The masked.data package

Introduction

The R package `masked.data` is a framework for estimating the parameters of latent node lifetimes from *masked data* in a series system. Presently, we only provide support for series systems with node lifetimes in the `exponential` and `lomax` parametric families.

Install the library with:

```
devtools::install_github("queelius/masked.data")
```

Once installed, you may load the library with:

```
# load the masked data library
library(masked.data)
# load other dependencies
library(tidyverse)
```

Masked data

Masked data is given by an i.i.d. sample of system lifetime data and *plausible* candidate sets that contain the failed node.

The `masked.data` R package contains several simulated masked data sets. For example, a series system with 10 exponentially distributed nodes is stored in `exp_series_data_3`:

```
print(exp_series_data_3,drop_latent=T,pprint=T)
#> # A tibble: 100,000 x 3
#>          s     w C
#>      <dbl> <int> <chr>
#>  1 0.00524     5 2, 4, 5, 7, 8
#>  2 0.0277      3 5, 6, 9
#>  3 0.0487      3 3, 5, 7
#>  4 0.00223     2 6, 9
#>  5 0.00521     7 1, 3, 4, 6, 7, 9, 10
#>  6 0.00913     8 1, 2, 5, 6, 7, 8, 9, 10
#>  7 0.0356      6 2, 3, 5, 6, 8, 9
#>  8 0.0264      7 1, 2, 3, 4, 5, 9, 10
#>  9 0.0156      5 1, 2, 4, 5, 6
#> 10 0.00701     7 1, 4, 5, 6, 8, 9, 10
#> # ... with 99,990 more rows
```

You can get help on any object in `masked.data` using the built-in help. For instance, to get information on the data set `exp_series_data_1`, type `?exp_series_data_1` in your R console.

Statistical model

TODO: Cut-and-paste a lot of the material in statistical model section and distribution derivation section into here, but only just enough to be able to derive the log-likelihood. We will also need a few key distribution

functions and hazard functions for some of the other material.

In this vignette, We consider two candidate set models, m_0 and m_1.

Candidate model m_0

In model m_0, the j-th observation in the masked data has a candidate set C_j that contains the failed component, in addition to a random selection (without replacement) of $w_j - 1$ additional components.

Candidate model m_1: α-masked candidates

In model m_1, the j-th observation in the masked data has a candidate set C_j described by the following:

1. C_j contains the failed node with probability α_j. Additionally, it contains a random selection (without replacement) of $w_j - 1$ non-failed nodes.
2. C_j contains a random selection (without replacement) of w_j non-failed nodes with probability $1 - \alpha_j$.

Exponential series system

The most straightforward series system to estimate is the series system with exponentially distributed node lifetimes.

Candidate model m_0

Suppose an exponential series system with m nodes is parameterized by $\theta = (3, 4, 5)'$. Then, the j-th node has an exponentially distributed lifetime with a failure rate θ_j.

In what follows, we consider the case of a series system with exponentially distributed node lifetimes under candidate model m_0, as described earlier. Later, we consider model m_1, the α-masked candidate model.

The candidate model m_0 for masked data (of size $n = 1000$) for this exponential series system, with $w = 2$ candidates for each observation, is given by:

```
n <- 1000
theta <- c(3,4,5)
m <- length(theta)
w <- rep(2,n)
# todo: rename to md_exp_series_data_m0
md <- md_exp_series(n,theta,w,md_candidate_m0)
#> Warning: The `x` argument of `as_tibble.matrix()` must have unique column names if `.name_repair` is
#> Using compatibility `.name_repair`.
#> This warning is displayed once every 8 hours.
#> Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated.
print(md,pprint=T,drop_latent=T)
#> # A tibble: 1,000 x 3
#>         s     w C
#>     <dbl> <dbl> <chr>
#>  1 0.0986     2 1, 2
#>  2 0.411      2 2, 3
#>  3 0.0104     2 2, 3
#>  4 0.0968     2 2, 3
#>  5 0.0550     2 2, 3
#>  6 0.117      2 1, 2
#>  7 0.0918     2 1, 2
#>  8 0.0970     2 1, 3
#>  9 0.203      2 2, 3
```

```
#> 10 0.0956     2 2, 3
#> # ... with 990 more rows
```

In the above, we used the function `md_exp_series`. To get more help on it, type `?md_exp_series`.

