GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN

Imputing censored covariates with distributional regression - GAMLSS

Tim Ruhkopf, Petros Christanas

tim.ruhkopf@outlook.de, petroschristanas@gmail.com

Statistical Programming with R

Supervisor

Maike Hohberg

September 3, 2018

Introduction

The purpose of the package *Imputegamlss* is to provide a way for imputing values of a covariate which are defected, i.e. missing or censored. To do so, we make use of the MICE Algorithm, which imputes missing valus at random. In addition, we apply inverse sampling to utilize the additional information contained in the values at which censoring occured.

Basic Assumption

Data are generated and analyzed according to $y_i = \alpha + x_i^T\beta + u_i,\ i = 1, ..., N$,
where β the parameter vector of main interest, u_i latent iid error with $E(u_i) = 0$.
Unknown parameters are estimated by OLS.

Without loss of generality, missing or censored values are generated in the first predictor x_{i1}, where $w_i = (y, x_{i,-1}^T)^T$.

Further, let $x_{1,obs}$ be the vector of observed x_{i1}'s and W_{obs} the matrix of all w_i's for which x_{i1} is observed. Correspondingly, $x_{1,mis}$ and W_{mis} are denoting missing cases (if we have censoring instead of missing, we denote *cens*).

NOTE: In imputation models, x_{i1} becomes the response variable.

The MICE Algorithm (Multiple Imputation by Chained Equations)

Step 1
Using the observed data $\{x_{1,obs}, W_{obs}\}$, w.l.o.g. fit the below model as follows:

For $x_{1,obs} \sim \mathcal{D}(\mu, \sigma)$ where $\mathcal{D}$ some specified gamlss family distribution with

$$\mu = g_1^{-1}(\tilde{\alpha}_{(1)} + \sum_{j=1}^{k_1} h_{1j}(w_j)) \quad \& \quad \sigma = g_2^{-1}(\tilde{\alpha}_{(2)} + \sum_{j=1}^{k_2} h_{2j}(w_j)),$$

(For additional scale and shape parameters $x_{1,obs} \sim \mathcal{D}(\mu, \sigma, \nu, \tau)$)

where $g_p^-1(\cdot),\ p = 1, 2$, are inverse monotonic link functions,
$j = 1, ...k_p$ and $h_{pj}(\cdot)$ the type of effect of the j-th covariate.

Step 2
Resample $x_{1,obs}$ as follows:

$$x_{1,obs}^* \sim \mathcal{D}(\hat{\mu}, \hat{\sigma} \mid W_{obs})$$

Draw a bootstrab sample B from the observation set $\{x_{1,obs}^*, W_{obs}\}$.

Step 3
Refit the model using the bootstrap sample B. Draw n_{mis} imputations for $x_{1,mis}$ as follows:

$$\tilde{x}_{1,mis} \sim \mathcal{D}(\dot{\mu}, \dot{\sigma} \mid W_{mis})$$

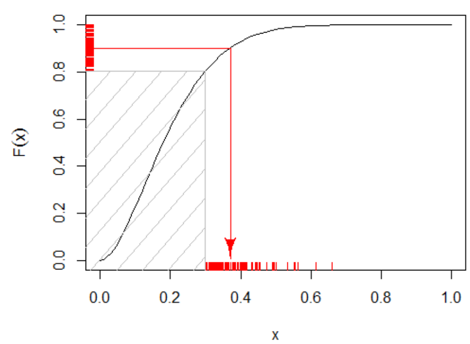
Step 4
Repeat step 2 and 3 m independent times, where m the number of imputations (number of rounds the algorithm is run).

Implementation

In step 3, the MICE algorithm doesn't take into account the information contained in the censored values. To deal with this issue and trying to represent the conditional distribution of the true censored values as accurate as possible, we employ inverse sampling.

