libsemx Manuscript Draft

Deniz Akdemir and contributors

2025-12-29

Contents

0.1 Abstract

libsemx is a unified C++20 engine for structural equation modeling (SEM) and generalized linear mixed models (GLMM) with Python and R bindings. It couples a shared intermediate representation (ModelIR), flexible covariance structures (including genomic kernels), and support for non-Gaussian families and survival outcomes. We illustrate the modeling API, compare runtime/estimates against lme4 (Bates et al. 2015), sommer (Covarrubias-Pazaran 2016), and statsmodels (Seabold and Perktold 2010), and discuss when to prefer libsemx (SEM+mixed, non-Gaussian, custom covariance) versus specialized libraries. The project is openly developed on GitHub and intended for preprint submission (e.g., arXiv).

0.2 Introduction

libsemx is a unified C++20 engine for structural equation modeling (SEM) and generalized linear mixed models (GLMM), exposed through Python and R. It keeps a shared intermediate representation (ModelIR) for variables, paths, covariances, and random effects so both bindings produce identical likelihoods, gradients, and diagnostics.

Architecture at a glance

- Variables: observed, latent, grouping, exogenous; observed nodes carry families (gaussian, binomial, poisson, negative binomial, gamma, Weibull, exponential, log-normal, log-logistic, ordinal).
- Edges: loadings (`=~`), regressions (`~`), covariances (`~~`) tied to parameter IDs for constraints/equality.
- Covariances: unstructured, diagonal, compound symmetry, AR(1), Toeplitz, Kronecker, fixed/scaled fixed, multi-kernel simplex blends.
- Random effects: named effects referencing covariances; optional `lambda` shrinkage for ridge and genomic prediction.
- Estimation: ML/REML with L-BFGS or gradient descent, Average Information REML for variance components, Laplace for non-Gaussian random effects, spectral shortcuts for dense genomic kernels, FIML for missing data, SEM fit indices (^{2}, CFI, TLI, RMSEA, SRMR).

0.3 Example 1: Random slope LMM (sleepstudy, R)

Real data (`data/sleepstudy.csv`) with subject-specific intercepts and slopes:

```r
library(dplyr)
sleep_df <- readr::read_csv("data/sleepstudy.csv", show_col_types = FALSE) %>%
  select(-rownames) %>%
  mutate(Reaction = Reaction / 100)

model_lmm <- semx_model(
  equations = c("Reaction ~ Days + (Days | Subject)"),
  families = c(Reaction = "gaussian")
)

t_semx <- system.time({
  fit_lmm <- semx_fit(model_lmm, sleep_df, optimizer_name = "lbfgs")
})
fit_lmm_elapsed <- t_semx[["elapsed"]]

summary(fit_lmm)                 # fixed effects, variances, fit indices
```

```
#> Optimization converged: TRUE
#> Iterations: 129
#> Log-likelihood: -47.039
#> Chi-square: 106.078 (df=NA)
#> AIC: 106.1, BIC: 125.2
#>
#>                                 Estimate   Std.Error    z.value      P.value
#> beta_Reaction_on_Days        0.104672860 0.015022367  6.9678006 3.219425e-12
#> alpha_Reaction_on__intercept 2.514051048 0.066322768 37.9063046 0.000000e+00
#> psi_Reaction_Reaction        0.065494103 0.007718554  8.4852813 0.000000e+00
#> cov_re_1_0                   0.237805670 0.055773710  4.2637592 2.010161e-05
#> cov_re_1_1                   0.004648935 0.018279776  0.2543212 7.992474e-01
#> cov_re_1_2                   0.056979003 0.012291142  4.6357778 3.555979e-06
#>
#> Variance Components:
#>    Group      Name1 Name2    Variance    Std.Dev      Corr
#>  Subject _intercept       0.056551536 0.23780567        NA
#>  Subject _intercept  Days 0.001105543         NA 0.0813201
#>  Subject       Days       0.003268219 0.05716834        NA
```

```
semx_ranef(fit_lmm) %>% head()   # BLUPs per random-effect block
```

```
#> $re_Subject_1
#>       _intercept         Days
#> 308  0.028156841  0.090755338
#> 309 -0.400484878 -0.086440674
#> 310 -0.384331534 -0.055133789
#> 330  0.228322947 -0.046587502
#> 331  0.215499891 -0.029445199
#> 332  0.088155864 -0.002352091
#> 333  0.164419872 -0.001588237
#> 334 -0.069967202  0.010327362
#> 335 -0.010374193 -0.105994435
#> 337  0.346663149  0.086323772
#> 349 -0.245581581  0.010644008
#> 350 -0.123346311  0.064717035
#> 351  0.042740664 -0.029553436
#> 352  0.206222197  0.035617042
#> 369  0.032585362  0.008717103
#> 370 -0.247103332  0.046597354
#> 371  0.007232813 -0.009710559
#> 372  0.121189431  0.013106909
```