Log-likelihood of θ given masked data

The log-likelihood function (technically, it is the log of the likelihood kernel) is given by

$$\text{kloglik}(\theta) = -\left(\sum_{i=1}^{n} s_i\right)\left(\sum_{j=1}^{m}\right) + \sum_{i=1}^{n} \log\left(\sum_{k\in c_i} \lambda_k\right).$$

The following log-likelihood constructor, `md_kloglike_exp_series_m0`, is implemented using minimally sufficient statistics, which significantly improves the computational efficiency of computing the log-likelihood.

We compute the log-likelihood function as a function of the masked data `md` with:

```
kloglik <- md_kloglike_exp_series_m0(md)
```

The log-likelihood function contains the maximum amount of information about parameter θ given the sample of masked data `md`.

Suppose we do not know that $\theta = (3, 4, 5)'$. With the log-likelihood, we may estimate θ with $\hat{\theta}$ by solving

$$\hat{\theta} = \text{argmax}_\theta \, \text{kloglik}(\theta),$$

i.e., finding the point that *maximizes* the log-likelihood on the observed sample `md`. This is known as *maximum likelihood estimation*.

We typically solve for the MLE by solving

$$\nabla \, \text{kloglik}(\theta)|_{\theta=\hat{\theta}} = 0.$$

We use the iterative method known as the Fisher scoring algorithm to solve this,

$$\theta^{(n+1)} = \theta^n + I^{-1}(\theta^n) \nabla \, \text{kloglik}(\theta^n),$$

where I is the information matrix (or observed information matrix).

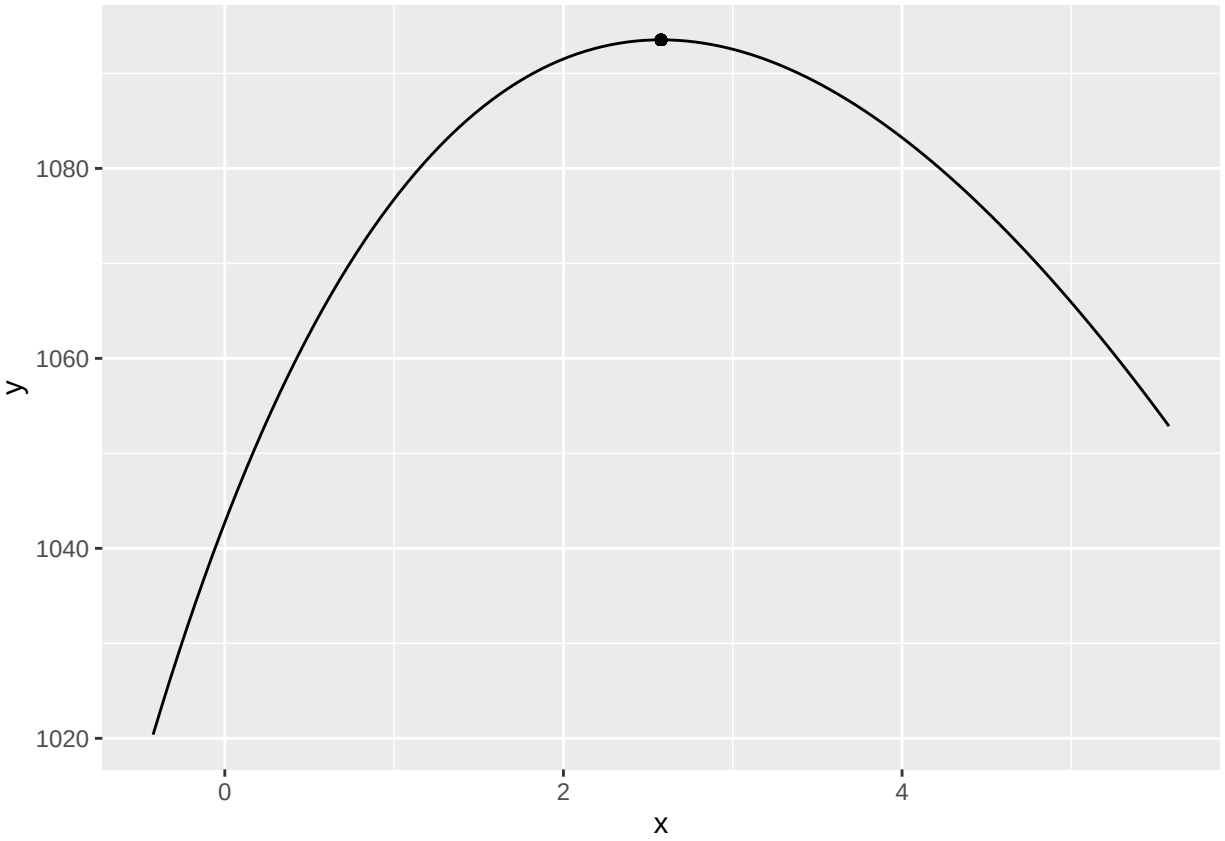
The function `md_mle_exp_series_m0` is a small wrapper for this function, providing `md_fisher_scoring` with the appropriate arguments. We find $\hat{\theta}$ by running the following R code:

```
mle <- md_mle_exp_series_m0(md)
theta.hat <- point(mle)
print(round(theta.hat,digits=3))
#>       [,1]
#> [1,] 2.576
#> [2,] 3.997
#> [3,] 5.466
```