Inverse sampling:
We estimate the effects of the covariates W_{obs} on $x_{1,obs}$ and predict the parameters of the censored observations' distributions under the specified family assumption. We thereby predicted the associated cumulative distribution functions (cdf) of $x_{1,mis}$. Then, in case of e.g. right censoring, we can draw random values from a uniform distribution with $min = p_i = F_i(x_{i1,cens})$ and $max = 1$, $i = 1, ..., n_{mis}$.
Evaluating these draws in the respective quantile function, we sample for each observation from the valid regions of the m fitted distributions $\mathcal{D}_j(\dot{\mu}, \dot{\sigma} \mid B_j;\, W_{mis};\, x_{1,mis})$, $j = 1, ..., m$.

Figure 1: Inverse Sampling



imputex()

The function *imputex* is the main function of our package which serves the actual purpose, namely imputing censored covariates. *imputex* can of course also impute data missing at random.

To improve computational performance, we re-arranged the algorithm in such a way, that all m rounds of imputation (step 4) are mostly done simultaneously. The internal workflow of the algorithm now goes as follows:

Step 1

- Split dataset in fully observed data and missing/censored data (W_{obs} & W_{cens}).
- Fit a gamlss model based on W_{obs} with user specified family and formula on the observed data.

Step 2

- Resample m independent draws of size nrow(W_{obs}) from the fitted model using the helper funtion *family_fun* for random number generation. In particular, from the conditional distributions of the oberved x_{i1} values, *family_fun* predicts the distributional parameters of $x_{1,mis}$. From there, we draw a sample of size nrow(W_{obs})*m, which yields a stacked vector.
- Unstack the vector such that there are m columns, each column representing an imputation round.
- We perform m row-wise bootstraps from the previous simulation to receive a better approximation of these distributions. This yields m new bootstrap samples $B_j := \{x^*_{1,obs}, W_{obs}\}$, $j = 1, ..., m$.

Step 3

- Using B_j, $j = 1, ..., m$, fit the respective models.
- From each of the m*n_{mis} fitted distributions, draw a sample from the respective valid regions, using the helper function *samplecensored* and save the resulting proposal vectors columns-wise.
  Note that in *samplecensored* the distributions are being rescaled to the appropriate regions such that, in case of right censoring, $\int\limits_{x_{i,cens_L}}^{\infty} f(x)dx = 1$, given the restrictions induced by the respective censoring bounds (and analogously for left censoring $\int\limits_{-\infty}^{x_{i,cens_U}} f(x)dx = 1$ and for interval censoring $\int\limits_{x_{i,cens_L}}^{x_{i,cens_U}} f(x)dx = 1$).
- Each of our censored observations has now m proposal values as imputation canditates. For each proposal vector, we choose the median as replacement value since it's robust against outliers. As a result, we obtain the vector of replacement values *imputedx*.

Use case

Let's get right into the usage of the function. First, we simulate a data set with the function *simulateData* (which is also at the user's disposal), where e.g. the dependent variable is linearly related to the predictors. The simulated data consists, among others, of a covariate with right censored values, as well as a dummy column indicating the presence or absence of censoring.
This function follows a strict mechanism which produces censored data. That mechanism may also be explicitly customized by the user.

```
# Install and Load package:
if(!require(devtools))
  install.packages("devtools")
devtools::install_github(("TiStat/Imputegamlss"))
library(Imputegamlss)

# Simulate data with right censored values:
right_data <- simulateData(n = 300,
                           param.formula = list(mu = ~ x1 + x2 + x3,
                           name = 'x1', subset = ~ (x1 > 0.68 & x2 < 0.79 & x3 > 0.5),
                           prob = 0.81, damage =c(0.55, 0.95), family = 'NO',
                           correlation = NULL)$defected
```

See documentation of simuateData for further details.