0.4 Example 1b: Same model via Python

The same IR is built and fit from Python using `reticulate`:

```
import pandas as pd
import numpy as np
from semx import Model

sleep_df = pd.read_csv("data/sleepstudy.csv")
sleep_df["Reaction"] = sleep_df["Reaction"] / 100.0
# Encode grouping as integer codes (required by C++ backend)
sleep_df["Subject"] = pd.Categorical(sleep_df["Subject"]).codes.astype(int)

# Explicit residual variance term and random slope
spec = Model(
    equations=[
        "Reaction ~ 1 + Days + (Days | Subject)",
        "Reaction ~~ Reaction",
    ],
    families={"Reaction": "gaussian", "Days": "gaussian"},
    kinds={"Subject": "grouping"},
)

fit = spec.fit(sleep_df)

param_map = dict(zip(fit.fit_result.parameter_names, fit.fit_result.optimization_result.parameters))
print("Parameters (name=value):")
```

```
#> Parameters (name=value):
```

```
for k, v in param_map.items():
    print(f"  {k}: {v:.6f}")
```

```
#>   beta_Reaction_on__intercept: 2.514051
#>   beta_Reaction_on_Days: 0.104673
#>   psi_Reaction_Reaction: 0.065494
#>   cov_re_1_0: 0.237806
#>   cov_re_1_1: 0.004649
#>   cov_re_1_2: 0.056979
```

```
# Manual BLUPs (pybind currently returns zeros for random_effects; compute directly)
beta = np.array([param_map["beta_Reaction_on__intercept"], param_map["beta_Reaction_on_Days"]])
sigma_e = param_map["psi_Reaction_Reaction"]
G = np.array([
    [param_map["cov_re_1_0"], param_map["cov_re_1_1"]],
    [param_map["cov_re_1_1"], param_map["cov_re_1_2"]],
])

blup_rows = []
for subj, df_g in sleep_df.groupby("Subject"):
    y = df_g["Reaction"].to_numpy()
    days = df_g["Days"].to_numpy()
    X = np.column_stack([np.ones_like(days), days])  # fixed + random design
    resid = y - X.dot(beta)
    V = X.dot(G).dot(X.T) + sigma_e * np.eye(len(y))
    u = G.dot(X.T).dot(np.linalg.solve(V, resid))
    blup_rows.append({"Subject": subj, "u_intercept": u[0], "u_days": u[1]})

blups = pd.DataFrame(blup_rows).sort_values("Subject")
print("\nBLUPs (first 6 subjects):")
```

```
#>
#> BLUPs (first 6 subjects):
```

```
print(blups.head(6))
```

```
#>    Subject  u_intercept    u_days
#> 0        0    -0.058670  0.110399
#> 1        1    -0.428320 -0.087291
#> 2        2    -0.440213 -0.049516
#> 3        3     0.344710 -0.068275
#> 4        4     0.310120 -0.046555
#> 5        5     0.116699 -0.007110
```

