Advanced and Very Advanced Modeling Techniques in CLVTools

Markus Meierer Patrik Schilter Jeffrey Näf Patrick Bachmann

September 21, 2025

Abstract

This document provides an overview of advanced modeling techniques for the probabilistic models implemented in the R package `CLVTools`. `CLVTools` offers various advanced modeling options that go beyond basic customer base analysis with or without covariates. These include the ability (a) to add regularization for covariate parameters, (b) to account for the correlation between the transaction and dropout process, (c) set equality constraints on covariate parameters, (d) control for endogenous covariates, (e) add a Hessian matrix to an already fitted model, and (f) run fast bootstrapping by sampling model parameters directly.

Contents

1 Data setup

First, we create a data object that will be used throughout this vignette.

```
R> library(CLVTools)
R> library(data.table)
R> data("apparelTrans")
R> data("apparelStaticCov")

# Create transaction data object with static covariates
R> clv.apparel <- clvdata(
+     data.transactions = apparelTrans,
+     date.format = "ymd",
+     time.unit = "week",
+     estimation.split = 104,
+     name.id = "Id",
+     name.date = "Date",
+     name.price = "Price"
+ )
# Store all available covariates for both processes
R> clv.apparel.static <- SetStaticCovariates(
+   clv.data = clv.apparel,
+   data.cov.life = apparelStaticCov,
+   data.cov.trans = apparelStaticCov,
+   names.cov.life = c("Gender", "Channel"),
+   names.cov.trans = c("Gender", "Channel"),
+   name.id = "Id"
+ )
```

2 Regularization of covariate parameters

When a large number of covariates are included in the analysis, regularization can help prevent overfitting. To this end, it is possible to apply a normal prior on the covariate parameters (L2 regularization). This requires specifying a regularization weight λ^{reg} per process. The value of λ^{reg} is the same for all covariate parameters of a process. The larger λ^{reg}, the stronger the effect of the regularization while a value of 0 results in no regularization. To find the optimal λ^{reg}, any hyperparameter optimization procedure can be applied.

To regularize covariate parameters, the regularization weights for both processes must be defined in the parameter `reg.lambdas`. For example, `reg.lambdas = c(trans = 0.1, life = 0.2)` sets λ^{reg} to 0.1 for the transaction process and 0.2 for the lifetime processes. The use of regularization and weights is indicated at the end of the output of `summary()`.

```
# Fit model while applying regularization to the covariate parameters
R> est.pnbd.regularization <- latentAttrition(
+   formula = ~ Gender + Channel | Gender + Channel,
+   family = pnbd,
+   data = clv.apparel.static,
+   verbose = FALSE,
+   reg.lambdas = c(trans = 0.1, life = 0.2)
+ )
R> summary(est.pnbd.regularization)

Pareto/NBD with Static Covariates  Model

Call:
latentAttrition(formula = ~Gender + Channel | Gender + Channel,
```

```
    family = pnbd, data = clv.static, reg.lambdas = c(trans = 0.1,
        life = 0.2))

Fitting period:
Estimation start  2005-01-02
Estimation end    2006-12-31
Estimation length 104.0000 Weeks

Coefficients:
               Estimate Std. Error  z-val Pr(>|z|)
r               1.73887    8.07414     NA       NA
alpha          69.85288  315.45779     NA       NA
s               0.53350    5.81354     NA       NA
beta           39.68346  704.57431     NA       NA
life.Gender    -0.04437    1.54979 -0.029    0.977
life.Channel    0.02465    1.54501  0.016    0.987
trans.Gender    0.17178    1.63462  0.105    0.916
trans.Channel   0.23676    1.65635  0.143    0.886

Optimization info:
LL     -9.7313
AIC    35.4626
BIC    70.6380
KKT 1  TRUE
KKT 2  TRUE
fevals 33.0000
Method L-BFGS-B

Used Options:
Correlation     FALSE
Regularization  TRUE
   lambda.life  0.2000
   lambda.trans 0.1000
Constraint covs FALSE
```

3 Adding a correlation between the transaction and attrition process

To relax the assumption of independence between the transaction and the attrition process, specify the argument `use.cor` in the `latentAttrition()` command. This is independent of whether the model includes covariates or not. With regard to the latter, this is an extension of the model presented in Bachmann et al. (2021). In the case of `use.cor=TRUE`, a Sarmanov approach is used to correlate the attrition and transaction process. The argument `start.param.cor` allows us to optionally specify a starting value for the correlation parameter.

