A Guide to the EVD R Package (Version 1.0).

Alec Stephenson

25th January 2002

1 Introduction

1.1 What is the EVD package?

The EVD (Extreme Value Distributions) package is an add-on package for the R (Ihaka and Gentleman, 1996) statistical computing system. The package extends simulation, distribution, inverse distribution (quantile) and density functions to univariate, bivariate and (for simulation) multivariate parametric extreme value distributions. It also provides fitting functions which calculate maximum likelihood estimates for univariate and bivariate models.

The package does not currently implement non-stationary fitting.

1.2 Contents

This guide contains examples on the use of the EVD package. The examples do not include theoretical justification. See Coles (2001) for an introduction to the statistics of extreme values. See Kotz and Nadarajah (2000) for a theoretical treatment of univariate and multivariate extreme value distributions. Section 2 covers the standard functions for univariate extreme value distributions. Section 3 does the same for bivariate and multivariate models. Maximum likelihood fitting of univariate and bivariate models is discussed in Sections 4.1 and 4.2 respectively. Two extended examples (one univariate and one bivariate) using the data sets `oxford` and `sealevel` (both included in the EVD package) are given in Section 4.3. Within these examples I implement various diagnostic plots and procedures.

This guide should not be viewed as an alternative to the help files included within the package. These remain the definitive source of help. All of the examples presented in this guide are called with `options(digits = 4)`.

1.3 Citing this package/document

Do you want to cite this package? No? Spoil sport.

```
@MANUAL{key,
TITLE = {A Guide to the EVD R Package (Version 1.0)},
AUTHOR = {Stephenson, A. G.},
YEAR = {2002},
NOTE = {Available from \verb+http://www.maths.lancs.ac.uk/~stephena/+}
}
```

1.4 Caveat (a.k.a the only way you can get me to buy a round)

I've checked these functions as best I can but, as ever, they may contain bugs. If you find a bug or suspected bug in the code or the documentation please report it to me at `a.stephenson@lancaster.ac.uk`. If you do find a bug and are the first person to report it, I guarantee to buy you the drink of your choice. If you ever manage to find me.

1.5 Thank-U

Thanks to Paulo Ribeiro Jr. for much needed technical advice.

2 Standard Univariate Functions

The Gumbel, Fréchet and (reversed) Weibull distribution functions are respectively given by

$$G(z) = \exp\left\{-\exp\left[-\left(\frac{z-a}{b}\right)\right]\right\}, \quad -\infty < z < \infty \tag{1}$$

$$G(z) = \begin{cases} 0, & z \le a, \\ \exp\left\{-\left(\frac{z-a}{b}\right)^{-\alpha}\right\}, & z > a, \end{cases} \tag{2}$$

$$G(z) = \begin{cases} \exp\left\{-\left[-\left(\frac{z-a}{b}\right)\right]^{\alpha}\right\}, & z < a, \\ 1, & z \ge a, \end{cases} \tag{3}$$

where a is a location parameter, $b > 0$ is a scale parameter and $\alpha > 0$ is a shape parameter. The distribution (3) is often referred to as the Weibull distribution. To avoid confusion I will call this the reversed Weibull, since it is related to the three parameter Weibull distribution used in survival analysis by a change of sign.

It is standard practice within R to concatenate the letters r, p, q and d with an abbreviated distribution name to yield the names of the corresponding simulation, distribution, quantile (inverse distribution) and density functions respectively. The EVD package follows this convention. Each distribution defined above has an associated set of functions, as shown here for the reversed Weibull.[*]

```
rrweibull(n, loc = 0, scale = 1, shape = 1)
prweibull(q, loc = 0, scale = 1, shape = 1, lower.tail = TRUE)
qrweibull(p, loc = 0, scale = 1, shape = 1, lower.tail = TRUE)
drweibull(x, loc = 0, scale = 1, shape = 1, log = FALSE)

> rrweibull(6, loc = 2, scale = .5, shape = c(1, 1.2))
[1] -0.1373  1.4492  1.5704  1.6512  1.9856  1.3174

> qrweibull(seq(0.1, 0.4, 0.1), 2, 0.5, 1, lower.tail = FALSE)
> qrweibull(seq(0.9, 0.6, -0.1), loc = 2, scale = 0.5, shape = 1)
# Both give
[1] 1.947 1.888 1.822 1.745

> prweibull(-1:3, 2, 0.5, 1)
[1] 0.002479 0.018316 0.135335 1.000000 1.000000
> prweibull(-1:3, 2, 0.5, 1, low = FALSE)
[1] 0.9975 0.9817 0.8647 0.0000 0.0000

> drweibull(-1:3, loc = 2, scale = 0.5, shape = 1)
[1] 0.004958 0.036631 0.270671 0.000000 0.000000
```

[*]The equivalent set of functions for the three parameter Weibull distribution used in survival analysis can be derived from the `rweibull` functions as follows. For simulation, negate the output. For the distribution function, reverse the logical value passed to `lower.tail` and negate the argument `q`. For the quantile function, reverse the logical value passed to `lower.tail` and negate the output. For the density function, negate the argument `x`.

```
> drweibull(-1:3, 2, 0.5, 1, log = TRUE)
[1] -5.307 -3.307 -1.307   -Inf   -Inf
```

Standard stuff. Parameters can be given as vectors (the standard recycling rules are applied). An error occurs if any of the parameters are specified outside of the valid parameter space.

The GEV (generalized extreme value) distribution function is given by

$$G(z) = \exp\left[- \left\{ 1 + \xi \left(z - \mu \right) / \sigma \right\}_+^{-1/\xi} \right],$$

where (μ, σ, ξ) are the location, scale and shape parameters respectively, $\sigma > 0$ and $h_+ = \max(h, 0)$. The parametric form of the GEV encompasses that of the Gumbel, Frèchet and reversed Weibull distributions. The Gumbel distribution is obtained in the limit as $\xi \to 0$. The Fréchet and Weibull distributions are obtained when $\xi > 0$ and $\xi < 0$ respectively. To recover the parameterization of the Fréchet distribution (2) set $\xi = 1/\alpha > 0$, $\sigma = b/\alpha > 0$ and $\mu = a + b$. To recover the parameterization of the reversed Weibull distribution (3) set $\xi = -1/\alpha < 0$, $\sigma = b/\alpha > 0$ and $\mu = a - b$. The set of functions associated with the GEV distribution should be familiar.

```
rgev(n, loc = 0, scale = 1, shape = 0)
pgev(q, loc = 0, scale = 1, shape = 0, lower.tail = TRUE)
qgev(p, loc = 0, scale = 1, shape = 0, lower.tail = TRUE)
dgev(x, loc = 0, scale = 1, shape = 0, log = FALSE)
```

These can be used in the same manner as the corresponding functions for the reversed Weibull distribution.[*]

Let F be an arbitrary distribution function, and let $X_1, \dots, X_m$ be a random sample from F. Define $U_m = \max\{X_1, \dots, X_m\}$ and $L_m = \min\{X_1, \dots, X_m\}$. The distributions of U_m and L_m are given by

$$\Pr(U_m \le x) = [F(x)]^m \tag{4}$$
$$\Pr(L_m \le x) = 1 - [1 - F(x)]^m. \tag{5}$$

Simulation, distribution, quantile and density functions for the distributions of U_m and L_m, given an integer m and an arbitrary distribution function F, are provided by[†]

```
rext(n, quantfun, ..., distn, mlen = 1, largest = TRUE)
pext(q, distnfun, ..., distn, mlen = 1, largest = TRUE, lower.tail = TRUE)
qext(p, quantfun, ..., distn, mlen = 1, largest = TRUE, lower.tail = TRUE)
dext(x, densfun, distnfun, ..., distn, mlen = 1, largest = TRUE, log = FALSE)
```

The integer m should be given to the argument `mlen`. The distribution F can be specified by passing the corresponding quantile, distribution and density functions to `quantfun`, `distnfun` and `densfun` respectively (both the distribution and density functions are required for `dext`). Alternatively, a string can be passed to `distn` such that the name of the quantile/distribution/density function is derived when prefixed by the letter q/p/d. If maxima are required use `largest = TRUE` (the default). For minima use `largest = FALSE`. Some examples should make this clear.

[*]The `shape` argument cannot be a vector.

[†]The simulation function `rext` does not literally simulate from $F(\cdot)$; the simulation speed is invariant to `mlen`.

```
> rext(4, qexp, rate = 1, mlen = 5)
> rext(4, distn = "exp", rate = 1, mlen = 5)
> rext(4, distn = "exp", mlen = 5)
# With set.seed(50) these all give
[1] 2.356 1.558 4.650 1.856

> rext(1, distn = "norm", mean = 0.5, sd = 2, mlen = 20)
[1] 3.379
# Simulates from the same distribution as
> max(rnorm(20, 0.5, 2))
[1] 4.354

> rext(1, distn="norm", sd = 2, mlen = 20, largest = FALSE)
[1] -4.482
# Simulates from the same distribution as
> min(rnorm(20, 0, 2))
[1] -4.286

> pext(c(.4, .5), distn="norm", sd = c(1, 2), mlen = 4)
> pext(c(.4, .5), distn="norm", mean = 0, sd = c(1, 2), mlen = 4)
# Both give
[1] 0.1845 0.1285

> dext(c(1, 4), distn="gamma", shape = 1, scale = 0.3, mlen = 100)
[1] 0.3261328 0.0005398
```

Parameters can be given as vectors assuming that this is implemented in the functions passed as arguments (either directly or indirectly via `distn`). If any of the parameters of F are omitted the defaults defined in the corresponding distribution/density/quantile function are used. If default values do not exist an error occurs. Although the examples above use functions provided in R, user defined functions can be specified. Density functions must have `log` arguments.*

Let $X_{(1)} \geq X_{(2)} \geq \cdots \geq X_{(m)}$ be the order statistics of the random sample $X_1, \ldots, X_m$. The distribution of the jth largest order statistic, for $j = 1, \ldots, m$, is

$$\Pr(X_{(j)} \leq x) = \sum_{k=0}^{j-1} \binom{m}{k} [F(x)]^{m-k}[1 - F(x)]^k. \tag{6}$$

The distribution of the jth smallest order statistic is obtained by setting $j = m+1-j$. Simulation, distribution and density functions for the distribution of $X_{(j)}$ for given integers m and $j \in \{1, \ldots, m\}$, and for an arbitrary distribution function F, are provided by

```
rorder(n, quantfun, ..., distn, mlen = 1, j = 1, largest = TRUE)
porder(q, distn, ..., mlen = 1, j = 1, largest = TRUE, lower.tail = TRUE)
dorder(x, densfun, distnfun, ..., distn, mlen = 1, j = 1, largest = TRUE,
       log = FALSE)
```

The integer m should again be given to the argument `mlen`. If `largest = TRUE` (the default) the distribution of the `j`th largest order statistic $X_{(j)}$ is used. If `largest = FALSE` the distribution of the `j`th smallest order statistic $X_{(m+j-1)}$ is used.

