linemodels R package (v.0.3.0)

Matti Pirinen, University of Helsinki, Finland

Nov 20, 2023

Summary Estimation of effects of multiple explanatory variables on multiple outcome measures has become routine across life sciences with high-throughput molecular technologies. The `linemodels` R-package allows a probabilistic clustering of variables based on their observed effect sizes on two outcomes. User can specify each candidate model by three parameters:

- scale, i.e., the magnitude of the effects,
- slope, i.e., the multiplicative relationship between the expected values of the two effects, and
- correlation, i.e., the allowed deviation from the expectation.

`linemodels` provides functions to estimate model probabilities of each variable together with the proportion parameters of the models, by taking into account uncertainty in the effect estimates and a possible correlation of the two effect estimators. The package further allows for optimization of the model parameters and use of Gaussian mixture models as the prior distribution of effect sizes.

Available at: https://github.com/mjpirinen/linemodels

1. Motivating example

Consider effect estimates of $n = 100$ genetic variants on two subtypes Y_1 and Y_2 of a disease (Fig. 1A). Suppose that we are interested whether each variant has

(1) similar effects on both subtypes ($\beta_1 \approx \beta_2$), or
(2) a smaller effect on Y_2 than on Y_1 ($\beta_2 < \beta_1$), or
(3) no effect at all on Y_2 ($\beta_2 \approx 0$).

(While additional models might also be of interest, we keep this motivating example simple by only considering the three models above.)

We apply `linemodels` by specifying the three models that are visualized in Fig. 1B as colored lines. Output includes variable-specific posterior probabilities of the membership in each model (Fig. 1C) as well as proportion estimates for each model (Fig. 1D). From panel C we learn, for example, that while variant 3 has a very high probability of belonging to M_1, the data for variant 2 are indecisive between M_1 and $M_{0.5}$. From panel D we see, for example, that about 18% (95% credible interval from 0.12 to 0.27) of the variants belong to M_0, and hence have $\beta_2 \approx 0$.

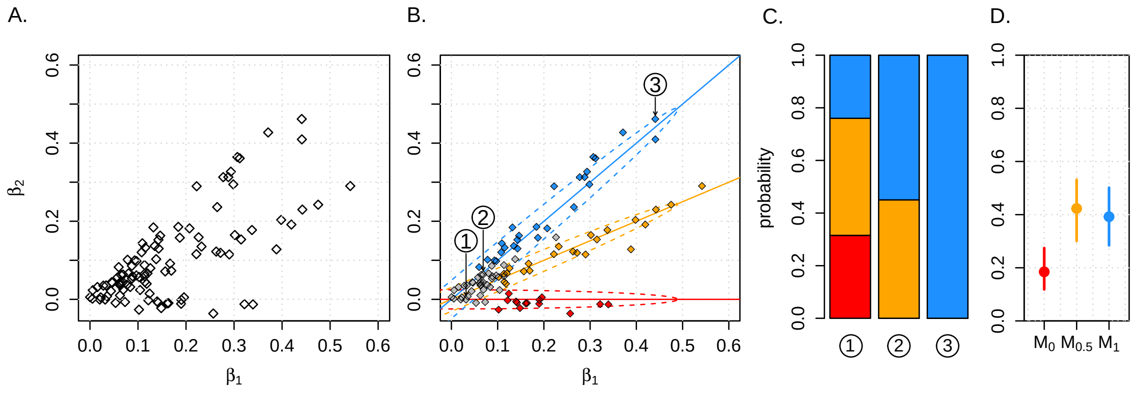


Figure 1. **A.** Effect estimates of $n = 100$ variables on two outcomes. **B.** Three line models shown as colored lines with their 95% highest probability regions (dashed lines). The slopes of the models are 1 (model M_1, blue), 0.5 ($M_{0.5}$, orange) and 0 (M_0, red) corresponding to the above mentioned three relationships $\beta_1 \approx \beta_2$, $\beta_2 < \beta_1$ and $\beta_2 \approx 0$, respectively. Three exemplar variables are labelled as "1", "2" and "3". **C.** Variable-specific posterior probabilities of the membership in each model for the three variables labelled in panel **B**. **D.** Posterior distribution of proportion estimates of the models across all $n = 100$ variables.

Note that the analysis using `linemodels` is not a trivial distance comparison between the points and the lines since

- we account for varying uncertainty of the effects on the two outcomes,
- we account for possible correlation between the effects on the two outcomes, e.g., due to shared control individuals in disease studies,
- we have defined Gaussian probability models surrounding the lines.

With the `linemodels` package, it is also possible to optimize the parameters of each line model rather than fix them before the analysis as we have done in Fig. 1.

Next, we introduce the functions available in the `linemodels` package. Mathematical description of the model and algorithms can be found at the end of this document.