The model output will then list an additional parameter `Cor(life,trans)`, which may be directly interpreted as a correlation:

- If the correlation is zero, it indicates that there is no relationship between customers' transaction and attrition rate.
- If the correlation is positive and significant, customers with a higher (lower) transaction rate are more (less) likely to churn. The underlying mechanism is as follows: a higher transaction rate λ is associated with a higher attrition rate μ, i.e., a reduction in the customer's lifetime.
- If the correlation is negative and significant, customers with a higher (lower) transaction rate are less (more) likely to churn.

The impact of adding a correlation parameter depends on the dataset. In many applications that focus on prediction rather than on an in-depth understanding of customers' purchase behavior, the modeling of the additional parameter is neglected. A key reason for this is the increase in computational complexity compared to the often only marginal change in predictive accuracy. While this is a common decision among practitioners, it depends on the data at hand and the modeling objective.

4 Including equality constraints for covariate parameters

If more complex hypothesis testing is required, users can leverage parameter constraints to compare effect sizes between the attrition and transaction process. All latent attrition models that can account for time-invariant and time-varying covariates support equality constraints for the respective covariate parameters. For example, it is possible to test whether the parameter value of the covariate `Gender` is the same for both processes. This potentially facilitates testing of novel hypotheses and thus, helps to increase the understanding how a particular covariate impacts each process.

Here, we add such a constraint for the parameter estimates of the covariate `Gender`. In the following, we present both the unconstrained and the constrained model. First, the unconstrained model:

```
R> est.pnbd.full <- latentAttrition(
+   formula = ~ . | .,
+   family = pnbd,
+   data = clv.apparel.static,
+   verbose = FALSE)
R> summary(est.pnbd.full)
```

```
Pareto/NBD with Static Covariates  Model

Call:
latentAttrition(formula = ~. | ., family = pnbd, data = clv.apparel.static,
    verbose = FALSE)

Fitting period:
Estimation start  2005-01-02
Estimation end    2006-12-31
Estimation length 104.0000 Weeks

Coefficients:
              Estimate Std. Error  z-val Pr(>|z|)
r               1.8378     0.3455  5.320 1.04e-07 ***
alpha          92.9123    16.9670  5.476 4.35e-08 ***
s               0.5920     0.2609  2.269  0.02327 *
beta           49.6227    36.2509  1.369  0.17104
life.Gender    -0.6430     0.2955 -2.176  0.02957 *
life.Channel    0.7907     0.3059  2.585  0.00973 **
trans.Gender    0.2859     0.1041  2.745  0.00605 **
trans.Channel   0.6241     0.1050  5.946 2.74e-09 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Optimization info:
LL     -5821.0627
AIC    11658.1254
BIC    11693.3009
KKT 1  TRUE
KKT 2  TRUE
fevals 41.0000
Method L-BFGS-B
```

```
Used Options:
Correlation     FALSE
Regularization  FALSE
Constraint covs FALSE
```

Second, the constrained model:

```
R> est.pnbd.constr <- latentAttrition(
+   formula = ~ . | .,
+   names.cov.constr = "Gender",
+   family = pnbd,
+   data = clv.apparel.static,
+   verbose = FALSE)
R> summary(est.pnbd.constr)

Pareto/NBD with Static Covariates  Model

Call:
latentAttrition(formula = ~. | ., family = pnbd, data = clv.apparel.static,
    verbose = FALSE, names.cov.constr = "Gender")

Fitting period:
Estimation start  2005-01-02
Estimation end    2006-12-31
Estimation length 104.0000 Weeks

Coefficients:
              Estimate Std. Error z-val Pr(>|z|)
r               1.7939     0.3318 5.406 6.43e-08 ***
alpha          94.7223    17.2216 5.500 3.79e-08 ***
s               0.4287     0.1418 3.025  0.00249 **
beta           59.0743    34.5098 1.712  0.08693 .
life.Channel    1.0228     0.3542 2.888  0.00388 **
trans.Channel   0.6384     0.1064 5.998 2.00e-09 ***
constr.Gender   0.3283     0.1074 3.056  0.00224 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Optimization info:
LL     -5826.5342
AIC    11667.0684
BIC    11697.8469
KKT 1  TRUE
KKT 2  TRUE
fevals 48.0000
Method L-BFGS-B

Used Options:
Correlation          FALSE
Regularization       FALSE
Constraint covs      TRUE
   Constraint params Gender
```

