might in practice never converge for a large number of different hyper-parameters. For these data sets, it is often high auto-correlation between the iterations, which induce bad mixing and slow convergence.

To speed up convergence, with respect to the number of iterations used, one might increase the number of iterations the MH-part of the algorithm performs for each Gibbs-step. Doing this will make the Markov chain realizations converge more quickly towards being realizations from the estimated Phase-type model, which is the aim with the MH-part. This will speed up the convergence of the Markov chain realizations, with respect to the number of Gibbs-steps used, but not necessarily with respect to the time spent running the algorithm. The reason is that increasing the number of iterations of Markov chain realizations will also increase the time spent performing each Gibbs-step.
In fact, experience shows that the total run time of the algorithm becomes smaller with a very low number of Metropolis-Hastings iterations for each Gibbs-step. This has been the strategy while sampling the results in this text.

As mentioned, high auto-correlation induce bad mixing, which essentially means that the MCMC iterations for the parameter values traverse the sampling space of the conditional posterior distributions slowly. It is obvious that this will slow down the convergence of the algorithm, but what is also true is that high auto-correlation decreases the quality of the parameter samples. In this case, when convergence is reached the mixing is still slow, and so the different sample values of the parameters are very dependent on the previous values. A way to go around this problem is to sample parameter values which are far away from each other in the sampling chain, for example by only using every 10th iteration in the sample. This strategy, which is called thinning, will decrease the dependencies between the sample parameters and increase the quality of the samples, as opposed to using every iteration after the burn-in period in the sample.

A general way to decrease auto-correlation, is to optimize the step-lengths used in the MH-methods for the η-values. Choosing too large or too small step-lengths will surely increase the auto-correlation between the η-iterations, so an optimal step-length is preferable. Trial and error must be used here, as there is no general and easy way to find optimal step-lengths.

At last it is worth mentioning what often has the largest effect on reducing the auto-correlation in the η-iterations, and generally making the estimates as good as possible. For a data set the magnitude order of the covariate values in the data seems to dictate how the estimation of the η-values is performed in a substantial way. Especially, if the values of the covariates in the data are too large, the algorithm will perform poorly. The auto-correlation seems to become very high, and thus convergence behavior and sample quality is poor. Experience shows that simply down-scaling the covariate values will improve the situation a lot, and the algorithm might run a lot smoother and produce higher quality samples.

5.2 Results without covariates

Before moving on to results from the full covariate model, some results for the model without covariates can be useful to present. Three data sets have been analyzed, and the functions chosen to be estimated are the sub-density functions in (17), the sub-distribution functions in (16), and the cause-specific hazard rates in (22).
Posterior mean estimates of all the functions have been calculated and presented. The estimates of the sub-density functions and the cause-specific hazard rates have been presented together with estimated 95%-credibility intervals, and all the posterior estimates used in calculating the posterior mean, to show the uncertainties of the estimates.

The methods and code for estimating the posterior mean and credibility intervals can be found in the appendix, in chapter 7.1.

The Phase-type estimates of the sub-distribution functions have been presented together with non-parametric estimates based on the function *cuminc* in the R-package *cmprsk*. Details on this function and this package can be found in section 7.3 in the appendix.

To ensure convergence, parallel runs of the given model has been performed with the hyper-parameters constant and with different initial values for the parameters in the model.

Before moving on to the specific results, it is worth mentioning that for almost all the data sets analyzed in the work on this model, the sub-density values at $t = 0$ seem to be unstable, so good estimates of these functional values have been difficult to obtain.

5.2.1 Phase-type data

A Phase-type distribution with an intensity matrix not allowing any transitions backwards in the state space, is called a coxian model. These models are based on Markov chains with intensity values of zero for all transitions to the left of the diagonal. Such a model, with intensity matrix

$$A = \begin{bmatrix} -0.3 & 0.2 & 0.1 & 0 \\ 0 & -0.6 & 0.5 & 0.1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

has been analyzed here.

The theoretical initial distribution has been set to

$$p = \begin{bmatrix} 1, & 0, & 0, & 0 \end{bmatrix}.$$

The simulating model is on the same form as the model which the Phase-type algorithm for the case without covariates is based on.

The generated data set used here is complete and consists of 150 failures, where each data point is generated by simulating a Markov chain realization. This is performed by using the same method as in the *rejection procedure* in (45).
All the code and functions used to simulate this data set is found in chapter 7.2 and 7.3 in the appendix.

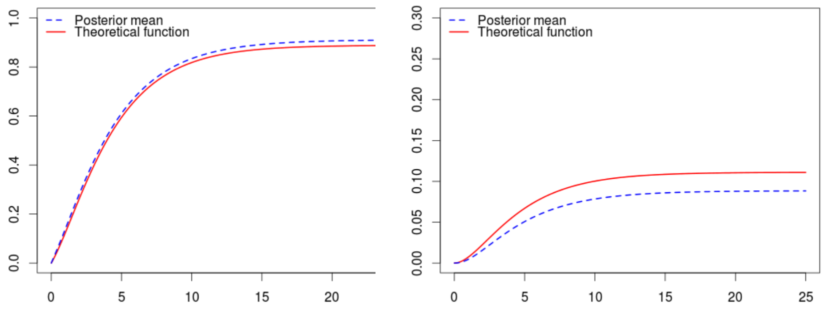
A Phase-type fit has been made with 5 states, where 3 are transient, and 2 are absorbing. States 4 and 5 correspond to causes 1 and 2, respectively.

The hyper-parameters chosen to get these results are as follows for any transient state, i, and absorbing state j:

$$\begin{aligned} c &= 5000 \\ d &= 10 \\ \tau_i &= 10 \\ \nu_{ij} &= 0.1 \\ \beta_i &= 0.01. \end{aligned}$$

The burn-in period is on 1500 iterations, and a sample of 500 posterior estimates has been obtained.

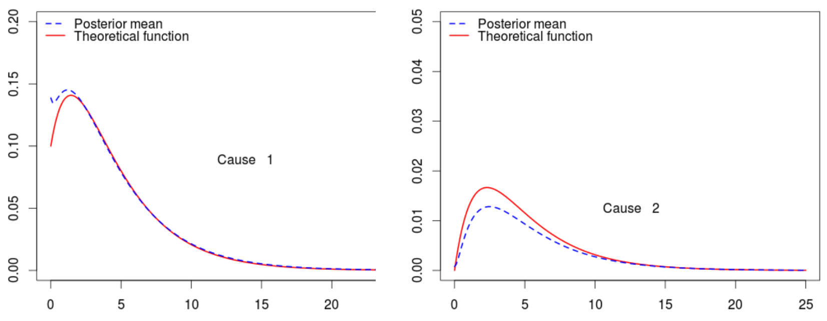
In figure 1, sub-distribution functions for causes 1 and 2 can be found. The sub-distribution function for cause 2 is seen to be very small, and this could have caused estimation problems, since it indicates that there are very few observations of this type of failure in the data. Even so, all the obtained function estimates for this data set are reasonable, as we will see.



Cause 1 Cause 2

Figure 1: Sub-distribution estimates for the Phase-type data without covariates

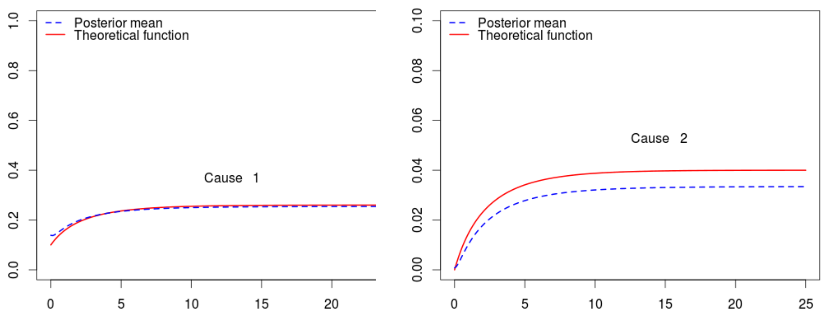
The sub-density estimates are seen in figure 2, and they both are close to the theoretical curves.



Cause 1 Cause 2

Figure 2: Sub-density estimates for the Phase-type data without covariates

The cause-specific hazard rates are plotted in figure 3. These estimates are also close to the theoretical functions.



Cause 1 Cause 2

Figure 3: Sub-hazard estimates for the Phase-type data without covariates

The Phase-type data are in general estimated well, indicating that the complete algorithm for the case without covariates manages to work as intended.

5.2.2 Breast Cancer Data (Boag [7], 1949)

Boag presented a data set with 121 breast cancer patients, taken from clinical records of a hospital over the years 1929 to 1938. It contains the survival times of the patients in months. The two competing risks in this data set are:

Cause 1: Breast cancer

Cause 2: Other

There are 25 right censored observations in this data set.

8 states have been used in the Phase-type fit, where states $1, ..., 6$ are transient and states 7 and 8 are absorbing. States 8 and 9 correspond to causes 1 and 2, respectively.

The hyper-parameters chosen to get these results are as follows for any transient state, i, and absorbing state j:

$$\begin{aligned} c &= 80 \\ d &= 50 \\ \tau_i &= 10 \\ \nu_{ij} &= 0.125 \\ \beta_i &= 0.1. \end{aligned}$$

The burn-in period is on 5000 iterations, and a sample of 1000 posterior estimates has been obtained.

Estimates of sub-distribution functions for both of the causes are plotted in figure 4. We see that the non-parametric estimates are very close to the Phase-type estimates, indicating that the Phase-type model gives good approximations for this data set.

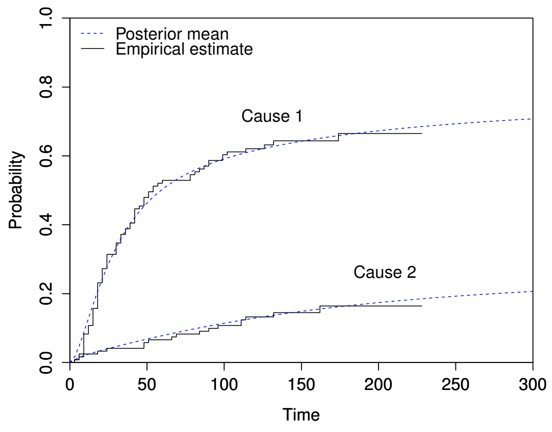
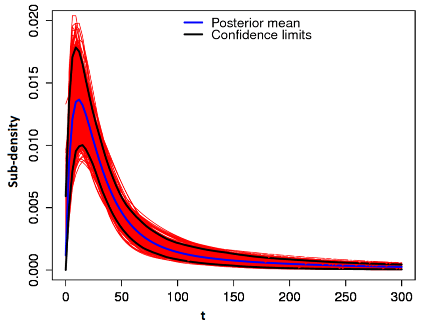


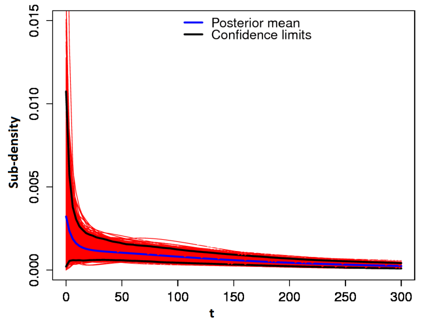
Figure 4: Sub-distribution estimates for the Breast cancer data

Estimates of sub-density functions are given i figure 5.

For cause 2 the sub-density estimate at time 0 has a large uncertainty, which is expected behavior for some cases.



Cause 1



Cause 2

Figure 5: Sub-density estimates for the Breast cancer data

At last, the cause-specific hazard estimates are plotted in figure 6. It is clear that the estimates of the cause-specific hazard rates have larger uncertainties than the estimates for the sub-densities, which is is especially true in the tails. This behavior will be discussed more later, as it seems to be a general behavior for all the cause-specific hazard rate estimates.

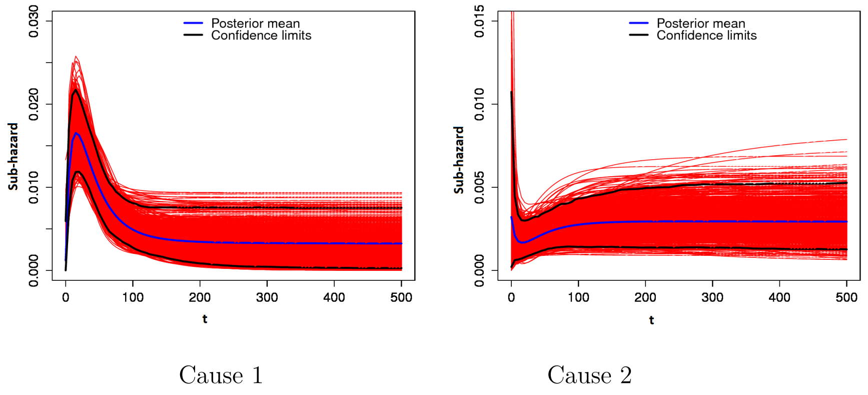


Figure 6: Sub-hazard estimates for the Breast cancer data

5.2.3 Data from Hoel

Hoel presented data from an experiment made on mice with two populations (Hoel 1972 [11], page 483). In this experiment both populations of mice received a radiation dose of 300r. Population 1 lived in a conventional laboratory environment, while population 2 lived in a germ free environment. The data for population 1 is analyzed in this text.

The data set is complete and consists of 99 observations with three possible causes of death:

Cause 1: Thymic Lymphoma

Cause 2: Reticulum Cell Sarcoma

Cause 2: Other Causes.

A Phase-type fit has been made with 10 states, where states $1, ..., 7$ are transient and states 8, 9 and 10 are absorbing. States 8, 9 and 10 correspond to causes 1, 2 and 3, respectively.

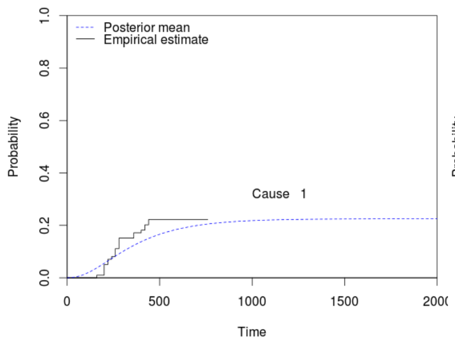
The hyper-parameters chosen to get these results are as follows for any transient state, i, and absorbing state j:

$$\begin{aligned} c &= 100 \\ d &= 10 \\ \tau_i &= 10 \\ \nu_{ij} &= 0.1 \\ \beta_i &= 0.01. \end{aligned}$$

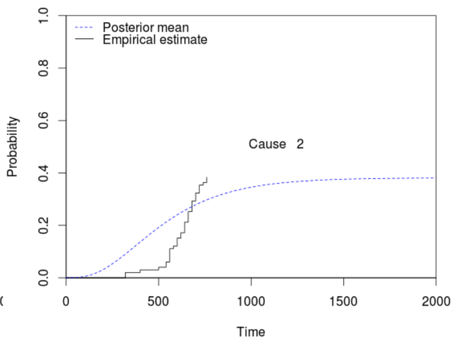
A burn-in period of 6000 iterations have been used, and a sample of 1000 posterior estimates have been sampled.

The plots of sub-distribution functions are given in figure 7.