In this example, the observations where x_{i1} is potentially censored are restricted to a specific subset of the entire domain of x_1 (which is representative for real censoring occurences).
Furthermore, we state that a covariate has a certain probability to be defected ($prob = 0.81$) only if it belongs to the subset. In this case, the argument *damage* produces random factors drawn from $U \sim (0.55, 0.95)$ and multiplies them with those values of the subset which are actually defected. Note that with both *min* and *min* parameters of the uniform distribution being < 1, we ensure that the visible values are smaller than the true values, which is a necessary condition of right censoring.

```
# Current data set:
head(right_data)

##           y         x1        x2        x3 indicator
## 1 1.9152603 0.97412670 0.5423221 0.2517920         0
## 2 1.4882373 0.03091175 0.4031305 0.7516456         0
## 3 1.6194429 0.35938114 0.7370158 0.7007831         0
## 4 0.6939264 0.41226635 0.1741900 0.0198100         0
## 5 1.5259951 0.51279599 0.3386034 0.8140948         0
## 6 1.4399806 0.79379074 0.7458761 0.2914004         0
```

The defect interpretation on the generated true data set is outsourced in the function *simulatedefect*. It is available for the user to defect an existing data set at his own discretion.

```
# Execute the MICE Algorithm:
right_impute <- imputex(xmu_formula = x1 ~ pb(y) + pb(x2) + pb(x3),
                        data = right_data, indicator = "indicator",
                        censtype = "right")
```

See documentation of imputex for further details.

After the execution, the user now has access to several features and results supplied by the algorithm such as the average quantiles of the proposal values *right_impute$imputequantiles*, the new data set with all imputed values

right_impute$fulldata, the impute-variance *right_impute$imputevariance* and much more.
The average quantiles of the m proposal values for each observation are constructed such that each of the respective D_j are evaluated for their quantiles and averaged over all j for each quantile level. The quantiles refer to the valid region of which the proposals are drawn from.
The impute-variance is a vector containing the variances of the m proposals for each observation.

Methods

Such objects have their own S3 class, i.e. *"imputed"*, and they come with basic methods such as *print.imputed*, *summary.imputed* and *plot.imputed*.

```
class(right_impute)

## [1] "imputed"
```

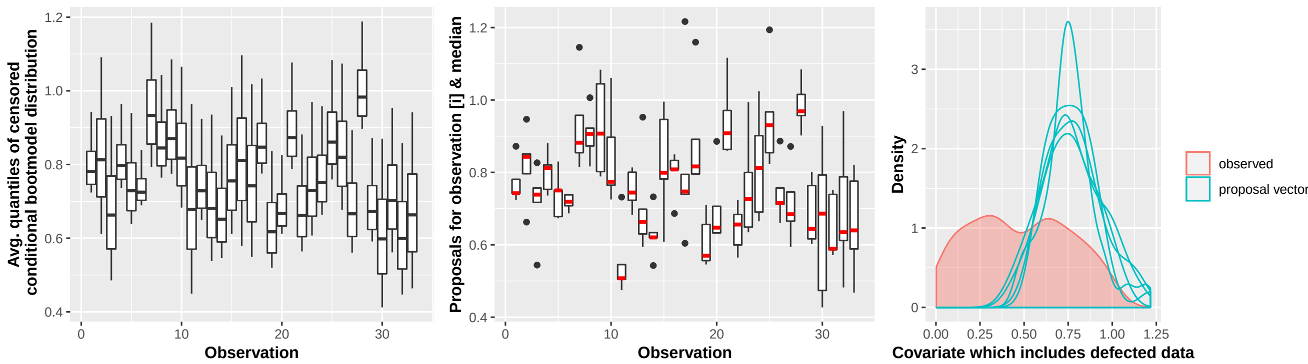
```
# Print:
right_impute

## Call:
##  imputex(xmu_formula = x1 ~ pb(y) + pb(x2) + pb(x3), data = right_data,
##     indicator = "indicator", censtype = "right")
##
##  Number of observations: 300
##  Replaced 33 right censored values
```

```
summary(right_impute)

## Call:
##  imputex(xmu_formula = x1 ~ pb(y) + pb(x2) + pb(x3), data = right_data,
##     indicator = "indicator", censtype = "right")
##
##  11% of the observations are defected
##
##  Number of observations: 300
##  Type of censoring: right
##  Number of proposals for each defected observation: 5
##  Number of replacements: 33
##
##  Imputed values:
##  Min.     1st Qu.     Median     Mean     3rd Qu.     Max.
##  0.507      0.656      0.742      0.745      0.812      0.969
##
##  Distances of imputations to censorings:
##  Min.     1st Qu.     Median     Mean     3rd Qu.     Max.
##  0.023      0.08      0.121      0.134      0.181      0.305
##
##  Imputation variances of proposals:
##  Min.     1st Qu.     Median     Mean     3rd Qu.     Max.
##  0      0.005      0.011      0.014      0.02      0.054
```

```
plot(right_impute, boxes = TRUE) # boxes = FALSE is the default
```

