Statistical Computing in R: Strategies for Turning Ideas into Software

Achim Zeileis

http://statmath.wu.ac.at/~zeileis/

Overview

Computational statistics: Methods requiring substantial computation.

Statistical computing: Translating statistical ideas into software.

- Why:
  - Why should we write software (*and make it available*)?
  - Why should it be open-source software?
- How:
  - What should be the guiding principles for implementation?
  - Linear regression in base R.
  - Robust sandwich covariances in package **sandwich**.
  - Hurdle models for count regression in package **pscl**.

Why software?

Authors of statistical methodology usually have an implementation for own applications and running simulations and benchmarks, *but not necessarily in production quality*.

Why should they be interested in taking the extra effort to adapt them to more general situations, document it and make it available to others?

Supplying software that is sufficiently easy to use is an excellent way of *communicating ideas and concepts* to researchers and practitioners.

Given the description of an excellent method and code for a good one, you choose . . . ?

Why open source?

Claerbout's principle

> *An article about computational science in a scientific publication is not the scholarship itself, it is merely advertising of the scholarship. The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures.*

To evaluate the correctness of all the results in such an article, the source code must also be available for inspection. Only this way gradual refinement of computational (and conceptual) tools is possible.

Implementation principles

Task: Turn conceptual tools into computational tools

Goals: desirable features

- easy to use,
- numerically reliable,
- computationally efficient,
- flexible and extensible,
- reusable components,
- object oriented,
- reflect features of the conceptual method.

Problem: often antagonistic, e.g., computational efficiency vs. extensibility.

Implementation principles

Guiding principle: The implementation should be guided by the properties of the underlying methods while trying to ensure as much efficiency and accuracy as possible.

The resulting functions should do what we think a method does conceptually.

In practice: Many implementations are still guided by the limitations that programming languages used to have (and some still have) where everything has to be represented by numeric vectors and matrices.

What language features are helpful for improving this?

Implementation principles

Object orientation: Create (potentially complex) objects that represent an abstraction of a procedure or type of data. Methods performing typical tasks can be implemented.

Functions as first-class objects: Functions are a basic data type that can be passed to and returned by another function.

Lexical scope: more precisely *nested lexically scoped functions*. Returned functions can have free variables stored in function closure.

Compiled code: Combine convenience of interpreted code and efficiency of compiled code by (byte) compilation or dynamic linking.

Reusable components: Programming environment should provide tools that implementations can build on. Likewise, implementations should create objects that can be reused in other programs.

How can this be used in practice?

Example: Linear regression in R.

- **Object orientation:** `lm()` returns an "`lm`" object with suitable methods and extractor functions.
- **Reusable components:** Underlying workhorse `lm.fit()` without pre- and postprocessing is also provided.
- **Compiled code:** At its core `lm.fit()` has a `.Fortran("dqrls", ...)` call.

Application: Time series regression (Greene 1993).

```
R> library("AER")
R> data("Investment", package = "sandwich")
R> fm <- lm(RealInv ~ RealGNP + RealInt, data = Investment)
```

Object orientation

```
R> coef(fm)

(Intercept)     RealGNP     RealInt
    -12.534       0.169      -1.001

R> vcov(fm)

            (Intercept)   RealGNP RealInt
(Intercept)     620.771 -0.503830  8.4748
RealGNP          -0.504  0.000423 -0.0115
RealInt           8.475 -0.011457  5.6110

R> logLik(fm)

'log Lik.' -79.4 (df=4)
```

Object orientation

`print()`	simple printed display with coefficients
`summary()`	standard regression summary; returns "`summary.`*class*" object (with `print()` method)
`plot()`	diagnostic plots
`coef()`	extract coefficients
`vcov()`	associated covariance matrix
`predict()`	(different types of) predictions for new data
`fitted()`	fitted values for observed data
`residuals()`	extract (different types of) residuals
`terms()`	extract terms
`model.matrix()`	extract model matrix (or matrices)
`df.residual()`	extract residual degrees of freedom
`logLik()`	extract fitted log-likelihood

Reusable components

Provide: Important building blocks, e.g., `lm.fit()` (so that users *never call*: `solve(t(X) %*% X) %*% t(X) %*% y`).

Reuse: Exploit available tools, e.g., "smart" generics can rely on suitable methods such as `coef()`, `vcov()`, `logLik()`, etc.