The function `md_mle_exp_series_m0` returns an `md_estimate` object, which has various methods implemented for it, e.g., `confint` (computes the estimators confidence interval). In the above, we see use of the `point` method, which takes an estimator object and obtains its *best* point estimate, or in this case, the MLE $\hat{\theta}$. We see that $\hat{\theta} = (2.576, 3.997, 5.466)$.

If we let the third argument in the log-likelihood function be fixed at $\hat{\theta}_3 = 5.466$, then we may profile the log-likelihood function over the first two parameters:

```r
prof <- function(theta1) { kloglik(c(theta1,theta.hat[2],theta.hat[3])) }
data <- tibble(x=seq(theta.hat[1]-3,theta.hat[1]+3,.05))
data$y <- numeric(nrow(data))
for (i in 1:nrow(data))
    data$y[i] <- prof(data$x[i])
data %>% ggplot(aes(x=x,y=y)) + geom_line() +
    geom_point(aes(x=theta.hat[1],prof(theta.hat[1])))
```



Due to sampling variability, different runs of the experiment will result in different outcomes, i.e., $\hat{\theta}$ has a sampling distribution. We see that $\hat{\theta} \neq \theta$, but it is reasonably close. We may measure this sampling variability using the variance-covariance matrix, bias, mean squared error (MSE), and confidence intervals.

Variance-covariance matrix

The estimator $\hat{\theta}$ as a function of a random sample of masked data has a sampling distribution.

Information matrix Theoretically, $\hat{\theta}$ converges in distribution to the multivariate normal with a mean θ and a variance-covariance given the the inverse of the observed Fisher matrix,

$$J(\theta) = -\nabla^2 \operatorname{loglik}(\theta).$$

Since J is a function of a random sample, it is a random matrix.

However, when the nodes are exponentially distributed, we may take the expectation

$$I(\theta) = -E(\nabla^2 \operatorname{loglik}(\theta)),$$

and derive a closed-form solution.