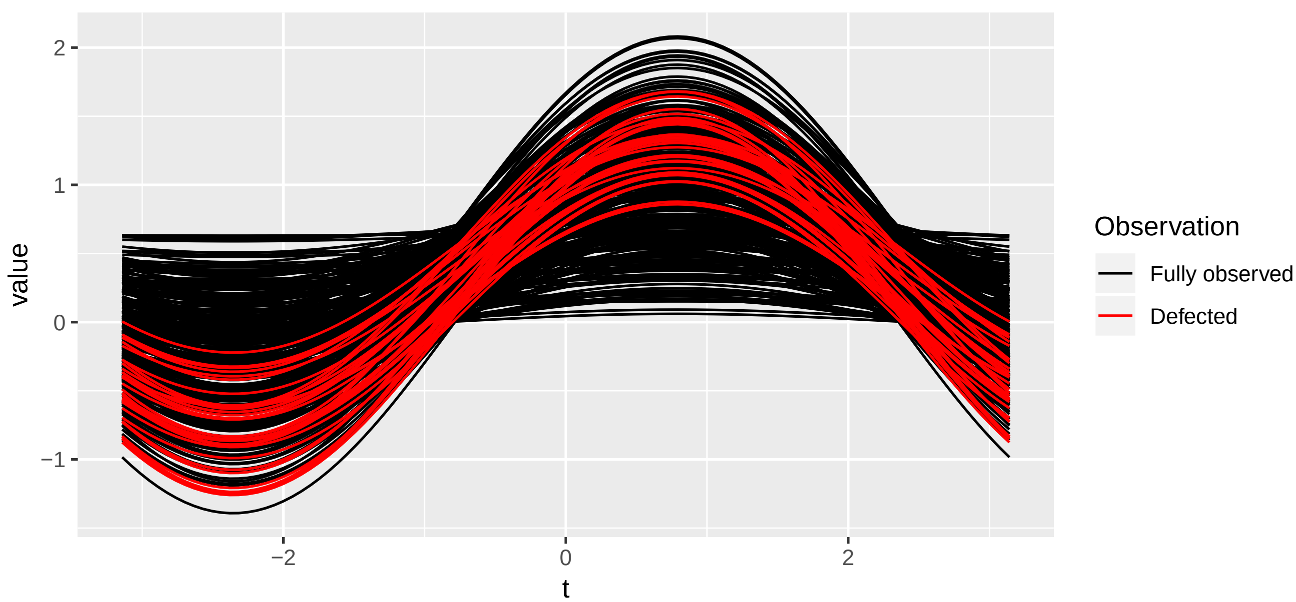


See documentation of plot.imputed for further details.

In the first figure, note that the quantiles are more heavily skewed, if $x_{i1,cens}$ was in the tails of the estimated distributions D_j, $j = 1, ..., m$.
The second plot displays for each observation $x_{i1,cens}$ the realisations of the m draws from each respective D_j. The median value (red) is the chosen replacement value.
The third figure depicts the fingerprint of each bootmodel's n_{mis} realisations (blue). Each of these realisations is drawn from the respective $D_j(\dot{\mu}_i, \dot{\sigma}_i \mid B_j;\ w_{i,cens}\ ; x_{i1,cens})$.
First, note that each of those density estimates is a realisation of the mixture distribution of the m D_j's.
Furthermore, as the defect was most likely for large x_1 (subset), we see that the proposals for the censored values are relatively shifted to the right in comparison to the density estimate of the fully observed data (red).
It is desireable that the fingerprint estimates are consimilar.

