distrMod — an S4-class based package for statistical models

Peter Ruckdeschel[1] Matthias Kohl[2]

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

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

UseR 2008, Dortmund, August 12, 2008

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:
    - **(median, mad)**
    - method of moment estimators (MMEs, not here),
    - minimum distance estimators (MDEs),
    - robust estimators (next 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` (without data argument!), `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)))
```

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** (without data argument!), **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

- MLE should take into account particular distributional settings

e.g. in Gaussian location-scale model $\hat{\theta}^{\text{MLE}}(x) = (\mathbf{mean}(x), \mathbf{sd}(x))$

- `fitdistr` does this with **9** if-clauses for particular models
- `mle` has *no* particular cases
- if there were *distribution classes* good case for method dispatch

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 code of `fitdistr` resp. `mle`
  (no extra `if`-clauses, no modification in `R-Core` code...)

$\rightsquigarrow$ *good for distributed programming*

Fraunhofer Institut Techno- und Wirtschaftsmathematik

How to realize particular methods

- MLE should take into account particular distributional settings

e.g. in Gaussian location-scale model $\hat{\theta}^{\text{MLE}}(\boldsymbol{x}) = (\mathbf{mean}(\boldsymbol{x}), \mathbf{sd}(\boldsymbol{x}))$

- `fitdistr` does this with **9** if-clauses for particular models
- `mle` has *no* particular cases
- if there were *distribution classes* good case for method dispatch

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 code of `fitdistr` resp. `mle`