The inverse of the information matrix I (or J) computes the variance-covariance matrix. For 3-out-of-3 exponential series system, this results in

$$V(\theta) = \frac{\lambda_1 + \lambda_2 + \lambda_3}{n} \begin{pmatrix} \lambda_1 + \lambda_2 + \lambda_3 & -\lambda_3 & -\lambda_2 \\ -\lambda_3 & \lambda_1 + \lambda_2 + \lambda_3 & -\lambda_1 \\ -\lambda_2 & -\lambda_1 & \lambda_1 + \lambda_2 + \lambda_3 \end{pmatrix}.$$

Asymptotically, $\hat{\theta}$ is the UMVUE, i.e., it is unbiased and obtains the minimum sampling variance. An estimate of the variance-covariance V may be obtained with:

```
(V.hat <- vcov(mle))
#>              [,1]        [,2]        [,3]
#> [1,]  0.14493168 -0.06579898 -0.04811732
#> [2,] -0.06579898  0.14493168 -0.03101538
#> [3,] -0.04811732 -0.03101538  0.14493168
```

Bootstrap: sample mean and sample variance-covariance Another way we may estimate the variance-covariance matrix is with parameter Bootstrapping.

Using the MLE $\hat{\theta}$, we may sample from the masked data model with:

```
B <- 200 # bootstrap replicates
theta.hat.sim <- matrix(nrow=B,ncol=length(theta.hat))
for (i in 1:B)
{
    theta.hat.sim[i,] <- as.vector(point(md_mle_exp_series_m0(
        md_exp_series(
            n=n,
            theta=theta.hat,
            w=w,
            candidate_model=md_candidate_m0))))
}
```

Now that we have Bootstrapped sample of MLEs, $\hat{\theta}^{(1)}, \ldots, \hat{\theta}^{(B)}$, we can compute its sample covariance to estimate V:

```
(V.bs <- cov(theta.hat.sim))
#>              [,1]        [,2]        [,3]
#> [1,]  0.14478123 -0.07171169 -0.04928572
#> [2,] -0.07171169  0.16179076 -0.02891225
#> [3,] -0.04928572 -0.02891225  0.12847910
```

This estimate of the variance-covariance matrix is similar to the inverse Fisher information matrix.

Bias, MSE, and confidence intervals

We would like to measure the accuracy and precision of $\hat{\theta}$. In statistical literature, the bias

$$\mathrm{b}(\hat{\theta}) = E(\hat{\theta}) - \theta$$

is a measure of accuracy and variance is a measure of precision.

The mean squared error, denoted by MSE, is a measure of estimator error that incorporates both the bias and the variance,

$$\mathrm{MSE}(\hat{\theta}) = \mathrm{trace}(V) + \mathrm{b}^2(\hat{\theta}).$$

Since $\hat{\theta}$ is asymptotically unbiased and minimum variance,

$$\lim_{n \to \infty} \mathrm{MSE}(\hat{\theta}) = \mathrm{trace}(V).$$

Thus, for sufficiently large samples, $\text{MSE}(\hat{\theta})$ is approximately given by the `trace` of the estimated variance-covariance matrix:

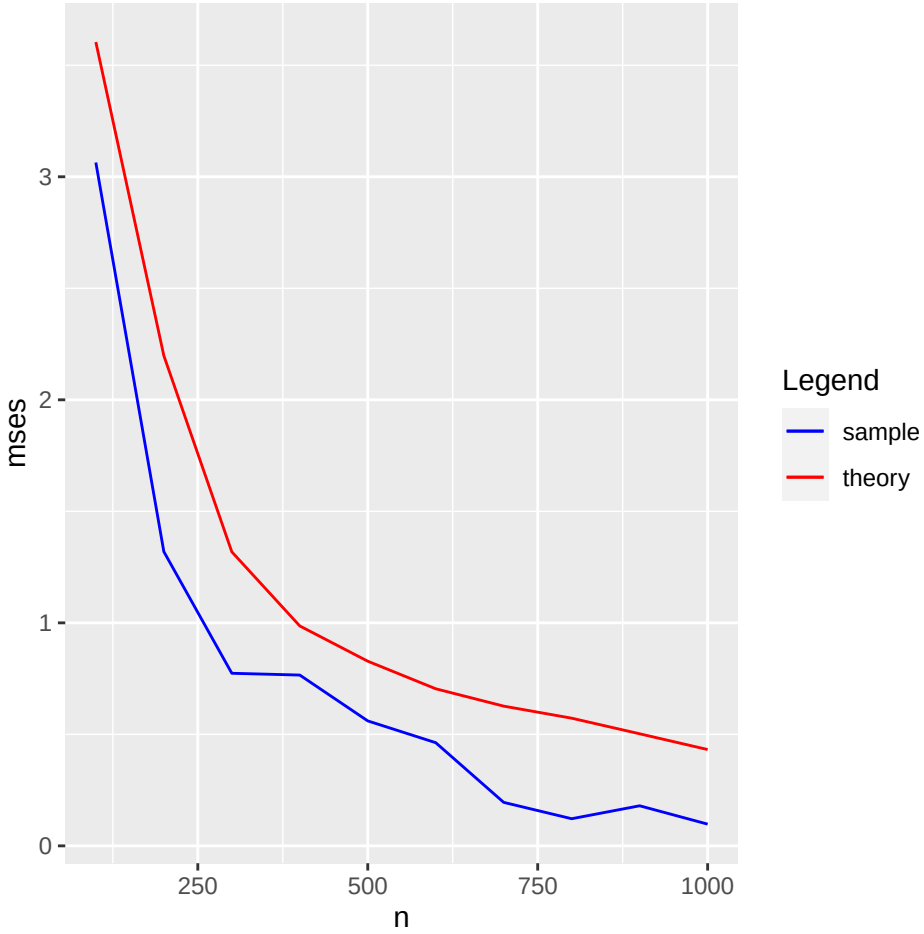
```
(mse <- sum(diag(V.hat)))
#> [1] 0.434795
```

Mean squared error as a function of sample size n:

```
Ns <- seq(100, nrow(exp_series_data_1), 100)
mses.vcov <- numeric(length=length(Ns))
mses <- numeric(length=length(Ns))

for (i in 1:length(Ns))
{
    mses.vcov[i] <- sum(diag(vcov(md_mle_exp_series_m0(exp_series_data_1[1:Ns[i],]))))
    mses[i] <- mean((point(md_mle_exp_series_m0(exp_series_data_1[1:Ns[i],])) - theta)^2)
}

tibble(mse=mses,mses.vcov=mses.vcov,n=Ns) %>% ggplot() +
    geom_line(aes(x=n,y=mses,color="sample")) +
    geom_line(aes(x=n,y=mses.vcov,color="theory")) +
    scale_color_manual(name="Legend",values=c("sample"="blue", "theory"="red"))
```



A primary statistic is the *confidence interval*. A $(1-\alpha)100\%$ confidence interval for θ_j may be estimated with $\hat{\theta}_j \pm z_{1-\alpha/2}\sqrt{\hat{V}_{jj}}$. We provide a method for doing this calculation:

```
as_tibble(confint(mle)) %>% mutate(length=.[[2]]-.[[1]])
#> # A tibble: 3 x 3
#>   `2.5%` `97.5%` length
#>    <dbl>   <dbl>  <dbl>
#> 1   1.95    3.20   1.25
#> 2   3.37    4.62   1.25
#> 3   4.84    6.09   1.25
```

How does this compare to the confidence intervals given that all candidate sets in the sample have size $w = 1$?

```
as_tibble(confint(md_mle_exp_series_m0(
    md_exp_series(n=n,
                  theta=theta,
                  w=rep(1,n),
                  candidate_model=md_candidate_m0)))) %>%
    mutate(length=.[[2]]-.[[1]])
#> # A tibble: 3 x 3
#>   `2.5%` `97.5%` length
#>    <dbl>   <dbl>  <dbl>
#> 1   2.49    3.10  0.614
#> 2   4.21    4.82  0.614
#> 3   4.86    5.47  0.614
```

We see that the lengths of the confidence intervals are significantly shorter.