We also implemented a generic function *andrew* which produces Andrew's curves in an unsupervised framework, i.e. excluding the dependent variable of the linear problem. Also, the defected covariate is excluded from the Fourier series. This gives us an insight on possible peculiar characteristics that the defected observations might share in contrast to the remaining complete observations and therefore are more likely to become defected in the first place. Below we can see that the defected observations have a similar structure in their remaining covariates. Due to the subset condition, the probability for an observation x_{i1} to be a subject of defect depends on the level of the other covariates. This is reflected in the Andrew's curves, where these observations are viewed as lines restricted to a certain area, which eventually form a group.
Mind that this function only calls the method *andrew.imputed*.

```
andrew(object = right_impute, dependent = "y", ordering = NULL)
```



See documentation of andrew.imputed for further details.

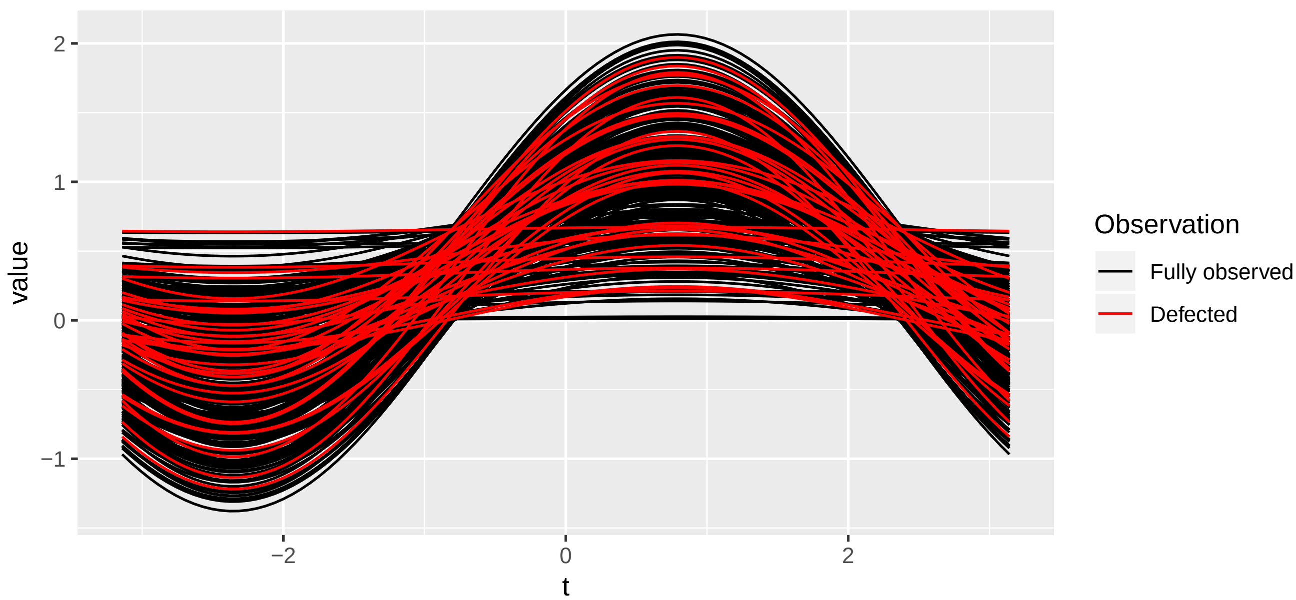
For comparison's sake, let's simulate a data set where the probability of a covariate to be defected depends only on the level of x_{i1}. Hence, the probability that a value is defected should be perceptibly smaller as the subset grows

larger. All the other parameters are held constant.
Notice that the red lines in the following figure are expected not to be limited to a certain region:

```
# Simulate data set:
c_data <- simulateData(n = 300,
                       param.formula = list(mu = ~ x1 + x2 + x3, sigma = ~ 0.35*sqrt(x2)),
                       name = 'x1', subset = ~ x1 > 0.68,
                       prob = 0.3, damage =c(0.55, 0.95), family = 'NO',
                       correlation = NULL)$defected

# Execute the MICE Algorithm:
c_impute <- imputex(xmu_formula = x1 ~ pb(y) + pb(x2) + pb(x3), data = c_data,
                    indicator = "indicator", censtype = "right")
```

```
# Plot Andrew's curves:
andrew(c_impute, "y")
```