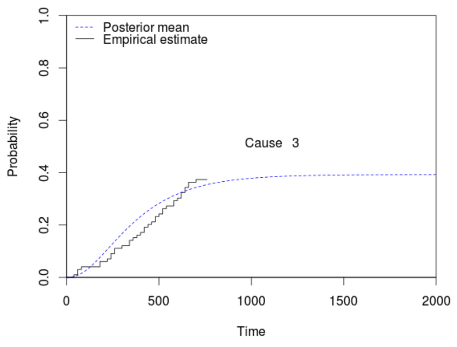
These plots indicate that the Phase-type model has difficulties approximating the sub-distribution functions of this data set, especially for causes 2 and 3. For cause 2 we see that the non-parametric curve has a steep rise around $t = 500$, and that the curve is almost zero for all time points before this.
This steep and sudden rise in the sub-distribution function is difficult to model with a Phase-type model, which has tail behavior like the exponential distribution, and normally has smooth sub-distribution behavior, with slow and gradual changes in the function values.
If the number of states used in the Phase-type fit increases, the fit will move closer to the empirical estimates, so a large model will for this data set probably give better functional estimates. Even so, a large increase in the number of model states seems to make the estimates only slightly better, and because of numerical and statistical instability in large systems it has proven more effective to obtain good estimates with a reasonably sized model.



Cause 1



Cause 2

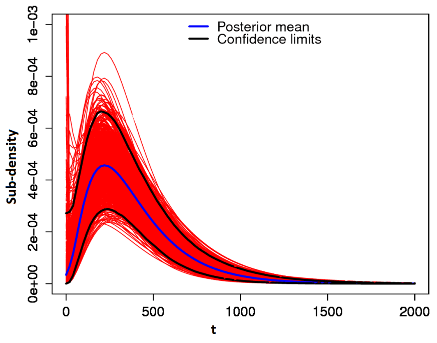


Cause 3

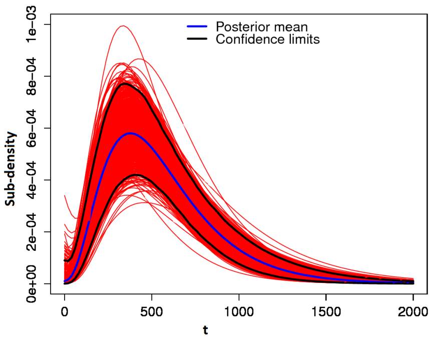
Figure 7: Sub-distribution estimates for the Hoel data

The plots of sub-density functions are given in figure 8.

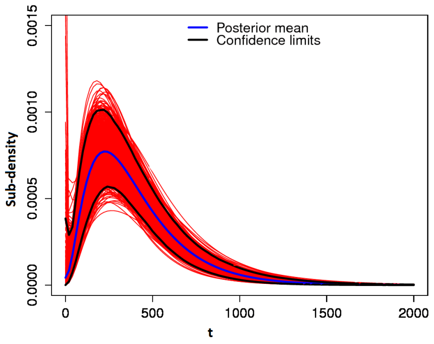
Here, the functional behaviors at $t = 0$ are clearly seen to be volatile, as $f_1(0)$ and $f_2(0)$ sometimes reaches very large values. It has been difficult to obtain estimates with more stable behavior at $t = 0$.



Cause 1



Cause 2

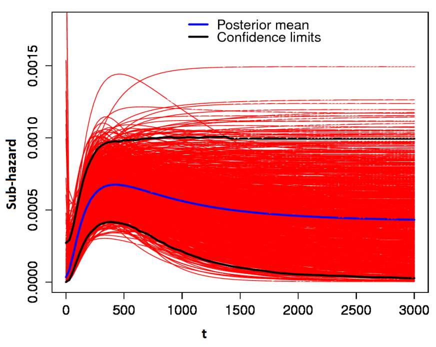


Cause 3

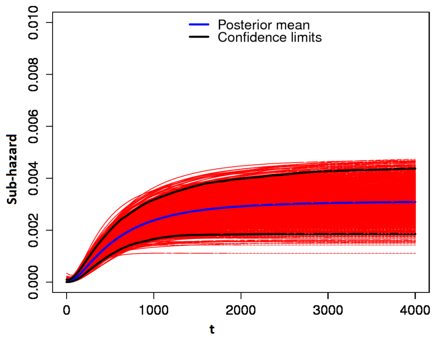
Figure 8: Sub-density estimates for the Hoel data

The plots of cause-specific hazard rates are given in figure 9.

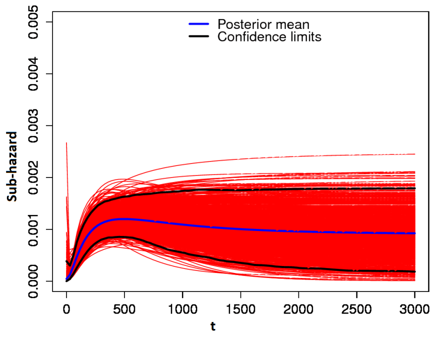
As for the data set presented by Boag, the tails of these estimates have large uncertainties.



Cause 1



Cause 2



Cause 3

Figure 9: Sub-hazard estimates for the Hoel data

Remarks

It is interesting to see how the cause specific hazard rates behave. The estimates for these are generally more unstable than the estimates for the sub-densities. A reason for this might be seen from the definition of the cause-specific hazard rate for cause j

$$h_j(t) = \frac{f_j(t)}{R(t)}.$$

This is a fraction of two functionals, and to estimate $h_j(t)$ we need to estimate these first. The estimation uncertainty for $f_j(t)$ will be amplified through the fractional expression, meaning that the estimation uncertainty for this expression becomes larger than for $f_j(t)$. The reason is that

$$R(t) \leq 1.$$

5.3 Results with covariates

In this section, three data sets have been analyzed with the Phase-type model. Two of the data sets are simulated and one is real data.

One of the simulated data sets consists of simulations from a Weibull-model with covariates introduced in an exponential expression which is multiplied with the cause-specific hazard rates, like in the Cox regression model.

The other simulated data set consists of simulations from a Phase-type model, which is a model on the form of (73), with cause-specific η-values. This is also the model the Phase-type algorithm uses to fit the data.

The real data set is from a study on the effect of chemotherapy on cancer patients.

For the data sets, non-parametric estimates of the sub-distribution functions have been produced with both the Fine & Gray method and the method based on the R-package *timereg*, for comparison with the Phase-type estimates.
For the real data, plots of the cause-specific hazard rates have been made as in the last section, by adding 95%-credibility intervals and the plots of the posterior estimates used to calculate the posterior mean.
Estimates of the regression coefficients, and corresponding standard errors, have been presented in tables, together with estimates made by Cox regression, the Fine & Gray method, and the method based on the *timereg* package.

The methods and code for estimating the posterior mean, credibility intervals and standard errors can be found in chapter 7.1, which is in the appendix.

Both the Fine & Gray method and the method in the *timereg* package have been explained briefly in the theory section, and detailed explanations on specific functions and code used in relation to these methods can be found in section 7.3.1 in the appendix.

As for the results without covariates, convergence has been ensured for all the following results using parallel runs of the Phase-type method, with different initial values for the parameters.

5.3.1 Phase-type data

This data set consists of 200 failures, with 10 right censored data points.

The simulating model, which the data is sampled from, is a model on the form of (73), where each covariate coefficient vector is constant for a given absorbing state, j, and does not depend on any transient states. This is the same model as the model which the Phase-type algorithm is based on.

Simulating the data points is done by simulating Markov chain realizations, and this is done by using the same method as in the *rejection procedure* in (45). The covariate values are also sampled from a distribution. Each entry in the covariate vector, w_k, for data point k, is sampled from the distribution

$$\text{Normal}\left(0, \frac{3}{2}\right).$$

All the code and functions used to simulate this data set is found in sections 7.2.2 and 7.3, together with the baseline intensity matrix (7.2.2).

The intensity matrix used to generate the data consists of 3 absorbing states and 7 transient states.

The following cause-specific regression coefficient vectors are used in the simulating model:

$$\eta_1 = \begin{bmatrix} -1, & -0.2 \end{bmatrix}$$

$$\eta_2 = \begin{bmatrix} -0.3, & -0.7 \end{bmatrix}$$

$$\eta_3 = \begin{bmatrix} 0, & 1.2 \end{bmatrix}$$

These define the coefficients in a model on the form of (73).

The censoring distribution is Weibull(10,10).

A Phase-type fit has been made with the same dimensions as the simulating model; 10 states where 7 are transient, and 3 are absorbing. States 8, 9 and 10 correspond to causes 1, 2 and 3, respectively.

The hyper-parameters chosen to get these results are as follows for any transient state, i, and absorbing state j:

$$\begin{aligned} c &= 500 \\ d &= 10 \\ \tau_i &= 10 \\ \nu_{ij} &= 0.1 \\ \beta_i &= 0.1 \\ \sigma &= 0.5. \end{aligned}$$

In the MH-methods for the regression coefficients, η_{jr}, step lengths are 0.01 for all the individual coefficients.

The burn-in period is on 5000 iterations, and a sample of 1001 posterior estimates has been obtained, by thinning out a sample of 5000 posterior estimates, choosing every 5th sample.

A trace-plot of all the η-estimates is shown in figure 10. The convergence of the η-estimates is observed from this trace-plot.

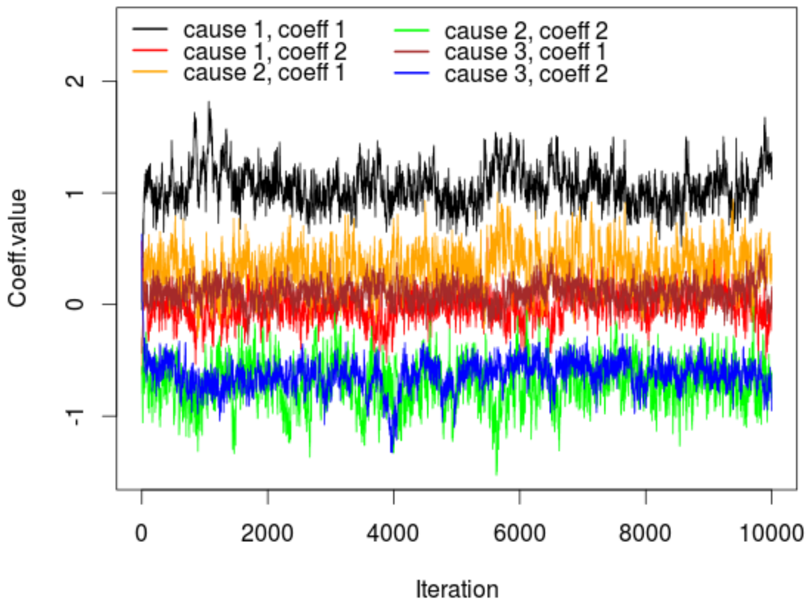
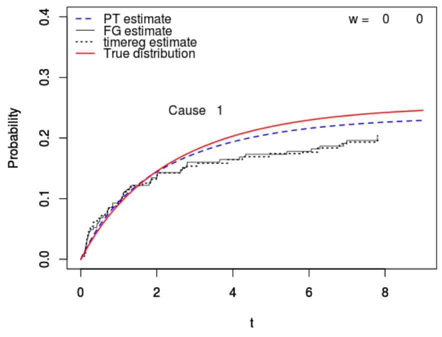


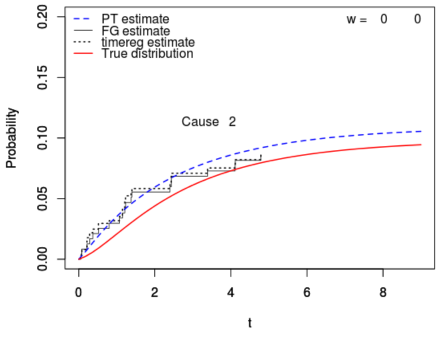
Figure 10: Trace-plot of η-estimates for Phase-type data

The baseline sub-distribution functions for causes 1, 2 and 3 are plotted in figure 11.

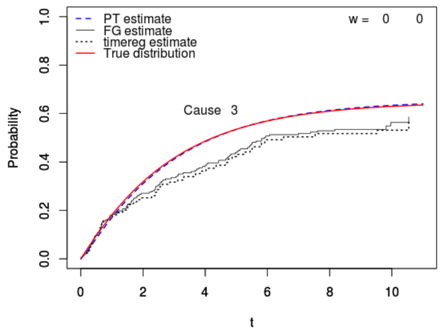
It is evident that the Phase-type fit approximates the underlying sub-distribution functions better than the non-parametric methods. This is thus to be expected, since the Phase-type fit is based on a model on the same form as the underlying model.
The non-parametric methods tend to under-estimate these sub-distribution functions, most probably because they are both based on regression through the sub-distributions, which differs a lot from the theoretical covariate model used to generate the data. The fact that the non-parametric methods are both based on regression through the sub-distribution functions, is probably also the reason why the estimates from these methods are very similar.



Cause 1



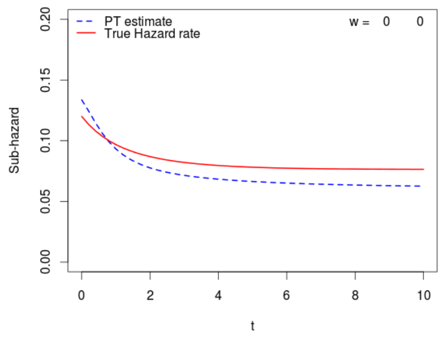
Cause 2



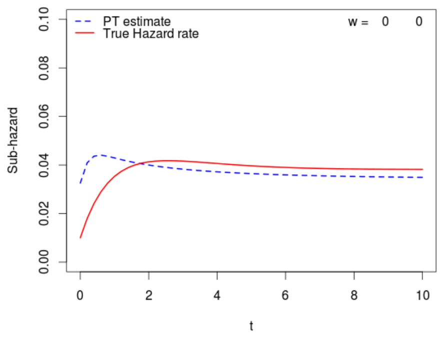
Cause 3

Figure 11: Baseline sub-distribution estimates for the Phase-type data with covariates

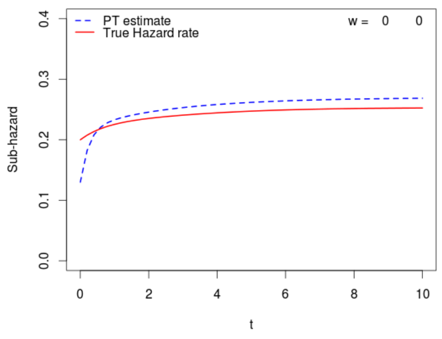
The baseline cause-specific hazard rates are plotted in figure 12. The estimates are close to the theoretical functions.



Cause 1



Cause 2



Cause 3

Figure 12: Baseline sub-hazard estimates for the Phase-type data with covariates

Estimates of the vectors η_1, η_2 and η_3 are presented in table 2. We see that the estimates produced with the Phase-type model and Cox regression are similar, and close to the theoretical values. The Cox estimates seem to have smaller standard errors, indicating that despite the similarities between the two methods, they are essentially based on different models.
It is worth mentioning that the Phase-type estimates should be closest to the theoretical values, since the Phase-type model is also the model used to generate the data. Still, it is difficult to decide if Cox regression or the Phase-type method give the closest estimates.
The estimates from the non-parametric methods are also similar, and they deviate from the other estimates and the theoretical values. As for the estimates of the sub-distribution functions, a reason for this is probably that they are based on very different models than the two other methods, and performs regression through transformations of the sub-distribution functions.

