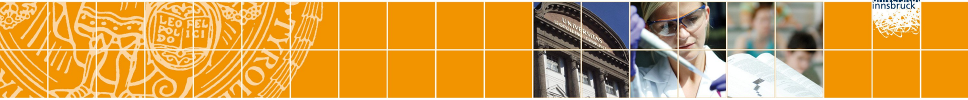
Universität Innsbruck



Statistical Computing in R: Strategies for Turning Ideas into Software

Achim Zeileis

`http://eeecon.uibk.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?
  - Why R?

- How:
  - What should be the guiding principles for implementation?
  - Linear regression in base R.
  - Heteroscedastic censored and truncated regression models in package *crch*.

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.

Why R?

R offers all these features and more:

- R is a full-featured interactive computational environment for data analysis, inference and visualization.
- R is an open-source project, released under GPL.
- Developed for the Unix, Windows and Macintosh families of operating systems by the R Core Team.
- Several object orientation systems, including S3 and S4 classes.
- Everything in R is an object, including functions and function calls.
- Nested functions are lexically scoped.
- Allows for dynamic linking of code in C, C++, Fortran, ...
- Highly extensible with a fast-growing list of add-on packages.

Why R?

Software delivery is particularly easy:

R itself and ∼9000 packages are available (most of them under the GPL) from the Comprehensive R Archive Network (CRAN):

`http://CRAN.R-project.org/`

and can easily be installed from within R via, e.g.

```
R> install.packages("crch")
```

CRAN Task Views:

- `http://CRAN.R-project.org/view=Econometrics`,
- `http://CRAN.R-project.org/view=SocialSciences`,
- and 31 others.

How can this be used in practice?

Examples:

- Linear regression in base R.
- Heteroscedastic censored and truncated regression models in package *crch*.

Illustration:

- Precipitation forecasts for Innsbruck, Austria (`RainIbk`).
- Observed 3 day-accumulated precipitation amounts (`rain`) from SYNOP station Innsbruck Airport from 2000-01-01 to 2013-09-17.
- Corresponding GEFS 11-member ensemble reforecasts of total accumulated precipitation between 5 and 8 days in advance (`rainfc.1`, `rainfc.2`, ..., `rainfc.11`).

Illustration: Precipitation EMOS

Data preprocessing: Load data, transform to square-root scale, compute ensemble statistics, and omit ‘perfect’ ensemble predictions.

```
R> data("RainIbk", package = "crch")
R> RainIbk <- sqrt(RainIbk)
R> RainIbk$ensmean <- apply(RainIbk[, 2:12], 1, mean)
R> RainIbk$enssd <- apply(RainIbk[, 2:12], 1, sd)
R> RainIbk <- subset(RainIbk, enssd > 0)
```

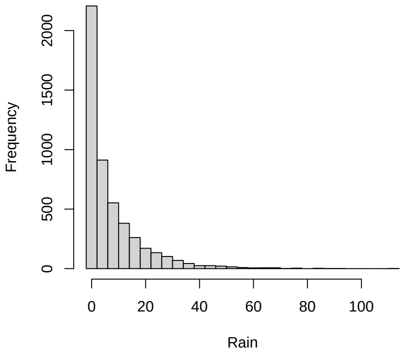
Distribution: Histogram on original and square-root scale.

```
R> hist(RainIbk$rain^2, breaks = 4 * 0:29 - 2)
R> hist(RainIbk$rain, breaks = 0:22/2 - 0.25)
```

Regression: Dependence of observations on ensemble mean.

```
R> plot(rain ~ ensmean, data = RainIbk,
+    pch = 19, col = gray(0, alpha = 0.2))
R> abline(0, 1, col = "green3")
```