Validation

After the imputation process, we want to be sure that the replacement values actually had a positive impact on the data set. To demonstrate this, let's refer to our basic assumption, i.e. our dependent variable is linearly related to the predictors. We therefore will fit two linear models for comparison, one with the initially censored values *right_data* and one with the imputations. Keep in mind that the indicator variable must be excluded to prevent multicollinearity!
Also consider that the $x_{1,cens}$ values are not predicted by a single model fit, but by multiple proposals of a posterior predictive distribution, which is surrogated by multiple bootmodel procedures.

```
old_model <- lm(y ~ . - indicator, data = right_data)
summary(old_model)

##
## Call:
## lm(formula = y ~ . - indicator, data = right_data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.80116 -0.13517 -0.00339  0.15539  0.96424
##
## Coefficients:
```

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.004875   0.047835   0.102    0.919
## x1          0.981253   0.055340  17.731   <2e-16 ***
## x2          1.007586   0.050783  19.841   <2e-16 ***
## x3          1.081629   0.053564  20.193   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2572 on 296 degrees of freedom
## Multiple R-squared:  0.786,Adjusted R-squared:  0.7839
## F-statistic: 362.5 on 3 and 296 DF,  p-value: < 2.2e-16
```

```r
AIC(old_model)
```

```
## [1] 42.63538
```

```r
new_model <- lm(y ~ . - indicator, data = right_impute$fulldata)
summary(new_model)
```

```
##
## Call:
## lm(formula = y ~ . - indicator, data = right_impute$fulldata)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.80204 -0.12280 -0.00301  0.13131  0.98410
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.00583    0.04599   0.127    0.899
## x1           0.97164    0.05181  18.755   <2e-16 ***
## x2           1.03417    0.04925  20.999   <2e-16 ***
## x3           1.03252    0.05185  19.913   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2497 on 296 degrees of freedom
## Multiple R-squared:  0.7984,Adjusted R-squared:  0.7963
## F-statistic: 390.7 on 3 and 296 DF,  p-value: < 2.2e-16
```

```r
AIC(new_model)
```

```
## [1] 24.8158
```

Looking at Akaike's Information Criterion as well as the adjusted R-squares for the two models, we can observe an improvement in the new one.
Note that the improvement is gained by the additional information provided. The classic measures do not account for the uncertainty, which is associated with the imputation process. Therefore, any gained improvement is not necessarily quantifiably ascertainable.

Computational performance

Performance of *imputex*

```r
if(!require(profvis))
  install.packages("profvis")
library(profvis)