*A simple wrapper can always be constructed to achieve this.

For computational reasons it is better to specify `j` to be an integer in the interval [1, `ceiling(mlen/2)`]. This can always be achieved using the argument **`largest`**. Some examples? Okay then.

```
> rorder(1, distn = "norm", mlen = 20, j = 2)
> rorder(1, distn = "norm", mlen = 20, j = 19, largest = FALSE)
# Simulates the second largest order statistic from 20 standard normals
# With set.seed(50) both give
[1] 1.685

> porder(c(1, 2), distn="gamma", shape =c(.5, .7), mlen = 10, j = 2)
[1] 0.5177 0.8259
> dorder(c(1, 2), distn="gamma", shape =c(.5, .7), mlen = 10, j = 2)
[1] 0.7473 0.3081
```

3 Standard Bivariate and Multivariate Functions

The EVD package contains functions associated with the following three symmetric bivariate distributions

$$G(z_1, z_2) = \exp\left[-(y_1^{1/\alpha} + y_2^{1/\alpha})^\alpha\right], \quad 0 < \alpha \leq 1,$$
$$G(z_1, z_2) = \exp\left[-y_1 - y_2 + (y_1^{-r} + y_2^{-r})^{-1/r}\right], \quad r > 0,$$
$$G(z_1, z_2) = \exp\left(-y_1\Phi\{\lambda^{-1} + \tfrac{1}{2}\lambda[\log(y_1/y_2)]\} - y_2\Phi\{\lambda^{-1} + \tfrac{1}{2}\lambda[\log(y_2/y_1)]\}\right), \quad \lambda > 0,$$

known as the (symmetric) logistic (Gumbel, 1960, 1961), (symmetric) negative logistic (Galambos, 1975) and Hüsler-Reiss (Hüsler and Reiss, 1989) models respectively, where

$$y_j = y_j(z_j) = \{1 + \xi_j(z_j - \mu_j)/\sigma_j\}_+^{-1/\xi_j} \tag{7}$$

for $j = 1, 2$. The jth univariate marginal distribution of each model is GEV, with parameters (μ_j, σ_j, ξ_j), where $\sigma_j > 0$. Independence* is obtained when $\alpha = 1$, $r \downarrow 0$ or $\lambda \downarrow 0$. Complete dependence† is obtained when $\alpha \downarrow 0$, $r \to \infty$ or $\lambda \to \infty$.

The package also contains functions for the asymmetric distributions

$$G(z_1, z_2) = \exp\left\{-(1-\theta_1)y_1 - (1-\theta_2)y_2 - [(\theta_1 y_1)^{1/\alpha} + (\theta_2 y_2)^{1/\alpha}]^\alpha\right\}, \quad 0 < \alpha \leq 1,$$
$$G(z_1, z_2) = \exp\left\{-y_1 - y_2 + [(\theta_1 y_1)^{-r} + (\theta_2 y_2)^{-r}]^{-1/r}\right\}, \quad r > 0,$$

known as the asymmetric logistic (Tawn, 1988) and asymmetric negative logistic (Joe, 1990) models respectively, where the asymmetry parameters $0 \leq \theta_1, \theta_2 \leq 1$, and where (y_1, y_2) are again defined by (7). The univariate margins are again GEV.

These models (in fact, any bivariate extreme value distribution function) can be represented in the form

$$G(z_1, z_2) = \exp\left[-(y_1 + y_2)A\left(\frac{y_1}{y_1 + y_2}\right)\right],$$

so that $A(\omega) = -\log[G(\omega, 1-\omega)]$, defined on $0 \leq \omega \leq 1$. $A(\cdot)$ is called (by some authors) the dependence function.‡ It follows that $A(0) = A(1) = 1$, and that $A(\cdot)$ is a convex function with $\max(\omega, 1-\omega) \leq A(\omega) \leq 1$ for all $0 \leq \omega \leq 1$.

Each model has a set of functions of the type given here for the asymmetric logistic distribution.

*Independence occurs when $G(z_1, z_2) = \exp[-(y_1 + y_2)]$.

†Complete dependence occurs when $G(z_1, z_2) = \exp[-\max(y_1, y_2)]$.

‡This is not quite the same as e.g. Pickands (1981), who uses $A(\omega) = -\log[G(1-\omega, \omega)]$.

```
rbvalog(n, dep, asy, mar1 = c(1, 1, 0), mar2 = mar1)
pbvalog(q, dep, asy, mar1 = c(1, 1, 0), mar2 = mar1)
dbvalog(x, dep, asy, mar1 = c(1, 1, 0), mar2 = mar1, log = FALSE)
abvalog(x = 0.5, dep, asy, plot = FALSE, border = TRUE, add = FALSE, lty = 1,
        blty = 3, xlim = c(0, 1), ylim = c(0.5, 1), xlab = "", ylab = "", ...)
```

The first three functions, for simulation and for the calculation of the distribution and density functions, are standard. The arguments `q` and `x` in `pbvalog` and `dbvalog` respectively should be vectors of length two or matrices with two columns, so that each row specifies a value for (z_1, z_2). The argument `dep` is the dependence parameter (in this case α, but the same formal argument represents r and λ). The argument `asy` is a vector containing the asymmetry parameters (θ_1, θ_2). The marginal parameters (μ_1, σ_1, ξ_1) and (μ_2, σ_2, ξ_2) should be passed to `mar1` and `mar2` respectively. The arguments `mar1` and `mar2` can also be given as matrices with three columns, in which case each column represents a vector of values to be passed to the corresponding marginal parameter (the standard recycling rules are applied). Vector/matrix arguments for `dep` and `asy` are not implemented.

The `abvalog` function calculates (by default) or plots[*] the dependence function $A(\cdot)$. The value $A(1/2) \in [0.5, 1]$ is returned by default. The lower and upper end points of $[0.5, 1]$ are obtained by $A(1/2)$ at complete dependence and at independence respectively.

```
> rbvalog(3, dep = .8, asy = c(.4, 1))
       [,1]  [,2]
[1,] 0.2242 2.230
[2,] 1.5712 2.593
[3,] 2.7550 1.830

> rbvalog(3, dep = .8, asy = c(.4, 1), mar1 = c(1, 1, 1))
        [,1]  [,2]
[1,]  1.7686 0.908
[2,] 22.5297 7.489
[3,]  0.5617 1.057

> pbvalog(c(1, 1.2), dep = .4, asy = c(.4, .6), mar1 = c(1, 1, 1))
[1] 0.216
> tmp.quant <- matrix(c(1,1.2,1,2),ncol = 2, byrow = TRUE)
> tmp.mar <- matrix(c(1,1,1,1.2,1.2,1.2), ncol = 3, byrow = TRUE)
> pbvalog(tmp.quant, dep = .4, asy = c(.4, .6), mar1 = tmp.mar)
[1] 0.2160 0.2153

> dbvalog(c(1, 1.2), dep = .4, asy = c(.4, .6), mar1 = c(1, 1, 1))
[1] 0.1427
> dbvalog(tmp.quant, dep = .4, asy = c(.4, .6), mar1 = tmp.mar)
[1] 0.1427 0.0696

> abvalog(dep = .3, asy= c(.7, .9))
[1] 0.7013
> abvalog(seq(0, 1, 0.25), dep = .3, asy = c(.7, .9))
[1] 1.0000 0.8272 0.7013 0.7842 1.0000

> abvalog(dep = .3, asy = c(.5, .9), plot = TRUE, blty = 1)
```

[*]If plotted, the values used for the plot are returned invisibly.

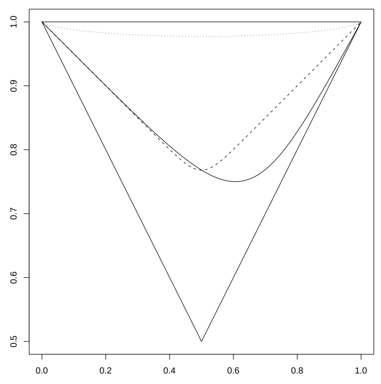

Figure 1: Dependence functions for various bivariate extreme value distributions. All dependence functions must be convex and must lie within the triangular region.

```
> abvalog(dep = .1, asy = c(.5, .5), add = TRUE, lty = 2)
> abvhr(dep = .5, add = TRUE, lty = 3)
```

The last three lines of code produce Figure 1. The function `abvhr` plots the dependence function for the Hüsler-Reiss model. The simulation functions `rbvneglog`, `rbvaneglog` and `rbvhr` for the negative logistic models and for the Hüsler-Reiss model respectively use a root finding algorithm for each realization and are therefore relatively slow. Ghoudi *et al.* (1998) derive a simulation method for the negative logistic model which is not currently implemented. The simulation functions `rbvlog` and `rbvalog` use bivariate versions of Algorithms 1.1 and 1.2 in Stephenson (2002) respectively.

Let $z = (z_1, \ldots, z_d)$. The d-dimensional (symmetric) logistic distribution function (Gumbel, 1960) is given by

$$G(z) = \exp\left[-\left(\sum\nolimits_{j=1}^{d} y_j^{-1/\alpha}\right)^{\alpha}\right]$$

where $\alpha \in (0, 1]$ and $(y_1, \ldots, y_d)$ is defined by the transformations (7).