If *no* information is provided about the node failure in a series system with m nodes ($w_i = m$ for $i = 1, \ldots, n$), then the m_0 estimator is inconsistent.

Sampling distribution of $\hat{\theta}$

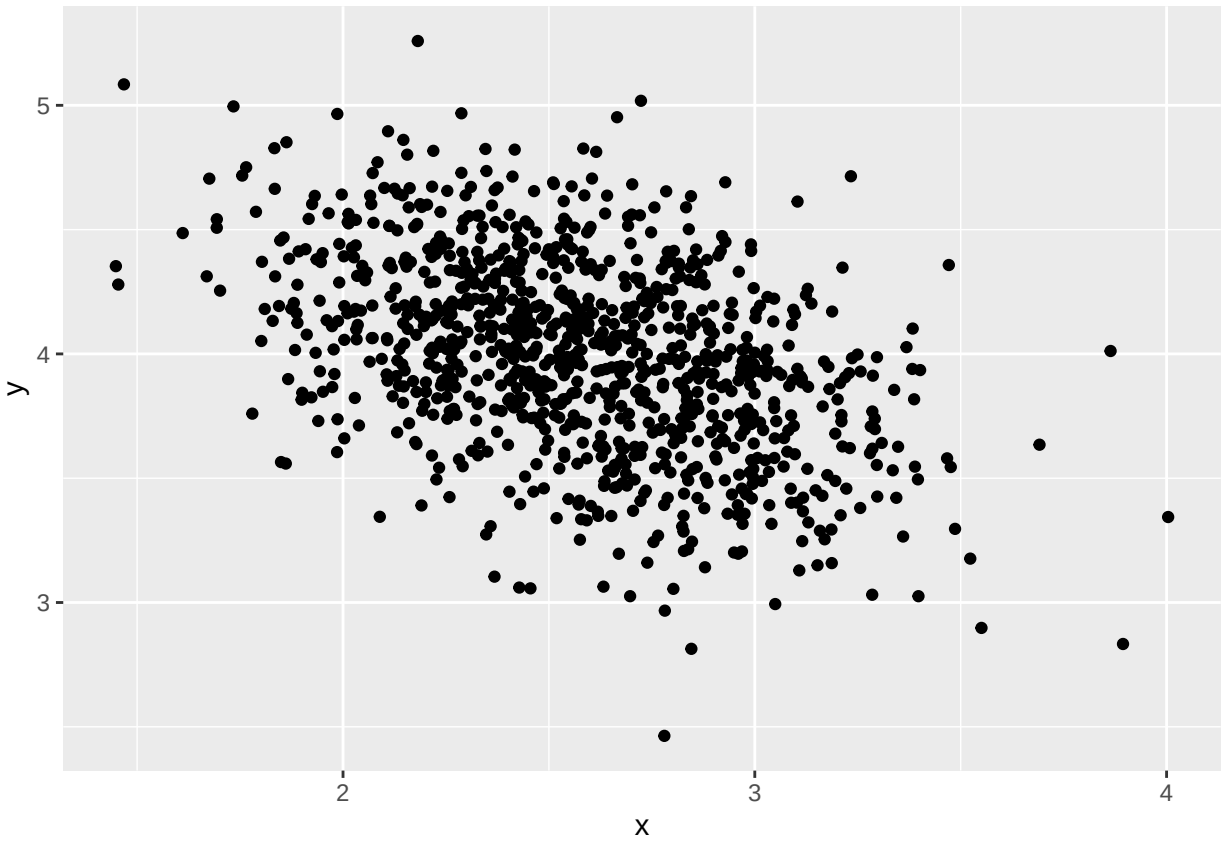
We know that $\hat{\theta} \sim \mathcal{N}(\theta, I^{-1}(\theta))$. We can estimate the sampling distribution of $\hat{\theta}$ with $\mathcal{N}(\hat{\theta}, I^{-1}(\hat{\theta}))$.

