Semiparametrically Efficient Population and Permutation Inference in Distribution-free Stratified K-sample Oneway Layouts[1]

Torsten Hothorn
Universität Zürich

Kurt Hornik
WU Wirtschaftsuniversität Wien

[1]Please cite this document as: Torsten Hothorn and Kurt Hornik (), Semiparametrically Efficient Population and Permutation Inference in Distribution-free Stratified K-sample Oneway Layouts, R package vignette version 1.1-0, DOI:10.32614/CRAN.package.free1way.docreg

Contents

1 Model and Parameterisation **1**

2 Parameter Estimation **3**

3 Link Functions **18**

4 Optimisation **24**

5 ML Estimation **27**

6 ML Inference **39**
6.1 Wald Statistics . . . 39
6.2 Likelihood-ratio Statistics . . . 39
6.3 Rao Score Statistics . . . 40

7 Permutation Inference **41**

8 Distribution-free Tests in Stratified K-sample Oneway Layouts **48**
8.1 `free1way` . . . 48
8.2 `free1way` Methods . . . 55

9 Special Test Procedures **67**
9.1 Wilcoxon Test . . . 67
9.2 Mantel-Haenszel Test . . . 70
9.3 `prop.test` . . . 71
9.4 Kruskal-Wallis Test . . . 71
9.5 Savage Test . . . 72
9.6 Log-rank Test . . . 75
9.7 van der Waerden Test . . . 80
9.8 Friedman Test . . . 80
9.9 McNemar Test . . . 82
9.10 Incomplete Block Designs . . . 83
9.11 Contrasts Tests . . . 83
9.12 Scale Tests . . . 84

10 Model Diagnostics **86**
10.1 Transformation Plots . . . 86
10.2 Probability-probability Plots . . . 89

11 Random Number Generation **96**

12 Power and Sample Size **101**

13 Penalisation **111**

14 Acknowledgements **114**

Abstract

Starting with R 4.6-0, the **stats** package provides infrastructure for distribution-free model-based inference in possibly stratified K-sample oneway layouts via the novel `free1way` model function. Treatment effects to be estimated using `free1way` include odds- and hazard ratios, Lehmann parameters, and a generalised version of Cohen's d for at least ordered and possibly right-censored outcomes.

In addition to nonparametric maximum-likelihood estimators of treatment effects, the procedure allows Wald, Rao score, and likelihood ratio tests with corresponding confidence intervals to be computed. Asymptotic and approximate Monte-Carlo-based permutation tests and confidence intervals are also available. In proportional odds models, exact permutation inference is implemented based on exact permutation distributions derived via the Streitberg-Röhmel algorithm.

Graphical tools for model diagnostics, including model-based confidence bands for receiver operating characteristic (ROC) curves in probability-probability plots in the new `ppplot` function, allow data-driven criticism of model assumptions.

Power assessment and sample-size planning is facilitated either in a simulation-based way relying on random number generation via `rfree1way` or based on approximations of the information matrix in `power.free1way.test`, the latter approach being much faster but slightly less accurate.

The new `free1way` function can be understood as a unification and generalisation of some of the classical "nonparametric" test procedures in **stats**, including `kruskal.test`, `wilcox.test`, `friedman.test`, `mantelhaen.test`, `prop.test`, `mcnemar.test`, as well as `power.prop.test`, allowing the magnitude of treatment effects to be interpreted on various scales, providing the possibility to assessment variability by means of confidence intervals and corresponding tests for these parameters, and offering tools for sample-size planning and model criticism.

This document explains the technical underpinnings of the implementation. The **free1way** package provides this vignette as additional documentation and serves as a home for extensive regression tests.

Introduction

Comparing two or more independent samples with respect to some outcome measure is a common task. Many procedures are available in **stats** and other add-on packages, most of these implementations making rather strict assumptions regarding the outcome distribution, the number of samples, the presence of blocks or strata and typically offer either conditional or unconditional (exact or asymptotic) inference.

This document presents a unified, dense, and yet holistic implementation covering many classical procedures as special cases. Leveraging transformation models, likelihood-based parameter estimation as well as permutation- and likelihood-based inference are formulated and implemented. One can understand this contribution as a unification of many of `stats::*.test` procedures, the models available in `MASS::polr`, `rms::orm`, `rms::lrm`, `survival::coxph`, or the **tram** add-on package (among many others), and permutation-based inference in **coin**.

This implementation is, however, free of any strong dependencies and only uses functionality available in R itself and the **stats**, **graphics**, and **Matrix** recommended packages.

Chapter 1

Model and Parameterisation

We consider K treatment groups $G \in \{1, \ldots, K\}, K \geq 2$ for an at least ordered outcome $Y \in \mathcal{Y}$ observed in stratum $S \in \{1, \ldots, B\}$ out of $B \geq 1$ blocks with conditional cumulative distribution function (cdf) $F_Y(y \mid G = k, S = b) = \mathbb{P}(Y \leq y \mid G = k, S = b)$. Detecting and describing differential distributions arising from different treatments is our main objective. We refer to the first treatment $G = 1$ as "control".

Model With model function $m : [0, 1] \times \mathbb{R} \to [0, 1]$, we describe the conditional distribution under treatment k as a function of the conditional distribution under control and a scalar parameter δ_k:

$$F_Y(y \mid G = k, S = b) = m(F_Y(y \mid G = 1, S = b), \delta_k).$$

The model is assumed to hold for all blocks $b = 1, \ldots, B$, treatments $k = 2, \ldots, K$, and outcome values $y \in \mathcal{Y}$ based on parameters $\delta_2, \ldots, \delta_K \in \mathbb{R}$. For notational convenience, we define $\delta_1 := 0$.

This model formulation gives rise to several specific models, for example, $m_{\mathrm{L}}(p, \delta) = p^{\exp(-\delta)}$ (Lehmann alternatives), $m_{\mathrm{PH}}(p, \delta) = 1 - (1 - p)^{\exp(-\delta)}$ (proportional hazards), $m_{\mathrm{PO}}(p, \delta) = \mathrm{expit}(\mathrm{logit}(p) - \delta)$ (proportional odds), or $m_{\mathrm{Cd}}(p, \delta) = \Phi(\Phi^{-1}(p) - \delta)$ (generalised Cohen's d).

Instead of directly working with g, we parameterise the model in terms of some absolute continuous cdf F with log-concave density $f = F'$ and corresponding derivative f'. The location model

$$F_Y(y \mid G = k, S = b) = F\left(F^{-1}\left(F_Y(y \mid G = 1, S = b)\right) - \delta_k\right), \quad k = 2, \ldots, K \tag{1.1}$$

describes different distributions by means of shift parameter on a latent scale defined by F. The negative shift term ensures that positive values of δ_k correspond to the situation of outcomes being stochastically larger in group k compared to control. The shift parameters are invariant with respect to monotone transformations of the response values, that is, transforming the observations of all treatment groups by the same function does not affect the values of δ_k.

The choice $F(z) = \exp(-\exp(-z))$ gives rise to m_{L}, $F(z) = 1 - \exp(-\exp(z))$ corresponds to m_{PH}, $F = \mathrm{expit}$ leads to m_{PO}, and $F = \Phi$ results in m_{Cd}. The choice of F is made a priori and determines the interpretation of δ_k.

This document describes the implementation of estimators of these shift parameters, as well as of confidence intervals and formal hypothesis tests for contrasts thereof under the permutation and population model. Proportional odds models (m_{PO}) are explained in-depths by Harrell (2015), although the models are presented in terms of survivor, not distribution, functions.

We provide short-cuts for evaluating the link function F^{-1}, its inverse F, its density f and the derivative f':

⟨ *short-cuts for link functions* 1 ⟩ ≡

```
### short-cuts for link functions
F <- function(q, ...) .p(link, q = q, ...)
Q <- function(p, ...) .q(link, p = p, ...)
f <- function(q, ...) .d(link, x = q, ...)
fp <- function(q) .dd(link, x = q)
```

◇

Fragment referenced in 43.

Hypothesis We are interested in inference for $\delta_2, \dots, \delta_K$, in terms of confidence intervals and hypothesis tests of the form

$$\begin{aligned} H_0 &: \delta_k - \mu_k = 0, \text{“two.sided”}, \quad k = 2, \dots, K, \\ H_0 &: \delta_k - \mu_k \geq 0, \text{“less”}, \quad k = K = 2, \\ H_0 &: \delta_k - \mu_k \leq 0, \text{“greater”}, \quad k = K = 2, \end{aligned}$$

with the latter two options only for the two-sample case ($K = 2$).

Likelihood For an ordered categorical outcome Y from sample space $\mathcal{Y} = \{\upsilon_1 < \upsilon_2 < \cdots < \upsilon_C\}$, we parameterise the model in terms of intercept ($\vartheta.$) and shift ($\delta.$) parameters

$$F_Y(\upsilon_c \mid G = k, S = b) = F(\vartheta_{c,b} - \delta_k), \quad c = 1, \dots, C,$$

that is we replace the transformed control outcome $F^{-1}\left(F_Y(\upsilon_c \mid G = 1, S = b)\right) = \vartheta_{c,b}$ with a corresponding intercept parameter. These $C - 1$ intercept parameters are block-specific and monotone increasing $\vartheta_{0,b} = -\infty < \vartheta_{1,b} < \cdots < \vartheta_{C,b} = \infty$ within each block $b = 1, \dots, B$.

We collect all model parameters in a vector

$$\begin{aligned} \boldsymbol{\theta} = (\theta_1 &:= \delta_2, \\ &\dots \quad , \\ \theta_{K-1} &:= \delta_K, \\ \theta_K &:= \vartheta_{1,1}, \\ \theta_{K+1} &:= \vartheta_{2,1} > \vartheta_{1,1}, \\ &\dots, \\ \theta_{K+C-2} &:= \vartheta_{C-1,1} > \vartheta_{C-2,1}, \\ \theta_{K+C-1} &:= \vartheta_{1,2}, \\ &\dots \quad , \\ \theta_{B(C-1)+K-1} &:= \vartheta_{C-1,B} > \vartheta_{C-2,B}). \end{aligned}$$

If there is no observation for level c in block b, the corresponding parameter is not identified and removed from $\boldsymbol{\theta}$. The parameter space is defined by all parameter vectors $\boldsymbol{\theta}$ satisfying the monotonicity of the intercept parameters. Violations always lead to the log-likelihood function being undefined and thus taking the value $-\infty$. Harrell (2024) evaluates unconstrained optimisation in this context and recommends Newton-based algorithms leveraging the analytically available Hessian (see below).

For the ith observation $(y_i = \upsilon_c, g_i = k, s_i = b)$ from block b under treatment k, the log-likelihood contribution is

$$\log(\mathbb{P}(\upsilon_{c-1} < Y \leq \upsilon_c \mid G = k, S = b)) = \log(F(\vartheta_{c,b} - \delta_k) - F(\vartheta_{c-1,b} - \delta_k))$$

with $\upsilon_0 = -\infty$.

For an absolutely continuous outcome $Y \in \mathbb{R}$, we define $\upsilon_c := y_{(c)}$, the cth distinct ordered observation in the sample. The log-likelihood above is then the empirical or nonparametric log-likelihood.

If observations were independently right-censored, the contribution of the event $Y > \tilde{y}$ to the log-likelihood is

$$\log(\mathbb{P}(Y > \tilde{y} \mid G = k, S = b)) = \log(1 - F(\vartheta_{c-1,b} - \delta_k))$$

where $\upsilon_{c-1} = \max\{\upsilon \in \mathcal{Y} \mid \upsilon \leq \tilde{y}\}$, that is, observations right-censored between υ_{c-1} and υ_c correspond to the parameter $\vartheta_{c-1,b}$.

Maximising this form of the log-likelihood leads to semiparametrically efficient estimators (Chapter 15.5 Van der Vaart, 1998). In this framework, tests against deviations from the hypothesis H_0 above are locally most powerful rank tests, for example against proportional odds ($F = \text{expit}$) or proportional hazards alternatives ($F(z) = 1 - \exp(-\exp(z))$, Van der Vaart, 1998, Example 15.16).

We represent the data in form of a $C \times K \times B$ contingency table, whose element (c, k, b) is the number of observations with configuration $(y = y_c, g = k, s = b)$. In the presence of right-censoring, a fourth dimension is added ($C \times K \times B \times 2$) whose first $C \times K \times B$ table presents right-censoring and the second table contains numbers of events.

Chapter 2

Parameter Estimation

We start implementing the log-likelihood function for parameters `parm` $= \boldsymbol{\theta}$ (assuming only a single block) with data from a two-way $C \times K$ contingency table `x`.

From $\boldsymbol{\theta}$, we first extract the shift parameters $\delta_k, k = 2, \ldots, K$ and then the intercept parameters $\vartheta_{c,1}, c = 1, \ldots, C-1$ and evaluate the probabilities `prb` $= \mathbb{P}(y_{c-1} < Y \leq y_c \mid G = k, S = 1)$ for all groups.

Before we begin, let's contemplate a little about the log-likelihood contributions. For probabilities $a > b$, we need to compute $\log(a-b)$. The problem is that the difference might be close to zero or, because of inadmissible parameter configurations, $a < b$. In the former case, we still need a precise assessment of $\log(a - b)$ and in the latter case, we need to return an infinite value, signalling back to the optimisation routine that we went off rails.

For scalars $a > b$, we can compute $\log(a - b)$ using the log-sum-exp trick (adopted to differences), that is with $\alpha = \log(a)$ and $\beta = \log(b)$

$$\begin{aligned} \log(a - b) &= \log\left(\exp(\alpha) - \exp(\beta)\right) \\ &= \alpha + \log\left(\exp(-\alpha)(\exp(\alpha) - \exp(\beta))\right) \\ &= \alpha + \log\left(\exp(\alpha - \alpha) - \exp(\beta - \alpha)\right) \\ &= \alpha + \log\left(1 - \exp(\beta - \alpha)\right). \end{aligned}$$

Because $\beta - \alpha < 0$, the second term in this expression can be computed in a numerically stable way by `pexp(,log.p = TRUE)` (with argument $\alpha - \beta$).

Now, we start with log-probabilities and use the log-sum-exp trick after making sure that inadmissible parameter configurations correspond to a zero difference for log-probabilities and, therefore, an infinite value will be returned.

⟨ *parm to prob* 2 ⟩ ≡

```
bidx <- seq_len(ncol(x) - 1L)
delta <- c(0, mu + parm[bidx])
intercepts <- c(-Inf, parm[- bidx], Inf)
tmb <- intercepts - matrix(delta, nrow = length(intercepts),
                                  ncol = ncol(x),
                                  byrow = TRUE)

Ftmb <- exp(logFtmb <- F(tmb, log.p = TRUE))
### log-sum-exp trick and log(1 - exp()) via pexp
if (rightcensored) {
    logprb <- log1p(- Ftmb[- nrow(Ftmb), , drop = FALSE])
} else {
    logprb <- logFtmb[- 1L, , drop = FALSE] +
              pexp(pmax(0, logFtmb[- 1L, , drop = FALSE] -
                           logFtmb[- nrow(Ftmb), , drop = FALSE]),
                   log.p = TRUE)
}
```

◇

Fragment referenced in 3, 5, 6, 7, 12.

If the table `x` represents right-censored observations, we compute $\texttt{prb} = 1 - \mathbb{P}(Y \leq y_c \mid G = k, S = 1)$.

With default null values $\mu_k = 0, k = 2, \ldots, K$, we define the negative log-likelihood function as the weighted (by number of observations) sum of the log-probabilities. The nice thing here is that we don't need to handle near zero probabilities explicitly, because we computed `logprb` in a stable way.

⟨ *negative logLik* 3 ⟩ ≡

```
.nll <- function(parm, x, mu = 0, rightcensored = FALSE)
{
    ⟨ parm to prob 2 ⟩
    return(- sum(x * logprb))
}
◇
```

Fragment referenced in 43.

The code assumes that all elements of the margins of the table `x` are larger than zero; otherwise, the corresponding parameter is not identified. We will handle such situation at a higher level later on.

It is important to note that, with F corresponding to distribution with log-concave density f, the negative log-likelihood is a convex function of the parameters $\boldsymbol{\theta}$, and thus we can solve the corresponding constrained minimisation problem quickly and reliably.

Next, we implement the gradient of the negative log-likelihood, the negative score function for the parameters in $\boldsymbol{\theta}$. The score function for the empirical likelihood, evaluated at parameters is given in many places (for example in Hothorn et al., 2018, Formula (2)). We begin computing the ratio of $f(\vartheta_{c,1} - \delta_k)$ and the corresponding likelihood

⟨ *density prob ratio* 4 ⟩ ≡

```
ftmb <- f(tmb, log = TRUE)
zu <- x * exp(ftmb[- 1, , drop = FALSE] - logprb)
if (rightcensored) zu[] <- 0 ### derivative of a constant
zl <- x * exp(ftmb[- nrow(ftmb), , drop = FALSE] - logprb)
◇
```

Fragment referenced in 5, 6, 7.

and then compute the negative score function:

⟨ *negative score* 5 ⟩ ≡

```
.nsc <- function(parm, x, mu = 0, rightcensored = FALSE)
{

    ⟨ parm to prob 2 ⟩

    ⟨ density prob ratio 4 ⟩

    ret <- numeric(length(parm))
    ret[bidx] <- .colSums(zl, m = nrow(zl), n = ncol(zl))[-1L] -
                 .colSums(zu[-nrow(zu),,drop = FALSE],
                          m = nrow(zu) - 1L, n = ncol(zu))[-1L]
    ret[- bidx] <- .rowSums(zu[-nrow(zu),,drop = FALSE] -
                            zl[-1,,drop = FALSE],
                            m = nrow(zu) - 1L, n = ncol(zu))
    return(- ret)
}
◇
```

Fragment referenced in 43.

In addition, we define negative score residuals, that is, the derivative of the negative log-likelihood with respect to an intercept term constrained to zero:

⟨ *negative score residuals* 6 ⟩ ≡

```
.nsr <- function(parm, x, mu = 0, rightcensored = FALSE)
{

    ⟨ parm to prob 2 ⟩

    ⟨ density prob ratio 4 ⟩

    ret <- .rowSums(zl - zu, m = nrow(zl), n = ncol(zl)) /
           .rowSums(x, m = nrow(x), n = ncol(x))
    ret[!is.finite(ret)] <- 0
    return(- ret)
}
```

◇

Fragment referenced in 43.

For the sake of completeness, we add negative scale score residuals defined by the derivative of the negative log-likelihood with respect to an scale term constrained to one:

⟨ *negative scale score residuals* 7 ⟩ ≡

```
.nssr <- function(parm, x, mu = 0, rightcensored = FALSE)
{

    ⟨ parm to prob 2 ⟩

    ⟨ density prob ratio 4 ⟩

    ret <- .rowSums(zl * c(0, parm[-bidx]) - zu * c(parm[-bidx], 0),
                    m = nrow(zl), n = ncol(zl)) /
           .rowSums(x, m = nrow(x), n = ncol(x))
    ret[!is.finite(ret)] <- 0
    return(- ret)
}
```

◇

Fragment referenced in 43.

We also need access to the observed Fisher information of the shift parameters. We proceed by implementing the Hessian for the intercept ($\vartheta.$) and shift ($\delta.$) parameters, as given in Formula (4) of Hothorn et al. (2018) first. This partitioned matrix (for the parameter vector of one block starting with the shift parameters)

$$\mathbf{H}(\delta_2, \ldots, \delta_K, \vartheta_1, \ldots, \vartheta_{C-1}) = \begin{pmatrix} \mathbf{Z} & \mathbf{X} \\ \mathbf{X}^\top & \mathbf{A} \end{pmatrix}$$

consists of a symmetric tridiagonal $\mathbf{A} \sim (C-1, C-1)$, a diagonal $\mathbf{Z} \sim (K-1, K-1)$, and a full $\mathbf{X} \sim (C-1, K-1)$ matrix. In a second step, we compute the Fisher information matrix for the shift parameters only by means of the Schur complement $\mathbf{Z} - \mathbf{X}^\top \mathbf{A}^{-1} \mathbf{X}$.

In addition to probabilities `prb`, the Hessian necessitates the computation of $f(\vartheta_{c,1} - \delta_k)$ and $f'(\vartheta_{c,1} - \delta_k)$. We start preparing these objects, keeping in mind to remove terms not being present under right-censoring:

⟨ *Hessian prep* 8 ⟩ ≡

```
## ratios are computed on the log-scale in .nsr()
## however, fp(tmb) can be negative here so we refrain from
## this exercise here
prb <- exp(logprb)
ftmb <- f(tmb)
fptmb <- fp(tmb)

dl <- ftmb[- nrow(ftmb), , drop = FALSE]
du <- ftmb[- 1, , drop = FALSE]
if (rightcensored) du[] <- 0
dpl <- fptmb[- nrow(ftmb), , drop = FALSE]
dpu <- fptmb[- 1, , drop = FALSE]
if (rightcensored) dpu[] <- 0
dlm1 <- dl[,-1L, drop = FALSE]
dum1 <- du[,-1L, drop = FALSE]
dplm1 <- dpl[,-1L, drop = FALSE]
dpum1 <- dpu[,-1L, drop = FALSE]
prbm1 <- prb[,-1L, drop = FALSE]

i1 <- length(intercepts) - 1L
i2 <- 1L
◇
```

Fragment referenced in 12.

The off-diagonal elements of $\mathbf{A}$ are now available as

⟨ *off-diagonal elements for Hessian of intercepts* 9 ⟩ ≡

```
Aoffdiag <- - .rowSums(x * du * dl / prb^2, m = nrow(x), n = ncol(x))[-i2]
Aoffdiag <- Aoffdiag[-length(Aoffdiag)]
◇
```

Fragment referenced in 12.

and the diagonal elements of $\mathbf{A}$ as

⟨ *diagonal elements for Hessian of intercepts* 10 ⟩ ≡

```
Adiag <- - .rowSums((x * dpu / prb)[-i1,,drop = FALSE] -
                    (x * dpl / prb)[-i2,,drop = FALSE] -
                    ((x * du^2 / prb^2)[-i1,,drop = FALSE] +
                     (x * dl^2 / prb^2)[-i2,,drop = FALSE] ),
                    m = nrow(x) - length(i1), n = ncol(x)
                   )

◇
```

Fragment referenced in 12.

For the computation of $\mathbf{X}$ and $\mathbf{Z}$, the observations corresponding to the control group ($k = 1$) are irrelevant, we remove these first

⟨ *intercept / shift contributions to Hessian* 11 ⟩ ≡

```
xm1 <- x[,-1L,drop = FALSE]
X <- ((xm1 * dpum1 / prbm1)[-i1,,drop = FALSE] -
      (xm1 * dplm1 / prbm1)[-i2,,drop = FALSE] -
      ((xm1 * dum1^2 / prbm1^2)[-i1,,drop = FALSE] -
       (xm1 * dum1 * dlm1 / prbm1^2)[-i2,,drop = FALSE] -
       (xm1 * dum1 * dlm1 / prbm1^2)[-i1,,drop = FALSE] +
       (xm1 * dlm1^2 / prbm1^2)[-i2,,drop = FALSE]
      )
     )

Z <- - .colSums(xm1 * (dpum1 / prbm1 -
                       dplm1 / prbm1 -
                       (dum1^2 / prbm1^2 -
                        2 * dum1 * dlm1 / prbm1^2 +
                        dlm1^2 / prbm1^2
                       )
                      ),
                m = nrow(xm1), n = ncol(xm1)
                )
if (length(Z) > 1L) Z <- diag(Z)
◇
```

Fragment referenced in 12.

We return the three matrices $\mathbf{A}$, $\mathbf{X}$, and $\mathbf{Z}$ necessary for two different purposes: We need the `full` Hessian for all parameters $\boldsymbol{\theta}$ as a dense `matrix` such that `nlminb` can compute updates from this object. In addition, the computation of the Fisher information for $\delta_2, \ldots, \delta_K$ as the Schur complement $\mathbf{Z} - \mathbf{X}^\top \mathbf{A}^{-1} \mathbf{X}$. Because the matrix $\mathbf{A}$ is symmetric tridiagonal, we use infrastructure from the **Matrix** package to represent this matrix:

⟨ *Hessian* 12 ⟩ ≡

```
.hes <- function(parm, x, mu = 0, rightcensored = FALSE, full = FALSE)
{

    ⟨ parm to prob 2 ⟩

    ⟨ Hessian prep 8 ⟩

    ⟨ off-diagonal elements for Hessian of intercepts 9 ⟩
    ⟨ diagonal elements for Hessian of intercepts 10 ⟩
    ⟨ intercept / shift contributions to Hessian 11 ⟩

    if (length(Adiag) > 1L) {
        if (!isFALSE(full)) {
            A <- list(Adiag = Adiag, Aoffdiag = Aoffdiag)
        } else {
            A <- Matrix::bandSparse(length(Adiag),
                k = 0:1, diagonals = list(Adiag, Aoffdiag),
                symmetric = TRUE)
        }
    } else {
        if (!isFALSE(full)) {
            A <- list(Adiag = Adiag, Aoffdiag = NULL)
        } else {
            A <- matrix(Adiag)
        }
    }
    return(list(A = A, X = X, Z = Z))
}
```

◇

Fragment referenced in 43.

We start with an example involving $K = 3$ groups for a binary outcome and use a binary logistic regression model to estimate the two log-odds ratios δ_2 and δ_3 along with their estimated covariance

```
R> library("free1way.docreg")
R> (x <- matrix(c(10, 5, 7, 11, 8, 9), nrow = 2))
```

```
     [,1] [,2] [,3]
[1,]   10    7    8
[2,]    5   11    9
```

```
R> d <- expand.grid(y = relevel(gl(2, 1), "2"), t = gl(3, 1))
R> d$x <- c(x)
R> m <- glm(y ~ t, data = d, weights = x, family = binomial())
R> (cf <- coef(m))
```

```
(Intercept)          t2          t3
    0.69315    -1.14513    -0.81093
```

Replicating these results requires specification of the inverse link function $F =$ expit and the density function f of the standard logistic. We use `optim` with numerically approximated Hessian to be able to check the correctness of the analytical Hessian. Note that `glm` operates with a positive linear predictor, so we need to change the sign of the log-odds ratios:

```
R> F <- plogis
R> f <- dlogis
R> op <- optim(par = c("mt2" = 0, "mt3" = 0, "(Intercept)" = 0),
+              fn = .nll, gr = .nsc,
+              x = x, method = "BFGS", hessian = TRUE)
R> cbind(glm = c(cf[-1] * -1, cf[1]), free1way = op$par)
```

```
                glm free1way
t2          1.14513  1.14513
t3          0.81093  0.81093
(Intercept) 0.69315  0.69315
```

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

```
'log Lik.' -33.33 (df=3)
```

```
R> -op$value
```

```
[1] -33.33
```

Parameter estimates and the in-sample log-likelihood are practically identical. We now turn to the inverse Hessian of the shift terms, first defining the derivative of the density of the standard logistic distribution

```
R> fp <- function(x)
+  {
+      p <- plogis(x)
+      p * (1 - p)^2 - p^2 * (1 - p)
+  }
R> H <- .hes(op$par, x)
R> ### analytical covariance of parameters
R> solve(H$Z - crossprod(H$X, Matrix::solve(H$A, H$X)))
```

```
        [,1]    [,2]
[1,] 0.53377 0.30000
[2,] 0.30000 0.53611
```

```
R> ### numerical covariance
R> solve(op$hessian)[1:2,1:2]
```

```
        mt2     mt3
mt2 0.53377 0.30000
mt3 0.30000 0.53611
```

```
R> ### from glm
R> vcov(m)[-1,-1]
```

```
        t2      t3
t2 0.53377 0.30000
t3 0.30000 0.53611
```

Also here we see practically identical results. We will later implement a low-level function `.free1way` taking a table and an object describing the inverse link F as arguments; these results are also in line with `glm`:

```
R> obj <- .free1wayML(as.table(x), link = logit())
R> obj$coefficients
```

```
      B       C
1.14513 0.81093
```

```
R> -obj$value
```

```
[1] -33.33
```

```
R> ### analytical covariance
R> obj$vcov
```

```
        B       C
B 0.53377 0.30000
C 0.30000 0.53611
```

In the next step, we extend our results to the stratified case. We iterate over all blocks and evaluate the negative log-likelihood for the same values of the shift parameters but block-specific values of the intercept parameters. Because `x` is a `table`, it may happen (especially in the presence of blocks) that some outcome values (rows) were not observed in any group (row sum is zero). Thus, the distribution function does not jump at this value and therefore no parameter for this value is needed. We remove these cases. In the right-censored case, we have to pay attention to censoring happening at these outcome values. We count how many obervations were censored between observations and assign the corresponding weights to the subsequent outcome value. Furthermore, we need to be able to undo the removal of observations later, mainly for computing residuals. We store an attribute `idx` for later use in `.snsr` on page 13.

