Estimating nonlinear equationsystems in R using **nlsur**

Jan Marvin Garbuszus*

Abstract

nlsur is an R-package to estimate Nonlinear Least Squares (NLS). Estimation is possible for single equations as well as for equation systems. In addition to NLS it is possible to estimate Feasible Generalized NLS (FGNLS) or Iterative FGNLS (IFGNLS). The latter is equivalent to a Maximum Likelihood estimation but easier to implement which makes IFGNLS a straight forward estimation procedure for many econometric models. This paper gives an short overview on the theory of nonlinear equation system estimation and provides examples of nlsur applications using common demand systems.

Introduction

Although it is possible to estimate nonlinear least squares in R using `nls()` it is only possible to estimate single equations, but not equation systems which are of crucial interesst for econometric demand analysis. While it is possible to estimate a single equation in **nlsur** the main focus is on the estimation of equation systems. Estimation of linear equation systems using *seemingly unrelated regression* [SUR; Zellner (1962)] is implemented in **systemfit** (Henningsen and Hamann 2007).[1] The **nlsur**-package extends R for the estimation of nonlinear equation systems, which implicitly follows the theory of linear equation system estimation. Nonlinear equation systems as well as FGNLS and IFGNLS are covered in Greene (2012, 345ff.). The linear pendant of Feasible GLS estimation is covered in Wooldridge (2002, 157ff.).

The packages most important command is `nlsur()`. This can be called with option `type = 1` to `3`. The first estimates a NLS, the second a FGNLS and the last an IFGNLS. The main reason for IFGNLS is that its result is comparable with those of a maximum likelihood estimation (see Greene 2012, 349). The estimation process of equation systems is comparable to single equation estimation. Because of this either equation systems or single equations can be solved with **nlsur**. In the case of single equation estimation the result is comparable to `nls()` which uses the relative offset convergence criteria (D. M. Bates and Watts 1981). In both cases a Gauss-Newton approach is used for solving.

A nonlinear equation estimation can be seen as follows, for a equation system of N observations and M equations it can be written as a stacked equation system

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

with

$$\epsilon \sim \mathcal{N}(0, \boldsymbol{\Sigma}). \tag{2}$$

*Contact:
Ruhr-Universität Bochum
Universitätsstr. 150; D-44780 Bochum
jan.garbuszus@ruhr-uni-bochum.de

[1] In **systemfit** a nlsur function using nonlinear optimization is implemented in command `nlsystemfit()`. This uses `nlm()` which is one of many different minimization algorithms implemented in R. Using this as a reference, different variations were tested to achieve estimation of IFGNLS. Unfortunately the many different algorithms lead to a wide range of results. **ucminf** (Nielsen and Mortensen 2012) using a optimization algorithm by Nielsen (2000) appeared to provide the best results. On the contrary nonlinear optimization is computationally demanding and two estimations must not – using identical starting values – lead to identical results. Not to mention the absurd amount of memory and time required.

This equation system contains K parameters that spread over the equations. Aside from that, no further restriction is set on the distribution of the parameters in the equations. They can all be in a single equation or spread over the full equation system.

Written as (1) NLS minimizes

$$SSR(\hat{\boldsymbol{\beta}}) = \sum_{i=1}^{N} \{\mathbf{y}_i - \mathbf{f}_i(\boldsymbol{\beta}, \mathbf{X})\}' \, \boldsymbol{\Sigma}^{-1} \, \{\mathbf{y}_i - \mathbf{f}_i(\boldsymbol{\beta}, \mathbf{X})\} \tag{3}$$

The $M \times M$ weighting matrix $\boldsymbol{\Sigma} = E(\epsilon_i' \epsilon_i)$ is mostly unknown. Because of this the identity matrix is used. In case of stacked equations $\boldsymbol{\Sigma}$ can be written as $\boldsymbol{\Sigma} \otimes \mathbf{I}$. Therefore in the first step NLS is an inefficient estimator, not controlling for parameters spreading over the equations. As a variant in a first step equation per equation could be solved using NLS (Greene 2012, 346). Although inefficient the estimator is consistent. From its residuals the new weighting matrix $\hat{\boldsymbol{\Sigma}}$ can be estimated as

$$\hat{\boldsymbol{\Sigma}} = \frac{1}{N} \epsilon' \epsilon \tag{4}$$

For FGNLS estimation this new weighting matrix is chosen, which creates another weighting matrix. This is selected in IFGNLS and is iteratively repeated until convergence is reached.[2]

Following Judge et al. (1988) the asymptotic co-variance matrix $\mathbf{V}$ is obtained as

$$\mathbf{V} = \left(\mathbf{J}' \hat{\boldsymbol{\Sigma}}^{-1} \mathbf{J} \right)^{-1}. \tag{5}$$

$\mathbf{J}$ is the Jacobian, the matrix of the partial deviations evaluated at the parameters.

