Schätzung nichtlinearer Gleichungssysteme in R mit `nlsur`

26. Juli 2016

`nlsur` ist ein R-Paket zur Schätzung von Nonlinear Least Squares (NLS). Schätzung können sowohl für einzelne Gleichungen wie für Gleichungssysteme ausgeführt werden. Neben NLS können auch Feasible Generalized NLS (FGNLS) oder Iterative FGNLS (IFGNLS) Schätzungen erfolgen. Die Funktionen des R Pakets orientiert sich an denen des Stata Befehls nlsur (StataCorp LP, 2015b).

1. Hintergrund

Die Schätzung einer einzelnen nichtlinearen Gleichung mittels Least Squares ist in R (R Core Team, 2015) über den Befehl `nls` möglich. Eine einzelne Gleichung kann auch mit `nlsur` geschätzt werden, der Fokus von `nlsur` liegt allerdings auf der Schätzung von nichtlinearen Gleichungssystemen. Die Schätzung von linearen Gleichungssystems ist seit dem *Seemingly Unrelated Regression* Modell von Zellner (1962) Usus und bereits im R-Paket `systemfit` (Henningsen und Hamann, 2007) implementiert.[1] Das `nlsur` Paket erweitert R um die Fähigkeit einer nichtlinearen Systemschätzung, die der zugrundeliegenden Theorie der linearen Systemschätzung gleicht. Nichtlineare Systemschätzungen wie FGNLS und IFGNLS werden u. a. in Greene (2012, 345ff.) besprochen. Das lineare Pendant der Feasible GLS Schätzer wird bei Wooldridge (2002, 157ff.) diskutiert.

[1] In `systemfit` ist mit `nlsystemfit()` eine Variante implementiert, die mittels nichtlinearer Optimierung ein nichtlineares SUR lösen soll. Diese Variante greift auf `nlm()` zurück, welches einer von vielen in R implementierten Optimierungsalgorithmen ist. Ausgehend von dieser Funktion wurden einige Veränderungen vorgenommen, um IFGNLS zu ermöglichen. Die Ergebnisse verschiedener nichtlinearer Optimierungsverfahren wiesen jedoch jeweils zum Teil deutlich unterschiedliche Ergebnisse auf. Die vielversprechendsten dabei konnten mit dem Paket ucminf (Nielsen und Mortensen, 2012) erzielt werden, welches ein von Nielsen (2000) entwickeltes Optimierungsverfahren nutzt. Die Variante mit nichtlinearer Optimierung ist jedoch grundsätzlich sehr rechenintensiv und führen auch bei identischen Startwerten nicht zwangsläufig zu ein und dem selben Ergebnis. Dazu kommt ein immenser Speicherverbrauch, weshalb die Variante der nichtlinearen Optimierung verworfen wurde.

Der wichtigste Befehl des Pakets ist `nlsur()`. Diese wird mit der Option `type =` 1 bis 3 aufgerufen. Dabei entspricht 1 einer NLS, 2 FGNLS und 3 einer IFGNLS Schätzung. Der Vorteil einer IFGNLS Schätzung ist, dass die Ergebnisse mit denen einer Maximum Likeliehood Schätzung übereinstimmen (vgl. Greene, 2012, S. 349). Die zugrundeliegende Theorie entspricht bei Systemschätzungen der einfacher Gleichungen. Für die Schätzung in `nlsur` können deshalb sowohl Gleichungssysteme, wie auch einzelne Gleichungen genutzt werden. Für den Fall, dass nur eine einzelne Gleichung geschätzt wird, entspricht das Ergebnis dem von `nls`, wobei letzteres ein relatives Achsenabschnitts Konvergenkriterium nutzt (s. Bates und Watts, 2008, 49f.). In beiden Fällen wird jedoch mit Gauss-Newton gelöst.

Eine nichlineare Systemschätzung gestaltet sich dann wie folgt, es existiert ein Gleichungssystem mit N Beobachtungen und M Gleichungen der Form

$$\mathbf{y}_i = \mathbf{f}_i(\boldsymbol{\beta}, \mathbf{X}) + \epsilon_i, \qquad \text{für } i = 1, \ldots, M. \tag{1}$$

Die Anzahl der Beobachtungen ist T und die Anzahl der Parameter ist K, die über die Gleichungen verteilt sind. Weitere Einschränkungen gibt es für die K nicht, sie können in den Gleichungen exklusiv verteilt sein, aber auch in allen Gleichungen vorkommen.

Für die Fehlerterme wird angenommen, dass sie im Mittel 0 sind und die Kovarianzmatrix $\boldsymbol{\Sigma}$ aufweisen. NLS minimiert dann dann:

$$RSS(\hat{\boldsymbol{\beta}}) = \sum_{i=1}^{N} \{\mathbf{y}_i - \mathbf{f}_i(\boldsymbol{\beta}, \mathbf{X})\} \boldsymbol{\Sigma}^{-1} \{\mathbf{y}_i - \mathbf{f}_i(\boldsymbol{\beta}, \mathbf{X})\} \tag{2}$$

Werden die Gleichungen gestackt, kann $\boldsymbol{\Sigma}$ als $\boldsymbol{\Sigma} \otimes \mathbf{I}$ geschrieben werden.