  (no extra `if`-clauses, no modification in `R-Core` code...)

$\rightsquigarrow$ *good for distributed programming*

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`

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`

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`

`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
- a lot of distributions implemented to R
  naming convention: `[r,d,p,q]<distr.name>` for
  - `p` cdf
  - `d` density / probability function
  - `q` quantile function
  - `r` RNG
- 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
- then: `q(x)` returns the quantile function $\rightsquigarrow$

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

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

Fraunhofer 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
- a lot of distributions implemented to R
  naming convention: [`r,d,p,q`]<`distr.name`> for
  - `p` cdf
  - `d` density / probability function
  - `q` quantile function
  - `r` RNG
- 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
- then: `q(x)` returns the quantile function $\rightsquigarrow$

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

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

Fraunhofer 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
- a lot of distributions implemented to R
  naming convention: `[r,d,p,q]<distr.name>` for
  - `p` cdf
  - `d` density / probability function
  - `q` quantile function
  - `r` RNG
- 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
- then: `q(x)` returns the quantile function $\rightsquigarrow$

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

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

`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
- a lot of distributions implemented to R
  naming convention: `[r,d,p,q]<distr.name>` for
  - `p` cdf
  - `d` density / probability function
  - `q` quantile function
  - `r` RNG
- solution by **eval**, **parse**, **paste**

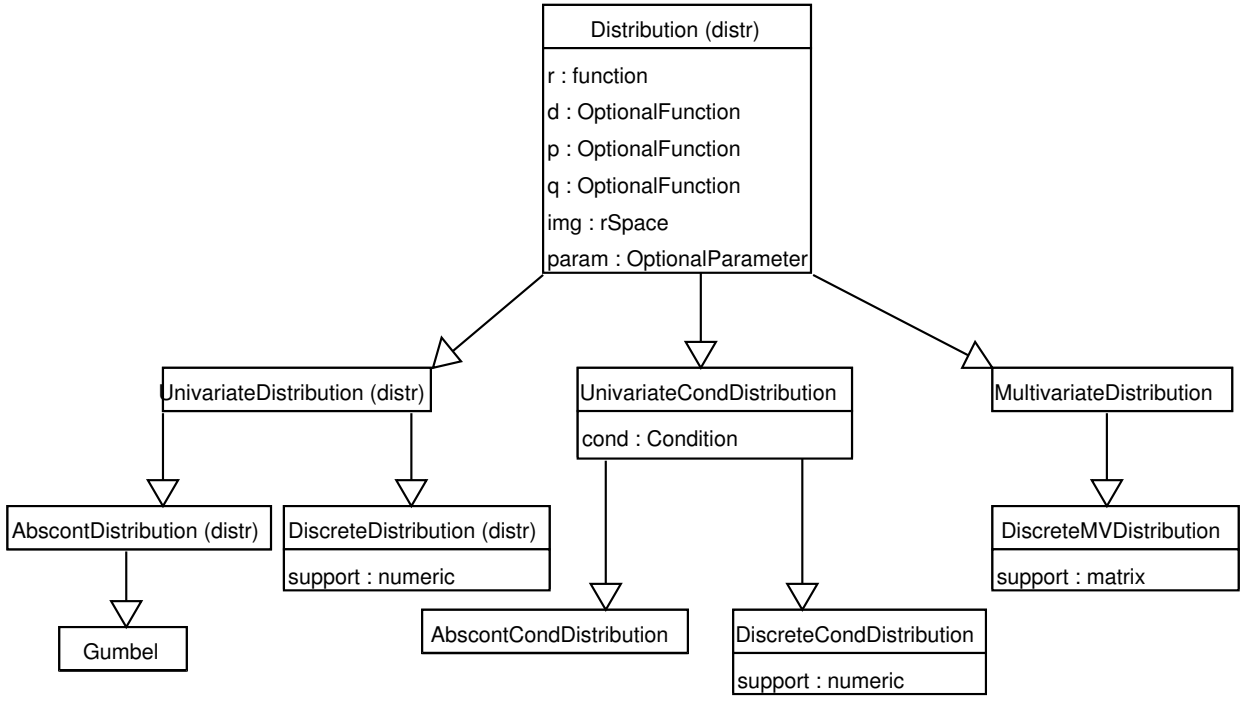
```
mymedian ← function(distr, ...){
 eval(parse(text = paste( "x = q", distr,
      "(1/2,...)", sep = "")))); return(x)}
```

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

  median ←**function**(X){**q**(X)(0.5)}

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

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 and Lebesgue-decomposed distributions

Methods

Arithmetics for distributions

- automatic generation of image distributions (of r.v.'s)

$\rightsquigarrow$ overloaded operators `"+"`, `"-"`, `"*"`, `"/"`, `"^"`

- group `math` of unary mathematical operations is available

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
- `RtoDPQ`: default simulation-based method for filling slots `d`, `p`, `q`
- default FFT-based convolution method for two indep. r.v.'s; c.f. K., R., & Stabla[04]
- by method dispatch: use of analytic expressions where possible — 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

Methods

Arithmetics for distributions

- automatic generation of image distributions (of r.v.'s)

$\rightsquigarrow$ overloaded operators `"+"`, `"-"`, `"*"`, `"/"`, `"^"`

- group `math` of unary mathematical operations is available

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

simil.: `exp(sin(3*X+5)/4)`

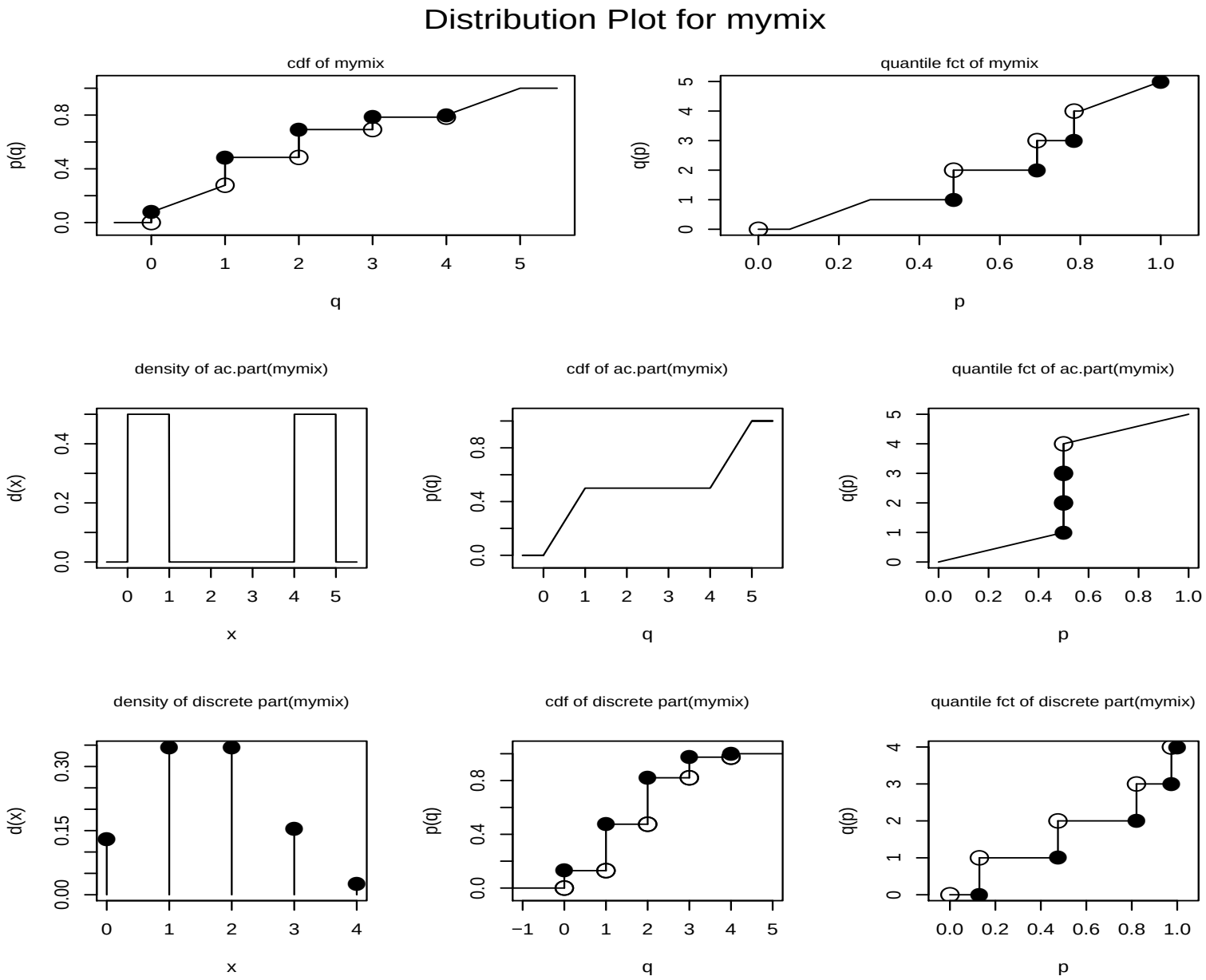
Implementation details

- binary operators interpret operands as stoch. independent
- `RtoDPQ`: default simulation-based method for filling slots `d`, `p`, `q`
- default FFT-based convolution method for two indep. r.v.'s;
  c.f. K., R., & Stabla[04]
- by method dispatch: use of analytic expressions where possible — 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

Example

