distrMod — an S4-class based package for statistical models

Peter Ruckdeschel[1] Matthias Kohl[2]

[1] Fraunhofer Institut Techno- und Wirtschaftsmathematik
Abteilung Finanzmathematik
Peter.Ruckdeschel@itwm.fraunhofer.de

[2] UNIVERSITÄT BAYREUTH
Lehrstuhl Stochastik
Matthias.Kohl@uni-bayreuth.de

UseR! – The R User Conference 2008,

Dortmund, August 12

A non-standard model

- one-dim. location scale model:
  - $X_i \overset{\text{i.i.d.}}{\sim} P_\theta, \quad \theta = (\mu, \sigma), \quad \mathcal{L}_\theta(X_i) = \mathcal{L}(\mu + \sigma v_i)$
  - $v_i \overset{\text{i.i.d.}}{\sim} P, \quad P(dx) = p(x)\,dx, \quad p(x) \propto e^{-|x|^3}$

  $\Longrightarrow$ Scores $\Lambda_\theta(x) = (3\,\mathrm{sign}(y)y^2, 3|y|^3 - 1)/\sigma, \quad y = (x-\mu)/\sigma$
- goal: estimate θ from $X_1, \ldots, X_n$
  - risk: mean squared error (MSE)
  - asymptotically optimal: maximum likelihood (MLE)
  - alternatives:
    - $(\mathrm{median}, \mathrm{mad})$
    - method of moment estimators (MMEs, not here),
    - minimum distance estimators (MDEs),
    - robust estimators (Matthias Kohl's talk)

Implementations in R so far

- fitdistr from B. Ripley's package MASS
  - arguments: x, densfun, start (and ...)
  - return value: object of S3-class fitdistr
    — a list with components estimate, sd, loglik
  - here:

```
>  ## data already in object x
>  mydf ← function(x, loc, scale) {
+       y ← (x-loc)/scale; exp(-abs(y)^3)/scale}
>  mleMASS ← fitdistr(x, mydf, start = list("loc" =
+                          median(x), "scale" = mad(x)))
```

- mle from package stats4
  - arguments: minuslogl, start, method, (and ...)
  - return value: object of S4-class mle
    — with slots call, coef, full, vcov, min, details, minuslogl, method
  - here:

```
> ll ← function(loc,scale){-sum(log(mydf(x, loc,scale)))}
> mlestats4 ← mle(ll, start = list("loc" = median(x),
+                   "scale" = mad(x)))
```


How to realize particular methods

Beyond the default method:

- for the MLE
  - particular tuning of the optimization routines is helpful
  - numerical optimization can totally be avoided in special cases

  e.g. under normality $\hat{\theta}^{\mathrm{MLE}}(x) = (\mathrm{mean}(x), \mathrm{sd}(x))$
- fitdistr does this with 9 if-clauses for particular models
- mle has *no* particular cases
- good case for method dispatch if there were *distribution classes*

Advantages of method dispatch in this case:

- could react on different particular settings
- would automatically dispatch according to inheritance structure
- would avoid need to modify existing (R-Core) code (in particular: no extra if-clauses for any new model ...)

$\rightsquigarrow$ need less coordination with pkg. maintainer

$\rightsquigarrow$ *good for collaborative/distributed programming*

Packages for Distributions Classes

The distrXXX Family of Packages

(Co-)Authors (besides M. Kohl)

- Thomas Stabla: `statho3@web.de`
- Florian Camphausen: `fcampi@gmx.de`

Organization in packages

- `distr`, `distrEx`; [and `distrSim`, `distrTEst`]
- `distrDoc`, `distrTeach`, `distrMod`

Availability

- published on `CRAN`; current version `1.9`
- devel version `2.0` on `R-forge`, `r-forge.r-project.org`

Fraunhofer ITWM Institut Techno- und Wirtschaftsmathematik

`distr`: what is this good for?