Die $M \times M$ Gewichtungsmatrix $\boldsymbol{\Sigma} = E(\epsilon_i' \epsilon_i)$ ist jedoch zumeist unbekannt. Aus diesem Grund wird bei unbekannter Gewichtungsmatrix auf die Einheitsmatrix zurück gegriffen. Dadurch wird der erste Schritt in NLS für eine Systemschätzung ohne weitere Iterationen ein ineffizienter Schätzer, da nicht beachtet werden kann, ob Variablen über die Gleichungen verteilt sind. Als Variante kann in diesem Falls auch Gleichung für Gleichung mit NLS gelöst werden (s. Greene, 2012, S. 346). Obwohl ineffizient, ist der Schätzer jedoch konsistent. Aus den Residuen lässt sich dann eine neue Gewichtungsmatrix $\hat{\boldsymbol{\Sigma}}$ ermitteln.

$$\hat{\boldsymbol{\Sigma}} = \frac{1}{N} \mathbf{u}'\mathbf{u} \tag{3}$$

Für eine FGNLS Schätzung wird die neue Gewichtungsmatrix verwendet, welche dann erneut eine Gewichtungsmatrix erzeugt. Für eine IFGNLS Schätzung wird die Berechnung so lange iterativ wiederholt, bis ein gewähltes Konvergenzkriterium erreicht ist.[2]

[2] Als Konvergenzkriterium wird üblicherweise die Veränderung der Koeffizienten und/oder die Veränderung der Gewichtungsmatrix gewählt.

Die asymptotische Kovarianzmatrix $\mathbf{V}$ der Schätzung ergibt sich nach Judge et al. (1988, S. 360) als

$$\mathbf{V} = \left(\mathbf{X}^{0\prime}\hat{\mathbf{\Sigma}}^{-1}\mathbf{X}^{0}\right)^{-1}. \tag{4}$$

Wobei $\mathbf{X}^0$ die Jacobi-Matrix ist, die Matrix der Partiellen Ableitungen evaluiert an den Parametern.

Wooldridge (2002, S. 160) diskutiert FGLS und notiert die Berechnung der robusten Kovarianzmatrix nach White

$$\mathbf{V} = \left(\mathbf{X}^{0\prime}\hat{\mathbf{\Sigma}}^{-1}\mathbf{X}^{0}\right)^{-1}\left(\mathbf{X}^{0\prime}\hat{\mathbf{\Sigma}}^{-1}\hat{\mathbf{u}}\hat{\mathbf{u}}'\hat{\mathbf{\Sigma}}^{-1}\mathbf{X}^{0}\right)\left(\mathbf{X}^{0\prime}\hat{\mathbf{\Sigma}}^{-1}\mathbf{X}^{0}\right)^{-1} \tag{5}$$

Für IFGNLS kann ergänzend eine log Likeliehood ausgegeben werden. Diese berechnet sich mit:

$$LL = -\frac{MN}{2}(1 + \log 2\pi) - \frac{N}{2}\log|\hat{\mathbf{\Sigma}}| \tag{6}$$

Zur Schätzung in R bieten sich zwei Variante an. Zum einen eine Variante mit dem Matrix Paket (Bates und Maechler, 2015) zum anderen über das lösen mittels Blockmatrizen einer Variante analog zur Stata Funktion.

2. Implementierung

Das Matrix Paket erlaubt eine Schätzung des NLS bei größeren Datensätzen. Das Matrix Paket wird benötigt, da beim stacken der Matrizen $\hat{\mathbf{\Sigma}}$ als $\hat{\mathbf{\Sigma}} \otimes \mathbf{I}$ genutzt wird. Die daraus resultierende Matrix wird eine $MN \times MN$ Matrix. Bereits bei wenigen Gleichungen und einem moderaten N wird die Matrix sehr groß und benötigt entsprechend viel Arbeitsspeicher (bspw. ist eine Matrix für $N = 3$ und $M = 10000$ bereits 3.6 GB groß). Mit der Matrix Bibliothek wird das Kronecker-Produkt aus $\hat{\mathbf{\Sigma}}$ und $\mathbf{I}$ als Sparse Matrix erzeugt. Diese Variante ist zwar wesentlich sparsamer, benötigt häufig trotzdem einen Computer mit großem Arbeitsspeicher.

2.1. Stata-Variante

NLS mit der Stata Variante funktioniert Speichersparsamer. Ausgangspunkt ist die Überlegung, dass $\mathbf{\Sigma}^{-1}$ mittels Cholesky-Dekomposition in eine neue Matrix $\mathbf{D}$ zerlegt werden kann. $\mathbf{DD}' = \mathbf{\Sigma}^{-1}$.

