nlsur

Schätzung nichtlinearer Gleichungssysteme in R

Jan Marvin Garbuszus

12. September 2017

1 Inhalt

- Einleitung und Hinführung
- Nichtlineare Optimierung
- Nonlinear Least Squares
- NLSUR
- Speicherprobleme
- Beispiele

Einleitung und Hinführung

2.1 Einzelgleichung

Eine Einzelgleichung ist bspw. folgende OLS-Gleichung

$$q = a + b * x. \tag{1}$$

Mit der Güterkategorie q und den Gesamtausgaben x.

2.2 Gleichungssysteme

Im Gleichungssystem werden m-Gleichungen parallel geschätzt

$$\begin{aligned} q_1 &= a_1 + b_1 x \\ q_2 &= a_2 + b_2 x \\ &\vdots \\ q_m &= a_m + b_m x. \end{aligned} \tag{2}$$

2.3 SUR

Hier kann Gleichung für Gleichung gelöst werden mittels OLS.

Oder mit *seemingly unrelated regression* (SUR) als Gleichungssystem (Zellner 1962).

R Pakete die SUR Schätzungen ermöglichen sind **systemfit** (Henningsen und Hamann 2007) und **gmm** (Chaussé 2010).

Das kann nützlich sein, aufgrund von Restriktionen auf einzelne Parameter. Beispielsweise können einzelne Terme über die Gleichungen verteilt sein:

$$\begin{aligned} q_1 &= a_1 + b_1(x - \sum a_j) \\ q_2 &= a_2 + b_2(x - \sum a_j) \\ &\vdots \\ q_m &= a_m + b_m(x - \sum a_j). \end{aligned} \tag{3}$$

2 Einleitung und Hinführung

Das Gleichungssystem (3) ist auch als Linear Expenditure System (LES; Klein und Rubin 1947) bekannt.

Als Restriktion für (3) gilt $\sum b_m = 1$. Damit ist die sogenannte Adding-Up Restriktion erfüllt.

Geschätzt werden $m - 1$ Gleichungen für k Koeffizienten.

2 Einleitung und Hinführung

SUR ermöglicht das Lösen von linearen Gleichungssystemen.

Seit Mitte der 1970er Jahre sind in der Nachfragetheorie nichtlineare Gleichungssysteme en vogue (bspw. Translog und Almost-Ideal Demand System).

Zur Schätzung solcher Gleichungssysteme bedarf es entweder eines ML-, GMM- oder eines NLS-Systemschätzers, eines NLSUR.

2.4 nls & nlm

R bietet unter dem Befehl **nls** Schätzungen mittels Nonlinear Least Squares an. Ein Anwendungsbeispiel sieht wie folgt aus:

```
lm("q1 ~ x", data = dat)

nls("q1 ~ a + b * x", data = dat,
    start = c(a = 0, b = 0))
```

Mit **nlm** kann die Funktion `f` minimiert werden:

```
f <- function(x, dat) {
  a <- x[1]; b <- x[2]
  x <- dat$x; y <- dat$q1

  e <- y - (a + b * x)

  e <- sum(e^2)
  e
}

nlm(f, c(0,0), dat = dat)
```

Für NLSUR gilt es nun folgende Funktion zu minimieren

$$SSR(\hat{\boldsymbol{\beta}}) = \sum_{i=1}^{N} \{\mathbf{y}_i - \mathbf{f}_i(\boldsymbol{\beta}, \mathbf{X})\}' \, \boldsymbol{\Sigma}^{-1} \, \{\mathbf{y}_i - \mathbf{f}_i(\boldsymbol{\beta}, \mathbf{X})\} \tag{4}$$

$$\hat{\boldsymbol{\Sigma}} = \frac{1}{N} \varepsilon' \varepsilon \tag{5}$$

Das geht auch mit **nlm** bzw. **optim** . . .

Nichtlineare Optimierung

3 Nichtlineare Optimierung

Es gibt in R eine ganze Reihe unterschiedlicher Optimierungsalgorithmen.

- **nlm**
- **optim**
- **optimx** (Nash und Varadhan 2011; Nash 2014)
- ...