Wooldridge (2002, 160) discusses FGLS and notes the estimation of a robust co-variance matrix following White as

$$\mathbf{V} = \left(\mathbf{J}' \hat{\boldsymbol{\Sigma}}^{-1} \mathbf{J} \right)^{-1} \left(\mathbf{J}' \hat{\boldsymbol{\Sigma}}^{-1} \hat{\epsilon} \hat{\epsilon}' \hat{\boldsymbol{\Sigma}}^{-1} \mathbf{J} \right) \left(\mathbf{J}' \hat{\boldsymbol{\Sigma}}^{-1} \mathbf{J} \right)^{-1} \tag{6}$$

For IFGNLS an additional log likelihood can be estimated (Davidson and MacKinnon 2004, 521). This is given by

$$\ln L = -\frac{MN}{2}(1 + \log 2\pi) - \frac{N}{2} \log |\hat{\boldsymbol{\Sigma}}|. \tag{7}$$

For estimation in R two different approaches are usable. One relying on a weighted GLS comparable to the implementation of `lm.gls()` in **MASS** and a second approach relying on blockwise matrix multiplication using an approach close to the one implemented in StataCorp LP (2015b).[3]

Implementation

Using spare matrices from **Matrix** (D. Bates and Maechler 2015) NLS is able to estimate NLS using bigger data sets. **Matrix** is required because the stacking of $\hat{\boldsymbol{\Sigma}}$ as $\hat{\boldsymbol{\Sigma}} \otimes \mathbf{I}$. The resulting matrix is of size $MN \times MN$. Which leads even with a small equation set and a moderate N to a huge matrix demanding much memory (e.g., a matrix of size $N = 3$ and $M = 10000$ requires 3.6 GB ram). The **Matrix** package allows creation of the Kronecker product from $\hat{\boldsymbol{\Sigma}}$ and $\mathbf{I}$ as sparse matrix. Although this variant is way more efficient, huge datasets and moderate number of equations still require a workstation with lots of ram.

[2] Usually modifications of coefficients or changes in the weighting matrix is chosen.

[3] The **MASS** estimation using weighted generalized least squares cannot handle sparse matrices, thus a memory efficient replacement was implemented.

NLS using a blockwise matrix solution is memory efficient. Starting from the idea that $\mathbf{\Sigma}^{-1}$ can be decomposed into a new matrix $\mathbf{D}$ using cholesky-decomposition such that $\mathbf{D}\mathbf{D}' = \mathbf{\Sigma}^{-1}$. Postmultiplication with $\mathbf{D}$ leads to

$$\mathbf{y}_i\mathbf{D} = \mathbf{f}_i(\boldsymbol{\beta}, \mathbf{X})\mathbf{D} + \epsilon_i\mathbf{D}, \qquad \text{for } i = 1, \dots, M \tag{8}$$

Since $E(\mathbf{D}'\mathbf{u}_i'\mathbf{u}_i\mathbf{D}) = \mathbf{I}$ all the equations can be stacked and solved columnwise

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

This can be solved – like the univariate case – with Gauss-Newton (Davidson and MacKinnon 2004, 228ff.; D. M. Bates and Watts 2008, chap. 2).

$$SSR(\boldsymbol{\beta}) = \{\mathbf{y} - \mathbf{f}(\mathbf{x}, \boldsymbol{\beta})\}' \mathbf{\Sigma}^{-1} \{\mathbf{y} - \mathbf{f}(\mathbf{x}, \boldsymbol{\beta})\} \tag{9}$$

A second order Taylor-expansion centered on β_0 gives

$$SSR(\boldsymbol{\beta}) = SSR(\boldsymbol{\beta}_0) + g'(\boldsymbol{\beta}_0)(\boldsymbol{\beta} - \boldsymbol{\beta}_0) + \frac{1}{2}(\boldsymbol{\beta} - \boldsymbol{\beta}_0)'\mathbf{H}(\boldsymbol{\beta}_0)(\boldsymbol{\beta} - \boldsymbol{\beta}_0) \tag{10}$$

with the gradient $\mathbf{g}(\cdot)$ and the Hessian $\mathbf{H}(\cdot)$. Solving for β:

$$\mathbf{g}(\boldsymbol{\beta}) = -2\mathbf{J}'\mathbf{\Sigma}^{-1}\mathbf{u} \tag{11}$$

The Hessian can be approximated with

$$\mathbf{H}(\boldsymbol{\beta}) = 2\mathbf{J}'\mathbf{\Sigma}^{-1}\mathbf{J} \tag{12}$$

Solving (10) and applying first order conditions for a minimum:

$$\mathbf{g}(\boldsymbol{\beta}_0) + \mathbf{H}(\boldsymbol{\beta}_0)(\boldsymbol{\beta} - \boldsymbol{\beta}_0) = \mathbf{0}. \tag{13}$$

This can be solved iteratively as

$$\boldsymbol{\beta}_{j+1} = \boldsymbol{\beta}_j - \alpha\mathbf{H}^{-1}(\boldsymbol{\beta}_j)\mathbf{g}(\boldsymbol{\beta}_j) \tag{14}$$

and (11) and (12) give

$$\boldsymbol{\beta}_{j+1} = \boldsymbol{\beta}_j + \alpha(\mathbf{J}'\mathbf{\Sigma}^{-1}\mathbf{J})^{-1}\mathbf{J}'\mathbf{\Sigma}^{-1}\mathbf{u} \tag{15}$$

with the stepsize parameter α, based on Box (1960) and Hartley (1961) with $\alpha \in [0, 1]$. Aside from α (15) can be estimated via weighted regression of $\mathbf{J}$ on $\mathbf{u}$. If no weighting matrix is present and $\mathbf{\Sigma} = \mathbf{I}$ no weighting is required.