Illustration: Precipitation EMOS



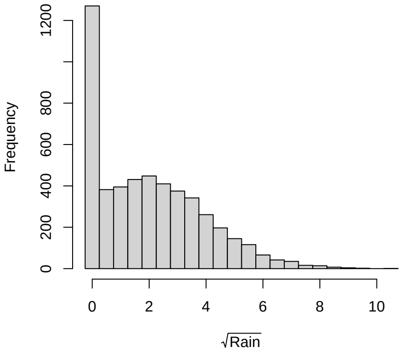


Illustration: Precipitation EMOS

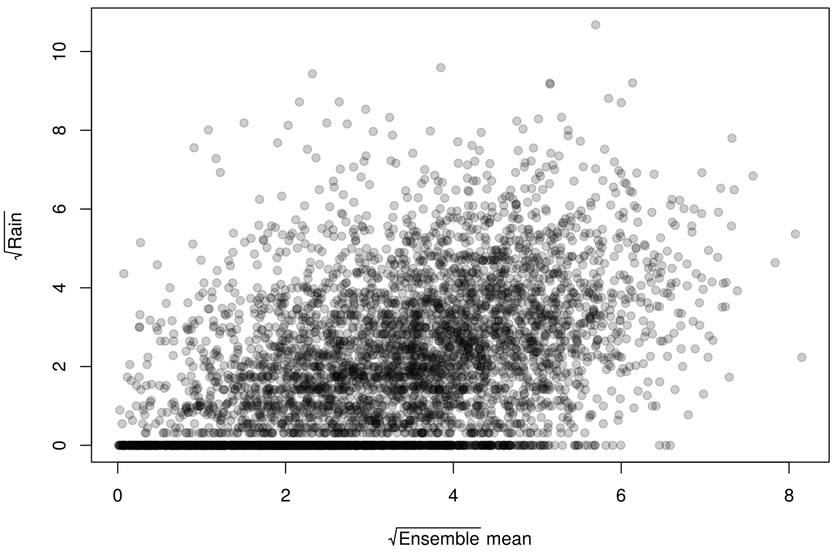


Illustration: Precipitation EMOS

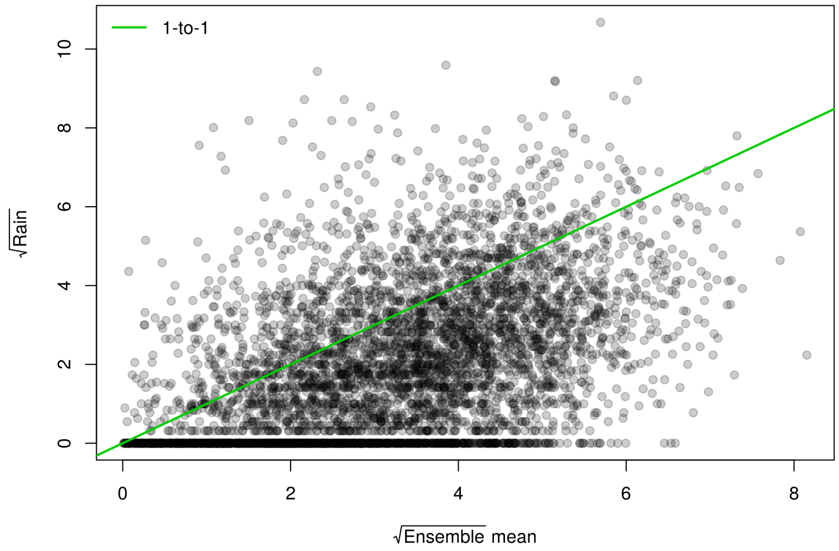
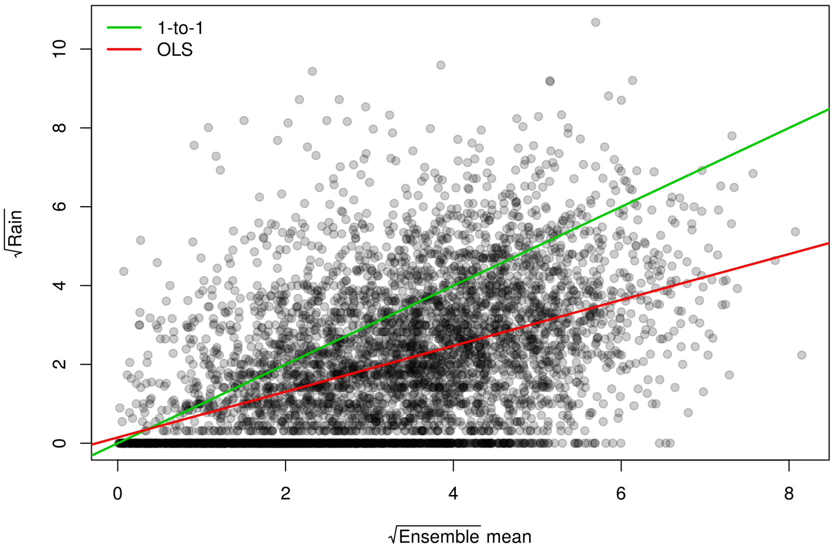