Postmultipliziert mit $\mathbf{D}$ ergibt sich

$$\mathbf{y}_i\mathbf{D} = \mathbf{f}_i(\boldsymbol{\beta}, \mathbf{X})\mathbf{D} + \epsilon_i\mathbf{D}, \qquad \text{für } i = 1, \ldots, M \tag{7}$$

Weil $E(\mathbf{D}'\mathbf{u}_i'\mathbf{u}_i\mathbf{D}) = \mathbf{I}$ können die Gleichungen gestackt werden und dann spaltenweise gelöst werden.

$$
\begin{aligned}
\mathbf{y}_1\mathbf{D}_1 &= \mathbf{f}(\mathbf{x}_1, \boldsymbol{\beta})\mathbf{D}_1 + \tilde{u}_{11} \\
\mathbf{y}_1\mathbf{D}_2 &= \mathbf{f}(\mathbf{x}_1, \boldsymbol{\beta})\mathbf{D}_2 + \tilde{u}_{12} \\
&\vdots \\
\mathbf{y}_1\mathbf{D}_M &= \mathbf{f}(\mathbf{x}_1, \boldsymbol{\beta})\mathbf{D}_M + \tilde{u}_{1M} \\
&\vdots \\
\mathbf{y}_N\mathbf{D}_1 &= \mathbf{f}(\mathbf{x}_N, \boldsymbol{\beta})\mathbf{D}_M + \tilde{u}_{N1} \\
\mathbf{y}_N\mathbf{D}_2 &= \mathbf{f}(\mathbf{x}_N, \boldsymbol{\beta})\mathbf{D}_M + \tilde{u}_{N2} \\
&\vdots \\
\mathbf{y}_N\mathbf{D}_M &= \mathbf{f}(\mathbf{x}_N, \boldsymbol{\beta})\mathbf{D}_M + \tilde{u}_{NM}
\end{aligned}
$$

Anschließend kann - wie im univariaten Fall - gelöst werden, mittels Gauss-Newton (Davidson und MacKinnon, 2004, 228ff. ; Bates und Watts, 2008, Kap. 2).

$$RSS(\beta) = \{\mathbf{y} - \mathbf{f}(\mathbf{x}, \boldsymbol{\beta})\}' \boldsymbol{\Sigma}^{-1} \{\mathbf{y} - \mathbf{f}(\mathbf{x}, \boldsymbol{\beta})\} \tag{8}$$

Mit einer Taylor-Erweiterung zweiter Ordnung zentriert auf β_0 erhält man

$$RSS(\beta) = RSS(\beta_0) + g'(\beta_0)(\beta - \beta_0) + \frac{1}{2}(\beta - \beta_0)'\mathbf{H}(\beta_0)(\beta - \beta_0) \tag{9}$$

dabei ist $\mathbf{g}(\cdot)$ der Gradient und $\mathbf{H}(\cdot)$ die Hesse-Matrix. Nach β abgeleitet erhält man:

$$\mathbf{g}(\beta) = -2\mathbf{X}^{0\prime}\boldsymbol{\Sigma}^{-1}\mathbf{u} \tag{10}$$

Die Hess'sche kann approximiert werden mit

$$\mathbf{H}(\beta) = 2\mathbf{X}^{0\prime}\boldsymbol{\Sigma}^{-1}\mathbf{X}^{0} \tag{11}$$

Wird (9) gelöst und null gesetzt ist die Bedingung für ein Minimum:

$$\mathbf{g}(\beta_0) + \mathbf{H}(\beta_0)(\beta - \beta_0) = \mathbf{0}. \tag{12}$$

Was iterativ gelöst werden kann als

$$\beta_{j+1} = \beta_j - \alpha\mathbf{H}^{-1}(\beta_j)\mathbf{g}(\beta_j) \tag{13}$$

und mit (10) und (11) zu

$$\beta_{j+1} = \beta_j + \alpha(\mathbf{X}^{0\prime}\boldsymbol{\Sigma}^{-1}\mathbf{X}^{0})^{-1}\mathbf{X}^{0\prime}\boldsymbol{\Sigma}^{-1}\mathbf{u} \tag{14}$$

führt. In (14) ist α ein *stepsize* Parameter, den Box (1960) und Hartley (1961) eingeführt haben und der zwischen 0 und 1 liegt. Abgesehen von α kann (14) mittels einer gewichteten Regression von $\mathbf{X}^0$ auf $\mathbf{u}$ geschätzt werden. Für den Fall, dass eine NLS ohne Gewichtungsmatrix gerechnet wird ($\boldsymbol{\Sigma} = \mathbf{I}$), entfällt die Gewichtung.

3. Syntax

Das Paket `nlsur` bringt eine wesentliche Funktion mit, die durch den Nutzer aufgerufen wird. Diese ist `nlsur()`.

```
nlsur(eqns, data, startvalues, type = NULL, S = NULL, debug = FALSE,
  trace = FALSE, stata = TRUE, qrsolve = FALSE, weights, MASS = FALSE,
  eps = 1e-05, ifgnlseps = 1e-10, tau = 1e-04, maxiter = 1000)
```