This distribution can be extended to an asymmetric form. Let B be the set of all non-empty subsets of $\{1, \ldots, d\}$, let $B_1 = \{b \in B : |b| = 1\}$ and let $B_{(i)} = \{b \in B : i \in b\}$. The multivariate asymmetric logistic model (Tawn, 1990) is given by

$$G(z) = \exp\left\{-\sum\nolimits_{b \in B}\left[\sum\nolimits_{i \in b}(\theta_{i,b} y_i)^{1/\alpha_b}\right]^{\alpha_b}\right\}$$

where the dependence parameters $\alpha_b \in (0, 1]$ for all $b \in B \setminus B_1$, and the asymmetry parameters $\theta_{i,b} \in [0, 1]$ for all $b \in B$ and $i \in b$. The constraints $\sum_{b \in B_{(i)}} \theta_{i,b} = 1$ for $i = 1, \ldots, d$ ensure that the marginal distributions GEV. There exists further constraints which arise from the possible redundancy of asymmetry parameters in the expansion of the distributional form. Specifically, if $\alpha_b = 1$ for some $b \in B \setminus B_1$ then $\theta_{i,b} = 0$ for all $i \in b$. Let $b_{-i_0} = \{i \in b : i \neq i_0\}$. If, for some $b \in B \setminus B_1$, $\theta_{i,b} = 0$ for all $i \in b_{-i_0}$ then $\theta_{i_0,b} = 0$. The model contains $2^d - d - 1$ dependence parameters and $d2^{d-1}$ asymmetry parameters (excluding the constraints).

The EVD package provides the following functions for simulating from and calculating the distribution function of these models.

```
rmvlog(n, dep, d = 2, mar = c(1, 1, 0))
pmvlog(q, dep, d = 2, mar = c(1, 1, 0))
rmvalog(n, dep, asy, d = 2, mar = c(1, 1, 0))
pmvalog(q, dep, asy, d = 2, mar = c(1, 1, 0))
```

The argument `mar` represents the GEV marginal parameters for every univariate margin, and may again be a matrix. The interface for `rmvalog`/`pmvalog` is not particularly good. If anybody uses `rmvalog` or `pmvalog` above about five dimensions it may inspire me to make improvements. Possibly. The simulation functions `rmvlog` and `rmvalog` use Algorithms 2.1 and 2.2 in Stephenson (2002) respectively.

For the symmetric model $\texttt{dep} = \alpha$. For the asymmetric model `dep` should be a vector of length $2^{\texttt{d}} - \texttt{d} - 1$ containing the dependence parameters. Specifically, when $\texttt{d} = 4$

$$\texttt{dep} = (\alpha_{12}, \alpha_{13}, \alpha_{14}, \alpha_{23}, \alpha_{24}, \alpha_{34}, \alpha_{123}, \alpha_{124}, \alpha_{134}, \alpha_{234}, \alpha_{1234}).$$

The asymmetry parameters should be passed to `asy` in a list with $2^{\texttt{d}} - 1$ elements, where each element is a vector* corresponding to a set $b \in B$, containing $\{\theta_{i,b} : i \in b\}$. Specifically, when $\texttt{d} = 4$

$$\begin{aligned}\texttt{asy} = \texttt{list}(&\theta_{1,1}, \theta_{2,2}, \theta_{3,3}, \theta_{4,4}, \texttt{c}(\theta_{1,12}, \theta_{2,12}), \texttt{c}(\theta_{1,13}, \theta_{3,13}), \texttt{c}(\theta_{1,14}, \theta_{4,14}), \texttt{c}(\theta_{2,23}, \theta_{3,23}),\\ &\texttt{c}(\theta_{2,24}, \theta_{4,24}), \texttt{c}(\theta_{3,34}, \theta_{4,34}), \texttt{c}(\theta_{1,123}, \theta_{2,123}, \theta_{3,123}), \texttt{c}(\theta_{1,124}, \theta_{2,124}, \theta_{4,124}),\\ &\texttt{c}(\theta_{1,134}, \theta_{3,134}, \theta_{4,134}), \texttt{c}(\theta_{2,234}, \theta_{3,234}, \theta_{4,234}), \texttt{c}(\theta_{1,1234}, \theta_{2,1234}, \theta_{3,1234}, \theta_{4,1234})).\end{aligned}$$

All the constraints, including $\sum_{b \in B_{(i)}} \theta_{i,b} = 1$ for $i = 1, \ldots, d$, must be satisfied or an error will occur.

The code given below illustrates the symmetric logistic functions `rmvlog` and `pmvalog`.

```
> rmvlog(3, dep = .6, d = 5)
      [,1]    [,2]    [,3]  [,4]   [,5]
[1,] 1.738  1.1709  2.4161 2.249 1.7047
[2,] 4.052 -0.1505 -0.2324 2.063 1.3453
[3,] 0.886  0.9965  0.4698 0.788 0.7828

> tmp.mar <- matrix(c(1,1,1,1,1,1.5,1,1,2), ncol = 3, byrow = TRUE)
> rmvlog(3, dep = .6, d = 5, mar = tmp.mar)
       [,1]  [,2]   [,3]   [,4]  [,5]
[1,] 2.8440 2.700 2.1742 1.1078 1.213
[2,] 0.7029 2.894 0.5864 0.4811 0.851
[3,] 3.0829 1.810 3.5614 3.5234 1.927

> tmp.quant <- matrix(rep(c(1,1.5,2), 5), ncol = 5)
> pmvlog(tmp.quant, dep = .6, d = 5, mar = tmp.mar)
[1] 0.07233 0.16387 0.21949
```

The dependence parameters used in the following trivariate asymmetric logistic model are $(\alpha_{12}, \alpha_{13}, \alpha_{23}, \alpha_{123}) = (.6, .5, .8, .3)$. The asymmetry parameters are $\theta_{1,1} = .4$, $\theta_{2,2} = 0$, $\theta_{3,3} = .6$, $(\theta_{1,12}, \theta_{2,12}) = (.3, .2)$, $(\theta_{1,13}, \theta_{3,13}) = (.1, .1)$, $(\theta_{2,23}, \theta_{3,23}) = (.4, .1)$ and finally $(\theta_{1,123}, \theta_{2,123}, \theta_{3,123}) = (.2, .4, .1)$. Notice that the constraints are satisfied.

*Including vectors of length one.

```
> rmvalog(3, dep = c(.6,.5,.8,.3), asy =
  list(.4,0,.6,c(.3,.2),c(.1,.1),c(.4,.1),c(.2,.4,.2)), d = 3)
       [,1]   [,2]  [,3]
[1,] 4.0022 2.4301 1.925
[2,] 0.9138 0.2474 1.019
[3,] 1.5825 0.5251 0.897

> pmvalog(c(2, 2, 2), dep = c(.6,.5,.8,.3), asy =
  list(.4,.0,.6,c(.3,.2),c(.1,.1),c(.4,.1),c(.2,.4,.2)), d = 3)
[1] 0.4131

> tmp.quant <- matrix(rep(c(1,1.5,2), 3), ncol = 3)
> pmvalog(tmp.quant, dep = c(.6,.5,.8,.3), asy =
  list(.4,.0,.6,c(.3,.2),c(.1,.1),c(.4,.1),c(.2,.4,.2)), d = 3)
[1] 0.09042 0.23277 0.41307
```

The dependence parameters used in the following four dimensional asymmetric logistic model are $\alpha_b = 1$ for $|b| = 2$* and $(\alpha_{123}, \alpha_{124}, \alpha_{134}, \alpha_{234}, \alpha_{1234}) = (.7, .3, .8, .7, .5)$. The asymmetry parameters are $\theta_{i,b} = 0$ for all $i \in b$ such that $|b| \leq 2$, $(\theta_{1,123}, \theta_{2,123}, \theta_{3,123}) = (.2, .1, .2)$, $(\theta_{1,124}, \theta_{2,124}, \theta_{4,124}) = (.1, .1, .2)$, ..., and $(\theta_{1,1234}, \theta_{2,1234}, \theta_{3,1234}, \theta_{4,1234}) = (.4, .6, .2, .5)$.

```
> rmvalog(3, dep = c(rep(1,6),.7,.3,.8,.7,.5), asy =
  list(0, 0, 0, 0, c(0,0), c(0,0), c(0,0), c(0,0), c(0,0), c(0,0),
  c(.2,.1,.2), c(.1,.1,.2), c(.3,.4,.1), c(.2,.2,.2), c(.4,.6,.2,.5)), d = 4)
      [,1]  [,2]    [,3]  [,4]
[1,] 1.960 2.111  1.2831 1.427
[2,] 3.337 3.512  1.4408 2.470
[3,] 1.394 1.897 -0.2020 2.543
```

The following examples may be helpful in deciphering errors.

```
> rmvalog(3, dep = c(.6,.5,.8,.3), asy =
  list(.4,0,.5,c(.3,.2),c(.1,.1),c(.4,.1),c(.2,.4,.2)), d = 3)
Error in rmvalog(3, dep = c(0.6, 0.5, 0.8, 0.3), asy = list(0.4, 0, 0.5,  :
        ‘asy’ does not satisfy the appropriate constraints

# 0.5 + 0.1 + 0.1 + 0.2 does not equal one; the sum constraint on the third
margin is not satisfied

> rmvalog(3, dep = c(.6,1,.8,.3), asy =
  list(.4,0,.6,c(.3,.2),c(.1,.1),c(.4,.1),c(.2,.4,.2)), d = 3)
Error in rmvalog(3, dep = c(0.6, 1, 0.8, 0.3), asy = list(0.4, 0, 0.6,  :
        ‘asy’ does not satisfy the appropriate constraints

# A dependence parameter is equal to one but the corresponding asymmetry
parameters are not zero (the first ‘further constraint’).
# One possible alternative which preserves dep (and still satisfies the sum
constraints) is
```