3 Nichtlineare Optimierung

Beispielhaft hier der Output verschiedener in **optimx** zusammengefasster Algorithmen für die vorangestellte lineare Regression

```
# Output beschraenken auf Parameter und Konvergenzinformation
sel <- c("a","b","convcode")
optimx(par = c(a=1, b=1), fn = f, dat = dat,
       control=list(all.methods=TRUE))[,sel]
```

3 Nichtlineare Optimierung

```
##                       a         b convcode
## BFGS         167.294366 0.3770053        0
## CG             1.000797 0.4826279        1
## Nelder-Mead  167.571964 0.3768302        0
## L-BFGS-B     167.294625 0.3770051        0
## nlm          167.294625 0.3770051        0
## nlminb       167.294783 0.3770050        0
## spg          167.456521 0.3769022        1
## ucminf       167.305675 0.3769979        0
## Rcgmin               NA        NA     9999
## Rvmmin               NA        NA     9999
## newuoa        20.326427 0.4702204        0
## bobyqa       167.294625 0.3770051        0
## nmkb         167.294179 0.3770054        0
## hjkb           1.000000 0.1000000     9999
```

```
## [1] 167.2946252   0.3770051
```

3 Nichtlineare Optimierung

Nach einigen Tests (mit **optimx**) ergaben sich mit **ucminf** (Nielsen und Mortensen 2012) basierend auf einem Algorithmus von Nielsen (2000) die besten Ergebnisse.

- Alle paar Durchläufe.
- Mit viel Wartezeit dazwischen.
- Bei kleinen Datenmengen.

Fazit

1. Optimierungsverfahren ist sehr anfällig bzgl. lokaler Optima.

2. Einzelne Verfahren konvergieren in Abhängigkeit von den Startwerten.

3. Nicht jeder Durchlauf führt zu identischen Ergebnissen.

Nonlinear Least Squares

Nonlinear Least Squares mit Gauss-Newton

1. Ausgehend von Startparametern werden LHS, RHS und die Residuen sowie SSR ausgerechnet.

2. Die Jacobimatrix (Matrix der partiellen Ableitung nach den Parametern) wird bestimmt.

3. Koeffizienten über eine OLS der Residuen auf die Jacobimatrix bestimmt.

4. Damit werden erneut LHS, RHS, Residuen und SSR sowie die Jacobimatrix bestimmt.

5. Die Koeffizienten werden schrittweise variiert, dabei wird SSR minimiert und Schritt 4 wiederholt.

$$\theta_{t+1} = \theta_{t-1} + \alpha * \theta \tag{6}$$

6. Wenn SSR nicht weiter minimiert werden kann, werden die Schritte 3, 4, 5 wiederholt, bis Konvergenz besteht.

5 Konvergenz

Wenn sich die Parameter oder SSR nicht mehr wesentlich verändern (Gallant 1987; Bates und Watts 2008)

$$\left|ssr_{t-1} - ssr\right| \leq \epsilon \left|ssr_{t-1} + \tau\right|$$

bzw.

$$\left\|\theta - \theta_{t+1}\right\| \leq \epsilon \left(\left\|\theta\right\| + \tau\right) \tag{7}$$

Hinweis: **nls** selbst nutzt das relative Offset Kriterium nach Bates und Watts (1981)

NLSUR

6.1 Formeln

$$SSR(\hat{\boldsymbol{\beta}}) = \sum_{i=1}^{N} \{\mathbf{y}_i - \mathbf{f}_i(\boldsymbol{\beta}, \mathbf{X})\}' \, \boldsymbol{\Sigma}^{-1} \, \{\mathbf{y}_i - \mathbf{f}_i(\boldsymbol{\beta}, \mathbf{X})\} \tag{8}$$

$$\hat{\boldsymbol{\Sigma}} = \frac{1}{n} \varepsilon' \varepsilon \tag{9}$$

6 NLSUR

- NLSUR macht nicht viel anders als NLS. Schlüssel zum Erfolgt ist das Stacken der Residuen und der Jacobimatrix.
- Die einzelnen Gleichungen können Gleichung für Gleichung gelöst werden.
- Anschließend wird gestackt.
- Mit den gestackten Matrizen kann gelöst werden, bis Konvergenz erreicht ist.

6.2 FGNLS & IFGNLS

- Zusätzlich kann **nlsur** mit *Feasible Generalized NLS* gelöst werden.
- Dabei werden die Startwerte einer zweiten Schätzung aus den Koeffizienten der ersten gewählt.
- Außerdem wird mit WLS anstelle von OLS gelöst, wobei $\mathbf{W}$ aus dem $\hat{\mathbf{\Sigma}}$ der ersten Schätzung gewählt wird.
- Wird dieser Schritt iterativ wiederholt, wird mit *Iterativ Feasible Generalized Nonlinear Least Squares* gelöst, welches einer Maximum Likelihood Schätzung entspricht (Greene 2012, 345ff.).