- `eqns` ist entweder ein Listenobjekt bestehend aus einer Gleichung oder einem Gleichungssystem oder eine einzelne Gleichung als String oder formula-Objekt.
- `data` ist ein data frame Objekt mit Variablen.
- `startvalues` ist ein benannter Vektor , welcher für alle zu schätzenden Parameter aus `eqns` einen Wert bereitstellt.
- `type` kann die Ausprägungen 1, 2 oder 3 alternativ “nls”, “fgnls” oder “ifgnls” annehmen.
- `S` ist eine Gewichtungsmatrix, diese bleibt üblicherweise leer, kann aber falls bereits eine Schätzung durchgeführt wurde, hier übergeben werden.
- `debug` gibt eine Reihe von debug-Informationen aus.
- `trace` gibt Informationen darüber, welcher Schätzer läuft und welchen Wert die residual sum of squares zuletzt hatte.
- `stata` ist ein Bool. Wenn `TRUE` und nls gewählt ist, dann wird nach dem ersten NLS eine zweite Schätzung mit einer Gewichtungsmatrix aus der Diagonalen der im 1. Durchlauf ermittelten Gewichtungsmatrix berechnet.
- `qrsolve` ist ein Bool. Wenn `TRUE`, dann wird anstelle von `calc_reg` eine Lineare Regression mittels QR-Zerlegung ermittelt.
- `weights` ein möglicher Vektor mit Gewichten.
- `maxiter` maximale Anzahl an Iterationen.
- `eps` ist für das Konvergenzkriterium der NLS-Funktion.
- `tau` ist ebenfalls für das Konvergenzkriterium der NLS-Funktion.
- `ifgnlseps` ist für das Konvergenzkriterium der IFGNLS Funktion.

4. Code Review

Die Funktion `nlsur` ruft in Abhängigkeit von der gewählten Schätzvariante die Funktion `.nlsur()` auf, welche ein Ergebnisobjekt der Klasse "nlsur" erzeugt. Obwohl die Funktion `.nlsur()` jeweils für einen Aufruf von NLS gewählt wird, soll die Funktion selbst nicht durch den Nutzer aufgerufen werden.

4.1. NLS

4.1.1. Vorbereitung

Zunächst prüft die Funktion, ob eine Gewichtungsmatrix (`S`) existiert. Falls nicht, wird aus der Diagonalen eine neue Gewichtungsmatrix erzeugt. Sobald eine Gewichtungsmatrix existiert, wird diese invertiert (`qS`) und anschließend als Cholesky-Dekomposition gespeichert (`s`).
Der Vektor mit den Startwerten wird mittels `assign()` für `eval()` verfügbar gemacht. Daran anschließend werden in einer Schleife mit `eval()` die linke Seite der Gleichung (LHS) oder y in das Objekt `lhs`, die rechte Seite der Gleichung (RHS) oder $f(x, \beta)$ in das Objekt `rhs`, die Residuen (u) oder $y - f(x, \beta)$ in das Objekt `ri` und die Jacobi-Matrix in das Objekt `xi` berechnet. Dieser Schritt wird im Falle eines Gleichungssystems Gleichung für Gleichung vollzogen und die Ergebnisse werden jeweils in Listenobjekten abgespeichert.

Nach der Schleife werden die einzelnen Matrizen-Objekte aus den jeweiligen Liste nebeneinander angeordnet. Aus den Residuen und der Cholesky-Dekomposition der Inversen der Gewichtungsmatrix Σ wird die Summe der quadrierten Residuen berechnet (SSR). Dazu werden die Objekte `r` (die zeilenweise kombinierten Elemente aus `ri`) und `s` an eine Funktion übergeben. Die Funktion ist aus Geschwindigkeitsgründen in RcppArmadillo (Eddelbuettel und Sanderson, 2014) umgesetzt.

```
for (int j = 0; j < neqs; ++j) {
  for (int i = 0; i < n; ++i){
    ssr += pow( r.row(i) * s.col(j), 2);
  }
}
```

Das entspricht

$$RSS(\beta) = \mathbf{u}'\mathbf{D}'\mathbf{D}\mathbf{u} \tag{15}$$

4.1.2. While-Schleifen

Nach der RSS Schätzung wird der *stepsize* Parameter auf den Wert 1 gesetzt und eine while-Schleife gestartet, die auf Konvergenz prüft. Sofern ein NLS geschätzt wird, werden `r` und `xi` zunächst Spaltenweise kombiniert. Anschließend wird eine Lineare Regression

zwischen $\mathbf{X}^0$ und `r` geschätzt.[3] [4] Wird die Option `qrsolve = TRUE` gewählt und keine Matrix `S` mitgegeben, wird eine lineare Regression mithilfe einer QR-Dekomposition durchgeführt:

```
qr.coef(qr(x), r)
```

Das entspricht, weil jedes Element der Diagonalen von $\hat{\mathbf{\Sigma}}^{-1}$ 1 ist:

$$\theta = (\mathbf{X}^{0\prime}\mathbf{X}^0)^{-1}\mathbf{X}^{0\prime}\mathbf{u} \tag{16}$$

Nach der Schätzung beginnt eine zweite while-Schleife, die so lange läuft, wie

$$RSS(\beta_i) > RSS(\beta_{i-1}). \tag{17}$$

Eingangs der Schleife werden neue β-Werte berechnet. Dazu wird auf (14) zurückgegriffen:

$$\beta_{i+1} = \beta_i + \alpha\theta_i \tag{18}$$

Startwerte der Gleichung sind β_i, α ist 1 und θ_i sind die Koeffizienten der zuvor berechneten Regression. Die neuen β-Werte werden wieder für `eval()` verfügbar gemacht. Es werden wieder `lhs`, `rhs`, `ri` und `xi` ausgerechnet sowie basierend darauf ein neues RSS. Wird die Schleife nicht beendet, dann wird α halbiert und es werden neue β-Werte ermittelt.
Wird die Schleife beendet, wird geprüft ob Konvergenz erreicht ist. Konvergenz ist erreicht, wenn beide Stoppkriterien erfüllt sind:

