1 Simplified models

Instead of tackling the complete problem at once, we start with a simplified version of the problem, solve this and then make the model incrementally more realistic.

1.1 First model

In our first and simplest model, the machine has a lifetime that follows some distribution $f(l)$ (cumulative function $F(l)$). When the machine breaks, it needs to be repaired and a certain cost for corrective maintenance needs to be paid (c_c). We start with an open loop problem where we have to choose at the beginning at what time the machine will be repaired. When the machine is repaired, the problem starts again with a new lifetime according to the same distribution.
We want to minimize the average cost. Since the machine is renewed after each repair, we can do this by minimizing the expected average cost until the first repair.

If we decide to repair the machine at some time $u > 0$, then the expected cost will be

$$J(u) = \frac{(1 - F(u))c_p}{u} + \int_0^u \frac{f(l)c_c}{l} dl$$

We want to minimize this, so we search for a zero of its derivative

$$\frac{d}{du} J(x) = -\frac{c_p}{u^2} - \frac{f(u)c_p}{u} + \frac{F(u)c_p}{u^2} + \frac{f(u)c_c}{u} = 0$$

We multiply by u^2 ($u > 0$ since preventive maintenance at time 0 would result in infinite average cost)

$$c_p F(u) - c_p + u f(u)(c_c - c_p) = 0$$

Which can be solved either numerically or algebraic for some specific distribution. For example, for lifetimes uniformly distributed on the interval $[0, L]$, this would result in solving

$$c_p \frac{u}{L} - c_p + \frac{u(c_c - c_p)}{L} = 0 \Rightarrow u = L\frac{c_p}{c_c}$$

1.2 Closed loop

We now make the problem a little more difficult by discretizing the time and having a limited set of options at each stage. There are two states, one where the machine is broken (s_b), and one where it is not (s_0). If the machine is broken, the only available action is u_c (corrective maintenance) with cost c_c. After u_c, the machine is renewed. When the machine is not broken, there are two actions:

- Preventive maintenance (u_p) with cost $c_p < c_c$, renewing the machine.

- Do nothing (u_w) with cost 0.

For simplicity we introduce a discount α and are interested in the discounted cost instead of the average cost. The value function would then be

$$J_k(s_0) = \min\{c_p + \alpha J_0(s_0), \alpha p_k J_{k+1}(s_0) + (1-p_k)J_{k+1}(s_b)\}$$

$$J_k(s_b) = c_c + \alpha J_0(s_0)$$

Where p_k denotes the probability that the machine does not break until the next stage. If we discretize time as $t_k = k\Delta$, this probability would be $\mathbf{P}(L > t_{k+1}|L > t_k) = \frac{1-F(t_{k+1})}{1-F(t_k)}$.

1.3 Fluid models

The last problem could be seen as a fluid model with one state with rate -1 and a random initial fluid level (ignoring the broken state for now). We could extend this to fluid models of more states. We introduce a fluid model with two states:

- s_0: With fluid rate $r_0 < 0$
- s_1: With fluid level $r_1 < 0$.

The system transitions in the fluid model occur as a CTMC when u_w is chosen and the machine does not break, when u_p is chosen, the system is renewed and transitions to s_0. When the machine breaks, it transitions to s_b from which the only available action is u_c which transitions to s_0.

To calculate the probability that the machine breaks between two stages, we need to have the distribution of the fluid decrease in a period of length Δ. Let $\overline{r} = \max r_0, r_1$ and $\underline{r} = \min r_0, r_1$. Let $\Delta\underline{r} < q < \Delta\overline{r}$, then the probability that the fluid decreases less than q in the next period, equals the probability that the machine spends less than $\frac{q-\underline{r}\Delta}{\overline{r}-\underline{r}}$ time in the state with the lowest rate. Let $f_i^j(t^*, t)$ denote the density of spending t^* out of t time in state j given that it starts in state i. We have that for small h:

$$f_0^1(t^*, t) = \lambda_0 h f_1^1(t^*, t-h) + (1-\lambda_0 h) f_0^1(t^*, t-h)$$

$$\Rightarrow \frac{f_0^1(t^*, t) - f_0^1(t^*, t-h)}{h} = \lambda_0(f_1^1(t^*, t-h) - f_0^1(t^*, t-h))$$

And in the limit this results in

$$\frac{d}{dt} f_0^1(t^*, t) = \lambda_0(f_1^1(t^*, t) - f_0^1(t^*, t))$$

Similarly, for $f_1^1(t^*, t)$ we have

$$(\frac{d}{dt} + \frac{d}{dt^*}) f_1^1(t^*, t) = \lambda_1(f_0^1(t^*, t) - f_1^1(t^*, t))$$