In this case, we estimate an additional model that forces the covariate `Gender` to have the same parameter value for both transaction and attrition processes. We specify this constraint using the `names.cov.constr` argument. As a result, the model output displays only a single parameter value for this variable. The `summary()` output indicates the use of these parameter constraints at the end.

A likelihood ratio test helps to evaluate whether adding an equality constraint changes the model fit in a significant way.

```
lrtest(
  est.pnbd.constr,
  est.pnbd.full,
  name = c("Constrained Model", "Unconstrained Model")
)
```

```
Likelihood ratio test

Model 1: Constrained Model
Model 2: Unconstrained Model
  #Df  LogLik Df  Chisq Pr(>Chisq)
1   7 -5826.5
2   8 -5821.1  1 10.943  0.0009396 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

The test compares likelihood values between unconstrained and constrained models to determine if a covariate's effect size differs significantly between the attrition and transaction process. For the `Gender` covariate in our case study, the results reveal a significant difference between these models. This indicates that the effect size of `Gender` is statistically different between the attrition and transaction process. In other words, adding an equality constraint for the `Gender` parameter significantly worsened the model fit.

Further use cases are possible. For example, when exogenous information on marketing interventions is available, this analysis is particularly helpful to disentangle how marketing intervention impacts customers' purchase behavior.

5 Controlling for endogenous covariates

An additional use case for advanced modeling techniques is to control for endogenous covariates. The covariate parameter estimates for the covariates can give some insight into what drives customers' purchase behavior. If the exogeneity assumption of the covariates is violated, various techniques can be used to control for this.

Consequently, the models that support covariates in CLVTools can be used together with other packages that implement related two-step modeling techniques. The first option is to use instrumental variables, which can be implemented with R Base. If these are not available, internal instrumental variable approaches can serve as an alternative (Gui et al., 2023). As many of these are designed as two-step approaches, their application to latent attrition models is straightforward. In this case study, all covariates are assumed to be exogenous. For an exemplary case study detailing various approaches to controlling the endogeneity of marketing campaigns, see Bachmann et al. (2021).

6 Adding a Hessian matrix to an already fitted model

For various reasons, one might want to fit a model without estimating the Hessian matrix at the end of the optimization procedure. If the Hessian, however, is not derived, there is no variance-covariance matrix available, and therefore, also no standard errors. A numerical approximation of the Hessian matrix can still be calculated at the final parameters using the method `hessian()`. To this end, we are first going to fit a model without deriving the Hessian at the end of the parameter optimization. This requires to also disable the KKT criteria.

```
# Fit a model without calculating the Hessian
```

```
R> est.pnbd.noH <- latentAttrition(
+   family = pnbd,
+   data = clv.apparel,
+   verbose = FALSE,
+   optimx.args = list(hessian=FALSE, control=list(kkt=FALSE))
 )

# Hessian set to all NA
print(est.pnbd.noH@optimx.hessian)
```

```
Warning: Hessian could not be derived. Setting all entries to NA.

          log.r log.alpha log.s log.beta
log.r        NA        NA    NA       NA
log.alpha    NA        NA    NA       NA
log.s        NA        NA    NA       NA
log.beta     NA        NA    NA       NA
```

There is a warning that the Hessian is not calculated, and all its values are set to `NA`. As shown in the following, no variance-covariance matrix can be obtained by inverting the Hessian, and consequently, there are also no standard errors for any parameter.

```
# Variance-covariance matrix fails
R> print(vcov(est.pnbd.noH))

# No standard errors
R> print(coef(summary(est.pnbd.noH)))
```

```
Error: The vcov matrix cannot be calulated because the hessian contains non-finite values!