*The values taken by α_b when $|b| = 2$ are irrelevant here because $\theta_{i,b} = 0$ for all $i \in b$ such that $|b| = 2$; they do not therefore appear in the expansion of the distributional form (I arbitrarily set them to one). Test this yourself using `set.seed` to set the seed of the random generator.

```
> rmvalog(3, dep = c(.6,1,.8,.3), asy =
  list(.4,0,.6,c(.3,.2),c(.0,.0),c(.4,.1),c(.3,.4,.3)), d = 3)
      [,1]  [,2]   [,3]
[1,] 1.814 2.097 0.6302
[2,] 1.007 3.435 0.8756
[3,] 1.279 1.441 1.3642

> rmvalog(3, dep = c(.6,.5,.8,.3), asy =
  list(.5,0,.6,c(.3,.2),c(0,.1),c(.4,.1),c(.2,.4,.2)), d = 3)
Error in rmvalog(3, dep = c(0.6, 0.5, 0.8, 0.3), asy = list(0.5, 0, 0.6,  :
        'asy' does not satisfy the appropriate constraints

# The fifth element in asy contains exacly one non-zero asymmetry parameter
(the second 'further constraint')

> rmvalog(3, dep = c(.6,.5,.8,.3), asy =
  list(.4,0,.6,c(.3,.2),c(.1,.1),c(.4,.1),c(.2,.4,.2)))
Error in rmvalog(3, dep = c(0.6, 0.5, 0.8, 0.3), asy = list(0.4, 0, 0.6,  :
        'asy' is not of the correct form

# Actually, asy is not of the correct form only because the dimension has
been mis-specified (the default dimension is 2)
```

4 Fitting Distributions by Maximum Likelihood

This section presents the functions which produce maximum likelihood estimates for the distributions given previously. The fitting functions derive maximum likelihood estimates using the general purpose optimization function `optim`. This requires a fairly recent version of R. Univariate and bivariate distributions are discussed Sections 4.1 and 4.2 respectively. For illustrative purposes Sections 4.1 and 4.2 use only simulated data. Two extended examples (one univariate and one bivariate) using the data sets `oxford` and `sealevel` (both included in the EVD package) are given in Section 4.3. Within these examples I implement various diagnostic plots and procedures, and show how to choose good starting values for the optimization.

The maximum likelihood estimators of the GEV parameters do not necessarily have the usual asymptotic properties, since the (lower or upper) end point of the GEV distribution (given by $\mu - \sigma/\xi$) depends on the parameters. Smith (1985) shows that the usual asymptotic properties hold when $\xi > -0.5$. Fortunately, this is often the case.* When $-1 < \xi \leq -0.5$ the maximum likelihood estimators do not have the standard asymptotic properties, but generally exist. When $\xi \leq -1$ maximum likelihood estimators do not often exist.†‡

4.1 Univariate Fitting

The univariate fitting functions are§

*For environmental data sets, at least.

†This occurs because of the short tail (i.e. the large mass at the lower end point). The likelihood increases without bound as the lower end point is estimated to be closer and closer to the minimum of the data.

‡In terms of the Weibull shape parameter α, the usual asymptotic properties hold when $\alpha > 2$, the asymptotic properties are not standard for $1 < \alpha \leq 2$, and maximum likelihood estimators do not often exist for $\alpha < 1$.

§The first four functions are simple wrappers to `fdensfun` which is based on `fitdistr` in package MASS.

```
ffrechet(x, start, ...)
frweibull(x, start, ...)
fgumbel(x, start, ...)
fgev(x, start, ...)
fext(x, start, densfun, distnfun, ..., distn, mlen = 1, largest = TRUE)
forder(x, start, densfun, distnfun, ..., distn, mlen = 1, j = 1,
       largest = TRUE)
```

The argument `x` should be given a numeric vector containing data to be fitted. `start` should be a named list containing starting values, the names of which should be the parameters over which the likelihood is to be maximized. If some of the parameters are to be set to fixed values, they can be given as separate arguments. Any of the arguments that can be passed to `optim` can also be specified.

```
> data <- rgev(1000, loc = 0.13, scale = 1.1, shape = 0.2)
> fgev(data, start = list(loc = 0, scale = 1, shape = 0), method = "BFGS",
  control = list(trace = 1))
# Minimize the negative log-likelihood
initial  value 1956.771912
final  value 1830.364841
converged
$estimate
   loc  scale  shape
0.1594 1.1422 0.2101

$std.err
    loc   scale   shape
0.04088 0.03265 0.02549

$deviance
[1] 3661

$counts
function gradient
      39        9

> fgev(data, start = list(loc = 0, scale = 1), shape = 0, method = "BFGS")
$estimate
   loc  scale
0.3042 1.2836

$std.err
    loc   scale
0.04234 0.03341

$deviance
[1] 3778

$counts
function gradient
      34       10
```

```
> fgev(data, start = list(loc = 0), scale = 1, shape = 0)
> fgev(data, start = list(loc = 0))
# Both are equivalent, since the default values for the scale and
# shape arguments in dgev are one and zero respectively
$estimate
   loc
0.1844

$std.err
    loc
0.03162

$deviance
[1] 3882

$counts
function gradient
      20       NA
```

The first example maximizes over (`loc`, `scale shape`), using starting values $(0, 1, 0)$. The second example maximizes over (`loc`, `scale`), with the shape parameter fixed at zero, and the third maximizes over `loc`, with the shape and scale parameters fixed at zero and one respectively. In the third example the `shape` and `scale` arguments are not specified. When a parameter is not specified, it is taken to be fixed at the default value in the corresponding density function.

The value returned by the fitting function is a list of four elements. The `estimate` and `std.err` components give maximum likelihood estimates and their standard errors[*] (approximated using the inverse of the observed information matrix). The `deviance` component is minus twice the log-likelihood function. In the first example the minimum deviance is given by

```
> -2*sum(dgev(data, loc = 0.1594, scale = 1.1422, shape = 0.2101, log = TRUE))
[1] 3661
```

The `deviance` component can be used to compare models. Likelihood ratio tests can be performed by comparing the difference between the deviances of nested models with chi-squared distributions.[†] For example, $3778 - 3661 = 117$ yields a very small p-value when compared with a chi-squared distribution on one degree of freedom, so the evidence for a Gumbel distribution (zero shape) is non-existent. This is not surprising since the 95% confidence interval for the shape parameter given by the full fit is $(0.16, 0.26)$.[‡] Confidence intervals based on profile likelihoods are discussed in Section 4.3.1.

The `counts` component returns the number of function and gradient evaluations[§] used by `optim`. The arguments `method` and `control` are passed to `optim`. The default method used by `optim` 'uses only function values and is robust but relatively slow'. I prefer (in the first instance) the method `BFGS`, which uses gradient evaluations and tends to be much faster for more complex examples. The default maximum number of *gradient* evaluations used for derivative based optimization

[*] Assuming the global maximum has been found. A range of starting values can be used to examine the space fully.

[†] There are exceptions (see Section 4.3.2).

[‡] Specifically, the confidence interval does not contain zero. A Wald test would therefore be rejected at significance level 0.05.

[§] Finite-difference approximations.

methods is 100 and is rarely reached.[*] For non-derivative based methods the default maximum number of *function* evaluations is 500, which is often reached. If it has been reached set it to a higher value using the named list element `maxit` in the argument `control`. There are numerous options that can be passed to `optim` through `control`. See the help page for details.

The `fgumbel` function is a simple wrapper to `fgev`. The models

```
> somedata <- rgev(100,1,1,0.05)
> fgumbel(somedata, start = list(loc = 0.5, scale = 2))
> fgev(somedata, start = list(loc = 0.5, scale = 2))
```

are equivalent.

GEV estimates can be compared with Fréchet and Weibull estimates.

```
> ffrechet(data, start = list(loc=-2,scale=1,shape=5), method ="BFGS")
$estimate
   loc  scale  shape
-5.278  5.437  4.759

$std.err
   loc  scale  shape
0.6711 0.6872 0.5779

$deviance
[1] 3661

$counts
function gradient
      40       15
```

Compare this example with the full GEV fit. The deviances are the same. I am fitting the same model, but under a different parameterization. The relationship between the parameter estimates was given in Section 2. For example, the shape parameter estimate in the Fréchet model is the inverse of the shape parameter estimate in the GEV model ($4.759 \approx 1/0.2101$). The following snippet shows the analogous relationship with `frweibull`.

```
> data2 <- rgev(1000, loc = 0.13, scale = 1.1, shape = -0.25)
> frweibull(data2, start = list(loc = 5, scale = 5, shape = 5), method="BFGS")
$estimate
  loc scale shape
4.507 4.403 4.077

$std.err
   loc  scale  shape
0.2837 0.2965 0.3134

$deviance
[1] 3026

$counts
```

[*]So rare that `optim` gives no warning when it is reached.

```
function gradient
      37       12

> fgev(data2, start = list(loc=0.13,scale=1.1,shape=0), method="BFGS")
$estimate
    loc   scale   shape
 0.1036  1.0800 -0.2451

$std.err
    loc   scale   shape
0.03748 0.02607 0.01887

$deviance
[1] 3026

$counts
function gradient
      74       12
```

The `fext` function works slightly differently. It yields maximum likelihood estimates for the distributions (4) and (5) given an integer m and an arbitrary distribution function F.

```
fext(x, start, densfun, distnfun, ..., distn, mlen = 1, largest = TRUE)
```

If `yearly.maxima.data` contains yearly maxima taken from a process observed daily (with the daily observations independent, or at least approximately so), a normal distribution can be fitted to the (unrecorded or unavailable) daily observations as follows

```
fext(yearly.maxima.data, list(mean = 0, sd = 1), distn = "norm", mlen = 365)
```

The arguments `densfun`, `distnfun`, `distn`, `mlen` and `largest` are the same as those used in the density function `dext` (see Section 2). The argument `x` should be a numeric vector containing the data to be fitted, which should be assumed to represent maxima (if `largest` is `TRUE`, the default) or minima (if `largest` is `FALSE`). `start` should be a named list containing starting values, the names of which should be the parameters over which the likelihood is to be maximized. If some of the parameters are to be set to fixed values, they can be given as separate arguments. If parameters are missing, they are fixed at their default values specified in the density/distribution function. Any of the arguments that can be given to `optim` can also be specified.