⟨ *determine steps in blocks* 13 ⟩ ≡

```
xlist <- xrclist <- vector(mode = "list", length = B)

for (b in seq_len(B)) {
    xb <- matrix(x[,,b, drop = TRUE], ncol = K)
    ### remove undefined shift parameters
    ### determine rows w/o observation
    xw <- rowSums(abs(xb)) > 0L
    ### for penalised estimation, we do NOT want
    ### to remove blocks with constant outcomes
    if (MPL_Jeffreys > 0L && ### penalised likelihoods
        B >  1L &&           ### in the presence of blocks
        is.null(xrc)         ### absence of right-censoring
       )
    {
        ### keep one additional row w/o observation
        ### such that the block will not be removed
        ### NB: make sure starting values for intercepts
        ###     are still monotone increasing by assigning
        ###     not zero but a small number (.1) below
        if (sum(xw) == 1L && length(xw) > 1L)
            xw[which(!xw)[1L]] <- TRUE
    }
    if (sum(xw) > 1L &&         ### at least two different outcome values
        sum(abs(xb)) > 1L)      ### at least two observations per block
    {
        ### do not remove last parameter if there are corresponding
        ### right-censored observations
        wm <- which(xw)[sum(xw)]
        if (!is.null(xrc) && any(xrc[wm:dx[1],,b,drop = TRUE] > 0)) {
            ### determine last right-censoring time in this block
            wrc <- rowSums(abs(xrc[,,b,drop = TRUE])) > 0
            lastrc <- which(wrc)[sum(wrc)]
            xw[lastrc] <- TRUE
        }
        xlist[[b]] <- xb[xw,,drop = FALSE]
        Cidx <- rep.int(1L, times = C)
        Cidx[xw] <- Cidx[xw] + seq_len(sum(xw))
        attr(xlist[[b]], "idx") <- Cidx
        if (!is.null(xrc)) {
            ### count right-censored observations between distinct event
            ### times
            cs <- apply(xrc[,,b,drop = TRUE] * (!xw), 2, function(x)
                diff(c(0, cumsum(x)[xw])))
            xrclist[[b]] <- matrix(xrc[xw,,b,drop = TRUE], ncol = K) + cs
            idx <- seq_len(C)[xw]
            idx <- rep(seq_len(sum(xw)), times = c(idx[1], diff(idx)))
            Cidx <- rep.int(1L, times = C)
            Cidx[seq_along(idx)] <- Cidx[seq_along(idx)] + idx
            attr(xrclist[[b]], "idx") <- Cidx
        }
    }
}
### remove empty blocks
strata <- !vapply(xlist, is.null, NA)
xlist <- xlist[strata]
xrclist <- xrclist[strata]
```

◇

Fragment referenced in 14.

Before we begin, we convert the table $C \times K \times B(\times 2)$ table `x` into a list of non-empty $C' \times K$ tables (yet

still allowing zero row sums):

⟨ *table2list body* 14 ⟩ ≡

```
dx <- dim(x)
if (length(dx) == 1L)
    stop("incorrect dimensions")
if (length(dx) == 2L)
    x <- as.table(array(x, dim = c(dx, 1)))
dx <- dim(x)
if (length(dx) < 3L)
    stop("incorrect dimensions")
C <- dim(x)[1L]
K <- dim(x)[2L]
B <- dim(x)[3L]
if (C < 2L)
    stop("at least two response categories required")
if (K < 2L)
    stop("at least two groups required")
xrc <- NULL
if (length(dx) == 4L) {
    if (dx[4] == 2L) {
        xrc <- array(x[,,,"FALSE", drop = TRUE], dim = dx[1:3])
        x <- array(x[,,,"TRUE", drop = TRUE], dim = dx[1:3])
    } else {
        stop(gettextf("%s currently only allows independent right-censoring",
                      "free1way"),
             domain = NA)
    }
}

⟨ determine steps in blocks 13 ⟩
◇
```

Fragment referenced in 34.

We first extract the shift parameters δ. and then, separately for each stratum, the corresponding contrasts of the intercept parameters:

⟨ *stratum prep* 15 ⟩ ≡

```
C <- vapply(x, NROW, 0L) ### might differ by stratum
K <- unique(do.call("c", lapply(x, ncol))) ### the same
B <- length(x)
sidx <- factor(rep(seq_len(B), times = pmax(0, C - 1L)),
               levels = seq_len(B))
bidx <- seq_len(K - 1L)
delta <- parm[bidx]
intercepts <- split(parm[-bidx], sidx)
◇
```

Fragment referenced in 16, 17, 18, 19, 21, 40.

before we loop over the non-empty strata and return the sum of the corresponding log-likelihoods:

⟨ *stratified negative logLik* 16 ⟩ ≡

```
.snll <- function(parm, x, mu = 0, rightcensored = FALSE)
{

    ⟨ stratum prep 15 ⟩

    ret <- 0
    for (b in seq_len(B))
        ret <- ret + .nll(c(delta, intercepts[[b]]), x[[b]], mu = mu,
                          rightcensored = rightcensored)
    return(ret)
}
```

◇

Fragment referenced in 43.

In a similar way, we evaluate the gradients for each block and sum-up the contributions by the shift parameters whereas the gradients for the intercept parameters are only concatenated:

⟨ *stratified negative score* 17 ⟩ ≡

```
.snsc <- function(parm, x, mu = 0, rightcensored = FALSE)
{

    ⟨ stratum prep 15 ⟩

    ret <- numeric(length(bidx))
    for (b in seq_len(B)) {
        nsc <- .nsc(c(delta, intercepts[[b]]), x[[b]], mu = mu,
                    rightcensored = rightcensored)
        ret[bidx] <- ret[bidx] + nsc[bidx]
        ret <- c(ret, nsc[-bidx])
    }
    return(ret)
}
```

◇

Fragment referenced in 43.

The score residuum is zero for an observation with weight zero, that is, a row of zeros in the table:

⟨ *stratified negative score residual* 18 ⟩ ≡

```
.snsr <- function(parm, x, mu = 0, rightcensored = FALSE)
{

    ⟨ stratum prep 15 ⟩

    ret <- c()
    for (b in seq_len(B)) {
        idx <- attr(x[[b]], "idx")
        ### idx == 1L means zero residual, see definition of idx
        sr <- c(0, .nsr(c(delta, intercepts[[b]]), x[[b]], mu = mu,
                        rightcensored = rightcensored))
        ret <- c(ret, sr[idx])
    }
    return(ret)
}
```

◇

Fragment referenced in 43.

and repeat the same exercise for scale residuals

⟨ *stratified negative scale score residual* 19 ⟩ ≡

```
.snssr <- function(parm, x, mu = 0, rightcensored = FALSE)
{

    ⟨ stratum prep 15 ⟩

    ret <- c()
    for (b in seq_len(B)) {
        idx <- attr(x[[b]], "idx")
        ### idx == 1L means zero residual, see definition of idx
        sr <- c(0, .nssr(c(delta, intercepts[[b]]), x[[b]], mu = mu,
                         rightcensored = rightcensored))
        ret <- c(ret, sr[idx])
    }
    return(ret)
}
```

◇

Fragment referenced in 43.

```
R> (x <- as.table(array(c(10, 5, 7, 11, 8, 9,
+                         9, 4, 8, 15, 5, 4), dim = c(2, 3, 2))))

, , A

   A  B  C
A 10  7  8
B  5 11  9

, , B

   A  B  C
A  9  8  5
B  4 15  4

R> d <- expand.grid(y = relevel(gl(2, 1), "2"), t = gl(3, 1), s = gl(2, 1))
R> d$x <- c(x)
R> m <- glm(y ~ 0 + s + t, data = d, weights = x, family = binomial())
R> logLik(m)

'log Lik.' -62.523 (df=4)

R> (cf <- coef(m))

      s1       s2       t2       t3
 0.72433  0.77385 -1.30224 -0.74147

R> xl <- .table2list(x)$xlist
R> op <- optim(par = c("mt2" = 0, "mt3" = 0, "(Intercept 1)" = 0, "(Intercept 2)" = 0),
+              fn = .snll, gr = .snsc,
+              x = xl,
+              method = "BFGS",
+              hessian = TRUE)
R> cbind(glm = c(cf[-(1:2)] * -1, cf[1:2]), free1way = op$par)

       glm free1way
t2 1.30224  1.30225
t3 0.74147  0.74149
s1 0.72433  0.72435
s2 0.77385  0.77386
```

```
R> logLik(m)

'log Lik.' -62.523 (df=4)

R> -op$value

[1] -62.523
```

For the analytical Hessian, we sum-up over the stratum-specific Hessians of the shift parameters. For right-censored observations, we need to compute the contributions by the events and obtain the joint Hessian for shift- and intercept parameters first. We differentiate between computing the null Hessian for $\boldsymbol{\theta}$ as a dense `matrix`:

⟨ *full Hessian* 20 ⟩ ≡

```
for (b in seq_len(B)) {
    H <- .hes(c(delta, intercepts[[b]]), x[[b]], mu = mu, full = full)
    if (!is.null(xrc)) {
        Hrc <- .hes(c(delta, intercepts[[b]]), xrc[[b]], mu = mu,
                    rightcensored = TRUE, full = full)
        H$X <- H$X + Hrc$X
        H$A$Adiag <- H$A$Adiag + Hrc$A$Adiag
        H$A$Aoffdiag <- H$A$Aoffdiag + Hrc$A$Aoffdiag
        H$Z <- H$Z + Hrc$Z
    }
    if (b == 1L) {
        Adiag <- H$A$Adiag
        Aoffdiag <- H$A$Aoffdiag
        X <- H$X
        Z <- H$Z
    } else {
        Adiag <- c(Adiag, H$A$Adiag)
        Aoffdiag <- c(Aoffdiag, 0, H$A$Aoffdiag)
        X <- rbind(X, H$X)
        Z <- Z + H$Z
    }
}

if (length(Adiag) > 1L) {
    A <- Matrix::bandSparse(length(Adiag),
                            k = 0:1, diagonals = list(Adiag, Aoffdiag),
                            symmetric = TRUE)
} else {
    A <- matrix(Adiag)
}

ret <- cbind(Z, t(X))
ret <- rbind(ret, cbind(X, A))
if (retMatrix) return(ret)
return(as.matrix(ret))
◇
```

Fragment referenced in 21.

and the computation of the Hessian for the shift parameters using `Matrix` technology:

⟨ *stratified Hessian* 21 ⟩ ≡

```
.shes <- function(parm, x, mu = 0, xrc = NULL, full = FALSE,
                  retMatrix = FALSE)
{

    ⟨ stratum prep 15 ⟩

    if (!isFALSE(ret <- full)) {
        ⟨ full Hessian 20 ⟩
    }
    ret <- matrix(0, nrow = length(bidx), ncol = length(bidx))
    for (b in seq_len(B)) {
        H <- .hes(c(delta, intercepts[[b]]), x[[b]], mu = mu)
        if (!is.null(xrc)) {
            Hrc <- .hes(c(delta, intercepts[[b]]), xrc[[b]], mu = mu,
                        rightcensored = TRUE)
            H$X <- H$X + Hrc$X
            H$A <- H$A + Hrc$A
            H$Z <- H$Z + Hrc$Z
        }
        ### needs Matrix::solve
        sAH <- tryCatch(Matrix::solve(H$A, H$X), error = function(e) NULL)
        if (is.null(sAH))
            stop(gettextf("error computing the Hessian in %s",
                          "free1way"),
                 domain = NA)
        ret <- ret + (H$Z - crossprod(H$X, sAH))
    }
    as.matrix(ret)
}
◇
```

Fragment referenced in 43.

```
R> ### analytical covariance of parameters
R> solve(.shes(op$par, xl))

        [,1]    [,2]
[1,] 0.27095 0.16154
[2,] 0.16154 0.32006

R> ### numerical covariance
R> solve(op$hessian)[1:2,1:2]

        mt2     mt3
mt2 0.27095 0.16154
mt3 0.16154 0.32006

R> ### from glm
R> vcov(m)[-(1:2),-(1:2)]

        t2      t3
t2 0.27095 0.16153
t3 0.16153 0.32006

R> obj <- .free1wayML(as.table(x), link = logit())
R> obj$coefficients

      B       C
1.30224 0.74147

R> -obj$value
```

```
[1] -62.523
```

```
R> ### analytical covariance
R> obj$vcov
```

```
        B       C
B 0.27095 0.16153
C 0.16153 0.32006
```

Chapter 3

Link Functions

Similar to `family` objects, we provide some infrastructure for `link` functions F^{-1} and derived quantities (`linkinv` F, `dlinkinv` f, and `ddlinkinv` f'). If not provided, we also set-up the ratio f'/f in the constructor.

Although there is some overlap with `family` objects for binomial outcomes, it doesn't seem beneficial to extend this richer class.

"linkfun.R" 22≡

```
#  File src/library/stats/R/linkfun.R
#  generated by running nuweb on
#       system.file("free1way.w", package = "free1way.docreg")
#  Part of the R package, https://www.R-project.org
#
#  Copyright (C) 2026 The R Core Team
#
#  This program is free software; you can redistribute it and/or modify
#  it under the terms of the GNU General Public License as published by
#  the Free Software Foundation; either version 2 of the License, or
#  (at your option) any later version.
#
#  This program is distributed in the hope that it will be useful,
#  but WITHOUT ANY WARRANTY; without even the implied warranty of
#  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#  GNU General Public License for more details.
#
#  A copy of the GNU General Public License is available at
#  https://www.R-project.org/Licenses/

⟨ linkfun 23 ⟩
⟨ logit 24 ⟩
⟨ probit 27 ⟩
⟨ cloglog 26 ⟩
⟨ loglog 25 ⟩
◇
```

⟨ *linkfun* 23 ⟩ ≡

```
### distribution function
.p <- function(link, q, ...)
    link$linkinv(q = q, ...)

### quantile function
.q <- function(link, p, ...)
    link$link(p = p, ...)

### density function
.d <- function(link, x, ...)
    link$dlinkinv(x = x, ...)

### derivative of density function
.dd <- function(link, x, ...)
    link$ddlinkinv(x = x, ...)

### 2nd derivative of density function
.ddd <- function(link, x, ...)
    link$dddlinkinv(x = x, ...)

### ratio of derivative of density to
### density function
.dd2d <- function(link, x, ...)
    link$dd2dlinkinv(x = x, ...)

### constructor
linkfun <- function(name,       ### nickname
                    alias,      ### char
                    model,      ### char, semiparametric model name
                    parm,       ### char, parameter name
                    link,       ### quantile function
                    linkinv,    ### distribution function
                    dlinkinv,   ### density function
                    ddlinkinv,  ### derivative of density function
                    ...)
{

    ret <- list(name = name,
                alias = alias,
                model = model,
                parm = parm,
                link = link,
                linkinv = linkinv,
                dlinkinv = dlinkinv,
                ddlinkinv = ddlinkinv)
    if (is.null(ret$dd2d))
        ret$dd2d <- function(x)
            ret$ddlinkinv(x) / ret$dlinkinv(x)
    ret <- c(ret, list(...))
    class(ret) <- "linkfun"
    ret
}
◇
```

Fragment referenced in 22.

We start with the logit link, that is $F(z) = (1 + \exp(-z))^{-1}$, giving rise to Wilcoxon or Kruskal-Wallis type score residuals:

⟨ *logit* 24 ⟩ ≡

```
logit <- function()
    linkfun(name = "Logit",
            alias = c("Wilcoxon", "Kruskal-Wallis"),
            model = "proportional odds",
            parm = "log-odds ratio",
            link = qlogis,
            linkinv = plogis,
            dlinkinv = dlogis,
            ddlinkinv = function(x) {
                p <- plogis(x)
                p * (1 - p)^2 - p^2 * (1 - p)
            },
            dddlinkinv = function(x) {
                ex <- exp(x)
                ifelse(is.finite(x), (ex - 4 * ex^2 + ex^3) / (1 + ex)^4, 0.0)
            },
            dd2d = function(x) {
                ex <- exp(x)
                (1 - ex) / (1 + ex)
            },
            parm2PI = function(x) {
               OR <- exp(x)
               ret <- OR * (OR - 1 - x)/(OR - 1)^2
               ret[abs(x) < .Machine$double.eps] <- 0.5
               return(ret)
            },
            PI2parm = function(p) {
               f <- function(x, PI)
                   x + (exp(-x) * (PI +
                                   exp(2 * x) * (PI - 1) +
                                   exp(x) * (1 - 2 * PI)))
               ret <- vapply(p, function(p)
                   uniroot(f, PI = p, interval = 50 * c(-1, 1))$root, 0)
               return(ret)
            },
            parm2OVL = function(x) 2 * plogis(-abs(x / 2))
    )
```

◇

Fragment referenced in 22.

The `parm2PI` function converts log-odds ratios to probabilistic indices (or AUCs) and the inverse operation is implemented by `PI2parm`. The overlap coefficient can be obtained from a log-odds ratio via `parm2OVL`.

The log-log link, with $F(z) = \exp(-\exp(-z))$, is used to construct tests against Lehmann alternatives:

⟨ *loglog* 25 ⟩ ≡

```
loglog <- function()
    linkfun(name = "Log-log",
            alias = "Lehmann",
            model = "Lehmann",
            parm = "log-reverse time hazard ratio",
            link = function(p, log.p = FALSE) {
                if (!log.p) p <- log(p)
                -log(-p)
            },
            linkinv = function(q, lower.tail = TRUE, log.p = FALSE) {
                ### p = exp(-exp(-q))
                if (log.p) {
                    if (lower.tail)
                        return(-exp(-q))
                    return(pexp(exp(-q), log.p = TRUE))
                }
                if (lower.tail)
                    return(exp(-exp(-q)))
                -expm1(-exp(-q))
            },
            dlinkinv = function(x, log = FALSE) {
                ret <- - x - exp(-x)
                if (log)
                    return(ifelse(is.finite(x), ret, -Inf))
                ifelse(is.finite(x), exp(ret), 0.0)
            },
            ddlinkinv = function(x) {
               ex <- exp(-x)
               ifelse(is.finite(x), exp(-ex - x) * (ex - 1.0), 0.0)
            },
            dddlinkinv = function(x) {
               ex <- exp(-x)
               ifelse(is.finite(x), exp(-x - ex) * (ex - 1)^2 -
                                    exp(-ex - 2 * x),
                                    0.0)
            },
            dd2d = function(x)
                expm1(-x),
            parm2PI = plogis,
            PI2parm = qlogis,
            parm2OVL = function(x) {
                x <- abs(x)
                rt <- exp(-x / (exp(x) - 1))
                ret <- rt^exp(x) + 1 - rt
                ret[abs(x) < .Machine$double.eps] <- 1
                x[] <- ret
                return(x)
            }
    )
◇
```

Fragment referenced in 22.

The complementary log-log link, with $F(z) = 1 - \exp(-\exp(z))$, provides log-rank or Savage score residuals against proportional hazards alternatives:

⟨ *cloglog* 26 ⟩ ≡

```
cloglog <- function()
    linkfun(name = "Complementary Log-log",
            alias = "Savage",
            model = "proportional hazards",
            parm = "log-hazard ratio",
            link = function(p, log.p = FALSE) {
                if (log.p) p <- exp(p)
                log(-log1p(- p))
            },
            linkinv = function(q, lower.tail = TRUE, log.p = FALSE) {
                ### p = 1 - exp(-exp(q))
                ret <- exp(-exp(q))
                if (log.p) {
                    if (lower.tail)
                        return(log1p(-ret))
                    return(-exp(q))
                }
                if (lower.tail)
                    return(-expm1(-exp(q)))
                return(ret)
            },
            dlinkinv = function(x, log = FALSE) {
                ret <- x - exp(x)
                if (log)
                    return(ifelse(is.finite(x), ret, -Inf))
                ifelse(is.finite(x), exp(ret), 0.0)
            },
            ddlinkinv = function(x) {
                ex <- exp(x)
                ifelse(is.finite(x), (ex - ex^2) / exp(ex), 0.0)
            },
            dddlinkinv = function(x) {
                ex <- exp(x)
                ifelse(is.finite(x), (ex - 3*ex^2 + ex^3) / exp(ex), 0.0)
            },
            dd2d = function(x)
               -expm1(x),
            parm2PI = plogis,
            PI2parm = qlogis,
            parm2OVL = function(x) {
                x <- abs(x)
                ret <- exp(x / (exp(-x) - 1)) - exp(-x / (exp(x) - 1)) + 1
                ret[abs(x) < .Machine$double.eps] <- 1
                x[] <- ret
                return(x)
            }
    )
```

◇

Fragment referenced in 22.

The probit link, with $F(z) = \Phi$, leads to normal scores tests, where the shift effect can be interpreted as a generalised version of Cohen's d, that is, differences on a latent normal scale with variance one:

⟨ *probit* 27 ⟩ ≡

```
probit <- function()
    linkfun(name = "Probit",
            alias = "van der Waerden normal scores",
            model = "latent normal shift",
            parm = "generalised Cohen's d",
            link = qnorm,
            linkinv = pnorm,
            dlinkinv = dnorm,
            ddlinkinv = function(x)
                ifelse(is.finite(x), -dnorm(x = x) * x, 0.0),
            dddlinkinv = function(x)
                ifelse(is.finite(x), dnorm(x = x) * (x^2 - 1), 0.0),
            dd2d = function(x) -x,
            parm2PI = function(x) pnorm(x, sd = sqrt(2)),
            PI2parm = function(p) qnorm(p, sd = sqrt(2)),
            parm2OVL = function(x) 2 * pnorm(-abs(x / 2))
    )
```

◇

Fragment referenced in 22.

Chapter 4

Optimisation

Harrell (2024) reports on experiments with a number of optimisers for the specific optimisation problem arising here and recommends a Newton-Raphson algorithm leveraging the sparse matrix structure of the observed Fisher information matrix. The following code was adopted from his **rms** package, function `rms:::lrm.fit`.

⟨ *Newton update* 28 ⟩ ≡

```
gradthe <- gradient(theta)     # Compute the gradient vector
hessthe <- hessian(theta)      # Compute the Hessian matrix

delta <- tryCatch(Matrix::solve(hessthe, gradthe,
                                tol = control$tolsolve),
                  error = function(e) NULL)
if (is.null(delta)) {
    msg <- "computing Newton update failed in .NewtonRaphson"
    return(list(par = theta, objective = objthe, convergence = 1,
                message = msg))
}

if (control$trace)
    cat(iter, ': ', theta, "\n", sep = "")

step_size <- 1L                # Initialize step size for step-halving
```

◇

Fragment referenced in 31.

⟨ *Newton step halving* 29 ⟩ ≡

```
new_theta <- theta - step_size * delta # Update parameter vector
objnew_the <- objective(new_theta)

if (control$trace)
    cat("Old, new, old - new objective:",
        objthe, objnew_the, objthe - objnew_the, "\n")

# Objective function failed to be reduced or is infinite
if (!is.finite(objnew_the) || (objnew_the > objthe + 1e-6)) {
    step_size <- step_size / 2         # Reduce the step size

    if (control$trace)
        cat("Step size reduced to", step_size, "\n")

    if (step_size <= control$minstepsize) {
        msg <- paste("step size ", step_size,
                     " has reduced below minstepsize")
        return(list(par = theta, objective = objthe, convergence = 1,
                    message = msg))
    }
} else {
    theta  <- new_theta # accept the new parameter vector
    oldobj <- objthe
    objthe <- objnew_the
    break
}
◇
```

Fragment referenced in 31.

⟨ *Newton convergence* 30 ⟩ ≡

```
# Convergence check - must meet 3 criteria
if ((objthe <= oldobj + 1e-6 && (oldobj - objthe < control$objtol)) &&
    (max(abs(gradthe)) < control$gradtol) &&
    (max(abs(delta)) < control$paramtol))

    return(list(par            = theta,
                objective      = objthe,
                convergence    = 0,
                message        = "Normal convergence"))
◇
```

Fragment referenced in 31.

⟨ *NewtonRaphson* 31 ⟩ ≡

```
### adopted from rms:::lrm.fit
.NewtonRaphson <- function(start, objective, gradient, hessian,
                           control = list(iter.max = 150, trace = trace,
                                          objtol = 5e-4, gradtol = 1e-5,
                                          paramtol = 1e-5, minstepsize = 1e-2,
                                          tolsolve = .Machine$double.eps),
                           trace = FALSE)
{

    theta  <- start # Initialize the parameter vector
    oldobj <- Inf
    objthe <- objective(theta)
    if (!is.finite(objthe)) {
        msg <- "infeasible starting values"
        return(list(par = theta, objective = objthe, convergence = 1,
                    message = msg))
    }

    if(!suppressPackageStartupMessages(requireNamespace("Matrix")))
        stop(gettextf("%s needs package 'Matrix' correctly installed",
                      ".NewtonRaphson"),
             domain = NA)

    for (iter in seq_len(control$iter.max)) {

        ⟨ Newton update 28 ⟩

        # Step-halving loop
        repeat {
            ⟨ Newton step halving 29 ⟩
        }

        ⟨ Newton convergence 30 ⟩
    }

    msg <- paste("reached", control$iter.max, "iterations without convergence")
    return(list(par = theta, objective = objthe, convergence = 1, message = msg))
}
```

◇

Fragment referenced in 33.

We can now test the optimiser on a least-squares problem

```
R> N <- 10000
R> P <- 30
R> X <- matrix(rnorm(N * P), ncol = P)
R> y <- X %*% runif(P) + rnorm(nrow(X))
R> f <- function(par) sum((y - X %*% par)^2)
R> g <- function(par) colSums(- 2 * c(y - X %*% par) * X)
R> h <- function(par) 2 * crossprod(X)
R> start <- runif(P)
R> cf <- .NewtonRaphson(start = start, objective = f, gradient = g, hessian = h)
R> cf2 <- coef(m <- lm(y ~ 0 + X))
R> all.equal(sum((y - fitted(m))^2), cf$objective)

[1] TRUE

R> all.equal(unname(cf$par), unname(cf2))

[1] TRUE
```

Chapter 5

ML Estimation

We use one internal **stats** function, which is fine in **stats** but not when running the code as part of the **free1way.docreg** package, so we include it here

"utils.R" 32≡

```
### gives warnings but no diffs
assert_NULL_or_prob <- stats:::assert_NULL_or_prob
```
◇

The file `free1way.R` goes into `src/library/stats/R`, so we add copyright statement here as well (such that we can simply copy the file in case of updates).