        Estimate Std. Error z-val Pr(>|z|)
r      1.4489768         NA    NA       NA
alpha 48.6360845         NA    NA       NA
s      0.5612598         NA    NA       NA
beta  46.8843633         NA    NA       NA
Warning message:
For some parameters the standard error could not be calculated.
```

We can manually derive the Hessian matrix at the final parameters using `hessian()`. The model parameters were estimated at "log-scale" to ensure that they all remain greater than 0. This explains why the parameters are prefixed with "log." in the column and row names. For the variance-covariance matrix, the Hessian is not only inverted but also appropriately transformed to ensure that the variances and standard errors are correctly calculated at the parameters' "original scale".

```
# Given the final parameters, derive the Hessian
R> H <- hessian(est.pnbd.noH)
R> print(H)
```

```
               log.r  log.alpha      log.s  log.beta
log.r      433.22658 -388.22026 -117.48760  86.38645
log.alpha -388.22026  402.65104   78.92603 -57.69810
log.s     -117.48760   78.92603  117.95191 -78.20295
log.beta    86.38645  -57.69810  -78.20295  53.99615
```

By adding the Hessian to the fitted model, both the variance-covariance matrix and the standard errors become available. Note that the output of `vcov` is at the original parameter scale. Recall that p-values only make sense for covariate parameters. Although standard errors are shown, the p-values therefore remain `NA` for all coefficients here.

```
# Add the Hessian permanently to the estimated model
R> est.pnbd.noH@optimx.hessian <- H

R> print(vcov(est.pnbd.noH))
R> print(coef(summary(est.pnbd.noH)))
```

```
                r       alpha           s        beta
r      0.05925727   1.7049763 -0.01786467   -3.472616
alpha  1.70497626  56.0878415 -0.43375972  -82.963756
s     -0.01786467  -0.4337597  0.07346698    9.366245
beta  -3.47261643 -82.9637556  9.36624519 1268.172624

        Estimate Std. Error z-val Pr(>|z|)
r      1.4489768  0.2434282    NA       NA
alpha 48.6360845  7.4891816    NA       NA
s      0.5612598  0.2710479    NA       NA
beta  46.8843633 35.6114114    NA       NA
```

7 Running fast bootstrapping by sampling model parameters directly

Using the internal methods of `CLVTools`, it is possible to run a fast bootstrapping procedure that samples the model parameters directly. The implementation outlined below is a preview of a future feature that will be included in `CLVTools` once additional research has provided more insights on the advantages and limitations compared to the regular bootstrapping procedure in `CLVTools`.

Let $\hat{\theta}$ be the (log) parameters obtained through maximum likelihood. According to standard likelihood theory, $\hat{\theta}$ approximately follows a Gaussian distribution $N(\theta, H^{-1}(\hat{\theta}))$, where $H^{-1}(\hat{\theta})$ is the inverse of the Hessian matrix. A similar result holds under bootstrapping, where the estimate of θ obtained after bootstrapping, $\dot{\hat{\theta}}$, follows an approximate $N(\hat{\theta}, H^{-1}(\hat{\theta}))$ distribution, given the original data; see, e.g., Cheng and Huang (2010). As such, instead of bootstrapping the data and calculating $\dot{\hat{\theta}}_n$, we could simply draw $\dot{\hat{\theta}}$ from $N(\hat{\theta}, H^{-1}(\hat{\theta}))$ to obtain approximate bootstrap samples. We detail this approach here.

```
# For reproducibility
R> set.seed(42)

# Fit a PNBD model where "Gender" is constrained to be equal for both processes
R> p.apparel.constr <- pnbd(
+    clv.data.apparel.cov,
+    names.cov.constr = "Gender",
+    verbose = FALSE
+    )
R> gg.apparel <- gg(clv.data.apparel, verbose=FALSE)
```

We begin by defining the log-likelihood for the joint model that combines the individually estimated latent attrition and spending models for the purpose of this bootstrap procedure. To arrive at the likelihood of the joint model, the individual model likelihoods are multiplied. Because we are operating with the log-likelihoods, however, the individual model log-likelihoods have to be summed.

To implement this, we use the internal method `clv.get.LL()` in the `CLVTools` package which returns a method to calculate the LL with the exact same specification used to fit the model originally. The returned function also contains all the required inputs besides the parameters. Note that the parameters are not all at original scale. The model parameters (r, α, s, β) are at "log-scale" and transformed back in the likelihood. Extracting them from the original optimizer output is the most straightforward to ensure they are at the correct scale and correctly named.