The optimizer will be effected by the way in which the density and distribution functions passed to `fext` behave when given values outside of the valid parameter space. Functions in R base generally produce `NA` values which may result in warnings being printed which can usually be ignored.*

If the density and distribution functions are user defined, the order of the arguments must mimic those in R base (i.e. data first, parameters second). The density function must have a `log` argument.

```
> data3 <- rext(100, qnorm, mean = 0.56, mlen = 365)
# Simulate yearly maxima using normal distribution
```

*See the help page for `optim` if you wish to avoid this.

```
> fext(data3, list(mean = 0, sd = 1), distn = "norm", mlen = 365)
$estimate
  mean     sd
0.5876 0.9833

$std.err
   mean      sd
0.20215 0.07374

$deviance
[1] 78.91

$counts
function gradient
      61       NA

> fext(data3, list(rate = 1), distn = "exp", mlen = 365)
# Incorrectly fit exponential distribution
$estimate
 rate
1.785

$std.err
   rate
0.03165

$deviance
[1] 123.9

$counts
function gradient
      26       NA

> fext(data3, list(scale = 1), shape = 0.5, distn = "gamma", mlen = 365)
# Incorrectly fit gamma distribution with shape fixed at 0.5
$estimate
 scale
0.7323

$std.err
  scale
0.01560

$deviance
[1] 149.7

$counts
function gradient
      28       NA
```

The `forder` function yields maximum likelihood estimates for the distribution (6) given integers m and $j \in \{1, \dots, m\}$, and an arbitrary distribution function F.

```
forder(x, start, densfun, distnfun, ..., distn, mlen = 1, j = 1,
largest = TRUE)
```

The arguments `densfun`, `distnfun`, `distn`, `mlen`, `j` and `largest` are the same as those used in the density function `dorder` (see Section 2). The argument `x` should be a numeric vector containing the data to be fitted, and `start` should again be a named list containing starting values. And no, I have never encountered a data set where it would be of much use either, but it fitted easily within the design of the package!

4.2 Bivariate Fitting

The bivariate fitting functions are

```
fbvlog(x, start, ...)
fbvalog(x, start, ...)
fbvhr(x, start, ...)
fbvneglog(x, start, ...)
fbvaneglog(x, start, ...)
```

The argument `x` should be given a numeric matrix (or a data frame) containing two columns of data to be fitted. `start` should be a named list containing starting values, the names of which should be the parameters over which the likelihood is to be maximized. These names can be any or possibly all of `dep`, `asy` (a vector of length two), `mar1` and `mar2` (vectors of length three). Alternatively, `mar1` can be passed as three separate arguments; `loc1`, `scale1` and `shape1`. `mar2` can similarly be passed as three separate arguments; `loc2`, `scale2` and `shape2`. The asymmetry parameters can also be given separately as `asy1` and `asy2`. As usual, any arguments that can passed to `optim` can be specified. The following example yields maximum likelihood estimates for the symmetric logistic model.

```
> bvdata <- rbvlog(100, dep = 0.6, mar1 = c(1.2,1.4,0.4),
  mar2 = c(1.2,1.4,0.4))
# Simulate data

> fbvlog(bvdata, start = list(mar1 = c(2,1,0), mar2 = c(1,1,0), dep = 0.75),
  control = list(maxit = 2000))
> fbvlog(bvdata, start = list(loc1=2, scale1=1, shape1=0, loc2=1, scale2=1,
  shape2=0, dep=0.75), control = list(maxit = 2000))
# Both eventually give the following output
$estimate
  loc1 scale1 shape1   loc2 scale2 shape2    dep
0.9104 1.2590 0.3652 0.9784 1.2315 0.4569 0.6495

$std.err
   loc1   scale1  shape1    loc2  scale2  shape2     dep
0.14052 0.12319 0.08053 0.14138 0.13158 0.09832 0.06020

$deviance
```

```
[1] 777.9

$counts
function gradient
     626       NA

> fbvlog(bvdata, start = list(mar1 = c(2,1,0), mar2 = c(1,1,0), dep = 0.75),
  method = "BFGS")$counts
# This method is faster
function gradient
      85       28
```

Associating a separate name with each parameter allows any parameter subset to be fixed at specified values. If parameters are to be set to fixed values, they can be given as separate arguments. All parameters with fixed values must be passed individually (e.g. to fix the parameters on the first margin all of `loc1`, `scale1` and `shape1` must be specified; using `mar1` to specify all three simultaneously results in an error).

```
> fbvlog(bvdata, start = list(mar1 = c(2,1,0), mar2 = c(1,1,0)), dep = 1,
  method="BFGS")
# Fix the dependence parameter at one
$estimate
  loc1 scale1 shape1   loc2 scale2 shape2
0.9080 1.2482 0.3435 0.9822 1.2438 0.4832

$std.err
   loc1  scale1  shape1    loc2  scale2  shape2
0.14173 0.12214 0.08596 0.14321 0.13580 0.10192

$deviance
[1] 813.1

$counts
function gradient
      71       24

# Marginal GEV estimates
> fgev(bvdata[,1], start = list(loc=1,scale=1,shape=0), method="BFGS")$estimate
   loc  scale  shape
0.9079 1.2482 0.3434

> fgev(bvdata[,2], start = list(loc=1,scale=1,shape=0), method="BFGS")$estimate
   loc  scale  shape
0.9821 1.2438 0.4832
```

This example shows how to fix the dependence parameter at one (at independence). The estimates are seen to tally with the separate marginal GEV fits. In real (data) examples this is a good way of finding reasonable starting values on the margins (see Section 4.3.2).

To illustrate fixed parameters the following snippet performs likelihood ratio tests.[*]

[*]These are for illustrative purposes only. I know what the process generating the data is!

```
# Test for Gumbel (zero shape) margins
# Deviance increses by 109.6
# P-value is pchisq(109.6, 2, low = FALSE) = 1.587e-24
# Hypothesis rejected

> fbvlog(bvdata, start = list(loc1=1, scale1=1, loc2=1, scale2=1, dep=0.75),
  shape1 = 0, shape2 = 0, method = "BFGS")
$estimate
  loc1 scale1   loc2 scale2    dep
1.2422 1.7526 1.3554 1.6286 0.8431

$std.err
   loc1  scale1    loc2  scale2     dep
0.18019 0.15039 0.16808 0.13883 0.05581

$deviance
[1] 887.5

$counts
function gradient
      29       12

> pchisq(109.6, 2, low = FALSE)
[1] 1.587e-24

# Test whether the parameters on the first margin are (1,1.5,0.5)
# Deviance increses by 4.3
# P-value is pchisq(4.3, 3, low = FALSE) = 0.2308

> fbvlog(bvdata, start = list(mar2 = c(1,1,0), dep = 0.75), loc1 = 1,
  scale1 = 1.5, shape1 = 0.5, method = "BFGS")
# Do not use mar1 = c(1,1.5,0.5)
# It will produce an error
$estimate
  loc2 scale2 shape2    dep
1.0552 1.3686 0.5359 0.5962

$std.err
   loc2  scale2  shape2     dep
0.12982 0.12808 0.10190 0.05638

$deviance
[1] 782.2

$counts
function gradient
      61       16

> pchisq(4.3, 3, low = FALSE)
[1] 0.2308
```

There follows a few examples of asymmetric logistic fits. I use the same simulated data set, distributed as symmetric logistic (where both asymmetry parameters equal one). This illustrates the difficulties that arise when the fit is on the edge of the parameter space. Although I have made a distinction between the asymmetry and dependence parameters, they all affect the dependence structure, and are highly correlated. Try plotting the fitted dependence function with `abvalog`. The likelihood function is complex, and you must ensure good starting values (see Section 4.3.2).

```
> fbvalog(bvdata, start = list(mar1 = c(1,1,0), mar2 = c(1,1,0),
  asy = c(.7,.7), dep = 0.75), method = "BFGS")
$estimate
  loc1 scale1 shape1   loc2 scale2 shape2   asy1   asy2    dep
0.9130 1.1753 0.2340 1.0240 1.1933 0.2639 0.9994 0.7233 0.5849

$std.err
     loc1    scale1    shape1      loc2    scale2    shape2      asy1      asy2
1.309e-01 1.045e-01 5.158e-02 1.408e-01 1.211e-01 7.490e-02 2.000e-06 1.524e-01
      dep
7.429e-02

$deviance
[1] 782.1

$counts
function gradient
     144       18
```

There are problems here. The first asymmetry parameter is almost one. This will cause difficulties for the optimizer. A lower deviance can be obtained by setting this parameter to one. There are two approaches. The first approach fixes the parameter directly, while updating the starting values.

```
> fbvalog(bvdata, start = list(mar1 = c(0.9,1.2,0.2), mar2 = c(1,1.2,0.3),
  asy2 = .72, dep = 0.58), asy1 = 1, method = "BFGS")
$estimate
  loc1 scale1 shape1   loc2 scale2 shape2   asy2    dep
0.8808 1.2381 0.3826 1.0037 1.2702 0.4598 0.7013 0.5362

$std.err
   loc1   scale1  shape1    loc2  scale2  shape2    asy2     dep
0.13948 0.12219 0.08068 0.14631 0.13809 0.10182 0.14706 0.07484

$deviance
[1] 774.3

$counts
function gradient
      46       14
```

This now appears to be a global maxima. The standard errors relate to the reduced model. The second approach uses the optimization method `L-BFGS-B`, which can incorporate box-constraints using the arguments `lower` and `upper`.

```
> fbvalog(bvdata, start = list(mar1 = c(0.9,1.2,0.2), mar2 = c(1,1.2,0.3),
  asy = c(.99,.72), dep = 0.58), method = "L-BFGS-B", lower =
  c(rep(-Inf, 6), 0, 0, -Inf), upper = c(rep(Inf, 6), 1, 1, 1))
$estimate
  loc1 scale1 shape1   loc2 scale2 shape2   asy1   asy2    dep
0.8808 1.2381 0.3826 1.0037 1.2702 0.4598 1.0000 0.7013 0.5362

$std.err
     loc1    scale1    shape1      loc2    scale2    shape2      asy1      asy2
1.395e-01 1.222e-01 8.068e-02 1.463e-01 1.381e-01 1.018e-01 2.000e-06 1.471e-01
      dep
7.484e-02

$deviance
[1] 774.3

$counts
function gradient
      20       20
```

The first asymmetry parameter is (not surprisingly) estimated to be one. Using the `L-BFGS-B` method prevents the optimizer stopping at parameter boundaries. In fact, if I had used this method to start with, I would have avoided the initial problem.*

Using `fbvalog` with both asymmetry parameters fixed at one returns the same as the symmetric fit (`fbvlog`) given previously.

```
> fbvalog(bvdata, start = list(mar1 = c(1.5,1.4,0.1), mar2 = c(1.4,1.4,0.2),
dep = 0.73), asy1 = 1, asy2 = 1, method="BFGS")
$estimate
  loc1 scale1 shape1   loc2 scale2 shape2    dep
0.9101 1.2591 0.3651 0.9777 1.2309 0.4570 0.6495

$std.err
   loc1   scale1   shape1    loc2   scale2   shape2     dep
0.14051 0.12322 0.08052 0.14131 0.13148 0.09830 0.06020

$deviance
[1] 777.9

$counts
function gradient
      41       16
```

4.3 Examples

In this section I present two extended examples (one univariate and one bivariate) using the data sets `oxford` and `sealevel`, included in the EVD package.