0.5 Example 2: CFA / SEM (BFI personality items, R)

Confirmatory factor analysis on the Big Five inventory (`data/bfi.csv`). Latent variances are fixed to 1 for scale identification.

```r
bfi_df <- readr::read_csv("data/bfi.csv", show_col_types = FALSE) %>%
  select(-rownames) %>%
  na.omit() %>%
  slice_head(n = 600)

item_cols <- c(paste0("A", 1:5), paste0("C", 1:5), paste0("E", 1:5), paste0("N", 1:5), paste0("O", 1:5)
bfi_df[item_cols] <- scale(bfi_df[item_cols])

equations <- c(
  "Agreeableness =~ NA*A1 + A2 + A3 + A4 + A5",
  "Conscientiousness =~ NA*C1 + C2 + C3 + C4 + C5",
  "Extraversion =~ NA*E1 + E2 + E3 + E4 + E5",
  "Neuroticism =~ NA*N1 + N2 + N3 + N4 + N5",
  "Openness =~ NA*O1 + O2 + O3 + O4 + O5",
  "Agreeableness ~~ 1*Agreeableness",
  "Conscientiousness ~~ 1*Conscientiousness",
  "Extraversion ~~ 1*Extraversion",
  "Neuroticism ~~ 1*Neuroticism",
  "Openness ~~ 1*Openness"
)

families <- setNames(rep("gaussian", length(item_cols)), item_cols)

# Mildly informative starting values to stabilize L-BFGS
init_params <- list()
for (eq in equations) {
  if (grepl("=~", eq)) {
    parts <- strsplit(eq, "=~")[[1]]
    latent <- trimws(parts[1])
    inds <- trimws(strsplit(parts[2], "\\+")[[1]])
    for (ind_raw in inds) {
      ind <- gsub("NA\\*", "", trimws(ind_raw))
      init_params[[paste0("lambda_", ind, "_on_", latent)]] <- 0.5
      init_params[[paste0("psi_", ind, "_", ind)]] <- 0.5
    }
  }
}

model_cfa <- semx_model(
  equations = equations,
  families = families,
  parameters = init_params
)

fit_cfa <- semx_fit(model_cfa, bfi_df, optimizer_name = "lbfgs")
summary(fit_cfa)           # loadings, residual variances, fit indices
```

```
#> Optimization converged: TRUE
#> Iterations: 34
#> Log-likelihood: -19816.803
#> Chi-square: 39733.606 (df=300)
#> P-value: 0.000
#> CFI: 0.671, TLI: 0.671, RMSEA: 0.089, SRMR: 0.142
#> AIC: 39733.6, BIC: 39953.5
#>
#>                                   Estimate  Std.Error    z.value      P.value
#> lambda_A1_on_Agreeableness      -0.3363039 0.04668813  -7.203200 5.881962e-13
#> lambda_A2_on_Agreeableness       0.6610674 0.04379622  15.094167 0.000000e+00
#> lambda_A3_on_Agreeableness       0.7153662 0.04306594  16.610952 0.000000e+00
#> lambda_A4_on_Agreeableness       0.4863282 0.04485702  10.841742 0.000000e+00
#> lambda_A5_on_Agreeableness       0.6278129 0.04361900  14.393107 0.000000e+00
#> lambda_C1_on_Conscientiousness   0.4821873 0.04616018  10.445958 0.000000e+00
#> lambda_C2_on_Conscientiousness   0.6434928 0.04560025  14.111606 0.000000e+00
#> lambda_C3_on_Conscientiousness   0.5688728 0.04560263  12.474562 0.000000e+00
#> lambda_C4_on_Conscientiousness  -0.6281092 0.04603912 -13.642946 0.000000e+00
#> lambda_C5_on_Conscientiousness  -0.5268256 0.04667944 -11.286031 0.000000e+00
#> lambda_E1_on_Extraversion       -0.5969958 0.04274728 -13.965701 0.000000e+00
#> lambda_E2_on_Extraversion       -0.7148705 0.04177410 -17.112767 0.000000e+00
#> lambda_E3_on_Extraversion        0.6008709 0.04329340  13.879042 0.000000e+00