Speicherbedarf

7 Speicherbedarf

Für ein Gleichungssystem ergibt sich für jede einzelne Gleichung jeweils eine Matrix:

- $\mathbf{r}$ eine $n \times 1$ Matrix: Residuen
- $\mathbf{J}$ eine $n \times k$ Matrix: Jacobimatrix

Diese werden m-fach gestacked: also $n * m \times 1$ und $n * m \times k$.

7 Speicherbedarf

Für NLSUR wird ein WLS geschätzt. Die Gewichtungsmatrix $\mathbf{W}$ ist eine $n * m \times n * m$ Matrix. Im ersten Schritt ist es die Einheitsmatrix $\mathbf{I}_{n*m}$.

Für FGNLS und IFGNLS wird $\mathbf{W}$ mit $\sigma = 1/n \sum r'_m r_m$ gewählt.

σ ist eine $m \times m$ Matrix, weshalb $\mathbf{W} = \sigma \otimes \mathbf{I}_n$ gewählt wird.

$$\theta = (\mathbf{J}'\mathbf{J})^{-1}\mathbf{W}^{-1}(\mathbf{J}'\mathbf{r}) \tag{10}$$

7 Speicherbedarf

```r
# Fallzahl; Gleichungen; Koeffizienten
n <- 1000; m <- 3; k <- 9

# resid
object.size(matrix(1, nrow = n*m, ncol = 1))
```

```
## 24200 bytes
```

```r
# Gradient
object.size(matrix(1, nrow = n*m, ncol = k))
```

```
## 216200 bytes
```

```r
# W
object.size(matrix(1, nrow = n*m, ncol = n*m))
```

```
## 72000200 bytes
```

7 Speicherbedarf

```
# Fallzahl; Gleichungen; Koeffizienten
n <- 50000; m <- 10; k <- 500

# resid
3.8 Mb

# Gradient
1907.3 Mb

# W
1862.6 GB
```

7 Speicherbedarf

Die Gewichtungsmatrix kann mit dem **Matrix**-Paket (Bates und Maechler 2015) und dessen Sparse Matrizen umgesetzt werden.

```
# Fallzahl; Gleichungen; Koeffizienten
n <- 50000; m <- 10; k <- 500
W <- Matrix::kronecker(X = Matrix(1,m,m),
                       Y = Matrix::Diagonal(n))
dim(W)
```

```
## [1] 500000 500000
```

```
object.size(W)
```

```
## 80001416 bytes
```

7 Speicherbedarf

```
# Wie sieht das aus?
W[1:3,1:3]
```

```
## 3 x 3 sparse Matrix of class "dgTMatrix"
##
## [1,] 1 . .
## [2,] . 1 .
## [3,] . . 1
```

Alternativ kann die Matrizenmultiplikation Blockweise erfolgen. Dazu ist ein Verfahren analog zu Statas Vorgehen über **RcppArmadillo** (Eddelbuettel und Sanderson 2014) in **nlsur** implementiert.

Beispiele

8.1 NLSUR installieren

Aktuell ist NLSUR noch nicht auf CRAN, kann aber von github installiert werden. Dazu müssen die Build Tools installiert sein.

```
# install.packages("devtools")
devtools::install_github("JanMarvin/nlsur")

library(nlsur)
```

8.2 Translog

Das Translog aus Berndt und Wood (1975) und Greene (2012, 353f.)

$$\begin{aligned}
s_k &= \beta_k + \delta_{kk} \ln\left(\frac{p_k}{p_m}\right) + \delta_{kl} \ln\left(\frac{p_l}{p_m}\right) + \delta_{ke} \ln\left(\frac{p_e}{p_m}\right) \\
s_l &= \beta_l + \delta_{kl} \ln\left(\frac{p_k}{p_m}\right) + \delta_{ll} \ln\left(\frac{p_l}{p_m}\right) + \delta_{le} \ln\left(\frac{p_e}{p_m}\right) \\
s_e &= \beta_e + \delta_{ke} \ln\left(\frac{p_k}{p_m}\right) + \delta_{le} \ln\left(\frac{p_l}{p_m}\right) + \delta_{ee} \ln\left(\frac{p_e}{p_m}\right)
\end{aligned}$$

mit

$$\sum_{i=1}^{M} \beta_i = 1; \sum_{i=1}^{M} \delta_{ij} = \sum_{j=1}^{M} \delta_{ij} = 0 \tag{11}$$