Illustration: Precipitation EMOS



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:

```
R> m <- lm(rain ~ ensmean, data = RainIbk)
```

Object orientation

```
R> coef(m)
```

```
(Intercept)     ensmean
      0.147       0.582
```

```
R> vcov(m)
```

```
            (Intercept)   ensmean
(Intercept)    0.003026 -0.000772
ensmean       -0.000772  0.000240
```

```
R> logLik(m)
```

```
'log Lik.' -9495 (df=3)
```

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)
`nobs()`	Extract number of observations
`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*)
`linearHypothesis()`	Wald tests of linear hypotheses (*car*)
`lrtest()`	Likelihood ratio tests of nested models (*lmtest*)

Reusable components

```
R> BIC(m)
```

```
[1] 19015
```

```
R> confint(m)
```

```
             2.5 % 97.5 %
(Intercept) 0.0395  0.255
ensmean     0.5512  0.612
```

```
R> library("lmtest")
R> coeftest(m)
```

```
t test of coefficients:

            Estimate Std. Error t value Pr(>|t|)
(Intercept)   0.1473     0.0550    2.68   0.0074
ensmean       0.5816     0.0155   37.51   <2e-16
```

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> predict_rain <- predict_fun(RainIbk$ensmean, RainIbk$rain)
R> predict_rain

function(x) cf[1] + cf[2] * x
<environment: 0x4040788>

R> predict_rain(2)

1.31

R> predict_rain(sqrt(4))^2

1.72
```

Heteroscedastic censored regression: Ideas

Extension of OLS:

- Employ censoring to some interval $[\mathit{left}, \mathit{right}]$ to accomodate point mass at the limit(s). Here, $\mathit{left} = 0$.
- Allow for conditional heteroscedasticity depending on regressors.
- In addition to Gaussian responses support distributions with fatter tails (logistic, t_ν).

Latent response: Latent response y^* with location and scale parameters μ and σ follows distribution $\mathcal{D}$.

$$\frac{y^* - \mu}{\sigma} \sim \mathcal{D} \tag{1}$$

with cumulative distribution function $F^*(\cdot)$ and probability density function $f^*(\cdot)$.

Heteroscedastic censored regression: Ideas

Regression: For observations $i = 1, \dots, n$ and regressor vectors $\mathbf{x}_i$, $\mathbf{z}_i$

$$\begin{aligned} \mu_i &= \mathbf{x}_i^\top \beta, \\ g(\sigma_i) &= \mathbf{z}_i^\top \gamma \end{aligned}$$

with monotonic link function $g(\cdot)$. Here, $g(\sigma) = \log(\sigma)$ to assure positivity.

Distributions:

- Standard normal.
- Standard logistic.
- Student-*t* with $\nu = \exp(\delta)$ degrees of freedom.

Heteroscedastic censored regression: Ideas

Observation rule: Observations outside $[\mathit{left}, \mathit{right}]$ are mapped to the interval limits.

$$y = \begin{cases} \mathit{left} & y^* \leq \mathit{left} \\ y^* & \mathit{left} < y^* < \mathit{right} \\ \mathit{right} & y^* \geq \mathit{right} \end{cases}$$

Estimation: By maximum likelihood. Maximize the sum of log-likelihood contributions $\log(f(y_i, \mu_i, \sigma_i))$, where

$$f(y, \mu, \sigma) = \begin{cases} F^*\left(\frac{\mathit{left} - \mu}{\sigma}\right) & y \leq \mathit{left} \\ f^*\left(\frac{y - \mu}{\sigma}\right)/\sigma & \mathit{left} < y < \mathit{right} \\ \left(1 - F^*\left(\frac{\mathit{right} - \mu}{\sigma}\right)\right) & y \geq \mathit{right} \end{cases}$$

Heteroscedastic censored regression: Software

Translation to R: `crch()` for censored regression with conditional heteroscedasticity provides an interface similar to `lm()`.