Code

Depending on the selected estimation type `nlsur()` calls the function `.nlsur()` which will create an object of class `nlsur()`. While the function `.nlsur()` is called to estimate a single iteration of `nls()` it should not be called by the user directly.

NLS

Per default **nlsur** sets the vector of starting values to 0 and makes them available for evaluation. In addition LHS and RHS are evaluated. This leads to estimates of the residuals and the Jacobian. Estimation of the sum of squared residuals (SSR) requires the residuals and the cholesky decomposition of the weighting matrix. To speed this up, the calculation is done using **RcppArmadillo** (Eddelbuettel and Sanderson 2014).

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

This is exactly

$$SSR(\boldsymbol{\beta}) = \mathbf{u}'\mathbf{D}'\mathbf{D}\mathbf{u}. \tag{16}$$

Now a while-loop is started and repeated until convergence is reached. The inital stepsize parameter α is set to one. If no weighting matrix is provided for a first step NLS can be solved without a weighting matrix in form of a QR-decomposition if `qrsolve = TRUE`. Otherwise a initial weighting $\hat{\boldsymbol{\Sigma}}^{-1}$ is the identity matrix $\mathbf{I}$ where every element on the diagonal equals 1 which reduces the WLS to the default matrix equation

$$\theta = (\mathbf{J}'\mathbf{J})^{-1}\mathbf{J}'\mathbf{u} \tag{17}$$

This result is added to a new while loop that estimates a new $\boldsymbol{\beta}$ until

$$SSR(\boldsymbol{\beta}_i) > SSR(\boldsymbol{\beta}_{i-1}). \tag{18}$$

$\boldsymbol{\beta}_i$ is calculated using (15) as

$$\boldsymbol{\beta}_{i+1} = \boldsymbol{\beta}_i + \alpha\boldsymbol{\theta}_i. \tag{19}$$

Startvalues for this are $\boldsymbol{\beta}$, $\alpha = 1$ and $\boldsymbol{\theta}_i$ the previous calculated regression coefficients. The newly estimated $\boldsymbol{\beta}$-values are used to calculate a new SSR. If convergence is not reached, α will become $\alpha/2$ and new $\boldsymbol{\beta}$-candidates are estimated.

If this loop is stopped a check for convergence is done. Convergence is reached if both criteria are met:

1. $|SSR_{i-1} - SSR| \leq \epsilon(SSR_{i-1} + \tau)$ This is a minor deviation from Gallant (1987, 29) who suggests $<$ instead.
2. The second is $m = 1, \ldots, k$ the convergence criteria is $\alpha|\boldsymbol{\theta}_{jm}| \leq \epsilon(|\boldsymbol{\beta}_{j-1,m}| + \tau)$. While $\boldsymbol{\theta}_j$ is the result of the weighted regression.

If no convergence is reached, the last SSR is the new convergence criteria. The stepsize parameter is doubled. The dublication can continue until a maximum value of 1 is reached. This is identical to the implementation of `nls()`.

Once this first estimation is done the result is comparable to the results of `nls()`. To get `nls()` results matching to the Stata command `nlsur()` a second evaluation is done using the diagonal of the last weighting matrix $\boldsymbol{\Sigma}$ which replaces the diagonal values of $\mathbf{I}$. In **nlsur** this can be achieved using the option `stata = TRUE`. This option only has influence on the results of the NLS estimation.

FGNLS

If FGNLS is estimated the β coefficients of the NLS stage and the weighting matrix $\mathbf{\Sigma}$ are selected as start values for the NLS estimation. The most important difference is that the estimation can no longer rely only on GLS but requires a WLS estimation. This can be calculated using QR-decomposition and sparse matrices or blockwise using a matrix algorithm suggested by StataCorp LP (2015b). As said before the `lm.gls()` function of **MASS** cannot be used since it calculates a singular value decomposition of the weight matrix and even if the latter can be a sparse matrix, during calculation of the eigen values a dense matrix is returned.

Fortunately this can be overcome using the smaller $\hat{\mathbf{\Sigma}}$ returned by `nlsur()` and estimation of the eigen values before calculating the Kronecker product. This requires the option `MASS = TRUE`.

```
lm_gls <- function(X, Y, W, neqs, tol = 1e-7, covb = FALSE) {

  eW <- eigen(W, TRUE)
  d <- eW$values
  if (any(d <= 0))
    stop("'W' is not positive definite")

  A <- diag(d^-0.5,
            nrow = length(d),
            ncol = length(d)) %*% t(eW$vectors)

  n <- nrow(X)/neqs

  A <- Matrix::kronecker(X = A,
                         Y = Matrix::Diagonal(n))

  X <- as(X, "sparseMatrix"); Y <- as(Y, "sparseMatrix")

  if (covb)
    fit <- Matrix::crossprod(A %*% X)

  if (!covb)
    fit <- qr.coef(qr(A %*% X, tol = tol), A %*% Y)

  fit
}
```