8 Beispiele

Der Datensatz aus Berndt und Wood (1975) ist in **nlsur** enthalten, zum Vergleich mit Greene (2012) müssen jedoch drei Felder angepasst werden.

```
library(nlsur)

data( "costs" ) # 25 Obs.

dd <- costs
# Patch zum Abgleich mit Greenes 7. Auflage
dd$Sm[dd$Year == 1958] <- 0.61886
dd$Pe[dd$Year == 1950] <- 1.12442
dd$Pm[dd$Year == 1949] <- 1.06625
```

8 Beispiele

Gleichungen als Liste formulieren und schätzen

```
eqns <- list(
  Sk ~ bk + dkk*log(Pk/Pm) + dkl*log(Pl/Pm) + dke*log(Pe/Pm),
  Sl ~ bl + dkl*log(Pk/Pm) + dll*log(Pl/Pm) + dle*log(Pe/Pm),
  Se ~ be + dke*log(Pk/Pm) + dle*log(Pl/Pm) + dee*log(Pe/Pm)
)

erg <- nlsur(eqns = eqns, data = dd, type = 2,
             trace = FALSE, eps = 1e-10)
erg
```

```
##             be             bk             bl            dee            dke
##  4.383281e-02  5.682400e-02  2.535458e-01  2.938303e-02 -8.203481e-03
##            dkk            dkl            dle            dll
##  2.987036e-02  2.207618e-05 -3.211908e-03  7.487719e-02
```

8 Beispiele

Jetzt mit **nlcom** aus den Restriktionen die übrigen Parameter und deren Standardfehler ermitteln

```
# nlcom
bm  <- nlcom(object = erg, form = "1 -be -bk -bl",  rname = "bm")
dkm <- nlcom(object = erg, form = "-dkk -dkl -dke", rname = "dkm")
dlm <- nlcom(object = erg, form = "-dkl -dll -dle", rname = "dlm")
dem <- nlcom(object = erg, form = "-dke -dle -dee", rname = "dem")
dmm <- nlcom(object = erg, form = "-dkm -dlm -dem", rname = "dmm")

# Alles kombinieren
est <- summary(erg)$coefficients     # direkt geschaetzt
ind <- rbind(bm, dkm, dlm, dem, dmm) # indirekt

res <- rbind(est, ind) %>% .[order(rownames(.)),]
```

8 Beispiele

```r
# Direkte Schaetzung und indirekt kombiniert plus Sternchen
printCoefmat(as.data.frame(res))
```

```
##        Estimate  Std. Error  z value  Pr(>|z|)
## be   4.3833e-02  1.0489e-03  41.7892 < 2.2e-16 ***
## bk   5.6824e-02  1.3072e-03  43.4698 < 2.2e-16 ***
## bl   2.5355e-01  1.9873e-03 127.5844 < 2.2e-16 ***
## bm   6.4580e-01  2.9936e-03 215.7275 < 2.2e-16 ***
## dee  2.9383e-02  7.4058e-03   3.9676 7.838e-05 ***
## dem -1.7968e-02  1.0754e-02  -1.6708   0.09511 .
## dke -8.2035e-03  4.0609e-03  -2.0201   0.04367 *
## dkk  2.9870e-02  5.7502e-03   5.1947 2.537e-07 ***
## dkl  2.2076e-05  3.6748e-03   0.0060   0.99521
## dkm -2.1689e-02  9.6307e-03  -2.2521   0.02456 *
## dle -3.2119e-03  2.7481e-03  -1.1688   0.24280
## dll  7.4877e-02  6.3935e-03  11.7114 < 2.2e-16 ***
## dlm -7.1687e-02  9.4093e-03  -7.6188 6.484e-14 ***
## dmm  1.1134e-01  2.2398e-02   4.9711 7.971e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

8.3 Almost-Ideal Demand System

Beispiel 2: Das Almost-Ideal Demand System (Deaton und Muellbauer 1980a,b)

$$w_i = \alpha_i + \sum_j \gamma_{ij} \log p_j + \beta_i \log\{x/P\} \tag{12}$$

$$\text{mit: } \log P = \alpha_0 + \sum_k \alpha_k \log p_k + \frac{1}{2} \sum_j \sum_k \gamma_{kj} \log p_k \log p_j . \tag{13}$$

$$\sum_{i=1}^{n} \alpha_i = 1 \qquad \sum_{i=1}^{n} \gamma_{ij} = 0 \qquad \sum_j \gamma_{ij} = 0 \qquad \gamma_{ij} = \gamma_{ji} \tag{14}$$

8 Beispiele

Beispieldaten aus Blanciforti et al. (1986) laden und AI-System schätzen. Die restringierten Parameter werden ignoriert