	True val.	Cox regr.	Phase-type	Fine & Gray	timereg
η_1^T	$\begin{bmatrix}1\\-0.2\end{bmatrix}$	$\begin{bmatrix}0.9069\\-0.0538\end{bmatrix}$	$\begin{bmatrix}1.060437902\\0.009915118\end{bmatrix}$	$\begin{bmatrix}0.7846\\0.2275\end{bmatrix}$	$\begin{bmatrix}0.819\\0.135\end{bmatrix}$
s.e	-	$\begin{bmatrix}0.117\\0.104\end{bmatrix}$	$\begin{bmatrix}0.1742115\\0.1388436\end{bmatrix}$	$\begin{bmatrix}0.1048\\0.0995\end{bmatrix}$	$\begin{bmatrix}0.130\\0.115\end{bmatrix}$
η_2^T	$\begin{bmatrix}0.3\\-0.7\end{bmatrix}$	$\begin{bmatrix}0.323\\-0.755\end{bmatrix}$	$\begin{bmatrix}0.3630679\\-0.6950387\end{bmatrix}$	$\begin{bmatrix}0.0303\\-0.3576\end{bmatrix}$	$\begin{bmatrix}0.137\\-0.302\end{bmatrix}$
s.e	-	$\begin{bmatrix}0.182\\0.199\end{bmatrix}$	$\begin{bmatrix}0.1935263\\0.2006831\end{bmatrix}$	$\begin{bmatrix}0.1317\\0.1238\end{bmatrix}$	$\begin{bmatrix}0.189\\0.184\end{bmatrix}$
η_3^T	$\begin{bmatrix}0\\-0.5\end{bmatrix}$	$\begin{bmatrix}0.107\\-0.536\end{bmatrix}$	$\begin{bmatrix}0.1062896\\-0.5987580\end{bmatrix}$	$\begin{bmatrix}-0.2446\\-0.2557\end{bmatrix}$	$\begin{bmatrix}-0.184\\-0.387\end{bmatrix}$
s.e	-	$\begin{bmatrix}0.0726\\0.0814\end{bmatrix}$	$\begin{bmatrix}0.1047259\\0.1095020\end{bmatrix}$	$\begin{bmatrix}0.06333\\0.07566\end{bmatrix}$	$\begin{bmatrix}0.0728\\0.0794\end{bmatrix}$

Table 2: Table of η-estimates for the Phase-type data with covariates

5.3.2 Weibull data

This data set consists of 150 observations, with 58 censorings and 92 failures.

Each data point can have 2 causes of failure and has been generated by sampling a latent failure time for each cause, selecting the minimum of these latent failure times as the failure time, and selecting the cause corresponding to this latent failure time as the cause in the data point.
The code for simulating this data set can be found in the back of section 7.3 in the appendix.

The latent failure time distribution for cause j is Weibull$(2, \beta_j)$, where 2 is the *shape* parameter and β_j is the *scale* parameter. β_j is depending on the covariates, and is also cause-specific. For data point k, it is on the following form:

$$\beta_j = \sqrt{2} \cdot \exp\left(-\frac{1}{2} w_k \eta_j^T\right), \tag{88}$$

where w_k is the covariate vector for data point k, and η_j is a cause-specific coefficient vector.

Choosing a *scale* parameter on the form above will give hazard rates for the latent failure time distributions like

$$h_j(t) = t \cdot \exp\left(w_k \eta_j^T\right).$$

Thus, the simulated data is from a Cox regression model with linear baseline sub-hazard rates. These hazard rates can be difficult to estimate with a Phase-type model, since all Phase-type models have asymptotically constant hazard rates.

The following cause-specific coefficient vectors are used in this simulating model:

$$\eta_1 = \begin{bmatrix} 0.9, & -0.8 \end{bmatrix}$$

$$\eta_2 = \begin{bmatrix} 0.4, & -1.3 \end{bmatrix}.$$

The censoring distribution is Weibull$\Big(shape = \frac{3}{2}, scale = 2\Big)$, and the distribution used to obtain the covariate values in the data is Normal$\Big(0, \frac{3}{2}\Big)$.

An important remark must be made about the cause-specific hazard rates for this simulating model. The way the data is simulated is by sampling latent failure times, T_j, for each failure type, j, and interpreting the hazard rates of these as the cause-specific hazard rates of the underlying competing risks model. As Lindqvist writes in his paper on competing risks, this way of interpreting the hazard rates of T_j is only possible if the T_j are independent of each other (Lindqvist [14], page 5). This is due to the identifiability problem in competing risks. If T_j would not be independently sampled, like they have been for this data set, the marginal hazard rates of each T_j would generally not be equal to the cause-specific hazard rates.

A Phase-type fit has been made with 10 states, where states $1, ..., 8$ are transient and states 9 and 10 are absorbing. States 9 and 10 correspond to causes 1 and 2, respectively.

The hyper-parameters chosen to get these results are as follows for any transient state, i, and absorbing state j:

$$\begin{aligned} c &= 2000 \\ d &= 10 \\ \tau_i &= 10 \\ \nu_{ij} &= 0.1 \\ \beta_i &= 1 \\ \sigma &= 0.4. \end{aligned}$$

In the MH-methods for the regression coefficients, η_{jr}, step lengths are 0.01 for all the individual coefficients.

The burn-in period is on 1000 iterations, and a sample of 1000 posterior estimates has been obtained.

A trace-plot of all the η-estimates is shown in figure 13. The η-values seem stable, which indicates convergence.

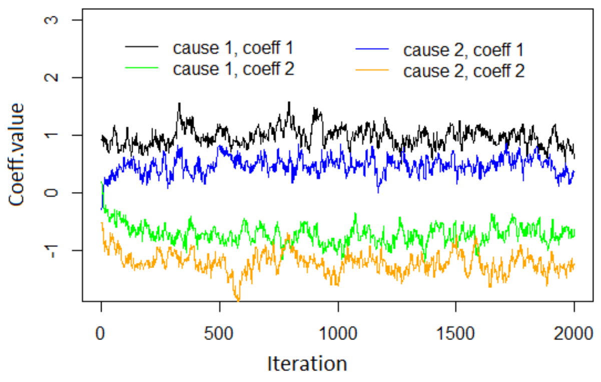
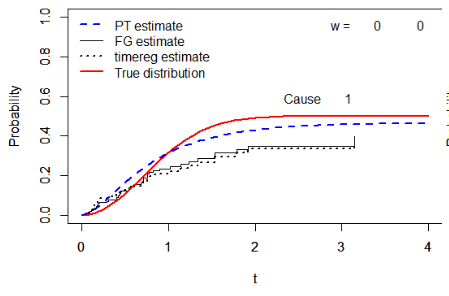
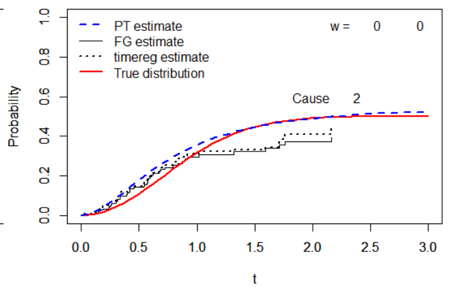


Figure 13: Trace-plot of η-estimates for Weibull data

The baseline sub-distribution functions for causes 1 and 2 are plotted in figure 14.

The Phase-type model gives good approximations for the baseline sub-distributions. As for the Phase-type data, the non-parametric methods give very similar estimates, and they seem to underestimate the baseline sub-distributions. The reason for this behavior is probably roughly the same as for the Phase-type data; the non-parametric methods are based on similar models which are different from the underlying model and the Phase-type model.

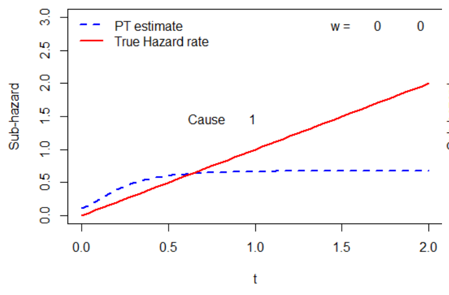


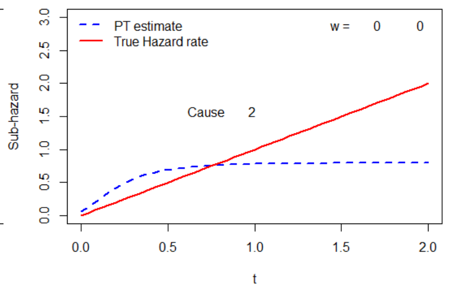


Cause 1 Cause 2

Figure 14: Baseline sub-distribution estimates for Weibull data

Estimates of the cause-specific hazard rates have also been produced. The baseline hazard rates are given in figure 15. The plots show that the weibull-model is quite different from the Phase-type model. The weibull-model has hazard rates which are linear in time, seen from expression (88), not constant as the Phase-type hazard rates are asymptotically. It is thus impossible to achieve good asymptotic estimates of the hazard rates for the Weibull data, using the Phase-type model.





Cause 1 Cause 2

Figure 15: Baseline sub-hazard estimates for Weibull data

At last, estimates of the vectors η_1 and η_2 are presented in table 3. The pairwise similarities between the Cox regression- and Phase-type estimates, and the Fine & Gray- and *timereg* estimates, are quite striking.
As before, differences in the covariate influence on the sub-distribution functions and the cause-specific hazard rates are probably the reasons why the non-parametric methods give different results than the other methods and the underlying model.

It is difficult to decide whether the Cox regression- or Phase-type estimates are overall closer to the real parameter values used to simulate the data.
Because the simulated data are from a proportional hazards model, the Cox regression estimates would be expected to be closest to the real parameter values.

	True val.	Cox regr.	Phase-type	Fine & Gray	timereg
η_1^T	$\begin{bmatrix} 0.9 \\ -0.8 \end{bmatrix}$	$\begin{bmatrix} 1.014 \\ -0.916 \end{bmatrix}$	$\begin{bmatrix} 0.9486004 \\ -0.7398985 \end{bmatrix}$	$\begin{bmatrix} 0.7287 \\ -0.1881 \end{bmatrix}$	$\begin{bmatrix} 0.699 \\ -0.165 \end{bmatrix}$
s.e	-	$\begin{bmatrix} 0.141 \\ 0.155 \end{bmatrix}$	$\begin{bmatrix} 0.1432722 \\ 0.1528616 \end{bmatrix}$	$\begin{bmatrix} 0.1230 \\ 0.0975 \end{bmatrix}$	$\begin{bmatrix} 0.134 \\ 0.107 \end{bmatrix}$
η_2^T	$\begin{bmatrix} 0.4 \\ -1.3 \end{bmatrix}$	$\begin{bmatrix} 0.482 \\ -1.334 \end{bmatrix}$	$\begin{bmatrix} 0.4774822 \\ -1.2491768 \end{bmatrix}$	$\begin{bmatrix} -0.1219 \\ -0.6400 \end{bmatrix}$	$\begin{bmatrix} -0.0943 \\ -0.6090 \end{bmatrix}$
s.e	-	$\begin{bmatrix} 0.133 \\ 0.162 \end{bmatrix}$	$\begin{bmatrix} 0.1431508 \\ 0.1566439 \end{bmatrix}$	$\begin{bmatrix} 0.09394 \\ 0.11770 \end{bmatrix}$	$\begin{bmatrix} 0.118 \\ 0.139 \end{bmatrix}$

Table 3: Table of η-estimates for the Weibull data

5.3.3 Follicular cancer data

This data set is from a study on follicular cell lymphoma, and was also analysed by Scheike and Zhang in their paper on the *timereg* package in R (Scheike and Zhang [18], page 4).

It can be downloaded from
http://www.uhnres.utoronto.ca/labs/hill/datasets/Pintilie/datasets/follic.txt.

Originally, this data set was presented in a paper by Pintilie in 2007 [15].

The data set consists of 541 patients having early disease stage *follicular cell lymphoma*, which are treated with radiation therapy alone, or with a combination treatment of radiation- and chemotherapy. This is modeled by the binary covariate, *Chemo*, which is 0 for radiation treatment alone, and 1 for the combination treatment.

The competing risks are

$$\text{I: Disease relapse or no response to therapy}$$

$$\text{II: Death in remission}$$

There are 272 events of type I and 76 events of type II, in addition to 193 censored events.

The covariates in this data set are

$$Age : \text{Covariate 1, continuous variable}$$

$$Hgb : \text{Covariate 2, continuous variable}$$

$$Stage : \text{Covariate 3, discrete variable, 0 or 1}$$

$$Chemo : \text{Covariate 4, discrete variable, 0 or 1.}$$

Age is the subject age when treatment is started, *Hgb* represents the level of

Hemoglobin in the blood of a subject, *Stage* represents the stage of cancer development in a subject, and *Chemo* represents which treatment the subject undergoes.

In the data, *Hgb* takes a value between 100 and 200 for all the subjects, but in the Phase-type analysis the values of *Hgb* are scaled down with a factor of 100, such that the values for *Hgb* in practice becomes values between 1 and 2. The reason for this is that the model converges poorly for the unscaled case. Scaling down *Hgb* improves the mixing of its corresponding coefficients in the MCMC method. The same is true for the covariate *Age*, though now only a factor of 10 is needed to downscale the covariate values. The unscaled values of *Age* ranges from 17 to 86, while the scaled values ranges from 1.7 to 8.6.

Since the covariates in the data are scaled down, the η-estimates will be correspondingly larger than before scaling.

A Phase-type fit has been made with 10 states, where states $1, ..., 8$ are transient and states 9 and 10 are absorbing. States 9 and 10 correspond to type I and II events, respectively.

The hyper-parameters chosen to get these results are as follows for any transient state, i, and absorbing state j:

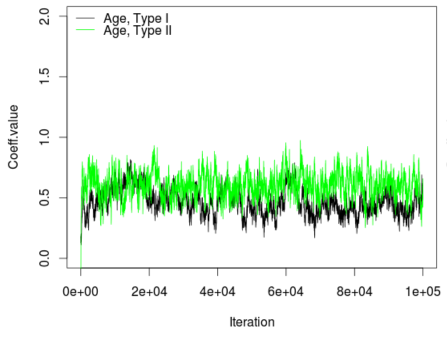
$$\begin{aligned} c &= 100 \\ d &= 10 \\ \tau_i &= 10 \\ \nu_{ij} &= 0.1 \\ \beta_i &= 1 \\ \sigma &= 0.3. \end{aligned}$$

In the MH-methods for the regression coefficients, η_{jr}, step lengths are 0.01 for all the individual coefficients, except the coefficients for covariate *Age*, which have step-lengths 0.001.

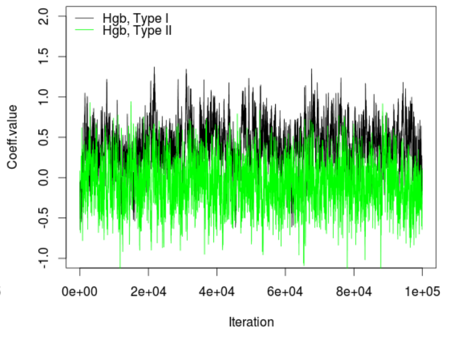
The burn-in period is on 90000 iterations, and a sample of 1001 posterior estimates for the parameter values have been obtained. The sample is obtained by thinning out a sample of 10000 posterior estimates, choosing every 10th posterior estimate as a sample value.

Trace-plots of all the η-estimates are shown in figure 16. Stability of the η-values

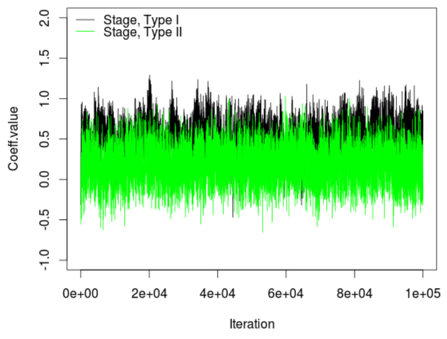
is evident.
The η-values for the covariate *Age* seem to locally fluctuate slightly, indicating that auto-correlation is high. This is the reason why a large sample with thinning and a large burn-in period of 90000 iterations has been used. It should be mentioned that the burn-in period is very large, and this is probably not necessary to reach convergence. From the trace-plots it seems sufficient with a burn-in on only a few thousand iterations.