This is the numerically stable option. In a blockwise matrix algebra approach the same can be solved as

```
SEXP calc_reg (arma::Mat<double> x, arma::Mat<double> r, arma::Mat<double> qS,
               arma::Col<double> w, int sizetheta, bool fullreg, double tol) {

  arma::Mat<double> XDX(sizetheta, sizetheta, fill::zeros);
  arma::Mat<double> XDy(sizetheta, 1, fill::zeros);

  Function Rf_qr("qr");
  Function Rf_qrcoef("qr.coef");

  int n = r.n_rows;
  int k = r.n_cols;

  for (int i = 0; i < n; ++i) {

    arma::Mat<double> XI = arma_reshape(x.row(i), k);
```

```
    XDX += w(i) * XI.t() * qS * XI;

    if (fullreg) {
      arma::Mat<double> YI = r.row(i).t();
      XDy += w(i) * XI.t() * qS * YI;
    }

  }

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

  if (fullreg) /* weighted regression */
    return Rf_qrcoef(Rf_qr(XDX, _["tol"] = tol), XDy);
  else         /* covb */
    return wrap(XDX);
}
```

This is the memoryefficient default estimation option implemented for speed in **RcppArmadillo**.

IFGNLS

At the initial stage for IFGNLS a FGNLS is estimated. Based on the results of the FGNSL estimates a new while-loop using its own convergence criterium is started. Again another NLS estimation is started based on the last coefficients and the weighting matrix.

The estimated residuals and the cholesky-decomposition of the inverse of the weighting matrix are used to calculate a new SSR. Based on the relative change of the weighting matrix and the coefficients two criteria are needed. For the coefficients the change should be smaller than ϵ and for $\mathbf{\Sigma}$ it is 10^{-10}. Once both criteria are met convergence is declared and the log likelihood is estimated.

Following I/FG/NLS

Following the estimation a number of different results may be obtained. Calling print on the nlsur-object returns the coefficients as well as `coef()`. A summary is returned by `summary()`. In addition this command tries to evaluate weather or not a equation contains a constant or not. The variance co-variance matrix is returned by `cvov()` which works for `nlsur` just like `residuals()`, `deviance()`, `df.residuals()`, `fitted()`, `logLik()`, `convint()` or `predict()`.

Application

nlsur vs nls

`nlsur` can be used as well as `nls` for the estimation of nonlinear least squares. Differences are in the implementation. `nls` uses the relative offset criteria by D. M. Bates and Watts (1981) this was not implemented for `nlsur` because it would require weighting of $\mathbf{J}$ in (15). Therefore `nlsur` uses the same convergence criteria StataCorp LP (2015b) uses. For a simple linear model estimation using `lm`, `nls` and `nlsur` returns the same coefficients

```
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(res2)
```

```
## NLSUR Object of type: NLS
##
##        n k  RMSE   MAE R-squared Adj-R-sqr. Const
## 1 mpg 32 3 2.912 2.309     0.759     0.7424 beta0
##
## Coefficients:
##       Estimate Std. Error t value Pr(>|t|)
## beta0  34.5224     2.6032  13.262 7.69e-14 ***
## beta1  -2.5010     0.3608  -6.931 1.28e-07 ***
## beta2   2.5670     1.2914   1.988   0.0564 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

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

`nlsur` is not as strict as `nls` is concerning artificial data and finds solutions in models containing constant variables.

```
# estimate the same model with a constant variable
mtcars <- subset(mtcars, am == 1)

res1 <- res2 <- res3 <- NULL

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

```
## Error in nlsModel(formula, mf, start, wts, scaleOffset = scOff, nDcentral = nDcntr) :
##   singular gradient matrix at initial parameter estimates
```

```
# nls does not find a solution

summary(res2)
```

```
## NLSUR Object of type: NLS
##
##        n k RMSE   MAE R-squared Adj-R-sqr. Const
## 1 mpg 13 2 3.34 2.553    0.6823     0.6534 beta0
##
## Coefficients:
##       Estimate Std. Error t value Pr(>|t|)
## beta0  41.0489     3.5720   11.49 1.81e-07 ***
## beta1  -3.2809     0.6751   -4.86 0.000503 ***
## beta2       NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

equation systems I: Translog

A well known application for FGNLS is *example 10.3 a cost function for U.S. Manufacturing* in Greene (2012, 353f.). A Translog is estimated for four goods. Model and data are from Berndt and Wood (1975). For estimation purposes a single equation is dropped and the parameters of the dropped equation are evaluated after the estimation.

$$\begin{aligned} s_k &= \beta_k + \delta_{kk} \ln\left(\frac{p_k}{p_m}\right) + \delta_{kl} \ln\left(\frac{p_l}{p_m}\right) + \delta_{ke} \ln\left(\frac{p_e}{p_m}\right) \\ s_l &= \beta_l + \delta_{kl} \ln\left(\frac{p_k}{p_m}\right) + \delta_{ll} \ln\left(\frac{p_l}{p_m}\right) + \delta_{le} \ln\left(\frac{p_e}{p_m}\right) \\ s_e &= \beta_e + \delta_{ke} \ln\left(\frac{p_k}{p_m}\right) + \delta_{le} \ln\left(\frac{p_l}{p_m}\right) + \delta_{ee} \ln\left(\frac{p_e}{p_m}\right) \end{aligned} \tag{20}$$

with

$$\sum_{i=1}^{M} \beta_i = 1; \sum_{i=1}^{M} \delta_{ij} = \sum_{j=1}^{M} \delta_{ij} = 0 \tag{21}$$