```
library(micEconAids)

data( "Blanciforti86" )
Blanciforti86 <- Blanciforti86[ 1:32, ]

w <- c( "wFood1", "wFood2", "wFood3", "wFood4" )
p <- c( "pFood1", "pFood2", "pFood3", "pFood4" )
x <- "xFood"

estAi <- ai(w = w, p = p, x = x, data = Blanciforti86)
```

8 Beispiele

```
## NLSUR Object of type: IFGNLS
##
##           n k     RMSE      MAE R-squared Adj-R-sqr. Const
## 1 wFood1 32 5 0.007465 0.005893    0.7258     0.6852   a01
## 2 wFood2 32 5 0.008151 0.006253    0.7482     0.7109   a02
## 3 wFood3 32 5 0.003819 0.003012    0.3384     0.2403   a03
##
## Coefficients:
##         Estimate Std. Error z value Pr(>|z|)
## a01    2.702e-01  4.218e-03  64.061  < 2e-16 ***
## a02    1.891e-01  4.406e-03  42.919  < 2e-16 ***
## a03    1.412e-01  2.151e-03  65.649  < 2e-16 ***
## b01    9.876e-05  1.086e-05   9.096  < 2e-16 ***
## b02    4.145e-05  1.129e-05   3.671 0.000251 ***
## b03   -2.153e-05  5.736e-06  -3.753 0.000181 ***
## g0101  1.575e-03  1.971e-04   7.993 2.66e-15 ***
## g0102 -1.346e-03  1.877e-04  -7.170 1.20e-12 ***
## g0103 -1.766e-04  1.039e-04  -1.700 0.089407 .
## g0202  2.198e-03  3.239e-04   6.788 1.65e-11 ***
## g0203 -1.890e-04  1.573e-04  -1.201 0.229792
## g0303  2.218e-04  1.340e-04   1.655 0.098099 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## Log-Likelihood: 358.5924
```

8.4 nlsur vs nls

```
data( "mtcars" )

model <- c("mpg ~ beta0 + beta1 * cyl + beta2 * am")
strt <- c(beta0=0,beta1=0,beta2=0)

res1 <- lm(mpg ~ cyl + am, data = mtcars)
res2 <- nlsur(model, data = mtcars, type = 1, startvalues = strt)
res3 <- nls(model, data = mtcars, start = strt)
```

```
##           beta0     beta1    beta2
## lm    34.52244 -2.500958 2.567035
## nlsur 34.52244 -2.500958 2.567035
## nls   34.52244 -2.500958 2.567035
```

8 Beispiele

```r
# Das Modell mit einer konstanten Variablen
mtcars <- subset(mtcars, am == 1)
strt <- c(beta0=0,beta1=0,beta2=0)

res1 <- lm(mpg ~ cyl + am, data = mtcars)
res2 <- nlsur(model, data = mtcars, type = 1, startvalues = strt)
try(res3 <- nls(model, data = mtcars, start = strt))
# nls findet keine Loesung
```