The first equation is an ordinary differential equation and can be solved using the method of the integrating factor, this leads to

$$f_0^1(t^*, t) = \int \lambda_0 f_1^1(t^*, t) e^{\lambda_0 t} dt e^{-\lambda_0 t}$$

(constants omitted for readability). The second equation is a partial differential equation and could be approached using the method of characteristics, which leads to

$$f_1^1(t_0, t_0 + x) = \int^x \lambda_1 f_0^1(t_0, t_0 + y) e^{\lambda_1 y} dy d^{-\lambda_1 x}$$

(Lower limit of the integral must be set such that $f_1^1(t_0, t_0) = e^{-\lambda_1 t_0}$) I currently have not found a way to finish these differential equations.

2 Preliminary Data Analysis

In this section, the given data will be explored and visualised.

2.0.1 Hazard Rate

Before trying to find a policy to optimally schedule preventive maintenance, it would be useful to first consider whether maintenance is helpful at all. To do this, we can consider the hazard rate of the machine. An increasing hazard rate results in a decreased expected lifetime over time while a decreasing hazard rate results in an increased expected lifetime. Below, you see a plot of this.

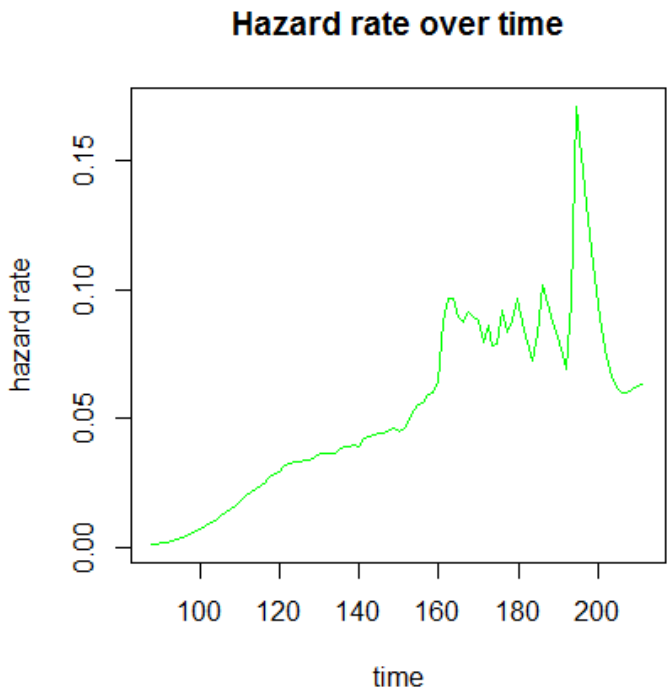

Figure 1: Hazard rate of the machine under consideration.

The hazard rate seems to be decrease after 200 hours, but this can be explained by the fact that there is only one trace that lasted over 200 hours.

Hence, we can conclude that the hazard rate rises over time (at least for the first 200 hours). This means that repairing would increase the expected life time of the machine.

2.1 Time Until Failure

To be able to predict the remaining time until a failure, it is helpful to know how the total lifetime of the machine is distributed. In this section we will attempt to fit the lifetime to a distribution.

2.1.1 Normality

The first guess for a fitting distribution would be a normal distribution. However, the Shapiro-Wilk normality test rejected the hypothesis that the lifetimes are normally distributed with a p-value of 1.26×10^{-5}. Hence, we can safely conclude that the data do not follow a normal distribution. This is also visible on the following quantile-quantile plot.

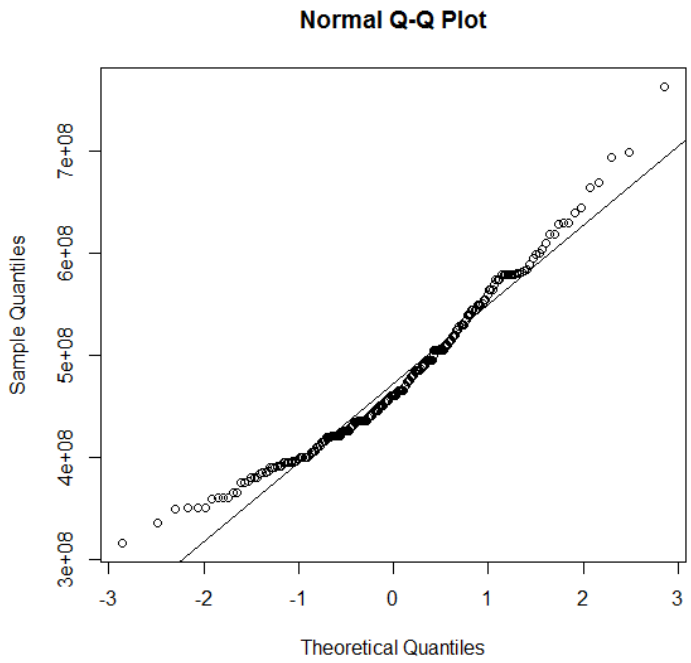

Figure 2: Quantile-quantile plot of the trace lifetimes.