2. Install `linemodels`

To install `linemodels` from GitHub, you need to install 'devtools' package (in case you don't already have it).

```
install.packages("devtools")
```

After that, install `linemodels` with the following commands

```
library(devtools)
install_github("mjpirinen/linemodels")
library(linemodels)
```

If you can't get the above working, all functions are in a single R file which you can also read in directly to R using command below

```
source("https://raw.githubusercontent.com/mjpirinen/linemodels/main/R/linemodels.R")
```

3. Specifying models

Each line model k is specified by three parameters called **scale** (s_k), **slope** (b_k) and **correlation** (r_k). These parameters are given as separate vectors `scales`, `slopes` and `cors`, each having K values where K is the number of line models. Thus, to specify the following $K = 3$ models from Fig.1,

- M_1 with $s = 0.2, b = 1, r = 0.995$,
- $M_{0.5}$ with $s = 0.2, b = 0.5, r = 0.995$,
- M_0 with $s = 0.2, b = 0, r = 0.995$,

we will define vectors:

```
scales = c(0.2, 0.2, 0.2)
slopes = c(1, 0.5, 0)
cors = c(0.995, 0.995, 0.995)
```

visualize.line.models( ) We can visualize the models using `visualize.line.models( )` to check that they seem reasonable. (Note: This function allows also other plotting parameters to be adjusted; see the help page of the function.)

```
visualize.line.models(scales, slopes, cors,
                      model.names = c("M1","M.5","M0"),
                      model.cols = c("dodgerblue", "orange", "red"),
                      legend.position = "topleft",
                      xlim = c(0, 0.6), ylim = c(0, 0.6),
                      xlab = expression(beta[1]),
                      ylab = expression(beta[2]),
                      emphasize.axes = FALSE)
```

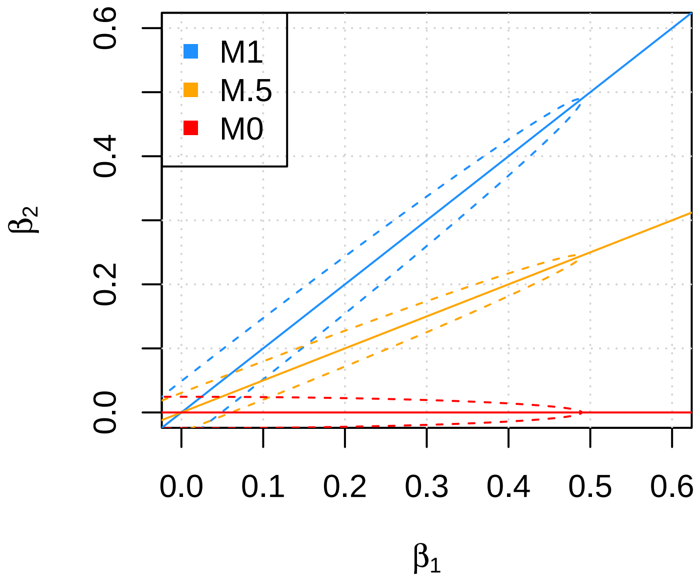


Figure 2. The scale parameter s defines the standard deviation of the zero-centered effect size distribution of the larger effect size. The correlation r defines how much deviation we allow around the exact line. Here the 95% regions determined by r are shown by the dashed lines.

4. Estimating line models

We will use the example data from Fig. 1 to demonstrate the analyses. The data set is included in the package under the name `linemodels_ex1`, and can also be found from the GitHub page under /data.

```
dat = linemodels.ex1
head(dat, 2)
```

```
##       beta1        beta2        se1        se2
## 1 0.3394143 -0.012909064 0.01555013 0.01555013
## 2 0.1634416 -0.009596917 0.01510127 0.01510127
```

For each variable, we need its effect size estimates on the two outcomes (here `beta1` and `beta2`) and the corresponding standard errors of these estimates (`se1` and `se2`). Additionally, we will need the correlation `r.lkhood` between the estimators of the two effect sizes. Here we assume that the two estimates come from independent data sets, and hence `r.lkhood = 0`. If overlapping samples and/or correlated outcome measures were used, then a non-zero `r.lkhood` value should be specified. (See section 8 for examples.)

line.models( ) To estimate the model probabilities separately for each variable, we need to fix the prior probabilities of the models. Let's use equal priors.

```
model.priors = c(1/3, 1/3, 1/3)
model.names = c("M1","M.5","M0")
```

Now we can apply `line.models()` that gives us the membership probabilities of each variable in each model.

```
res = line.models(X = dat[,c("beta1","beta2")], SE = dat[,c("se1","se2")],
                  scales = scales, slopes = slopes, cors = cors,
                  model.names = model.names,
                  model.priors = model.priors,
                  r.lkhood = 0)
head(res, 2)
```

```
##                M1          M.5        M0
## [1,] 7.458412e-31 3.139269e-16 1.0000000
## [2,] 2.350610e-08 8.985146e-05 0.9999101
```

The result has one row per each variable and one column per each model. The example output above tells that for the first two variables the posterior probability is close to 1 that they follow the model M_0 when the other two candidates were M_1 or $M_{0.5}$.

line.models.with.proportions( ) If we do not want to fix the prior probabilities of the line models before the analysis, we can assign a prior distribution on those probabilities and estimate them. The prior distribution is Dirichlet$(a_1, \ldots, a_K)$, where the user can give the values a_k in a K-dimensional vector called `diri.prior`. By default, we have $a_k = \frac{1}{K}$ for all $k \leq K$. Since this approach uses an iterative Gibbs sampler, we can also specify the number of iterations (default 200) and the length of the burn-in period (default 20). Larger numbers of iterations may improve accuracy in some data sets.

```
res.2 = line.models.with.proportions(
  X = dat[,c("beta1","beta2")],
  SE = dat[,c("se1","se2")],
  scales = scales, slopes = slopes, cors = cors,
  model.names = model.names,
  r.lkhood = 0,
  n.iter = 200, n.burnin = 20)
```

The result is a list with two members:

- `params` contains the posterior distribution of the proportion of variables belonging to each of the models (mean, 95% highest posterior region and standard deviation)
- `groups` has the posterior probabilities of each variable belonging to each of the models (similar to output of `line.models( )` above)

```
res.2$params
```

```
##          mean    95%low     95%up         sd
## M1  0.4090314 0.2999994 0.5268116 0.06013744
## M.5 0.4032098 0.2822057 0.5242396 0.05948362
## M0  0.1877588 0.1148746 0.2749059 0.04440714
```