1. $|SSR_{i-1} - SSR| \leq \epsilon(SSR_{i-1} + \tau)$ Diese Konvergenzkriterium weicht geringfügig von Gallant (1987, S. 29) ab, auf dessen Konvergenkriterium Stata sich bezieht. Bei Gallant wird anstelle von $\leq$ lediglich $<$ genutzt.

2. Als zweites wird für $m = 1, \ldots, k$ das Konvergenzkriterium $\alpha|\theta_{jm}| \leq \epsilon(|\beta_{j-1,m}|+\tau)$ geprüft. Wobei θ_j das Ergebnis der gewichteten Regression ist. [5]

[3] Die mittels `rbind()` kombinierten Matrizen von `r` und `xi` sind wieder groß und die anschließende Regression mittels QR-Zerlegung ist speicherintensiv, gibt jedoch auch für "schlecht spezifizierte" Modelle Koeffizienten aus. Wenn einer oder mehrere der resultierenden Koeffizienten wegen Singulärer Matrizen Missings erzeugen, dann gibt die QR-Variante Ergebnisse aus, die Variante mittels `calc_reg` in RcppArmadillo findet hingegen hier keine (oder nur eine approximative) Lösung.

[4] Die Funktion `lm.gls` aus dem Paket `MASS` kann ebenfalls eine GLS mit Gewichtungsmatrix rechnen. Wird die Option `MASS = TRUE` gewählt, wird anstelle der Stata Variante auf diese Funktion zurückgegriffen. Dafür wird jedoch zunächst mittels Eigenvektor eine Matrix erzeugt, die mit $\mathbf{y}$ und $\mathbf{X}$ multipliziert jeweils die Matrizen gewichtet ergibt. Die Berechnung des Eigenvektors erfolgt dann in R jedoch mit "normalen" Matrizen. D.h. die Menge an benötigtem RAM steigt wieder drastisch an.

[5] Stata dokumentiert für den Befehl `nl`, dass das Konvergenkriterium $\alpha|\beta_{j+1,m}| \leq \epsilon(|\beta_{jm}| + \tau)$ sei. Das widerspricht allerdings auch Gallant (1987, S. 29), dort wird $||\beta_i - \beta_{i+1}|| < \epsilon(||\beta_i|| + \tau)$ vorgeschlagen. Letztlich sollte die Wahl des Konvergenkriteriums unerheblich sein, solange es denn funktioniert.

Ist keine Konvergenz erreicht, so wird das letzte RSS zum neuen Zielkriterium. In Stata wird dazu die letzte while-Schleife erneut begonnen und der *stepsize* Parameter wieder auf 1 gesetzt. In `nlsur` wird α nach jeder erfolgreichen Schleife verdoppelt, bis maximal wieder 1 erreicht ist. Das entspricht der Variante von `nls` in R. Ist Konvergenz erreicht, werden die berechneten Koeffizienten ausgegeben.

Stata notiert für den Fall, wenn die Option `nls` gewählt wird, dass das Ergebnis nun vollständig sei und keine weitere Berechnung ausgeführt würde. Intern nimmt Stata nun die Diagonale der letzte Gewichtungsmatrix und ersetzt die Werte auf der Diagonalen von $\mathbf{I}$ damit. Mit dieser Gewichtungsmatrix und den βs wird damit ein weiterer "stiller" Durchlauf des oben beschriebenen Vorgehens durchgeführt. Für `nlsur` lässt sich dieses Verhalten über die Option `stata = TRUE` erreichen. Die Option hat keinen Einfluss auf (I)FGNLS.

4.2. FGNLS

Für den Fall, dass FGNLS gewählt wurde, werden die zuletzt berechneten β-Werte genommen und die zuletzt berechnete Gewichtungsmatrix. Basierend darauf wird eine NLS-Schätzung durchgeführt analog zum zuvor beschriebene Vorgang. Der - von der verwendeten Gewichtungsmatrix abgesehen - einzige Unterschied: Die WLS erfolgt nun ebenfalls in einer C++-Schleife.
Die WLS kann, wenn die Option `MASS = TRUE` gewählt ist, mittels der Funktion `lm.gls()` aus dem Paket `MASS` (Venables und Ripley, 2002) berechnet werden. Dabei erfolgt die Gewichtung über die Multiplikation mit den Eigenwerten der Gewichtungsmatrix, damit anschließend eine lineare Regression mittels QR-Dekomposition erfolgen kann. Die Eigenwertzerlegung der Gewichtungsmatrix wandelt allerdings potentielle Sparse Matrizen in gewöhnliche Matrizen um, wobei bei entsprechend großer Fallzahl wieder das Eingangs skizzierten Speicherproblem auftreten kann. Alternativ kann die WLS durch Lösen in Matrix Notation wie in (14) erfolgen, wobei die Inverse Gewichtungsmatrix genutzt werden kann. Anstatt diese mit dem Kroneckerprodukt aufzublähen, lässt sich die Berechnung wesentlich speichereffizienter lösen.

```
int n = r.n_rows;
int xicol = x.n_cols / neqs;

for (int i = 0; i < n; ++i) {
  arma::Mat<double> xi = x.row(i);
  arma::Mat<double> XI = arma_reshape(xi, xicol);

  arma::Mat<double> YI = r.row(i).t();

  XDX += XI.t() * qS * XI;
  XDy += XI.t() * qS * YI;
}
```

```
XDX = 0.5 * ( XDX + XDX.t() );
```