```
# Functions to call log-likelihoods with their original specification
R> LL.pnbd <- CLVTools:::clv.fitted.get.LL(p.apparel.constr)
R> LL.gg <- CLVTools:::clv.fitted.get.LL(gg.apparel)

# Extract parameters required for log-likelihoods from optimx output
R> final.coefs.pnbd <- drop(tail(coef(p.apparel.constr@optimx.estimation.output), n=1))
R> final.coefs.gg <- drop(tail(coef(gg.apparel@optimx.estimation.output)))

# Define parameter names:
# Used in ‘fn.joint.LL‘ to forward parameters to relevant model log-likelihoods
# (and some other places)
R> names.params.pnbd <- names(final.coefs.pnbd)
R> names.params.gg <- names(final.coefs.gg)

# Log-Likelihood of joint model
# Accepts a vector that contains parameters for both sub-models
R> fn.joint.LL <- function(params){
+    return(
+      # Call the per-model LL only with the parameters of the respective
+      # models, using names to extract the relevant ones
+      LL.pnbd(params[names.params.pnbd]) + LL.gg(params[names.params.gg])
+    )
+  }
```

Given the log-likelihood of the joint model, the Hessian matrix is numerically approximated at the final coefficients. By inverting the Hessian, the variance-covariance matrix is obtained.

```
# Parameters for the joint-LL
R> final.params.joint <- c(final.coefs.pnbd, final.coefs.gg)

# Approximate hessian of joint model at final parameters
R> H.joint <- numDeriv::hessian(
+    func = fn.joint.LL,
+    x = final.params.joint
+  )
R> rownames(H.joint) <- colnames(H.joint) <- names(final.params.joint)

# Invert the Hessian
R> vcov.joint <- solve(H.joint)
```

Given the variance-covariance matrix $H^{-1}(\hat{\theta})$, we can then sample parameters $\dot{\hat{\theta}}$ from the Gaussian distribution $N(\hat{\theta}, H^{-1}(\hat{\theta}))$. It is to note that these sampled parameters are at the scale that was required for calling the log-likelihoods. In detail, mostly at the log-scale, as can be easily recognized by their names.

```
# Sample parameters
R> params.sampled.joint <- mvrnorm(n = 100, mu = final.params.joint, Sigma = vcov.joint)
R> head(round(params.sampled.joint, 3))
```

```
                log.r  log.alpha   log.s  log.beta  life.Channel  trans.Channel
[1,]            0.383      4.568  -0.795     4.993         1.714          0.861
[2,]            0.788      4.661  -0.891     3.723         0.913          0.710
[3,]            0.583      4.516  -0.549     4.225         0.644          0.596
[4,]            0.698      4.619  -0.350     4.372         0.348          0.579
[5,]            0.487      4.517  -0.830     4.317         1.563          0.621
[6,]            0.575      4.522  -0.937     4.008         1.468          0.657
```

```
     constr.Gender  log.p  log.q  log.gamma
[1,]         0.470  0.952  1.899      4.439
[2,]         0.224  0.923  1.842      4.373
[3,]         0.357  1.238  1.529      3.699
[4,]         0.315  0.756  1.920      4.634
[5,]         0.345  1.309  1.671      3.838
[6,]         0.286  0.981  1.664      4.034
```

We can then use the sampled parameters to make predictions and diagnostic plots. For this, we use a copy of the fitted model object and replace the existing parameters with these new parameters. This gives us access to all the functionalities that we require, but let us use the new parameters. To do so, we replace the original estimated parameters in the optimizer outputs with the sampled ones. We then use an internal method to set the model's "prediction parameters": The parameters in original scale that are used for all downstream calculations after the optimization, such as making predictions or diagnostic plots. Setting these parameters manually is challenging, since they are estimated at different scales and can only equal 1 for both processes when using equality constraints.