2.1.2 Cullen and Frey graph

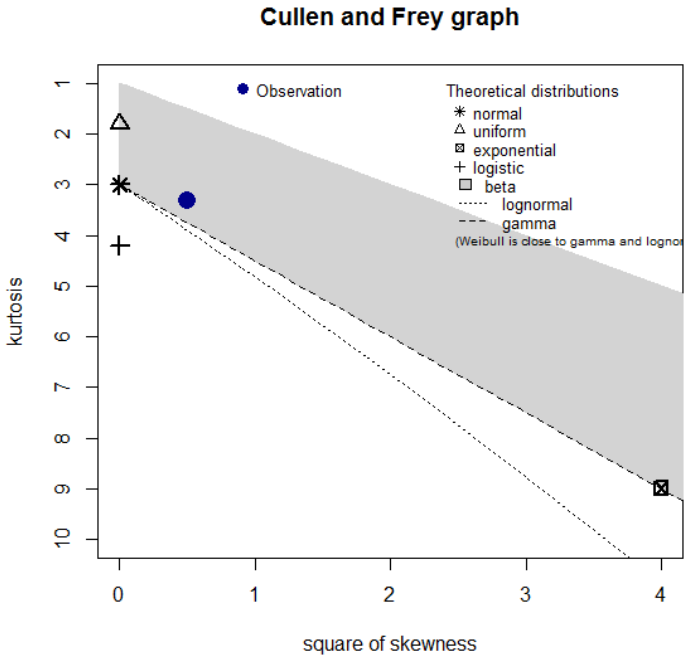

Figure 3: Cullen and Fray graph of the data.

Above, a Cullen and Fray graph is plotted. The data is placed based on its kurtosis and skewness. A few well-known distributions are also plotted on this plane. This plot would suggest a beta distribution. However, the beta distribution has a support of $[0, 1]$ while the lifetime is not bounded. Other distributions that have similar kurtosis and skewness are the Weibull distribution and the gamma distribution.
After estimating the parameters for each of these distributions and performing a Anderson-Darling Goodness-of-Fit test, the gamma-distribution seems to fit the data best with a p-value of 0.195 versus a p-value of 0.00444. Below, a plot shows the density of this distribution plotted over the observed density. As you can see, it does not fit the data very well.

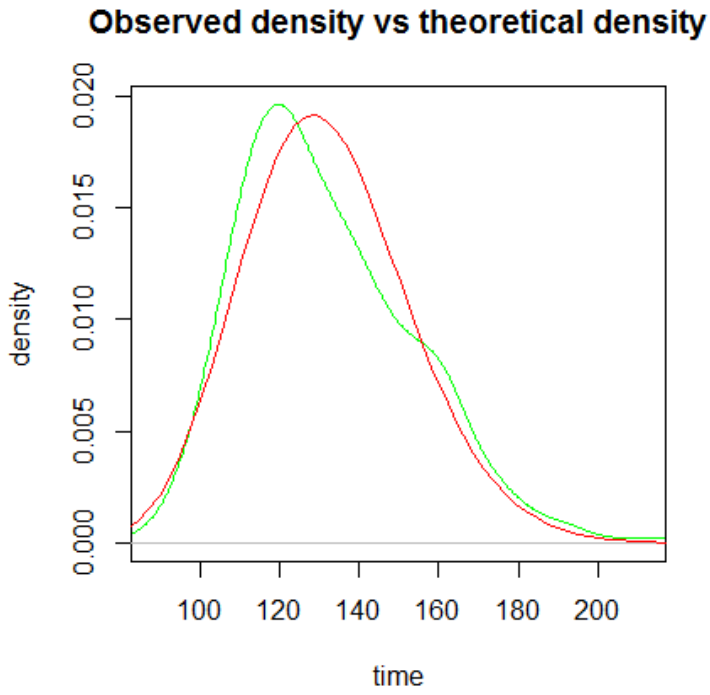

Figure 4: The observed density plotted over the density of the Gamma distribution

2.1.3 Phase-Type distribution

As the class of Phase-Type distributions is dense in the space of positive continuous distributions, a Phase-Type distribution could also be used to model the lifetimes. However, Phase-Type distributions have a few disadvantages: The number of parameters grows quadratically with the amount of states and a lot of these parameters are redundant. Furthermore, convergence of the EM-algorithm (to estimate the parameters) is slow and can get stuck in saddle points and local maxima [1].