```
crch(formula, data, subset, na.action, weights, offset,
  link.scale = "log", dist = "gaussian", df = NULL,
  left = -Inf, right = Inf, truncated = FALSE,
  control = crch.control(...), ...)
```

Implementation:

- Data is preprocessed internally.
- Workhorse function `crch.fit()` sets up log-likelihood and corresponding gradient (or score) and Hessian function.
- Quasi-Newton optimization (BFGS) with base R's `optim()`.
- Returns an object of class "`crch`".
- Methods for all standard generics and extractor functions.

Heteroscedastic censored regression: Illustration

Application: Capture heteroscedasticity and/or censoring and/or heavy-tailed distribution.

```
R> library("crch")
R> m1 <- crch(rain ~ ensmean | 1,          data = RainIbk)
R> m2 <- crch(rain ~ ensmean | log(enssd), data = RainIbk)
R> m3 <- crch(rain ~ ensmean | 1,          data = RainIbk, left = 0)
R> m4 <- crch(rain ~ ensmean | log(enssd), data = RainIbk, left = 0)
R> m5 <- crch(rain ~ ensmean | log(enssd), data = RainIbk, left = 0,
+    dist = "logistic")
```

Model selection: All additions lead to model improvements.

```
R> BIC(m, m1, m2, m3, m4, m5)
```

```
   df   BIC
m   3 19015
m1  3 19015
m2  4 18867
m3  3 17966
m4  4 17923
m5  4 17876
```

Heteroscedastic censored regression: Illustration

```
R> barplot(BIC(m) - BIC(m1, m2, m3, m4, m5)[, 2])
```

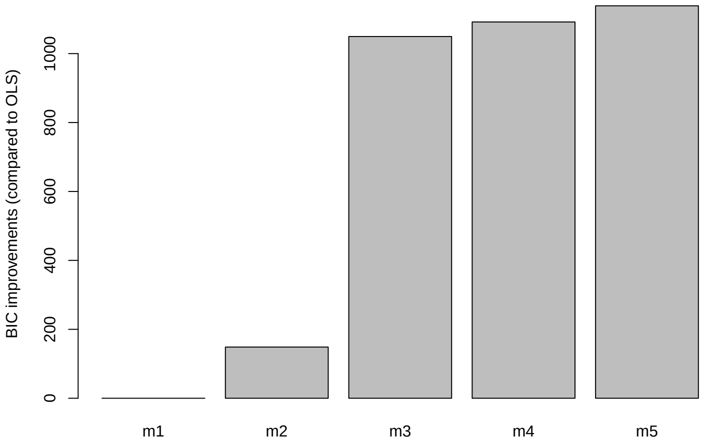
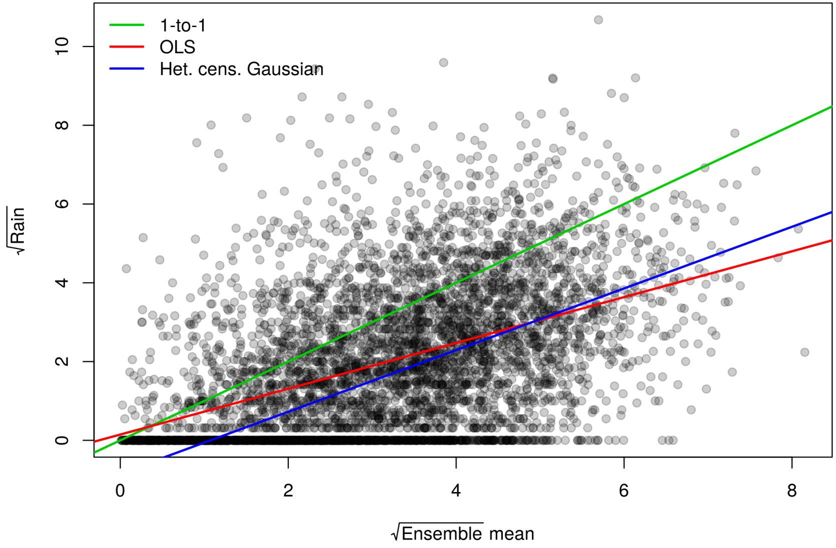


Illustration: Precipitation EMOS



Heteroscedastic censored regression: Illustration

	Gaussian (m1)		Het. cens. Gaussian (m4)	
	location	scale	location	scale
(Intercept)	0.147^{**}	0.496^{***}	-0.840^{***}	0.687^{***}
	(0.055)	(0.010)	(0.073)	(0.013)
ensmean	0.582^{***}		0.783^{***}	
	(0.016)		(0.020)	
log(enssd)				0.220^{***}
				(0.030)
Log-likelihood	-9494.8		-8944.6	
AIC	18995.5		17897.2	
BIC	19015.1		17923.3	
N	4959		4959	

Heteroscedastic censored regression: Software

This implementation uses

- **Object orientation**: Fitted model object with standard interface and methods.
- **Functions as first-class objects**: Several model components can be supplied as functions, e.g., the log-likelihood (and its gradient and Hessian) or the link function (and its inverse and derivative).
- **Lexical scope**: Log-likelihood (and gradient and Hessian) are set up internally as functions of parameters with data accessed via lexical scoping.
- **Compiled code:** Density/score/Hessian functions for censored distributions are implemented in C.
- **Reusable components**: Building blocks like `crch.fit()` and `dcnorm()` may be useful in other applications.

Object orientation