- Problem: How to pass a distribution as an argument?
  arises e.g. in a function returning the population median
- lots of distributions already implemented to R
  naming convention: `[r,d,p,q]<distr.name>` for
  - `r` RNG
  - `d` density / probability function
  - `p` cdf
  - `q` quantile function
- solution by **eval**, **parse**, **paste**

```
>   mymedian ← function(distr, ...){
+    eval(parse(text = paste( "x␣=␣q", distr,
+          "(1/2,...)", sep = ""))); return(x)}
```

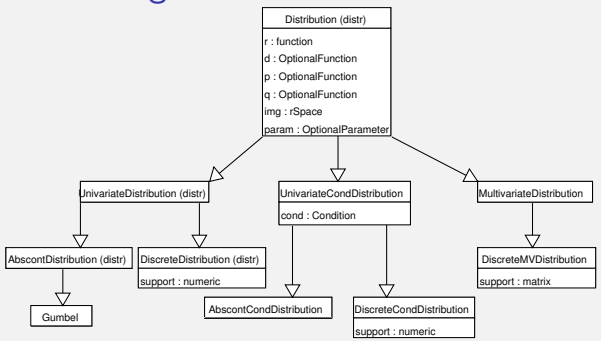
- better idea: having a "variable type" *distribution*
  and functions `p`, `d`, `q`, `r` defined for this type
- i.e.: **q**(x) returns the quantile function $\rightsquigarrow$

```
>  median ← function(X){q(X)(0.5)}
```

$\Longrightarrow$ Development of this concept in package `distr`

Fraunhofer ITWM Institut Techno- und Wirtschaftsmathematik

Concept of R-Packages `distr`



- `AbscontDistribtion` → `Beta`, `Cauchy`, `Chisq`, `Exp`, `Fd`, `Gammad`, `Logis`, `Lnorm`, `Norm`, `Td`, `Unif`, `Weibull`
- `DiscreteDistribtion` → `Binom`, `Dirac`, `Geom`, `Hyper`, `Nbinom`, `Pois` (...all from `stats` package)
- particular methods for **plot**, **show**, **summary**,...
- easy generating functions DiscreteDistribution (), AbscontDistribution ()
- classes for mixing distributions

Fraunhofer ITWM Institut Techno- und Wirtschaftsmathematik

Methods

Arithmetics for distributions

- automatic generation of image distributions (of r.v.'s)
- $\rightsquigarrow$ overloaded operators `"+"`, `"-"`, `"*"`, `"/"`, `"^"`
- group `math` of unary math. op.'s available, i.e., `sin`, `exp`, ...

e.g. Y←(3∗X∗Z+5)/4 —generates $\mathcal{L}(Y)$ for $Y = (3XZ + 5)/4$

simil.: **exp**(**sin**(3∗X+5)/4)

Implementation details

- binary operators interpret operands as stoch. independent
- default simulation-based method for filling slots `d`, `p`, `q`
- default FFT-based convolution methods for two indep. r.v.'s; c.f. K., R., & Stabla[04]
- by method dispatch: use of analytic expressions
  — e.g. $\mathcal{N}(\mu_1, \sigma_1^2) * \mathcal{N}(\mu_2, \sigma_2^2) = \mathcal{N}(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2)$
- distributions with non-trivial discrete and (abs.)continuous part realized as mixing distributions

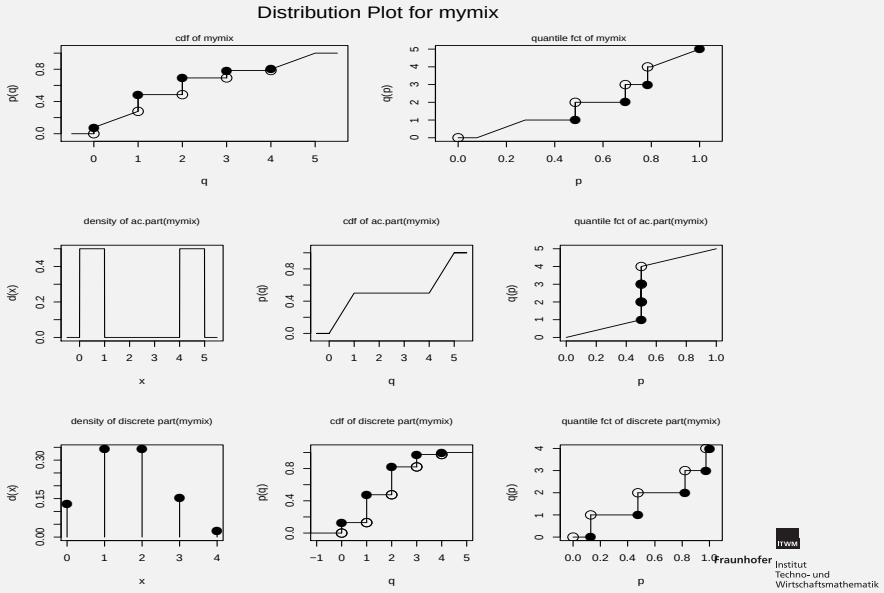
Fraunhofer ITWM Institut Techno- und Wirtschaftsmathematik