*And it would not have been a very good illustration of boundary problems!

4.3.1 A Univariate Example

The numeric vector `oxford` contains annual maximum temperatures (in degrees Fahrenheit) at Oxford, England, from 1901 to 1980. The data has previously been analysed by Tabony (1983).

```
> data(oxford)
# Loads the data
> plot(1901:1980, oxford)
> sqrt(6 * var(oxford))/pi
# Moment estimate of scale
[1] 3.326
> mean(oxford) - 0.58 * sqrt(6 * var(oxford))/pi
# Moment estimate of location
[1] 83.4
> oxford.fit <- fgev(oxford, start = list(loc=83.5, scale=3.5, shape=0))
> oxford.fit
$estimate
    loc   scale   shape
83.8386  4.2597 -0.2871

$std.err
    loc   scale   shape
0.52310 0.36575 0.06835

$deviance
[1] 457.8

$counts
function gradient
     106       NA

> fgev(oxford, start = list(loc=83.8, scale=4.25))$deviance -
  oxford.fit$deviance
[1] 12
> pchisq(12, 1, low = FALSE)
[1] 0.000532
```

Always begin by plotting the data. The plots generated during this example are presented (with better labeling) in Figure 2 at the end of this section. The assumptions of stationarity and independence seem sensible. To specify decent starting values I use moment estimators for the location and scale parameters under the assumption that the shape is zero. The value `0.58`, used in the moment estimator for the location, is the Euler constant[*] (to 1dp). The fitted shape is negative, so the fitted distribution is Weibull. The hypothesis that the shape is zero (the Gumbel distribution) is rejected at any significance level above about 0.0005.

```
> mle <- oxford.fit$estimate
> as.vector(mle[1] - mle[2]/mle[3])
# End point of the fitted GEV distribution
[1] 98.67
> range(oxford)
[1] 75 95
```

[*]The Euler constant is defined as the limit as $n \to \infty$ of $\sum_{x=1}^{n} 1/x - \log(n)$.

The (upper) end point* of the fitted distribution is 98.67. The maximum temperature recorded was 95 degrees Fahrenheit, so we are not near the edge of the parameter space.† Basic diagnostic plots can be produced using

```
> pvec <- seq(70, 100, len=200)
> plot(pvec, dgev(pvec, mle[1], mle[2], mle[3]), type = "l")
> rug(jitter(oxford))

> plot(c(70, sort(oxford), 100), c(seq(0, 1, len=81), 1),
  xlim = c(70, 100), type = "s")
> lines(pvec, pgev(pvec, mle[1], mle[2], mle[3]), lty = 2)

> plot(qgev(ppoints(oxford), mle[1], mle[2], mle[3]), sort(oxford))
> abline(0,1)
```

The plots compare parametric distributions, densities and quantiles to their empirical counterparts. The profile deviance (minus twice the profile likelihood) of the shape parameter can be plotted using

```
> shape.profile <- numeric(20)
> shapes <- seq(-0.5, 0.1, len = 20)
> for(i in 1:20)
  shape.profile[i] <- fgev(oxford, start = list(loc=83.8, scale=4.25),
  shape = shapes[i])$deviance
Warning messages:
[...]

# In four of the twenty optimizations the deviance was infinite at the starting
# values. Infinite values are converted to 1e6 when passed to the optimizer.
# Check that the optimizer always found its way to a ‘finite’ value with
> any(shape.profile == 1e6)
[1] FALSE

> plot(sp <- spline(shapes, shape.profile), type = "l")
> climit <- oxford.fit$deviance + qchisq(0.95, 1)
# One degree of freedom
> climit
[1] 461.6
> abline(h = climit, lty = 2)
> approx(sp$y[sp$x < -0.29], sp$x[sp$x < -0.29], climit)$y
[1] -0.4143
> approx(sp$y[sp$x > -0.29], sp$x[sp$x > -0.29], climit)$y
[1] -0.1388
```

A horizontal line is drawn such that the two intersections of the line with the profile deviance yield a profile confidence interval, with confidence coefficient 0.95. The end points of the interval are calculated explicitly using `approx`, which performs linear interpolation. The `spline` function also performs interpolation which, when plotted, draws a curve between the evaluated points. Profile deviances for the location and scale parameters can be calculated in the same manner.

*The upper limit of the distribution function F is defined as $\sup\{x : F(x) < 1\}$.

†When I refer to the parameter space I include the constraints imposed by the data.

```
> scale.profile <- numeric(20)
> scales <- seq(3, 6, len = 20)
> for(i in 1:20)
  scale.profile[i] <- fgev(oxford, start = list(loc=83.8, shape=-0.28),
  scale = scales[i])$deviance
Warning message:
[...]
> plot(spline(scales, scale.profile), type = "l")
> abline(h = climit, lty = 2)

> loc.profile <- numeric(20)
> locs <- seq(82, 86, len = 20)
> for(i in 1:20)
  loc.profile[i] <- fgev(oxford, start = list(scale=4.25, shape=-0.28),
  loc = locs[i])$deviance
> plot(spline(locs, loc.profile), type = "l")
> abline(h = climit, lty = 2)
```

The joint profile deviance of the scale and shape parameters (which are often highly correlated) can be plotted using

```
> scaleshape.profile <- numeric(20^2)
> scaleshapes <- expand.grid(scales, shapes)
> for(i in 1:(20^2))
  scaleshape.profile[i] <- fgev(oxford, start = list(loc=83.8),
  scale = scaleshapes[i,1], shape = scaleshapes[i,2])$deviance

There were 50 or more warnings (use warnings() to see the first 50)
# In more than 50 optimizations the deviance was infinite at the starting values
> any(shape.profile == 1e6)
[1] FALSE
# Again, the optimizer always found its way to a finite value

> br.pts <- oxford.fit$deviance +
  qchisq(c(0,0.5,0.8,0.9,0.95,0.99,0.999,0.9999), 2)
# Two degrees of freedom needed
> image(scales, shapes, matrix(scaleshape.profile, nrow = 20),
  col = heat.colors(8), breaks = c(br.pts,max(scaleshape.profile)))
```

The colours of the image plot represent confidence sets with different confidence coefficients. The lightest colour (ignoring the background colour) represents a confidence ellipse with coefficient 0.9999; the darkest colour represents a confidence ellipse with coefficient 0.5. It looks a bit 'blocky' though, because points were only evaluated on a 20 by 20 grid and no interpolation has been performed. The number of points evaluated could be increased, but interpolation is the better option. The following snippet requires the `akima` package.

```
> library(akima)
> profile.interp <- interp(scaleshapes[,1], scaleshapes[,2], scaleshape.profile,
  xo = seq(3, 6, len = 75), yo = seq(-0.5, 0.1, len = 75))
> image(profile.interp, col = heat.colors(8), breaks =
  c(br.pts, max(scaleshape.profile)))
```

Much better. Increasing the `len` arguments to `xo` and `yo` provides a better resolution. A return period plot can be produced using

```
> ret.period <- 10^(seq(0, 3, len = 100))[-1]
> plot(ret.period, qgev(1/ret.period, mle[1], mle[2], mle[3], low = FALSE),
  log = "x", type = "l")
> points(rev(1/ppoints(oxford)), sort(oxford))
```

Return values are simply quantiles in the upper tail. Return values can be generated using `qgev(probs, low = FALSE)`, where `probs` is a vector of probabilities. The return periods associated with these return values are `1/prob`. The return value associated with a return period of 100 years (or months, or whatever) is therefore defined as the upper quantile evaluated at the probability 0.01. The return period plot compares empirical return values with the fitted model. Confidence intervals for the return value at any specific return period can be calculated using e.g. the delta method, and profile deviances can be calculated by re-parameterizing the GEV likelihood.*

Imagine that Oxford is on another planet (it's easy if you try), and that maximum daily temperatures are both stationary and independent. The snippet below fits normal and gamma distributions to the daily observations.

```
> fext(oxford, start = list(mean = 40, sd = 1), distn = "norm", mlen = 365)
$estimate
 mean    sd
48.85 12.43

$std.err
  mean     sd
2.7204 0.9928

$deviance
[1] 464.2

$counts
function gradient
      51       NA

> fext(oxford, start = list(scale = 1, shape = 1), distn = "gamma",
  mlen = 365)
$estimate
scale shape
 1.63 32.94

$std.err
 scale  shape
0.2407 6.0378

$deviance
[1] 465.9
```

*Which can only be done by hacking the code or using another routine (or waiting until I decide to implement explicit diagnostic functions).

```
$counts
function gradient
     353       NA

> plot(pvec, dgev(pvec, mle[1], mle[2], mle[3]), type = "l")
> lines(pvec, dext(pvec, mean = 48.85, sd = 12.43, distn = "norm", mlen = 365),
  lty = 2)
> lines(pvec, dext(pvec, scale = 1.63, shape = 32.94, distn = "gamma",
  mlen = 365), lty = 3)
> rug(jitter(oxford))
```

The fitted densities for the yearly maxima, derived by passing normal and gamma parameter estimates to `dgev`, are compared to the GEV model. The normal and gamma models yield very similar distributions, and both are marginally more right skewed than the GEV fit.[*] I'll admit that this example isn't particularly relevant for the `oxford` data (Oxford isn't on a different planet - I checked), but it may be relevant for other data sets.

[*]Which will not be surprising to those who know about domains of attraction. The limiting distribution (as `mlen` tends to ∞) of both the normal and gamma models is Gumbel.