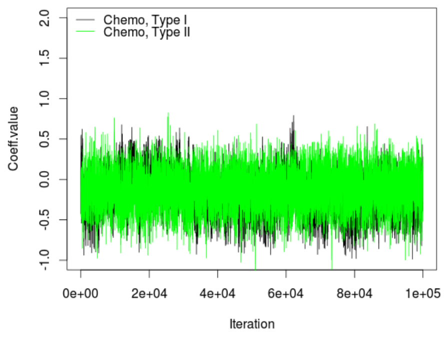
Age



Hgb



Stage



Chemo

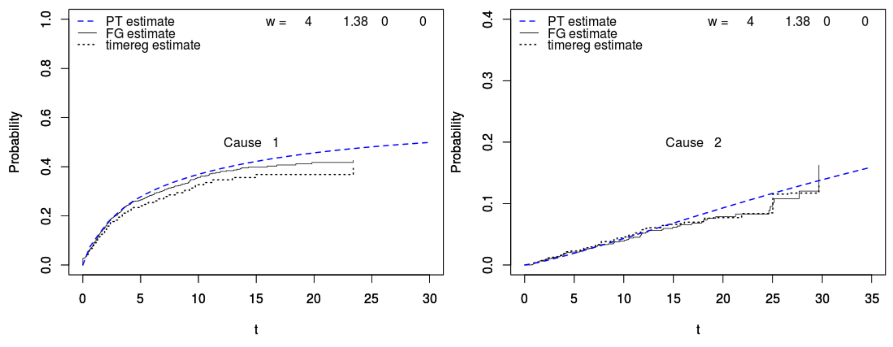
Figure 16: Trace-plots of the η-values for *follicular cancer* data

Plots of the sub-distribution estimates made by the Phase-type method, together with Fine & Gray- and *timereg* estimates, have been obtained for the covariate vector

$$w = \left[4,\ 1.38,\ 0,\ 0\right].$$

This is the scaled down version of the covariate vector used in the paper by Scheike and Zhang [18]. The plots are in figure 17.

It is interesting to see that all the estimation methods give similar estimates, and because the existing estimation methods are known to work reasonably well, this indicates that the Phase-type model gives a good fit for the sub-distribution functions.



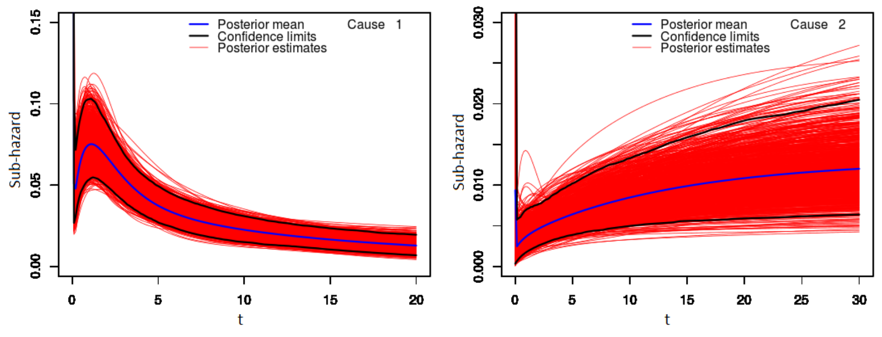
Type I Type II

Figure 17: Sub-distribution estimates for *follicular cancer* data, with covariate $[4, 1.38, 0, 0]$

Estimates of the cause-specific hazard rates are given in figure 18.

All the sampled posterior estimates are also plotted, together with 95%-credibility intervals, to give a view of the uncertainty in the estimation. The uncertainty is high in the type II estimate, probably by the same reasons as explained for the cause-specific hazard rates in the results without covariates.
The hazard rates act strangely around $t = 0$, as the rates seem to be discontinuous in this region. This behavior is probably due to some specific failure times

in the data. In the data set, there exists 24 failure times with the same size, 0.002737851. These failure times force the hazard rates to become unnaturally high around $t = 0$. Taking them out of the data set would give estimates of cause-specific hazard rates very similar to the ones in figure 18, but without the discontinuous behavior around $t = 0$.



Type I Type II

Figure 18: Sub-hazard estimates for *follicular cancer* data, with covariate $[4, 1.38, 0, 0]$

Finally, estimates of the η-values are presented in table 4. Here, we see that the Phase-type estimates have slightly larger standard errors than the Cox regression estimates, but that the same tendencies are present in the estimates for both these methods.

For this data set, the Phase-type estimates and the Cox regression estimates deviate more than for the simulated data. It seems like the data set has underlying properties that manage to capture the theoretical differences between the Cox regression model and the Phase-type model.

For the estimates of both the Fine & Gray and *timereg* models, all results show the same tendencies, and they also have roughly the same tendencies as the Cox regression an Phase-type estimates. Thus, for this data set it seems like the covariates influence all the models similarly.

	Cox regr.	Phase-type	Fine & Gray	timereg
η_1^T	$\begin{bmatrix} 0.230 \\ 0.228 \\ 0.566 \\ -0.302 \end{bmatrix}$	$\begin{bmatrix} 0.4344674 \\ 0.3960603 \\ 0.5777734 \\ -0.2457615 \end{bmatrix}$	$\begin{bmatrix} 0.173 \\ 0.232 \\ 0.557 \\ -0.332 \end{bmatrix}$	$\begin{bmatrix} 0.1940 \\ 0.0321 \\ 0.6690 \\ -0.3070 \end{bmatrix}$
s.e	$\begin{bmatrix} 0.0473 \\ 0.4086 \\ 0.1323 \\ 0.1664 \end{bmatrix}$	$\begin{bmatrix} 0.08379612 \\ 0.28312458 \\ 0.19135806 \\ 0.20112267 \end{bmatrix}$	$\begin{bmatrix} 0.0479 \\ 0.3982 \\ 0.1350 \\ 0.1729 \end{bmatrix}$	$\begin{bmatrix} 0.0507 \\ 0.4120 \\ 0.1380 \\ 0.1750 \end{bmatrix}$
η_2^T	$\begin{bmatrix} 0.868 \\ 0.417 \\ 0.488 \\ -0.116 \end{bmatrix}$	$\begin{bmatrix} 0.5843109 \\ -0.1546399 \\ 0.1595203 \\ -0.1230117 \end{bmatrix}$	$\begin{bmatrix} 0.4726 \\ -0.6202 \\ -0.0416 \\ -0.3026 \end{bmatrix}$	$\begin{bmatrix} 0.444 \\ -0.558 \\ -0.167 \\ 0.181 \end{bmatrix}$
s.e	$\begin{bmatrix} 0.114 \\ 0.835 \\ 0.282 \\ 0.355 \end{bmatrix}$	$\begin{bmatrix} 0.09589716 \\ 0.24868610 \\ 0.21065539 \\ 0.22730924 \end{bmatrix}$	$\begin{bmatrix} 0.0872 \\ 0.8636 \\ 0.2420 \\ 0.3446 \end{bmatrix}$	$\begin{bmatrix} 0.118 \\ 0.988 \\ 0.318 \\ 0.437 \end{bmatrix}$

Table 4: Table of η-estimates for the *follicular cancer* data

6 General discussion and conclusion

The Phase-type model seems to work very well for many different data sets and underlying distributions, and manage to produce good estimates of the sub-distribution functions, sub-density functions and the covariate regression coefficients.
For the cause-specific hazard rates it is possible to obtain good estimates with reasonably sized data sets, as in the results presented here, but if the data sets become too small or the number of censored data points become too large, experience shows that cause-specific hazard rates might be poorly estimated.
The estimates of the cause-specific hazard rates also seem to have larger estimation uncertainties, especially in the tails. This is due to the fact that these functions are fractional expressions with the survival function, $R(t)$ in the denominator, and this function is never larger than 1 and decreases with time.
The Phase-type model is flexible, but it has its limitations, which is why this method has difficulties approximating more general underlying models, like the Weibull model or the theoretical model for the Hoel data set.

Sometimes estimates of the sub-distribution functions and the η-values for the non-parametric methods are different from the Phase-type estimates and the Cox regression estimates. It seems like this is especially true when the data is from an underlying model where the covariates are heavily influencing the risks of absorption, like in both of the simulated data sets. In these situations the sub-distribution functions are probably influenced very differently by the covariates than the absorbing intensities and the cause-specific hazard rates, which are used for covariate regression in the Phase-type model and Cox regression respectively. Both the non-parametric methods are based on models where covariate regression is performed through transformations of the sub-distribution functions, and this might thus be the reason for the different estimates in these situations.

The estimates of the η-values are similar to the estimates produced by Cox regression. A reason for this is probably that both the methods perform covariate regression through parameters and functions that influence the failure rates of the different failure types, and that the covariates are introduced using equal regression expressions.

The method used to fit the model works well, but it is an inevitable fact that a lot of the parameters in the method must be chosen with care, such that the method converges within reasonable time and produces parameter samples of good quality.
For some data sets convergence is reached after a very large number of iterations, or after a very long time. A way to deal with this problem is simply optimizing the

hyper-parameters, choosing appropriate step-lengths in the MH-samplers for the η-values, and choosing an appropriate number of iterations for the Markov chain realization samplers to be run at each Gibbs-step.
When the problem seems to be that auto-correlation for the η-values is too high, one might also improve the run-time considerably by appropriately scaling the covariates in the data.

The size of the Phase-type fit can be chosen, and a larger sized model will have larger potential of approximating more general competing risks distributions. Still, it should be mentioned that the size of the Phase-type fit has a substantial impact on the run-time of the algorithm, meaning that a larger model increases the run-time. Also, numerical instabilities in the algorithm might prevent extremely large Phase-type models to be fitted. This rarely pose any problems though, since large enough Phase-type models are usually numerically attainable.

The results in this text show that the Phase-type model, although having its limitations, can be used as a powerful tool to approximate the underlying models of competing risks data, with or without covariates. It is especially interesting to see that it often manage to estimate the cause-specific hazard functions. To this day, it exists no other efficient methods capable of doing this.

Further work on this area could be centered around optimizing the MCMC method, making it more efficient both numerically and statistically. This can partly be done by finding optimal strategies to define hyper-parameters and other parameters needed to run the method. In addition, there is a lot of potential in optimizing the code, such that the algorithm runs faster and more smoothly.

It is also possible to theoretically improve the method, by introducing new concepts that simplify parts of the algorithm. The *rejection procedure* in (45) is a part of the algorithm which can be viewed as a bottleneck, because it can spend too much time sampling the Markov chain realizations. Aslett & Wilson proposed a different way to sample these Markov chain realizations in their doctoral dissertation (Aslett & Wilson [3], pages 78-86), and introducing these concepts in this method might result in substantial efficiency improvements.

The method proposed in this text is just one example on how Phase-type models can be used to make inference on competing risks data. The model in (73) was chosen to be implemented, but a more general covariate model, like a combination of (73) and (72) is theoretically more flexible. An efficient method based on such a model might thus have a larger potential for producing good estimates. Even so, it should be mentioned that a model like this would contain a very large number of parameters to be estimated, so the quality of the estimates might suffer from this.

7 Appendix

7.1 Estimation

7.1.1 Function calculations

When we talk about estimating functions, we essentially talk about estimating each functional value at a specified set of time points, t.
This is done by calculating functional expressions for every parameter sample, k, at every t, and then calculating the posterior mean at these time points. For each k, calculating the functional expression at time t is done by first obtaining every sample observation of parameters, denoted as $L^{(k)}$, $Q^{(k)}$, $\mathbf{p^{(k)}}$ for the situation without covariates, and $l^{(k)}$, $Q^{(k)}$, $\mathbf{p^{(k)}}$ and $\eta_j^{(k)}$ for the situation with covariates. Then we calculate the functional value using the Phase-type expressions in chapter 2.3, and model (73) if covariates are present.

Three different functions have been estimated in this text, and their expressions are based on (34), (33) and (35), in addition to (73) if covariates are in the model. They are calculated in the following way for each posterior sample, k:

Case without covariates:

$$f_j(t):\quad \mathbf{p}^{(k)} e^{tQ^{(k)}} L^{(k)} v_j^T \tag{89}$$

$$F_j(t):\quad \mathbf{p}^{(k)} \mathrm{Inv}\left(Q^{(k)}\right)\left(e^{tQ^{(k)}} - I\right) L^{(k)} v_j^T$$

$$\lambda_j(t):\quad \frac{\mathbf{p}^{(k)} e^{tQ^{(k)}} L^{(k)} v_j^T}{p e^{Qt}\mathbf{1}}$$

If covariates are present, it follows from model (73) that the matrix entries of $L^{(k)}$ are on the form

$$L_{ij}^{(k)} = l_{ij}^{(k)} \exp\left(\eta_j^{(k)} w^T\right),$$

for every transient state i and absorbing state j. Here w is as usual the covariate vector, and l_{ij} is the baseline transition intensity.

$L^{(k)}$ and $Q^{(k)}$ (because of the diagonal intensities) are then calculated first using this expression, and then they are used in the expressions in (89).

The expressions above have been used directly in the code when calculating the function estimates.

7.1.2 Posterior mean

This is the method used for estimating the functional values. The method consists of first obtaining samples for the target function, $f(t)$, for each chosen time point, t, using the expressions in (89) for each parameter sample. At each t the mean of the corresponding functional value sample is calculated and this represents the posterior mean estimate for $f(t)$. When this is done for all the time points chosen, the complete function estimate has been obtained.

7.1.3 Posterior standard deviation

The standard deviance of the posterior distribution for a parameter value, x, can be estimated by simply using the ordinary standard deviance estimator given by

$$s = \frac{1}{N-1} \sum_{k=1}^{N} \left(x_k - \bar{x} \right),$$

where N is the number of posterior estimates, x_k is the value of posterior estimate i and $\bar{x}$ is the posterior mean of x.

This is the method used to estimate the standard deviations of the η-values in the results.

7.1.4 Credibility intervals

Estimating the credibility intervals for a function, $f(t)$, is done for each evaluated time point, t. At each t we first create a sample of functional values for $f(t)$ by calculating the posterior functional estimates using the expressions in (89). The posterior functional estimate for sample observation k is denoted as $f_k(t)$. Then we exclude 2.5% of the most extreme values of $f_k(t)$ in each end of the sample. The new extreme estimates in each end of the remaining sample can be viewed as the estimated 95%-credibility limits.

7.2 Simulation of data

7.2.1 Coxian model without covariates

This data set has been simulated from a Markov chain with the intensity matrix

$$A = \begin{bmatrix} -0.3 & 0.2 & 0.1 & 0 \\ 0 & -0.6 & 0.5 & 0.1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

and initial distribution

$$p = \begin{bmatrix} 1, & 0, & 0, & 0 \end{bmatrix}.$$

Each entry, k, in this data set has been calculated by performing the following algorithm described:

Basic Phase-type simulating algorithm, without censoring:

1 Sample initial state from given discrete initial distribution

2 With the sampled initial state, simulate a Markov chain realization up to absorption, based on the intensity matrix above. The simulation method consists of sampling visiting times for visits in states i from the distributions $\mathrm{Exp}(-a_{ii})$, and sampling state transitions with the discrete distributions $\left(\frac{a_{i1}}{-a_{ii}}, \ldots, \frac{a_{i(i-1)}}{-a_{ii}}, \frac{a_{i(i+1)}}{-a_{ii}}, \ldots, \frac{a_{i(K+m)}}{-a_{ii}}\right)$.

3 Set the time to absorption in this Markov chain realization as the failure time, x_k, and the final absorbing state as the competing risks failure type, c_k. (x_k, c_k) constitutes data point k.