#> lambda_E4_on_Extraversion        0.6412587 0.04212011  15.224525 0.000000e+00
#> lambda_E5_on_Extraversion        0.5474789 0.04359838  12.557322 0.000000e+00
#> lambda_N1_on_Neuroticism         0.8109395 0.03619465  22.404956 0.000000e+00
#> lambda_N2_on_Neuroticism         0.8346101 0.03587671  23.263286 0.000000e+00
#> lambda_N3_on_Neuroticism         0.7297904 0.03820746  19.100730 0.000000e+00
#> lambda_N4_on_Neuroticism         0.5120498 0.04198083  12.197227 0.000000e+00
#> lambda_N5_on_Neuroticism         0.5223308 0.04120977  12.674927 0.000000e+00
#> lambda_O1_on_Openness            0.5377736 0.04733319  11.361448 0.000000e+00
#> lambda_O2_on_Openness           -0.4039174 0.04914854  -8.218298 2.220446e-16
#> lambda_O3_on_Openness            0.7047112 0.05009588  14.067250 0.000000e+00
#> lambda_O4_on_Openness            0.3442046 0.04870034   7.067807 1.574074e-12
#> lambda_O5_on_Openness           -0.5396297 0.04986444 -10.821935 0.000000e+00
#> psi_A1_A1                        0.8852330 0.05374155  16.472040 0.000000e+00
#> psi_A2_A2                        0.5613232 0.04616556  12.158917 0.000000e+00
#> psi_A3_A3                        0.4865845 0.04530684  10.739758 0.000000e+00
#> psi_A4_A4                        0.7618182 0.04950252  15.389482 0.000000e+00
#> psi_A5_A5                        0.6041843 0.04594582  13.149931 0.000000e+00
#> psi_C1_C1                        0.7658288 0.05068899  15.108383 0.000000e+00
#> psi_C2_C2                        0.5842504 0.04896502  11.931996 0.000000e+00
#> psi_C3_C3                        0.6747170 0.04903766  13.759160 0.000000e+00
#> psi_C4_C4                        0.6038121 0.04953052  12.190709 0.000000e+00
#> psi_C5_C5                        0.7207881 0.05060029  14.244743 0.000000e+00
#> psi_E1_E1                        0.6419294 0.04505546  14.247538 0.000000e+00
#> psi_E2_E2                        0.4872935 0.04275529  11.397268 0.000000e+00
#> psi_E3_E3                        0.6372875 0.04574054  13.932665 0.000000e+00
#> psi_E4_E4                        0.5871206 0.04351983  13.490877 0.000000e+00
#> psi_E5_E5                        0.6986002 0.04701422  14.859337 0.000000e+00
#> psi_N1_N1                        0.3407105 0.02996233  11.371297 0.000000e+00
#> psi_N2_N2                        0.3017593 0.02951567  10.223697 0.000000e+00
#> psi_N3_N3                        0.4657393 0.03512550  13.259293 0.000000e+00
#> psi_N4_N4                        0.7361384 0.04624708  15.917510 0.000000e+00
#> psi_N5_N5                        0.7255039 0.04516783  16.062403 0.000000e+00
#> psi_O1_O1                        0.7091329 0.05120514  13.848861 0.000000e+00
#> psi_O2_O2                        0.8351841 0.05389234  15.497268 0.000000e+00
#> psi_O3_O3                        0.5017154 0.05780255   8.679815 0.000000e+00
#> psi_O4_O4                        0.8798565 0.05443095  16.164635 0.000000e+00
#> psi_O5_O5                        0.7071331 0.05392201  13.113997 0.000000e+00
```

```
# Path diagram (optional; not exported in NAMESPACE, use :::)
# semx:::semx_plot_path(fit_cfa)
```