The proportion estimates above and their 95% intervals correspond to Fig.1D.

```
head(res.2$groups, 2)
```

```
##      M1 M.5 M0
## [1,]  0   0  1
## [2,]  0   0  1
```

In these data, the membership probabilities are highly similar between `line.models.with.proportions( )` and `line.models( )` since the estimated proportions of the three groups do not differ much from the uniform prior that we used in `line.models( )`. Note that in `line.models.with.proportions( )`, the unit of accuracy of the probabilities is determined by the number of iterations and, for example, with 200 iterations the unit is 1/200 = 0.5%. This explains why we see here probabilities that are exactly 1 whereas earlier with `line.models( )` we saw some tiny deviations from 1, such as value of 0.9999101 for the second variable.

visualize.classification.regions( ) For any two line models, we can visualize the regions on the plane according to the posterior probabilities of the putative observations if they were observed in the region. The probabilities will depend strongly on the standard errors (parameter `SE`) and the prior probabilities of the models (parameter `model.priors`).

Let's visualize the comparison of the first two models with both effects estimated with an SE = 0.05 and both models being equally probable *a priori*.

```
visualize.classification.regions(
  scales[1:2], slopes[1:2], cors[1:2],
  SE = c(0.05, 0.05), r.lkhood = 0,
  model.priors = c(0.5, 0.5),
  add.legend = TRUE,
  breakpoints = 200,
  xlab = expression(beta[1]),
  ylab = expression(beta[2]),
  emphasize.axes = TRUE)
```

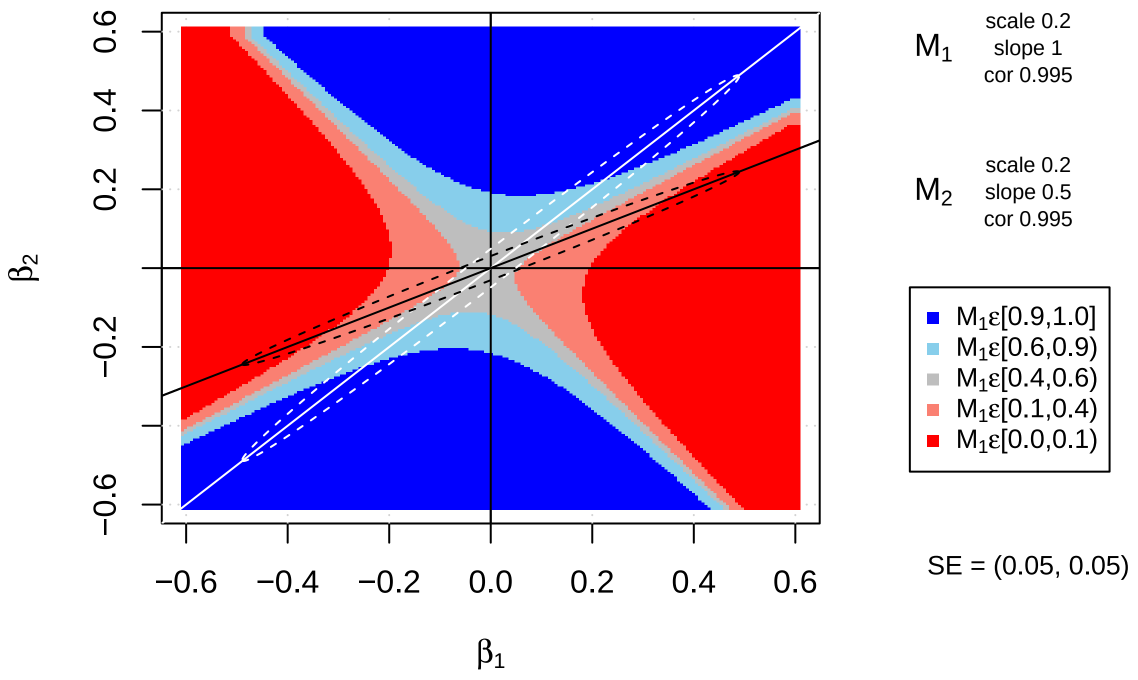


Figure 3. Each point on the plane within 3 scale units from the origin is colored according to the posterior probability of the first model (here model with slope = 1) where blue colors mark higher probabilities and red colors mark lower probabilities (see legend). This plot depends on the values of standard errors (`SE`), prior probabilities of the models (`model.priors`) and the correlation of the estimators (`r.lkhood`).

5. Optimizing parameters of line models

Sometimes we may not have a good reason to fix the parameters of a line model to any particular values of scale, slope or correlation, but rather we may want to estimate from the data a set of suitable values for the parameters.

line.models.optimize( ) The optimization function takes in initial values of all parameters and a matrix named `par.include` containing `TRUE/FALSE` entries that describe which parameters will be included in the optimization process. This matrix has one row per each model and the three columns correspond to scale, slope and correlation, respectively.