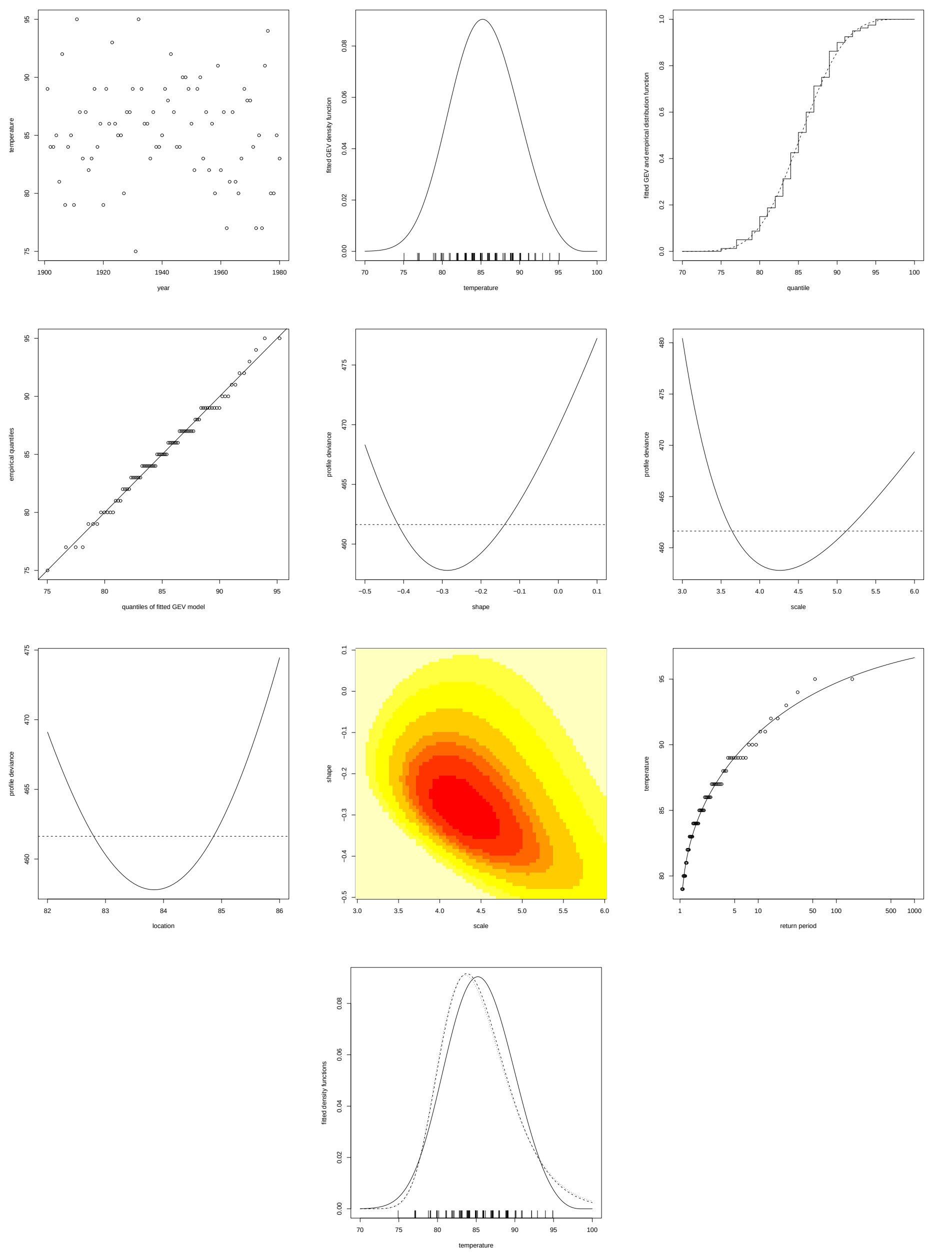

Figure 2: Lots of diagnostic plots. Reading from left to right: the data itself; the fitted GEV density including a rug plot of the (jittered) data; the fitted GEV and empirical distributions; a quantile-quantile plot; profile deviances for the shape, scale and location parameters respectively, including horizontal lines representing profile 95% confidence intervals; the joint profile deviance of the scale and shape parameters; a return level plot; and finally, the first plot is repeated along with fitted distributions for models assuming normal and gamma distributions for observed daily maxima (valid under extremely tenuous assumptions).

4.3.2 A Bivariate Example

The `sealevel` data frame (Coles and Tawn, 1990) has two columns containing annual sea level maxima from 1938 to 1976 at Dover and Harwich respectively, two sites on the coast of Britain.

```
> data(sealevel)
# Load the data
> plot(sealevel)
> plot(1938:1976, sealevel[,1])
> plot(1938:1976, sealevel[,2])
> sl <- sealevel
# Save on typing
```

The Dover data arguably contain a slight trend. It may be advisable to de-trend the data and repeat the analysis given subsequently. Although not currently implemented in the EVD package, it is possible to model the location parameter with an explicit trend term which can be estimated and tested for significance. Non-stationary models such as this may be implemented at a later date.

The plot of the Harwich maxima against the Dover maxima shows that there exists a reasonable degree of dependence. The following code attempts to find decent starting values for the overall fit.

```
> sqrt(6 * c(var(sl[,1]), var(sl[,2])))/pi
[1] 0.2073 0.2383
> c(mean(sl[,1]), mean(sl[,2])) - 0.58 * c(0.21, 0.24)
[1] 3.623 2.619

> tmp <- fbvlog(sl, start = list(mar1 = c(3.6, 0.2, 0), mar2 = c(2.6, 0.25,
  0)), dep = 1, method = "BFGS")
> tmp <- fbvalog(sl, start = list(mar1 = c(3.6, 0.2, 0), mar2 = c(2.6, 0.25,
  0)), asy1 = 1, asy2 = 1, dep = 1, method = "BFGS")
> tmp <- fbvhr(sl, start = list(mar1 = c(3.6, 0.2, 0), mar2 = c(2.6, 0.25,
  0)), dep = 0.01, method = "BFGS")
# Equivalent. All produce the same object.
> tmp$estimate
   loc1  scale1  shape1    loc2  scale2  shape2
3.62212 0.18449 0.07967 2.62140 0.19891 0.10083
```

The first two lines of code find method of moments estimators for the marginal GEV location and scale marginal parameters assuming independence and zero shapes (as in Section 4.3.1). These starting values are then passed to a bivariate fitting routine which finds the maximum likelihood estimators of the marginal parameters assuming independence (any bivariate fitting function can be used; three examples are given above). This is equivalent (for estimates and standard errors) to calling `fgev` on both margins. In this case the estimates produced are close to the original moment estimators, so I am going to stick with the latter to use as my starting values.

```
> sl.fit <- fbvalog(sl, start = list(mar1 = c(3.6, 0.2, 0), mar2 = c(2.6, 0.25,
0), asy = c(0.8, 0.8), dep = 0.6), method = "BFGS", control = list(trace=1))
# Takes about 40 seconds for me
initial  value -7.938962
```

```
iter  10 value -11.402720
iter  20 value -12.772531
final  value -12.772534

> sl.fit
$estimate
    loc1   scale1   shape1     loc2   scale2   shape2     asy1     asy2      dep
 3.63086  0.18639 -0.04301  2.62542  0.20160  0.08512  0.69554  0.44967  0.24316

$std.err
   loc1  scale1  shape1    loc2  scale2  shape2    asy1    asy2     dep
0.03351 0.02495 0.10145 0.03598 0.02585 0.10739 0.21302 0.14489 0.10671

$deviance
[1] -25.55

$counts
function gradient
      82       22
```

The above code estimates the parameters of the asymmetric logistic model. The fit took about 40 seconds (on my machine), including the time used to find the inverse of the 9×9 observed information matrix, needed to calculate the standard errors. The fitted marginal distributions for the Dover and Harwich maxima are Weibull and Fréchet respectively. The shape parameter on the first margin (Dover) has become negative due to the dependence between the sites (and hence the information provided by the Harwich data). Both of the shape parameter estimates are within two standard errors of zero, so they could feasibly be Gumbel. A likelihood ratio test of this hypothesis is given by*

```
> fbvalog(sl, start = list(loc1 = 3.6, scale1 = 0.19, loc2 = 2.6, scale2 = 0.2,
asy = c(0.7, 0.45), dep = 0.24), method = "BFGS")$deviance - sl.fit$deviance
# The shapes need not be specified since zero is the default value in dbvalog
[1] 4.001
> pchisq(4.001, 2, low = FALSE)
[1] 0.1353
```

Which you would accept if you favour significance levels below 0.1. A likelihood ratio test for independence can be performed using

```
> tmp$deviance - sl.fit$deviance
[1] 22.05
```

The annual maxima are clearly nowhere near independent. The asymptotic distribution of the test statistic is non-regular because the dependence parameter in the restricted model is fixed at the edge of the parameter space. Tawn (1988) discusses non-regular testing procedures within bivariate extreme value models.

```
> mle <- sl.fit$estimate
```

*I'm cheating, since I perform the test only because I have already seen the parameters estimates. I'll be cheating throughout this section.

```
> as.vector(mle[1] - mle[2]/mle[3])
[1] 7.965
> range(sl[,1])
[1] 3.32 4.57
> as.vector(mle[4] - mle[5]/mle[6])
[1] 0.2571
> range(sl[,2])
[1] 2.35 3.99
```

I like to know how near I am to the edge of the parameter space. For the Dover maxima, the *upper* limit* of the fitted marginal distribution is 7.96. For the Harwich maxima, the *lower* limit† of the fitted marginal distribution is 0.26. The ranges of the data on each margin are also shown.