0.6 Example 3: Survival regression (ovarian, R)

Weibull survival with censoring (`data/ovarian_survival.csv`), fixed effects on age and treatment arm:

```
ovarian <- readr::read_csv("data/ovarian_survival.csv", show_col_types = FALSE) %>%
  select(-rownames)

surv_model <- semx_model(
  equations = c("Surv(futime, fustat) ~ age + rx + ecog.ps"),
  families = c(futime = "weibull", age = "gaussian", rx = "gaussian", ecog.ps = "gaussian")
)

fit_surv <- semx_fit(
  surv_model,
  ovarian,
  options = list(max_iterations = 300, tolerance = 1e-5, force_laplace = TRUE),
  optimizer_name = "lbfgs"
)

summary(fit_surv)   # regression coefficients, scale/shape
```

```
#> Optimization converged: TRUE
#> Iterations: 6
#> Log-likelihood: -327.766
#> Chi-square: 671.532 (df=6)
#> P-value: 0.000
#> CFI: -0.885, TLI: -0.885, RMSEA: 0.394, SRMR: 10.911
#> AIC: 671.5, BIC: 681.6
#>
#>                                Estimate    Std.Error    z.value      P.value
#> beta_futime_on_age          -0.07965424   0.01999171 -3.9843643 6.766100e-05
#> beta_futime_on_rx            0.56114462   0.34087855  1.6461717 9.972842e-02
#> beta_futime_on_ecog.ps       0.06019814   0.33114847  0.1817859 8.557507e-01
#> alpha_futime_on__intercept  10.40851489   1.46333168  7.1128884 1.136424e-12
#> psi_futime_futime            1.82751637   0.43298116  4.2207757 2.434631e-05
#> psi_age_age               3252.65051220 902.12294613  3.6055512 3.114910e-04
#> psi_rx_rx                    2.49999983   0.69337517  3.6055514 3.114908e-04
#> psi_ecog.ps_ecog.ps          2.38461539   0.66137331  3.6055513 3.114910e-04
```

```
# Predict survival at selected times
semx_predict_survival(fit_surv, newdata = ovarian[1:3, ], times = c(100, 300, 500), outcome = "futime")
```

```
#>         100        300          500
#> 1 0.7425899 0.10902957 0.0035644391
#> 2 0.6651962 0.04804094 0.0004432768
#> 3 0.8926935 0.42944693 0.1164941986
```