7.2.2 Phase-type model with covariates

This data set has been simulated from a Markov chain with the baseline intensity matrix

$$A = \begin{bmatrix} -1.13 & 0.40 & 0.05 & 0.20 & 0.10 & 0.010 & 0.040 & 0.120 & 0.01 & 0.20 \\ 0.10 & -0.94 & 0.20 & 0.05 & 0.05 & 0.010 & 0.080 & 0.050 & 0.10 & 0.30 \\ 0.03 & 0.01 & -0.81 & 0.10 & 0.07 & 0.220 & 0.300 & 0.000 & 0.02 & 0.06 \\ 0.20 & 0.03 & 0.01 & -0.91 & 0.30 & 0.020 & 0.010 & 0.090 & 0.05 & 0.20 \\ 0.02 & 0.30 & 0.06 & 0.01 & -0.93 & 0.100 & 0.030 & 0.200 & 0.01 & 0.20 \\ 0.20 & 0.10 & 0.01 & 0.03 & 0.04 & -0.945 & 0.100 & 0.045 & 0.02 & 0.40 \\ 0.20 & 0.10 & 0.01 & 0.03 & 0.04 & 0.100 & -0.945 & 0.045 & 0.02 & 0.40 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.000 & 0.000 & 0.000 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.000 & 0.000 & 0.000 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.000 & 0.000 & 0.000 & 0.00 & 0.00 \end{bmatrix},$$

and initial distribution

$$p = \begin{bmatrix} 1, & 0, & 0, & 0, & 0, & 0, & 0 \end{bmatrix}.$$

In addition, the cause-specific regression coefficient vectors are given in the results section. These vectors are used to express the absorbing intensities as in (73).

Each entry, k, in this data set has been calculated by performing the algorithm below.

The covariate Phase-type simulating algorithm :

1 Sample initial state from given discrete initial distribution

2 Sample a covariate realization, w_k, from the covariate distribution given in the results section.

3 Using the sampled initial state and realization, simulate a Markov chain realization up to absorption, based on the covariate Phase-type model described above. The simulation method consists of sampling visiting times for visits in states i from the distributions $\text{Exp}(-a_{ii})$, and sampling state transitions with the discrete distributions $\left(\frac{a_{i1}}{-a_{ii}}, ..., \frac{a_{i(i-1)}}{-a_{ii}}, \frac{a_{i(i+1)}}{-a_{ii}}, ..., \frac{a_{i(K+m)}}{-a_{ii}}\right)$.

4 Sample a realization, t_0, from the given censoring distribution in the results chapter.

5 Set the event time in the data point, x_k, equal to $x_k = \min(t, t_0)$. Set the event code in the data point, c_k, equal to the absorbing state in the Markov chain realization if $t < t_0$, and set it equal to the preferred censoring code if $t_0 < t$. The final data point is (x_k, c_k, w_k).

7.2.3 Weibull model with covariates

This data set has been simulated based on a proportional hazards Weibull model. The method used is to sample latent failures, T_j, for causes $j = 1, 2$.
T_j is from a Weibull$(2, \beta_j)$-distribution, where β_j is the scale parameter given as in the results section.

For each data point, k, two latent failure times, T_1 and T_2, are sampled from their corresponding distributions, in addition to the censoring time, T_0, which is sampled from the censoring distribution given in the results section.
The event time, x_k is defined as

$$x_k = \min\left(T_0, T_1, T_2\right),$$

and the event, c_k, is defined as

$$c_k = \operatorname{argmin}\left(T_0, T_1, T_2\right),$$

where 0 is the censoring code.

The covariate value, w_k is sampled from the covariate distribution given in the results section.

7.3 Code

Here, the most important code is presented, such that it is possible to reproduce the results in this text, and to apply the methods on other data sets.
The code used to apply the functions *cuminc*, *crr*, *comp.risk* and *coxph* in the R-packages *cmprsk*, *timereg* and *survival* is presented first.
After this, the implemented estimation methods used to produce the results are presented briefly.
In the end, the complete MCMC methods for both the cases with and without

covariates are presented, along with the needed support-functions.

It is worth mentioning that the packages needed to run this code are the following:

- *MCMCpack* - for using the function *rdirichlet*, which samples from the dirichlet distribution
- *msm* - for using the function *MatrixExp*, which calculates the matrix exponential
- *survival* - for using the function *coxph*, which runs Cox regression on lifetime data
- *timereg* - for using the function *comp.risk*, used for competing risks regression
- *cmprsk* - for using the functions *cuminc* and *crr*, used to estimate subdistribution functions and implementing the Fine & Gray method, respectively

Many parts of the code are depending on the data set used in the Phase-type method, so the way to represent the data sets in the code must be shown before moving on to the other code.

A specific data set is represented by the matrix x, where each row represents a data point. The failure times are in the first column, the causes of failure are in the second column, the censoring codes are in the third column (1 for censored observation, 0 for failure) and the different covariate values are in corresponding columns after this if the data is not without covariates, in which there are only three columns in the data matrix.
For the case without covariates, the first part of a data matrix has been presented as an example

```
                 [,1] [,2] [,3]
     [1,]  2.69306283    4    0
x =  [2,]  1.90742789    5    0
     [3,]  5.82563631    3    1
     [4,]  7.93480666    4    0
```

In this model there are 5 states in total, and 2 causes of failure. States 4 and 5 are absorbing. Data point 3 is censored, which is not just seen from the censoring code 1, but also that no absorbing state is in column 2. The censored observations are always represented in this column by the last transient state before failure.

For the case with covariates, the first part of a data matrix is presented below

```
                 [,1] [,2] [,3]        [,4]        [,5]
    [1,] 1.34162770    9    0  1.91963244  3.49198239
x = [2,] 0.34483496    8    1  0.39540589  0.57374830
    [3,] 1.22977702   10    0  0.69644240  2.08822711
    [4,] 0.70131724    8    1 -0.48788337  1.09445013
```

Here, there are 10 states in total, two causes of failure, represented by states 9 and 10, and two covariates. The covariate values are given in columns 4 and 5 of this data matrix.

For the rest of this text, x will represent the data matrix, on one of the forms above.

In all the code below, Nstates is the number of states in the Phase-type model, Nabs is the number of absorbing states, and Neta is the number of covariates.

7.3.1 Application of other methods

cuminc

This function is in the *cmprsk* package.

This is the non-parametric method used to estimate the sub-distribution functions for the case without covariates. To plot the sub-distribution functions the following code can be applied:

```
z = cuminc(ftime=x[,1],fstatus=x[,2], cencode = Nstates-Nabs)
plot(z)
```

In the code used to compare the Phase-type results with the non-parametric method in the *cuminc* function, a modified version of this function has been applied, such that the x- and y-axis can be adjusted, and legends and titles can be added. This function is presented in the source code, not in this text, and has been called *Cumincidence*.

crr

This function is in the *cmprsk* package.

This is the function which implements the Fine & Gray method, calculating the η-estimates and plotting the sub-distribution functions.

The following code calculates and prints the η-estimates and plots the sub-distribution functions, for the case with the chosen covariate vector, *xcov*. Here, *type* is the type of failure analyzed, corresponding to one of the absorbing states in the model, and has one of the values $1, 2, .., K + m$.
x[,2] is here re-coded by the formula: $x[, 2] - (Nstates - Nabs)$. The censored observations will then have code 0, which is the chosen censoring code. *cov1* is the variable in the *crr* function representing a matrix of covariate vectors where each row is one such vector, corresponding to a data point.

```
etaestimates=crr(ftime=x[,1],fstatus=x[,2]-(Nstates-Nabs),
                     cov1=x[,4:(4+Neta-1)],failcode=type,cencode=0)

pred=predict(etaestimates,cov1=xcov)
print(summary(etaestimates))
plot(pred)
```

comp.risk

This function is in the *timereg* package, and is developed by Scheike and Zhang [18].

It is used to perform non-parametric regression on competing risks data.

The code below calculates and prints the η-estimates, and plots the sub-distribution functions for the case with covariate vector *xcov*.
In the code below, *type* is the analyzed failure type corresponding to an absorbing state, and takes one of the values $1, 2, .., K + m$.
$x[, 2]$ is in this code re-coded to having values $1, 2, .., K + m$, by the formula: $x[, 2] - (Nstates - Nabs)$. The censored observations will then have code 0, which is chosen as the censoring code.
The data matrix is here transformed to a *data.frame* first, which is the format this function uses. The variables $V1, V2, V3$ and $V4$ are the names of the columns of covariate values in the data for this *data.frame* object. Here, 4 covariates are present.

```
xdata=as.data.frame(x)
add1=comp.risk(Hist(x[,1],x[,2]-(Nstates-Nabs),cens.code=0)~
     const(V4)+const(V5)+const(V6)+const(V7),
     data=xdata,cause=type,model='prop',cens.model='cox')
newdata=data.frame(V4=xcov[1],V5=xcov[2],V6=xcov[3],V7=xcov[4])
out=predict(add1,newdata=newdata)
print(summary(add1))
```

```
  plot(out, multiple = 1, se = 0, uniform = 0)
```

coxph

This function is in the *survival* package.

It is used for performing Cox regression on survival data with covariates, and can also be applied on competing risks data.

In the code below, this function is applied on a data set with 3 causes of failure. The model has 10 states so the corresponding absorbing states are 8, 9 and 10, and the censoring code is 7. 2 covariates are in the analyzed model.
First, the causes of failure not under analysis is re-coded to 0, so they represent censored data. The cause under analysis is coded to 1. $x1$ is the re-coded data, and $x1[,2]$ is the new status vector, which has value 1 for failures of the chosen type, and 0 for all other failures or censored data points. *coxph* is then called, using the *data.frame* version of x1. In this *data.frame* the covariate columns are called $V4$ and $V5$. The chosen cause of failure to be analyzed is 1 in this code.

```
x1=x
for(i in 1:length(x1[,2])){
   if((x1[i,2]==9)||(x1[i,2]==10)||(x1[i,2]==7)){
        x1[i,2]=0
   }
   else{
        x1[i,2]=1
   }
}

coxph(Surv(V1, V2) ~ V4 + V5, as.data.frame(x1))
```

Estimation

Calculating posterior mean of functions

The code below can be applied to produce functional estimates for many different functions expressed in Phase-type form, when a sample of model parameters has been obtained. In this text, the expressions in (89) have been used for this purpose. In this particular code, the sub-density function is calculated.

```
#Nsamples = number of parameter observations in the complete
```

```
#           sample

#Nabs = number of absorbing states

#t= vector of time points

#a=thinning factor

#Avalues = list of sampled intensity matrices, A
#          for the case without covariates

#Qvalues = list of sampled transient intensity matrices if
#          covariates are used, but without diagonal values since
#          they are covariate dependent and must be calculated for
#          every covariate

#Lvalues = list of sampled baseline absorbing intensity matrices

#etavalues = list of sampled matrices with estimate cause-specific
#            coefficient vectors as rows

#pvalues = list of sampled initial distributions, p

#cov = specific covariate vector in analysis

meanvalues=c(0)
counter=0
for(k in 1:Nsamples){
    if(((k%%a)==0) || (k==1)){
      Atemp=matrix(0,nrow=Nstates,ncol=Nstates)
      L = t(t(Lvalues[[k]])*exp(as.double(cov%*%
                              (t(t(etavalues[[k]]))))))
      Atemp[1:(Nstates-Nabs),1:(Nstates-Nabs)]=Qvalues[[k]]
      Atemp[1:(Nstates-Nabs),(Nstates-Nabs+1):Nstates] = L

      Q = (Qvalues[[k]]-diag(diag(Qvalues[[k]])))-
                  diag(rowSums(Atemp[1:(Nstates-Nabs),1:Nstates]))

      Qinv = solve(Q)
      for(i in 1:length(t)){
```

```
      #sub-density
      flist[i] = pvalues[[k]]%*%(MatrixExp(t[i]*Q))%*%L%*%v_type


      meanvalues[i] = meanvalues[i]+flist[i]
    }
    counter=counter+1
  }
}
meanvalues=meanvalues/counter
```

Calculating posterior mean of η-values

To estimate the η-values the code below has been used. Here, many of the variables can be interpreted as in the code above.

```
#Neta = number of covariates

counter=0
etamean = matrix(0,nrow=Neta,ncol=Nabs)
for(k in 1:Nsamples){
  if(((k%%a)==0) || (k==1)){
   Q = Qvalues[[k]]
   L = Lvalues[[k]]*exp(as.double(cov%*%etavalues[[k]]))
   Q = Q - diag(rowSums(Q))-diag(rowSums(L))
   etamean=etamean+etavalues[[k]]
   counter=counter+1
  }
}
etamean=etamean/counter
```

Calculating posterior standard deviation of η-values

To estimate the standard deviation of the η-values the code below has been used, where many of the variables can be interpreted as in the code above.

```
#etamean = posterior mean of eta

counter=0
etasd=matrix(0,nrow=Neta,ncol=Nabs)
for(k in 1:Nsamples){
  if(((k%%a)==0) || (k==1)){
    etasd=etasd+(etavalues[[k]]-etamean)^2
    counter=counter+1
  }
}
etasd=sqrt(etasd/(counter-1))
```

Calculating credibility intervals of functions

Following the method described for calculating credibility intervals, the code below calculates the credibility limits for the case with covariates. All the variables in this code have new interpretations. *hvalues* is the sample of Q's without the diagonals, *betavalues* is the sample of η-values, *tvalues* is the sample of L's, *pivalues* is the sample of the initial distributions, and *Nbeta* is the number of covariates.

```
MCFj_conf2= function(xcov,s,w,tvalues,pivalues,
                 betavalues,hvalues,Nstates,Nbeta,Nabs,x,type,lim){

  #This calculates credibility intervals for a sample obtained
  t=seq(0,s,w)
  Alist = c(0)
  Plist = c(0)

  v_type=rep(0,Nabs)
  v_type[type]=1
  if(full==TRUE){
    v_type=rep(1,Nabs)
  }

  a=100
  conflist_low=c(0)
  conflist_high=c(0)
  fvalues=matrix(0,nrow=(length(tvalues)/a),ncol=length(t))
  for(i in 1:length(t)){
    n=1
    for(k in 1:length(tvalues)){
      if(((k%%a)==0)){
        Atemp=matrix(0,nrow=Nstates,ncol=Nstates)
        L = t(t(tvalues[[k]])*
           exp(as.double(xcov%*%(t(t(betavalues[[k]]))))))
        Atemp[1:(Nstates-Nabs),
                    1:(Nstates-Nabs)]=hvalues[[k]]
        Atemp[1:(Nstates-Nabs),
                        (Nstates-Nabs+1):Nstates] = L
        Q = hvalues[[k]]-diag(rowSums
                       (Atemp[1:(Nstates-Nabs),1:Nstates]))
        Qinv=solve(Q)
        flist = pivalues[[k]]%*%
                         (MatrixExp(t[i]*Q))%*%L%*%v_type

        fvalues[n,i] = flist
        n=n+1
      }

    }
    print(i)
    confint_low=sort(fvalues[,i])[ceiling((length(tvalues)/a)*0.025)]
    confint_high=sort(fvalues[,i])[floor((length(tvalues)/a)*0.975)]
    conflist_low[i]=confint_low
    conflist_high[i]=confint_high
  }
  n=1
  for(k in 1:length(tvalues)){
    if(((k%%a)==0)){
      if(k!=a){
        par(new = TRUE)
      }
      plot(t,fvalues[n,],col = 'red',type =
                   'l',lty = 1,xlab ='',ylab='', ylim=c(0,lim))
      n=n+1
    }
  }



  meanlist=rep(0,length(t))
  counter=0
  for(k in 1:length(tvalues)){
    if(((k%%(a))==0)){
      Atemp=matrix(0,nrow=Nstates,ncol=Nstates)
      L = t(t(tvalues[[k]])*exp(as.double
                            (xcov%*%(t(t(betavalues[[k]]))))))
      Atemp[1:(Nstates-Nabs),1:(Nstates-Nabs)]=hvalues[[k]]
      Atemp[1:(Nstates-Nabs),(Nstates-Nabs+1):Nstates] = L
      Q = (hvalues[[k]]-diag(diag(hvalues[[k]])))
                  -diag(rowSums(Atemp[1:(Nstates-Nabs),1:Nstates]))
      Qinv=solve(Q)
      for(i in 1:length(t)){
        #subdens
        meanlist[i] = meanlist[i]+pivalues[[k]]
                              %*%(MatrixExp(t[i]*Q))%*%L%*%v_type
      #subhaz
       # meanlist[i] = meanlist[i] +
                   pivalues[[k]]%*%(MatrixExp(t[i]*Q))%*%L%*%v_type/(1-
                   pivalues[[k]]%*%Qinv%*%(MatrixExp(t[i]*Q)
                         -diag(Nstates-Nabs))%*%L%*%rep(1,Nabs))
     #subdistr
       meanlist[i] = meanlist[i]+pivalues[[k]]%*%
         Qinv%*%(MatrixExp(t[i]*Q)-diag(Nstates-Nabs))%*%L%*%v_type

      }
      print(k)
      counter=counter+1
    }
  }
  meanlist=meanlist/counter
  lines(t,meanlist,col='blue',lty = 1,lwd=3)
  lines(t,conflist_low,col='black',lty = 1,lwd=3)
  lines(t,conflist_high,col='black',lty = 1,lwd=3,
                    xlab='t',ylab='Sub-hazard')
legend('top',legend = c('Posterior mean','Confidence limits',
               'Posterior estimates'),col = c('blue','black','red'),
            lty = c(1,1,1),lwd=c(3,3,1),bty = 'n')
legend('topright',legend = c('Cause',type),horiz=TRUE,bty = 'n')

}
```

Complete algorithm

The complete algorithm for the case with covariates and the case without covariates are presented here.