Example

```
> ### generate some distribution mymix
> wg ← flat.mix(UnivarMixingDistribution(Unif(0,1),Unif(4,5),
+                       withSimplify=FALSE))
> mymix ← UnivarLebDecDistribution(acPart = wg,
+                 discretePart = Binom(4,.4),  acWeight = 0.4)
> #slots r,p,q:
> r(mymix)(5)              ## RNG
[1] 0.7672470 0.9290900 4.0000000 2.0000000 0.4746638
> p(mymix)(c(0,1.2,3.2)) ## cdf
[1] 0.07776 0.48512 0.78464
> q(mymix)((0:4)/4)       ## quartiles
[1] 0.000000 0.861200 1.571420 3.124360 5.000000
> #some new distribution as image law under trafo
> (mnew ← mymix*Norm(0,2)+3) # output shortened
An object of class "AffLinUnivarLebDecDistribution"
 --- a Lebesgue decomposed distribution:

    Its discrete part (with weight 0.078000) is a[...]
    Its absolutely continuous part (with weight 0.922000) is a[...]
> plot(mymix)
```


Example



Package distrEx

- general expectation operator
- functionals on distributions like
  median, var, sd, MAD and IQR
- distances between distributions
  (e.g. Kolmogoroff–, Total-Variation–, Hellinger-distance)
- (factorized) conditional distributions and expectations

Example: Expectation Operator

- for a normal variable D_1 try to realize $\mathrm{E}\, D_1$, $\mathrm{E}\, D_1^2$

```
> D1 ← Norm(mean=2)
> m1 ← E(D1)                    # = 2
> E(D1, function(x){ x^2 }) # E(D1^2)
```

- now —without changing the code— the same for a Poisson variable; ⇝ same calls but different dispatched methods

```
> D1 ← Pois(lambda=3)
> m1 ← E(D1)                    # = 3
> E(D1, function(x){ x^2 })
```


Models in the distrXXX-family: package distrMod

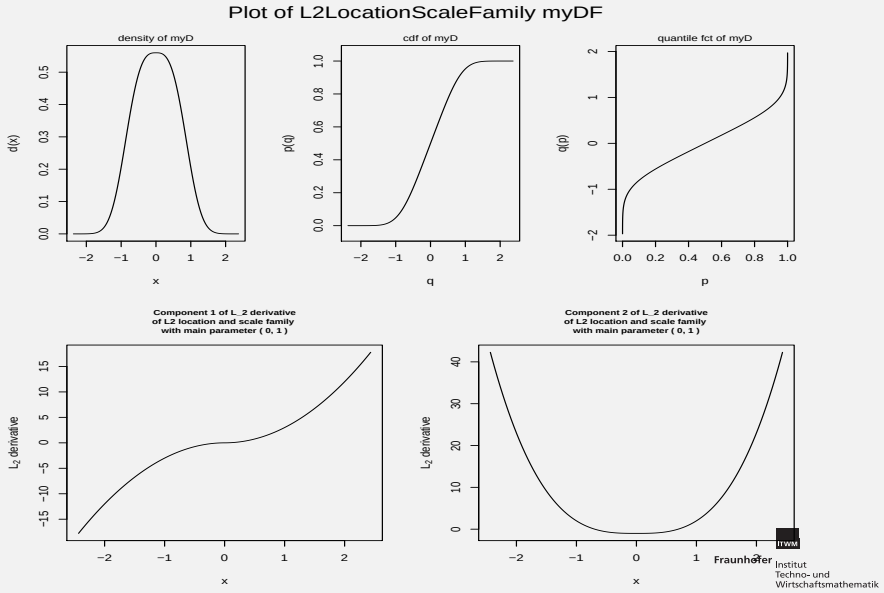
- new class L2ParamFamily with slots
  - distribution of the observations
  - parameter
  - L_2-derivative $\Lambda_\theta(x)$ and Fisher information $\mathcal{I}_\theta$
  - to "move" P_θ from θ to θ': functional slots realizing maps

$$\theta \mapsto \mathcal{I}_\theta, \qquad \theta \mapsto \Lambda_\theta(x), \qquad \theta \mapsto P_\theta$$

- subclass L2LocationScaleFamily
  - generating function L2LocationScaleFamily()
  - ## generation of distribution with density $\propto e^{-|x|^3}$

```
> myD ← AbscontDistribution(d = function(x){
+                              exp(-abs(x)^3)}, withS = TRUE)
## generating some data (already run earlier)
> x ← r(myD)(40)
## generation of L2Family
> myDF ← L2LocationScaleFamily(centraldistr = myD)
> plot(myDF)
```