The symmetric logistic distribution can be fitted directly using `fbvlog`, or by using `fbvalog` and fixing the asymmetry parameters to be one (both methods are shown below).

```
> sl.fit2 <- fbvalog(sl, start = list(mar1 = c(3.6, 0.19, -0.04), mar2 = c(2.6,
  0.2, 0.09), dep = 0.6), asy1 = 1, asy2 = 1, method = "BFGS")
> sl.fit2 <- fbvlog(sl, start = list(mar1 = c(3.6, 0.19, -0.04), mar2 = c(2.6,
  0.2, 0.09), dep = 0.6), method = "BFGS")
# Equivalent. Both produce the same object.
> sl.fit2$estimate
    loc1   scale1   shape1     loc2   scale2   shape2      dep
 3.62966  0.18774 -0.01478  2.62206  0.19696  0.06469  0.62474
> sl.fit2$deviance
[1] -22.08
> sl.fit2$deviance - sl.fit$deviance
[1] 3.469
# The asymptotic distribution of this test statistic is non-regular

> abvalog(dep = 0.24316, asy = c(0.69554, 0.44967), plot = TRUE)
> abvlog(dep = 0.62474, add = TRUE, lty = 2)
> abvalog(dep = 0.24316, asy = c(0.69554, 0.44967))
[1] 0.7884
> abvlog(dep = 0.62474)
[1] 0.771
```

The dependence functions for the symmetric and asymmetric fits are plotted using `abvalog` and `abvlog` (dependence functions for various models are given in Figure 3 at the end of this section). Both show a reasonable degree of the dependence, and the amount of asymmetry within the asymmetric model is seen to be negligible.

Brace yourself for a fair amount of code. Here are some alternative models (the symmetric and asymmetric negative logistic, and the Hüsler-Reiss).

```
> fbvhr(sl, start = list(mar1 = c(3.6, 0.19, -0.04), mar2 = c(2.6, 0.2, 0.09),
  dep = 1), method = "BFGS", control = list(trace=1))
initial  value -8.384898
iter  10 value -10.226785
final  value -10.227197
```

*The upper limit of the distribution function F is defined as $\sup\{x : F(x) < 1\}$.

†The lower limit of the distribution function F is defined as $\inf\{x : F(x) > 0\}$.

```
converged
$estimate
    loc1   scale1   shape1     loc2   scale2   shape2      dep
3.628634 0.186708 0.002070 2.624658 0.198828 0.049562 1.253410

$std.err
   loc1  scale1  shape1    loc2  scale2  shape2     dep
0.03332 0.02547 0.10978 0.03492 0.02649 0.08750 0.25854

$deviance
[1] -20.45

$counts
function gradient
      47       13

> abvhr(dep = 1.253410, add = TRUE, lty = 3)
> abvhr(dep = 1.253410)
[1] 0.7875

> fbvaneglog(sl, start = list(mar1 = c(3.6, 0.19, -0.04), mar2 = c(2.6, 0.2,
  0.09), dep = 1, asy = c(0.8,0.8)), method = "BFGS", control = list(trace=1))
initial  value -10.026860
iter  10 value -11.606034
iter  20 value -12.756318
final  value -12.756320
converged
$estimate
    loc1   scale1   shape1     loc2   scale2   shape2      dep     asy1     asy2
 3.63075  0.18641 -0.04278  2.62544  0.20154  0.08580  3.44762  0.69796  0.44601

$std.err
   loc1  scale1  shape1    loc2  scale2  shape2     dep    asy1    asy2
0.03353 0.02505 0.10219 0.03595 0.02589 0.10761 1.72055 0.21072 0.14088

$deviance
[1] -25.51

$counts
function gradient
      84       21

> abvaneglog(dep = 3.44762, asy = c(0.69796,0.44601), add = TRUE, lty = 3)
> abvaneglog(dep = 3.44762, asy = c(0.69796,0.44601))
[1] 0.7892

> fbvaneglog(sl, start = list(mar1 = c(3.6, 0.19, -0.04), mar2 = c(2.6, 0.2,
  0.09), dep = 1), asy1 = 1, asy2 = 1, method = "BFGS", control = list(trace=1))
initial  value -9.453044
iter  10 value -10.686188
final  value -10.686679
converged
```

```
$estimate
    loc1   scale1   shape1     loc2   scale2   shape2      dep
 3.62862  0.18750 -0.00417  2.62373  0.19825  0.06005  0.87394

$std.err
   loc1  scale1  shape1    loc2  scale2  shape2     dep
0.03335 0.02574 0.10864 0.03483 0.02639 0.08903 0.24120

$deviance
[1] -21.37

$counts
function gradient
      55       13

> abvneglog(dep = 0.87394, add = TRUE, lty = 3)
> abvneglog(dep = 0.87394)
[1] 0.7738
```

Models which are not nested can be compared by adding penalty terms to the deviances which take into account the number of parameters fitted (if both models have the same number of parameters the deviances can be compared directly). Three commonly used penalty terms are $2p$[*], $p\log(n)$[†] and $p[1+\log(n)]$[‡], where p is the number of parameters estimated and n is the number of observations. Adding these penalties to the deviances produced in this section gives (the sea level data contains $n = 39$ observations)

	$2p$	$p\log(n)$	$p[1+\log(n)]$
Symmetric Logistic	-8.08	3.57	10.56
Symmetric Negative Logistic	-7.37	4.28	11.27
Hüsler Reiss	-6.45	5.20	12.19
Asymmetric Logistic	-7.55	7.42	16.42
Asymmetric Negative Logistic	-7.51	7.46	16.46

Under these criteria the symmetric logistic distribution is seen to give the best fit.

I will finish this section by fitting the asymmetric model under the constraint $\theta = \theta_1 = \theta_2$. Constrained models of this type can be fitted indirectly using profile deviances/likelihoods. The profile deviance for θ is given by

```
> theta.profile <- numeric(10)
> thetas <- seq(0.2, 1, len = 10)
> for(i in 1:10)
  theta.profile[i] <- fbvalog(sl, start = list(mar1 = c(3.63, 0.19, -0.04),
  mar2 = c(2.63, 0.2, 0.09), dep = 0.25), asy1 = thetas[i], asy2 = thetas[i],
  method = "BFGS", control = list(trace=1))$deviance
# Be prepared to wait!
initial  value -8.135660
[...]
```

[*]Akaike's information criterion, or AIC.
[†]Schwarz's criterion.
[‡]Bayesian information criterion, or BIC.

```
final  value -11.038110
converged

> plot(th.sp <- spline(thetas, theta.profile, n = 200), type="l")

> min(th.sp$y)
[1] -23.99
# The minimum profile deviance
# This is also the deviance evaluated at the maximum likelihood estimates

> th.sp$x[min(th.sp$y) == th.sp$y]
[1] 0.4492
# The maximum likelihood estimate for theta
```

The the profile deviance is lowest (-23.99) when $\theta = 0.45$. The full model can then be fitted using

```
> fbvalog(sl, start = list(mar1 = c(3.63, 0.18, -0.04), mar2 = c(2.63, 0.2,
  0.09), dep = 0.24), asy1 = 0.4492, asy2 = 0.4492, method = "BFGS",
  control = list(trace=1))
initial  value -8.630673
iter  10 value -11.920427
final  value -11.994322
converged
$estimate
    loc1   scale1   shape1     loc2   scale2   shape2      dep
 3.64858  0.19343 -0.06808  2.61243  0.19516  0.06455  0.16361

$std.err
   loc1  scale1  shape1    loc2  scale2  shape2     dep
0.03039 0.02314 0.09274 0.03254 0.02419 0.10000 0.07127

$deviance
[1] -23.99

$counts
function gradient
      64       16
```

The standard errors correspond to the constrained model where $\theta = \theta_1 = \theta_2 = 0.45$.

References

Coles, S. G. (2001) *An Introduction to Statistical Modeling of Extreme Values*. London: Springer–Verlag.

Coles, S. G. and Tawn, J. A. (1990) Statistics of coastal flood prevention. *Phil. Trans. R. Soc. Lond., A*, **332**, 457–476.

Galambos, J. (1975) Order statistics of samples from multivariate distributions. *J. Amer. Statist. Assoc.*, **70**, 674–680.

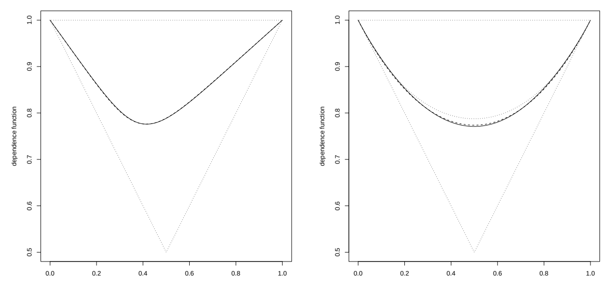

Figure 3: The dependence functions of fitted bivariate models for the sea level data. The asymmetric dependence functions in the left plot are for the asymmetric logistic (solid line) and asymmetric negative logistic (dashed line) models respectively (they are almost indistinguishable). The symmetric dependence functions in the right plot are for the logistic (solid line), negative logistic (dashed line) and Hüsler-Reiss (dotted line) models respectively.

Ghoudi, K., Khoudraji, A. and Rivest, L.-P. (1998) Propriétés statistiques des copules de valeurs extrêmes bidimensionnelles. *Canad. J. Statist.*, **26**, 187–197.

Gumbel, E. J. (1960) Distributions des valeurs extrêmes en plusieurs dimensions. *Publ. Inst. Statist. Univ. Paris*, **9**, 171–173.

Gumbel, E. J. (1961) Bivariate logistic distributions. *J. Amer. Statist. Assoc.*, **56**, 335–349.

Hüsler, J. and Reiss, R.-D. (1989) Maxima of normal random vectors: between independence and complete dependence. *Statist. Probab. Letters*, **7**, 283–286.

Ihaka, R. and Gentleman, R. (1996) R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, **5**, 299–314.

Joe, H. (1990) Families of min-stable multivariate exponential and multivariate extreme value distributions. *Statist. Probab. Letters*, **9**, 75–81.

Kotz, S. and Nadarajah, S. (2000) *Extreme Value Distributions*. London: Imperial College Press.

Pickands, J. (1981) Multivariate extreme value distributions. *Proc. 43rd Sess. Int. Statist. Inst.*, **49**, 859–878.

Smith, R. L. (1985) Maximum likelihood estimation in a class of non-regular cases. *Biometrika*, **72**, 67–90.

Stephenson, A. G. (2002) Simulating multivariate extreme value distributions of logistic type. To be published - available on request.

Tabony, R. C. (1983) Extreme value analysis in meteorology. *The Meteorological Magazine*, **112**, 77–98.

Tawn, J. A. (1988) Bivariate extreme value theory: models and estimation. *Biometrika*, **75**, 397–415.

Tawn, J. A. (1990) Modelling multivariate extreme value distributions. *Biometrika*, **77**, 245–253.