# fairy: an overview of all functions

**Marc Jekel**

**2026-11-25**

**fairy** turns verbal hypotheses about proportions into order-constrained models and answers the questions the paper “Wait, what are you saying, exactly?” asks of them: What exactly does the hypothesis say (constraints)? Which extreme cases does it allow? How parsimonious is it? How does it relate to other models? This page introduces the idea and lists every function. The vignette `vignette("paper-examples")` then reproduces the paper’s examples.

# The idea

A hypothesis about `n` proportions `p1, ..., pn` is a set of linear (in)equalities. Together they describe a convex polytope inside the unit cube `[0, 1]^n`. The package describes that polytope in two equivalent ways:

- the **H-representation**: the shortest list of constraints (equalities and facet-defining inequalities);
- the **V-representation**: the extreme cases (vertices) of the model.

Its **volume** is the share of all conceivable proportions the model allows (the smaller, the more restrictive and more precise the model); its inverse is the largest Bayes factor the model can reach against the unconstrained model.

# Writing a model

A model is one character string. Constraints are separated by `;`.

| Write | Meaning |
|---|---|
| `p1 > p2` | `p1` at least `p2` (`>=` and the unicode symbol mean the same: models are closed sets) |
| `p1 > p2 > p3` | chain: `p1 > p2` and `p2 > p3` |
| `p1 = p2` | equality (the model then has volume zero) |
| `p1 > .3; p1 < .7` | bounds |
| `{p1, p2} > {p3, p4}` | every `p` on the left above every `p` on the right |
| `(p1 + p2)/2 > p3` | averages and other linear expressions |

`approx_tol = 0.05` (in `fairy_h()`) turns every equality into “within 0.05”.

# Constraints and extreme cases

```
h <- fairy_h("p1 > p2 > p3", n_p = 3)     # H-representation (rcdd format)
fairy_constraints(h)                       # the same, as readable text
#> [1] "- p1 + p2 <= 0" "- p2 + p3 <= 0" "p1 <= 1"        "- p3 <= 0"
```

```
fairy_vertices(h)                          # extreme cases, numbers
#>      p_{1} p_{2} p_{3}
#> [1,]     1     1     1
#> [2,]     1     1     0
#> [3,]     1     0     0
#> [4,]     0     0     0
fairy_words(h, labels = c("A", "B", "C"))  # extreme cases in words: "all" / "none"
#>       A    B    C
#> V1 none none none
#> V2  all none none
#> V3  all  all none
#> V4  all  all  all
```

## How to read the H- and V-description

Both describe the same set of allowed probability combinations (a polytope), from two sides:

- **H-description (constraints).** Each row is one constraint, and the model is the set of all combinations that satisfy *all* rows at once. `p2 <= p1` says that `p2` is at most `p1`; a row of type `equality` fixes a relation exactly. Constraints that are redundant are left out; the bounds 0 <= p <= 1 appear only where they restrict the model (e.g. `p1 <= 1`).
- **V-description (extreme cases).** Each row is a vertex: an allowed combination that cannot be written as a mixture of two other allowed combinations. Every allowed combination is a weighted mixture of the rows.

`fairy_h_table()` and `fairy_v_table()` put them into tables, and `fairy_table_write()` turns a table into Markdown, LaTeX or CSV:

```
fairy_h_table(h)
#>    Constraint       Type
#> C1   p2 <= p1 inequality
#> C2   p3 <= p2 inequality
#> C3    p1 <= 1 inequality
#> C4    0 <= p3 inequality
fairy_h_table(h, style = "paper")    # the form of the paper's Figure 2: - p1 + p2 <= 0
#>        Constraint       Type
#> C1 - p1 + p2 <= 0 inequality
#> C2 - p2 + p3 <= 0 inequality
#> C3        p1 <= 1 inequality
#> C4      - p3 <= 0 inequality
fairy_h_table(h, style = "matrix")   # a1*p1 + ... + an*pn <= b, one row per constraint
#>    p1 p2 p3 relation b
#> C1 -1  1  0       <= 0
#> C2  0 -1  1       <= 0
#> C3  1  0  0       <= 1
#> C4  0  0 -1       <= 0
fairy_v_table(h)
#>    p1 p2 p3
#> V1  0  0  0
#> V2  1  0  0
```

```
#> V3  1  1  0
#> V4  1  1  1
fairy_table_write(fairy_v_table(h, words = TRUE))
#> [1] "|    | p_.1. | p_.2. | p_.3. |" "| -- | ----- | ----- | ----- |"
#> [3] "| V1 |  none |  none |  none |" "| V2 |   all |  none |  none |"
#> [5] "| V3 |   all |   all |  none |" "| V4 |   all |   all |   all |"
```