0.7 Example 4: Genomic mixed model (Maize Diversity Panel, R)

Single-kernel GBLUP on standardized height (`EarHT`) using SNP markers (`data/mdp_numeric.csv`, `data/mdp_traits.csv`). To keep knitting fast, a small marker subset is used; increase `p_subset` for fuller analyses.

```r
library(Matrix)

numeric_df <- readr::read_csv("data/mdp_numeric.csv", show_col_types = FALSE)
traits_df <- readr::read_csv("data/mdp_traits.csv", show_col_types = FALSE) %>%
  rename(taxa = Taxa)

merged_df <- numeric_df %>%
  inner_join(traits_df, by = c("taxa" = "taxa")) %>%
  filter(!is.na(EarHT))

marker_cols <- setdiff(names(numeric_df), "taxa")
p_subset <- 300
marker_cols <- marker_cols[seq_len(min(p_subset, length(marker_cols)))]

M <- as.matrix(merged_df[, marker_cols])
merged_df$EarHT_std <- scale(merged_df$EarHT)
merged_df$all_groups <- 1  # single grouping indicator for random effect

model_gblup <- semx_model(
  equations = c("EarHT_std ~ 1"),
  families = c(EarHT_std = "gaussian"),
  genomic = list(polygenic = list(markers = M, structure = "grm")),
  random_effects = list(
    list(name = "u", variables = c("all_groups"), covariance = "polygenic")
  )
)

fit_gblup <- semx_fit(model_gblup, merged_df, optimizer_name = "lbfgs")
summary(fit_gblup)                   # genetic vs residual variance
```

```
#> Optimization converged: TRUE
#> Iterations: 8
#> Log-likelihood: -365.835
#> Chi-square: 737.670 (df=NA)
#> AIC: 737.7, BIC: 748.6
#>
#>                                   Estimate  Std.Error      z.value      P.value
#> alpha_EarHT_std_on__intercept 3.242287e-10 0.04460759 7.268465e-09 1.000000e+00
#> psi_EarHT_std_EarHT_std       5.551634e-01 0.06747569 8.227607e+00 2.220446e-16
#> polygenic_0                   3.819819e-01 0.10538957 3.624475e+00 2.895486e-04
#>
#> Variance Components:
#>       Group       Name1 Name2  Variance   Std.Dev Corr
#>  all_groups (Intercept)       0.3819829 0.6180476   NA
```

```r
# Extract variance components and heritability (h2)
param_names <- fit_gblup$parameter_names
```

```r
if (is.null(param_names) || length(param_names) == 0) {
  # Fallback to IR parameter ids
  param_names <- fit_gblup$model$ir$parameter_ids()
}
params <- setNames(fit_gblup$optimization_result$parameters, param_names)

genetic_key <- "polygenic_0"
if (is.null(params[[genetic_key]])) {
  poly_keys <- grep("^polygenic", names(params), value = TRUE)
  if (length(poly_keys)) genetic_key <- poly_keys[[1]]
}
resid_key <- "psi_EarHT_std_EarHT_std"

genetic_var <- params[[genetic_key]]
resid_var <- params[[resid_key]]

if (!is.null(genetic_var) && !is.null(resid_var)) {
  h2 <- genetic_var / (genetic_var + resid_var)
  vc_tbl <- data.frame(
    component = c("Genetic (GRM)", "Residual"),
    variance = c(genetic_var, resid_var),
    proportion = c(genetic_var, resid_var) / (genetic_var + resid_var)
  )
  print(vc_tbl)
  cat(sprintf("Narrow-sense heritability (h2): %.3f\n", h2))
} else {
  cat(
    "Could not locate variance components to compute h2; available parameters:\n",
    paste(names(params), collapse = ", "),
    "\n"
  )
}
```

```
#>       component  variance proportion
#> 1 Genetic (GRM) 0.3819819  0.4076015
#> 2      Residual 0.5551634  0.5923985
#> Narrow-sense heritability (h2): 0.408
```