Die Berechnung erfolgt hierbei ebenfalls unter Berücksichtigung der Geschwindigkeit in C++. Die Funktion `arma_reshape` ahmt die Stata Funktion `rowshape` nach.

4.3. IFGNLS

Für IFGNLS wird zunächst ein FGNLS geschätzt. Ausgehend davon beginnt eine neue while-Schleife mit einem neuen Konvergenzkriterium. Es erfolgt abermals eine NLS Schätzung, mit der zuletzt berechneten Gewichtungsmatrix und den zuletzt berechneten Koeffizienten.
Die ermittelten Residuen und die Cholesky-Dekomposition der Inversen der Gewichtungsmatrix werden genutzt, um ein neues RSS zu berechnen. Anschließend wird die maximale relative Differenz der Gewichtungsmatrix und die der ermittelten Koeffizienten berechnet. Das Konvergenzkriterium ist für die Koeffizienten, dass die Differenz der β kleiner ϵ und für die $\mathbf{\Sigma}$ kleiner 10^{-10} ist. Es muss eins der beiden Kriterien erfüllt sein, damit Konvergenz erklärt wird. Wenn Konvergenz erreicht ist, wird die log Likelihood ermittelt.

4.4. Nach der ((I)FG)NLS Schätzung

Im Anschluss an eine erfolgreiche nlsur Schätzung lassen sich verschiedentliche Ergebnisse ausgeben. Der Aufruf des print Befehls gibt die Koeffizienten aus, ebenso wie `coef()`. Eine zusammenfassende Darstellung des Ergebnisobjekts ermöglicht `summary()`. Dieser Befehl versucht zudem zu ermitteln, ob die einzelnen geschätzten Gleichungen eine Konstante aufweisen. Die Kovarianzmatrix ermittelt `vcov()`, welches ganz analog zu den bekannten und ebenfalls für nlsur verfügbaren Befehlen wie `residuals()`, `deviance()`, `df.residuals()`, `fitted()`, `logLik()`, `convint()` oder `predict()` arbeitet.

5. Zusätzlich

5.1. `nlsur` vs `nls`

`nlsur` kann wie `nls` zur Schätzung nichtlinearer Least Squares genutzt werden. Unterschiede liegen im Detail. `nls` nutzt das relative Offset Kriterium nach Bates und Watts (1981) auf welches in `nlsur` nicht ohne weiteres möglich ist, hier müsste $\mathbf{X}^0$ aus (14) zunächst gewichtet werden, was aufgrund der geschilderten großen Matrizen problematisch ist. So bietet `nlsur` ein auf einfacheren Konvergenkriterien beruhendes Verfahren, was die Möglichkeit eines Vergleichs bietet.

Für den Fall eines einfachen linearen Modells, welches mit lm, nlsur und nls geschätzt wird, ergeben sich identische

```
data(mtcars)
model <- c("mpg ~ beta0 + beta1 * cyl + beta2 * am")
```

```
res1 <- res2 <- res3 <- NULL

res1 <- lm(mpg ~ cyl + am, data = mtcars)
res2 <- nlsur(model, data = mtcars, type = 1, stata = FALSE)
res3 <- nls(model, data = mtcars, start =c(beta0=0,beta1=0,beta2=0))

summary(res1)
summary(res2)
summary(res3)
```

Über den Befehl `nlcom()` können nichtlineare Kombinationen aus Parameterschätzern bestimmt und dazugehörige Konfidenzintervalle ausgegeben werden. Dazu wird die delta-Methode genutzt. Im wesentlichen ist `nlcom()` ein Wrapper um `deltaMethod()` aus dem Paket `car` (Fox und Weisberg, 2011). `nlcom()` funktioniert analog zum Stata Befehl `nlcom` (StataCorp LP, 2015a).

Beispiel Zur Anwendung kann nlcom kommen, wenn Parameter ökonometrischen Modellen mittels Restriktionen indirekt ermittelt werden.

```
nlcom(object, form, alpha, rname)
```

- object ist ein nlsur Ergebnisobjekt.
- form ist ist die zu ermittelnde Parameterkombination als String (z. B. `"a/b"` für die Differenz aus den Koeffizienten a und b).
- alpha ist das zur Bestimmung des Konfidenzintervals genutzte α standardmäßig 5%.
- rname ist ein optionaler Zeilenname, wird keiner gesetzt, wird standardmäßig form genommen.