"free1way.R" 33≡

```
#  File src/library/stats/R/free1way.R
#  generated by running nuweb on
#       system.file("free1way.w", package = "free1way.docreg")
#  Part of the R package, https://www.R-project.org
#
#  Copyright (C) 2026 The R Core Team
#
#  This program is free software; you can redistribute it and/or modify
#  it under the terms of the GNU General Public License as published by
#  the Free Software Foundation; either version 2 of the License, or
#  (at your option) any later version.
#
#  This program is distributed in the hope that it will be useful,
#  but WITHOUT ANY WARRANTY; without even the implied warranty of
#  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#  GNU General Public License for more details.
#
#  A copy of the GNU General Public License is available at
#  https://www.R-project.org/Licenses/

⟨ NewtonRaphson 31 ⟩
⟨ ML estimation 43 ⟩
⟨ free1way generic and table method (main workhorse) 54 ⟩
⟨ free1way methods 61 ⟩
⟨ free1way print 62 ⟩
⟨ free1way summary 63 ⟩
⟨ free1way confint 68 ⟩
⟨ MPLfree1way methods 92 ⟩
⟨ free1way formula 57 ⟩
⟨ free1way numeric 59 ⟩
⟨ free1way factor 60 ⟩
⟨ plot free1way 78 ⟩
⟨ ppplot 80 ⟩
⟨ rfree1way 82 ⟩
⟨ power 91 ⟩
◇
```

We now put together a low-level function for parameter estimation and evaluation of scores, Hessians, and residuals. We also set-up a profile likelihood function for later re-use.

Assuming all shift effects been zero, we compute starting values for the intercept parameters from the empirical cumulative distribution function after merging all treatment groups:

⟨ *setup and starting values* 34 ⟩ ≡

```
⟨ table2list body 14 ⟩
## allow specification of start = delta and fix = 1:K
## for evaluating the likelihood at given delta parameters
## without having to specify all intercept parameters
if (is.null(start))
    start <- rep.int(0, K - 1L)
NS <- length(start) == (K - 1L)
lwr <- rep(-Inf, times = K - 1L)
for (b in seq_len(length(xlist))) {
    bC <- nrow(xlist[[b]]) - 1L
    lwr <- c(lwr, -Inf, rep.int(0, times = bC - 1L))
    if (NS) {
        ecdf0 <- cumsum(pmax(.1, rowSums(xlist[[b]])))
        ### ensure that 0 < ecdf0 < 1 such that quantiles exist
        ecdf0 <- pmax(1, ecdf0[-length(ecdf0)]) / (max(ecdf0) + 1)
        Qecdf <- Q(ecdf0)
        start <- c(start, Qecdf)
    }
}
◇
```

Fragment referenced in 43.

The profile negative log-likelihood can be evaluated for some of the parameters in $\boldsymbol{\theta}$ (denoted as `fix`), the remaining parameters are updated. Note that `start` can either just contain a subset of the shift parameter or must contain the full and feasible (meeting monotonicity constraints for the intercept parameters) parameter vector $\boldsymbol{\theta}$.

We call `nlminb` and will increase the `eval.max` and `iter.max` control parameters if we encounter optimisation issues and restart at the current solution:

⟨ *do optim* 35 ⟩ ≡

```
maxit <- control[[1L]]$iter.max
while(maxit < 10001) {
   ret <- do.call(names(control)[[1L]], opargs)
   maxit <- 5 * maxit
   if (ret$convergence > 0) {
       opargs$control$eval.max <- maxit
       opargs$control$iter.max <- maxit
       opargs$start <- ret$par
   } else {
       break()
   }
}

if (ret$convergence > 0)
    warning(gettextf("unsuccessful optimisation in %s", "free1way"),
            ": ", ret$message, domain = NA)

ret$value <- ret$objective
ret$objective <- NULL
◇
```

Fragment referenced in 37, 39.

We first set-up the target function (the negative log-likelihood, also dealing with right-censoring) and the corresponding gradient. We then add the profile negative log-likelihood, which in turn calls the two functions defined first. We start with the log-likelihood, its gradient and Hessian

⟨ *logLik, gradient, Hessian* 36 ⟩ ≡

```
fn <- function(par)
{
    ret <- .snll(par, x = xlist, mu = mu)
    if (!is.null(xrc))
        ret <- ret + .snll(par, x = xrclist, mu = mu,
                           rightcensored = TRUE)
    return(ret)
}
gr <- function(par)
{
    ret <- .snsc(par, x = xlist, mu = mu)
    if (!is.null(xrc))
        ret <- ret + .snsc(par, x = xrclist, mu = mu,
                           rightcensored = TRUE)
    return(ret)
}

### allocate memory for hessian
Hess <- Matrix::Matrix(0, nrow = length(start), ncol = length(start))

he <- function(par)
{
    if (!is.null(xrc)) {
        ret <- .shes(par, x = xlist, mu = mu, xrc = xrclist, full = Hess,
                     retMatrix = names(control)[1L] == ".NewtonRaphson")
    } else {
        ret <- .shes(par, x = xlist, mu = mu, full = Hess,
                     retMatrix = names(control)[1L] == ".NewtonRaphson")
    }
    return(ret)
}
◇
```

Fragment referenced in 37.

and define the profile log-likelihood based on these functions

⟨ *profile* 37 ⟩ ≡

⟨ *logLik, gradient, Hessian* 36 ⟩

```
.profile <- function(start, fix = seq_len(K - 1))
{
    if (!all(fix %in% seq_len(K - 1)))
        stop(gettextf("invalid argument '%s'", "fix"), domain = NA)
    delta <- start[fix]
    opargs <- list(start = start[-fix],
                   objective = function(par) {
                       p <- numeric(length(par) + length(fix))
                       p[fix] <- delta
                       p[-fix] <- par
                       fn(p)
                   },
                   gradient = function(par) {
                       p <- numeric(length(par) + length(fix))
                       p[fix] <- delta
                       p[-fix] <- par
                       gr(p)[-fix]
                   },
                   hessian = function(par) {
                       p <- numeric(length(par) + length(fix))
                       p[fix] <- delta
                       p[-fix] <- par
                       he(p)[-fix, -fix, drop = FALSE]
                   })
    opargs$control <- control[[1L]]
    MPL_Jeffreys <- FALSE ### turn off Jeffreys penalisation in .profile

    ⟨ do optim 35 ⟩

    ret$MPL_Jeffreys <- MPL_Jeffreys

    p <- numeric(length(start))
    p[fix] <- delta
    p[-fix] <- ret$par
    ret$par <- p
    ret
}
```

◇

Fragment referenced in 43.

Chapter 13 introduces a bias correction (Firth, 1993), essentially by adding a penalty (Jeffreys prior) term to the log-likelihood. We start computing the log-determinant of the Hessian. We write

$$\mathbf{H}(\delta_2, \ldots, \delta_K, \vartheta_1, \ldots, \vartheta_{C-1}) = \begin{pmatrix} \mathbf{Z} & \mathbf{X} \\ \mathbf{X}^\top & \mathbf{A} \end{pmatrix}$$

and compute its determinant via

$$\det(\mathbf{H}) = \det(\mathbf{Z} - \mathbf{X}\mathbf{A}^{-1}\mathbf{X}^\top)\det(\mathbf{A}).$$

Because $\mathbf{A}$ is tridiagnonal, the quadratic form can be computed relatively fast. The dimension of $\mathbf{Z} - \mathbf{X}\mathbf{A}^{-1}\mathbf{X}^\top$ is $(K-1) \times (K-1)$ and thus the determinant is also relatively simple to compute. For the determinant of $\mathbf{A}$, we partition $\mathbf{A}$ as

$$\mathbf{A}^{(k)} = \begin{pmatrix} \mathbf{A}^{(k-1)} & \mathbf{A}_{k,1:k} \\ \mathbf{A}_{1:k,k} & \mathbf{A}_{k,k} \end{pmatrix}$$

for each $k = 2$ to the number of rows of $\mathbf{A}$. Here, $\mathbf{A}_{1:k,k} = (0, \ldots, 0, \mathbf{A}_{k-1,k})$. The determinant of the

tridiagonal matrix $\mathbf{A}$ is computed via its generalized continuant [1]

$$\det\left(\mathbf{A}^{(k-1)}\right) \det\left(\mathbf{A}_{k,k} - \mathbf{A}^2_{k-1,k}(\mathbf{A}^{(k-1)}_{k-1,k-1})^{-1}\right)$$

which simplifies[2] to

$$\det\left(\mathbf{A}^{(k)}\right) = \mathbf{A}_{k,k} \det\left(\mathbf{A}^{(k-1)}\right) - \mathbf{A}^2_{k-1,k} \det\left(\mathbf{A}^{(k-2)}\right).$$

We certainly want to compute this on the log-scale, so

$$\log\left(\det\left(\mathbf{A}^{(k)}\right)\right) = \log\left(\mathbf{A}_{k,k} \det\left(\mathbf{A}^{(k-1)}\right) - \mathbf{A}^2_{k-1,k} \det\left(\mathbf{A}^{(k-2)}\right)\right).$$

It would be possible to speed this up a little by iterating over the block diagonal matrix $\mathbf{A}$ in case of blocks, however, the code below also works well with some off-diagonal elements being zero:

⟨ *logdet* 38 ⟩ ≡

```
ld <- function(H) {
    bidx <- seq_len(K - 1)
    Z <- as.matrix(H[bidx, bidx, drop = FALSE])
    X <- as.matrix(H[-bidx, bidx, drop = FALSE])
    A <- H[-bidx,-bidx, drop = FALSE]

    ### needs Matrix::solve
    sAX <- tryCatch(Matrix::solve(A, X), error = function(e) NULL)
    if (is.null(sAX))
        stop(gettextf("error computing the Hessian determinant in %s",
                      "free1way"),
             domain = NA)

    ret <- determinant(Z - crossprod(X, as.matrix(sAX)), logarithm = TRUE)$modulus

    dm1 <- log(A[1,1])
    dm2 <- 0
    for (i in seq_len(nrow(A))[-1L]) {
        loga <- log(A[i, i]) + dm1
        logb <- (2 * log(abs(A[i-1, i])) + dm2)
        dm2 <- dm1
        dm1 <- loga + pexp(loga - logb, log.p = TRUE)
    }
    ret <- ret + dm1
    ret
}
```

◇

Fragment referenced in 43.

The code uses the log-sum-exp trick explained at the beginning of Chapter 2.

The `MPL_Jeffreys` argument triggers this bias correction via penalisation with Jeffreys prior to by applied when `TRUE` (with shrinkage parameter 0.5 or with `.5 * MPL_Jeffreys` of `MPL_Jeffreys` is numeric) or not to be applied when `FALSE`. To be more precise, with $\boldsymbol{\theta}$ denoting all parameters of the model and $\ell(\boldsymbol{\theta})$ the corresponding log-likelihood with Fisher information $\mathbf{H}(\boldsymbol{\theta})$, the penalised log-likelihood is

$$\ell(\boldsymbol{\theta}) + \frac{1}{2}\det(\mathbf{H}(\boldsymbol{\theta}))$$

(Section 3.1. in Firth, 1993) and thus the negative log-likelihood features $-\frac{1}{2}\det(\mathbf{H}(\boldsymbol{\theta}))$ as penality term. Maximum penalised likelihood estimation in `free1way` is still experimental and needs more testing. It also seems unclear if and how the Fisher information needs additional correction and it is certainly unclear if one can proceed with permutation testing after correcting the bias.

[1] https://en.wikipedia.org/wiki/Continuant_(mathematics)

[2] https://math.stackexchange.com/questions/266998/how-to-compute-the-determinant-of-a-tridiagonal-toeplitz-matrix

With the optimisation problem now properly defined, we can proceed trying to solve it. The heart of the optimisation chunk is a call to `.NewtonRaphson` or `nlminb`, trying to obtain parameter estimates of $\boldsymbol{\theta}$ by minimising the negative log-likelihood. We allow some (or all) parameters to be fixed at some constants, and provide a profile version of the likelihood. For the penalised version, we only proceed when `nlminb` was specified as an optimiser, because it also works without gradients and Hessians (maybe we can provide these later). We also handle inadmissible parameter configurations (non-finite log-likelihood) in the objective, as we optimise without gradients, hessians, or constraints at the moment.

⟨ *optim* 39 ⟩ ≡

```
if (!length(fix)) {
    if ((MPL_Jeffreys > 0L)) {
        opargs <- list(start = start,
                       ### note: negative log-likelihood!
                       objective = function(par) {
                           ret <- fn(par)
                           if (!is.finite(ret)) return(Inf)
                           return(ret - .5 * MPL_Jeffreys * ld(he(par)))
                       },
                       gradient = NULL,
                       hessian = NULL)
    } else {
        opargs <- list(start = start,
                       objective = fn,
                       gradient = gr,
                       hessian = he)
    }
    opargs$control <- control[[1L]]
    ⟨ do optim 35 ⟩
    ret$MPL_Jeffreys <- MPL_Jeffreys
} else if (length(fix) == length(start)) {
    ret <- list(par = start,
                value = fn(start),
                MPL_Jeffreys = FALSE)
} else {
    ret <- .profile(start, fix = fix)
}
◇
```

Fragment referenced in 43.

After parameter estimation, we evaluate negative scores, the Hessian, and negative residuals as requested:

⟨ *post processing* 40 ⟩ ≡

```
if (is.null(fix) || (length(fix) == length(start)))
    parm <- seq_len(K - 1)
else
    parm <- fix
if (any(parm >= K)) return(ret)

ret$coefficients <- ret$par[parm]
dn2 <- dimnames(xt)[2L]
names(ret$coefficients) <- cnames <- paste0(names(dn2), dn2[[1L]][1L + parm])

par <- ret$par
intercepts <- function(parm, x)
{
    ⟨ stratum prep 15 ⟩
    return(intercepts)
}
ret$intercepts <- intercepts(par, x = xlist)

if (score) {
    ret$negscore <- .snsc(par, x = xlist, mu = mu)[parm]
    if (!is.null(xrc))
        ret$negscore <- ret$negscore + .snsc(par, x = xrclist, mu = mu,
                                             rightcensored = TRUE)[parm]
}
if (hessian) {
    if (!is.null(xrc)) {
        ret$hessian <- .shes(par, x = xlist, mu = mu, xrc = xrclist)
    } else {
        ret$hessian <- .shes(par, x = xlist, mu = mu)
    }
    ret$vcov <- solve(ret$hessian)
    if (length(parm) != nrow(ret$hessian))
       ret$hessian <- solve(ret$vcov <- ret$vcov[parm, parm, drop = FALSE])
    rownames(ret$vcov) <- colnames(ret$vcov) <- rownames(ret$hessian) <-
        colnames(ret$hessian) <-  cnames
}
if (residuals) {
    ret$negresiduals <- .snsr(par, x = xlist, mu = mu)
    if (!is.null(xrc)) {
        rcr <- .snsr(par, x = xrclist, mu = mu, rightcensored = TRUE)
        ret$negresiduals <- c(rbind(matrix(ret$negresiduals, nrow = C),
                                    matrix(rcr, nrow = C)))
    }
    ret$negscaleresiduals <- .snssr(par, x = xlist, mu = mu)
    if (!is.null(xrc)) {
        rcr <- .snssr(par, x = xrclist, mu = mu, rightcensored = TRUE)
        ret$negscaleresiduals <-
            c(rbind(matrix(ret$negscaleresiduals, nrow = C),
                    matrix(rcr, nrow = C)))
    }
}

if (logdet && hessian) ret$logdet <- ld(he(par))
◇
```

Fragment referenced in 43.

We add a function allowing us to evaluate the profile log-likelihood

⟨ *profile function* 41 ⟩ ≡

```
ret$profile <- function(start, fix)
    .free1wayML(xt, link = link, mu = mu, start = start, fix = fix,
                residuals = residuals, score = score, hessian = hessian,
                logdet = logdet, MPL_Jeffreys = MPL_Jeffreys,
                dooptim = dooptim, control = control, trace = trace, tol = tol,
                ...)
ret$table <- xt

ret$strata <- strata
ret$mu <- mu
if (length(ret$mu) == 1) {
    names(ret$mu) <- link$parm
} else {
    names(ret$mu) <- c(paste(link$parm, cnames[1L], sep = ":"), cnames[-1L])
}
```

◇

Fragment referenced in 43.

and, finally, we put everything into one function which returns an object of class `free1wayML` for later use. The control parameters for `.NewtonRaphson` and `stats::nlminb` are the ones suggested by Harrell (2024). By default, the internal Newton-Raphson implementation is used, we can switch to `stats::nlminb` by specifying `dooptim = "nlminb"`. The latter option cannot handle Fisher information matrices in form of a `Matrix` object and thus computing the updates takes more time whenever a larger number of intercept parameters in present in the problem.

The first step is now to make sure the main argument `x` is a tree-way table

⟨ *convert to three-way table* 42 ⟩ ≡

```
xt <- x
if (!is.table(x))
     stop(gettextf("invalid argument '%s'", "x"), domain = NA) # 'y' in free1way ...
dx <- dim(x)
dn <- dimnames(x)
if (length(dx) == 2L) {
    x <- as.table(array(c(x), dim = dx <- c(dx, 1L)))
    dimnames(x) <- dn <- c(dn, list(A = "A"))
}
```

◇

Fragment referenced in 43.

and we are now ready to setup the main workhorse

⟨ *ML estimation* 43 ⟩ ≡

```
.free1wayML <- function(x, link, mu = 0, start = NULL, fix = NULL,
                        residuals = TRUE, score = TRUE, hessian = TRUE,
                        logdet = FALSE, MPL_Jeffreys = FALSE,
                        ### use nlminb for small sample sizes or MPL
                        dooptim = c(".NewtonRaphson", "nlminb")[1 +
                            ((sum(x) < 20) || (MPL_Jeffreys > 0))],
                        control = list(
                            "nlminb" = list(trace = trace, iter.max = 200,
                                            eval.max = 200, rel.tol = 1e-10,
                                            abs.tol = 1e-20, xf.tol = 1e-16),
                            ".NewtonRaphson" = list(iter.max = 200, trace = trace,
                                             objtol = 5e-4,
                                             gradtol = 1e-5 * sum(x) / 1000,
                                             paramtol = 1e-5, minstepsize = 1e-2,
                                             tolsolve = .Machine$double.eps)
                        )[dooptim],
                        trace = FALSE,
                        tol = sqrt(.Machine$double.eps), ...)
{

    ⟨ convert to three-way table 42 ⟩
    ⟨ short-cuts for link functions 1 ⟩

    if(!suppressPackageStartupMessages(requireNamespace("Matrix")))
        stop(gettextf("%s needs package 'Matrix' correctly installed",
                      ".free1wayML"),
             domain = NA)

    if (length(MPL_Jeffreys) > 1L || any(MPL_Jeffreys < 0L))
        stop(gettextf("invalid argument '%s'", "MPL_Jeffreys"), domain = NA)

    ⟨ setup and starting values 34 ⟩
    ⟨ negative logLik 3 ⟩
    ⟨ negative score 5 ⟩
    ⟨ negative score residuals 6 ⟩
    ⟨ negative scale score residuals 7 ⟩
    ⟨ Hessian 12 ⟩
    ⟨ stratified negative logLik 16 ⟩
    ⟨ stratified negative score 17 ⟩
    ⟨ stratified Hessian 21 ⟩
    ⟨ stratified negative score residual 18 ⟩
    ⟨ stratified negative scale score residual 19 ⟩
    ⟨ logdet 38 ⟩
    ⟨ profile 37 ⟩
    ⟨ optim 39 ⟩
    ⟨ post processing 40 ⟩
    ⟨ profile function 41 ⟩

    if (MPL_Jeffreys > 0L)
        class(ret) <- c("MPLfree1wayML", "free1wayML")
    else
        class(ret) <- "free1wayML"
    return(ret)
}
```

◇

Fragment referenced in 33.

As an example, consider a stratified (two stata) 3×3 problem where outcome category B is missing from the second stratum:

```
R> N <- 10
R> a <- matrix(c(5, 6, 4,
+                      3, 5, 7,
+                      3, 4, 5,
+                      3, 5, 6,
+                      0, 0, 0,
+                      4, 6, 5), ncol = 3, byrow = TRUE)
R> x <- as.table(array(c(a[1:3,], a[-(1:3),]), dim = c(3, 3, 2)))
R> x

, , A

  A B C
A 5 6 4
B 3 5 7
C 3 4 5

, , B

  A B C
A 3 5 6
B 0 0 0
C 4 6 5

R> ret <- .free1wayML(x, logit())
R> ret[c("value", "par")]

$value
[1] 65.932

$par
[1]  0.052627  0.204750 -0.489670  1.020466  0.028488

R> cf <- ret$par
R> cf[1:2] <- cf[1:2] + .5
R> ### new2old parameterisation
R> c(cf[1:2], cf[3], log(cf[4] - cf[3]), cf[5])

[1]  0.552627  0.704750 -0.489670  0.412200  0.028488

R> ### profile for cf[1:2]
R> .free1wayML(x, logit(), start = cf, fix = 1:2)[c("value", "par")]

$value
[1] 66.381

$par
[1]  0.55263  0.70475 -0.11885  1.40796  0.40841

R> ### profile for cf[2]
R> .free1wayML(x, logit(), start = cf, fix = 2)[c("value", "par")]

$value
[1] 66.299

$par
[1]  0.36095  0.70475 -0.18771  1.34153  0.33443

R> ### evaluate log-likelihood at cf
R> .free1wayML(x, logit(), start = cf,
+              fix = seq_along(ret$par))[c("value", "par")]
```

```
$value
[1] 67.771

$par
[1]  0.552627  0.704750 -0.489670  1.020466  0.028488
```

Chapter 6

ML Inference

Based on an object of class `free1wayML`, we can setup different test statistics and obtain the limiting null distribution based on classical ML theory under the population model:

⟨ *statistics* 44 ⟩ ≡

```
if (test == "Wald") {
    ⟨ Wald statistic 45 ⟩
} else if (test == "LRT") {
    ⟨ LRT 46 ⟩
} else if (test == "Rao") {
    ⟨ Rao 47 ⟩
} else if (test == "Permutation") {
    ⟨ Permutation statistics 48 ⟩
}
```

◇

Fragment referenced in 62, 68.

6.1 Wald Statistics

We only need access to the parameter estimates $\hat{\delta}_2, \ldots, \hat{\delta}_K$ and the corresponding Hessian:

⟨ *Wald statistic* 45 ⟩ ≡

```
if (alternative == "two.sided") {
    STATISTIC <- c("Wald chi-squared" =
                   c(crossprod(cf, x$hessian %*% cf)))
    DF <- c("df" = length(parm))
    PVAL <- pchisq(STATISTIC, df = DF, lower.tail = FALSE)
} else {
    STATISTIC <- c("Wald Z" = unname(c(cf * sqrt(c(x$hessian)))))
    PVAL <- pnorm(STATISTIC, lower.tail = alternative == "less")
}
```

◇

Fragment referenced in 44.

6.2 Likelihood-ratio Statistics

In addition to the log-likelihood evaluated at the ML estimates, we need to evaluate the profile log-likelihood at some value corresponding the null hypothesis to be tested:

⟨ *LRT* 46 ⟩ ≡

```
par <- x$par
par[parm] <- value
unll <- x$value ### neg logLik
rnll <- x$profile(par, parm)$value ### neg logLik
STATISTIC <- c("logLR chi-squared" = - 2 * (unll - rnll))
DF <- c("df" = length(parm))
PVAL <- pchisq(STATISTIC, df = DF, lower.tail = FALSE)
◇
```

Fragment referenced in 44.

6.3 Rao Score Statistics

For the Rao score test, the inverse of the Hessian as well as the score function of the shift parameters evaluated for some null values need to be computed:

⟨ *Rao* 47 ⟩ ≡

```
par <- x$par
par[parm] <- value
ret <- x$profile(par, parm)
if (alternative == "two.sided") {
    STATISTIC <- c("Rao chi-squared" = c(crossprod(ret$negscore,
                                                   ret$vcov %*%
                                                   ret$negscore)))
    DF <- c("df" = length(parm))
    PVAL <- pchisq(STATISTIC, df = DF, lower.tail = FALSE)
} else {
    STATISTIC <- c("Rao Z" = unname(- ret$negscore *
                                      sqrt(c(ret$vcov))))
    PVAL <- pnorm(STATISTIC, lower.tail = alternative == "less")
}
◇
```

Fragment referenced in 44.

Chapter 7

Permutation Inference

Under the permutation model, that is, in randomised experiments where the random treatment allocation is the only relevant source of randomness, we compute a permutation variant of the Rao score test, based on the conditional asymptotic distribution or based on a Monte-Carlo estimate of the reference distribution:

⟨ *Permutation statistics* 48 ⟩ ≡

```
par <- x$par
par[parm] <- value
ret <- x$profile(par, parm)
sc <- - ret$negscore
if (length(cf) == 1L)
    sc <- sc / sqrt(c(ret$hessian))
if (!is.null(x$exact)) {
    STATISTIC = c("W" = sc)
} else {
    Esc <- sc - x$perm$Expectation

    if (alternative == "two.sided" && length(cf) > 1L) {
        STATISTIC <- c("Perm chi-squared" =
                       sum(Esc * solve(x$perm$Covariance, Esc)))
    } else {
        STATISTIC <- c("Perm Z" = Esc / sqrt(c(x$perm$Covariance)))
    }
}
```

◇

Fragment referenced in 44.

In addition, we compute permutation p-values

⟨ *Permutation p-values* 49 ⟩ ≡

```
if (!is.null(x$exact)) {
    PVAL <- switch(alternative,
                   "two.sided" = pmin(1, 2 * min(c(x$exact$ple(sc),
                                                   x$exact$pgr(sc)))),
                   "less" = x$exact$ple(sc),
                   "greater" = x$exact$pgr(sc))
} else {
    .pm <- function(x) sum(x) / length(x)
    ps <- x$perm$permStat

    .GE <- function(x, y)
        (y - x) <= sqrt(.Machine$double.eps)

    .LE <- function(x, y)
        (x - y) <= sqrt(.Machine$double.eps)

    if (alternative == "two.sided" && length(cf) > 1L) {
        if (!is.null(ps)) {
            PVAL <- .pm(.GE(ps, STATISTIC))
        } else {
            DF <- c("df" = x$perm$DF)
            PVAL <- pchisq(STATISTIC, df = DF, lower.tail = FALSE)
        }
    } else {
        if (!is.null(ps)) {
            PVALle <- .pm(.LE(ps, STATISTIC))
            PVALge <- .pm(.GE(ps, STATISTIC))
            if (alternative == "two.sided")
                PVAL <- 2 * min(c(PVALle, PVALge))
            else if (alternative == "less")
                PVAL <- PVALle
            else
                PVAL <- PVALge
        } else {
            if (alternative == "two.sided")
                PVAL <- pchisq(STATISTIC^2, df = 1, lower.tail = FALSE)
            else
                PVAL <- pnorm(STATISTIC,
                              lower.tail = alternative == "less")
        }
    }
}
```

◇

Fragment referenced in 62, 70.