```
R> coef(m4)
```

```
       (Intercept)                ensmean (scale)_(Intercept) 
            -0.840                  0.783               0.687 
 (scale)_log(enssd) 
             0.220 
```

```
R> vcov(m4)
```

```
                    (Intercept)   ensmean
(Intercept)            0.005323 -1.33e-03
ensmean               -0.001333  4.05e-04
(scale)_(Intercept)   -0.000222  4.36e-05
(scale)_log(enssd)     0.000322 -6.48e-05
                    (scale)_(Intercept) (scale)_log(enssd)
(Intercept)                   -2.22e-04           3.22e-04
ensmean                        4.36e-05          -6.48e-05
(scale)_(Intercept)            1.65e-04          -1.14e-04
(scale)_log(enssd)            -1.14e-04           8.97e-04
```

```
R> logLik(m4)
```

```
'log Lik.' -8945 (df=4)
```

Reusable components

```
R> confint(m4)

                      2.5 % 97.5 %
(Intercept)          -0.983 -0.697
ensmean               0.743  0.822
(scale)_(Intercept)   0.662  0.712
(scale)_log(enssd)    0.161  0.279
```

```
R> coeftest(m4)

z test of coefficients:

                     Estimate Std. Error z value Pr(>|z|)
(Intercept)           -0.8405     0.0730  -11.52  < 2e-16
ensmean                0.7829     0.0201   38.92  < 2e-16
(scale)_(Intercept)    0.6870     0.0128   53.52  < 2e-16
(scale)_log(enssd)     0.2199     0.0299    7.34  2.1e-13
```

Lexical scope

Internally: Log-likelihood is defined similar to this standalone example.