`nlcom()` kann mit Objekten der Klasse nlcom aus dem globalen Environment kombiniert werden d. h. in form kann ein nlcom-Objekt gewählt werden. Eine Anwendung dazu findet sich im Beispiel.

6. Anwendung 1

Eine beispielhafte Anwendung für eine nichtlineare Systemschätzung liefert *Example 10.3 A Cost Function for U.S. Manufacturing* aus Greene (2012, 353f.). Es wird ein Translog für vier Güterkategorien als FGNLS geschätzt:

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

mit

$$
\begin{aligned}
\beta_1 + \beta_2 + \cdots + \beta_M &= 1 \\
\sum_{i=1}^{M} \delta_{ij} &= 0 \\
\sum_{j=1}^{M} \delta_{ij} &= 0
\end{aligned}
\tag{19}
$$

Modell und Daten des Beispiels entstammen Berndt und Wood (1975), als Vergleich sind in Tabelle 1 die von Greene (2012) berechneten FGNLS Parameterschätzungen angegeben.

7. Anwendung 2

Ein weiteres Beispiel für eine Anwendung ist eine Systemschätzung. Diesmal wird zum Vergleich das Almost Ideal Demand System von Deaton und Muellbauer (1980) herangezogen. Das Gleichungssystem kann wie folgt geschrieben werden:

$$
w_i = \alpha_i + \sum_j \gamma_{ij} \log p_j + \beta_i \log\{x/P\} \tag{20}
$$

$$
\sum_{i=1}^{n} \alpha_i = 1 \qquad \sum_{i=1}^{n} \gamma_{ij} = 0 \qquad \sum_{j} \gamma_{ij} = 0 \qquad \gamma_{ij} = \gamma_{ji} \tag{21}
$$

Deaton und Muellbauer (ebd.) schlagen neben einer Variante mit dem translog Preisindex eine Schätzung mit dem Stone-Preisindex $\log P = \sum w_k * \log p_k$ als vollständig linearisierte Variante des AI-System vor. Eine solche Variante ist im Paket `micEconAids` von Henningsen (2014) implementiert. Zur Schätzung des Almost Ideal Demand Systems wird auf den Datensatz von Blanciforti et al. (1986) aus Henningsen (2014) zurückgegriffen. Die Schätzung erfolgt über vier Lebensmittel Gruppen.

A. Anhang

```r
# Greene Example 10.3
library(nlsur)
```

```
## Loading required package: car
```

```
## Loading required package: Matrix
```

```
## Loading required package: MASS
```

```r
url <- "http://www.stern.nyu.edu/~wgreene/Text/Edition7/TableF10-2.txt"
dd <- read.table(url, header = T)

names(dd) <- c("Year", "Cost", "Sk", "Sl", "Se", "Sm", "Pk", "Pl", "Pe", "Pm")


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))

strtvls <- c(be = 0, bk = 0, bl = 0,
             dkk = 0, dkl = 0, dke = 0,
             dll = 0, dle = 0, dee = 0)


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

```
## -- NLS
## create initial weight matrix Sigma.
## Initial SSR:  2.009652
## SSR:  0.0009989223
## SSR:  0.0009989223
## -- FGNLS
## Initial SSR:  75
## SSR:  65.45196
## SSR:  65.45196
```

```r
erg
```

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

```r
summary(erg)
```

```
## NLSUR Object of type: FGNLS
##
##     n k     RMSE      MAE R.squared Adj.R.sqr. eqconst
## Sk 25 4 0.003121 0.002447    0.4942     0.4220       bk
```

```
## Sl 25 4 0.005354 0.003562    0.8200     0.7943      bl
## Se 25 4 0.001656 0.001550    0.7036     0.6613      be
##
## Coefficients:
##       Estimate Std. Error z value Pr(>|z|)
## be   4.383e-02  1.049e-03  41.789  < 2e-16 ***
## bk   5.682e-02  1.307e-03  43.470  < 2e-16 ***
## bl   2.535e-01  1.987e-03 127.584  < 2e-16 ***
## dkk  2.987e-02  5.750e-03   5.195 2.54e-07 ***
## dkl  2.208e-05  3.675e-03   0.006   0.9952
## dke -8.203e-03  4.061e-03  -2.020   0.0437 *
## dll  7.488e-02  6.394e-03  11.711  < 2e-16 ***
## dle -3.212e-03  2.748e-03  -1.169   0.2428
## dee  2.938e-02  7.406e-03   3.968 7.84e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
bm <- nlcom(object = erg, form = "1 -be -bk -bl", rname= "bm")
bm
```

```
## nlcom:  1 -be -bk -bl
##
##     Estimate Std. Error z value  Pr(>|z|)
## bm 0.6457974  0.0029936  215.73 < 2.2e-16 ***
```

```
dkm <- nlcom(object = erg, form = "-dkk -dkl -dke", rname = "dkm")
dkm
```

```
## nlcom:  -dkk -dkl -dke
##
##       Estimate Std. Error z value Pr(>|z|)
## dkm -0.0216890  0.0096307 -2.2521  0.02456 *
```