The mean and variance of the linear permutation statistic under the null was given by Strasser and Weber (1999):

⟨ *Strasser Weber* 50 ⟩ ≡

```
.SW <- function(res, xt)
{

    if (length(dim(xt)) == 3L) {
        res <- matrix(res, nrow = dim(xt)[1L], ncol = dim(xt)[3])
        STAT <-  Exp <- Cov <- 0
        for (b in seq_len(dim(xt)[3L])) {
            sw <- .SW(res[,b, drop = TRUE], xt[,,b, drop = TRUE])
            STAT <- STAT + sw$Statistic
            Exp <- Exp + sw$Expectation
            Cov <- Cov + sw$Covariance
        }
        return(list(Statistic = STAT, Expectation = as.vector(Exp),
                    Covariance = Cov))
    }

    Y <- matrix(res, ncol = 1, nrow = length(xt))
    weights <- c(xt)
    x <- gl(ncol(xt), nrow(xt))
    X <- model.matrix(~ x, data = data.frame(x = x))[,-1L,drop = FALSE]

    w. <- sum(weights)
    wX <- weights * X
    wY <- weights * Y
    ExpX <- colSums(wX)
    ExpY <- colSums(wY) / w.
    CovX <- crossprod(X, wX)
    Yc <- t(t(Y) - ExpY)
    CovY <- crossprod(Yc, weights * Yc) / w.
    Exp <- kronecker(ExpY, ExpX)
    Cov <- w. / (w. - 1) * kronecker(CovY, CovX) -
           1 / (w. - 1) * kronecker(CovY, tcrossprod(ExpX))
    STAT <- crossprod(X, wY)
    list(Statistic = STAT, Expectation = as.vector(Exp),
         Covariance = Cov)
}
```

◇

Fragment referenced in 55, 70.

For small samples, we used the `r2dtable` function to sample from tables with fixed marginal distributions:

⟨ *resampling* 51 ⟩ ≡

```
.resample <- function(res, xt, B = 10000)
{

    if (length(dim(xt)) == 2L)
        xt <- as.table(array(xt, dim = c(dim(xt), 1)))

    res <- matrix(res, nrow = dim(xt)[1L], ncol = dim(xt)[3L])
    stat <- 0
    ret <- .SW(res, xt)
    if (dim(xt)[2L] == 2L) {
        ret$testStat <- c((ret$Statistic - ret$Expectation) /
                          sqrt(c(ret$Covariance)))
    } else {
        ES <- ret$Statistic - ret$Expectation
        ret$testStat <- sum(ES * solve(ret$Covariance, ES))
    }
    ret$DF <- dim(xt)[2L] - 1L

    if (B) {
        for (j in 1:dim(xt)[3L]) {
           rt <- r2dtable(B, r = rowSums(xt[,,j]), c = colSums(xt[,,j]))
           stat <- stat + vapply(rt,
               function(x) .colSums(x[,-1L, drop = FALSE] * res[,j],
                                    m = nrow(x), n = ncol(x) - 1L),
                                  FUN.VALUE = rep(0, dim(xt)[[2L]] - 1L))
        }
        if (dim(xt)[2L] == 2L) {
             ret$permStat <- (stat - ret$Expectation) /
                             sqrt(c(ret$Covariance))
        } else {
            ES <- matrix(stat, ncol = B) - ret$Expectation
            ret$permStat <- .colSums(ES * solve(ret$Covariance, ES),
                                     m = dim(xt)[[2L]] - 1L, n = B)
        }
    }
    ret
}
```

◇

Fragment referenced in 55, 70.

For the special case of the unstratified Wilcoxon two-sample test, we can also provide exact p-values computed via the Streitberg-Röhmel shift algorithm, mainly because the scores can be mapped to integers:

⟨ *exact proportional odds* 52 ⟩ ≡

```
.exact <- function(z, grp, w = rep.int(1, length(z)))
{

    z <- rep(z, times = w)
    grp <- rep(grp, times = w)
    x <- rank(z)
    f <- 2 - all(x == floor(x))
    x <- as.integer(x * f)
    x <- x - min(x) + 1L
    sx <- sort(x)

    m <- as.integer(sum(grp > 0))
    stopifnot(m > 1)
    stopifnot(m < length(x))

    d <- .Call(C_dpermdist2, sx, m)
    s <- seq.int(from = 1L, to = sum(rev(sx)[seq_len(m)]), by = 1L)

    STATISTIC <- sum(x[grp > 0])
    F <- cumsum(d)
    S <- rev(cumsum(rev(d)))
    cf <- lm.fit(x = cbind(1, x), y = as.double(z))$coefficients

    z2x <- function(z) round((z - m * cf[1]) / cf[2])

    c(ple = function(z) sum(d[s <= z2x(z)]), # s and STATISTIC are integers
      pgr = function(z) sum(d[s >= z2x(z)]),
      qle = function(q) c(m, max(s[F < q + 1e-08])) %*% cf,
      qgr = function(q) c(m, min(s[S < q + 1e-08])) %*% cf)
}
◇
```

Fragment referenced in 55.

As an example, consider the Wilcoxon rank sum test, where the scores under the null are a linear function of the ranks of the data. We compute the asymptotic and approximated reference distribution and corresponding p-values for a test statistics in quadratic form:

```
R> set.seed(29)
R> w <- gl(2, 15)
R> (s <- .SW(r <- rank(u <- runif(length(w))), model.matrix(~ 0 + w)))

$Statistic
   [,1]
x2  287

$Expectation
[1] 232.5

$Covariance
       [,1]
[1,] 581.25

R> ps <- .resample(r, model.matrix(~ 0 + w), B = 100000)
R> ps$testStat^2

[1] 5.1101

R> mean(abs(ps$permStat) > abs(ps$testStat) - .Machine$double.eps)

[1] 0.02435
```

```
R> pchisq(ps$testStat^ifelse(ps$DF == 1, 2, 1), df = ps$DF, lower.tail = FALSE)

[1] 0.023787

R> ### exactly the same
R> kruskal.test(u ~ w)

        Kruskal-Wallis rank sum test

data:  u by w
Kruskal-Wallis chi-squared = 5.11, df = 1, p-value = 0.024

R> library("coin")
R> ### almost the same
R> kruskal_test(u ~ w, distribution = approximate(100000))

        Approximative Kruskal-Wallis Test

data:  u by w (1, 2)
chi-squared = 5.11, p-value = 0.024
```

and the exact versions are

```
R> wilcox_test(u ~ w, distribution = "exact")

        Exact Wilcoxon-Mann-Whitney Test

data:  u by w (1, 2)
Z = -2.26, p-value = 0.023
alternative hypothesis: true mu is not equal to 0

R> free1way(u ~ w, exact = TRUE)

        2-sample Wilcoxon test against proportional odds alternatives

data:  u by w (1, 2)
W = 2.38, p-value = 0.023
alternative hypothesis: true log-odds ratio is not equal to 0

R> wilcox_test(u ~ w, distribution = "exact", alternative = "less")

        Exact Wilcoxon-Mann-Whitney Test

data:  u by w (1, 2)
Z = -2.26, p-value = 0.012
alternative hypothesis: true mu is less than 0

R> print(free1way(u ~ w, exact = TRUE), alternative = "greater")

        2-sample Wilcoxon test against proportional odds alternatives

data:  u by w (1, 2)
W = 2.38, p-value = 0.012
alternative hypothesis: true log-odds ratio is greater than 0

R> wilcox_test(u ~ w, distribution = "exact", alternative = "greater")

        Exact Wilcoxon-Mann-Whitney Test

data:  u by w (1, 2)
Z = -2.26, p-value = 0.99
alternative hypothesis: true mu is greater than 0
```

```
R> print(free1way(u ~ w, exact = TRUE), alternative = "less")

        2-sample Wilcoxon test against proportional odds alternatives

data:  u by w (1, 2)
W = 2.38, p-value = 0.99
alternative hypothesis: true log-odds ratio is less than 0
```

<TH>Ordered alternatives: Use contrast based tests in multcomp</TH>

Chapter 8

Distribution-free Tests in Stratified K-sample Oneway Layouts

8.1 `free1way`

We provide a new test procedure in a generic `free1way`, featuring a method for tables (the main workhorse) and additional user interfaces.

⟨ *link2fun* 53 ⟩ ≡

```
if (!inherits(link, "linkfun")) {
    link <- match.arg(link)
    link <- do.call(link, list())
}
```

◇

Fragment referenced in 54, 82, 83, 85.

We use the positive residuals for defining a permutation test with treatment effect coding using the first group as control, that is, the test statistic is defined through the sum of the positive residuals in all but the control group. Unfortunately, most `stats::*.test` procedures use the second group as control, so factors need to be releveled to obtain identical results (this is relevant for the one-sided case).

⟨ *free1way generic and table method (main workhorse)* 54 ⟩ ≡

```
free1way <- function(y, ...)
    UseMethod("free1way")

free1way.table <- function(y, link = c("logit", "probit", "cloglog", "loglog"),
                           mu = 0, B = 0, exact = FALSE, MPL_Jeffreys = FALSE, ...)
{

    cl <- match.call()

    d <- dim(y)
    dn <- dimnames(y)
    DNAME <- NULL
    if (!is.null(dn)) {
        DNAME <- paste(names(dn)[1], "by", names(dn)[2],
                       paste0("(", paste0(dn[2], collapse = ", "), ")"))
        if (length(dn) == 3L)
            DNAME <- paste(DNAME, "\n\t stratified by", names(dn)[3])
    }

    ⟨ link2fun 53 ⟩

    if (!(length(mu) == 1L || length(mu) == d[2L] - 1L)) {
        warning(gettextf("incompatible length of argument 'mu' in %s",
                         "free1way"),
                domain = NA)
        mu <- rep(mu, length.out = d[2L] - 1L)
    }

    ret <- .free1wayML(y, link = link, mu = mu, MPL_Jeffreys = MPL_Jeffreys,
                       ...)
    ret$link <- link
    ret$data.name <- DNAME
    ret$call <- cl

    ⟨ free1way permutation tests 55 ⟩

    if (ret$MPL_Jeffreys > 0L) {
        ret$method <- paste(ret$method,
            "with Jeffreys prior penalisation", sep = ", ")
        class(ret) <- c("MPLfree1way", "free1way")
    } else {
        class(ret) <- "free1way"
    }
    return(ret)
}
◇
```

Fragment referenced in 33.

where preparations for permutations tests are performed before returning the object

⟨ *free1way permutation tests* 55 ⟩ ≡

```
alias <- link$alias
if (length(link$alias) == 2L) alias <- alias[1L + (d[2] > 2L)]
stratified <- FALSE
if (length(d) == 3L) stratified <- d[3L] > 1
ret$method <- paste(ifelse(stratified, "Stratified", ""),
                    paste0(d[2L], "-sample"), alias,
                    "test against", link$model, "alternatives")

cf <- ret$par
### compute the permutation distribution always
### for H0: delta = 0, not delta = mu
### otherwise, permutation confidence intervals
### are not aligned with permutation p-values
cf[idx <- seq_len(d[2L] - 1L)] <- -mu
pr <- ret$profile(cf, idx)
res <- - pr$negresiduals
if (d[2L] == 2L)
    res <- res / sqrt(c(pr$hessian))

⟨ Strasser Weber 50 ⟩
⟨ resampling 51 ⟩

if (length(dim(y)) == 3L) y <- y[,,ret$strata, drop = FALSE]
if (length(dim(y)) == 4L) {
    y <- y[,,ret$strata,, drop = FALSE]
    dy <- dim(y)
    dy[1] <- dy[1] * 2
    y <- apply(y, 3, function(x) rbind(x[,,"TRUE"], x[,,"FALSE"]), simplify = FALSE)
    y <- array(unlist(y), dim = dy[1:3])
}

### exact two-sample Wilcoxon w/o stratification
if (exact) {
    if (!stratified && link$model == "proportional odds" && d[2L] == 2L) {
        ⟨ exact proportional odds 52 ⟩
        ret$exact <- .exact(c(res, res), grp = unclass(gl(2, d[1L])) - 1L,
                            w = c(y))
        B <- 0
    } else {
        warning(gettextf("cannot compute exact permutation distribution in %s",
                         "free1way"),
                domain = NA)
    }
}
ret$perm <- .resample(res, y, B = B)
◇
```

Fragment referenced in 54.

The `formula` method allows formulae

R> y ~ groups | blocks

for model specification. We start handling the formula

⟨ *formula business* 56 ⟩ ≡

```
if(missing(formula) || (length(formula) != 3L))
    stop("'formula' missing or incorrect")

if (stratum <- (length(formula[[3L]]) > 1)) {
  if ((length(formula[[3L]]) != 3L) ||
      (formula[[3L]][[1L]] != as.name("|")) ||
      (length(formula[[3L]][[2L]]) !=  1L) ||
      (length(formula[[3L]][[3L]]) != 1L))
      stop(gettextf("incorrect specification for '%s'", "formula"),
           domain = NA)
  formula[[3L]][[1L]] <- as.name("+")
}

formula <- terms(formula)
if (length(attr(formula, "term.labels")) > 1L + stratum)
   stop("'formula' missing or incorrect")
group <- attr(formula, "term.labels")[1L]

m <- match.call(expand.dots = FALSE)
m$formula <- formula
if (is.matrix(eval(m$data, parent.frame())))
    m$data <- as.data.frame(data)
## need stats:: for non-standard evaluation
m[[1L]] <- quote(stats::model.frame)
m$... <- NULL
mf <- eval(m, parent.frame())
```

◇

Fragment referenced in 57.

⟨ *free1way formula* 57 ⟩ ≡

```
free1way.formula <- function(formula, data, weights, subset, na.action = na.pass,
                             event = NULL, ...)
{
    cl <- match.call()

    ⟨ formula business 56 ⟩

    response <- attr(attr(mf, "terms"), "response")
    DNAME <- paste(vn <- c(names(mf)[response], group),
                   collapse = " by ") # works in all cases
    w <- as.vector(model.weights(mf))
    y <- mf[[response]]
    if (inherits(y, "Surv")) {
        if (!is.null(event))
            stop(gettextf("cannot have both a 'Surv()' response and an 'event' argument in %s",
                          "free1way"),
                 domain = NA)
        if (attr(y, "type") != "right")
            stop(gettextf("%s currently only allows independent right-censoring",
                          "free1way"),
                 domain = NA)
        event <- (y[,2] > 0)
        y <- y[,1]
    }
    g <- factor(mf[[group]])
    mf[[group]] <- g
    lev <- levels(g)
    DNAME <- paste(DNAME, paste0("(", paste0(lev, collapse = ", "), ")"))
    if (nlevels(g) < 2L)
        stop(gettextf("incorrect argument 'groups' in %s: at least two groups needed",
                      "free1way"),
             domain = NA)
    if (stratum) {
        st <- factor(mf[[3L]])
        mf[[3L]] <- st
        ### nlevels(st) == 1L is explicitly allowed
        vn <- c(vn, names(mf)[3L])
        RVAL <- free1way(y = y, groups = g, blocks = st, event = event,
                         weights = w, varnames = vn, ...)
        DNAME <- paste(DNAME, paste("\n\t stratified by", names(mf)[3L]))
    } else {
        ## Call the corresponding method
        RVAL <- free1way(y = y, groups = g, event = event, weights = w,
                         varnames = vn, ...)
    }
    RVAL$data <- mf
    RVAL$data.name <- DNAME
    RVAL$call <- cl
    RVAL
}
◇
```

Fragment referenced in 33.

The method for numeric outcomes provides a discretisation at the unique observed outcome values, or (for very large sample sizes), for binned outcomes. The `event` argument is a logical where `TRUE` is interpreted as an event and `FALSE` as right-censored observation

⟨ *variable names and checks* 58 ⟩ ≡

```
cl <- match.call()
if (is.null(varnames))
    varnames <- c(deparse1(substitute(y)),
                  deparse1(substitute(groups)),
                  deparse1(substitute(blocks)))

DNAME <- paste(varnames[1], "by", varnames[2])
groups <- factor(groups)
if (nlevels(groups) < 2L)
    stop(gettextf("incorrect argument 'groups' in %s: at least two groups needed",
                  "free1way"),
         domain = NA)
DNAME <- paste(DNAME, paste0("(", paste0(levels(groups), collapse = ", "),
                             ")"))

if (!is.null(blocks)) {
    if (length(unique(blocks)) < 2L) {
        blocks <- NULL
    } else {
        blocks <- factor(blocks)
        DNAME <- paste(DNAME, "\n\t stratified by", varnames[3])
    }
}
varnames <- varnames[varnames != "NULL"]
◇
```

Fragment referenced in 59, 60.

Note that the return value of `unique` might differ between platforms. Because users can decide about the unique values in the vector `y` (by using `round` or `trunc`, for example), before calling this function, we refrain from handling this issue internally. However, we offer an `nbins` argument for binning response observations at sample quantiles in the absence of right-censoring. Note that we ignore the blocks when calling `cut`. This is inefficient for many blocks with non-overlapping support of the outcome distribtion, as large sparse tables are resulting. We remove the corresponding elements from the first dimension of such a table later on (in `.free1wayML`). The reason for this inconvenice is that all the data going into `.free1wayML` can be stored as a `table` (and not as a list of things).

⟨ *free1way numeric* 59 ⟩ ≡

```
free1way.numeric <- function(y, groups, blocks = NULL, event = NULL,
                             weights = NULL, nbins = 0, varnames = NULL, ...)
{

    ⟨ variable names and checks 58 ⟩

    if (!is.null(event)) {
        if (!is.logical(event))
            stop(gettextf("%s currently only allows independent right-censoring",
                          "free1way"),
                 domain = NA)
        uy <- sort(unique(y[event]))
    } else {
        uy <- sort(unique(y))
    }
    if (nbins && nbins < length(uy) && is.null(event)) {
        nbins <- ceiling(nbins)
        breaks <- c(-Inf, quantile(y, probs = seq_len(nbins) / (nbins + 1L)),
                    Inf)
    } else {
        breaks <- c(-Inf, uy, Inf)
    }
    r <- ordered(cut(y, breaks = breaks, ordered_result = TRUE,
                     labels = FALSE)) ### avoids costly formatC call
    RVAL <- free1way(y = r, groups = groups, blocks = blocks,
                     event = event, weights = weights,
                     varnames = varnames, ...)
    RVAL$data.name <- DNAME
    RVAL$call <- cl
    RVAL
}
```

◇

Fragment referenced in 33.

The `factor` method also allows right-censoring but otherwise is just a call to `xtabs`:

⟨ *free1way factor* 60 ⟩ ≡

```
free1way.factor <- function(y, groups, blocks = NULL, event = NULL,
                            weights = NULL, varnames = NULL, ...)
{

    ⟨ variable names and checks 58 ⟩

    if (nlevels(y) > 2L && !is.ordered(y))
        stop(gettextf("%s is not defined for unordered responses",
                              "free1way"),
             domain = NA)
    d <- data.frame(w = 1, y = y, groups = groups)
    if (!is.null(weights)) d$w <- weights
    if (is.null(blocks)) blocks <- gl(1, nrow(d))
    d$blocks <- blocks
    if (!is.null(event)) {
       if (!is.logical(event))
            stop(gettextf("%s currently only allows independent right-censoring",
                          "free1way"),
                 domain = NA)
        d$event <- factor(event, levels = c(FALSE, TRUE),
                          labels = c("FALSE", "TRUE"))
    }
    tab <- xtabs(w ~ ., data = d)
    dn <- dimnames(tab)
    names(dn)[seq_along(varnames)] <- varnames
    dimnames(tab) <- dn
    RVAL <- free1way(tab, ...)
    RVAL$data.name <- DNAME
    RVAL$call <- cl
    RVAL
}
```

◇

Fragment referenced in 33.

8.2 `free1way` Methods

We start with `coef`, `vcov`, and `model.frame`/`model.matrix` methods such that multiple comparison procedures from **multcomp** will work out of the box. The `coef` method allows to obtain effects at alternative scales: probabilistic indices (`AUC` = `PI`) or the overlap coefficient:

⟨ *free1way methods* 61 ⟩ ≡

```
coef.free1way <- function(object, what = c("shift", "PI", "AUC", "OVL"), ...)
{
    what <- match.arg(what)
    cf <- object$coefficients
    return(switch(what, "shift" = cf,
                        "PI" = object$link$parm2PI(cf),
                        "AUC" = object$link$parm2PI(cf),        ### same as PI
                        "OVL" = object$link$parm2OVL(cf)))
}
vcov.free1way <- function(object, ...)
    object$vcov
logLik.free1way <- function(object, ...)
    -object$value
model.frame.free1way <- function(formula, ...) {
    if (!is.null(formula[["data"]])) return(formula[["data"]])
    ret <- as.data.frame(formula$table)
    ret <- ret[rep(seq_len(nrow(ret)), ret$Freq),,drop = FALSE]
    ret
}
### the next two might go into multcomp
terms.free1way <- function(x, ...)
{
    mf <- model.frame(x)
    terms(as.formula(paste(names(mf)[1:2], collapse = "~")),
          data = mf)
}
model.matrix.free1way <- function (object, ...)
{
    mf <- model.frame(object)
    tm <- terms(object)
    mm <- model.matrix(delete.response(tm), data = mf)
    at <- attributes(mm)
    mm <- mm[, -1]
    at$dim[2] <- at$dim[2] - 1
    at$dimnames[[2]] <- at$dimnames[[2]][-1]
    at$assign <- at$assign[-1]
    attributes(mm) <- at
    mm
}
```

◇

Fragment referenced in 33.

We use the `print` method to report different test statistics and corresponding p-values via the `test` and `alternative` arguments. The reason for doing so is that the parameter estimation only needs to be performed once in cases users are interested in different tests or (see below) confidence intervals. By default, an asymptotic permutation test is performed, mainly because the p-values coincide with some special cases (`wilcox,kruskal,friedman.test`):

⟨ *free1way print* 62 ⟩ ≡

```
.print.free1way <- function(x, test = c("Permutation", "Wald", "LRT", "Rao"),
                            alternative = c("two.sided", "less", "greater"),
                            tol = sqrt(.Machine$double.eps),
                            mu = 0, ### allow permutation testing non-null hypotheses
                                    ### in alignment with confint(free1way(, B > 0))
                            ...)
{

    test <- match.arg(test)
    alternative <- match.arg(alternative)

    ### global
    cf <- coef(x)
    if ((length(cf) > 1L || test == "LRT") && alternative != "two.sided")
        stop(gettextf("cannot compute one-sided p-values in %s",
                      "free1way"),
             domain = NA)

    DF <- NULL
    parm <- seq_along(cf)
    value <- mu

    ⟨ statistics 44 ⟩

    if (test == "Permutation") {
        ⟨ Permutation p-values 49 ⟩
    }

    RVAL <- list(statistic = STATISTIC, parameter = DF, p.value = PVAL,
        null.value = x$mu, alternative = alternative, method = x$method,
        data.name = x$data.name)
    class(RVAL) <- "htest"
    return(RVAL)
}

print.free1way <- function(x, ...)
{
    print(ret <- .print.free1way(x, ...))
    return(invisible(x))
}
◇
```

Fragment referenced in 33.

The `summary` method performs population Wald inference unless the `test` argument is specified:

⟨ *free1way summary* 63 ⟩ ≡

```
summary.free1way <- function(object, test,
                             alternative = c("two.sided", "less", "greater"),
                             tol = .Machine$double.eps, ...)
{

    if (!missing(test))
        return(.print.free1way(object, test = test,
                               alternative = alternative, tol = tol, ...))

    alternative <- match.arg(alternative)

    ESTIMATE <- coef(object)
    SE <- sqrt(diag(vcov(object)))
    STATISTIC <- unname(ESTIMATE / SE)
    if (alternative == "less") {
        PVAL <- pnorm(STATISTIC)
    } else if (alternative == "greater") {
        PVAL <- pnorm(STATISTIC, lower.tail = FALSE)
    } else {
        PVAL <- 2 * pnorm(-abs(STATISTIC))
    }
    cfmat <- cbind(ESTIMATE, SE, STATISTIC, PVAL)
    colnames(cfmat) <- c(object$link$parm, "Std. Error", "z value",
                         switch(alternative, "two.sided" = "Pr(>|z|)",
                                             "less" = "Pr(<z)",
                                             "greater" = "Pr(>z)"))
    ret <- list(call = object$call, coefficients = cfmat)
    class(ret) <- "summary.free1way"
    return(ret)
}

print.summary.free1way <- function(x, ...)
{
    cat("\nCall:\n", paste(deparse(x$call), sep = "\n", collapse = "\n"),
        "\n\n", sep = "")
    cat("Coefficients:\n")
    printCoefmat(x$coefficients)
}
◇
```

Fragment referenced in 33.

Confidence intervals are computed by inversion of the corresponding test statistics. Because LRT and Rao confidence intervals are invariant with respect to transformations, proper LRT or Rao confidence intervals for probabilistic indices or overlap coefficients can also be computed.

We begin computing the critical values for permutation tests, making sure the confidence intervals will be in line with one- and two-sided p-values:

⟨ *permutation confint* 64 ⟩ ≡

```
if (length(cf) > 1L)
    stop(gettextf("permutation confidence intervals only available for 2-sample comparisons in %s",
                  "confint.free1way"),
         domain = NA)
if (!is.null(object$exact)) {
    qu <- c(object$exact$qle(1 - conf.level),
            object$exact$qgr(1 - conf.level))
} else {
    if (is.null(object$perm$permStat)) {
        qu <- qnorm(conf.level) * c(-1, 1)
    } else {
        .pq <- function(s, alpha)
        {
            su <- sort(unique(s))
            ### F = P(T <= t), S = P(T >= t)
            Fs <- cumsum(st <- table(match(s, su)))
            Ss <- length(s) - Fs + st
            c(max(su[Fs <= alpha * length(s)]),
              min(su[Ss <= alpha * length(s)]))
        }
        ### cf PVAL computation!!!
        rs <- object$perm$permStat
        qu <- .pq(round(rs, 10), alpha = 1 - conf.level)
        att.level <- mean(rs > qu[1] & rs < qu[2])

    }
}
```

◇

Fragment referenced in 68.

The `confint` method starts with Wald intervals, which are either returned or used as starting values for the inversion. We start with the lower bound. Sometimes (for example in case of complete separation), the information is very low so the Wald intervals are extremely wide and the profile log-likelihood cannot be computed. So we try to make the starting interval wider in a step-wise manner. However, this may still fail and we thus exit gently, returning `NA` and issuing a warning.

⟨ *confint lower* 65 ⟩ ≡

```
CINT[p,1] <- max(CINT[p, 1], cf[p] - 1)
sdlwr <- sign(sfun(cf[p], parm = p, quantile = qu[2]))
slwr <- try(sfun(CINT[p,1], parm = p, quantile = qu[2]))
k <- 1
if (inherits(slwr, "try-error")) {
    CINT[p,1] <- NA
} else {
    while ((is.na(slwr) ||
            sign(slwr) == sdlwr) && k < 30) {
        CINT[p,1] <- CINT[p,1] - 1
        slwr <- try(sfun(CINT[p,1], parm = p, quantile = qu[2]))
        if (inherits(slwr, "try-error")) {
            CINT[p,1] <- NA
            break()
        }
        k <- k + 1
    }
}
if (k == 30) {
    CINT[p,1] <- NA
} else {
    lwr <- try(uniroot(sfun, interval = c(CINT[p,1], cf[p]),
                       parm = p, quantile = qu[2])$root)
    if (inherits(lwr, "try-error")) {
        CINT[p,1] <- NA
    } else {
        CINT[p,1] <- lwr
    }
}
if (is.na(CINT[p,1]))
    warning(gettextf("failed to compute confidence interval in %s",
                     "confint.free1way"),
            domain = NA)
```

◇

Fragment referenced in 68.

The upper bound works in the very same way.