```
data( "costs" )

dd <- costs
# apply a patch to create Greenes Ed. 7 Data
dd$Sm[dd$Year == 1958] <- 0.61886
dd$Pe[dd$Year == 1950] <- 1.12442
dd$Pm[dd$Year == 1949] <- 1.06625

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)
```

The missing parameters are estimated using `nlcom()` using the parameter restrictions of (20).

```
# 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")
```

Once all parameters of the Translog are estimated, they can be combined. The full coefficients of the translog model above are:

```
##           Estimate     Std. Error   z value   Pr(>|z|)
## be   4.3832896e-02  1.0510857e-03  41.70249 < 2.22e-16 ***
## bk   5.6823973e-02  1.3110057e-03  43.34380 < 2.22e-16 ***
## bl   2.5354586e-01  1.9896655e-03 127.43140 < 2.22e-16 ***
## bm   6.4579727e-01  3.0087289e-03 214.64123 < 2.22e-16 ***
## dee  2.9382817e-02  7.4064843e-03   3.96717 7.8512e-05 ***
## dem -1.7967300e-02  1.0760329e-02  -1.66977   0.095312 .
## dke -8.2033738e-03  4.0608435e-03  -2.02012   0.043667 *
```

```
## dkk  2.9870626e-02  5.7503473e-03   5.19458 2.5386e-07 ***
## dkl  2.2255039e-05  3.6891634e-03   0.00603   0.995188
## dkm -2.1689507e-02  9.6273895e-03  -2.25290   0.024506 *
## dle -3.2121434e-03  2.7549370e-03  -1.16596   0.243940
## dll  7.4877271e-02  6.4082308e-03  11.68455 < 2.22e-16 ***
## dlm -7.1687383e-02  9.4633929e-03  -7.57523 8.8818e-14 ***
## dmm  1.1134419e-01  2.2457966e-02   4.95789 8.5154e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This application uses `nlcom()` a command to estimate nonlinear combinations from objects of class nlsur. Mainly `nlcom()` is a wrapper around the delta method function `dm()` and mimics the `nlcom()` command of StataCorp LP (2015a). The command can be used to estimate parameters that are combinations of other nlsur based parameters and select other nlcom parameters out of the same environment. In the case above this is used to estimate standard errors and confidence intervals for the parameter restrictions of the Translog.

equation systems II: AI-system

A second example is given by the application of the Almost-Ideal Demand System (AI-system) by Deaton and Muellbauer (1980). The AI-system is given by

$$w_i = \alpha_i + \sum_j \gamma_{ij} \log p_j + \beta_i \log\{x/P\} \tag{22}$$

$$\tag{23}$$

with:

$$\log P = \alpha_0 + \sum_k \alpha_k \log p_k + \frac{1}{2} \sum_j \sum_k \gamma_{kj} \log p_k \log p_j \tag{24}$$

$$\sum_{i=1}^{n} \alpha_i = 1 \qquad \sum_{i=1}^{n} \gamma_{ij} = 0 \qquad \sum_j \gamma_{ij} = 0 \qquad \gamma_{ij} = \gamma_{ji} \tag{25}$$