This makes it trivial to estimate any other function of θ by sampling from the approximation:

```
samp <- sampler(make_normal(point(mle),vcov(mle)))
mles <- samp(n=1000)
```

We show a contour plot of the first two components of each point in the MLE sample with:

```
ggplot(tibble(x=mles[,1],y=mles[,2]),aes(x=x,y=y)) + geom_point()
```



Estimating node failures from a sample

Another characteristic we may wish to estimate is the probability that a particular node in an observation caused the system failure.

We wish to use as much information as possible to do this estimation. We consider three cases:

1. We have masked data `md` with candidate sets and system failure times and seek to estimate the node failure probabilities of observations in this data. This case provides the most accurate estimates of the node probability failures, as have both system failure times and candidate sets as predictors of the node failure.
2. We have a new observation of a system failure time and an estimate of θ from `md`. In this case, we cannot condition on candidate sets, since the observation does not include that information. However, we do have a system failure time.
3. We have an estimate of θ from `md` but wish to predict the node failure of a system that has failed, but we do not know when it failed.

We consider case 1 describe above where we have masked data `md` that includes both candidate sets and system failure times.

In this case, we are interested in

$$f_{K|C,S}(k|c,s,\theta) = \frac{h_k(s)}{\sum_{j\in c} h_j(s)},$$

which in the exponential series case simplifies to

$$f_{K|C,S}(k|c,s,\theta) = \frac{\lambda_k}{\sum_{j\in c} \lambda_j}.$$

We model this probability distribution with `md_exp_series_node_failure_m0`. We also provide a decorator function `md_series_node_failure_decorator_m0` that accepts masked data as input and an `md_exp_series_node_failure_m0` object and returns the masked data with columns for node failure probabilities given by `k.1`,...,`k.m`.

```
fk <- md_exp_series_node_failure_m0(theta.hat)
md %>% select(-c("w",starts_with("t."))) %>%
    md_series_node_failure_decorator_m0(fk)
#> # A tibble: 1,000 x 8
#>         s     k c.1   c.2   c.3     k.1   k.2   k.3
#>     <dbl> <int> <lgl> <lgl> <lgl> <dbl> <dbl> <dbl>
#>  1 0.0986     1 TRUE  TRUE  FALSE 0.392 0.608 0
#>  2 0.411      2 FALSE TRUE  TRUE  0     0.422 0.578
#>  3 0.0104     3 FALSE TRUE  TRUE  0     0.422 0.578
#>  4 0.0968     3 FALSE TRUE  TRUE  0     0.422 0.578
#>  5 0.0550     3 FALSE TRUE  TRUE  0     0.422 0.578
#>  6 0.117      1 TRUE  TRUE  FALSE 0.392 0.608 0
#>  7 0.0918     2 TRUE  TRUE  FALSE 0.392 0.608 0
#>  8 0.0970     1 TRUE  FALSE TRUE  0.320 0     0.680
#>  9 0.203      3 FALSE TRUE  TRUE  0     0.422 0.578
#> 10 0.0956     3 FALSE TRUE  TRUE  0     0.422 0.578
#> # ... with 990 more rows
```

We notice that every row over the columns `k.1`, `k.2`, and `k.3` given a specific candidate set are the same. This is as expected, since in the case of the exponential series, the node failure rates $f_{K|C,S}$ are not a function of system failure time.

If we already had an estimate of θ and we sought to predict the failed nodes from only system lifetime data, we would just let the candidate sets contain all of the nodes.

Also, observe that the node failure probabilities

$$\hat{k}(\theta) = (\hat{k}_1, \hat{k}_2, \hat{k}_3)'$$

is a random vector whose sampling distribution is a multivariate normal

$$\hat{k}(\theta) \sim \mathcal{N}(k(\theta), V(k(\theta))).$$

While these parameters could possibly be derived as functions of θ, earlier we discussed how we can simulate draws from $\hat{\theta}$ and applying the statistic of interest.

For instance, once we have a simulated sample $\hat{k}^{(1)}, \ldots, \hat{k}^{(B)}$, we can operate on this data to estimate desired characteristics like confidence intervals for $k(\theta)$.