2.1.4 Transition Times

If we view each event as a state and view switching from one event to another as a transition. Then we could model the machine as a transitioning system. It would then be interesting to see how the transition times are distributed; whether they are of the same family of distributions and whether they are independent.
When using the Anderson-Darling Goodness-of-Fit test for testing whether the transition times are exponentially distributed, the resulting p-values are rather low.

3 Technical report

During this project, several tools were used. In this section, it will be described how these tools were used. The encountered technical difficulties and their solutions or workarounds will also be reported.

3.1 ProM

ProM is an extensible framework that supports a wide variety of process mining techniques in the form of plug-ins[**?**]. During this project, ProM is used to analyse the event logs. Since it might be that not every process mining technique that is useful for this project is already implemented in ProM, I figured it would be useful to be able to create my own.

3.1.1 Plug-in development

To learn how to make such plug-ins, I found the following tutorial by M. Westergaard very helpful:
`https://westergaard.eu/category/prom-tutorial/`
Naturally, this short tutorial did not cover all information necessary to be able to deploy ProM plug-ins. Hence, I also had to occasionally consult ProM's Javadoc and forum. I used the Eclipse IDE to write the Java code for the plugin. There was some difficulty in being able to load the plugins into ProM since the ProM package manager gets the available plugins from a number of repositories on the `promtools.org` website and I was not able to add my own plugins to those repositories. After some research, trial and error, I came to the following workaround: The ProM package manager keeps a list of repositories in an xml-file that is locally stored in the 'packages'-folder. This file ('packages.xml') directs to a number of remote xml-files (also named 'packages.xml') which refer to zip-files which contain the jar-file of the plugin and a folder 'lib' which contains the jar-files of the libraries that the plug-in needs to be able to run. After downloading one of those remote xml-files, I figured out the format of these files and created a local packages.xml-file for my own plugins to which I referred to in the first packages.xml file as a new repository. I also exported my project as a jar-file and zipped it together with the required libraries. This allowed me to install my own package using the ProM package manager. The downside of this method is that it is very time-consuming to do all of this each time the plug-in is changed. Luckily, Eclipse also allows for running the plug-ins in debug mode. The downside of using debug mode, however, is that other plug-ins are not available unless you have imported their containing packages as libraries in the Eclipse project, and for certain plug-ins (mostly export and import plug-ins) it is difficult to find out to which package they belong. To workaround this issue, I created my own xes-importer so that I could test the plug-ins in debug mode. But since the full event-log of this project is rather large (over 3GB) and my basic xes-importer was not disk-buffered, I was only able to test the plug-ins on a small subset of the data in debug mode. Another problem that I encountered, is that some basic functions (for instance putIfAbsent() from java.util.HashMap) only worked in debug mode. This was hard to find out, but could easily be worked around by simply not using this function.

3.1.2 Plug-ins created

Below is a list of the various plug-ins that were created during this project together with a short description:

- **XesImportPlugin**: This plug-in imports xes-files, it was made to work around the fact that the existing importers were not available in debug mode.
- **DiscreteTimeMarkovChainPlugin**: This plug-in takes a XLog and produces a DiscreteTimeMarkovChain, using the events from the log as states.
- **StationaryDistributionPlugin**: This plug-in takes a DiscreteTimeMarkovChain and computes the eigenvalue-decomposition of the transition-matrix, from which the stationary distribution can be read. The output is a StationaryDistribution object.
- **StationaryDistributionExportPlugin**: This plug-in exports a StationaryDistribution object to a txt-file.
- **TraceLifetimesPlugin**: This plug-in takes an XLog as input and returns a list with the lifetimes of the traces. This list is encapsulated by an TraceLifetimes object.
- **TraceLifetimesExportPlugin**: This takes a TraceLifetimes object as input and stores the lifetimes in a csv-file so that they can easily be analysed using an R-script.

3.2 R

R is a scripting language for statistics. During this project, it has been used to perform statistical analysis on data extracted from the event logs.