It is an unintuitive code which can be difficult to understand on paper. All the code above, or very similar code has been used to find estimates of functions and parameters after the complete algorithm has produced a sample of Phase-type parameters. The notation is different than for the scripts above. The η-values are referred to as *beta*-values, and ζ, β and ν are referred to as *Zhyp*, *Bhyp* and *Nhyp*. In addition, L is referred to as t, Q is referred to as h, and $\mathbf{p}$ is referred to as *pi*.

Some of the expressions are very long, and they have been shortened down, such that they start at the next line with an indentation.

The help functions are also given below, and are important to look at if one wants to understand the algorithm.

The help functions called *SimulateMarkov* and *SimulateMarkov_cens* are used in both the algorithms, without any modifications.

Complete algorithm without covariates

```
#Initialization of dimensions and iterations

Nstates = 5
Nabs = 2
Nfailures =300
Nburnin = 0
Nsamples = 1000

#Number of MH-steps for each Gibbs-step
Ntune = 1

#Initializing parameters which is to be sampled
t = matrix(0, nrow = Nstates, ncol = Nstates)
pi = rep(0,Nstates-Nabs)

#Creating theoretical parameters

Astates = 4
```

```
A = matrix(0,nrow = Astates,ncol = Astates)
A[1,1] = -0.3
A[1,2] = 0.2
A[1,3] = 0.1
A[2,1] = 0
A[2,2] = -0.6
A[2,3] = 0.5
A[2,4] = 0.1
Api = c(1,0)

#Specifying hyper-parameters Nhyp and Bhyp
Nhyp = matrix(0.1,nrow = Nstates,ncol = Nstates)

for(i in 1:(Nstates-Nabs)){
  Nhyp[i,i] = 0
}
for(i in (Nstates-Nabs+1):Nstates){
  Nhyp[,i] = Lscale
}

Nhyp[(Nstates-Nabs+1):Nstates,] = 0


Bhyp = rep(0.01,(Nstates-Nabs))

#Drawing from priors
pi = rdirichlet(1,Bhyp)

c = 500
d = 50
tau = c(0)
theta = c(0)
Zhyp = c(0)
for(i in 1:(Nstates-Nabs)){
  tau[i] = 10
  theta[i] = rgamma(1,shape = c,scale = 1/d)
  Zhyp[i] = rgamma(1,shape = tau[i],scale = 1/theta[i])
}
print(theta)
print(sum(Zhyp))
```

```
for(i in 1:(Nstates-Nabs)){
  s = 0
  for(j in 1:Nstates){
    if(i!=j){
       t[i,j] = rgamma(1,shape = Nhyp[i,j],scale = 1/Zhyp[i])
       s = s + t[i,j]
    }
  }
  t[i,i] = -s
}

#Saving prior parameters
priort = t
priorpi = pi


#Creating theoretical and prior parameters to be used in plots
P1 = genQLp(A,Astates,Nabs)
Q1 = P1[[1]]
L1 = P1[[2]]


#Simulating the data to be used

x = SimulateFailures_cens(Nfailures,A,Astates,Nabs,Api)
x[,2] = x[,2]+Nstates-Astates

#Sampling posterior values

initial = list(0)
for(i in 1:Nfailures){
  if(x[i,3]==1){
    initial[[i]] = SimulateMarkov_cens(x[i,1],
                              x[i,2],t,Nstates,Nabs,pi)
  }
  else{
    initial[[i]] = SimulateMarkov(x[i,1],
                               x[i,2],t,Nstates,Nabs,pi)
  }
  print(i)
```

```
}

fjvalues = list(0)
pitrace1=c(0)
pitrace2=c(0)
pitrace3=c(0)
tvalues = list(0)
pivalues = list(0)

for(k in 1:(Nburnin+Nsamples)){
  print(k)

  #Running Metropolis-Hastings
  Y = MH_cens(x,t,Ntune,Nstates,Nabs,Nfailures,pi,initial)
  B = Y[[1]]
  Z = Y[[2]]
  N = Y[[3]]
  initial = Y[[5]]

  #Running Gibbs-step
  pi = rdirichlet(1,(Bhyp+B))
  for(i in 1:(Nstates-Nabs)){
    theta[i] = rgamma(1,shape = c+tau[i], scale = 1/(d+Zhyp[i]))
    Zhyp[i] = rgamma(1,shape = (tau[i]+sum(Nhyp[i,])),
                                    scale = 1/(theta[i]-t[i,i]))
  }
  for(i in 1:(Nstates-Nabs)){
    s = 0
    for(j in 1:Nstates){
      if(i!=j){
        t[i,j] = rgamma(1,shape = (Nhyp[i,j] + N[i,j]),
                                       scale = 1/(Zhyp[i]+Z[i]))
        s = s + t[i,j]
      }
    }
    t[i,i] = -s
  }

  #Saving sample
  if(k>Nburnin){
    tvalues[[k-Nburnin]] = t
    pivalues[[k-Nburnin]] = pi
  }
}

#Creating functional plots

#Functional plots
f = posteriormean(10,0.10,tvalues[900:1000],
    pivalues[900:1000],Api,Nstates,Astates,Nabs,Q1,L1,x,2,1)
```

MH_cens

This is the Metropolis-Hastings sampler used to sample Markov chain realizations.

```
MH_cens = function(X,A,m,Nstates,Nabs,Nfailures,pi,initial){

#This is the MH-method for estimating Markov chain realizations

  B = rep(0,(Nstates-Nabs))
  Z = rep(0,(Nstates-Nabs))
  N = matrix(0,nrow = Nstates,ncol = Nstates)
  ttemp = rep(0,Nfailures)
  laststates = rep(0,Nfailures)
  Q = genQLp(A,Nstates,Nabs)[[1]]
  L = genQLp(A,Nstates,Nabs)[[2]]
  Qinv = solve(Q)
  for(i in 1:length(X[,1])){
    if(X[i,3] == 0){
      v_c = rep(0,Nabs)
      v_c[X[i,2]-Nstates+Nabs] = 1
      temp = initial[[i]]
      B_temp = temp[[1]]
      Z_temp = temp[[3]]
      N_temp = temp[[2]]
      ttemp[i] = temp[[4]]
      laststates[i] = temp[[5]]
      for(j in 1:m){
        temp = SimulateMarkov(X[i,1],X[i,2],A,Nstates,Nabs,pi)
        v_temp = rep(0,(Nstates-Nabs))
        v_temp[temp[[5]]] = 1
        v_ttemp = rep(0,(Nstates-Nabs))
        v_ttemp[laststates[i]] = 1
        f_temp = function(t){as.double(v_temp%*%
                      Qinv%*%(-diag(Nstates-Nabs))%*%L%*%t(t(v_c)))}

        marginal_temp = f_temp(0)

        f_ttemp = function(t){as.double(v_ttemp%*%
                      Qinv%*%(-diag(Nstates-Nabs))%*%L%*%t(t(v_c)))}

        marginal_ttemp = f_ttemp(0)
        u = runif(1,0)
        if(((ttemp[i]*marginal_temp)==0) || (u<=(min(1,
           ((temp[[4]]*marginal_ttemp)/(ttemp[i]*marginal_temp)))))){

          B_temp = temp[[1]]
          Z_temp = temp[[3]]
          N_temp = temp[[2]]
          ttemp[i] = temp[[4]]
          laststates[i] = temp[[5]]
          initial[[i]] = temp
        }
      }
    }
    if(X[i,3]==1){
        temp = SimulateMarkov_cens(X[i,1],X[i,2],A,Nstates,Nabs,pi)
        B_temp = temp[[1]]
        Z_temp = temp[[3]]
        N_temp = temp[[2]]
        initial[[i]] = temp
    }
    B = B + B_temp
    Z = Z + Z_temp
    N = N + N_temp
  }
  output = list(B,Z,N,ttemp,initial)
}
```

genQLp

This function simply divides an intensity matrix into Q and L matrices

```
genQLp= function(Z,Nstates,Nabs){
  Q = Z[1:(Nstates-Nabs),1:(Nstates-Nabs)]
  L = t(t(Z[1:(Nstates-Nabs),(Nstates-Nabs+1):Nstates]))
  T = list(Q,L)

}
```

Simulatemarkov

This function is used to simulate Markov chain realizations at each Gibbs-step.

```
SimulateMarkov = function(x,c,A,Nstates,Nabs,pi){

#This function simulates Markov chain
          realizations from the Phase-type #model

  j = 0
  t = 0
  iterations = 0
  while(!((j == c) && (t>=x))){
   t_xj = NA
   laststate = NA
   B_temp = rep(0,(Nstates-Nabs))
   Z_temp = rep(0,(Nstates-Nabs))
   N_temp = matrix(0,nrow = Nstates,ncol=Nstates)

   initial = sample(1:(Nstates-Nabs),1,replace=T,prob = pi)
   j = initial

   t = 0
   B_temp[j] = B_temp[j] + 1
   while(j < (Nstates-Nabs + 1)){

    told = t
    t = t + rexp(1,rate = -A[j,j])

    if(t<x){
      Z_temp[j] = Z_temp[j] + (t-told)
    }
    if((told<x) && (t>=x)){
      t_xj=A[j,c]
      laststate = j
      Z_temp[j] = Z_temp[j] + (x-told)
    }

    p = A[j,]
    p[j] = 0
    p = p/(-A[j,j])

    jold = j
    j = sample(1:Nstates,1, replace=T, prob=p)

    if(t<x){
      N_temp[jold,j] = N_temp[jold,j] + 1
    }
    if((t>=x)&&(told<x)){
      N_temp[jold,c] = N_temp[jold,c] + 1
    }
   }
  }
  output = list(B_temp,N_temp,Z_temp,t_xj,laststate)
}
```

Simulatemarkov_cens

This has the same function as *Simulatemarkov*, but are only called for censored observations.

```
SimulateFailures_cens=function(m,A,Astates,Nabs,Api){
#This function simulates censored Phase-type data

  x = matrix(0,nrow = m,ncol = 3)
  for(i in 1:m){
     initial = sample(1:(Astates-Nabs),1,replace=T,prob = Api)
     j = initial
     while(j < (Astates-Nabs + 1)){
       x[i,1] = x[i,1] + rexp(1,rate = -A[j,j])
       p = A[j,]
```

```
      p[j] = 0
      p = p/(-A[j,j])
      j = sample(1:Astates,1,replace=T, prob=p)
    }
    C = rweibull(1,scale = 5,shape = 1.5)
    if(C<x[i,1]){
      x[i,1] = C
      x[i,3] = 1
      x[i,2] = Astates-Nabs
    }
    else{
      x[i,3] = 0
      x[i,2] = j
    }
  }
  print(sum(x[,3])/length(x[,1]))
  x
}
```

SimulateFailures_cens

This function simulates a Phase-type data set, taking in theoretical Phase-type parameters. It is also partially used for simulating data in the covariate case, within another simulating function.

```
SimulateFailures_cens = function(m,A,Astates,Nabs,Api){
  x = matrix(0,nrow = m,ncol = 3)

#Phase-type
  for(i in 1:m){
    initial = sample(1:(Astates-Nabs),1,replace=T,prob = Api)
    j = initial
    while(j < (Astates-Nabs + 1)){
      x[i,1] = x[i,1] + rexp(1,rate = -A[j,j])
      p = A[j,]
      p[j] = 0
      p = p/(-A[j,j])
      j = sample(1:Astates,1,replace=T, prob=p)
    }
    C = rweibull(1,scale = 10,shape = 10)
```

```
      if(C<x[i,1]){
        x[i,1] = C
        x[i,3] = 1
        x[i,2] = Astates-Nabs
      }
      else{
        x[i,3] = 0
        x[i,2] = j
      }
  }
  x
}
```

posteriormean

Function used to estimate and plot the functional values. It also uses the function *Cumincidence* to estimate non-parametric sub-distribution curves. This is based on *cuminc*, but har been customized such that it can plot the estimates with this Phase-type method.

```
posteriormean= function(s,w,tvalues,
               pivalues,Api,Nstates,Astates,
                       Nabs,Q1,L1,x,type,ylimit){


  t = seq(0,s,w)

  v_type=rep(0,Nabs)
  v_type[type]=1

  Alist=c(0)
  mean=rep(0,length(t))

  Q1inv=solve(Q1)
  for(k in 1:length(tvalues)){
      flist = c(0)
      Q = genQLp(tvalues[[k]],Nstates,Nabs)[[1]]
      L = genQLp(tvalues[[k]],Nstates,Nabs)[[2]]
      Qinv = solve(Q)
      for(i in 1:length(t)){
#       flist[i] = pivalues[[k]]%*%
```

```
                (MatrixExp(t[i]*Q))%*%L%*%v_type

#       flist[i] = pivalues[[k]]%*%Qinv%*%
           (MatrixExp(t[i]*Q)-diag(Nstates-Nabs))%*%L%*%v_type

       flist[i] = pivalues[[k]]%*%(MatrixExp(t[i]*Q))
            %*%L%*%v_type/(1-pivalues[[k]]%*%solve(Q)
               %*%(MatrixExp(t[i]*Q)-diag(Nstates-Nabs))
                                          %*%L%*%rep(1,Nabs))

              mean[i] = mean[i]+flist[i]

#      Alist[i] = Api%*%(MatrixExp(t[i]*Q1))%*%L1%*%v_type
#      Alist[i] = Api%*%Q1inv%*%(MatrixExp(t[i]*Q1)-
                          diag(Astates-Nabs))%*%L1%*%v_type
       Alist[i] = Api%*%(MatrixExp(t[i]*Q1))%*%
            L1%*%v_type/(1-Api%*%Q1inv%*%(MatrixExp(t[i]*Q1)
                    -diag(Astates-Nabs))%*%L1%*%rep(1,Nabs))
      }
      print(k)
  }
  mean = mean/(length(tvalues))

# Non-parametric estimate, specific cause
       to be analyzed must be changed within this function
#   CumIncidence(group=type,ftime=x[,1],fstatus=x[,2],
                               cencode = Nstates-Nabs, t=t)
#   par(new = TRUE)

  plot(t,Alist,col = 'red',type = 'l',lty = 1,
                    lwd=2,xlab ='',ylab='', ylim = c(0,ylimit))
  lines(t,mean,col = 'blue',type = 'l',lty = 2,
                    lwd=2,xlab ='',ylab='', ylim = c(0,ylimit))

legend('topleft',legend = c('Posterior mean','Empirical estimate',
             'True function'),col = c('blue','black','red'),
                             lty = c(2,1,1),lwd=c(1,1,2),bty = 'n')


# legend('topleft',legend = c('Posterior mean',
                   'Theoretical function'),col = c('blue',
                       'red'),lty = c(2,1),lwd=c(2,2),bty = 'n')
}
```