For example, let's optimize the slope of the second model that represents variants where $\beta_1 > \beta_2 > 0$. We may not want to assume that $\beta_2 = 0.5 \cdot \beta_1$ in case that some other slope would clearly better describe the relationship in the data. Let's optimize over b_2 and use an initial value of $b_2 = 0.3$.

```
#par.include here says that only the slope of 2nd model will be optimized
par.include = rbind(c(FALSE, FALSE, FALSE),
                    c(FALSE, TRUE, FALSE),
                    c(FALSE, FALSE, FALSE))
line.models.optimize(
  X = dat[,c("beta1","beta2")],
  SE = dat[,c("se1","se2")],
  par.include = par.include,
  init.scales = scales,
  init.slopes = c(0, 0.3, 1), #initial value for 2nd slope is 0.3
  init.cors = cors,
  model.priors = model.priors,
  model.names = model.names,
  r.lkhood = 0)
```

```
##
##
## Initial values
## scales: 0.2, 0.2, 0.2
## slopes: 0, 0.3, 1
## cors: 0.995, 0.995, 0.995
## proportions: 0.333, 0.333, 0.333
## Initial log-lkhood: 62.54773
## Optimizing w.r.t: slope2, over the range: (-Inf,Inf)
## Convergence criteria:
##  Relative diff in parameters <= 0  or
##  Difference in log-likelihood < 0.001
##
## iter: 6 ; log-lkhood: 180.28254
## Relative diffs in optimized parameters: 0.000205
## proportions: 0.187, 0.417, 0.395
## scales: 0.2, 0.2, 0.2
## slopes: 0, 0.5065, 1
## cors: 0.995, 0.995, 0.995
##
## Log-likelihood value converged.
```

```
##      scales    slopes  cors
## [1,]    0.2 0.0000000 0.995
## [2,]    0.2 0.5064719 0.995
## [3,]    0.2 1.0000000 0.995
```

As the optimization proceeds, the function prints on the console some status of the process (see `print.steps` parameter to control output). The final result shows that the EM-algorithm converged to value $b_2 = 0.506$. This is very close to the value 0.5 that we had been using, and hence updating this slope value in the line model analysis would not lead to a noticeable difference from the results of our earlier analyses.

Additional arguments of line.models.optimize()

- `force.same.scales`, logical, if TRUE then all the scale parameters will be forced to be equal if any of them are being optimized. Defaults to FALSE.
- `tol.loglk`, defines the largest increase in log-likelihood between adjacent iterations that is considered small enough that the algorithm has converged. Defaults to 0.001.
- `tol.par`, defines the largest absolute value of the relative change in parameter values that is considered small enough that the algorithm has converged. Defaults to 0, that is, convergence is determined purely by the log-likelihood value.
- `print.steps` takes value of 0 (no output to console), 1 (only initial and final results shown, default) or 2 (prints state after every EM-step).

When to optimize? We shouldn't just blindly optimize the model parameters for model comparison since we may overfit to the data and the optimized model may not anymore describe exactly that property which we wanted it to describe in the first place. In cases where we can come up with a particular model that exactly describes our hypothesis of interest, such as no effect on disease subtype 2 ($\beta_2 = 0$) or same effect in both subtypes ($\beta_1 = \beta_2$), we should simply use such a model rather than use the optimization to improve the fit. On the other hand, optimization is useful when we can see from the data that there is a particular relationship that we want to model, but we do not know exactly what would be an appropriate parameter value that captured well that relationship.

Suitable scales. The suitable values for scale parameters are crucial in order to get sensible results from the line models. If, for example, scales are too small for the observed data, then none of the line models can describe the data well and the resulting assignment probabilities may not be sensible. To let the data decide suitable scales, we can optimize the scale parameters first and then estimate the proportion parameters using an empirical Bayes method with scales set to their maximum likelihood estimates. If it is reasonable to assume that all line models have the same scale, then we can set `force.same.scales = TRUE` to force the scale parameter to be shared by all the line models. This may add robustness to the parameter estimation escpecially when there is not much data. If the scales of the line models are expected to be different, then forcing them to be the same may lead to misleading results.

6. Likelihood ratios for model comparison

A way to compare two mixture models built from sets of line models is via likelihood ratio. The optimization procedure can report both the initial and the optimized log-likelihoods of the whole mixture model that is being considered in the optimization. The difference in log-likelihoods between the mixtures of line models can be used as a statistic for model comparison. For example, we can see whether forcing same scales causes a big decrease in log-likelihood or whether a mixture of two line models improves a lot over a model with only a single line model. Unfortunately, there is no theoretically known distribution for the logarithm of the likelihood ratio under different null hypotheses so one needs to estimate the null distribution by simulation. Such a procedure is implemented in function `simulate.loglr()`.

This function takes in the null model (NULL) and the alternative model (ALT) specified similar as previously for the optimization function. That is, for the null model, we need to give `par.include.null`, `force.same.scales.null`, `init.scales.null`, `init.slopes.null`, `init.cors.null` and `model.priors.null`. And to specify the alternative model, the same parameters are used except `.null` is always replaced by `.alt`. Additional control parameters used in optimization can be specified as well.

This function first optimizes both ALT and NULL on the observed data and records log of likelihood ratio (logLR) between ALT and NULL.

Then the function generates such data according to the optimized null model that mimicks the observed data in terms of sample size and scales. For each data set, ALT and NULL are optimized and logLR is recorded. At the end, observed logLR can be compared to the values of logLR generated under the null, for example, to estimate empirical P-value for the need of ALT model compared to NULL.