`fairy_v(h)` returns the vertices in rcdd's own format. `fairy_dim()` gives the dimension of the model. Models with an equality have a dimension below `n` and volume zero:

```
c(fairy_dim("p1 > p2 > p3"), fairy_dim("p1 = p2; p2 > p3"))
#> [1] 3 2
```

# Starting from a model you already have

You never need to recompute a model. Every function accepts a model string, an H-representation from `fairy_h()`, **or** a V-representation (the vertices): pass `h` or `v` straight to `fairy_volume()`, `fairy_max_bf()`, `fairy_info()`, `fairy_nested()`, `fairy_compare()`, `fairy_replicate()` and so on.

```
v <- fairy_v(h)                 # V-representation in rcdd format
fairy_max_bf(v)                 # the same as fairy_max_bf(h)
#> [1] 6
```

If you only have your own table of vertices (one row per extreme case, probabilities as numbers or exact fractions such as `"1/3"`), `fairy_from_vertices()` turns it into the H-representation:

```
corners <- rbind(c(0, 0), c(1, 0), c(1, 1))      # a triangle
h_tri <- fairy_from_vertices(corners)
fairy_constraints(h_tri)
#> [1] "p1 <= 1"        "- p1 + p2 <= 0" "- p2 <= 0"
fairy_volume(h_tri)$volume
#> [1] 0.5
```

# Parsimony

```
fairy_volume("p1 > p2 > p3")$volume          # 1/6
#> [1] 0.1666667
fairy_max_bf("p1 > p2 > p3")                 # 6
#> [1] 6
fairy_info("p1 > p2 > p3")                   # dimension, vertices, constraints, volume, max.
        BF in one row
#>   dim vertices inequalities equalities    volume max_bf
#> 1   3        4            4          0 0.1666667      6
```

Outside this vignette, `fairy_volume()` by default estimates the volume with the volesti package and its own default settings; any non-default volesti option can be passed via `...` (e.g. `settings = list(error = 0.01)`).

`method = "exact"` (convex hull of the vertices) and `method = "mc"` (Monte Carlo) are available on request. `fairy_parsimony()` is the older name of the same computation.

### The same as in the app: exact first

The **Shiny app** calculates the exact volume first (it is a number such as 1/6, not an estimate) and uses an algorithm only for the models that have no exact volume. In the package this is `method = "auto"`:

```
v <- fairy::fairy_volume("p1 > p2 > p3", n_p = 3, method = "auto")   # the original function
        (this vignette fixes the default of fairy_volume() to "exact")
v$volume                                    # 1/6, exact
#> [1] 0.1666667
v$method                                    # "exact"
#> [1] "exact"
```

The exact volume is calculated however long it takes (up to about 10 dimensions it is usually quick, with 12 and more it can take minutes or longer). `time_limit = 300` (seconds) is an option: the exact calculation then runs in a separate R process (this needs the package callr) and is replaced by the volesti estimate after that time, and `$method` says `"volesti (the exact volume took too long)"`. The default of `fairy_volume()` itself is `"volesti"`, an estimate.

# Several models at once

`fairy()` computes constraints, extreme cases, volume and maximal Bayes factor for a named set of models:

```
fairy(c(H1 = "p1 > p2 > p3", H2 = "p1 > p2; p3 > p4"), n_p = 4)
#> $H1
#> fairy model: H1
#>   Model:        p1 > p2 > p3
#>   Parameters:   4
#>   H-rep rows:   6
#>   Vertices:     8
#>   Volume:       0.166667
#>   Max. BF:       6
#>
#> $H2
#> fairy model: H2
#>   Model:        p1 > p2; p3 > p4
#>   Parameters:   4
#>   H-rep rows:   6
#>   Vertices:     9
#>   Volume:       0.25
#>   Max. BF:       4
```

# Overlap, nesting, mixtures, comparison

```
a <- "p1 > p2 > p3"; b <- "p1 > p2"
fairy_nested(a, b, n_p = 3)                  # is a nested in b?  TRUE
#> [1] TRUE
ab <- fairy_intersect(a, b, n_p = 3)         # constraints of both hold
mx <- fairy_mixture("p1 > .8; p2 < .2", "p1 < .2; p2 > .8", n_p = 2)   # convex hull of two
        models
fairy_constraints(mx)
#> [1] "- p2 <= 0"                  "- 5/4 * p1 - 5/4 * p2 <= -1"
#> [3] "- p1 <= 0"                  "p2 <= 1"
#> [5] "p1 + p2 <= 6/5"             "p1 <= 1"
fairy_compare(list(A = a, B = b, C = "p3 > p1"), n_p = 3)        # volumes, overlaps,
        nesting, max. BF
#>       A        B    C
#> A     0.17     0.17 0
#> B     superset 0.5  0.17
#> C     1D       3D   0.5
#> maxBF    6        2    2
```

`fairy_compare()` returns the table layout of the paper: volumes on the diagonal, overlap volumes above it, below it “subset” / “superset” (printed as the set symbols) or the dimension of the overlap, and the largest Bayes factors in the last row.