⟨ *confint upper* 66 ⟩ ≡

```
CINT[p,2] <- min(CINT[p, 2], cf[p] + 1)
sdupr <- sign(sfun(cf[p], parm = p, quantile = qu[1]))
supr <- try(sfun(CINT[p,2], parm = p, quantile = qu[1]))
k <- 1
if (inherits(supr, "try-error")) {
    CINT[p,2] <- NA
} else {
    while ((is.na(supr) ||
            sign(supr) == sdupr) && k < 30) {
        CINT[p,2] <- CINT[p,2] + 1
        supr <- try(sfun(CINT[p,2], parm = p, quantile = qu[1]))
        if (inherits(supr, "try-error")) {
            CINT[p,2] <- NA
            break()
        }
        k <- k + 1
    }
}
if (k == 30) {
    CINT[p,2] <- NA
} else {
    upr <- try(uniroot(sfun, interval = c(cf[p], CINT[p, 2]),
                       parm = p, quantile = qu[1])$root)
    if (inherits(upr, "try-error")) {
        CINT[p, 2] <- NA
    } else {
        CINT[p, 2] <- upr
    }
}
if (is.na(CINT[p,2]))
    warning(gettextf("failed to compute confidence interval in %s",
                     "confint.free1way"),
            domain = NA)
```

◇

Fragment referenced in 68.

If requested, we post-process the confidence intervals and compute probabilistic indices (aka AUCs) or overlap coefficients

⟨ *PI OVL confint* 67 ⟩ ≡

```
what <- match.arg(what)
CINT <- switch(what, "shift" = CINT,
                     ### probabilistic index = AUC monotone increasing
                     "PI" = object$link$parm2PI(CINT),
                     "AUC" = object$link$parm2PI(CINT), ### same as PI
                     ### overlap coefficient increasing on (-inf, 0)
                     ### and decreasing on (0, inf)
                     "OVL" = {
                         ofun <- object$link$parm2OVL
                         rs <- rowSums(CINT < 0)
                         ### upper < 0
                         if (any(i <- (rs == 2L)))
                             CINT[i,] <- ofun(CINT[i,])
                         ### lower > 0
                         if (any(i <- (rs == 0L)))
                             CINT[i,] <- ofun(-CINT[i,2:1])
                         ### lower < 0 < upper
                         if (any(i <- (rs == 1L))) {
                             aCINT <- abs(CINT[i,,drop = FALSE])
                             CINT[i,1L] <- ofun(-apply(aCINT, 1L, max))
                             CINT[i,2L] <- 1
                         }
                         return(CINT)
                     })[parm, , drop = FALSE]
```

◇

Fragment referenced in 68.

an ensemble everything into one method

⟨ *free1way confint* 68 ⟩ ≡

```
confint.free1way <- function(object, parm,
    level = .95, test = c("Permutation", "Wald", "LRT", "Rao"),
    what = c("shift", "PI", "AUC", "OVL"), ...)
{

    test <- match.arg(test)
    conf.level <- 1 - (1 - level) / 2

    cf <- coef(object)
    if (missing(parm))
        parm <- seq_along(cf)

    att.level <- NULL
    CINT <- confint.default(object, level = level)
    if (test != "Wald") {
        wlevel <- level
        wlevel <- 1 - (1 - level) / 2
        CINT[] <- confint.default(object, level = wlevel)

        sfun <- function(value, parm, quantile)
        {
            x <- object
            alternative <- "two.sided"
            tol <- .Machine$double.eps
            ⟨ statistics 44 ⟩
            ### we also could invert p-values, but the
            ### p-value function might be discrete for permutation
            ### tests, in contrast to the test statistic
            return(STATISTIC - quantile)
        }

        if (test == "Permutation") {
            ⟨ permutation confint 64 ⟩
        } else {
            qu <- rep.int(qchisq(level, df = 1), 2) ### always two.sided
        }

        for (p in parm) {
            ⟨ confint lower 65 ⟩
            ⟨ confint upper 66 ⟩
        }
    }

    ⟨ PI OVL confint 67 ⟩

    if (!is.null(att.level))
        attr(CINT, "Attained level") <- att.level
    return(CINT)
}
◇
```

Fragment referenced in 33.

As an example, we compute log-odds ratios for the table introduced above and report some tests and confidence intervals:

```
R> x

, , A

  A B C
```

```
A 5 6 4
B 3 5 7
C 3 4 5

, , B

  A B C
A 3 5 6
B 0 0 0
C 4 6 5

R> ### asymptotic permutation test
R> (ft <- free1way(x))

        Stratified 3-sample Kruskal-Wallis test against proportional
        odds alternatives

data:   by  (c("A", "B", "C"))
         stratified by
Perm chi-squared = 0.145, df = 2, p-value = 0.93
alternative hypothesis: true log-odds ratio is not equal to 0

R> coef(ft)

       B        C
0.052627 0.204750

R> vcov(ft)

        B       C
B 0.34916 0.20834
C 0.20834 0.33850

R> ### Wald per parameter
R> summary(ft)

Call:
free1way.table(y = x)

Coefficients:
  log-odds ratio Std. Error z value Pr(>|z|)
B         0.0526     0.5909    0.09     0.93
C         0.2047     0.5818    0.35     0.72

R> library("multcomp")
R> summary(glht(ft), test = univariate())

         Simultaneous Tests for General Linear Hypotheses

Fit: free1way.table(y = x)

Linear Hypotheses:
       Estimate Std. Error z value Pr(>|z|)
B == 0   0.0526     0.5909    0.09     0.93
C == 0   0.2047     0.5818    0.35     0.72
(Univariate p values reported)

R> ### global Wald
R> summary(ft, test = "Wald")
```

```
        Stratified 3-sample Kruskal-Wallis test against proportional
        odds alternatives

data:   by  (c("A", "B", "C"))
         stratified by
Wald chi-squared = 0.148, df = 2, p-value = 0.93
alternative hypothesis: true log-odds ratio is not equal to 0

R> summary(glht(ft), test = Chisqtest())

         General Linear Hypotheses

Linear Hypotheses:
       Estimate
B == 0   0.0526
C == 0   0.2047

Global Test:
  Chisq DF Pr(>Chisq)
1 0.148  2      0.929

R> ### Rao score, Permutation score, LRT
R> summary(ft, test = "Rao")

        Stratified 3-sample Kruskal-Wallis test against proportional
        odds alternatives

data:   by  (c("A", "B", "C"))
         stratified by
Rao chi-squared = 0.148, df = 2, p-value = 0.93
alternative hypothesis: true log-odds ratio is not equal to 0

R> summary(ft, test = "Permutation")

        Stratified 3-sample Kruskal-Wallis test against proportional
        odds alternatives

data:   by  (c("A", "B", "C"))
         stratified by
Perm chi-squared = 0.145, df = 2, p-value = 0.93
alternative hypothesis: true log-odds ratio is not equal to 0

R> summary(ft, test = "LRT")

        Stratified 3-sample Kruskal-Wallis test against proportional
        odds alternatives

data:   by  (c("A", "B", "C"))
         stratified by
logLR chi-squared = 0.148, df = 2, p-value = 0.93
alternative hypothesis: true log-odds ratio is not equal to 0

R> ### Wald confidence intervals, unadjusted
R> confint(glht(ft), calpha = univariate_calpha())

         Simultaneous Confidence Intervals

Fit: free1way.table(y = x)

Quantile = 1.96
95% confidence level
```

```
Linear Hypotheses:
       Estimate lwr     upr
B == 0  0.0526  -1.1055  1.2108
C == 0  0.2047  -0.9356  1.3451
```

```
R> confint(ft, test = "Wald")
```

```
     2.5 % 97.5 %
B -1.10552 1.2108
C -0.93558 1.3451
```

```
R> ### Rao and LRT intervals
R> confint(ft, test = "Rao")
```

```
     2.5 % 97.5 %
B -1.09075 1.1954
C -0.92072 1.3301
```

```
R> confint(ft, test = "LRT")
```

```
    2.5 % 97.5 %
B -1.1113 1.2205
C -0.9387 1.3573
```

Chapter 9

Special Test Procedures

We now demonstrate that `free1way` produces the exact same results as some of the classical test procedures implemented in the **stats** package, and how the new implementation extends the existing functionality.

9.1 Wilcoxon Test

The first example is a Wilcoxon test for a single log-odds ratio comparing to treatment groups. The Wilcoxon test is the score test in a 2-sample proportional odds model

```
R> N <- 25
R> w <- gl(2, N)
R> y <- rlogis(length(w), location = c(0, 1)[w])
R> #### link = logit is default
R> ft <- free1way(y ~ w)
R> ### Wald
R> summary(ft)

Call:
free1way.formula(formula = y ~ w)

Coefficients:
   log-odds ratio Std. Error z value Pr(>|z|)
w2          1.290      0.524    2.46    0.014 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

R> ### Permutation test
R> wilcox.test(y ~ w, alternative = "greater", correct = FALSE, exact = FALSE)$p.value

[1] 0.99215

R> pvalue(wilcox_test(y ~ w, alternative = "greater"))

[1] 0.99215

R> summary(ft, test = "Permutation", alternative = "less")$p.value

 Perm Z
0.99215

R> wilcox.test(y ~ w, alternative = "less", correct = FALSE, exact = FALSE)$p.value

[1] 0.0078535

R> pvalue(wilcox_test(y ~ w, alternative = "less"))
```

```
[1] 0.0078535

R> summary(ft, test = "Permutation", alternative = "greater")$p.value

   Perm Z
0.0078535

R> wilcox.test(y ~ w, correct = FALSE, exact = FALSE)$p.value

[1] 0.015707

R> kruskal.test(y ~ w)$p.value

[1] 0.015707

R> pvalue(wilcox_test(y ~ w))

[1] 0.015707

R> summary(ft, test = "Permutation")$p.value

  Perm Z
0.015707

R> ### Wald tests
R> summary(ft, test = "Wald", alternative = "less")

        2-sample Wilcoxon test against proportional odds alternatives

data:  y by w (1, 2)
Wald Z = 2.46, p-value = 0.99
alternative hypothesis: true log-odds ratio is less than 0

R> summary(ft, test = "Wald", alternative = "greater")

        2-sample Wilcoxon test against proportional odds alternatives

data:  y by w (1, 2)
Wald Z = 2.46, p-value = 0.0069
alternative hypothesis: true log-odds ratio is greater than 0

R> summary(ft, test = "Wald")

        2-sample Wilcoxon test against proportional odds alternatives

data:  y by w (1, 2)
Wald chi-squared = 6.06, df = 1, p-value = 0.014
alternative hypothesis: true log-odds ratio is not equal to 0

R> ### Rao score tests
R> summary(ft, test = "Rao", alternative = "less")

        2-sample Wilcoxon test against proportional odds alternatives

data:  y by w (1, 2)
Rao Z = 2.5, p-value = 0.99
alternative hypothesis: true log-odds ratio is less than 0

R> summary(ft, test = "Rao", alternative = "greater")

        2-sample Wilcoxon test against proportional odds alternatives

data:  y by w (1, 2)
Rao Z = 2.5, p-value = 0.0063
alternative hypothesis: true log-odds ratio is greater than 0
```

```
R> summary(ft, test = "Rao")

        2-sample Wilcoxon test against proportional odds alternatives

data:  y by w (1, 2)
Rao chi-squared = 6.24, df = 1, p-value = 0.013
alternative hypothesis: true log-odds ratio is not equal to 0

R> ### LRT (only two-sided)
R> summary(ft, test = "LRT")

        2-sample Wilcoxon test against proportional odds alternatives

data:  y by w (1, 2)
logLR chi-squared = 6.33, df = 1, p-value = 0.012
alternative hypothesis: true log-odds ratio is not equal to 0

R> ### confidence intervals for log-odds ratios
R> confint(ft, test = "Permutation")

     2.5 % 97.5 %
w2 0.23829 2.3423

R> confint(ft, test = "LRT")

     2.5 % 97.5 %
w2 0.28151 2.3446

R> confint(ft, test = "Wald")

     2.5 % 97.5 %
w2 0.26308 2.3168

R> confint(ft, test = "Rao")

     2.5 % 97.5 %
w2 0.27214 2.3084

R> ### confidence interval for "Wilcoxon Parameter" = PI = AUC
R> confint(ft, test = "Rao", what = "AUC")

     2.5 %  97.5 %
w2 0.54524 0.82742

R> ### comparison with rms::orm
R> library("rms")
R> rev(coef(or <- orm(y ~ w)))[1]

 w=2
1.29

R> coef(ft)

  w2
1.29

R> logLik(or)

'log Lik.' -192.44 (df=50)

R> logLik(ft)

[1] -192.44
```

```
R> vcov(or)[2,2]

[1] 0.2745

R> vcov(ft)

       w2
w2 0.2745

R> ci <- confint(or)
R> ci[nrow(ci),]

  2.5 %  97.5 %
0.26308 2.31683

R> confint(ft, test = "Wald")

     2.5 % 97.5 %
w2 0.26308 2.3168
```

9.2 Mantel-Haenszel Test

The Cochran-Mantel-Haenszel test for conditional independence in 2×2 tables also relies on a proportional odds model.

```
R> example(mantelhaen.test, echo = FALSE)
R> mantelhaen.test(UCBAdmissions, correct = FALSE)

        Mantel-Haenszel chi-squared test without continuity correction

data:  UCBAdmissions
Mantel-Haenszel X-squared = 1.52, df = 1, p-value = 0.22
alternative hypothesis: true common odds ratio is not equal to 1
95 percent confidence interval:
 0.77191 1.06033
sample estimates:
common odds ratio
           0.9047

R> ft <- free1way(UCBAdmissions)
R> summary(ft, test = "Wald")

        Stratified 2-sample Wilcoxon test against proportional odds
        alternatives

data:  Admit by Gender (c("Male", "Female"))
         stratified by Dept
Wald chi-squared = 1.53, df = 1, p-value = 0.22
alternative hypothesis: true log-odds ratio is not equal to 0

R> exp(coef(ft))

GenderFemale
     0.90495

R> exp(confint(ft, test = "Wald"))

               2.5 % 97.5 %
GenderFemale 0.77234 1.0603

R> exp(sapply(dimnames(UCBAdmissions)[[3L]], function(dept)
+         confint(free1way(UCBAdmissions[,,dept]), test = "Permutation")))
```

```
          A       B      C       D       E      F
[1,] 0.20967 0.34796 0.8547 0.68644 0.82584 0.4583
[2,] 0.58184 1.85295 1.5021 1.23646 1.80738 1.4956
```

```
R> sapply(dimnames(UCBAdmissions)[[3L]], function(dept)
+         fisher.test(UCBAdmissions[,,dept], conf.int = TRUE)$conf.int)
```

```
          A      B       C       D       E       F
[1,] 0.19704 0.2945 0.84522 0.67896 0.80648 0.43329
[2,] 0.59204 2.0040 1.51629 1.25047 1.83852 1.57563
```

9.3 prop.test

For a single 2×2 table, all tests are nonparametric (as the model is saturated) and therefore also inference procedures result in the same p-values, for example.

```
R> prop.test(UCBAdmissions[,,1], correct = FALSE)
```

```
        2-sample test for equality of proportions without continuity
        correction

data:  UCBAdmissions[, , 1]
X-squared = 17.2, df = 1, p-value = 3.3e-05
alternative hypothesis: two.sided
95 percent confidence interval:
 -0.128682 -0.053034
sample estimates:
 prop 1  prop 2
0.85191 0.94277
```

```
R> summary(free1way(UCBAdmissions[,,1]), test = "Rao")
```

```
        2-sample Wilcoxon test against proportional odds alternatives

data:  Admit by Gender (c("Male", "Female"))
Rao chi-squared = 17.2, df = 1, p-value = 3.3e-05
alternative hypothesis: true log-odds ratio is not equal to 0
```

9.4 Kruskal-Wallis Test

The Kruskal-Wallis test is the score test in a K-sample proportional odds model

```
R> example(kruskal.test, echo = FALSE)
R> kruskal.test(x ~ g)
```

```
        Kruskal-Wallis rank sum test

data:  x by g
Kruskal-Wallis chi-squared = 0.771, df = 2, p-value = 0.68
```

```
R> free1way(x ~ g)
```

```
        3-sample Kruskal-Wallis test against proportional odds
        alternatives

data:  x by g (Normal subjects, Subjects with obstructive airway disease, Subjects with asbestosis)
Perm chi-squared = 0.771, df = 2, p-value = 0.68
alternative hypothesis: true log-odds ratio is not equal to 0
```

9.5 Savage Test

The Savage test assumes proportional odds and, consequently, is the score test in a proportional odds model. We start without censoring (Savage test) and add strata

```
R> library("survival")
R> N <- 10
R> nd <- expand.grid(g = gl(3, N), s = gl(3, N))
R> nd$tm <- rexp(nrow(nd))
R> nd$ev <- TRUE
R> cm <- coxph(Surv(tm, ev) ~ g + strata(s), data = nd)
R> (ft <- free1way(tm ~ g | s, data = nd, link = "cloglog"))

        Stratified 3-sample Savage test against proportional hazards
        alternatives

data:  tm by g (1, 2, 3)
         stratified by s
Perm chi-squared = 4.68, df = 2, p-value = 0.096
alternative hypothesis: true log-hazard ratio is not equal to 0

R> coef(cm)

      g2       g3
-0.15981 -0.16021

R> coef(ft)

     g2      g3
0.16038 0.16058

R> vcov(cm)

          g2        g3
g2 0.0067826 0.0034163
g3 0.0034163 0.0067785

R> vcov(ft)

          g2        g3
g2 0.0067888 0.0034204
g3 0.0034204 0.0067851

R> ### Rao score tests
R> summary(cm)$sctest

   test      df  pvalue
5.03230 2.00000 0.08077

R> summary(ft, test = "Rao")

        Stratified 3-sample Savage test against proportional hazards
        alternatives

data:  tm by g (1, 2, 3)
         stratified by s
Rao chi-squared = 5.06, df = 2, p-value = 0.08
alternative hypothesis: true log-hazard ratio is not equal to 0

R> ### likelihood ratio tests
R> summary(cm)$logtest
```

```
    test       df   pvalue
4.933154 2.000000 0.084875

R> summary(ft, test = "LRT")

        Stratified 3-sample Savage test against proportional hazards
        alternatives

data:  tm by g (1, 2, 3)
         stratified by s
logLR chi-squared = 4.96, df = 2, p-value = 0.084
alternative hypothesis: true log-hazard ratio is not equal to 0

R> ### Wald tests
R> summary(cm)$waldtest

    test       df   pvalue
5.020000 2.000000 0.081193

R> summary(ft, test = "Wald")

        Stratified 3-sample Savage test against proportional hazards
        alternatives

data:  tm by g (1, 2, 3)
         stratified by s
Wald chi-squared = 5.05, df = 2, p-value = 0.08
alternative hypothesis: true log-hazard ratio is not equal to 0

R> ### asymptotic permutation tests
R> survdiff(Surv(tm, ev) ~ g + strata(s), data = nd, rho = 0)[c("chisq", "pvalue")]

$chisq
[1] 5.0323

$pvalue
[1] 0.08077

R> summary(ft, test = "Permutation")

        Stratified 3-sample Savage test against proportional hazards
        alternatives

data:  tm by g (1, 2, 3)
         stratified by s
Perm chi-squared = 4.68, df = 2, p-value = 0.096
alternative hypothesis: true log-hazard ratio is not equal to 0

R> library("coin")
R> independence_test(Surv(tm, ev) ~ g | s, data = nd, ytrafo = function(...)
+                     trafo(..., numeric_trafo = logrank_trafo, block = nd$s),
+                     teststat = "quad")

        Asymptotic General Independence Test

data:  Surv(tm, ev) by g (1, 2, 3)
         stratified by s
chi-squared = 4.75, df = 2, p-value = 0.093
```

Wilcoxon against proportional odds

```
R> survdiff(Surv(tm, ev) ~ g + strata(s), data = nd, rho = 1)[c("chisq", "pvalue")]
```

```
$chisq
[1] 5.3318

$pvalue
[1] 0.069535

R> (ft <- free1way(tm ~ g | s, data = nd, link = "logit"))

        Stratified 3-sample Kruskal-Wallis test against proportional
        odds alternatives

data:  tm by g (1, 2, 3)
         stratified by s
Perm chi-squared = 5.2, df = 2, p-value = 0.074
alternative hypothesis: true log-odds ratio is not equal to 0

R> summary(ft)

Call:
free1way.formula(formula = tm ~ g | s, data = nd, link = "logit")

Coefficients:
   log-odds ratio Std. Error z value Pr(>|z|)
g2          0.301      0.142    2.12    0.034 *
g3          0.261      0.142    1.83    0.067 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

R> summary(ft, test = "Rao")

        Stratified 3-sample Kruskal-Wallis test against proportional
        odds alternatives

data:  tm by g (1, 2, 3)
         stratified by s
Rao chi-squared = 5.27, df = 2, p-value = 0.072
alternative hypothesis: true log-odds ratio is not equal to 0

R> summary(ft, test = "LRT")

        Stratified 3-sample Kruskal-Wallis test against proportional
        odds alternatives

data:  tm by g (1, 2, 3)
         stratified by s
logLR chi-squared = 5.27, df = 2, p-value = 0.072
alternative hypothesis: true log-odds ratio is not equal to 0

R> summary(ft, test = "Wald")

        Stratified 3-sample Kruskal-Wallis test against proportional
        odds alternatives

data:  tm by g (1, 2, 3)
         stratified by s
Wald chi-squared = 5.26, df = 2, p-value = 0.072
alternative hypothesis: true log-odds ratio is not equal to 0

R> summary(ft, test = "Permutation")
```

```
        Stratified 3-sample Kruskal-Wallis test against proportional
        odds alternatives

data:  tm by g (1, 2, 3)
         stratified by s
Perm chi-squared = 5.2, df = 2, p-value = 0.074
alternative hypothesis: true log-odds ratio is not equal to 0
```

9.6 Log-rank Test

And now with censoring. We cannot expect this to be identical with what **survival** reports, as this package is based on the partial likelihood and we operate on the full likelihood.

```
R> library("survival")
R> data("GBSG2", package = "TH.data")
R> cm <- coxph(Surv(time, cens) ~ horTh + strata(tgrade), data = GBSG2)
R> ft <- with(GBSG2, free1way(Surv(time, cens) ~ horTh | tgrade,
+                             link = "cloglog"))
R> coef(cm)

horThyes
-0.33972

R> coef(ft)

horThyes
 0.34056

R> vcov(cm)

         horThyes
horThyes 0.015754

R> vcov(ft)

         horThyes
horThyes 0.015751

R> ### Rao score tests
R> summary(cm)$sctest

     test        df    pvalue
7.3935325 1.0000000 0.0065459

R> summary(ft, test = "Rao")

        2-sample Savage test against proportional hazards alternatives

data:  Surv(time, cens) by horTh (no, yes)
         stratified by tgrade
Rao chi-squared = 7.43, df = 1, p-value = 0.0064
alternative hypothesis: true log-hazard ratio is not equal to 0

R> ### likelihood ratio tests
R> summary(cm)$logtest

     test        df    pvalue
7.6027550 1.0000000 0.0058279

R> summary(ft, test = "LRT")
```

```
        2-sample Savage test against proportional hazards alternatives

data:  Surv(time, cens) by horTh (no, yes)
         stratified by tgrade
logLR chi-squared = 7.64, df = 1, p-value = 0.0057
alternative hypothesis: true log-hazard ratio is not equal to 0

R> ### Wald tests
R> summary(cm)$waldtest

     test        df    pvalue
7.3300000 1.0000000 0.0067971

R> summary(ft, test = "Wald")

        2-sample Savage test against proportional hazards alternatives

data:  Surv(time, cens) by horTh (no, yes)
         stratified by tgrade
Wald chi-squared = 7.36, df = 1, p-value = 0.0067
alternative hypothesis: true log-hazard ratio is not equal to 0

R> ### asymptotic permutation tests
R> survdiff(Surv(time, cens) ~ horTh + strata(tgrade), data = GBSG2, rho = 0)[c("chisq", "pvalue")]

$chisq
[1] 7.3958

$pvalue
[1] 0.0065376

R> summary(ft, test = "Permutation")

        2-sample Savage test against proportional hazards alternatives

data:  Surv(time, cens) by horTh (no, yes)
         stratified by tgrade
Perm Z = 2.76, p-value = 0.0058
alternative hypothesis: true log-hazard ratio is not equal to 0

R> independence_test(Surv(time, cens) ~ horTh | tgrade, data = GBSG2, ytrafo = function(...)
+                    trafo(..., numeric_trafo = logrank_trafo, block = GBSG2$tgrade),
+                    teststat = "quad")

        Asymptotic General Independence Test

data:  Surv(time, cens) by
         horTh (no, yes)
         stratified by tgrade
chi-squared = 7.63, df = 1, p-value = 0.0057
```

Wilcoxon against proportional odds

```
R> survdiff(Surv(time, cens) ~ horTh + strata(tgrade), data = GBSG2, rho = 1)[c("chisq", "pvalue")]

$chisq
[1] 7.5938

$pvalue
[1] 0.0058569
```

```
R> (ft <- with(GBSG2, free1way(Surv(time, cens) ~ horTh | tgrade,
+                               link = "logit")))

        2-sample Wilcoxon test against proportional odds alternatives

data:  Surv(time, cens) by horTh (no, yes)
         stratified by tgrade
Perm Z = 2.78, p-value = 0.0054
alternative hypothesis: true log-odds ratio is not equal to 0

R> summary(ft, test = "Rao")

        2-sample Wilcoxon test against proportional odds alternatives

data:  Surv(time, cens) by horTh (no, yes)
         stratified by tgrade
Rao chi-squared = 7.8, df = 1, p-value = 0.0052
alternative hypothesis: true log-odds ratio is not equal to 0

R> summary(ft, test = "LRT")

        2-sample Wilcoxon test against proportional odds alternatives

data:  Surv(time, cens) by horTh (no, yes)
         stratified by tgrade
logLR chi-squared = 7.89, df = 1, p-value = 0.005
alternative hypothesis: true log-odds ratio is not equal to 0

R> summary(ft, test = "Wald")

        2-sample Wilcoxon test against proportional odds alternatives

data:  Surv(time, cens) by horTh (no, yes)
         stratified by tgrade
Wald chi-squared = 7.75, df = 1, p-value = 0.0054
alternative hypothesis: true log-odds ratio is not equal to 0

R> summary(ft, test = "Permutation")

        2-sample Wilcoxon test against proportional odds alternatives

data:  Surv(time, cens) by horTh (no, yes)
         stratified by tgrade
Perm Z = 2.78, p-value = 0.0054
alternative hypothesis: true log-odds ratio is not equal to 0
```