```
dlm <- nlcom(object = erg, form = "-dkl -dll -dle", rname = "dlm")
dlm
```

```
## nlcom:  -dkl -dll -dle
##
##       Estimate Std. Error z value  Pr(>|z|)
## dlm -0.0716874  0.0094093 -7.6188 6.484e-14 ***
```

```
dem <- nlcom(object = erg, form = "-dke -dle -dee", rname = "dem")
dem
```

```
## nlcom:  -dke -dle -dee
##
##      Estimate Std. Error z value Pr(>|z|)
## dem -0.017968   0.010754 -1.6708  0.09511 .
```

```
dmm <- nlcom(object = erg, form = "-dkm -dlm -dem", rname = "dmm")
dmm
```

```
## nlcom:  -dkm -dlm -dem
##
##     Estimate Std. Error z value  Pr(>|z|)
## dmm 0.111344   0.022398  4.9711 7.971e-07 ***
##
## nlcom object estimated with prior nlcom estimate.
```

```
# last one is equivalent to the longer form of:
# dmm <- nlcom(object = erg,
#  form = "-(-dkk -dkl -dke) -(-dkl -dll -dle) -(-dke -dle -dee)")
# dmm

est <- summary(erg)$coefficients
ind <- rbind(bm, dkm, dlm, dem, dmm)

res <- rbind(est, ind)
res <- res[order(rownames(res)),]

# results comparable to Greene 2012 p.353
res[, 1:2]
```

```
##     Estimate     Std. Error
## be  0.04383281   0.001048904
## bk  0.056824     0.001307207
## bl  0.2535458    0.001987279
## bm  0.6457974    0.002993579
## dee 0.02938303   0.007405766
## dem -0.01796764  0.01075402
## dke -0.008203481 0.004060895
## dkk 0.02987036   0.005750185
## dkl 2.207618e-05 0.00367483
## dkm -0.02168896  0.009630666
## dle -0.003211908 0.00274809
## dll 0.07487719   0.006393546
## dlm -0.07168736  0.009409309
## dmm 0.111344     0.02239838
```

β_K	0.05682	(0.00131)	δ_{KM}	−0.02169*	(0.00963)
β_L	0.25355	(0.001987)	δ_{LL}	0.07488	(0.00639)
β_E	0.04383	(0.00105)	δ_{LE}	−0.00321	(0.00275)
β_M	0.64580*	(0.00299)	δ_{LM}	−0.07169*	(0.00941)
δ_{KK}	0.02987	(0.00575)	δ_{EE}	0.02938	(0.00741)
δ_{KL}	0.0000221	(0.00367)	δ_{EM}	−0.01797*	(0.01075)
δ_{KE}	−0.00820	(0.00406)	δ_{MM}	0.11134*	(0.02239)

Tabelle 1: Parameterschätzungen von Beispiel 10.3 aus Greene, 2012, S. 353. Mit * gekennzeichnete Parameter sind indirekte Schätzungen mittels (19).

Literatur

Bates, D. und M. Maechler (2015). *Matrix: Sparse and Dense Matrix Classes and Methods*. R package version 1.2-2.

Bates, D. M. und D. 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, E. R. und D. O. Wood (1975). »Technology, prices, and the derived demand for energy«. In: *The review of Economics and Statistics*, S. 259–268.

Blanciforti, L., R. D. Green und G. 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.

Box, G. E. P. (1960). »Fitting Empirical Data«. In: *Annals of the New York Academy of Sciences* 86.3, S. 792–816.

Davidson, R. und J. G. MacKinnon (2004). »Nonlinear Regression«. In: *Econometric Theory and Methods*. New York: Oxford University Press. Kap. 6, S. 213–256.

Deaton, A. und J. Muellbauer (1980). »An Almost Ideal Demand System«. In: *The American Economic Review* 70.3, S. 312–326.

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

Fox, J. und S. Weisberg (2011). *An R Companion to Applied Regression*. Second. Thousand Oaks CA: Sage.

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

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

Hartley, H. O. (1961). »The Modified Gauss-Newton Method for the Fitting of Non-Linear Regression Functions by Least Squares«. In: *Technometrics* 3.2, pp. 269–280.

Henningsen, A. (2014). *micEconAids: Demand Analysis with the Almost Ideal Demand System (AIDS)*. R package version 0.6-16.

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

Judge, G., R. C. Hill, W. Griffiths, H. Lütkepohl und T. Lee (1988). *Introduction to the theory and practice of econometrics*. 2. ed. New York [u.a.]: Wiley. XXXVII, 1024.

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

Nielsen, H. B. und S. B. Mortensen (2012). *ucminf: General-purpose unconstrained nonlinear optimization*. R package version 1.1-3.

R Core Team (2015). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria.

StataCorp LP (2015a). »nlcom - Nonlinear combinations of estimators«. In: *Stata Base Rerference Manual Release 14*. Version 14. College Station, Texas: Stata Press.

– (2015b). »nlsur - Estimation of nonlinear systems of equations«. In: *Stata Base Rerference Manual Release 14*. Version 14. College Station, Texas: Stata Press.

Venables, W. N. und B. D. Ripley (2002). *Modern Applied Statistics with S*. Fourth. ISBN 0-387-95457-0. New York: Springer.

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

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