4 Literature

- [1] Discusses the Phase-Type distribution and methods to estimate the parameters.
- [3] Book on dynamic programming and optimal control.
- [2] Introduces a new policy iteration-like algorithm for finding the optimal state costs or Q-factors and proves convergence properties.
- [4]
- [5] Concerns the Phase-Type distribution. Shows properties and proposes methods to estimate the parameters. Mentions the difficulties: Many parameters (n^2 for n states), of which, most are redundant but in most cases there is no canonical representation for each class ($2n-1$ independent parameters).
- [6] Uses BIDE to mine closed frequent sequential patterns.

- [7] A Survey of Sequential Pattern Mining. Describes general concepts, problem variants and the existing algorithms.
- [8] Intoduces Reward Model, Fluid Model (first and second order, reflecting and absorbing boundary) and Fluid Stochastic Petri Nets. Gives differential equations for transient behavior and proposes methods to solve/compute them.
- [9] Estimates transition probabilities for countably infinite state spaces.
- [10] Concerns Fluid Stochastic Petri Nets, summarizes the theory and provides examples.
- [11] Concerns estimating CTMC's with discretely observed data.
- [12] Considers three states (perfect, satisfactory and failed) and maintenance at scheduled (fixed interval) and unscheduled (exponential interval) times and finds the policy that minimizes the cost.
- [13] Proposes a method of modelling insurance losses via mixtures of Erlang distributions. Also explains why Phase-type distributions are not practical for this situation.
- [14] Models software faults with intervals of phase type distribution.
- [15] Introduces jumps that can occur at transition times, the size of the jumps are according to some distribution. The paper gives the resulting differential equations and methods to solve them.
- [16] Introduces the BIDE algorithm for finding closed frequent sequential patterns.

References

[1] Søren Asmussen, Olle Nerman, and Marita Olsson. Fitting phase-type distributions via the em algorithm. *Scandinavian Journal of Statistics*, pages 419–441, 1996.

[2] Dimitri P Bertsekas and Huizhen Yu. Q-learning and enhanced policy iteration in discounted dynamic programming. *Mathematics of Operations Research*, 37(1):66–94, 2012.

[3] Dimitri P Bertsekas, Dimitri P Bertsekas, Dimitri P Bertsekas, and Dimitri P Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena Scientific Belmont, MA, 1995.

[4] John R Birge and Francois Louveaux. *Introduction to stochastic programming*. Springer Science & Business Media, 2011.

[5] Peter Buchholz, Jan Kriege, and Iryna Felko. *Input modeling with phase-type distributions and Markov models: theory and applications*. Springer, 2014.

[6] Jun Chen and Ratnesh Kumar. Pattern mining for predicting critical events from sequential event data log. *IFAC Proceedings Volumes*, 47(2):1–6, 2014.

[7] Philippe Fournier-Viger, Jerry Chun-Wei Lin, Rage Uday Kiran, Yun Sing Koh, and R Thomas. A survey of sequential pattern mining. *Data Science and Pattern Recognition*, 1(1):54–77, 2017.

[8] Marco Gribaudo and Miklós Telek. Fluid models in performance analysis. In *International School on Formal Methods for the Design of Computer, Communication and Software Systems*, pages 271–317. Springer, 2007.

[9] Monir Hajiaghayi, Bonnie Kirkpatrick, Liangliang Wang, and Alexandre Bouchard-Côté. Efficient continuous-time markov chain estimation. In *ICML*, pages 638–646, 2014.

[10] Graham Horton, Vidyadhar G Kulkarni, David M Nicol, and Kishor S Trivedi. Fluid stochastic petri nets: Theory, applications, and solution techniques. *European Journal of Operational Research*, 105(1):184–201, 1998.

[11] Yasunari Inamura et al. Estimating continuous time transition matrices from discretely observed data. Technical report, Bank of Japan, 2006.

[12] Szilard Kalosi, Stella Kapodistria, and Jacques AC Resing. Condition-based maintenance at both scheduled and unscheduled opportunities. *arXiv preprint arXiv:1607.02299*, 2016.

[13] Simon CK Lee and X Sheldon Lin. Modeling and evaluating insurance losses via mixtures of erlang distributions. *North American Actuarial Journal*, 14 (1):107–130, 2010.

[14] Hiroyuki Okamura and Tadashi Dohi. Building phase-type software reliability models. In *Software Reliability Engineering, 2006. ISSRE'06. 17th International Symposium on*, pages 289–298. IEEE, 2006.

[15] Elena I Tzenova, Ivo JBF Adan, and Vidyadhar G Kulkarni. Fluid models with jumps. *Stochastic models*, 21(1):37–55, 2005.

[16] Jianyong Wang and Jiawei Han. Bide: Efficient mining of frequent closed sequences. In *Data Engineering, 2004. Proceedings. 20th International Conference on*, pages 79–90. IEEE, 2004.