`confint()`	confidence intervals
`AIC()`	information criteria (AIC, BIC, ...)
`coeftest()`	partial Wald tests of coefficients (**lmtest**)
`waldtest()`	Wald tests of nested models (**lmtest**)
`linear.hypothesis()`	Wald tests of linear hypotheses (**car**)
`lrtest()`	likelihood ratio tests of nested models (**lmtest**)

Reusable components

```
R> coeftest(fm)

t test of coefficients:

             Estimate Std. Error t value Pr(>|t|)
(Intercept) -12.5336    24.9153   -0.50     0.62
RealGNP       0.1691     0.0206    8.22  3.9e-07
RealInt      -1.0014     2.3687   -0.42     0.68


R> confint(fm)

              2.5 % 97.5 %
(Intercept) -65.352 40.284
RealGNP       0.126  0.213
RealInt      -6.023  4.020
```

Lexical scope

Return nested lexically scoped function for $f(x) = \hat{\alpha} + \hat{\beta} \cdot x$:

```
R> predict_fun <- function(x, y) {
+     cf <- lm.fit(cbind(1, x), y)$coefficients
+     return(function(x) cf[1] + cf[2] * x)
+ }
```

Set up and evaluate prediction function:

```
R> inv <- as.data.frame(Investment)
R> predict_invest <- predict_fun(inv$RealGNP, inv$RealInv)
R> predict_invest
```

```
function(x) cf[1] + cf[2] * x
<environment: 0x965dd68>
```

```
R> predict_invest(1500)
```

```
240
```

Sandwich covariances: Ideas

Inference for models estimated by estimating equations

$$\sum_{i=1}^{n} \psi(y_i, x_i, \hat{\theta}) \quad = \quad 0$$

(including maximum likelihood and least squares estimators) is typically based on a central limit theorem

$$\sqrt{n}\,(\hat{\theta} - \theta) \quad \xrightarrow{\text{d}} \quad \mathcal{N}(0, S(\theta)),$$

where the covariance matrix is of a sandwich form:

$$\begin{aligned} \text{sandwich:} \quad S(\theta) &= B(\theta)\, M(\theta)\, B(\theta), \\ \text{bread:} \quad B(\theta) &= \mathrm{E}[-\psi'(y, x, \theta)]^{-1}, \\ \text{meat:} \quad M(\theta) &= \mathrm{VAR}[\psi(y, x, \theta)]. \end{aligned}$$

Sandwich covariances: Ideas

Correctly specified likelihood: $M(\theta) = B(\theta)^{-1}$ is Fisher information matrix $\Rightarrow$ covariance can be estimated by bread $\hat{B}$.

Misspecification: If $\psi(y, x, \theta)$ is correct, but not the remaining likelihood $\Rightarrow$ full sandwich estimate is more robust.

Meat estimator: Weighted cross-products of estimating functions.

$$\hat{M} \quad = \quad n^{-1} \sum_{i,j=1}^{n} w_{|i-j|} \, \psi(y_i, x_i, \hat{\theta}) \, \psi(y_j, x_j, \hat{\theta})^\top$$

- HC for cross section data: $w_0 = 1$, $w_i = 0$ $(i > 0)$,
- HAC for time series data: w_k chosen by kernel function (plus bandwidth selection).

Sandwich covariances: Software

Translation to R: sandwich provides functions similar to `vcov()`.

```
sandwich(obj)
  vcovHC(obj, ...)
 vcovHAC(obj, weights, ...)
 vcovOPG(obj)
```

where

- `obj`: Arbitrary fitted object with
  - `estfun()` method: extract $\psi(y_i, x_i, \hat{\theta})$ ($i = 1, \ldots, n$).
  - `bread()` method: extract $\hat{B}$.
- `weights:` Specification of weights via
  - numeric vector, or
  - function for data-driven computation of weights and bandwidth.

Sandwich covariances: Software

This implementation uses

- **Object orientation**: Different models can be plugged in via `estfun()` and `bread()` methods.
- **Functions as first-class objects**: Weight/bandwidth selection can be passed to `vcovHAC()` as functions.
- **Lexical scope**: Selected bandwidth only needs to be defined locally.
- **Reusable components**: Takes fitted models, provides new covariances (to be reused in inference).

Sandwich covariances: Illustration

Reuse in partial Wald tests:

```
R> vcovHAC(fm)

             (Intercept)   RealGNP RealInt
(Intercept)      615.599 -0.567954  9.2417
RealGNP           -0.568  0.000545 -0.0222
RealInt            9.242 -0.022224 14.5397


R> coeftest(fm, vcov = vcovHAC(fm))

t test of coefficients:

             Estimate Std. Error t value Pr(>|t|)
(Intercept) -12.5336    24.8113   -0.51     0.62
RealGNP       0.1691     0.0233    7.25  2.0e-06
RealInt      -1.0014     3.8131   -0.26     0.80
```

Sandwich covariances: Illustration

Or even simpler:

```
R> vcovHAC(fm)

            (Intercept)   RealGNP RealInt
(Intercept)     615.599 -0.567954  9.2417
RealGNP          -0.568  0.000545 -0.0222
RealInt           9.242 -0.022224 14.5397


R> coeftest(fm, vcov = vcovHAC)

t test of coefficients:

            Estimate Std. Error t value Pr(>|t|)
(Intercept) -12.5336    24.8113   -0.51     0.62
RealGNP       0.1691     0.0233    7.25  2.0e-06
RealInt      -1.0014     3.8131   -0.26     0.80
```

Count regression: Hurdle models

Task: Hurdle count regression (two-component model with truncated count component and zero hurdle component, estimated via ML).