Example



Estimators in package `distrMod`

- possible in our framework:
  general, dispatchable "Minimum Criterium Estimator" (MCE)
- examples of MCEs:
  - MLE : criterium $\hat{=}$ neg. Loglikelihood
  - MDE : criterium $\hat{=}$ distance(empirical, P_θ)
- implementation as function MCEstimator()
  (+ [essentially] wrapper functions MDEstimator(), MLEstimator())
  - arguments: `x`, `ParamFamily`, `criterion`, `startPar` (and some optional ones)
  - return value class `MCEstimate` with slots `estimate`, `criterion`, `samplesize`, `asvar` and some more); subclass of class `Estimate`
- for confidence intervals:
  confint method for objects of class `Estimate`

Enhancements by package `distrMod`

Estimators

- estimators are available for *any* object of class `L2ParamFamily`
  (e.g. Poisson, Beta, Gamma and many more)
- internal dispatch according to argument `ParamFamily`

$\rightsquigarrow$ new particular methods without modifying existing code by

1. deriving a new subclass of `L2ParamFamily` —usually by
   setClass(<**new class** name>, contains = "L2ParamFamily")
2. specifying a method mceCalc resp. mleCalc for this class, e.g.

```
setMethod("mleCalc", signature(x = "numeric",
                  PFam = "NormLocationScaleFamily"),
      function(x, PFam) c(mean(x),sd(x)) )
```

Confidence Intervals

- method dispatch against arguments `object`, `method`

$\rightsquigarrow$ unified interface confint for several methods for one estimator class (yet to be done)

again: may be used by foreign code, without modification of our code

Example

```
> (mledistrMod <- MLEstimator(x,myDF))
Evaluations of Maximum likelihood estimate:
-------------------------------------------
estimate:
       loc          scale
  -0.01880420    0.86496139
 ( 0.07853252) ( 0.07896397)
> # comparison
> mleMASS ## mlestats4 gives the same without SE
       loc          scale
  -0.01880576    0.86499847
 ( 0.08024227) ( 0.07896108)
> confint(mledistrMod)
A[n] asymptotic (CLT-based) confidence interval:
          2.5 %    97.5 %
loc   -0.1727251 0.1351167
scale  0.7101949 1.0197279
Type of estimator: Maximum likelihood estimate:
samplesize:   40
Call by which estimate was produced:
MLEstimator(x = x, ParamFamily = myDF)
```

Example (continued)

```
> mdeCvM ← MDEstimator(x,myDF, distance=CvMDist)
> asvar(mdeCvM)← distrMod:::.CvMMDCovariance(myDF, param =
+               ParamFamParameter(main = estimate(mdeCvM)),exp=2)
> mdeCvM
Evaluations of Minimum CvM distance estimate:
---------------------------------------------
estimate:
       loc          scale
  -0.05985862    0.80168752
 ( 0.12791561) ( 0.19698762)
> (mdeKolm ← MDEstimator(x,myDF))
Evaluations of Minimum Kolmogorov distance estimate:
----------------------------------------------------
estimate:
       loc       scale
-0.08251711  0.78486426
> (mdeTV ← MDEstimator(x,myDF,distance=TotalVarDist))
Evaluations of Minimum Total variation distance estimate:
---------------------------------------------------------
estimate:
       loc      scale
-0.0604611  0.6104529
```

Fraunhofer ITWM Institut Techno- und Wirtschaftsmathematik

Bibliography

J. M. Chambers. *Programming with Data. A guide to the S language*. Springer, 1998. URL http://cm.bell-labs.com/cm/ms/departments/sia/Sbook/.

R Development Core Team. R*: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria, 2005 URL http://www.R-project.org.

M. Kohl, P. Ruckdeschel, and T. Stabla. General Purpose Convolution Algorithm for Distributions in S4-Classes by means of FFT. Technical Report. Feb. 2005. URL http://www.uni-bayreuth.de/departments/math/org/mathe7/RUCKDESCHEL/pubs/comp.pdf

P. Ruckdeschel, M. Kohl, T. Stabla, and F. Camphausen. S4 Classes for Distributions. *R-News*, **6**(2): 10–13. http://CRAN.R-project.org/doc/Rnews/Rnews_2006-2.pdf. Also available as manual for packages distr, distrSim, distrTEst version 1.8, Oct. 2006. URL http://www.uni-bayreuth.de/departments/math/org/mathe7/DISTR/distr.pdf.

Fraunhofer ITWM Institut Techno- und Wirtschaftsmathematik