```
R> make_censnorm_loglik <- function(x, z, y) {
+    loglik <- function(par) {
+      k <- length(par)
+      beta <- par[1:ncol(x)]
+      gamma <- par[-(1:ncol(x))]
+      mu <- x %*% beta
+      sigma <- exp(z %*% gamma)
+      ll <- ifelse(y <= 0,
+        pnorm(0, mean = mu, sd = sigma, log.p = TRUE),
+        dnorm(y, mean = mu, sd = sigma, log = TRUE)
+      )
+      -sum(ll)
+    }
+  }
R> nll <- make_censnorm_loglik(
+    x = cbind(1, RainIbk$ensmean),
+    z = cbind(1, log(RainIbk$enssd)),
+    y = RainIbk$rain
+  )
```

Lexical scope

```
R> nll

function(par) {
    k <- length(par)
    beta <- par[1:ncol(x)]
    gamma <- par[-(1:ncol(x))]
    mu <- x %*% beta
    sigma <- exp(z %*% gamma)
    ll <- ifelse(y <= 0,
      pnorm(0, mean = mu, sd = sigma, log.p = TRUE),
      dnorm(y, mean = mu, sd = sigma, log = TRUE)
    )
    -sum(ll)
  }
<environment: 0xac83ee0>
```

Lexical scope

Optimization: Minimize negative log-likelihood as a function of the parameters only (using numerical gradients).

```
R> t4_opt <- system.time(
+  m4_opt <- optim(par = rep(0, 4), fn = nll, method = "BFGS")
+  )
R> m4_opt[1:4]
```

```
$par
[1] -0.840  0.783  0.687  0.220

$value
[1] 8945

$counts
function gradient
      87       24

$convergence
[1] 0
```

Functions as first-class objects

Link function: Is stored (and can be supplied) as an actual function, e.g., for computing predictions on new data.

```
R> m4$link$scale$linkfun
```

```
function (mu)
log(mu)
<environment: namespace:stats>
```

```
R> m4$link$scale$linkinv
```

```
function (eta)
pmax(exp(eta), .Machine$double.eps)
<environment: namespace:stats>
```

Functions as first-class objects

Similarly: Distribution can be specified by a probability density *function* rather than a string.

```
R> dcensnorm <- function(y, location = 1, scale = 1, df = NULL,
+    left = -Inf, right = Inf, log = FALSE)
+  {
+    ifelse(y <= left,
+      pnorm(left, mean = location, sd = scale, log.p = log),
+    ifelse(y >= right,
+      pnorm(right, mean = location, sd = scale, log.p = log,
+        lower.tail = FALSE),
+      dnorm(y, mean = location, sd = scale, log = log)
+    ))
+  }
```

Optimization: Using numerical gradients and Hessian.

```
R> t4_d <- system.time(
+  m4_d <- crch(rain ~ ensmean | log(enssd), data = RainIbk, left = 0,
+    dist = list(ddist = dcensnorm), hessian = TRUE)
+  )
```

Compiled code

Comparison: Estimated parameters are (almost) the same but computation times are different.

```
R> cbind(coef(m4), m4_opt$par, coef(m4_d))
```

```
                       [,1]   [,2]   [,3]
(Intercept)          -0.840 -0.840 -0.840
ensmean               0.783  0.783  0.783
(scale)_(Intercept)   0.687  0.687  0.687
(scale)_log(enssd)    0.220  0.220  0.220
```

```
R> cbind(t4, t4_opt, t4_d)[1,]
```

```
    t4 t4_opt   t4_d
 0.108  0.352  0.636
```

Compiled code

Main reason: C implementation of the distribution (plus analytical rather than numeric gradients/Hessian, also in C).