And now with more and smaller blocks

```
R> library("survival")
R> GBSG2$str <- cut(GBSG2$tsize, breaks = c(0, 1:9 * 10, Inf))
R> cm <- coxph(Surv(time, cens) ~ horTh + strata(str), data = GBSG2)
R> ft <- with(GBSG2, free1way(Surv(time, cens) ~ horTh | str,
+                             link = "cloglog"))
R> coef(cm)

horThyes
-0.33921

R> coef(ft)

horThyes
 0.33879
```

```
R> vcov(cm)

         horThyes
horThyes 0.016147

R> vcov(ft)

         horThyes
horThyes 0.016204

R> ### Rao score tests
R> summary(cm)$sctest

     test        df    pvalue
7.1917181 1.0000000 0.0073241

R> summary(ft, test = "Rao")

        2-sample Savage test against proportional hazards alternatives

data:  Surv(time, cens) by horTh (no, yes)
         stratified by str
Rao chi-squared = 7.15, df = 1, p-value = 0.0075
alternative hypothesis: true log-hazard ratio is not equal to 0

R> ### likelihood ratio tests
R> summary(cm)$logtest

     test        df    pvalue
7.3938622 1.0000000 0.0065447

R> summary(ft, test = "LRT")

        2-sample Savage test against proportional hazards alternatives

data:  Surv(time, cens) by horTh (no, yes)
         stratified by str
logLR chi-squared = 7.35, df = 1, p-value = 0.0067
alternative hypothesis: true log-hazard ratio is not equal to 0

R> ### Wald tests
R> summary(cm)$waldtest

     test        df    pvalue
7.1300000 1.0000000 0.0075971

R> summary(ft, test = "Wald")

        2-sample Savage test against proportional hazards alternatives

data:  Surv(time, cens) by horTh (no, yes)
         stratified by str
Wald chi-squared = 7.08, df = 1, p-value = 0.0078
alternative hypothesis: true log-hazard ratio is not equal to 0

R> ### asymptotic permutation tests
R> survdiff(Surv(time, cens) ~ horTh + strata(str), data = GBSG2, rho = 0)[c("chisq", "pvalue")]

$chisq
[1] 7.1868

$pvalue
[1] 0.0073441
```

```
R> summary(ft, test = "Permutation")

        2-sample Savage test against proportional hazards alternatives

data:  Surv(time, cens) by horTh (no, yes)
         stratified by str
Perm Z = 2.67, p-value = 0.0076
alternative hypothesis: true log-hazard ratio is not equal to 0
```

Wilcoxon against proportional odds

```
R> survdiff(Surv(time, cens) ~ horTh + strata(str), data = GBSG2, rho = 1)[c("chisq", "pvalue")]

$chisq
[1] 8.2045

$pvalue
[1] 0.0041787

R> (ft <- with(GBSG2, free1way(Surv(time, cens) ~ horTh | str,
+                               link = "logit")))

        2-sample Wilcoxon test against proportional odds alternatives

data:  Surv(time, cens) by horTh (no, yes)
         stratified by str
Perm Z = 2.88, p-value = 0.0039
alternative hypothesis: true log-odds ratio is not equal to 0

R> summary(ft, test = "Rao")

        2-sample Wilcoxon test against proportional odds alternatives

data:  Surv(time, cens) by horTh (no, yes)
         stratified by str
Rao chi-squared = 8.47, df = 1, p-value = 0.0036
alternative hypothesis: true log-odds ratio is not equal to 0

R> summary(ft, test = "LRT")

        2-sample Wilcoxon test against proportional odds alternatives

data:  Surv(time, cens) by horTh (no, yes)
         stratified by str
logLR chi-squared = 8.57, df = 1, p-value = 0.0034
alternative hypothesis: true log-odds ratio is not equal to 0

R> summary(ft, test = "Wald")

        2-sample Wilcoxon test against proportional odds alternatives

data:  Surv(time, cens) by horTh (no, yes)
         stratified by str
Wald chi-squared = 8.42, df = 1, p-value = 0.0037
alternative hypothesis: true log-odds ratio is not equal to 0

R> summary(ft, test = "Permutation")

        2-sample Wilcoxon test against proportional odds alternatives

data:  Surv(time, cens) by horTh (no, yes)
         stratified by str
Perm Z = 2.88, p-value = 0.0039
alternative hypothesis: true log-odds ratio is not equal to 0
```

9.7 van der Waerden Test

Normal scores test against a generalised Cohen's d:

```
R> nd$y <- rnorm(nrow(nd))
R> free1way(y ~ g | s, data = nd, link = "probit")

        Stratified 3-sample van der Waerden normal scores test against
        latent normal shift alternatives

data:  y by g (1, 2, 3)
         stratified by s
Perm chi-squared = 1.01, df = 2, p-value = 0.6
alternative hypothesis: true generalised Cohen's d is not equal to 0

R> independence_test(y ~ g | s, data = nd, ytrafo = function(...)
+                    trafo(..., numeric_trafo = normal_trafo, block = nd$s),
+                    teststat = "quad")

        Asymptotic General Independence Test

data:  y by g (1, 2, 3)
         stratified by s
chi-squared = 1.06, df = 2, p-value = 0.59
```

9.8 Friedman Test

Each observation is a block in a K-sample proportional odds model

```
R> friedman.test(RoundingTimes)

        Friedman rank sum test

data:  RoundingTimes
Friedman chi-squared = 11.1, df = 2, p-value = 0.0038

R> (ft <- free1way(time ~ me | id, data = d))

        Stratified 3-sample Kruskal-Wallis test against proportional
        odds alternatives

data:  time by me (Round Out, Narrow Angle, Wide Angle)
         stratified by id
Perm chi-squared = 11.1, df = 2, p-value = 0.0038
alternative hypothesis: true log-odds ratio is not equal to 0

R> logLik(ft)

[1] -55.857

R> logLik(free1way(time ~ me | id, data = d, link = "probit"))

[1] -56.579

R> logLik(free1way(time ~ me | id, data = d, link = "cloglog"))

[1] -53.993

R> logLik(free1way(time ~ me | id, data = d, link = "loglog"))

[1] -55.624
```

```
R> example(friedman.test, echo = FALSE)
R> ### Myles Hollander & Wolfe (2014, Example 7.1, page 294)
R> boxplot(RoundingTimes, xlab = "Method", ylab = "Rounding-First-Base Time",
+         las = 1)
R> matplot(t(RoundingTimes), add = TRUE, type = "l",
+         lty = 1, lwd = 2, col = rgb(.1, .1, .1, .1))
R> me <- colnames(RoundingTimes)
R> d <- expand.grid(me = factor(me, labels = me, levels = me),
+                   id = factor(seq_len(nrow(RoundingTimes))))
R> d$time <- c(t(RoundingTimes))
```

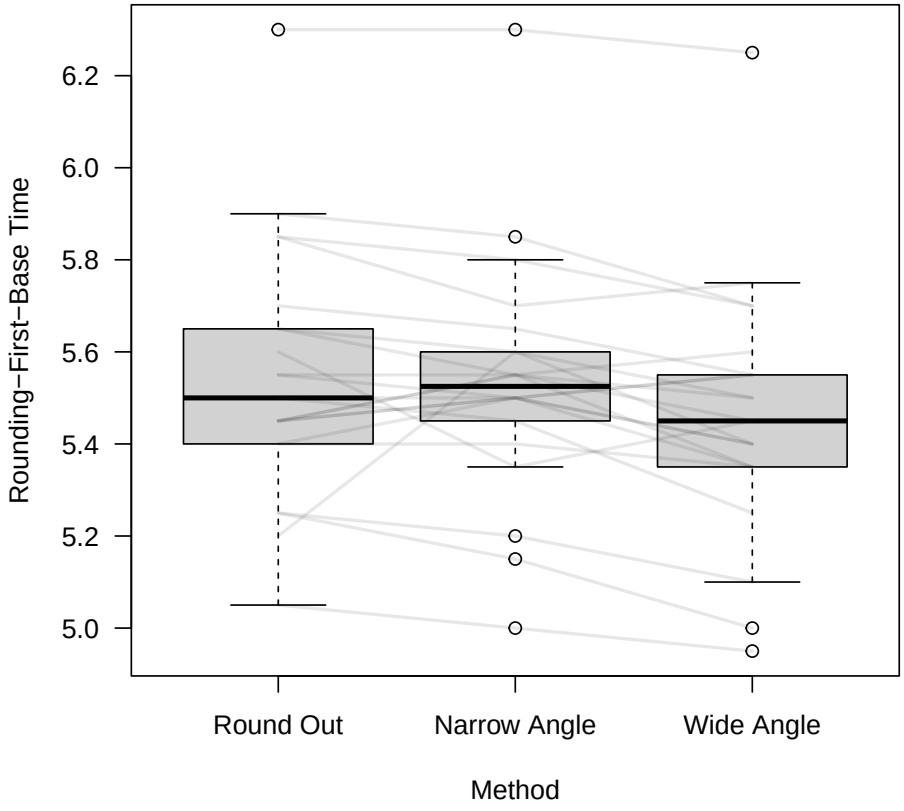


Figure 9.1: Rounding-first-base time data.

9.9 McNemar Test

```
R> example(mcnemar.test, echo = FALSE)
R> # set-up data frame with survey outcomes for voters
R> s <- gl(2, 1, labels = dimnames(Performance)[[1L]])
R> survey <- gl(2, 1, labels = c("1st", "2nd"))
R> nvoters <- c(Performance)
R> x <- expand.grid(survey = survey, voter = factor(seq_len(sum(nvoters))))
R> x$performance <- c(rep(s[c(1, 1)], nvoters[1]), rep(s[c(2, 1)], nvoters[2]),
+                     rep(s[c(1, 2)], nvoters[3]), rep(s[c(2, 2)], nvoters[4]))
R> # note that only those voters changing their minds are relevant
R> mcn <- free1way(xtabs(~ performance + survey + voter, data = x))
R> # same result as mcnemar.test w/o continuity correction
R> print(mcn)

        Stratified 2-sample Wilcoxon test against proportional odds
        alternatives

data:  performance by survey (c("1st", "2nd"))
         stratified by voter
Perm Z = 4.17, p-value = 3.1e-05
alternative hypothesis: true log-odds ratio is not equal to 0

R> # X^2 statistic
R> summary(mcn, test = "Permutation")$statistic^2

Perm Z
17.356

R> mcnemar.test(Performance, correct = FALSE)

        McNemar's Chi-squared test

data:  Performance
McNemar's chi-squared = 17.4, df = 1, p-value = 3.1e-05

R> # Wald inference
R> summary(mcn)

Call:
free1way.table(y = xtabs(~performance + survey + voter, data = x))

Coefficients:
          log-odds ratio Std. Error z value Pr(>|z|)
survey2nd          1.113      0.191    5.82    6e-09 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

R> confint(mcn, test = "Wald")

            2.5 % 97.5 %
survey2nd 0.73767 1.4875

R> ### because the model is saturated, the link function doesn't affect the
R> ### p-value (but the coefficients are of course different)
R> free1way(xtabs(~ performance + survey + voter, data = x), link = "probit")

        Stratified 2-sample van der Waerden normal scores test against
        latent normal shift alternatives

data:  performance by survey (c("1st", "2nd"))
         stratified by voter
Perm Z = 4.17, p-value = 3.1e-05
alternative hypothesis: true generalised Cohen's d is not equal to 0
```

9.10 Incomplete Block Designs

`friedman.test` expects all blocks to be complete and `kruskal.test` has no idea about blocks. When blocks are incomplete, `free1way` can be employed. Replacing the normality assumption inherit in `aov` (Chapter 8.3.1. in Meier, 2022) with a semiparametric proportional odds model, we get

```
R> data("taste", package = "daewr")
R> ### highly discrete
R> table(taste$score)

4 5 6 7 8
4 6 6 6 2

R> summary(free1way(score ~ recipe | panelist, data = taste))

Call:
free1way.formula(formula = score ~ recipe | panelist, data = taste)

Coefficients:
        log-odds ratio Std. Error z value Pr(>|z|)
recipeB           2.48       2.12    1.17     0.24
recipeC           2.93       1.93    1.52     0.13
recipeD          -1.70       1.79   -0.95     0.34
```

9.11 Contrasts Tests

`free1way` output can be used to define multiple contrasts tests and corresponding confidence intervals via the **multcomp** package. For example, Tukey-style simultaneous all-pair comparisons can be implemented via

```
R> tk <- free1way(Ozone ~ Month, data = airquality)
R> library("multcomp")
R> confint(glht(tk, linfct = mcp(Month = "Tukey")))

         Simultaneous Confidence Intervals

Multiple Comparisons of Means: Tukey Contrasts


Fit: free1way.formula(formula = Ozone ~ Month, data = airquality)

Quantile = 2.718
95% family-wise confidence level


Linear Hypotheses:
           Estimate lwr     upr
6 - 5 == 0  0.8124  -0.9382  2.5629
7 - 5 == 0  2.5282   1.0837  3.9726
8 - 5 == 0  2.3826   0.9322  3.8330
9 - 5 == 0  0.7513  -0.5308  2.0334
7 - 6 == 0  1.7158  -0.0692  3.5008
8 - 6 == 0  1.5702  -0.2219  3.3624
9 - 6 == 0 -0.0610  -1.7487  1.6266
8 - 7 == 0 -0.1456  -1.4745  1.1834
9 - 7 == 0 -1.7768  -3.1206 -0.4331
9 - 8 == 0 -1.6313  -2.9843 -0.2783
```

9.12 Scale Tests

"scaleTest.R" 69≡

⟨ *scale tests* 70 ⟩
◇

If, instead of the location model (1.1), a scale-model

$$F_Y(y \mid G = k, S = b) = F\left(F^{-1}\left(F_Y(y \mid G = 1, S = b)\right) / \exp(\xi_k)\right), \quad k = 2, \ldots, K \tag{9.1}$$

is assumed, we might be interested in testing the null hypotheses of the form $H_0 : \xi_k = 0$ instead. Unfortunately, we cannot estimate ξ_k (because the negative log-likelihood is no longer convex in its parameters) except under the null of all scale parameters being zero. However, for permutation score *tests*, it is sufficient to fit the model under the null and the experimental function `scaleTest` reports corresponding permutation p-values.

⟨ *scale tests* 70 ⟩ ≡

```
scaleTest <- function(object, B = 0, ...) {

    alternative <- "two.sided"
    y <- object$table
    d <- dim(y)
    ret <- object
    cf <- ret$par
    cf[idx <- seq_len(d[2L] - 1L)] <- 0
    pr <- ret$profile(cf, idx)
    res <- - pr$negscaleresiduals

    ⟨ Strasser Weber 50 ⟩
    ⟨ resampling 51 ⟩

    if (length(dim(y)) == 3L) y <- y[,,ret$strata, drop = FALSE]
    if (length(dim(y)) == 4L) {
        y <- y[,,ret$strata,, drop = FALSE]
        dy <- dim(y)
        dy[1] <- dy[1] * 2
        y <- apply(y, 3, function(x) rbind(x[,,"TRUE"], x[,,"FALSE"]), simplify = FALSE)
        y <- array(unlist(y), dim = dy[1:3])
    }
    ret$perm <- .resample(res, y, B = B)

    STATISTIC <- ret$perm$testStat
    x <- ret
    cf <- coef(object)
    sc <- 0
    ⟨ Permutation p-values 49 ⟩

    RVAL <- list(statistic = STATISTIC, parameter = NULL, p.value = PVAL,
        null.value = 1, alternative = alternative, method = "Permutation scale test",
        data.name = x$data.name)
    class(RVAL) <- "htest"
    return(RVAL)
}
◇
```

Fragment referenced in 69.

With a probit link, the corresponding test is known as Klotz test. A comparison to its implementation in the **coin** package does not result in identical p-values, because the normal scores $\Phi^{-1}(i/(N+1))$ used in

coin (with i being the rank of the observation) are close to but not identical to the empirical likelihood estimators of the intercept parameters.

```
R> example(klotz_test, echo = FALSE)
R> klotz_test(serum ~ method, data = sid)

        Asymptotic Two-Sample Klotz Test

data:  serum by method (Ramsay, Jung-Parekh)
Z = 0.783, p-value = 0.43
alternative hypothesis: true ratio of scales is not equal to 1

R> scaleTest(free1way(serum ~ method, data = sid, link = "probit"))

        Permutation scale test

data:  serum by method (Ramsay, Jung-Parekh)
= -0.763, p-value = 0.45
alternative hypothesis: true  is not equal to 1
```

Chapter 10

Model Diagnostics

10.1 Transformation Plots

The model formulation (1.1) suggests a simple graphical check of the main model assumption, that is, the existence of a constant shift on a latent scale defined by F. For one block and two samples, the model reads

$$\begin{aligned} F_Y(y \mid G = 1) &= F\left(F^{-1}(F_Y(y \mid G = 1))\right) = F(h(y)) \\ F_Y(y \mid G = 2) &= F\left(F^{-1}(F_Y(y \mid G = 1)) - \delta_2\right) = F(h(y) - \delta_2). \end{aligned}$$

We can now contrast the conditional distributions obtained from this model with the marginally estimated distribution functions of Y, that is, nonparametric estimates for the two distribution functions obtained within each treatment group separately. These latter estimates are typically simply the ECDF or Kaplan-Meier estimators in the presence of right-censoring. It is easier to see deviations from model (1.1) when the plot is presented on the scale of the link function F^{-1}. If the control distribution is close to $\hat{h}(y)$ and the distribution of those treated close to $\hat{h}(y) - \hat{\delta}_2$, model (1.1) provides a good approximation. If the two curves cross or if their horizontal distance varies considerably across the sample space, we should be concerned.

The model is assumed to hold in each block with overall treatment effects δ_k, but the intercept function h may differ between blocks. Therefore, we produce such a plot for each block separately. We also do not pay attention to the original observations of the outcome but plot the model on the scale of the ranked outcomes (the model is invariant with respect to monotone and therefore rank transformations).

All the necessary information can be extracted from an `object` of class `free1way`. We extract the subtable containing the data for `block`, paying attention to possible right-censoring (in the four dimension)

⟨ *extract plot data* 71 ⟩ ≡

```
object <- x
x <- object$table
if (RC <- (length(dim(x)) == 4L)) {
    x <- x[,,block,,drop = FALSE]
    x <- x[marginSums(x, margin = 1) > 0,,,,drop = FALSE]
} else {
    x <- x[,,block,drop = FALSE]
    x <- x[marginSums(x, margin = 1) > 0,,,drop = FALSE]
}
K <- dim(x)[2L]
ret0 <- matrix(NA, nrow = dim(x)[1L], ncol = K)
ln <- object$link
◇
```

Fragment referenced in 78.

We then refit the intercept parameters (that is, $\hat{h}(y)$) for this block re-using the treatment effects already contained in `object`. We will plot them later on as a means to directly compare the model to the data

⟨ *refit block intercepts* 72 ⟩ ≡

```
### refit for this block only
m1 <- .free1wayML(x, link = ln, start = coef(object),
                  fix = seq_along(coef(object)),
                  residuals = FALSE, hessian = FALSE)
intercepts <- m1$intercepts[[1L]]
j1 <- which(attr(get("xlist", environment(m1$profile))[[1L]], "idx") > 1)
j1 <- j1[-length(j1)]
cf <- c(0, coef(object))
```

◇

Fragment referenced in 78.

Last, we compute the marginal distributions, that is, the distribution function of the outcome separately for each group. We could have used `ecdf` or `survfit` here, but since we have everything available in `.free1wayML`, we simply remove the observations corresponding to other groups and refit the intercept parameters.

⟨ *marginal fit* 73 ⟩ ≡

```
for (k in seq_len(K)) {
    y <- x
    if (RC) {
        y[,-k,1,] <- 0
    } else {
        y[,-k,1] <- 0
    }
    start <- numeric(K - 1)
    m0 <- .free1wayML(y, link = ln, start = start,
                      fix = seq_len(K - 1), residuals = FALSE,
                      hessian = FALSE)
    j <- which(attr(get("xlist", environment(m0$profile))[[1L]], "idx") > 1)
    ret0[j[-length(j)],k] <- m0$intercepts[[1L]]
}
```

◇

Fragment referenced in 78.

We can now setup a `plot` method for `free1way` objects, we begin with an empty plot with appropriate axes annotations (we allow plotting on the scale of the CDF as well):

⟨ *setup canvas* 74 ⟩ ≡

```
if (cdf) {
    ylim <- c(0, 1)
    FUN <- function(x) ln$linkinv(x)
} else {
    ylim <- range(c(ret0, intercepts), na.rm = TRUE)
    FUN <- function(x) x
}

idx <- seq_len(nrow(x))
main <- list(...)$main
if (is.null(main) && dim(object$table)[3L] > 1L)
    main <- paste(names(dimnames(x))[3L], dimnames(x)[[3L]][1L], sep = "=")
plot(idx, rep(0, length(idx)), type = "n", ylim = ylim,
     xlab = paste("Rank(", names(dimnames(x))[1L], ")", sep = ""),
     ylab = ifelse(cdf, "Probability", paste(ln$name, "Link")),
     main = main, ...)
```

◇

Fragment referenced in 78.

The marginally estimated functions are plotted first

⟨ *marginal plot* 75 ⟩ ≡

```
out <- sapply(seq_len(K), function(k)
    lines(which(!is.na(ret0[,k])), FUN(ret0[!is.na(ret0[,k]),k]),
          type = "s", col = col[k], lty = lty[1]))
```

◇

Fragment referenced in 78.

followed by a plot of the model-based functions (which can be switched off)

⟨ *model plot* 76 ⟩ ≡

```
if (model)
    out <- sapply(seq_len(K), function(k)
        lines(j1, FUN(intercepts - cf[k]), type = "s", col = col[k],
              lty = lty[2]))
```

◇

Fragment referenced in 78.

and finally we add a legend

⟨ *add legend* 77 ⟩ ≡

```
if (legend) {
        legend("topleft", lty = lty[1], col = col,
               legend = paste(names(dimnames(x))[2L], dimnames(x)[[2L]]),
               title = "Nonparametric", bty = "n")
        if (model)
            legend("bottomright", lty = lty[2], col = col,
                   legend = paste(names(dimnames(x))[2L], dimnames(x)[[2L]]),
                   title = "Semiparametric", bty = "n")
    }
```

◇

Fragment referenced in 78.

We put everything together in a `plot` method

⟨ *plot free1way* 78 ⟩ ≡

```
plot.free1way <- function(x, ..., block = 1L, cdf = FALSE, model = TRUE,
                          col = seq_len(length(coef(object)) + 1L),
                          lty = 1:2, legend = TRUE)
{

    ⟨ extract plot data 71 ⟩
    ⟨ refit block intercepts 72 ⟩
    ⟨ marginal fit 73 ⟩

    ⟨ setup canvas 74 ⟩
    ⟨ marginal plot 75 ⟩
    ⟨ model plot 76 ⟩
    ⟨ add legend 77 ⟩
}
```

◇

Fragment referenced in 33.

```
R> tk <- free1way(Ozone ~ Month, data = airquality)
R> plot(tk, las = 1)
```

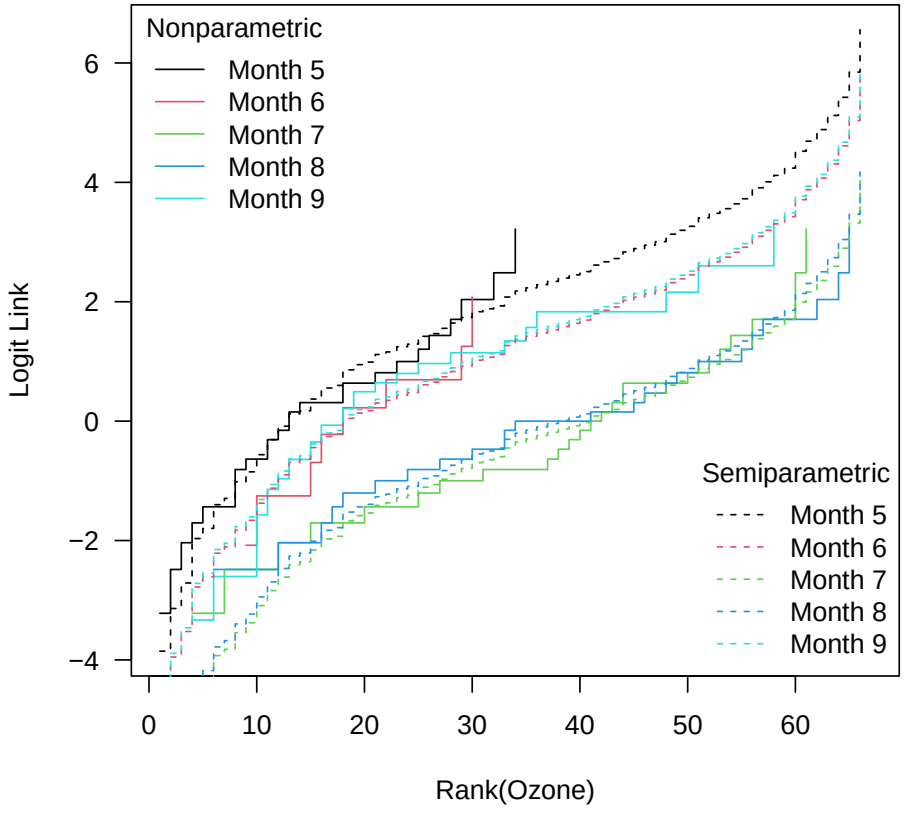


Figure 10.1: Model diagnostics for proportional odds model comparing ozone concentrations for different months.

By default, the plot shows the marginal ("nonparametric") and model-based ("semiparametric") estimates in the sample plot. For the ozone concentrations in different months in Figure 10.1, we see that the distributions differ, but the effects can be well understood as shifts on a log-odds scale. Note that the solid nonparametric curves agree quite well with the dashed model-based ones.

We can check if the plot is correct by comparing the result (Figure 10.2) to the one obtained with `ecdfplot` after rank transformation on the scale of the distribution functions (Figure 10.3)

10.2 Probability-probability Plots

The classical shift model $F_Y(y \mid T = 2) = F_Y(y - \mu \mid T = 1)$ can be criticised using confidence bands for QQ-plots in `qqplot`, because the parameter μ shows up as a vertical shift of the diagonal if the model is appropriate.