Implementation: `hurdle()` in package **pscl** with methods for `coef()`, `vcov()`, ..., `estfun()`, `bread()`.

Application: Demand for medical care count data regression (replication from Deb & Trivedi 1997).

```
R> data("NMES1988", package = "AER")
R> nmes <- NMES1988[, -(2:6)]
R> hm <- hurdle(visits ~ ., data = nmes, dist = "negbin")
```

Count regression: Hurdle models

```
R> coeftest(hm, vcov = sandwich, df = Inf)
```

	Estimate	Std. Error	z value	Pr(>|z|)
count_(Intercept)	1.63098	0.27105	6.02	1.8e-09
count_healthpoor	0.33251	0.05671	5.86	4.5e-09
count_healthexcellent	-0.37751	0.08756	-4.31	1.6e-05
count_chronic	0.14294	0.01359	10.52	< 2e-16
count_adllimited	0.12904	0.05154	2.50	0.01229
count_regionnortheast	0.10407	0.05271	1.97	0.04835
count_regionmidwest	-0.01632	0.04750	-0.34	0.73119
count_regionwest	0.12325	0.05042	2.44	0.01451
count_age	-0.07530	0.03220	-2.34	0.01936
count_afamyes	0.00162	0.07000	0.02	0.98158
count_gendermale	0.00413	0.04213	0.10	0.92196
count_marriedyes	-0.09203	0.04361	-2.11	0.03484
count_school	0.02161	0.00565	3.82	0.00013
count_income	-0.00224	0.00589	-0.38	0.70422
count_employedyes	0.02966	0.07396	0.40	0.68845
count_insuranceyes	0.22715	0.05668	4.01	6.1e-05
count_medicaidyes	0.18479	0.06654	2.78	0.00548

Count regression: Hurdle models

	Estimate	Std. Error	z value	Pr(>|z|)
zero_(Intercept)	-1.47531	0.64633	-2.28	0.0225
zero_healthpoor	0.07084	0.16871	0.42	0.6746
zero_healthexcellent	-0.32851	0.14223	-2.31	0.0209
zero_chronic	0.55651	0.05276	10.55	< 2e-16
zero_adllimited	-0.18817	0.12993	-1.45	0.1476
zero_regionnortheast	0.12922	0.12504	1.03	0.3014
zero_regionmidwest	0.10089	0.11462	0.88	0.3788
zero_regionwest	0.20166	0.13363	1.51	0.1313
zero_age	0.19050	0.08114	2.35	0.0189
zero_afamyes	-0.32697	0.13345	-2.45	0.0143
zero_gendermale	-0.46445	0.09851	-4.71	2.4e-06
zero_marriedyes	0.24726	0.10394	2.38	0.0174
zero_school	0.05421	0.01319	4.11	4.0e-05
zero_income	0.00674	0.01849	0.36	0.7154
zero_employedyes	-0.01232	0.14508	-0.08	0.9323
zero_insuranceyes	0.76246	0.11729	6.50	8.0e-11
zero_medicaidyes	0.55351	0.18121	3.05	0.0023

References

Koenker R, Zeileis A (2009). "On Reproducible Econometric Research." *Journal of Applied Econometrics*, Forthcoming. doi:10.1002/jae.1083

Kleiber C, Zeileis A (2008). *Applied Econometrics with R*. Springer-Verlag, New York. URL http://CRAN.R-project.org/package=AER

Zeileis A, Kleiber C, Jackman S (2008). "Count Data Regression in R." *Journal of Statistical Software*, **27**(8), 1–25. URL http://www.jstatsoft.org/v27/i08/

Zeileis A (2006). "Object-Oriented Computation of Sandwich Estimators." *Journal of Statistical Software*, **16**(9), 1–16. URL http://www.jstatsoft.org/v16/i09/

Zeileis A (2005). "Implementing a Class of Structural Change Tests: An Econometric Computing Approach." *Computational Statistics & Data Analysis*, **50**(11), 2987–3008. doi:10.1016/j.csda.2005.07.001

Zeileis A (2004). "Econometric Computing with HC and HAC Covariance Matrix Estimators." *Journal of Statistical Software*, **11**(10), 1–17. URL http://www.jstatsoft.org/v11/i10/