```
### generate some distribution mymix
>   wg <- flat.mix(UnivarMixingDistribution(Unif(0,1),Unif(4,5)))
>   mymix <- UnivarLebDecDistribution(acPart = wg,
+               discretePart = Binom(4,.4),  acWeight = 0.4)

#slots r,p,q:
>r(mymix)(5) ## RNG
[1] 3.000000 2.000000 4.335709 1.000000 1.000000
>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
>(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



Fraunhofer ITWM Institut Techno- und Wirtschaftsmathematik

Package `distrEx`

- a general expectation operator to a given distribution F
- several functionals on distributions like
  median, var, sd, MAD and IQR
- several 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 $\mathsf{E}\, D_1$, $\mathsf{E}\, D_1^2$

```
require("distrEx")
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; this gives the same calls but different dispatched methods

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

Fraunhofer Institut
Techno- und
Wirtschaftsmathematik

Package `distrEx`

- a general expectation operator to a given distribution F
- several functionals on distributions like
  median, var, sd, MAD and IQR
- several 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 $\mathsf{E}\, D_1$, $\mathsf{E}\, D_1^2$

```
require("distrEx")
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; this gives the same calls but different dispatched methods

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

Package `distrEx`

- a general expectation operator to a given distribution F
- several functionals on distributions like
  median, var, sd, MAD and IQR
- several 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 $\mathbf{E}\, D_1$, $\mathbf{E}\, D_1^2$

```
require("distrEx")
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; this gives the same calls but different dispatched methods

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

Fraunhofer Institut
Techno- und
Wirtschaftsmathematik

Models in the `distrXXX`-family: package `distrMod`

- new class `L2ParamFamily` with slots
  - distribution of the observations (of class `Distribution`)
  - parameter (partitioned in main and nuisance)
  - L_2-derivative $\Lambda_\theta(x)$ (of class `EuclRandVariable`)
  - Fisher information $\mathcal{I}_\theta$
  - functional slots realizing maps
    - $\theta \mapsto \mathcal{I}_\theta$ (matrix-valued)
    - $\theta \mapsto \Lambda_\theta(x)$ (function-valued)
    - $\theta \mapsto P_\theta$ (distribution-class-valued)
  - symmetry slots (to make use of it in integration)
- new class `L2LocationScaleFamily` with additional slot `LogDeriv`
  - generating function `L2LocationScaleFamily()`
  - example:

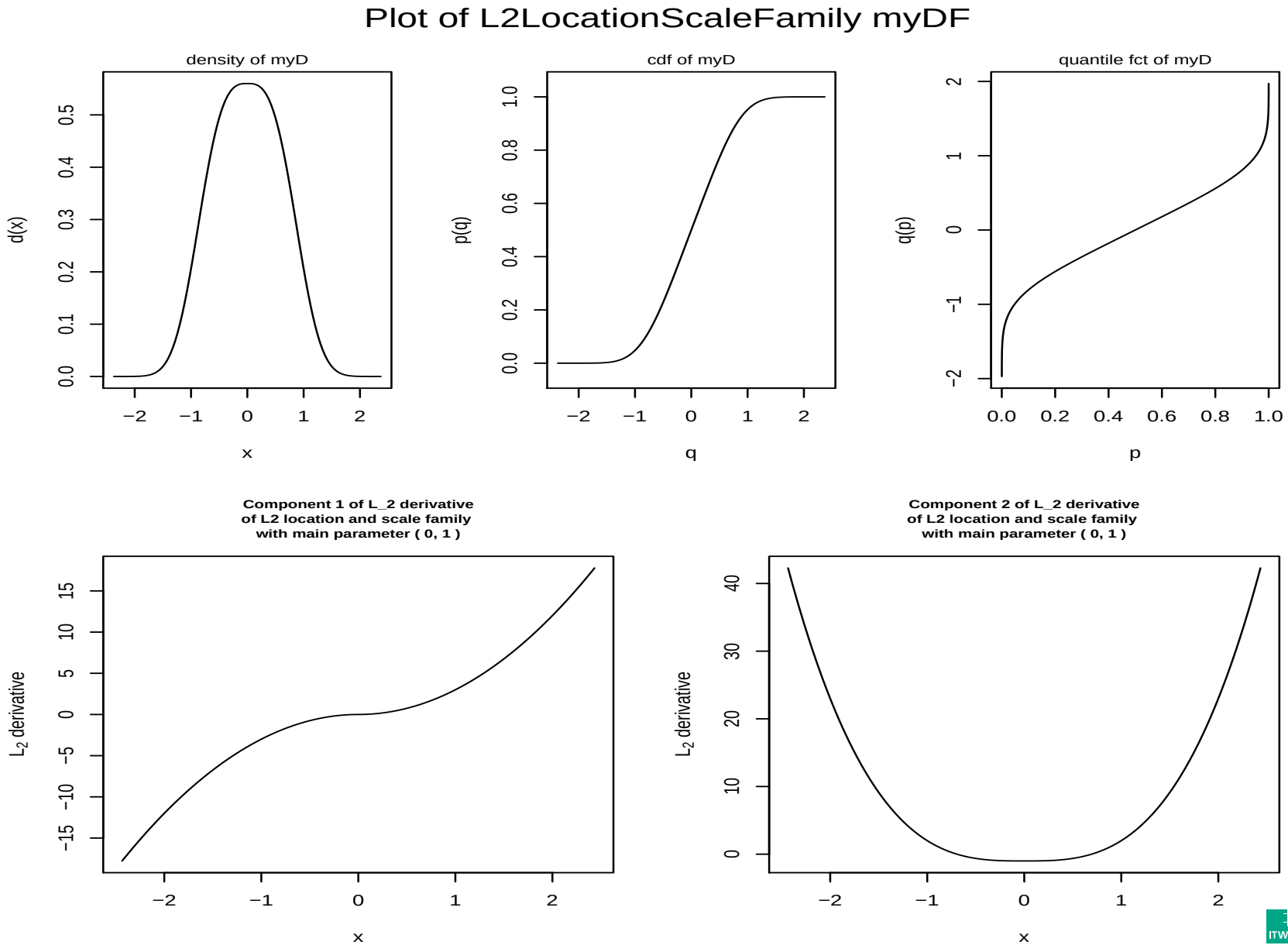
```
## generation of distribution with density ∝ e^(−|x|^3)
myD ← AbscontDistribution(d = function(x) exp(−abs(x)^3),
                          withS = TRUE)