0.8 Comparisons to lme4 and sommer (runtime + variance components)

Small side-by-side fits on shared datasets; guarded by `requireNamespace()` to keep knitting robust.

```r
library(tibble)

compare_rows <- list()

# libsemx (sleepstudy random slopes; already fit above)
if (exists("fit_lmm")) {
  pe <- fit_lmm$optimization_result$parameters
  pn <- fit_lmm$parameter_names
  if (!is.null(pn) && length(pn) == length(pe)) {
    pe <- setNames(pe, pn)
  }
  beta0 <- pe[["alpha_Reaction_on__intercept"]] %||% pe[["beta_Reaction_on__intercept"]] %||% pe[[1]]
  beta1 <- pe[["beta_Reaction_on_Days"]] %||% pe[[2]]
  # Random-effect Cholesky diagonals -> variances
  var_intercept <- pe[["cov_re_1_0"]] %||% pe[[length(pe) - 2]]
  var_slope <- pe[["cov_re_1_2"]] %||% pe[[length(pe) - 0]]
  if (!is.null(names(pe))) {
    if ("cov_re_1_0" %in% names(pe)) var_intercept <- var_intercept^2
    if ("cov_re_1_2" %in% names(pe)) var_slope <- var_slope^2
  }
  compare_rows[["libsemx_sleepstudy"]] <- tibble(
    package = "libsemx",
    model = "sleepstudy",
    beta0 = beta0,
    beta1 = beta1,
    var_intercept = var_intercept,
    var_slope = var_slope,
    elapsed = fit_lmm_elapsed %||% NA_real_
  )
}

# lme4 on sleepstudy
if (requireNamespace("lme4", quietly = TRUE)) {
  t_lme4 <- system.time({
    lme4_fit <- lme4::lmer(Reaction ~ Days + (Days | Subject), data = sleep_df)
  })
  vc_mat <- as.matrix(lme4::VarCorr(lme4_fit)$Subject)
  intercept_var <- vc_mat[1, 1]
  slope_var <- vc_mat[2, 2]
  compare_rows[["lme4_sleepstudy"]] <- tibble(
    package = "lme4",
    model = "sleepstudy",
    beta0 = lme4::fixef(lme4_fit)[["(Intercept)"]],
    beta1 = lme4::fixef(lme4_fit)[["Days"]],
    var_intercept = intercept_var,
    var_slope = slope_var,
    elapsed = t_lme4[["elapsed"]]
  )
}

# sommer on MDP GBLUP (same marker subset as above)
if (requireNamespace("sommer", quietly = TRUE)) {
  Gmat <- tcrossprod(scale(M)) / ncol(M)
  rownames(Gmat) <- merged_df$taxa
  colnames(Gmat) <- merged_df$taxa
  mdp_dat <- merged_df %>% mutate(taxa = factor(taxa))
  sommer_res <- tryCatch({
    t_sommer <- system.time({
      sommer_fit <- sommer::mmer(
        EarHT_std ~ 1,
        random = ~ sommer::vsr(taxa, Gu = Gmat),
        data = mdp_dat
      )
    })
    list(fit = sommer_fit, elapsed = t_sommer[["elapsed"]])
  }, error = function(e) {
    message("sommer failed: ", conditionMessage(e))
    NULL
  })
  if (!is.null(sommer_res)) {
    vc_sommer <- sommer::summary.mmer(sommer_res$fit)$varcomp
    compare_rows[["sommer_mdp"]] <- tibble(
      package = "sommer",
      model = "mdp_gblup",
      beta0 = sommer_res$fit$Beta[1, 1],
      beta1 = NA_real_,
      var_intercept = vc_sommer["u:taxa", "VarComp"],
      var_slope = NA_real_,
      elapsed = sommer_res$elapsed
    )
  }
}

if (length(compare_rows)) {
  dplyr::bind_rows(compare_rows)
} else {
  message("No comparison packages available.")
}
```

```
#> # A tibble: 2 x 7
#>   package model      beta0 beta1 var_intercept var_slope elapsed
#>   <chr>   <chr>      <dbl> <dbl>         <dbl>     <dbl>   <dbl>
#> 1 libsemx sleepstudy  2.51 0.105        0.0566   0.00325   1.36
#> 2 lme4    sleepstudy  2.51 0.105        0.0612   0.00351   0.463
```

0.9 Comparisons to related libraries

Library	Scope	Families	Covariance/kernels	SEM support	Notes
libsemx	SEM + GLMM unified	Gaussian, GLM, survival, ordinal	Unstructured, CS, AR(1), Toeplitz, Kronecker, genomic/multi-kernel	Yes	C++ core, Python/R front-ends, spectral short-cuts, FIML

Library	Scope	Families	Covariance/kernels	SEM support	Notes
statsmodels MixedLM	Linear mixed models	Mostly Gaussian	Random intercept/slope, limited structures	No	No Laplace for non-Gaussian; no latent variables
lme4	GLMM (R)	Gaussian, binomial, Poisson (others via glmer)	Random effects with simple variance components	Limited	No SEM paths or fit indices; no genomic kernels
nlme	Linear mixed models (R)	Gaussian	Correlation/covariance structures, no kernels	No	Older API, limited non-Gaussian support
lavaan	SEM (R)	Mostly Gaussian	Latent covariance structures	Yes	No mixed-model random effects or GLMM families
sommer	Mixed models with kernels (R)	Gaussian	Genomic kernels, multi-trait	No	Genomic focus; limited non-Gaussian families

0.10 Reproducibility and next steps

- Data live in `data/` and examples mirror `docs/examples/libsemx_starter.Rmd` and `docs/examples/mdp_analysis.Rm`
- Python examples: `python/examples/shrinkage_example.py`, `python/examples/crossed_effects_example.py`.
- R example: `Rpkg/semx/examples/shrinkage_example.R`.