Likewise, model (1.1) can be graphically assessed using the P-P-plot. We concentrate on the two-sample case. The shift parameter δ_2 gives rise to the model-based P-P graph $(p, F(F^{-1}(p) - \delta_2))$ and a confidence *band* can be obtained from a confidence *interval* for δ_2. The P-P-plot is, up to rescalings, identical to the ROC curve.

```
R> plot(tk, cdf = TRUE, model = FALSE, las = 1)
```

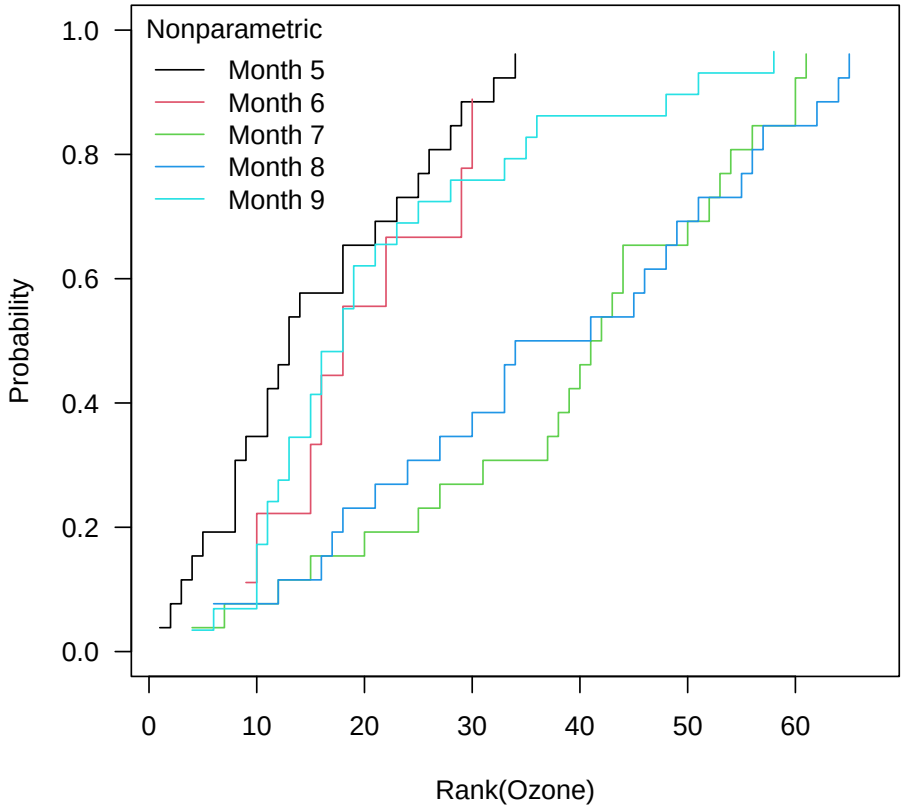


Figure 10.2: Nonparametric distributions of ozone concentrations for different months.

```
R> aq <- subset(airquality, !is.na(Ozone))
R> aq$r <- match(aq$Ozone, sort(unique(aq$Ozone)))
R> library("latticeExtra")
R> plot(ecdfplot(~ r, data = aq, groups = Month, col = 1:5))
```

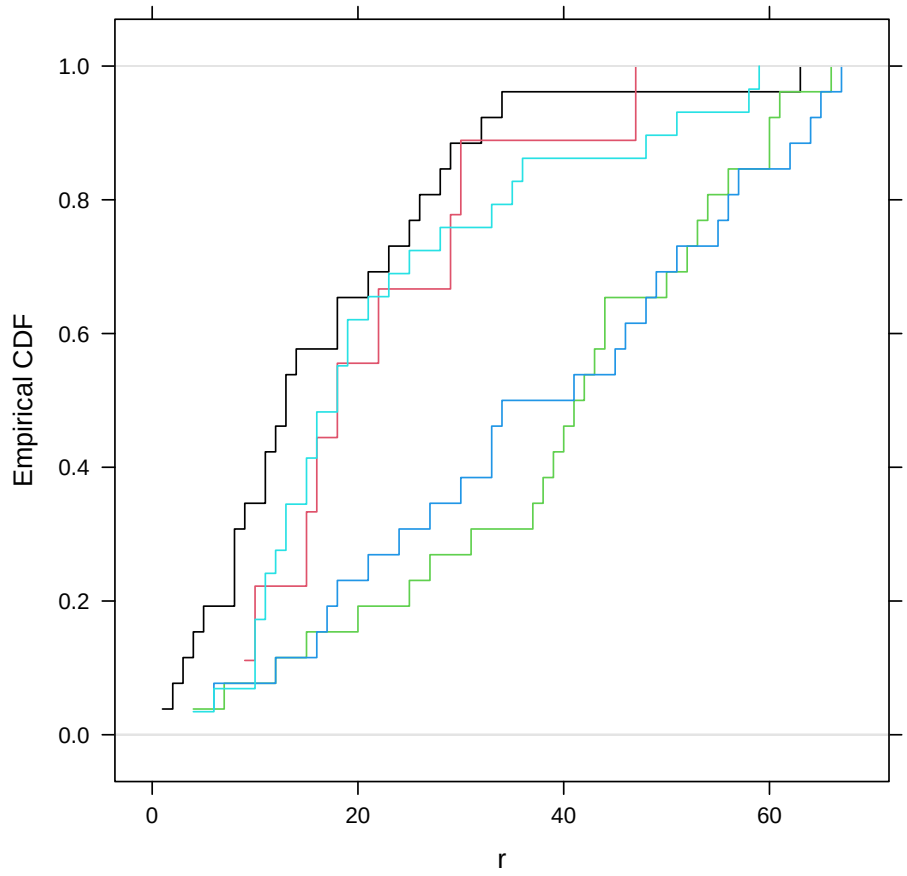


Figure 10.3: Nonparametric distributions of ozone concentrations for different months.

⟨ *ROC bands* 79 ⟩ ≡

```
if (!is.null(conf.level)) {
   prb <- seq_len(1000) / 1001
   res <- c(x, y)
   grp <- gl(2, 1, labels = c(xlab, ylab))
   grp <- grp[rep(1:2, c(length(x), length(y)))]
   args <- conf.args
   args$y <- res
   args$groups <- grp
   args$border <- args$col <- args$type <- NULL
   f1w <- do.call("free1way", args)

   ci <- confint(f1w, level = conf.level, type = args$type)
   lwr <- .p(f1w$link, .q(f1w$link, prb) - ci[1,1])
   upr <- .p(f1w$link, .q(f1w$link, prb) - ci[1,2])
   x <- c(prb, rev(prb))
   y <- c(lwr, rev(upr))
   xn <- c(x[1L], rep(x[-1L], each = 2))
   yn <- c(rep(y[-length(y)], each = 2), y[length(y)])
   polygon(x = xn, y = yn, col = conf.args$col, border = conf.args$border)
   lines(prb, .p(f1w$link, .q(f1w$link, prb) - coef(f1w)))
}
```

◇

Fragment referenced in 80.

We introduce a new function `pppplot`, closely following the implementation of `qqplot`, allowing to plot the empirical (Wilk and Gnanadesikan, 1968) and corresponding model-based (Sewak and Hothorn, 2023) probability-probability plot, the latter for a certain choice of link function:

⟨ *ppplot* 80 ⟩ ≡

```
ppplot <- function(x, y, plot.it = TRUE,
                   xlab = paste("Cumulative probabilities for",
                                deparse1(substitute(x))),
                   ylab = paste("Cumulative probabilities for",
                                deparse1(substitute(y))),
                   main = "P-P plot",
                   ..., conf.level = NULL,
                   conf.args = list(link = "logit", type = "Wald",
                                    col = NA, border = NULL))
{

    force(xlab)
    force(ylab)
    if (xlab == ylab) {
        xlab <- paste0("x = ", xlab)
        ylab <- paste0("y = ", ylab)
    }

    ex <- ecdf(x)
    sy <- sort(unique(c(x, y)))
    py <- ecdf(y)(sy)
    px <- ex(sy)
    ret <- stepfun(px, c(0, py))
    if (!plot.it)
        return(ret)

    plot(ret, xlim = c(0, 1), ylim = c(0, 1),
         xlab = xlab, ylab = ylab, main = main,
         verticals = FALSE, ...)

    ⟨ ROC bands 79 ⟩

    plot(ret, add = TRUE, verticals = FALSE, ...)

    return(invisible(ret))
}
```

◇

Fragment referenced in 33.

A correct logistic model with log-odds ratio three is shown in Figure 10.4 and an incorrect proportional hazards model for the same data in Figure 10.5.

```
R> y <- rlogis(50)
R> x <- rlogis(50, location = 3)
R> ppplot(y, x, conf.level = .95, las = 1)
```

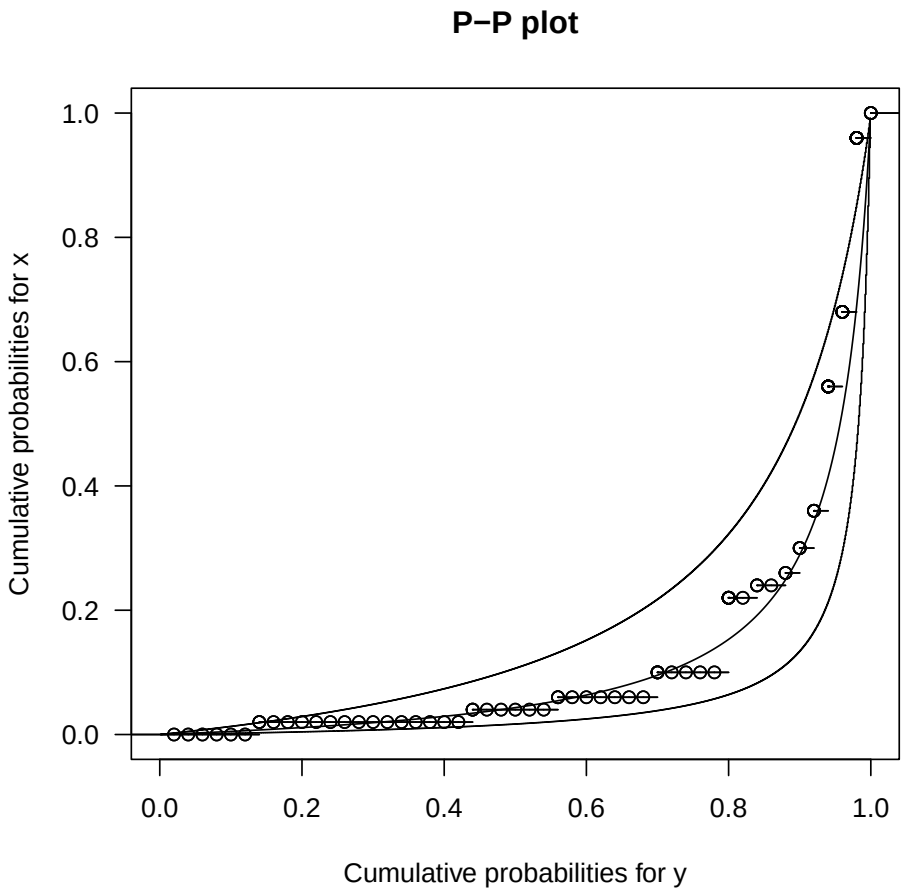


Figure 10.4: Data sampled from a proportional-odds model with probability-probability (P-P) curve and 95% confidence band obtained from a proportional-odds model.

```
R> ppplot(y, x, conf.args = list(link = "cloglog", type = "Wald",
+                                 col = NA, border = NULL),
+         conf.level = .95, las = 1)
```

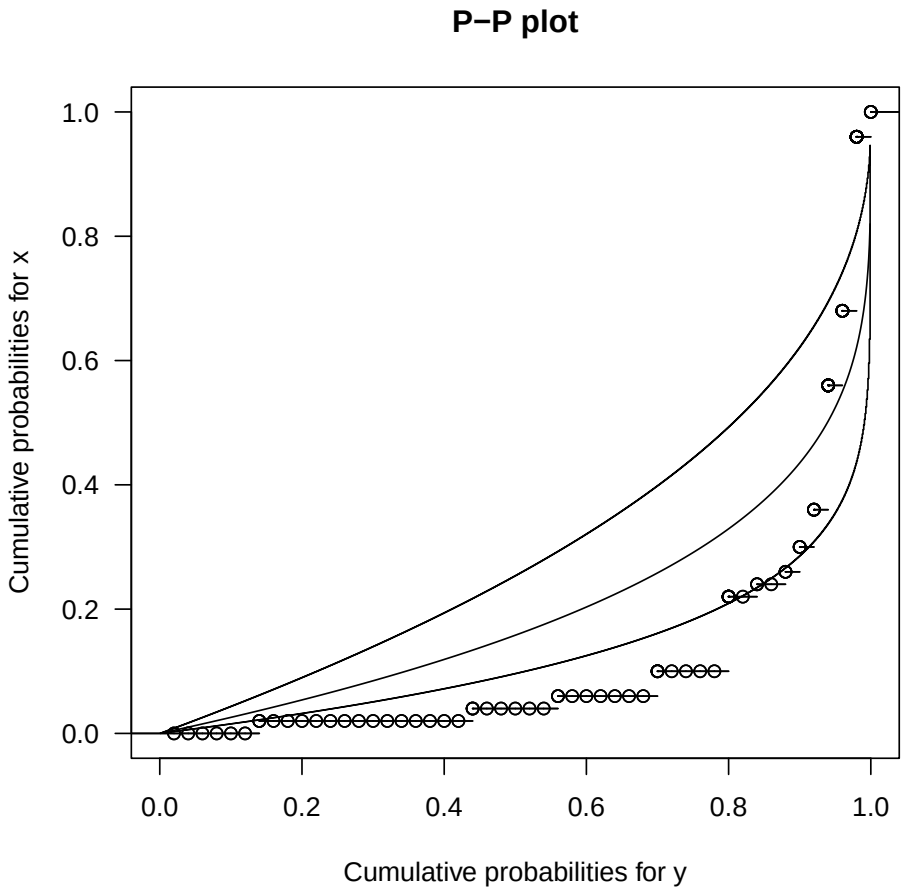


Figure 10.5: Data sampled from a proportional-odds model with probability-probability (P-P) curve and 95% confidence band obtained from a proportional-hazards model.

Chapter 11

Random Number Generation

With 1.1 we know that for an absolutely continuous random variable Y

$$U = F_Y(Y \mid G = k, S = b) = F\left(F^{-1}\left(F_Y(Y \mid G = 1, S = b)\right) - \delta_k\right), \quad k = 2, \ldots, K$$

follows a standard uniform distribution on the unit interval. This means that we can sample from the distribution of Y using

$$F_Y^{-1}\left(F(F^{-1}(U) + \delta_k) \mid G = 1, S = b\right).$$

It is therefore enough to draw samples from $F(F^{-1}(U) + \delta_k)$, that is, assuming a uniform distribution for F_Y in each control group. Because of the invariance with respect to monotone transformations, transforming all observations by the same quantile function changes the outcome distributions but not the shift effects. Discrete outcomes can be generated by post-hoc categorisation.

⟨ *design args* 81 ⟩ ≡

```
K <- length(delta) + 1L
if (is.null(names(delta)))
    names(delta) <- LETTERS[seq_len(K)[-1]]
if (length(alloc_ratio) == 1L)
    alloc_ratio <- rep_len(alloc_ratio, K - 1)
if (length(alloc_ratio) != K - 1L)
    stop(gettextf("invalid argument '%s'", "alloc_ratio"), domain = NA)
if (length(strata_ratio) == 1L)
    strata_ratio <- rep_len(strata_ratio, B - 1)
if (length(strata_ratio) != B - 1L)
    stop(gettextf("invalid argument '%s'", "strata_ratio"), domain = NA)
### sample size per group (columns) and stratum (rows)
N <- n * matrix(c(1, alloc_ratio), nrow = B, ncol = K, byrow = TRUE) *
         matrix(c(1, strata_ratio), nrow = B, ncol = K)
```

◇

Fragment referenced in 82, 85.

⟨ *rfree1way* 82 ⟩ ≡

```
.rfree1way <- function(n, delta = 0, link = c("logit", "probit",
                                              "cloglog", "loglog"))
{

    logU <- log(ret <- runif(n))

    ⟨ link2fun 53 ⟩

    trt <- (abs(delta) > 0)
    ret[trt] <- .p(link, .q(link, logU[trt], log.p = TRUE) + delta[trt])

    return(ret)
}

rfree1way <- function(n, prob = NULL, alloc_ratio = 1,
                      blocks = ifelse(is.null(prob), 1, NCOL(prob)),
                      strata_ratio = 1, delta = 0, offset = 0,
                      link = c("logit", "probit", "cloglog", "loglog"))
{

    B <- blocks

    ⟨ design args 81 ⟩

    rownames(N) <- paste0("block", seq_len(B))
    ctrl <- "Control"
    colnames(N) <- c(ctrl, names(delta))

    trt <- gl(K, 1, labels = colnames(N))
    blk <- gl(B, 1, labels = rownames(N))
    ret <- expand.grid(blocks = blk, groups = trt)
    if (B == 1L) ret$blocks <- NULL
    ret <- ret[rep(seq_len(nrow(ret)), times = N), , drop = FALSE]

    if (length(offset) <= K)
        offset <- rep_len(offset, K)[ret$groups]
    if (length(offset) != nrow(ret))
        stop(gettextf("invalid argument '%s'", "offset"), domain = NA)
    ret$y <- .rfree1way(nrow(ret),
                        delta = offset + c(0, delta)[ret$groups],
                        link = link)
    if (is.null(prob)) return(ret)

    ### return discrete distribution
    if (!is.matrix(prob))
        prob <- matrix(prob, nrow = NROW(prob), ncol = B)
    if (ncol(prob) != B)
        stop(gettextf("incorrect number of columns for 'prob' in %s",
                      "rfree1way"),
             domain = NA)
    prob <- prop.table(prob, margin = 2L)
    ret <- do.call("rbind", lapply(1:ncol(prob), function(b) {
        if (B > 1)
            ret <- subset(ret, blocks == levels(blocks)[b])
        ret$y <- cut(ret$y, breaks = c(-Inf, cumsum(prob[,b])),
                     labels = paste0("Y", 1:nrow(prob)), ordered_result = TRUE)
        ret
    }))
    return(ret)
}
◇
```

Fragment referenced in 33.

```
R> (logOR <- c(log(1.5), log(2)))

[1] 0.40547 0.69315

R> nd <- rfree1way(150, delta = logOR)
R> coef(ft <- free1way(y ~ groups, data = nd))

groupsB groupsC
0.33158 0.75694

R> sqrt(diag(vcov(ft)))

groupsB groupsC
0.20303 0.20108

R> logLik(ft)

[1] -2742

R> nd$y <- qchisq(nd$y, df = 3)
R> coef(ft <- free1way(y ~ groups, data = nd))

groupsB groupsC
0.33158 0.75694

R> sqrt(diag(vcov(ft)))

groupsB groupsC
0.20303 0.20108

R> logLik(ft)

[1] -2742

R> N <- 25
R> pvals <- replicate(Nsim,
+  {
+    nd <- rfree1way(n = N, blocks = 2, delta = c(.25, .5), alloc_ratio = 2)
+    summary(free1way(y ~ groups | blocks, data = nd), test = "Permutation")$p.value
+  })
R> power.free1way.test(n = N, blocks = 2, delta = c(.25, .5), alloc_ratio = 2)

     Stratified 3-sample Kruskal-Wallis test against proportional odds alternatives

                n = 25
Total sample size = 50 (Control) + 100 (B) + 100 (C) = 250
            power = 0.31041
        sig.level = 0.05
   log-odds ratio = 0.25, 0.50

NOTE: 'n' is sample size in control group of first stratum

R> mean(pvals < .05)

[1] 0.31
```

This function can also be used to simulate survival times, for example, from a proportional hazards model with a censoring probability of .25 in the control arm and of .5 in the treated arm, under random censoring (that is, event and censoring times are independent given treatment).

```
R> N <- 1000
R> nd <- rfree1way(N, delta = 1, link = "cloglog")
R> nd$C <- rfree1way(n = N, delta = 1, offset = -c(qlogis(.25), qlogis(.5)),
+                    link = "cloglog")$y
R> nd$y <- Surv(pmin(nd$y, nd$C), nd$y < nd$C)
R> ### check censoring probability
R> 1 - tapply(nd$y[,2], nd$groups, mean)

Control       B
  0.261   0.486

R> summary(free1way(y ~ groups, data = nd, link = "cloglog"))

Call:
free1way.formula(formula = y ~ groups, data = nd, link = "cloglog")

Coefficients:
        log-hazard ratio Std. Error z value Pr(>|z|)
groupsB           0.9870     0.0597    16.5   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

R> summary(coxph(y ~ groups, data = nd))

Call:
coxph(formula = y ~ groups, data = nd)

  n= 2000, number of events= 1253

           coef exp(coef) se(coef)     z Pr(>|z|)
groupsB -0.9857    0.3732   0.0596 -16.5   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

        exp(coef) exp(-coef) lower .95 upper .95
groupsB     0.373       2.68     0.332     0.419

Concordance= 0.621  (se = 0.007 )
Likelihood ratio test= 281  on 1 df,   p=<2e-16
Wald test            = 273  on 1 df,   p=<2e-16
Score (logrank) test = 293  on 1 df,   p=<2e-16
```

Next we start implementing a function for simulating $C \times K$ tables. We need to specify the number of observations in each treatment group (`c`), the discrete distribution of the control (`r`), a model (`link`), and a treatment effect (`delta`, in line with `power.XYZ.test`). In essence, we draw samples from the multinomial distribution after computing the relevant discrete density.

⟨ *r2dsim* 83 ⟩ ≡

```
.r2dsim <- function(n, r, c, delta = 0,
                    link = c("logit", "probit", "cloglog", "loglog"))
{

    if (length(n <- as.integer(n)) == 0L || (n < 0) || is.na(n))
        stop(gettextf("invalid argument '%s'", "n"), domain = NA)
    colsums <- c
    if (length(colsums[] <- as.integer(c)) <= 1L ||
        any(colsums < 0) || anyNA(colsums))
        stop(gettextf("invalid argument '%s'", "c"), domain = NA)

    prob <- r
    if (length(prob[] <- as.double(r / sum(r))) <= 1L ||
        any(prob < 0) || anyNA(prob))
        stop(gettextf("invalid argument '%s'", "r"), domain = NA)

    if (is.null(names(prob)))
        names(prob) <- paste0("i", seq_along(prob))

    K <- length(colsums)
    if (is.null(names(colsums)))
        names(colsums) <- LETTERS[seq_len(K)]
    delta <- rep_len(delta, K - 1L)

    ⟨ link2fun 53 ⟩

    p0 <- cumsum(prob) / sum(prob)
    h0 <- .q(link, p0[-length(p0)]) ### last element of p0 is one

    h1 <- h0 - matrix(delta, nrow = length(prob) - 1L, ncol = K - 1,
                      byrow = TRUE)
    p1 <- rbind(.p(link, h1), 1)
    p <- cbind(p0, p1)
    ret <- vector(mode = "list", length = n)

    for (i in seq_len(n)) {
        tab <- sapply(seq_len(K), function(k) {
            ### deal with zero prob categories,
            ### b/c cut() below will complain if p[,k] has duplicates
            ndp <- which(!duplicated(p[,k]))
            pu <- sort(p[ndp,k])
            ret <- integer(length(prob))
            ret[ndp] <- unclass(table(cut(runif(colsums[k]), breaks = c(-Inf, pu))))
            ret
        })
        ret[[i]] <- as.table(array(unlist(tab), dim = c(length(prob), K),
                                   dimnames = list(names(prob),
                                                   names(colsums))))
    }
    return(ret)
}
```

◇

Fragment referenced in 91.

Chapter 12

Power and Sample Size

The term "distribution-free" refers to the invariance of the reference distribution with respect to the distribution of an absolutely continuous outcome under control. Unfortunately, this is no longer true for non-continuous outcomes (due to ties) and under the alternative. That means that sample size assessments always take place under certain assumptions regarding the outcome distribution.

With the infrastructure from Chapter 11, we are now ready to put together a function for power evaluation and sample size assessment. The core idea is to draw samples from the relevant data (under a specific model in the alternative) and to estimate the Fisher information of the treatment effect parameters for this configuration. The power of the global Wald test can than be approximated by a non-central χ^2 distribution. This is much faster than approximating the power directly. Nevertheless, this is a random experiment, so we first make computations reproducible:

⟨ *random seed* 84 ⟩ ≡

```
if (!exists(".Random.seed", envir = .GlobalEnv, inherits = FALSE))
    runif(1)
if (is.null(seed))
    seed <- RNGstate <- get(".Random.seed", envir = .GlobalEnv)
else {
    R.seed <- get(".Random.seed", envir = .GlobalEnv)
    set.seed(seed)
    RNGstate <- structure(seed, kind = as.list(RNGkind()))
    on.exit(assign(".Random.seed", R.seed, envir = .GlobalEnv))
}
```

◇

Fragment referenced in 91.

⟨ *power setup* 85 ⟩ ≡

```
⟨ link2fun 53 ⟩

### matrix means control distributions in different strata
if (!is.matrix(prob))
    prob <- matrix(prob, nrow = NROW(prob), ncol = blocks)
prob <- prop.table(prob, margin = 2L)
C <- nrow(prob)
B <- ncol(prob)
if (is.null(colnames(prob)))
    colnames(prob) <- paste0("stratum", seq_len(B))
p0 <- apply(prob, 2, function(x) cumsum(x) / sum(x))
h0 <- .q(link, p0[-nrow(p0),,drop = FALSE])

⟨ design args 81 ⟩

rownames(N) <- colnames(prob)
ctrl <- "Control"
dn <- dimnames(prob)
if (!is.null(names(dn)[1L]))
    ctrl <- names(dn)[1L]
colnames(N) <- c(ctrl, names(delta))
◇
```

Fragment referenced in 91.

For estimating the Fisher information, we draw samples from the discrete outcome distribution and evaluate the observed Fisher information for the, here and now known true parameters. The average of these Fisher information matrices is then used as an estimate for the expected Fisher information. For small sample sizes less than 100, we draw larger samples (at least 1000) and adjust the obtained Fisher information accordingly to reduce sampling error.

⟨ *estimate Fisher information* 86 ⟩ ≡

```
he <- 0
deltamu <- delta - mu
Nboost <- ifelse(n < 100, ceiling(1000 / n), 1)
for (i in seq_len(nsim)) {
    parm <- deltamu
    x <- as.table(array(0, dim = c(C, K, B)))
    for (b in seq_len(B)) {
        x[,,b] <- .r2dsim(1L, r = prob[, b], c = Nboost * N[b,],
                          delta = delta, link = link)[[1L]]
        rs <- which(.rowSums(x[,,b], m = dim(x)[1L], n = dim(x)[2L]) > 0)
        theta <- h0[pmin(nrow(h0), rs), b]
        parm <- c(parm, theta[-length(theta)])
    }
    ### evaluate observed hessian for true parameters parm and x data
    he <- he + .free1wayML(x, link = link, mu = mu, start = parm,
                           fix = seq_along(parm))$hessian / Nboost
}
### estimate expected Fisher information
he <- he / nsim
◇
```

Fragment referenced in 91.

The power function now depends on sample size (`n`; the number of control observations in the first stratum), a discrete control distribution (`prob`, this can be a $C \times B$ matrix for stratum-specific control

distributions), a vector of allocation ratios (`alloc_ratio = 2` means control:treatment = 1:2) and the sample size ratios between strata.

The treatment effects are contained in $K - 1$ vector `delta`:

⟨ *power call* 87 ⟩ ≡

```
power.free1way.test(n = n, prob = prob,
                    alloc_ratio = alloc_ratio,
                    blocks = blocks,
                    strata_ratio = strata_ratio,
                    delta = delta, mu = mu,
                    sig.level = sig.level, link = link,
                    alternative = alternative,
                    nsim = nsim, seed = seed,
                    tol = tol)$power - power
```

◇

Fragment referenced in 90.

⟨ *power args check* 88 ⟩ ≡

```
if (sum(vapply(list(n, delta, power, sig.level), is.null,
    NA)) != 1)
    stop("exactly one of 'n', 'delta', 'power', and 'sig.level' must be NULL")
assert_NULL_or_prob(sig.level)
assert_NULL_or_prob(power)
```

◇

Fragment referenced in 91.

⟨ *power htest output* 89 ⟩ ≡

```
ss <- paste(colSums(N), paste0("(", colnames(N), ")"), collapse = " + ")
ret <- list(n = n,
            "Total sample size" = paste(ss, "=", sum(N)),
            power = power,
            sig.level = sig.level)
if (mu != 0) ret$mu <- mu
if (K == 2L) ret[["Standard error"]] <- se
ret[[link$parm]] <- delta
ret$note <- "'n' is sample size in control group"
if (B > 1) ret$note <- paste(ret$note, "of first stratum")
alias <- link$alias
if (length(link$alias) == 2L) alias <- alias[1L + (K > 2L)]
ret$method <- paste(ifelse(B > 1L, "Stratified", ""),
                    paste0(K, "-sample"), alias,
                    "test against", link$model, "alternatives")
class(ret) <- "power.htest"
```

◇

Fragment referenced in 91.

We can invert the power function for finding nominal levels, sample or effect sizes necessary to achieve a certain power. The option is available because the power approximation is relatively fast.

⟨ *power inversion* 90 ⟩ ≡

```
if (is.null(n))
    n <- ceiling(uniroot(function(n) {
             ⟨ power call 87 ⟩
         }, interval = c(5, 1e+03), tol = tol, extendInt = "upX")$root)
else if (is.null(delta)) {
    ### 2-sample only
    if (length(alloc_ratio) > 1L)
        stop(gettextf("effect size can only be computed for two-sample problems in %s",
                      "power.free1way.test"),
             domain = NA)
    delta <- uniroot(function(delta) {
             ⟨ power call 87 ⟩
### <TH> interval depending on alternative, symmetry? </TH>
        }, interval = c(0, 10), tol = tol, extendInt = "upX")$root
    }
else if (is.null(sig.level))
    sig.level <- uniroot(function(sig.level) {
            ⟨ power call 87 ⟩
        }, interval = c(1e-10, 1 - 1e-10), tol = tol, extendInt = "yes")$root
```

◇

Fragment referenced in 91.