Let's assess whether an addition of a 3rd line model helps in the example data. We fully specify the null model to have two lines (0.2, 0, 0.995) and (0.2, 1, 0.995), and then we let the alternative model to have an additional line with unknown slope but fixed scale (0.2) and correlation (0.995). For illustration, here we estimate the null distribution of logLR using only 10 simulated data sets.

```
set.seed(1655)
sim.loglr = simulate.loglr(
  X = dat[,c("beta1","beta2")],
  SE = dat[,c("se1","se2")],
  n.sims = 10,
  par.include.null = rbind(c(FALSE,FALSE,FALSE),
                           c(FALSE,FALSE,FALSE)),
  init.scales.null = c(0.2, 0.2),
  init.slopes.null = c(0, 1),
  init.cors.null = c(0.995, 0.995),
  par.include.alt = rbind(c(FALSE,FALSE,FALSE),
                          c(FALSE,FALSE,FALSE),
                          c(FALSE,TRUE,FALSE)),
  init.scales.alt = c(0.2, 0.2, 0.2),
  init.slopes.alt = c(0, 1, 0.5),
  init.cors.alt = c(0.995, 0.995, 0.995),
  r.lkhood = 0, tol.loglk = 1e-2,
  print.steps = 0)
```

The result object shows the logLR for the observed data and the distribution of the logLR values for the null simulations.

```
sim.loglr
```

```
## $obs.loglr
## [1] 261.6853
##
## $null.loglr
##  [1] 0.2599693 0.3224582 0.5169074 0.1645962 0.3424981 0.2542259 3.4768042
##  [8] 0.3522670 1.5473749 0.2507949
```

Here the observed logLR is much larger than values simulated from the null data, which suggests that the 3rd line model is very useful in explaining the data. NOTE: More simulations than done here should be made in order to have a reliable idea about statistical significance of the alternative model compared to the null.

7. A mixture of Gaussians as the prior distribution of effects

By default, the prior distribution of the larger effect is $\mathcal{N}(0, s^2)$ where s is the scale parameter. (The larger effect is β_1 when $|b| \leq 1$ and β_2 when $|b| > 1$ where b is the slope parameter.) For the `line.models( )` function, it is also possible to specify the effect size prior as a mixture of Gaussians. This can be done by defining `scale.weights` matrix, that has one row per each model and the number of columns is the number of mixture components (L). Each row is normalized to sum to 1. Denote by $w_{k\ell}$ the element (k,ℓ) of `scale.weights` matrix. Then the prior for the larger effect in model k is

$$\sum_{\ell=1}^{L} w_{k\ell}\,\mathcal{N}\left(0, \left(\frac{s_k}{L-\ell+1}\right)^2\right),$$

where s_k is the scale parameter of the model k. Note that s_k is given in standard deviation units and hence s_k^2 is the corresponding variance. (In case the weights of each model are the same, we can also give the weights as a single vector instead of a matrix that had K identical rows.)

For example, if we want to model the effect size of models

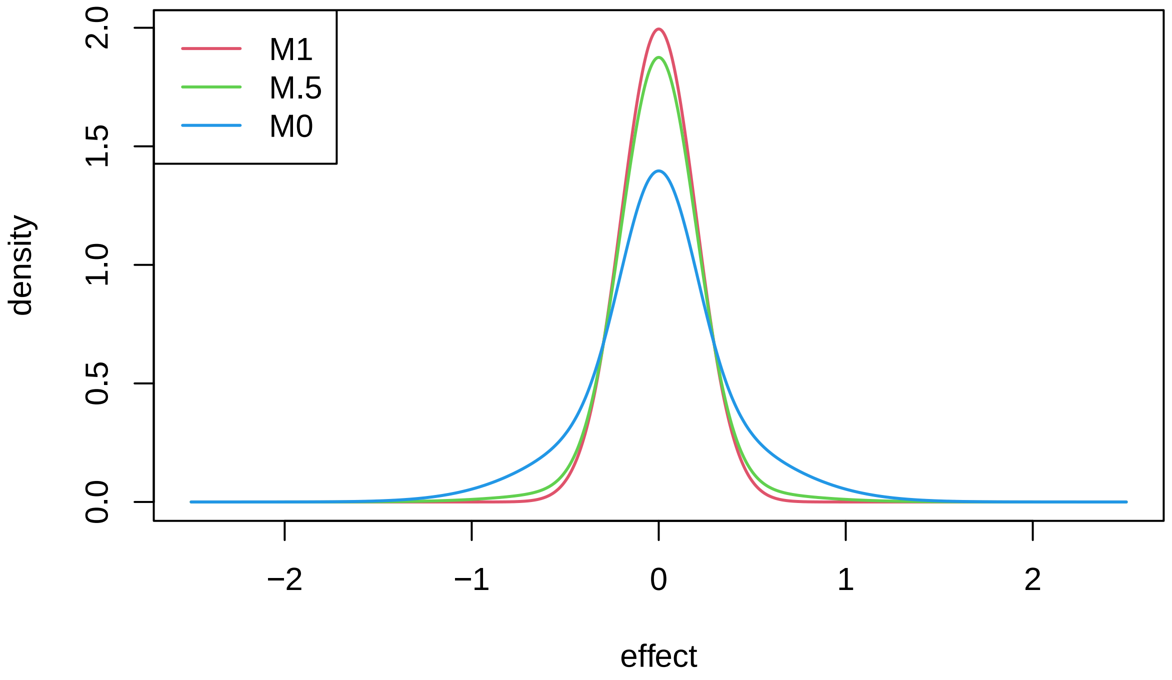
- M_1 as $\mathcal{N}(0, 0.2^2)$,
- $M_{0.5}$ as a 10%:90% mixture of $\mathcal{N}(0, 0.5^2)$ and $\mathcal{N}(0, 0.2^2)$,
- M_0 as a 50%:50% mixture of $\mathcal{N}(0, 0.5^2)$ and $\mathcal{N}(0, 0.2^2)$,