```
R> dcnorm

function (x, mean = 0, sd = 1, left = -Inf, right = Inf, log = FALSE)
{
    input <- data.frame(x = as.numeric(x), mean = as.numeric(mean),
        sd = as.numeric(sd), left = as.numeric(left), right = as.numeric
    with(input, .Call("dcnorm", x, mean, sd, left, right, log))
}
<environment: namespace:crch>
```

Heteroscedastic censored regression: Extensions

Further features: Supported by *crch* package.

- Truncated instead of censored response distributions, e.g., for two-part hurdle models or limited distributions without point mass (such as wind).
- Boosting instead of maximum likelihood estimation for variable selection and regularization/shrinkage of parameters.
- Other estimation techniques (such as CRPS) can be performed by using the specification of `dist` as a function.

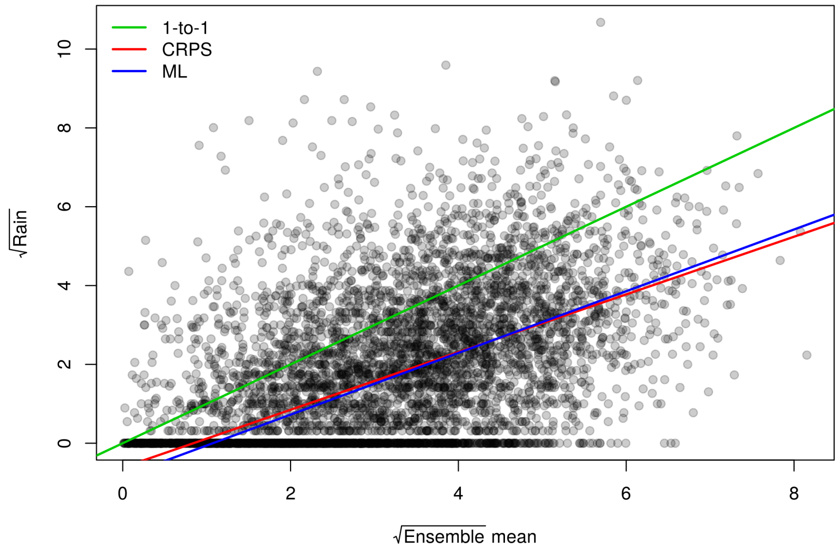
Heteroscedastic censored regression: Extensions

CRPS: Continuous ranked probability score implemented in *scoringRules* package (Jordan, Krüger, Lerch 2016).

```
R> library("scoringRules")
R> dcrps <- function(y, location = 1, scale = 1, df = NULL,
+    left = 0, right = Inf, log = FALSE)
+  {
+    -crps(y, family = "normal", location = location, scale = scale,
+      lower = left, upper = right, lmass = "cens", umass = "cens")
+  }
R> m4_crps <- crch(rain ~ ensmean | log(enssd), data = RainIbk,
+    left = 0, dist = list(ddist = dcrps), hessian = TRUE)
```

Comparison: ML and CRPS lead very similar results with comparable parameters and (in-sample) scores.

Heteroscedastic censored regression: Extensions



Heteroscedastic censored regression: Extensions

```
R> logLik(m4)

'log Lik.' -8945 (df=4)

R> sum(dcnorm(RainIbk$rain,
+    mean = predict(m4, type = "location"),
+    sd = predict(m4, type = "scale"),
+    left = 0, log = TRUE))

[1] -8945

R> sum(dcnorm(RainIbk$rain,
+    mean = predict(m4_crps, type = "location"),
+    sd = predict(m4_crps, type = "scale"),
+    left = 0, log = TRUE))

[1] -8974
```

Heteroscedastic censored regression: Extensions

```
R> logLik(m4_crps)/nobs(m4_crps)

'log Lik.' -0.875 (df=4)

R> mean(crps(RainIbk$rain, family = "normal",
+    mean = predict(m4_crps, type = "location"),
+    sd = predict(m4_crps, type = "scale"),
+    lower = 0, lmass = "cens"))

[1] 0.875

R> mean(crps(RainIbk$rain, family = "normal",
+    mean = predict(m4, type = "location"),
+    sd = predict(m4, type = "scale"),
+    lower = 0, lmass = "cens"))

[1] 0.877
```

References

Messner JW, Mayr GJ, Zeileis A (2016). "Heteroscedastic Censored and Truncated Regression with crch." *The R Journal*, Forthcoming. `https://journal.R-project.org/archive/accepted/messner-mayr-zeileis.pdf`

Messner JW, Mayr GJ, Zeileis A (2016). "Non-Homogeneous Boosting for Predictor Selection in Ensemble Post-Processing." *Working Paper 2016-04*, Working Papers in Economics and Statistics, Research Platform Empirical and Experimental Economics, Universität Innsbruck.
URL `http://EconPapers.RePEc.org/RePEc:inn:wpaper:2016-04`.

Koenker R, Zeileis A (2009). "On Reproducible Econometric Research." *Journal of Applied Econometrics*, **24**(5), 833–847. `doi:10.1002/jae.1083`

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`