# Check performance of imputex:
profvis(imputex(xmu_formula = x1 ~ pb(y) + pb(x2) + pb(x3),
                data = right_data, indicator = "indicator",
                censtype = "right"))
```

Code	File	Memory (MB)		Time (ms)
▼ imputex		-817.8	858.4	2700
▶ gamlss	imputex.R	-720.3	728.6	1960
▼ samplecensored	imputex.R	-97.5	123.9	690
▼ apply	samplecensored.R	-48.4	88.7	480
▼ FUN		-48.4	88.7	480
▼ ffamily	samplecensored.R	-48.4	88.7	480
▼ family_fun	samplecensored.R	-48.4	88.7	480
▼ predictAll	samplecensored.R	-48.4	88.5	470
▼ predict		-48.4	88.5	470
▶ predict.gamlss		-48.4	88.5	470
▶ sapply	samplecensored.R	0	0.2	10
▶ ffamily	samplecensored.R	-49.1	35.2	210
▶ family_fun	imputex.R	0	5.8	30
▶ [	imputex.R	0	0.0	20
▶ profvis		0	0	10

Figure 2: Speed of *imputex*

Taking a quick glance at the efficiency of the algorithm, we can observe that most of the time is consumed by the gamlss function calls and their methods, which are inevidable by construction of MICE.

The time consumed is directly dependent on the number of proposals m, which result in $m + 1$ gamlss calls (i.e. including the model on the observed data in Step 2). All of them must be evaluated seperately, as they refer to a different information set and result in different parameter estimates.
On the second place is *samplecensored*, a heavily used function, which evaluates the distributional parameters for each observation and accesses the families' cdf and random generator function for each fitted effect from the respective bootmodels.
Thereby it is somewhat dependent on n_mis. However, this is a minor issue, as *samplecensored* performs all of its operations vectorized while it lives for the current bootmodel. Note further, that the employed inverse sampling is also an efficient way to draw from the valid region only, as it avoids an unknown number of random draws from the respective distributions, from which only those are kept that are valid. Aside from assignment and copy operations, which are avoided, it also provides an additional feature:
While the bootmodel is alive, and *samplecensored* predicted the conditional distribution of the censored covariates from which proposals are drawn, it accesses them with ease to extract the respectively rescaled quantiles for each observation (also vectorized). The results are catched in an array for later processing.

Note that the actual parameter prediction step is outsourced in *family_fun*'s *predictAll*, which has to be called several times to evaluate the distributional functions for inverse sampling (once for missing, twice for left/right and thrice for interval censored). There exists a minor 'bottleneck' which grows large in m: *family_fun* is memoryless and predicts the same distributional parmeters at most thrice for one bootmodel. We noticed this to late before the package release to change it.

Performance of *andrew*

```r
andreas <- simulateData(n = 1000,
                        param.formula = list(mu = ~ x1 + x2 + x3 + x4 +x5,
                                             sigma = ~ 0.35*sqrt(x2)),
                        name = 'x1', subset = ~ x1 > 0.68,
                        prob = 0.3, damage =c(0.55, 0.95), family = 'NO',
                        correlation = NULL)$defected

# Execute the MICE Algorithm:
andreas_impute <- imputex(xmu_formula = x1 ~ pb(y) + pb(x2) + pb(x3) + pb(x4) + pb(x5),
                          data = andreas,
                          indicator = "indicator", censtype = "right")

# Check performance of andrew:
profvis(andrew(andreas_impute, dependent = "y"))
```

Code	File	Memory (MB)		Time (ms)
▼ andrew		0	6.1	20
▼ andrew.imputed	andrew.R	0	6.1	20
▼ andrewcore	andrew.R	0	6.1	20
▶ reshape2::melt	andrew.R	0	2.5	10
▶ geom_line	andrew.R	0	3.6	10
▶ profvis		0	0	10

Figure 3: Speed of *andrew*

The shipped *ggplot* Andrew's curves (about 40ms) are optimized for speed and efficiency and even outperform the plot method from the *andrews* package (about 260ms), when run on a parameter frame of size 1000x5.

The key to speed is the applied language processing instead of jointly evaluating the Fourier series with each observations' set of parameters and for all expansion points t over the range $[-\pi, \pi]$. The keynote is that, given a data frame, the formulas of the Fourier summands, stripped of their parameters are unevaluatedly expanded only once before being evaluated once at their expansion points t. The matrix product of the parameter frame with the expansion matrix evaluates in an instance all Andrew's curves. This saves millions of iterations. The 'bottleneck' is melting the data to longformat and passing it to *ggplot*.

References

de Jong, R., van Buuren, S., Spiess, M. (2016). Multiple imputation of predictor variables using generalized additive models. Communications in Statistics - Simulation and Computation, 45(3), 968-985. Retrieved from https://doi.org/10.1080/03610918.2014.911894 doi: 10.1080/03610918.2014.911894

Efficiently sampling a thresholded Beta distribution
https://stats.stackexchange.com/questions/274512/efficiently-sampling-a-thresholded-beta-distribution/274516274516