Deaton and Muellbauer (1980) suggest aside the translog price index (as in (23)) a modification using the Stone-price index $\log P = \sum w_k * \log p_k$ as a complete linearized AI-system (LA-AI). Such a model is implemented in the **micEconAids**-package by Henningsen (2014). The estimation of the AI-system is based on a data set by Blanciforti, Green, and King (1986) that is part of **micEconAids**. The model estimated is based on four food commodities. As shown before missing parameters of the AI-system are estimated using `nlcom()`. Final results are compared to the results of an LA-AI-system estimated with SUR using Henningsen (2014). The equation of the AI-system are created using `ai.model()`. This is an example function of a demand system model builder taking care of the restrictions of such models. This precise function is able to create AI-system equations using either the translog or the Stone price index, demographicaly scaled versions of this demand system as well as Quadratic Almost-Ideal Demand System variations of such models. Once such an implementation is complete it is straightforward to wrap the equation in a function such as `ai()` or `qai()`.

```
library( "micEconAids" )

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

# define parameters
bgs <- c( "wFood1", "wFood2", "wFood3", "wFood4" )
pid <- c( "pFood1", "pFood2", "pFood3", "pFood4" )
```

```r
exp <- "xFood"

# estimates using aidsEst
estResult <- aidsEst( priceNames = pid, shareNames = bgs, totExpName = exp,
                      data = Blanciforti86, priceIndex = "S",
                      method = "LA", estMethod = "SUR")

# build the AI-system equation
model <- ai.model(w = bgs, p = pid, exp = exp, priceindex = "S",
                  logp = FALSE, logexp = FALSE)

estNlsur <- nlsur(eqns = model, data = Blanciforti86, MASS = TRUE,
                  type = "FGNLS", qrsolve = FALSE)

# nlsur
summary(estNlsur)
```

```
## NLSUR Object of type: FGNLS
##
##           n k     RMSE      MAE R-squared Adj-R-sqr. Const
## 1 wFood1 32 5 0.008706 0.007134    0.6271     0.5719   a01
## 2 wFood2 32 5 0.006668 0.005271    0.8315     0.8066   a02
## 3 wFood3 32 5 0.003602 0.003063    0.4115     0.3243   a03
##
## Coefficients:
##        Estimate Std. Error z value Pr(>|z|)
## a01   -0.247298   0.064373  -3.842 0.000128 ***
## a02    0.109249   0.053162   2.055 0.040055 *
## a03    0.268238   0.031611   8.486  < 2e-16 ***
## b01    0.323989   0.037699   8.594  < 2e-16 ***
## b02    0.055863   0.030863   1.810 0.070498 .
## b03   -0.078626   0.018395  -4.274 2.04e-05 ***
## g0101  0.104150   0.019110   5.450 5.92e-08 ***
## g0102 -0.139880   0.014057  -9.951  < 2e-16 ***
## g0103 -0.011562   0.009017  -1.282 0.199980
## g0202  0.156909   0.026302   5.966 3.06e-09 ***
## g0203  0.003473   0.016123   0.215 0.829497
## g0303  0.012490   0.014919   0.837 0.402627
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
### caclulate missing parameters via AI-system restrictions

# sum a_k = 1
a04 <- nlcom(object = estNlsur,
             form = "1- (a01 + a02 + a03)", rname = "a04")

# sum b_k = 0
b04 <- nlcom(object = estNlsur,
             form = " - (b01 + b02 + b03)", rname = "b04")

# sum g_ij = 0
g0104 <- nlcom(object = estNlsur,
               form = " (-g0101 - g0102 - g0103)", rname = "g0104")
```

```
g0204 <- nlcom(object = estNlsur,
               form = " (-g0102 - g0202 - g0203)", rname = "g0204")
g0304 <- nlcom(object = estNlsur,
               form = " (-g0103 - g0203 - g0303)", rname = "g0304")
g0404 <- nlcom(object = estNlsur,
               form = " (-g0104 - g0204 - g0304)", rname = "g0404")

# g_ij = g_ji
g0201 <- nlcom(object = estNlsur, form = "g0102", rname = "g0201")
g0301 <- nlcom(object = estNlsur, form = "g0103", rname = "g0301")
g0302 <- nlcom(object = estNlsur, form = "g0203", rname = "g0302")
g0401 <- nlcom(object = estNlsur, form = "g0104", rname = "g0401")
g0402 <- nlcom(object = estNlsur, form = "g0204", rname = "g0402")
g0403 <- nlcom(object = estNlsur, form = "g0304", rname = "g0403")
```

Once all the parameters of the restricted model using `nlsur()` and the missing ones using `nlcom()` are obtained it is possible to directly compare `nlsur()` and `aidsEst()` results. Therefore the parameters and their differences are shown:

```
##              nlsur      micEcon          diff
## a01   -0.247298293 -0.247298293 -5.520659e-11
## a02    0.109249097  0.109249097 -6.920380e-11
## a03    0.268238436  0.268238436  2.996249e-10
## a04    0.869810760  0.869810761 -1.752145e-10
## b01    0.323989176  0.323989176  3.308370e-11
## b02    0.055863165  0.055863164  4.009205e-11
## b03   -0.078626097 -0.078626097 -1.781309e-10
## b04   -0.301226244 -0.301226244  1.049552e-10
## g0101  0.104150200  0.104150200  1.429327e-11
## g0102 -0.139880152 -0.139880152  4.274692e-12
## g0103 -0.011561838 -0.011561838 -7.425008e-11
## g0104  0.047291790  0.047291789  5.568211e-11
## g0201 -0.139880152 -0.139880152  4.275857e-12
## g0202  0.156908650  0.156908650 -1.429296e-11
## g0203  0.003472719  0.003472719 -1.754556e-11
## g0204 -0.020501216 -0.020501216  2.756265e-11
## g0301 -0.011561838 -0.011561838 -7.424749e-11
## g0302  0.003472719  0.003472719 -1.754556e-11
## g0303  0.012489861  0.012489861  3.354735e-11
## g0304 -0.004400742 -0.004400742  5.824569e-11
## g0401  0.047291790  0.047291789  5.567725e-11
## g0402 -0.020501216 -0.020501216  2.756354e-11
## g0403 -0.004400742 -0.004400742  5.824829e-11
## g0404 -0.022389831 -0.022389831 -1.414891e-10
```

equation systems III: QAI-system

In addition to the estimation of the AI-system Poi (2012) proposed a function to estimate the Quadratic Almost-Ideal Demand System (QAI-system) by Banks, Blundell, and Lewbel (1997) using demographic scaling as proposed by Ray (1983). Estimation of such a demographicaly scaled model is possible with **nlsur** as well. Either it is possible to create the equation system using the `ai.model()` builder as shown in the previous section or simply by using the `qai()` function.[4] This function is a wrapper around the model builder