## Which extreme cases do models share?

`fairy_compare_vertices()` lists, for every extreme case of every model, in which other models it is also an extreme case, and which ones only this model has (the Compare buttons of the app):

```
r <- fairy_compare_vertices(list(A = a, B = b, C = "p3 > p1"), n_p = 3)
r                                            # per model: vertices, shared, unique
#> Vertices shared between models
#>  model vertices shared unique compared_with
#>      A        4      4      0          B, C
#>      B        6      6      0          A, C
#>      C        6      4      2          A, B
head(r$vertices, 4)                          # per vertex: the models that share it
#>     model vertex shared_with n_shared unique
#> A.1     A      1        B, C        2  FALSE
#> A.2     A      2           B        1  FALSE
#> A.3     A      3           B        1  FALSE
#> A.4     A      4        B, C        2  FALSE
```

An intersection or a mixture is compared like in the app: ordinary models only with each other, and a model named in `derived` with all ordinary models except, for a mixture, its own components (its extreme cases are theirs by construction; name them in `components`).

# Replication across labs

```
h2 <- fairy_h("p1 > p2", n_p = 2)
fairy_info(fairy_replicate(h2, r = 3, type = "joint"))
```

```
#>   dim vertices inequalities equalities volume max_bf
#> 1   6       27           9          0  0.125      8
```

`type = "identical"` forces all labs to have the same probabilities, `"joint"` lets each lab satisfy the model independently, `"substitutable"` requires every constraint to hold whichever lab's probability is plugged in for each term.

# Replication where labs or items differ

`fairy_replicate()` assumes that every lab has every probability. With `fairy_replicate_labs()` a table says which lab has which probability; a lab that lacks one follows the model projected onto the rest (`p1 > p2 > p3` without `p2` becomes `p1 > p3`). The labs can belong together in four ways: `"joint"` (every lab follows the model on its own), `"substitutable"` (the order holds whichever lab's probability is plugged in), `"identical"` (a probability is the same in all labs, or differs by at most `approx_tol`) or `"average"` (the model holds for the averages over the labs).

```
m <- fairy_h("p1 > p2 > p3", n_p = 3)
present <- rbind(c(TRUE, TRUE, TRUE),    # lab 1 has all three probabilities
                 c(TRUE, FALSE, TRUE))   # lab 2 has no p2
fairy_info(fairy_replicate_labs(m, present, "joint"))
#>   dim vertices inequalities equalities     volume max_bf
#> 1   5       12           7          0 0.08333333     12
```

`fairy_replicate_items()` is the same for items (trials, stimuli, conditions): every probability has its own number of items, and item *k* of one probability goes with item *k* of the others. There is no `"joint"` type here, because items are not independent of each other.

```
h3 <- fairy_replicate_items(m, items = c(2, 2, 3))     # substitutable is the default
colnames(h3)[-(1:2)]
#> [1] "p_{1}^(1)" "p_{2}^(1)" "p_{3}^(1)" "p_{1}^(2)" "p_{2}^(2)" "p_{3}^(2)" "p_{3}^(3)"
fairy_info(h3)
#>   dim vertices inequalities equalities      volume max_bf
#> 1   7       14          15          0 0.004761905    210
```

Replication models with the same number of items for every probability live in the same space and can be combined. `fairy_intersect()`, `fairy_mixture()`, `fairy_nested()` and `fairy_compare()` match their columns by name. The package only checks that the numbers fit; that item *k* of one probability means the same thing as item *k* of another is up to you.

```
avg  <- fairy_replicate_items(m, items = 2, type = "average")
subs <- fairy_replicate_items(m, items = 2, type = "substitutable")
fairy_nested(subs, avg)                # the order for every combination of items implies the
        average
#> [1] TRUE
fairy_info(fairy_intersect(avg, subs), volume = FALSE)
#>   dim vertices inequalities equalities volume max_bf
#> 1   6       10          12          0     NA     NA
```

# Export

`fairy_write_qtest()` and `fairy_write_multinomineq()` write the models as input files for QTEST and for the R package multinomineq (one `.tar` archive with one file per model). The files are the same as the downloads of the app.

```
mods <- c(H1 = "p1 > p2 > p3", H2 = "p1 > p2; p2 = p3")
fairy_write_qtest(mods, n_p = 3, file = "qtest.tar")
fairy_write_multinomineq(mods, n_p = 3, file = "multinomineq.tar")
```