Cumincidence

This plots non-parametric estimates of the sub-distribution functions.

```
"CumIncidence" <- function(ftime, fstatus, group, t,strata,
                  rho = 0, cencode = 0, subset, na.action =
                   na.omit, level,xlab = "Time",
                   ylab = "Probability", col, lty,
                                         lwd, digits = 4)
{

  #This is used to plot cuminc function

  if(!require("cmprsk"))
    { stop("Package `cmprsk' is
                       required and must be installed.\n
                  See help(install.packages) or write the
                              following command at prompt
          and then follow the instructions:\n
          > install.packages(\"cmprsk\")") }
  # collect data
  mf  <- match.call(expand.dots = FALSE)
  mf[[1]] <- as.name("list")
  mf$t <- mf$digits <- mf$col <- mf$lty <- mf$lwd <- mf$level <-
  mf$xlab <- mf$ylab <- NULL
  mf <- eval(mf, parent.frame())
  g <- max(1, length(unique(mf$group)))
  s <- length(unique(mf$fstatus))
  if(missing(t))
    { time <- pretty(c(0, max(mf$ftime)), 6)
      ttime <- time <- time[time < max(mf$ftime)] }
  else { ttime <- time <- t }
  # fit model and estimates at time points
  fit   <- do.call("cuminc", mf)
  tfit <- timepoints(fit, time)
  # print result
  cat("\n+", paste(rep("-", 67), collapse=""), "+", sep ="")
  cat("\n| Cumulative incidence function
                     estimates from competing risks data |")

  cat("\n+", paste(rep("-", 67), collapse=""), "+\n", sep ="")
  tests <- NULL
  if(g > 1)
    { tests <- fit$Tests
    colnames(tests) <- c("Statistic", "p-value", "df")
      cat("Test equality across groups:\n")
      print(tests, digits = digits) }
  cat("\nEstimates at time points:\n")
  print(tfit$est, digits = digits)
  cat("\nStandard errors:\n")
  print(sqrt(tfit$var), digits = digits)
  #
  if(missing(level))
    { # plot cumulative incidence functions
      if(missing(t))
        { time <- sort(unique(c(ftime, time)))
          x <- timepoints(fit, time) }
      else x <- tfit
      col <- if(missing(col)) rep(1:(s-1),
                                rep(g,(s-1))) else col

      lty <- if(missing(lty)) rep(1:g, s-1) else lty
      lwd <- if(missing(lwd)) rep(1, g*(s-1)) else lwd


      plot(time, base::t(x$est)[,group],
                              type="s", ylim = c(0,1))

      out <- list(test = tests, est =
                         tfit$est, se = sqrt(tfit$var))
    }
  invisible(out)
}
```

Complete algorithm with covariates

This has very many similar help functions as the complete algorithm without covariates, but some different calculations must be made to handle the covariates, so they are still presented here.

One of the help functions is a MH-sampler for the η-values. It has been developed particularly for this covariate algorithm. It will be presented after the complete algorithm.

```
#Initialization of dimensions and iterations
Astates =10
Nstates=10
Nabs = 3
Aabs = 3
Nbeta = 2
Nfailures= 200
Nburnin = 1000
Nsamples = 1000
Ntune = 1
cstep = 0.01*matrix(1,nrow=Nbeta,ncol=Nabs)
sigma=0.5

# x=sim(Nfailures,H,L,beta,Nstates,Aabs,Api,Nbeta)
# x[,2] = x[,2]+Nstates-Astates


#Follic data set
# fol <- read.table("follic.txt", sep = ",", header = TRUE)

# evcens <- as.numeric(fol$resp == "NR" |
                                      fol$relsite != "")
# crcens <- as.numeric(fol$resp == "CR" &
                          fol$relsite == "" & fol$stat == 1)

# cause <- ifelse(evcens == 1, 1, ifelse(crcens == 1, 2, 0))
# table(cause)
# stage <- as.numeric(fol$clinstg == 2)
# chemo <- as.numeric(fol$ch == "Y")
# times1 <- sort(unique(fol$dftime[cause == 1]))
# x=matrix(0,nrow=Nfailures,ncol=(3+Nbeta))
# x[,1]=fol$dftime
# x[,2]=cause+Nstates-Nabs
# x[,3]=(1-fol$dfcens)
# x[,4]=fol$age/10
# x[,5]=fol$hgb/10
# x[,6]=stage
# x[,7]=chemo



#Initializing parameters which is to be sampled
h = matrix(0, nrow = Nstates-Nabs, ncol = Nstates-Nabs)
t = matrix(0,nrow=(Nstates-Nabs),ncol=Nabs)
A = matrix(0,nrow = Nstates,ncol = Nstates)
pi = rep(0,Nstates-Nabs)

#Specifying hyper-parameters Nhyp and Bhyp
Nhyp = matrix(0.1,nrow = Nstates,ncol = Nstates)

for(i in 1:(Nstates-Nabs)){
  Nhyp[i,i] = 0
}
for(i in (Nstates-Nabs+1):Nstates){
  Nhyp[,i] = 0.1
}

Nhyp[(Nstates-Nabs+1):Nstates,] = 0


Bhyp = rep(0.1,(Nstates-Nabs))


#Drawing from priors
pi = rdirichlet(1,Bhyp)

c = 1000
d = 10
tau = c(0)

theta = c(0)
Zhyp = c(0)
hdiag=matrix(0,nrow=Nfailures,ncol=(Nstates-Nabs))
for(i in 1:(Nstates-Nabs)){
  tau[i] = 1*10
  theta[i] = rgamma(1,shape = c,scale = 1/d)
  Zhyp[i] = rgamma(1,shape = tau[i],scale = 1/theta[i])
}
print(theta)
print(sum(Zhyp))

beta=matrix(0,nrow=Nbeta,ncol=Nabs)
for(j in 1:Nabs){
 beta[,j] = rnorm(Nbeta,mean=0,sd=sigma)

}

for(i in 1:(Nstates-Nabs)){
  for(j in 1:(Nstates-Nabs)){
    if(i!=j){
       h[i,j] = rgamma(1,shape = Nhyp[i,j],scale = 1/(Zhyp[i]))
    }
  }
}
for(i in 1:(Nstates-Nabs)){
    for(j in 1:Nabs){
        t[i,j] = rgamma(1,shape = Nhyp[i,j+
                          (Nstates-Nabs)],scale = 1/Zhyp[i])
    }

    for(data in 1:Nfailures){
      hdiag[data,i]=-((sum(h[i,])-h[i,i])+sum(t[i,]*
           exp(as.double(x[data,4:(4+Nbeta-1)]%*%t(t(beta))))))
    }
}

priort=t
priorh=h
priorbeta=beta
priorhdiag=hdiag
priorpi=pi
```

```
initial = list(0)
for(i in 1:Nfailures){
  A[1:(Nstates-Nabs),1:(Nstates-Nabs)] =
                (h-diag(diag(h)))+diag(hdiag[i,])

  A[1:(Nstates-Nabs),(Nstates-Nabs+1):Nstates] =
       t(t(t)*exp(as.double(x[i,4:(Nbeta+3)]%*%t(t(beta)))))

  if(x[i,3]==1){
    initial[[i]] = SimulateMarkov_cens
                    (x[i,1],x[i,2],A,Nstates,Nabs,pi)
  }
  else{
    initial[[i]] = SimulateMarkov
               (x[i,1],x[i,2],A,Nstates,Nabs,pi)
  }
  print(i)
}


tvalues = list(0)
pivalues = list(0)
betavalues=list(0)
hvalues=list(0)
for(k in 1:(Nburnin+Nsamples)){
  print(k)

  #Running Metropolis-Hastings
  Y = MH_cens(x,h,t,hdiag,beta,Ntune,Nstates,
                       Nabs,Nfailures,pi,initial,Nbeta)
  B = Y[[1]]
  Z = Y[[2]]
  N = Y[[3]]
  initial = Y[[5]]
  n=Y[[7]]
  z=Y[[6]]

  #Running Gibbs-step
  pi = rdirichlet(1,(Bhyp+B))
```

```
for(i in 1:(Nstates-Nabs)){
    theta[i] = rgamma(1,shape = c+tau[i], scale = 1/(d+Zhyp[i]))
    Zhyp[i] = rgamma(1,shape = (tau[i]+sum(Nhyp[i,])),
              scale = 1/(theta[i]+sum(h[i,])-h[i,i]+sum(t[i,])))
}


beta = betaMH(z,h,x,N,n,Z,Nhyp,Zhyp,pi,B,
                                Bhyp,beta,t,Nbeta,cstep,sigma)


for(i in 1:(Nstates-Nabs)){
  h[i,i] = 0
  for(j in 1:(Nstates-Nabs)){
    if(i!=j){
      h[i,j] = rgamma(1,shape = (Nhyp[i,j] +
                             N[i,j]),scale = 1/(Zhyp[i]+Z[i]))
    }
  }


  for(j_a in 1:Nabs){
    t[i,j_a] = rgamma(1,shape = (Nhyp[i,j_a+
             (Nstates-Nabs)]+N[i,j_a+(Nstates-Nabs)]),
                  scale = 1/(Zhyp[i]+sum(z[,i]*exp(as.double
                     (x[,4:(4+Nbeta-1)]%*%t(t(beta[,j_a])))))))


  }

  for(data in 1:Nfailures){
    hdiag[data,i]=-((sum(h[i,])-h[i,i])+sum(t[i,]*
           exp(as.double(x[data,4:(4+Nbeta-1)]%*%t(t(beta))))))
  }
}


#Generating trace plot value and saving sample
if(k>Nburnin){
  hvalues[[k-Nburnin]] = h
```

```
    tvalues[[k-Nburnin]] = t
    pivalues[[k-Nburnin]] = pi
    betavalues[[k-Nburnin]] = beta
  }
}


analysis(c(0,0),10,0.1,1,tvalues,pivalues,betavalues,
    hvalues,Api,AH,AL,beta,Nstates,Astates,Nabs,x,Nbeta,1,1)
```

etaMH

This is the MH-sampler for the η-values.

```
betaMH = function(z,H,x,N,n,Z,Nhyp,Zhyp,pi,B,
                                Bhyp,beta,t,Nbeta,cstep,sigma){
   for(v in 1:Nbeta){
     for(j in (Nstates-Nabs+1):Nstates){
        u=beta[v,(j-Nstates+Nabs)]
        theta = u
        phi=rnorm(1,mean=theta,sd=sqrt(cstep[v,j-Nstates+Nabs]))
        csumphi=0
        csumtheta = 0
        dsum=0
        dsumphi=1
        dsumtheta=1
        for(k in 1:length(x[,4+v-1])){
          Uphi=exp(phi*x[k,4+v-1])
          Utheta=exp(theta*x[k,4+v-1])
          csumphi=csumphi+sum(t[,(j-Nstates+Nabs)]*z[k,])*Uphi*
                   exp((x[k,4:(4+Nbeta-1)]%*%
                      t(t(beta[,(j-Nstates+Nabs)])))-
                        beta[v,(j-Nstates+Nabs)]*x[k,4+v-1])

          csumtheta=csumtheta+sum(t[,(j-Nstates+Nabs)]*z[k,])*
                    Utheta*exp((x[k,4:(4+Nbeta-1)]%*%
                         t(t(beta[,(j-Nstates+Nabs)])))-
                         beta[v,(j-Nstates+Nabs)]*x[k,4+v-1])

          dsum=dsum+x[k,4+v-1]*sum(n[[k]][,j])
        }
        csumphi=as.double(csumphi)
        csumtheta=as.double(csumtheta)

        w=runif(1)
          if((is.finite(min(1,exp((((phi*(dsum)-
           (((phi^2)/(2*sigma^2))))+(-csumphi))-
           (((theta*(dsum)-(((theta^2)/(2*sigma^2))))+
           (-csumtheta))))))))&&(w<=min(1,exp((((phi*(dsum)-
           (((phi^2)/(2*sigma^2))))+(-csumphi))-(((theta*(dsum)-
           (((theta^2)/(2*sigma^2))))+(-csumtheta)))))))){
         u=phi

        }
        else{
          u=theta
        }
        beta[v,(j-Nstates+Nabs)]=u
     }
   }
   beta
}
```