[4]For the Almost-Ideal Demand System using the translog price index there exists a matching function `ai()`. Both incorporate the same scaling functions.

and `nlsur()` and provides an easy way to estimate an QAI-system using the translog prices index. For now only the directly estimated parameters are returned although it is possible to estimate the missing parameters afterwards.

```
# dataset as proposed in Poi 2012. Stata results for comparisson available at
# http://www.stata-journal.com/article.html?article=st0268

w <- c("w1", "w2", "w3", "w4"); p <- c( "p1", "p2", "p3", "p4")
x <- "expfd"; z <- c("nkids", "rural")

# using a slightly different starting value
res <- qai(w = w, p = p, x = x, z = z, a0 = 10,
           data = dat, scale = TRUE,
           logp = FALSE, logexp = FALSE,
           MASS = TRUE, val = 0.001)

summary(res)
```

```
## NLSUR Object of type: IFGNLS
##
##         n  k    RMSE     MAE R-squared Adj-R-sqr. Const
## 1 w1 4048 21 0.13318 0.10519   0.11578    0.11139   a01
## 2 w2 4048 21 0.10236 0.07898   0.07608    0.07149   a02
## 3 w3 4048 21 0.05376 0.04069   0.14165    0.13739   a03
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## a01           9.105e-01  6.960e-02  13.082  < 2e-16 ***
## a02          -1.329e-01  7.071e-02  -1.879 0.060191 .
## a03           1.555e-02  4.199e-02   0.370 0.711213
## b01           2.130e-01  2.948e-02   7.226 4.98e-13 ***
## b02          -1.323e-01  2.771e-02  -4.775 1.80e-06 ***
## b03          -3.474e-02  1.561e-02  -2.225 0.026052 *
## eta_nkids_01  2.252e-04  3.160e-04   0.713 0.475977
## eta_nkids_02 -4.756e-04  3.542e-04  -1.343 0.179311
## eta_nkids_03  1.648e-05  1.179e-04   0.140 0.888874
## eta_rural_01 -1.050e-03  7.044e-04  -1.490 0.136222
## eta_rural_02  6.118e-04  8.155e-04   0.750 0.453121
## eta_rural_03  2.422e-05  2.797e-04   0.087 0.930989
## g0101         2.800e-01  4.650e-02   6.021 1.73e-09 ***
## g0102        -1.508e-01  3.053e-02  -4.940 7.83e-07 ***
## g0103        -6.074e-02  1.335e-02  -4.551 5.33e-06 ***
## g0202         1.263e-01  2.636e-02   4.791 1.66e-06 ***
## g0203         1.459e-02  7.547e-03   1.934 0.053133 .
## g0303         4.658e-02  4.111e-03  11.333  < 2e-16 ***
## l01           1.794e-02  2.948e-03   6.085 1.17e-09 ***
## l02          -9.753e-03  2.617e-03  -3.727 0.000193 ***
## l03          -3.288e-03  1.416e-03  -2.321 0.020277 *
## rho_nkids    -1.318e-01  4.653e-02  -2.833 0.004604 **
## rho_rural     2.314e-01  1.909e-01   1.212 0.225446
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## Log-Likelihood: 13098.97
```

Application: Weighting

Although weighting is possible in `nlsur()`, it is only available using the blockwise matrix variation of the nlsur solution.

```
# weighting
set.seed(123)

dd <- data.frame(y = rnorm(n = 100, mean = 5, sd = 5),
                 x = rnorm(n = 100, mean = 2, sd = 5),
                 w = sample(x = seq(0,1,0.1), size = 100, replace = TRUE))


model <- "y ~ b * x"
res_nlsur <- nlsur(eqns = model, data = dd, weights = w)
```

```
##     lm    nls  nlsur
## 0.2066 0.2066 0.2066
```

Application: QAI/AI elasticities

Using `elasticities()` one can has postestimation options after `ai()` or `qai()`.

Formula for the expenditure (income) elasticity

$$\mu_i = 1 + \frac{1}{w_i}\left[\beta_i + \frac{2\lambda_i}{b(\mathbf{p})} * ln\left\{\frac{m}{a(\mathbf{p})}\right\}\right] \tag{26}$$

Formula for the uncompensated price elasticity

$$\epsilon_{ij} = \delta_{ij} + \frac{1}{w_i}\left(\gamma_{ij} - \beta_i + \frac{2\lambda_i}{b(\mathbf{p})}\right)\left[\ln\left\{\frac{m}{a(\mathbf{p})}\right\}\right] \times \tag{27}$$

$$\left(\alpha_j + \sum_k \gamma_{jk} \ln p_k\right) - \frac{\beta_j \lambda_i}{b(\mathbf{p})}\left[\ln\left\{\frac{m}{a(\mathbf{p})}\right\}\right] \tag{28}$$