then we can set all scales to $s_k = 1.0$ and specify the `scale.weights` as

```r
scales = c(1, 1, 1)
# elements of scale.weights will now corresp. to scales of
# 0.2 = 1.0/5, 0.25 = 1.0/4, 0.33 = 1.0/3, 0.5 = 1.0/2 and 1.0 = 1.0/1:
scale.weights = rbind(
  c(1.0, 0, 0, 0.0, 0.0),  #N(0, 0.2^2)
  c(0.9, 0, 0, 0.1, 0.0),  #0.9*N(0, 0.2^2) + 0.1*N(0, 0.5^2)
  c(0.5, 0, 0, 0.5, 0.0))  #0.5*N(0, 0.2^2) + 0.5*N(0, 0.5^2)
```

visualize.scales( ) We can visualize the mixture prior distribution:

```r
visualize.scales(scales, scale.weights, model.names)
```



```
## Plotted the following effect distributions:
## M1: Scale = 1 weights = 1,0,0,0,0.
## M.5: Scale = 1 weights = 0.9,0,0,0.1,0.
## M0: Scale = 1 weights = 0.5,0,0,0.5,0.
```

Applying these priors to effect sizes in `line.models( )` is now straighforward: we just add `scale.weights` as an argument in the function call.

```r
res.sc = line.models(X = dat[,c("beta1","beta2")], SE = dat[,c("se1","se2")],
                     scales = scales, slopes = slopes, cors = cors,
                     model.names = model.names,
                     model.priors = model.priors,
                     r.lkhood = 0,
                     scale.weights = scale.weights)
head(res.sc, 2)
```

```
##                 M1          M.5        M0
## [1,] 7.573082e-31 2.160325e-07 0.9999998
## [2,] 3.455825e-08 1.840330e-03 0.9981596
```

8. How to set the value of `r.lkhood`?

When the effect estimators on the two outcomes are correlated, we must account that in our observation model through the parameter `r.lkhood`.

1. Suppose Y_1 (outcome 1) and Y_2 (outcome 2) are continuous outcomes and we have measured the effect of variable X on both of them separately on the same samples using linear regression models $Y_1 = \mu_1 + X\beta_1 + \varepsilon_1$ and $Y_2 = \mu_2 + X\beta_2 + \varepsilon_2$. Then,

$$\text{r.lkhood} = \text{cor}\left(\widehat{\beta}_1, \widehat{\beta}_2\right) = \text{cor}(\varepsilon_1, \varepsilon_2) \approx \text{cor}(Y_1, Y_2),$$

   where the last approximation assumes that X explains only a little of the variance of the outcome variables. Thus, to a first approximation, we may set `r.lkhood` equal to the correlation of the two outcome variables. If the samples are only partially overlapping, then we will need to shrink the correlation of the residuals towards zero by a factor of $n_{12}/\sqrt{n_1 n_2}$, where n_{12} is the number of samples that are shared between the analyses and n_1 and n_2 are the total sample sizes of each analysis.

2. If our data come from two case-control analyses A and B, where samples are partially overlapping, we can derive a value for `r.lkhood` using the formula by Bhattacharjee et al. (2012). Denote by $n_A^{(1)}$ the number of cases in A and by $n_A^{(0)}$ the number of controls in A, and similarly for study B. Mark by $n_{AB}^{(11)}$ and $n_{AB}^{(00)}$, respectively, the number of shared cases and shared controls between the studies and by $n_{AB}^{(10)}$ and $n_{AB}^{(01)}$, respectively, the number of cases of A that are controls of B and the number of controls of A that are cases of B. Then

$$\text{r.lkhood} = \sqrt{\frac{n_A^{(1)} n_A^{(0)}}{n_A^{(1)} + n_A^{(0)}}}\sqrt{\frac{n_B^{(1)} n_B^{(0)}}{n_B^{(1)} + n_B^{(0)}}}\left(\frac{n_{AB}^{(11)}}{n_A^{(1)} n_B^{(1)}} - \frac{n_{AB}^{(10)}}{n_A^{(1)} n_B^{(0)}} - \frac{n_{AB}^{(01)}}{n_A^{(0)} n_B^{(1)}} + \frac{n_{AB}^{(00)}}{n_A^{(0)} n_B^{(0)}}\right).$$

9. Mathematical and technical details

Let $\hat{\beta}_{ij}$ be the effect estimate of variable $i = 1, \dots, n$ on outcome $j = 1, 2$, and $\hat{\sigma}_{ij}$ its estimated standard error. Define, for $k = 1, \dots K$, a line model $\mathcal{M}_k$ via three parameters $\mathcal{M}_k = (s_k, b_k, r_k)$ called scale s_k, slope b_k and correlation r_k, respectively. Intuitively, $\mathcal{M}_k$ models the effects as centered around line $\beta_{i2} = b_k \cdot \beta_{i1}$, with larger of the two effects having prior standard deviation ("scale") of s_k and the deviation from the line determined by the correlation coefficient r_k. We first define the model for a diagonal case (slope = 1), and then use an orthogonal transformation to rotate the model to match the target slope.