- **Order of the probabilities:** `order = list(H1 = c(3, 1, 2))` makes the third probability the first column of the files of `H1` (QTEST and multinomineq may expect another order than you wrote the model in). Models that are not listed keep their own order.
- **Equalities:** an equality between two probabilities (`p2 = p3`) is merged into one column for multinomineq and listed in `map_<model>.csv` (the `map` argument of multinomineq); QTEST cannot express equalities, so the file lists them in an extra block with a note. Any other equality gives volume zero.
- **Numbers:** constants stay exact fractions (`1/3`, `0.0001`) up to the files; nothing goes through a floating-point step.

## Analysing data with multinomineq

The multinomineq archive also contains your data, the settings of the analysis and a script, `analysis_multinomineq.R`, that analyses every model in its folder and writes `results.csv` (Bayes factors against the unconstrained model and against the model's complement), `posterior.csv` (estimates of the probabilities) and `results.html` (the main results as a web page):

```
dat <- data.frame(probability = c("p1", "p2", "p3"), n = c(30, 30, 45), frequency = c(21, 12,
        9))
fairy_write_multinomineq(mods, n_p = 3, file = "multinomineq.tar", data = dat,
                         settings = list(posterior_samples = 5000, bf_samples = 10000,
                                         burnin_post = 10, burnin_bf = 5, prior = c(1, 1),
        seed = 1234))
```

`data` has one row per probability (`n` observations, `frequency` how often the outcome was observed); the numbers belong to the probability's name, so they stay with it when `order` changes. A list named by model gives that model its own file (`data_model_<name>.csv`, for unpacked models whose probabilities mean something of their own); without `data` the archive contains example numbers that you replace. `fairy_multinomineq_script()` returns the text of the script alone.

To run the analysis directly from R (needs the packages multinomineq and coda), use `fairy_analyze_multinomineq()`:

```
res <- fairy_analyze_multinomineq(mods, n_p = 3, data = dat, settings = list(seed = 1))
res$results      # Bayes factors
res$posterior    # estimates of the probabilities
res$report       # path of results.html
```

The Bayes factors depend on the prior (`prior = c(1, 1)` is the uniform prior on every probability), and the estimates have a Monte Carlo error: for the final numbers use more samples (`bf_samples`) and report the seed.

# All functions

| Function | What it does |
|---|---|
| `fairy()` | constraints, extreme cases, volume and max. Bayes factor for one or several models |
| `fairy_h()` | non-redundant H-representation of a model string |
| `fairy_v()` | V-representation in rcdd format |
| `fairy_from_vertices()` | H-representation from a table of vertices (V to H) |
| `fairy_constraints()` | H-representation as readable text lines |
| `fairy_vertices()` | extreme cases as a numeric matrix |
| `fairy_words()` | extreme cases in words (“all” / “none” / fractions) |
| `fairy_h_table()`, `fairy_v_table()` | H- and V-description as tables (`fairy_table_write()`: Markdown, LaTeX, CSV) |
| `fairy_dim()` | dimension of a model |
| `fairy_volume()` | parsimony: volume in the unit cube (exact, volesti, Monte Carlo, or `"auto"`: exact first as in the app) |
| `fairy_max_bf()` | largest possible Bayes factor against the unconstrained model (1 / volume) |
| `fairy_info()` | dimension, vertices, constraints, volume, max. Bayes factor in one row |
| `fairy_parsimony()` | older interface to the volume |
| `fairy_intersect()` | intersection of models |
| `fairy_mixture()` | mixture model (convex hull of the components) |
| `fairy_nested()` | is one model nested in another? |
| `fairy_compare()` | comparison table of a set of models (Table 1 of the paper) |
| `fairy_replicate()` | replication across labs: identical, joint, substitutable |
| `fairy_replicate_labs()` | replication across labs that do not all have every probability: joint, substitutable, identical (with tolerance), average |
| `fairy_replicate_items()` | replication across items, each probability with its own number: substitutable, identical (with tolerance), average |
| `fairy_compare_vertices()` | which extreme cases do models share, which are unique to a model |

| Function | What it does |
|---|---|
| `fairy_write_qtest()` | write input files for QTEST |
| `fairy_write_multinomineq()` | write input files for multinomineq: models, data, settings and the analysis script |
| `fairy_multinomineq_script()` | text of the analysis script of the multinomineq archive |
| `fairy_analyze_multinomineq()` | run the multinomineq analysis from R: Bayes factors and posterior estimates |

Help for any function: `?fairy_volume`; all of them: `help(package = "fairy")`.