Planned manuscript additions: benchmark tables against lme4/nlme/lavaan/sommer/statsmodels, multi-group invariance SEM, competing risks survival, and multi-kernel simplex genomic prediction with real kernels.

0.11 Why libsemx (and why it can be slower on tiny models)

- **General engine vs. special-case speed:** lme4/nlme have hand-optimized Gaussian REML/ML paths; they are very fast on small linear mixed models. libsemx runs a unified L-BFGS/Laplace/FIML stack that can handle non-Gaussian families, latent variables, survival, and custom covariances. That generality adds startup/solver overhead on toy problems (sleepstudy), but becomes advantageous as models get richer (ordinal, survival, multi-kernel, SEM).
- **Unified IR and cross-language parity:** Python and R bindings share the same ModelIR and likelihood driver, so estimates/gradients/fit indices match across languages and with the C++ core. This reduces drift between front-ends.
- **Non-Gaussian and survival support:** Built-in Laplace approximation, competing risks/survival families, and dispersion handling go beyond what lme4/nlme provide.
- **Flexible covariances and genomics:** Supports AR(1)/Toeplitz/Kronecker, multi-kernel simplex blends, and spectral shortcuts for dense kernels; integrates genomic prediction workflows out of the box.
- **SEM + mixed models together:** lavaan-style SEM paths with mixed-model random effects and GLMM families, plus SEM fit indices and modification indices.
- **Diagnostics and post-estimation:** Standard errors, BLUPs, fit indices, and (planned) exported Hessians/standardization stay consistent across languages.

When to pick libsemx

- You need SEM + mixed effects in one model (latent factors plus random effects).
- You have non-Gaussian outcomes (binomial, Poisson/neg-bin, survival, ordinal) with random effects.
- You need flexible or custom covariance structures (multi-kernel, Kronecker, AR/Toeplitz).
- You want reproducible parity between Python and R for the same analysis.

When lme4/nlme might be faster

- Small Gaussian LMMs with simple random structures where lme4's specialized code path dominates. For these, you can lower libsemx tolerances/iterations (`max_iterations = 50`, `tolerance = 1e-4`) to reduce overhead, but lme4 will likely remain faster on tiny datasets. On larger or more complex models, the gap narrows because likelihood evaluation—not setup overhead—dominates runtime.

0.12 Interpreting fit indices and degrees of freedom

- Chi-square/df/CFI/TLI/RMSEA/SRMR are computed only when SEM sample statistics are available (latent-variable models or multi-group SEM). GLMM-only fits (e.g., survival, GBLUP, simple mixed models) leave `df` as `NA` and the chi-square carries no SEM meaning there. Use AIC/BIC/variance components/BLUPs for mixed models; use SEM indices only for latent-variable analyses with covariance structure.

Bates, Douglas, Martin Mächler, Ben Bolker, and Steve Walker. 2015. "Fitting Linear Mixed-Effects Models Using Lme4." *Journal of Statistical Software* 67 (1): 1–48. https://doi.org/10.18637/jss.v067.i01.

Covarrubias-Pazaran, Giovanny. 2016. "Genome-Assisted Prediction of Quantitative Traits Using the r Package Sommer." *PLOS ONE* 11 (6): e0156744. https://doi.org/10.1371/journal.pone.0156744.

Seabold, Skipper, and Josef Perktold. 2010. "Statsmodels: Econometric and Statistical Modeling with Python." In *9th Python in Science Conference*.