Compensated price elasticitys (Slutsky equation)

$$\epsilon_{ij}^C = \epsilon_{ij} + \mu_i w_j \tag{29}$$

Conclusion

With its ability to estimate demand systems **nlsur** closes a gap considering the estimation of equation systems in R. Not only **nlsur** enable the estimation of feasible NLS moreover does it extend the focus to the estimation of parametric equation systems. Whilst the estimation of SUR is implemented quite some time now, **nlsur** allows the the estimation of nonlinear SUR and thus increases the application of R to allow the estimation of recent demand systems. As shown `nlsur()` allows the estimation of many flexible demand systems along the lines of the previous demonstrated Translog or variants of the AI-system. Additional models are easily to implement as shown with the `ai.model()` equation builder. These models often come with restrictions to its parameters. In such cases `nlcom()` can be used to estimate the restricted parameters. Examples of these were given.

References

Banks, James, Richard Blundell, and Arthur Lewbel. 1997. "Quadratic Engel Curves and Consumer Demand." *The Review of Economics and Statistics* 79 (4): 527–39. http://www.jstor.org/stable/2951405.

Bates, Douglas M., and Donald G. Watts. 1981. "A Relative Offset Orthogonality Convergence Criterion for Nonlinear Least Squares." *Technometrics* 23 (2).

———. 2008. *Nonlinear Regression Analysis and Its Applications*. New York: John Wiley & Sons, Inc. https://doi.org/10.1002/9780470316757.ch2.

Bates, Douglas, and Martin Maechler. 2015. *Matrix: Sparse and Dense Matrix Classes and Methods*. http://CRAN.R-project.org/package=Matrix.

Berndt, Ernst R, and David O Wood. 1975. "Technology, prices, and the derived demand for energy." *The Review of Economics and Statistics*, 259–68. http://www.jstor.org/stable/1923910.

Blanciforti, Laura, Richard D. Green, and 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. http://s.giannini.ucop.edu/uploads/giannini_public/dd/ca/ddca8064-0a64-46e7-bd3a-0720626071bd/40-postwar.pdf.

Box, George E. P. 1960. "Fitting Empirical Data." *Annals of the New York Academy of Sciences* 86 (3): 792–816. https://doi.org/10.1111/j.1749-6632.1960.tb42843.x.

Davidson, Russel, and James G. MacKinnon. 2004. "Nonlinear Regression." In *Econometric Theory and Methods*, 213–56. New York: Oxford University Press.

Deaton, Angus, and John Muellbauer. 1980. "An Almost Ideal Demand System." *The American Economic Review* 70 (3): 312–26. http://www.jstor.org/stable/1805222.

Eddelbuettel, Dirk, and Conrad Sanderson. 2014. "RcppArmadillo: Accelerating R with high-performance C++ linear algebra." *Computational Statistics and Data Analysis* 71: 1054–63. http://dx.doi.org/10.1016/j.csda.2013.02.005.

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.

Hartley, H. O. 1961. "The Modified Gauss-Newton Method for the Fitting of Non-Linear Regression Functions by Least Squares." *Technometrics* 3 (2): pp. 269–280. http://www.jstor.org/stable/1266117.

Henningsen, Arne. 2014. *micEconAids: Demand Analysis with the Almost Ideal Demand System (AIDS)*. https://CRAN.R-project.org/package=micEconAids.

Henningsen, Arne, and Jeff D. Hamann. 2007. "systemfit: A Package for Estimating Systems of Simultaneous Equations in R." *Journal of Statistical Software* 23 (4): 1–40. http://www.jstatsoft.org/v23/i04/.

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

Nielsen, Hans Bruun. 2000. "UCMINF – An Algorithm For Unconstrained, Nonlinear Optimization." Report IMM-REP-2000-19. IMM. Lyngby: Technical University of Denmark; Department Of Mathematical Modelling. http://www.imm.dtu.dk/documents/ftp/tr00/tr19_00.pdf.

Nielsen, Hans Bruun, and Stig Bousgaard Mortensen. 2012. *ucminf: General-purpose unconstrained non-linear optimization*. https://CRAN.R-project.org/package=ucminf.

Poi, Brian P. 2012. "Easy demand-system estimation with quaids." *Stata Journal* 12 (3): 433–446(14). http://www.stata-journal.com/article.html?article=st0268.

Ray, Ranjan. 1983. "Measuring the costs of children: An alternative approach." *Journal of Public Economics* 22 (1): 89–102. http://www.sciencedirect.com/science/article/pii/0047272783900580.

StataCorp LP. 2015a. "nlcom - Nonlinear combinations of estimators." In *Stata Base Rerference Manual Release 14*, Version 14. College Station, Texas: Stata Press. http://www.stata.com/manuals13/rnlsur.pdf.

———. 2015b. "nlsur - Estimation of nonlinear systems of equations." In *Stata Base Rerference Manual Release 14*, Version 14. College Station, Texas: Stata Press. http://www.stata.com/manuals13/rnlsur.pdf.

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." *Journal of the American Statistical Association* 57 (298): 348–68. http://www.jstor.org/stable/2281644.