MH_cens_cov

Just an extension of the original *MH_cens* function, to the covariate case.

```
MH_cens = function(X,h,a,hdiag,beta,m,Nstates,
                     Nabs,Nfailures,pi,initial,Nbeta){

  A=matrix(0,nrow=Nstates,ncol=(Nstates))
  B = rep(0,(Nstates-Nabs))
  Z = rep(0,(Nstates-Nabs))
  N = matrix(0,nrow = Nstates,ncol = Nstates)
  z = matrix(0,nrow=Nfailures,ncol=(Nstates-Nabs))
  n = list(matrix(0,nrow=Nstates,ncol=Nstates))
  ttemp = rep(0,Nfailures)
  laststates = rep(0,Nfailures)
  for(i in 1:length(X[,1])){
    Q = (h-diag(diag(h)))+diag(hdiag[i,])
```

```
L = t(t(a)*exp(as.double(X[i,4:(4+Nbeta-1)]%*%t(t(beta)))))
A[1:(Nstates-Nabs),1:(Nstates-Nabs)] = Q
A[1:(Nstates-Nabs),(Nstates-Nabs+1):Nstates] = L
Qinv = solve(Q)
if(X[i,3] == 0){
  v_c = rep(0,Nabs)
  v_c[X[i,2]-Nstates+Nabs] = 1
  temp = initial[[i]]
  B_temp = temp[[1]]
  Z_temp = temp[[3]]
  N_temp = temp[[2]]
  ttemp[i] = temp[[4]]
  laststates[i] = temp[[5]]
  for(j in 1:m){
    temp = SimulateMarkov(X[i,1],X[i,2],A,Nstates,Nabs,pi)
    v_temp = rep(0,(Nstates-Nabs))
    v_temp[temp[[5]]] = 1
    v_ttemp = rep(0,(Nstates-Nabs))
    v_ttemp[laststates[i]] = 1
    f_temp = function(t){as.double(v_temp%*%Qinv%*%
                     (-diag(Nstates-Nabs))%*%L%*%t(t(v_c)))}
    marginal_temp = f_temp(0)
    f_ttemp = function(t){as.double(v_ttemp%*%Qinv%*%
                      (-diag(Nstates-Nabs))%*%L%*%t(t(v_c)))}

    marginal_ttemp = f_ttemp(0)
    u = runif(1,0)
    if(((ttemp[i]*marginal_temp)==0) || (u<=(min(1,
            ((temp[[4]]*marginal_ttemp)/
                     (ttemp[i]*marginal_temp)))))){
      B_temp = temp[[1]]
      Z_temp = temp[[3]]
      N_temp = temp[[2]]
      ttemp[i] = temp[[4]]
      laststates[i] = temp[[5]]
      initial[[i]] = temp
    }
  }
}
if(X[i,3]==1){
      temp = SimulateMarkov_cens(X[i,1],X[i,2],A,Nstates,Nabs,pi)
      B_temp = temp[[1]]
      Z_temp = temp[[3]]
      N_temp = temp[[2]]
      initial[[i]] = temp
    }
    z[i,] = Z_temp
    n[[i]]= N_temp
    B = B + B_temp
    Z = Z + Z_temp
    N = N + N_temp
  }
  output = list(B,Z,N,ttemp,initial,z,n)
}
```

simulatedata_phase_type

This function simulates the covariate Phase-type data.

```
sim = function(Nfailures,H,L,beta,Astates,Aabs,pi,Nbeta){
  x=matrix(0,nrow=Nfailures,ncol=(4+Nbeta-1))
  for(k in 1:Nfailures){
    x[k,4:(4+Nbeta-1)]=rnorm(Nbeta,mean=0,sd=5)

    A=matrix(0,nrow=(length(H[1,])+length(L[1,])),
                     ncol=(length(H[1,])+length(L[1,])))

    A[1:length(H[1,]),1:length(H[1,])] = H

    temp=(exp(as.double(x[k,4:(4+Nbeta-1)]%*%t(t(beta)))))
#     print(temp)

    A[1:length(H[1,]),(1+length(H[1,])):
                              length(A[1,])]=t(t(L)*temp)
    d=rowSums(A)
    A=A-diag(d)
#     print(A)
    temp=SimulateFailures_cens(1,A,Astates,Aabs,pi)
    x[k,1]=temp[1]
    x[k,2]=temp[2]
    x[k,3]=temp[3]
  }
  x
}
```

simulatedata_weibull

This simulates Weibull-data.

```
sim = function(Nfailures,H,L,beta,Astates,Aabs,pi,Nbeta){
  x=matrix(0,nrow=Nfailures,ncol=(4+Nbeta-1))
  for(k in 1:Nfailures){
      x[k,4:(4+Nbeta-1)]=rnorm(Nbeta,mean=0,sd=1.5)

      t_temp=c(0)
      t_temp[1]=rweibull(1,scale=sqrt(2)*exp(as.double
              (-0.5*x[k,4:(4+Nbeta-1)]%*%
                     t(t(beta[,1])))),shape=2)

      t_temp[2]=rweibull(1,scale=sqrt(2)*exp(as.double
               (-0.5*x[k,4:(4+Nbeta-1)]%*%
                          t(t(beta[,2])))),shape=2)

      C = rweibull(1,scale = 2,shape = 1.5)
      if(C<min(t_temp)){
        x[k,1] = C
        x[k,3] = 1
        x[k,2] = 0
      }
      else{
        x[k,1]=min(t_temp)
        x[k,2]=which.min(t_temp)
        x[k,3]=0
      }
  }
  x
}
```

analysis

This function estimates functions, and run both *cmprsk* and *timereg*. It also plots the results.

```
analysis= function(xcov,s,w,a,tvalues,pivalues,betavalues,
   hvalues,Api,AH,AL,beta,Nstates,Astates,Nabs,x,Nbeta,type,ylimit){

  #Time vector
  t=seq(0,s,w)


  #Initializing
  Alist = c(0)
  flist = c(0)
  mean=rep(0,length(t))


  #Calcuating simulated Phase-type parameters,
                 take away if weibull, it will crash method
  Hdiag=c(0)
  A=matrix(0,nrow=(length(AH[1,])+length(L[1,])),
                       ncol=(length(AH[1,])+length(AL[1,])))

  temp=(exp(as.double(xcov%*%t(t(beta)))))
  A[1:length(AH[1,]),(1+length(AH[1,])):length(A[1,])]=t(t(AL)*temp)
  for(i in 1:(Astates-Nabs)){

     Hdiag[i]= -((sum(AH[i,])-AH[i,i])+
                     sum(AL[i,]*exp(as.double(xcov%*%t(t(beta))))))
  }
  A[1:length(AH[1,]),1:length(AH[1,])] =
                          (AH-diag(diag(AH)))+diag(Hdiag)

  QA=genQLp(A,Astates,Nabs)[[1]]
  LA=genQLp(A,Astates,Nabs)[[2]]
  QAinv=solve(QA)


  #Calculating mean parameter values
  hmean=matrix(0,nrow=(Nstates-Nabs),ncol=(Nstates-Nabs))
  tmean=matrix(0,nrow=(Nstates-Nabs),ncol=(Nabs))
  pimean=rep(0,(Nstates-Nabs))
  betamean=rep(0,Nbeta)
  counter=0
  for(k in 1:length(tvalues)){
    if(((k%%a)==0) || (k==1)){
      hmean=hmean+hvalues[[k]]
      tmean=tmean+tvalues[[k]]
      betamean=betamean+betavalues[[k]]
      pimean=pimean+pivalues[[k]]
      counter=counter+1
    }
  }
  hmean=hmean/counter
  tmean=tmean/counter
  pimean=pimean/counter
  betamean=betamean/counter

  #Estimating standard deviation of beta-values
  counter=0
  betadev=matrix(0,nrow=Nbeta,ncol=Nabs)
  for(k in 1:length(tvalues)){
    if(((k%%a)==0) || (k==1)){
      betadev=betadev+(betavalues[[k]]-betamean)^2
      counter=counter+1
    }
  }
  betadev=sqrt(betadev/(counter-1))

  v_type=rep(0,Nabs)
  v_type[type]=1


#Calculations for Weibull-model used in the report
#   if(type==1){
#      Afunc=function(t){(1-pweibull(t,shape=2,
                 scale=sqrt(2)*exp(as.double
        (-0.5*xcov%*%t(t(beta[,2]))))))*dweibull(t,shape=2,
           scale=sqrt(2)*exp(as.double
           (-0.5*xcov%*%t(t(beta[,1])))))}

#      Afunc1=function(t){(1-pweibull(t,shape=2,
              scale=sqrt(2)*exp(as.double
             (-0.5*xcov%*%t(t(beta[,1])))))
             *dweibull(t,shape=2,scale=sqrt(2)
            *exp(as.double(-0.5*xcov%*%t(t(beta[,2])))))}
#   }
#   else{
#     Afunc=function(t){(1-pweibull(t,shape=2,scale=
            sqrt(2)*exp(as.double(-0.5*xcov%*%
             t(t(beta[,1]))))))*dweibull(t,shape=2,
             scale=sqrt(2)*exp(as.double
            (-0.5*xcov%*%t(t(beta[,2])))))}

#     Afunc1=function(t){(1-pweibull(t,shape=2,scale=sqrt(2)
              *exp(as.double(-0.5*xcov%*%t(t(beta[,2]))))))*
              dweibull(t,shape=2,scale=sqrt(2)*exp(as.double
                     (-0.5*xcov%*%t(t(beta[,1])))))}
#   }


  for(i in 1:length(t)){
  #Phase-type simulated

  #sub-density
#       Alist[i] = Api%*%(MatrixExp(t[i]*QA))%*%LA%*%v_type
  #sub-distribution
  Alist[i] = Api%*%QAinv%*%
        (MatrixExp(t[i]*QA)-diag(Astates-Nabs))%*%LA%*%v_type
  #sub-hazard
#  Alist[i] = Api%*%(MatrixExp(t[i]*QA))%*%
         LA%*%v_type/(1-Api%*%QAinv%*%(MatrixExp
                 (t[i]*QA)-diag(Astates-Nabs))%*%
                        LA%*%rep(1,Nabs))


      #Weibull simulated

      #Hazard rate
      #Alist[i]=mvec[type]*pweibull(t[i],scale=sqrt(2)
       *exp(as.double(-0.5*xcov%*%t(t(beta[,type])))),shape=2)

      #sub-distribution
```

```
#       Alist[i]=integrate(Afunc,0,t[i])$value

  }
  counter=0
  for(k in 1:length(tvalues)){
    if(((k%%a)==0) || (k==1)){
      Atemp=matrix(0,nrow=Nstates,ncol=Nstates)
      L = t(t(tvalues[[k]])*exp(as.double(xcov%*%
                                (t(t(betavalues[[k]]))))))
      Atemp[1:(Nstates-Nabs),1:(Nstates-Nabs)]=hvalues[[k]]
      Atemp[1:(Nstates-Nabs),(Nstates-Nabs+1):Nstates] = L
      Q = (hvalues[[k]]-diag(diag(hvalues[[k]])))-
               diag(rowSums(Atemp[1:(Nstates-Nabs),1:Nstates]))

      Qinv = solve(Q)
      for(i in 1:length(t)){


        #Phase-type estimate

        #sub-density
#         flist[i] = pivalues[[k]]%*%
                   (MatrixExp(t[i]*Q))%*%L%*%v_type

        #sub-distribution
        flist[i] = pivalues[[k]]%*%Qinv%*%(MatrixExp
                   (t[i]*Q)-diag(Nstates-Nabs))%*%L%*%v_type

        #sub-hazard
#         flist[i] = pivalues[[k]]%*%(MatrixExp(t[i]*Q))
            %*%L%*%(t(t(v_type)))/(1-pivalues[[k]]
             %*%Qinv%*%(MatrixExp(t[i]*Q)
              -diag(Nstates-Nabs))%*%L%*%rep(1,Nabs))


        mean[i] = mean[i]+flist[i]
      }
#       print(k)
      counter=counter+1
    }
  }
  mean=mean/counter

    #timereg analysis, saves everything in out,
                        which can be plotted if necessary
    xdata=as.data.frame(x)
    add1=comp.risk(Hist(x[,1],x[,2]-
        (Nstates-Nabs),cens.code=0)~const(V4)+const(V5),
         data=xdata,cause=type,model='prop',cens.model="cox")
    newdata=data.frame(V4=xcov[1],V5=xcov[2])
    out<-predict(add1,newdata=newdata)


    #cmprsk analysis, pred can be plotted
    betaestimates=crr(ftime=x[,1],fstatus=x[,2]-
     (Nstates-Nabs), cov1=x[,4:(4+Nbeta-1)],
         failcode=type,cencode=0)
    pred=predict(betaestimates,cov1=xcov)

     #This is only for sub-distributions,
          use par if out or pred are plotted
     plot(pred,ylim=c(0,ylimit),
           xlim=c(0,s),xlab='t',ylab='Probability')
     par(new = TRUE)
     plot(out, multiple = 1, se = 0, uniform = 0,
          col = 1,lty=3,ylim=c(0,ylimit),xlim=c(0,s),
                       xlab='t',ylab='Probability')
     par(new = TRUE)
     lines(t,mean,col = 'blue',type = 'l',lty = 2,
                 lwd=1,xlab ='',ylab='', ylim = c(0,ylimit))
     lines(t,Alist,col='red',type='l',lty = 1,lwd=2,
                       xlab ='',ylab='', ylim = c(0,1.1))

#      #Use this if non-parametric is not plotted
#      plot(t,mean,col = 'blue',type = 'l',
            lty = 2,lwd=1,xlab ='',ylab='', ylim = c(0,ylimit))
#      lines(t,Alist,col='red',type='l',lty = 1,
                    lwd=2,xlab ='',ylab='', ylim = c(0,1.1))

    #legend('topleft',legend = c('PT estimate','FG estimate',
          'timereg estimate'),col = c('blue','black','black'),
             lty = c(2,1,3),lwd=c(2,1,2),bty = 'n')
 legend('topleft',legend = c('PT estimate','True Hazard rate'),
        col = c('blue','red'),lty = c(2,1),lwd=c(2,2),bty = 'n')
 # legend('topleft',legend = c('PT estimate',
         'FG estimate','timereg estimate','True distribution')
             ,col = c('blue','black','black','red')
                ,lty = c(2,1,3,1),lwd=c(1,1,2,2),bty = 'n')

  legend('topright',legend= c('w =',xcov), horiz=TRUE, bty = 'n')
  print(betamean)
  print(betadev)
}
```

8 References

[1] Aalen, O.O.(1995), 'Phase Type distributions in Survival Analysis', Scandinavian Journal of Statistics, Vol. 22, No. 4, pp. 447-463.

[2] Aalen, O.O. (1993), 'Phase Type distributions: computer algebra and a simple mixing model', Report of Medical Statistics, University of Oslo, Norway.

[3] Aslett, L.J.M. and Wilson, S.P. (2012), 'MCMC for Inference on Phase-type and Masked System Lifetime Models', partial Doctoral thesis, Department of Statistics, Trinity College Dublin, Dublin, Ireland.

[4] Ansell, J. I., & Phillips, M. J. (1994), Practical methods for reliability data analysis (No. 14), Oxford University Press, Oxford, U.K.

[5] Asmussen, S. Nerman, O. and Olsson, M. (1996), 'Fitting Phase-Type Distributions via the EM Algorithm', Scandinavian Journal of Statistics, Vol 23, No 4, pp. 419-441.

[6] Bladt, M., Gonzalez, A. and Lauritzen, S.A (2003), 'The estimation of phase-type related functionals using Markov chain Monte Carlo methods', Scandinavian Actuarial Journal, Issue 4, pp. 280-300.

[7] Boag, J.W. (1949). 'Maximum likelihood estimates of the proportion of patients cured by cancer therapy'. Journal of Royal Statistical Society, Series B, 11, 15-44.

[8] Braarud, I.H. (2012), 'Phase type and competing risk models in survival analysis', Master's project, Department of Mathematical Sciences, Norwegian University of Science and Technology.

[9] Fine, J.P. and Gray, R.J. (1999), 'A Proportional Hazards Model for the Subdistribution of a Competing Risk', Journal of the American Statistical Association, Vol. 94, No. 446, pp. 496-509.

[10] Gamerman, D. Lopes, H. (2006), 'Markov Chain Monte Carlo, Stochastic Simulation for Bayesian Inference', 2nd Edition, Chapman & Hall/CRC.

[11] Hoel, D.G (1972), 'A representation of mortality data by competing risks', Biometrics, Vol. 28, No. 2, Jun., International Biometric Society, Washington DC.

[12] Laache, C.H. (2014), 'Phase-type inference on competing risk models using Markov Chain Monte Carlo methods', Master's project, Department of Mathematical Sciences, Norwegian University of Science and Technology.

[13] Lindqvist, B.H. (2013), 'Phase-Type Distributions for Competing Risks'. Paper. Special Topic Session on Statistical analysis of competing risks data. 59th ISI World Statistics Congress. Hong Kong, China, 25-30 August 2013.

[14] Lindqvist,B.H. (2006), 'Competing Risks', In: Encyclopedia of Statistics in Quality and Reliability, Ruggeri, F., Kenett, R. and Faltin, F. W. (eds). John Wiley & Sons Ltd, Chichester, UK, pp 335-341.

[15] Pintilie M. (2007), 'Competing Risks: A Practical Perspective'. John Wiley & Sons; New York.

[16] Rausand, M. and Høyland, (2004), 'System Reliability Theory, Models, Statistical Methods, and Applications', 2nd Edition, Wiley.

[17] Ross, S.M. (2010), 'Introduction to Probability Models', 10th Edition, Academic Press.

[18] Scheike, T.H. and Zhang, M.J. (2011), 'Analyzing Competing Risk Data Using the R timereg Package', Journal of Statistical Software. Vol. 38. Issue 2.