⟨ *power* 91 ⟩ ≡

```
power.free1way.test <- function(n = NULL,
                                prob = if (is.null(n)) NULL else
                                                       rep.int(1 / n, n),
                                alloc_ratio = 1,
                                blocks = if (is.null(prob)) 1 else NCOL(prob),
                                strata_ratio = 1,
                                delta = NULL, mu = 0,
                                sig.level = .05, power = NULL,
                                link = c("logit", "probit", "cloglog", "loglog"),
                                alternative = c("two.sided", "less", "greater"),
                                nsim = 100, seed = NULL,
                                tol = .Machine$double.eps^0.25)
{

    ⟨ power args check 88 ⟩

    ⟨ random seed 84 ⟩

    ⟨ r2dsim 83 ⟩

    ⟨ power inversion 90 ⟩

    ### n is available now
    if (is.null(prob)) prob <- rep(1 / n, n)

    ⟨ power setup 85 ⟩
    ⟨ estimate Fisher information 86 ⟩

    alternative <- match.arg(alternative)
    if (K == 2L) {
        se <- 1 / sqrt(c(he))
        power  <- switch(alternative,
            "two.sided" = pnorm(qnorm(sig.level / 2) + deltamu / se) +
                          pnorm(qnorm(sig.level / 2) - deltamu / se),
            "less" = pnorm(qnorm(sig.level) - deltamu / se),
            "greater" = pnorm(qnorm(sig.level) + deltamu / se)
        )
    } else {
        if (alternative != "two.sided")
            stop(gettextf("%s only allows two-sided alternatives in the presence of more than two groups",
                          "power.free1way.test"),
                 domain = NA)
        ncp <- sum((chol(he) %*% deltamu)^2)
        qsig <- qchisq(sig.level, df = K - 1L, lower.tail = FALSE)
        power <- pchisq(qsig, df = K - 1L, ncp = ncp, lower.tail = FALSE)
    }

    ⟨ power htest output 89 ⟩

    ret
}
```

◇

Fragment referenced in 33.

We start with the power of a binomial experiment with $N = 2 \times 25$ observations. In the control group, the odds of winning is 1. Under treatment, we increase this odds by 50%. We compare the results with `power.prop.test`:

```
R> delta <- log(1.5)
R> power.prop.test(n = 25, p1 = .5, p2 = plogis(qlogis(.5) - delta))
```

```
     Two-sample comparison of proportions power calculation

              n = 25
             p1 = 0.5
             p2 = 0.4
      sig.level = 0.05
          power = 0.10462
    alternative = two.sided

NOTE: n is number in *each* group
```

```
R> power.free1way.test(n = 25, prob = c(.5, .5), delta = delta)

      2-sample Wilcoxon test against proportional odds alternatives

                n = 25
Total sample size = 25 (Control) + 25 (B) = 50
            power = 0.10935
        sig.level = 0.05
   Standard error = 0.57155
   log-odds ratio = 0.40547

NOTE: 'n' is sample size in control group
```

Under stratification (twice as many observations in the second stratum) and with an ordered outcome at four levels, we might want to compare four groups, with 25%, 50%, and 75% increase compared to the odds of the control:

```
R> prb <- matrix(c(.25, .25, .25, .25,
+                  .10, .20, .30, .40), ncol = 2)
R> colnames(prb) <- c("s1", "s2")
R> power.free1way.test(n = 20, prob = prb,
+                      strata_ratio = 2,
+                      alloc_ratio = c(1.5, 2, 2),
+                      delta = log(c("low" = 1.25, "med" = 1.5, "high" = 1.75)))

     Stratified 4-sample Kruskal-Wallis test against proportional odds alternatives

                n = 20
Total sample size = 60 (Control) + 90 (low) + 120 (med) + 120 (high) = 390
            power = 0.37907
        sig.level = 0.05
   log-odds ratio = 0.22314, 0.40547, 0.55962

NOTE: 'n' is sample size in control group of first stratum
```

We now estimate the power of a Wilcoxon test with, first by simulation from a logistic distribution, and then by our power function:

```
R> delta <- log(3)
R> N <- 15
R> w <- gl(2, N)
R> pw <- numeric(Nsim)
R> for (i in seq_along(pw)) {
+      y <- rlogis(length(w), location = c(0, delta)[w])
+      pw[i] <- wilcox.test(y ~ w)$p.value
+  }
R> mean(pw < .05)

[1] 0.44
```

```
R> power.free1way.test(n = N, delta = delta)

      2-sample Wilcoxon test against proportional odds alternatives

               n = 15
Total sample size = 15 (Control) + 15 (B) = 30
           power = 0.38383
         sig.level = 0.05
   Standard error = 0.66017
   log-odds ratio = 1.0986

NOTE: 'n' is sample size in control group

R> ### approximate formula in Hmisc::popower
R> library("Hmisc")
R> popower(p = rep(1 / N, N), odds.ratio = exp(delta), n = 2 * N)

Power: 0.389
Efficiency of design compared with continuous response: 0.997
Approximate standard error of log odds ratio: 0.655
```

The power of the Kruskal-Wallis test only needs one additional treatment effect

```
R> delta <- c("B" = log(2), "C" = log(3))
R> N <- 15
R> w <- gl(3, N)
R> pw <- numeric(Nsim)
R> for (i in seq_along(pw)) {
+      y <- rlogis(length(w), location = c(0, delta)[w])
+      pw[i] <- kruskal.test(y ~ w)$p.value
+  }
R> mean(pw < .05)

[1] 0.29

R> power.free1way.test(n = N, delta = delta)

      3-sample Kruskal-Wallis test against proportional odds alternatives

               n = 15
Total sample size = 15 (Control) + 15 (B) + 15 (C) = 45
           power = 0.31235
         sig.level = 0.05
   log-odds ratio = 0.69315, 1.09861

NOTE: 'n' is sample size in control group
```

We next use the `rfree1way` function to sample from 4×3 tables with odds ratios 2 and 3 and compare the resulting power with result obtained from the approximated Fisher information. By default, the continuous control distribution is uniform on the unit interval, thus `cut` with breaks defined by the target control discrete probability distribution generates the outcome. The plot shows the distribution of the parameter estimates and the corresponding population values as red dots (Figure 12.1).

In the last example, we sample from 4×3 tables with odds ratios 2 and 3 for three strata with different control distributions, see Figure 12.2, and again compare the simulation results to the power function.

```
R> power.free1way.test(n = N, prob = prb, delta = delta, seed = 3)

     Stratified 3-sample Kruskal-Wallis test against proportional odds alternatives

               n = 15
Total sample size = 45 (Ctrl) + 45 (B) + 45 (C) = 135
```

```
R> prb <- rep.int(1, 4) / 4
R> pw <- numeric(Nsim)
R> cf <- matrix(0, nrow = Nsim, ncol = length(delta))
R> colnames(cf) <- names(delta)
R> for (i in seq_along(pw)) {
+      nd <- rfree1way(n = N, prob = prb, delta = delta)
+      ft <- free1way(y ~ groups, data = nd)
+      cf[i,] <- coef(ft)
+      pw[i] <- summary(ft, test = "Permutation")$p.value
+  }
R> mean(pw < .05)

[1] 0.29

R> boxplot(cf, las = 1, ylab = expression(hat(delta)))
R> points(c(1:2), delta, pch = 19, col = "red")
R> power.free1way.test(n = N, prob = prb, delta = delta)

      3-sample Kruskal-Wallis test against proportional odds alternatives

                n = 15
Total sample size = 15 (Control) + 15 (B) + 15 (C) = 45
            power = 0.29154
        sig.level = 0.05
   log-odds ratio = 0.69315, 1.09861

NOTE: 'n' is sample size in control group
```

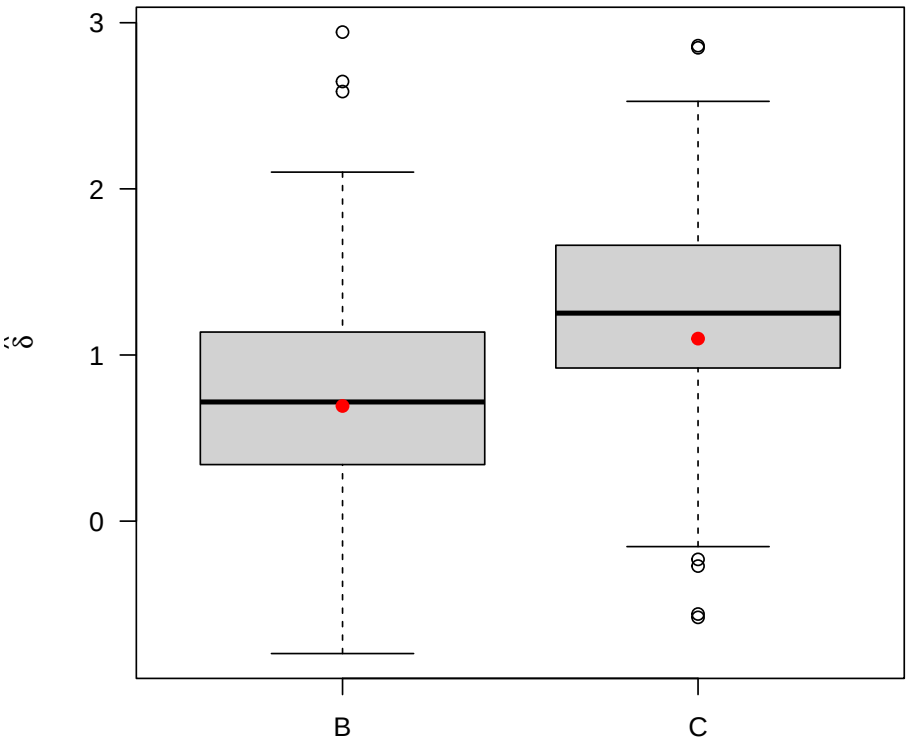


Figure 12.1: Power simulation for proportional-odds model and corresponding power approximation.

```
R> prb <- cbind(S1 = rep(1, 4),
+               S2 = c(1, 2, 1, 2),
+               S3 = 1:4)
R> dimnames(prb) <- list(Ctrl = paste0("i", seq_len(nrow(prb))),
+                        Strata = colnames(prb))
R> pw <- numeric(Nsim)
R> cf <- matrix(0, nrow = Nsim, ncol = length(delta))
R> colnames(cf) <- names(delta)
R> for (i in seq_along(pw)) {
+      nd <- rfree1way(n = N, prob = prb, delta = delta)
+      ft <- free1way(y ~ groups | blocks, data = nd)
+      cf[i,] <- coef(ft)
+      pw[i] <- summary(ft, test = "Permutation")$p.value
+  }
R> mean(pw < .05)

[1] 0.73

R> boxplot(cf, las = 1, ylab = expression(hat(delta)))
R> points(c(1:2), delta, pch = 19, col = "red")
```

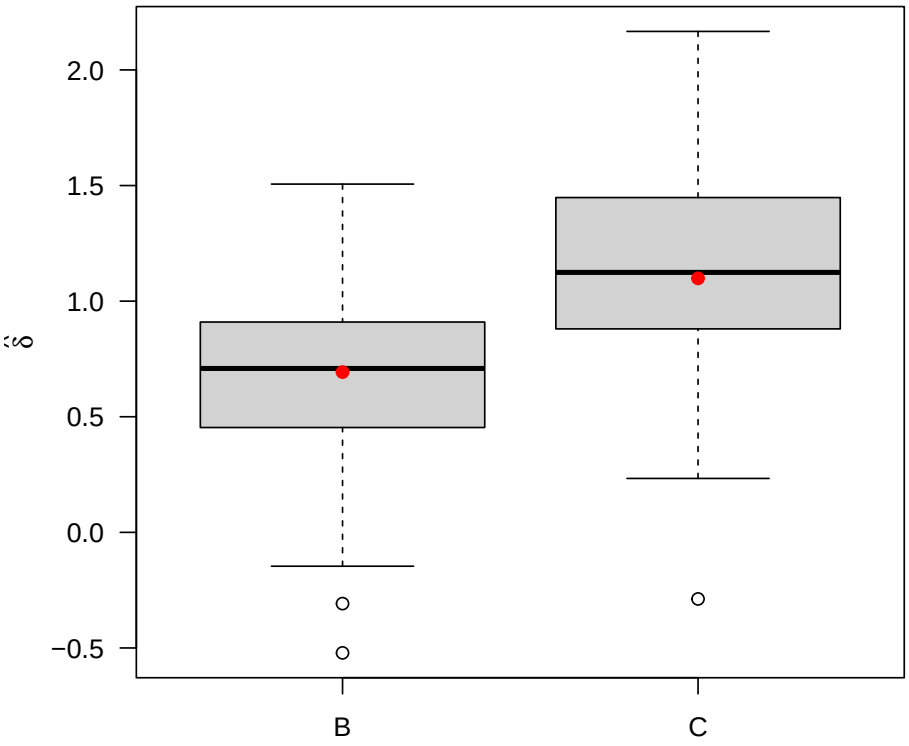


Figure 12.2: Power simulation for stratified proportional-odds model and corresponding power approximation.

```
           power = 0.69905
       sig.level = 0.05
  log-odds ratio = 0.69315, 1.09861

NOTE: 'n' is sample size in control group of first stratum

R> power.free1way.test(power = .8, prob = prb, delta = delta, seed = 3)

     Stratified 3-sample Kruskal-Wallis test against proportional odds alternatives

               n = 19
Total sample size = 57 (Ctrl) + 57 (B) + 57 (C) = 171
           power = 0.80451
       sig.level = 0.05
  log-odds ratio = 0.69315, 1.09861

NOTE: 'n' is sample size in control group of first stratum

R> power.free1way.test(n = 19, prob = prb, delta = delta, seed = 3)

     Stratified 3-sample Kruskal-Wallis test against proportional odds alternatives

               n = 19
Total sample size = 57 (Ctrl) + 57 (B) + 57 (C) = 171
           power = 0.80451
       sig.level = 0.05
  log-odds ratio = 0.69315, 1.09861

NOTE: 'n' is sample size in control group of first stratum
```

Chapter 13

Penalisation

Sometimes, especially under complete separation or when many blocks only contain small sample sizes, the maximum likelihood estimator does not exist or is biased. We could think of offering the option to add a penalty term to the log-likelihood, for example half of the log-determinant of the Hessian (Jeffreys prior) as suggested by Firth (1993) and studied in Kosmidis and Firth (2020). The `MPL_Jeffreys` argument can be used to request this type of penalisation from `free1way`. Currently, we only offer Wald-type inference procedures based on the observed Fisher information evaluated at the penalised parameter estimates, so we disable all other methods

⟨ *MPLfree1way methods* 92 ⟩ ≡

```
confint.MPLfree1way <- function(object, parm,
    level = .95, test = "Wald",
    what = c("shift", "PI", "AUC", "OVL"), ...) {

    test <- match.arg(test)
    return(confint.free1way(object, parm = parm, level = level,
                            test = test, what = what, ...))
}

summary.MPLfree1way <- function(object, test = "Wald",
                                alternative = c("two.sided", "less", "greater"),
                                tol = .Machine$double.eps, ...) {

    test <- match.arg(test)
    return(summary.free1way(object, test = test,
                            alternative = alternative, tol = tol, ...))
}

print.MPLfree1way <- function(x, test = "Wald", ...) {

    test <- match.arg(test)
    return(invisible(print.free1way(x, test = test, ...)))
}
◇
```

Fragment referenced in 33.

Note that penalised estimation is currently slow because neither the score nor the Fisher information corresponding to the penalised log-likelihood have been implemented.

As an example, consider the rabbits data from the manual page of the Mantel-Haenszel test. We first perform the test, starting with the classical implementation and then using `free1way`

```
R> example(mantelhaen.test, echo = FALSE)
R> ### "Response" is the response and comes first
R> Rabbits <- aperm(as.table(Rabbits), c(2, 1, 3))
R> ### identical p-values
R> (mh <- mantelhaen.test(Rabbits, correct = FALSE))
```

```
        Mantel-Haenszel chi-squared test without continuity correction

data:  Rabbits
Mantel-Haenszel X-squared = 5.66, df = 1, p-value = 0.017
alternative hypothesis: true common odds ratio is not equal to 1
95 percent confidence interval:
  1.0267 47.7251
sample estimates:
common odds ratio
                7

R> (ft <- free1way(Rabbits))

        Stratified 2-sample Wilcoxon test against proportional odds
        alternatives

data:  Response by Delay (c("None", "1.5h"))
         stratified by Penicillin.Level
Perm Z = 2.38, p-value = 0.017
alternative hypothesis: true log-odds ratio is not equal to 0

R> log(mh$estimate)

common odds ratio
           1.9459

R> coef(ft)

Delay1.5h
   2.5496
```

The p-values are identical, but the parameter estimates are not. The reason is that, by default, maximum likelihood estimation is performed by `free1way`, which in case of small blocks is biased.

We now solve the identical estimation problem using a binary logistic regression

```
R> rab <- as.data.frame(Rabbits)
R> lr <- glm(Response ~ 0 + Penicillin.Level +
+                        Delay, data = rab, weights = rab$Freq,
+            family = binomial(link = "logit"))
R> logLik(ft)

[1] -14.429

R> logLik(lr)

'log Lik.' -14.429 (df=6)

R> coef(ft)

Delay1.5h
   2.5496

R> rev(coef(lr))[1L]

Delay1.5h
   2.5496

R> sqrt(vcov(ft))

          Delay1.5h
Delay1.5h    1.1752

R> sqrt(rev(diag(vcov(lr)))[1L])
```

```
Delay1.5h
   1.1752
```

where the log-likelihoods, parameter estimates, and standard errors are the same as those obtained from `free1way`. Using package **brglm2**, we can perform Firth method of bias reduction

```
R> library("brglm2")
R> Flr <- glm(Response ~ 0 + Penicillin.Level + Delay, data = rab, weights = rab$Freq,
+             family = binomial(link = "logit"), method = "brglmFit", type = "MPL_Jeffreys")
```

and also request the same form of penalisation in `free1way`

```
R> ftJ <- free1way(Rabbits, MPL_Jeffreys = TRUE)
```

where the results are practically identical and much closer to the results reported by `mantelhaen.test`

```
R> logLik(ftJ)

[1] -15.583

R> logLik(Flr)

'log Lik.' -15.55 (df=6)

R> log(mh$estimate)

common odds ratio
           1.9459

R> coef(ftJ)

Delay1.5h
   2.0371

R> rev(coef(Flr))[1L]

Delay1.5h
   2.0371

R> diag(vcov(ftJ))

Delay1.5h
  0.91962

R> rev(diag(vcov(Flr)))[1L]

Delay1.5h
  0.91963

R> log(mh$conf.int)

[1] 0.026362 3.865458
attr(,"conf.level")
[1] 0.95

R> confint(ftJ)

            2.5 % 97.5 %
Delay1.5h 0.15755 3.9166

R> confint(Flr)[names(rev(coef(Flr))[1L]),]

  2.5 %  97.5 %
0.15756 3.91666
```

Chapter 14

Acknowledgements

We would like to thank Frank Harrell, Michael Fay, Bryan Shepherd, and Ioannis Kosmidis for insights and valuable discussions.

Index

Files

"free1way.R" Defined by 33.
"linkfun.R" Defined by 22.
"scaleTest.R" Defined by 69.
"utils.R" Defined by 32.

Fragments

⟨ add legend 77 ⟩ Referenced in 78.
⟨ cloglog 26 ⟩ Referenced in 22.
⟨ confint lower 65 ⟩ Referenced in 68.
⟨ confint upper 66 ⟩ Referenced in 68.
⟨ convert to three-way table 42 ⟩ Referenced in 43.
⟨ density prob ratio 4 ⟩ Referenced in 5, 6, 7.
⟨ design args 81 ⟩ Referenced in 82, 85.
⟨ determine steps in blocks 13 ⟩ Referenced in 14.
⟨ diagonal elements for Hessian of intercepts 10 ⟩ Referenced in 12.
⟨ do optim 35 ⟩ Referenced in 37, 39.
⟨ estimate Fisher information 86 ⟩ Referenced in 91.
⟨ exact proportional odds 52 ⟩ Referenced in 55.
⟨ extract plot data 71 ⟩ Referenced in 78.
⟨ formula business 56 ⟩ Referenced in 57.
⟨ free1way confint 68 ⟩ Referenced in 33.
⟨ free1way factor 60 ⟩ Referenced in 33.
⟨ free1way formula 57 ⟩ Referenced in 33.
⟨ free1way generic and table method (main workhorse) 54 ⟩ Referenced in 33.
⟨ free1way methods 61 ⟩ Referenced in 33.
⟨ free1way numeric 59 ⟩ Referenced in 33.
⟨ free1way permutation tests 55 ⟩ Referenced in 54.
⟨ free1way print 62 ⟩ Referenced in 33.
⟨ free1way summary 63 ⟩ Referenced in 33.
⟨ full Hessian 20 ⟩ Referenced in 21.
⟨ Hessian 12 ⟩ Referenced in 43.
⟨ Hessian prep 8 ⟩ Referenced in 12.
⟨ intercept / shift contributions to Hessian 11 ⟩ Referenced in 12.
⟨ link2fun 53 ⟩ Referenced in 54, 82, 83, 85.
⟨ linkfun 23 ⟩ Referenced in 22.
⟨ logdet 38 ⟩ Referenced in 43.
⟨ logit 24 ⟩ Referenced in 22.
⟨ logLik, gradient, Hessian 36 ⟩ Referenced in 37.
⟨ loglog 25 ⟩ Referenced in 22.
⟨ LRT 46 ⟩ Referenced in 44.
⟨ marginal fit 73 ⟩ Referenced in 78.
⟨ marginal plot 75 ⟩ Referenced in 78.
⟨ ML estimation 43 ⟩ Referenced in 33.
⟨ model plot 76 ⟩ Referenced in 78.
⟨ MPLfree1way methods 92 ⟩ Referenced in 33.
⟨ negative logLik 3 ⟩ Referenced in 43.
⟨ negative scale score residuals 7 ⟩ Referenced in 43.

⟨ negative score 5 ⟩ Referenced in 43.
⟨ negative score residuals 6 ⟩ Referenced in 43.
⟨ Newton ? ⟩ Not referenced.
⟨ Newton convergence 30 ⟩ Referenced in 31.
⟨ Newton step halving 29 ⟩ Referenced in 31.
⟨ Newton update 28 ⟩ Referenced in 31.
⟨ NewtonRaphson 31 ⟩ Referenced in 33.
⟨ off-diagonal elements for Hessian of intercepts 9 ⟩ Referenced in 12.
⟨ optim 39 ⟩ Referenced in 43.
⟨ parm to prob 2 ⟩ Referenced in 3, 5, 6, 7, 12.
⟨ permutation confint 64 ⟩ Referenced in 68.
⟨ Permutation p-values 49 ⟩ Referenced in 62, 70.
⟨ Permutation statistics 48 ⟩ Referenced in 44.
⟨ PI OVL confint 67 ⟩ Referenced in 68.
⟨ plot free1way 78 ⟩ Referenced in 33.
⟨ post processing 40 ⟩ Referenced in 43.
⟨ power 91 ⟩ Referenced in 33.
⟨ power args check 88 ⟩ Referenced in 91.
⟨ power call 87 ⟩ Referenced in 90.
⟨ power htest output 89 ⟩ Referenced in 91.
⟨ power inversion 90 ⟩ Referenced in 91.
⟨ power setup 85 ⟩ Referenced in 91.
⟨ ppplot 80 ⟩ Referenced in 33.
⟨ probit 27 ⟩ Referenced in 22.
⟨ profile 37 ⟩ Referenced in 43.
⟨ profile function 41 ⟩ Referenced in 43.
⟨ r2dsim 83 ⟩ Referenced in 91.
⟨ random seed 84 ⟩ Referenced in 91.
⟨ Rao 47 ⟩ Referenced in 44.
⟨ refit block intercepts 72 ⟩ Referenced in 78.
⟨ resampling 51 ⟩ Referenced in 55, 70.
⟨ rfree1way 82 ⟩ Referenced in 33.
⟨ ROC bands 79 ⟩ Referenced in 80.
⟨ scale tests 70 ⟩ Referenced in 69.
⟨ setup and starting values 34 ⟩ Referenced in 43.
⟨ setup canvas 74 ⟩ Referenced in 78.
⟨ short-cuts for link functions 1 ⟩ Referenced in 43.
⟨ statistics 44 ⟩ Referenced in 62, 68.
⟨ Strasser Weber 50 ⟩ Referenced in 55, 70.
⟨ stratified Hessian 21 ⟩ Referenced in 43.
⟨ stratified negative logLik 16 ⟩ Referenced in 43.
⟨ stratified negative scale score residual 19 ⟩ Referenced in 43.
⟨ stratified negative score 17 ⟩ Referenced in 43.
⟨ stratified negative score residual 18 ⟩ Referenced in 43.
⟨ stratum prep 15 ⟩ Referenced in 16, 17, 18, 19, 21, 40.
⟨ table2list body 14 ⟩ Referenced in 34.
⟨ variable names and checks 58 ⟩ Referenced in 59, 60.
⟨ Wald statistic 45 ⟩ Referenced in 44.

Identifiers

Bibliography

David Firth. Bias reduction of maximum likelihood estimates. *Biometrika*, 80(1):27–38, 1993. doi: 10.1093/biomet/80.1.27. 31, 32, 111

Frank E Harrell, Jr. Ordinal logistic regression. In *Regression Modeling Strategies*, pages 311–325. Springer, 2015. doi: 10.1007/978-3-319-19425-7_13. 1

Frank E Harrell, Jr. Statistical computing approaches to maximum likelihood estimation. Technical report, Blog Post, 2024. URL `https://www.fharrell.com/post/mle/`. 2, 24, 35

Torsten Hothorn, Lisa Möst, and Peter Bühlmann. Most likely transformations. *Scandinavian Journal of Statistics*, 45(1):110–134, 2018. doi: 10.1111/sjos.12291. 4, 5

Ioannis Kosmidis and David Firth. Jeffreys-prior penalty, finiteness and shrinkage in binomial-response generalized linear models. *Biometrika*, 108(1):71–82, 08 2020. doi: 10.1093/biomet/asaa052. 111

Lukas Meier. *ANOVA and Mixed Models: A Short Introduction Using R*. Chapman and Hall/CRC, New York, U.S.A., 2022. doi: 10.1201/9781003146216. 83

Ainesh Sewak and Torsten Hothorn. Estimating transformations for evaluating diagnostic tests with covariate adjustment. *Statistical Methods in Medical Research*, 32(7):1403–1419, 2023. doi: 10.1177/09622802231176030. 92

Helmut Strasser and Christian Weber. On the asymptotic theory of permutation statistics. *Mathematical Methods of Statistics*, 8(2):220–250, 1999. 42

Aart W. Van der Vaart. *Asymptotic Statistics*. Cambridge University Press, Cambridge, UK, 1998. doi: 10.1017/CBO9780511802256. 2

M. B. Wilk and R. Gnanadesikan. Probability plotting methods for the analysis of data. *Biometrika*, 55 (1):1–17, 1968. doi: 10.1093/biomet/55.1.1. 92