```
##           beta0     beta1 beta2
## lm    41.04894 -3.280851    NA
## nlsur 41.04894 -3.280851    NA
```

9 Literatur

Bates, Douglas und Martin Maechler (2015). *Matrix: Sparse and Dense Matrix Classes and Methods*.

Bates, Douglas M. und Donald G. Watts (1981). "A Relative Offset Orthogonality Convergence Criterion for Nonlinear Least Squares". In: *Technometrics* 23.2.

— (2008). *Nonlinear Regression Analysis and Its Applications*. New York: John Wiley & Sons, Inc.

Berndt, Ernst R und David O Wood (1975). "Technology, prices, and the derived demand for energy". In: *The review of Economics and Statistics*, 259–268.

Blanciforti, Laura, Richard D. Green und Gordon A. King (1986). *U.S. Consumer Behavior Over the Postwar Period: An Almost Ideal Demand System Analysis*. Giannini Foundation Monograph Number 40. University of California, Davis.

Chaussé, Pierre (2010). "Computing Generalized Method of Moments and Generalized Empirical Likelihood with R". In: *Journal of Statistical Software, Articles* 34.11, 1–35.

Davidson, Russel und James G. MacKinnon (2004). "Nonlinear Regression". In: *Econometric Theory and Methods*. New York: Oxford University Press. Kap. 6, 213–256.

Deaton, Angus und John Muellbauer (1980a). "An Almost Ideal Demand System". In: *The American Economic Review* 70.3, 312–326.

— (1980b). *Economics and consumer behavior*. Cambridge [u.a.]: Cambridge University Press. XIV, 450.

Eddelbuettel, Dirk und Conrad Sanderson (2014). "RcppArmadillo: Accelerating R with high-performance C++ linear algebra". In: *Computational Statistics and Data Analysis* 71, 1054–1063.

Gallant, A. Ronald (1987). *Nonlinear Statistical Models*. New York: John Wiley & Sons.

Greene, William H. (2012). *Econometric Analysis - International Edition*. 7. Auflage. Edinburgh Gate, Harlow: Pearson Education Limited.

Henningsen, Arne und Jeff D. Hamann (2007). "systemfit: A Package for Estimating Systems of Simultaneous Equations in R". In: *Journal of Statistical Software* 23.4, 1–40.

Judge, George G., R. Carter Hill, William E. Griffiths, Helmut Lütkepohl und Tsoung-Chao Lee (1988). *Introduction to the theory and practice of econometrics*. 2. ed. New York [u.a.]: Wiley. XXXVII, 1024.

Klein, L. R. und H. Rubin (1947). "A Constant-Utility Index of the Cost of Living". In: *The Review of Economic Studies* 15.2, 84–87.

Nash, John C. (2014). "On Best Practice Optimization Methods in R". In: *Journal of Statistical Software* 60.2, 1–14.

9 Literatur

Nash, John C. und Ravi Varadhan (2011). “Unifying Optimization Algorithms to Aid Software System Users: optimx for R”. In: *Journal of Statistical Software* 43.9, 1–14.

Nielsen, Hans Bruun (2000). *UCMINF – An Algorithm For Unconstrained, Nonlinear Optimization*. Report IMM-REP-2000-19. Lyngby: Technical University of Denmark.

Nielsen, Hans Bruun und Stig Bousgaard Mortensen (2012). *ucminf: General-purpose unconstrained non-linear optimization*.

Wooldridge, Jeffrey M. (2002). *Econometric Analysis of Cross Section and Panel Data*. Cambridge, Massachusetts: The MIT Press.

Zellner, Arnold (1962). “An Efficient Method of Estimating Seemingly Unrelated Regressions and Tests for Aggregation Bias”. In: *Journal of the American Statistical Association* 57.298, 348–368.

Vielen Dank für die Aufmerksamkeit

Fragen?

Jan Marvin Garbuszus

jan.garbuszus@ruhr-uni-bochum.de

A Anhang

```
set.seed(123)

dat <- data.frame(q1 = sample(x = 500:1000, size = 100,
                  replace = TRUE),
                  q2 = sample(x = 0:200, size = 100,
                  replace = TRUE),
                  q3 = sample(x = 400:1000, size = 100,
                  replace = TRUE))

dat$x <- rowSums(dat)
```

A Anhang

Nach Judge et al. (1988) ergibt sich die asymptotische Kovarianzmatrix $\mathbf{V}$ als

$$\mathbf{V} = \left(\mathbf{J}' \hat{\mathbf{\Sigma}}^{-1} \mathbf{J} \right)^{-1}. \tag{15}$$

A Anhang

Wooldridge (2002, 160) diskutiert FGLS und notiert die Schätzung der robusten Kovarianzmatrix nach White als

$$\mathbf{V} = \left(\mathbf{J}'\hat{\mathbf{\Sigma}}^{-1}\mathbf{J}\right)^{-1}\left(\mathbf{J}'\hat{\mathbf{\Sigma}}^{-1}\hat{\epsilon}\hat{\epsilon}'\hat{\mathbf{\Sigma}}^{-1}\mathbf{J}\right)\left(\mathbf{J}'\hat{\mathbf{\Sigma}}^{-1}\mathbf{J}\right)^{-1} \tag{16}$$

A Anhang

Für IFGNLS kann zusätzlich eine log Likelihood angegeben werden Davidson und MacKinnon 2004, 521. Diese ergibt sich als

$$\ln L = -\frac{mn}{2}(1 + \log 2\pi) - \frac{n}{2} \log \left| \hat{\mathbf{\Sigma}} \right| . \tag{17}$$