## generating some data
x ← r(myD)(40)
## generation of L2Family
myDF ← L2LocationScaleFamily(centraldistr = myD)
plot(myDF)
```

Models in the `distrXXX`-family: package `distrMod`

- new class `L2ParamFamily` with slots
  - distribution of the observations (of class `Distribution`)
  - parameter (partitioned in main and nuisance)
  - L_2-derivative $\Lambda_\theta(x)$ (of class `EuclRandVariable`)
  - Fisher information $\mathcal{I}_\theta$
  - functional slots realizing maps
    - $\theta \mapsto \mathcal{I}_\theta$ (matrix-valued)
    - $\theta \mapsto \Lambda_\theta(x)$ (function-valued)
    - $\theta \mapsto P_\theta$ (distribution-class-valued)
  - symmetry slots (to make use of it in integration)
- new class `L2LocationScaleFamily` with additional slot `LogDeriv`
  - generating function L2LocationScaleFamily()
  - example:

```
## generation of distribution with density ∝ e^(−|x|^3)
myD ← AbscontDistribution(d = function(x) exp(−abs(x)^3),
                          withS = TRUE)
## generating some data
x ← r(myD)(40)
## generation of L2Family
myDF ← L2LocationScaleFamily(centraldistr = myD)
plot(myDF)
```

Example


Plot of L2LocationScaleFamily myDF
density of myD
d(x)
x
cdf of myD
p(q)
q
quantile fct of myD
q(p)
p
Component 1 of L_2 derivative
of L2 location and scale family
with main parameter (0, 1)
L2 derivative
x
Component 2 of L_2 derivative
of L2 location and scale family
with main parameter (0, 1)
L2 derivative
x

Estimators in package `distrMod`

- possible in our framework:
  general, dispatchable "Minimum Criterium Estimator" (MCE)
- examples of MCEs:
  - MLE : criterium $\widehat{=}$ neg. Loglikelihood
  - MDE : criterium $\widehat{=}$ distance(empirical, $\boldsymbol{P}_\theta$)
- 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:
  made confint generic and defined methods for objects of class **`Estimate`**

Enhancements by package `distrMod`

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

⇝ 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))})
```

- new confidence interval methods accessible by same interface `confint` (specified by `method` argument) [other methods not yet implemented]
- dispatch in `confint` according to `method` ⇝ may be used by foreign code, without modification of our code

Example

```
> (mledistrMod <- MLEstimator(x,myDF))
Evaluations of Maximum likelihood estimate:
----------------------
       loc          scale
  -0.04601414    0.87459048
 ( 0.07940678) ( 0.07984302)

# comparison
> mleMASS ## mlestats4 gives the same without SE
       loc          scale
  -0.04597317    0.87454768
 ( 0.08024884) ( 0.07983405)

> confint(mledistrMod)
A[n] asymptotic (CLT-based) confidence interval:
           2.5 %    97.5 %
loc   -0.2016486 0.1096203
scale  0.7181010 1.0310799
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:
----------------------
      loc         scale
  -0.0708801    0.8423145
 ( 0.1158490) ( 0.1904424)
> (mdeKolm <- MDEstimator(x,myDF))
Evaluations of Minimum Kolmogorov distance estimate:
--------------------------
estimate:
       loc      scale
-0.0673269  0.8418084
> (mdeTV <- MDEstimator(x,myDF,distance=TotalVarDist))
Evaluations of Minimum Total variation distance estimate:
-----------------------------
estimate:
        loc       scale
-0.01643707  0.66559198
```

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.