For given $\mathcal{M}_k = (s_k, b_k, r_k)$, define the corresponding diagonal distribution of effect sizes as a bivariate Gaussian $\mathcal{N}_2(\mathbf{0}, \boldsymbol{D}_k)$, where the covariance matrix is

$$\boldsymbol{D}_k = \begin{pmatrix} 1 & r_k \\ r_k & 1 \end{pmatrix}.$$

Let $\boldsymbol{T}_k$ be the rotation matrix that transforms the diagonal line to the line with slope b_k:

$$\boldsymbol{T}_k = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix}, \quad \text{where} \quad \alpha = \arctan(b_k) - \frac{\pi}{4}.$$

The prior distribution of the effect sizes according to model $\mathcal{M}_k$ is then defined as $\mathcal{N}_2(\mathbf{0}, \boldsymbol{\Theta}_k)$, where covariance matrix $\boldsymbol{\Theta}_k = \frac{s_k^2}{m_k} \boldsymbol{T}_k \boldsymbol{D}_k \boldsymbol{T}_k^T$ and normalization by $m_k = \max\{\boldsymbol{T}_k \boldsymbol{D}_k \boldsymbol{T}_k^T\}$ confirms that the larger of the standard deviations of the two effects is s_k. It is also possible to specify the prior distribution of the effect size as a mixture of Gaussians, e.g., to model heavier tails (see section 7 above).

In terms of the parameters defined above, we can use line models to define, for example,

- null model ($s_k = 0$),
- independent effects model ($s_k > 0, r_k = 0$),
- fixed effects model ($s_k > 0, b_k = 1, r_k = 1$),
- outcome 1 specific effect ($s_k > 0, b_k = 0, r_k = 1$),
- outcome 2 specific effect ($s_k > 0, b_k = \infty, r_k = 1$),
- and any generalizations of these where deviation from exact line is allowed ($0 \leq r_k < 1$).

The observation model for the effect size estimates is a Gaussian distribution around the true effect sizes with a covariance matrix

$$\hat{\mathbf{\Sigma}}_i = \begin{pmatrix} \hat{\sigma}_{i1}^2 & \hat{\sigma}_{i1}\hat{\sigma}_{i2}\hat{\rho}_i \\ \hat{\sigma}_{i1}\hat{\sigma}_{i2}\hat{\rho}_i & \hat{\sigma}_{i2}^2 \end{pmatrix},$$

where $\hat{\rho}_i$ describes how the two effect size estimators are correlated, for example, because of the sample overlap in the data sets from where the two effects have been estimated (see section 8 above).

It follows that by combining the Gaussian prior with the Gaussian observation model, the marginal distribution for the observed effect size estimates $\widehat{\boldsymbol{\beta}}_i = \left(\widehat{\beta}_{i1}, \widehat{\beta}_{i2}\right)^T$ under model $\mathcal{M}_k$ is

$$\widehat{\boldsymbol{\beta}}_i \,|\, \mathcal{M}_k \sim \mathcal{N}_2\left(\mathbf{0}, \mathbf{\Theta}_k + \hat{\mathbf{\Sigma}}_i\right).$$

9.1. Probabilistic assignment of variables into models Given K line models $(\mathcal{M}_k)_{k=1}^K$ and their prior probabilities $\boldsymbol{\pi} = (\pi_k)_{k=1}^K$, we can estimate the posterior probabilities that variable i follows the model k as

$$\Pr(i \sim \mathcal{M}_k \,|\, \text{Data}_i) = \frac{\pi_k \, \mathcal{N}_2\left(\widehat{\boldsymbol{\beta}}_i \,|\, \mathbf{0}, \mathbf{\Theta}_k + \hat{\mathbf{\Sigma}}_i\right)}{\sum_{\ell=1}^K \pi_\ell \, \mathcal{N}_2\left(\widehat{\boldsymbol{\beta}}_i \,|\, \mathbf{0}, \mathbf{\Theta}_\ell + \hat{\mathbf{\Sigma}}_i\right)}.$$

This calculation can be done separately for each variable and is implemented in the `line.models( )` function.

If we do not want to pre-specify the exact prior probabilities of each model, we can set a prior distribution on $\boldsymbol{\pi}$ and estimate its posterior distribution together with assignment probabilities of variables into models. For this task, we use a prior distribution $\boldsymbol{\pi} \sim \text{Dirichlet}(\delta_1, \ldots, \delta_K)$, where the default values of the hyper-parameter are set to $\delta_k = \frac{1}{K}$ for each $k = 1, \ldots, K$. A Gibbs sampler to estimate the posterior distribution of this model is implemented in the `line.models.with.proportions( )` function. Note that this model estimates the probabilities jointly across all observations rather than separately for each observations as was done by `line.models( )`.

9.2. Likelihood function of the mixture model We can combine the K line models into a single mixture model that can be thought as generating the data in two steps. First, we choose one of the K possible line models according to their probabilities $\boldsymbol{\pi}$ and conditional on the chosen model we sample an observation from it. This mixture model gives the observed data a probability density function

$$\Pr\left(\widehat{\boldsymbol{\beta}} \,|\, \mathbf{\Gamma}, \boldsymbol{\pi}\right) = \prod_{i=1}^n \sum_{k=1}^K \pi_k \, \mathcal{N}_2\left(\widehat{\boldsymbol{\beta}}_i \,; 0, \widehat{\mathbf{\Sigma}}_i + \mathbf{\Theta}_k\right),$$

where $\mathbf{\Gamma} = \{s_k, b_k, r_k \,|\, k \leq K\}$ contains all $3K$ line parameters of the K models. Logarithm of this, considered as a function of the parameters, is the mixture model likelihood function of the parameters:

$$\log L\left(\mathbf{\Gamma}, \boldsymbol{\pi} \,;\, \text{Data}\right) = \sum_{i=1}^n \log\left(\sum_{k=1}^K \pi_k \, \mathcal{N}_2\left(\widehat{\boldsymbol{\beta}}_i \,; 0, \widehat{\mathbf{\Sigma}}_i + \mathbf{\Theta}_k\right)\right).$$

9.3. EM-algorithm to optimize line model parameters Using the `line.models.optimize()` function, one can optimize the likelihood function with respect to any set of the model parameters (see section 5 above for an example).