```
# List from applying the defined function to every row of sampled parameters
R> l.preds <- lapply(seq(NROW(params.sampled.joint)), function(i){
+    # Get i-th parameters for respective model
+    i.params.pnbd <- params.sampled.joint[i, names.params.pnbd]
+    i.params.gg <- params.sampled.joint[i, names.params.gg]
+
+  # Set the sampled parameters on a copy of the fitted model
+    i.pnbd <- p.apparel.constr
+    i.gg <- gg.apparel
+
+    i.pnbd@optimx.estimation.output[1, names.params.pnbd] <- i.params.pnbd
+    i.pnbd <- CLVTools:::clv.controlflow.predict.set.prediction.params(i.pnbd)
+    # PNBD with dynamic covs would further require to also re-calculate ‘@LL.data‘
+    # i.pnbd@LL.data<-pnbd_dyncov_getLLdata(clv.fitted=i.pnbd, params=i.params.pnbd)
+
+    i.gg@optimx.estimation.output[1, names.params.gg] <- i.params.gg
+    i.gg <- CLVTools:::clv.controlflow.predict.set.prediction.params(i.gg)
+
+    # Now use the fitted models on which the parameters were changed to make
+    # predictions. The predictions with the adapted gg model are made inside
+    # predict() using the provided fitted model object ‘i.gg‘.
+    dt.pred <- predict(
+      i.pnbd,
+      predict.spending = i.gg,
+      prediction.end = 104,
+      continuous.discount.factor = log(1+0.1)/52,
+      verbose = FALSE
+    )
+
+    return(dt.pred)
+  })

# Bind to single table
R> dt.preds <- rbindlist(l.preds)
```

Recall that only the parameters were replaced, with no sampling of customers or transactions. The transaction and covariate data in the model remain unchanged. As a result, we maintain the same number of predictions for every customer. This consistency is not guaranteed with conventional bootstrapping.

We proceed and now calculate the confidence intervals to quantify the parameter uncertainty of the model:

```
# Calculate some CIs for CLV which is the combination of PNBD and GG
R> dt.preds.ci <- dt.preds[, list(
+    CLV.05 = quantile(predicted.CLV, probs=0.05),
+    CLV.median = quantile(predicted.CLV, probs=0.5),
+    CLV.mean = mean(predicted.CLV),
+    CLV.95 = quantile(predicted.CLV, probs=0.95)
+    ),
+    by = Id]

# Predictions with the original models
R> dt.preds.original <- predict(
+    p.apparel.constr,
+    predict.spending = gg.apparel,
+    prediction.end = 104,
+    continuous.discount.factor = log(1+0.1)/52,
+    verbose=FALSE)

# Combine and print
R> dt.preds.ci[dt.preds.original, CLV.original := i.predicted.CLV, on = "Id"]
R> dt.preds.ci[, c("Id","CLV.original","CLV.05","CLV.median","CLV.mean","CLV.95")]
```

```
         Id CLV.original    CLV.05 CLV.median   CLV.mean     CLV.95
     <char>        <num>     <num>      <num>      <num>      <num>
  1:      1   1128.92389 944.22892 1123.42276 1130.70849 1352.47235
  2:     10    293.65589 236.46337  294.66638  294.19581  356.24903
  3:    100     42.82015  29.30950   42.68499   42.97384   55.27039
  4:    101     78.70005  55.86206   75.35320   79.76272  127.63147
  5:    102     42.82015  29.30950   42.68499   42.97384   55.27039
 ---
596:     95    176.28140 141.26828  174.64106  175.97826  211.93977
597:     96     24.87468  16.87976   23.37485   24.75444   33.97500
598:     97     51.31572  35.26809   51.45389   51.32866   63.40403
599:     98    118.75273  87.64524  116.18627  119.52325  172.83709
600:     99    167.33232 131.79023  166.17191  166.61359  210.12010
```

We note that due to the nonlinear transformations of the parameters from the predictions of the CLV, the mean of the bootstrapped values generally no longer corresponds to the original CLV predictions. However, in most cases, the deviations are minimal.

References

Patrick Bachmann, Markus Meierer, and Jeffrey Näf. The Role of Time-Varying Contextual Factors in Latent Attrition Models for Customer Base Analysis. *Marketing Science*, 40(4):783–809, 2021.

Guang Cheng and Jianhua Z. Huang. Bootstrap consistency for general semiparametric M-estimation. *The Annals of Statistics*, 38(5):2884–2915, 2010.

Raluca Gui, Markus Meierer, Patrik Schilter, and René Algesheimer. Rendo: internal instrumental variables to address endogeneity. *Journal of Statistical Software*, 107:1–43, 2023.