Estimating system lifetime intervals from candidate sets

Suppose we do not know the system failure time, but have masked data with candidate sets and we have already estimated the parameters of the series system.

Then, we can predict system failure times given these candidate sets. Observe that

$$f_{S|C}(s|c, \theta) = f_{S,C}(c, s|\theta)/f_C(c)$$

which simplifies to thinking about the series system as being over the subset of nodes in c. Thus,

$$S|c = \min\{T_j | j \in c\}.$$

We provide an estimate of the shortest interval that includes the system failure time $S|c$ with, say, 90% probability. In the case of the exponential series system, this is just $(0, F^{-1}_{S|c}(0.95))$.

We consider a larger and more complicated data set for this next example, `exp_series_data_3`. Type `?exp_series_data_3` to learn more about it. It has a true parameter value of $\theta = (3, 5, 4, 6, 7, 2, 8, 9, 10, 11)'$. We decorate its masked data with these upper and lower bounds and compare the result with the known system lifetimes, column `s`:

```
(theta.large.hat <- point(md_mle_exp_series_m0(exp_series_data_3)))
#>             [,1]
#>  [1,]  3.136045
#>  [2,]  4.732481
#>  [3,]  4.188199
#>  [4,]  5.675374
#>  [5,]  7.414832
#>  [6,]  2.271020
#>  [7,]  7.876326
#>  [8,]  8.908206
#>  [9,] 10.092389
#> [10,] 11.238213
q.hat <- md_exp_series_system_failure_interval_m0(theta.large.hat,.85)
(data <- exp_series_data_3 %>% md_series_system_failure_decorator_m0(q.hat) %>%
    select(-c("w","k",starts_with("t."),starts_with("c."))) %>%
    mutate(contains = s >= s.lower && s <= s.upper))
#> # A tibble: 100,000 x 4
#>          s s.lower s.upper contains
#>      <dbl>   <dbl>   <dbl> <lgl>
#>  1 0.00524       0  0.0548 TRUE
#>  2 0.0277        0  0.0959 TRUE
#>  3 0.0487        0  0.0974 TRUE
#>  4 0.00223       0  0.153  TRUE
#>  5 0.00521       0  0.0427 TRUE
#>  6 0.00913       0  0.0341 TRUE
#>  7 0.0356        0  0.0504 TRUE
#>  8 0.0264        0  0.0408 TRUE
#>  9 0.0156        0  0.0817 TRUE
#> 10 0.00701       0  0.0389 TRUE
#> # ... with 99,990 more rows
```

We can compute the proportion that contain the system lifetime with:

```
mean(data$contains)
#> [1] 0.952
```

We were expecting something closer to 0.85, but instead we obtained a result of around 0.95. TODO: investigate!

Candidate model m_1

Here, we derive similar results for the α-masked model.

TODO: finish this section, but provide fewer examples and details.

Lomax series systems

```
print(lomax_series_data_1,pprint=T,drop_latent=T)
#> # A tibble: 10,000 x 3
#>          s     w C
```

```
#>       <dbl> <dbl> <chr>
#>  1 0.0309      2 1, 2
#>  2 0.0248      2 1, 3
#>  3 0.00661     2 1, 2
#>  4 0.0367      2 2, 3
#>  5 0.0562      2 2, 3
#>  6 0.00641     2 1, 3
#>  7 0.0740      2 1, 2
#>  8 0.0289      2 1, 3
#>  9 0.0121      2 2, 3
#> 10 0.00952     2 2, 3
#> # ... with 9,990 more rows
```

TODO: fix likelihood function, the output seems wron – the algorithms I've tried either do not converge to a solution, or converge to the wrong solution. TODO: plot the surface of the profile of the log-likelihood, like I did for exponential. If it seems to be large in the wrong area (given that we roughly know what the MLE should be), find out why. If it just looks highly non-linear, then look into random restarts and other iterative methods, although starting near the true parameter value in the search should largely avoid the problem. If I can't figure it out, move to Weibull instead.