With K line models our model parameters are scales s_k, slopes b_k, correlations r_k and proportions π_k for each model $k \leq K$, but because of the constraint $\sum_k \pi_k = 1$, we can write $\pi_K = 1 - \pi_1 - \ldots - \pi_{k-1}$ and we have only $4K - 1$ free parameters. As before, denote the $3K$ line parameters by $\mathbf{\Gamma} = \{s_k, b_k, r_k \,|\, k \leq K\}$ and the K proportion parameters by $\boldsymbol{\pi} = \{\pi_k \,|\, k \leq K\}$, and their values at iteration t of the algorithm as $\mathbf{\Gamma}^{(t)}$ and $\boldsymbol{\pi}^{(t)}$, respectively, with the initial values corresponding to $t = 0$.

As is typical for EM-algorithms for mixture models, we introduce latent variable $z_i \in \{1, ..., K\}$ for each observation $i = 1, \ldots, n$ that tells to which line model the observation belongs. The augmented likelihood of the data consisting of the effect estimates $\left(\widehat{\beta}_i\right)_{i=1}^n$ and the corresponding standard errors $(\widehat{\sigma}_i)_{i=1}^n$ is

$$L(\mathbf{\Gamma}, \boldsymbol{\pi}, \boldsymbol{z}; \text{Data}) = p(\boldsymbol{\pi}) \prod_{i=1}^n p(z_i \,|\, \boldsymbol{\pi})\, p(\text{Data}_i \,|\, z_i) = c \cdot \prod_{i=1}^n \pi_{z_i} \mathcal{N}_2\left(\widehat{\boldsymbol{\beta}}_i \,|\, \mathbf{0}, \mathbf{\Theta}_{z_i} + \hat{\mathbf{\Sigma}}_i\right),$$

where c is the normalizing constant of the K-dimensional uniform distribution of the proportion parameter $\boldsymbol{\pi}$. The log-likelihood is

$$\mathcal{L}(\mathbf{\Gamma}, \boldsymbol{\pi}, \boldsymbol{z}) = \log L(\mathbf{\Gamma}, \boldsymbol{\pi}, \boldsymbol{z}; \text{Data}) = \log(c) + \sum_{i=1}^n \left[\log(\pi_{z_i}) + \log \mathcal{N}_2\left(\widehat{\boldsymbol{\beta}}_i \,|\, \mathbf{0}, \mathbf{\Theta}_{z_i} + \hat{\mathbf{\Sigma}}_i\right)\right].$$

The EM-algorithm at iteration $t > 0$ consists of two steps.

1. Compute the posterior distribution $\left(p_{ik}^{(t)}\right)_{i,k}$ of latent variables $(z_i)_{i=1}^n$ conditional on current values of parameters, $\mathbf{\Gamma}^{(t-1)}$ and $\boldsymbol{\pi}^{(t-1)}$. This computation can be done separately for each variable i using the assignment probability formula

$$p_{ik}^{(t)} = \Pr(i \sim \mathcal{M}_k \,|\, \mathbf{\Gamma}^{(t-1)}, \boldsymbol{\pi}^{(t-1)}, \text{Data}_i) = \frac{\pi_k^{(t-1)} \mathcal{N}_2\left(\widehat{\boldsymbol{\beta}}_i \,|\, \mathbf{0}, \mathbf{\Theta}_k^{(t-1)} + \hat{\mathbf{\Sigma}}_i\right)}{\sum_{\ell=1}^K \pi_\ell^{(t-1)} \mathcal{N}_2\left(\widehat{\boldsymbol{\beta}}_i \,|\, \mathbf{0}, \mathbf{\Theta}_\ell^{(t-1)} + \hat{\mathbf{\Sigma}}_i\right)}.$$

2. Define new values $\boldsymbol{\pi}^{(t)}$ and $\mathbf{\Gamma}^{(t)}$ as maximizers of the expected log-likelihood function where the expectation is over the distribution of latent variables $\boldsymbol{z}$. Thus, the target function to be optimized is

$$\text{E}_{\boldsymbol{z}^{(t)}}\left(\mathcal{L}(\mathbf{\Gamma}, \boldsymbol{\pi}, \boldsymbol{z})\right) = \log(c) + \sum_{i=1}^n \sum_{k=1}^K \left[p_{ik}^{(t)} \log(\pi_k) + p_{ik}^{(t)} \log \mathcal{N}_2\left(\widehat{\boldsymbol{\beta}}_i \,|\, \mathbf{0}, \mathbf{\Theta}_k + \hat{\mathbf{\Sigma}}_i\right)\right].$$

   We can analytically maximize with respect to $\boldsymbol{\pi}$ by setting $\pi_k^{(t)} = \frac{1}{n} \sum_i p_{ik}^{(t)}$. For the remaining parameters in $\mathbf{\Gamma}$, we use numerical optimization of the above function.

The EM-algorithm is run until the difference in the maxmized log-likelihood between adjacent iterations drops below a given tolerance (`tol.loglk`), such as the default value of 0.001. The convergence can also be defined by a threshold (`tol.par`) on the relative change in parameter values between adjacent iterations. Note that if `tol.par` is negative, then the log-likelihood value alone defines the convergence.

References

Bhattacharjee S., Rajaraman P., Jacobs K.B., et al. 2012. A Subset-Based Approach Improves Power and Interpretation for the Combined Analysis of Genetic Association Studies of Heterogeneous Traits, *The American Journal of Human Genetics*, 90(5): 821-835.