Appendix to Bayesian regularized SEM: Current capabilities and constraints

Sara van Erp

This appendix summarizes the functions and calls needed to use the different software packages for (Bayesian) regularized SEM that are compared in the manuscript. This appendix is meant as a quick reference for applying these methods. Please see the documentation for each package for all functionality. For reproducing the original empirical example, please review the complete code available on Github

Packages

```
library(lslx)
library(blavaan)
library(LAWBL)
library(rstan)
library(shinystan)
```

Data

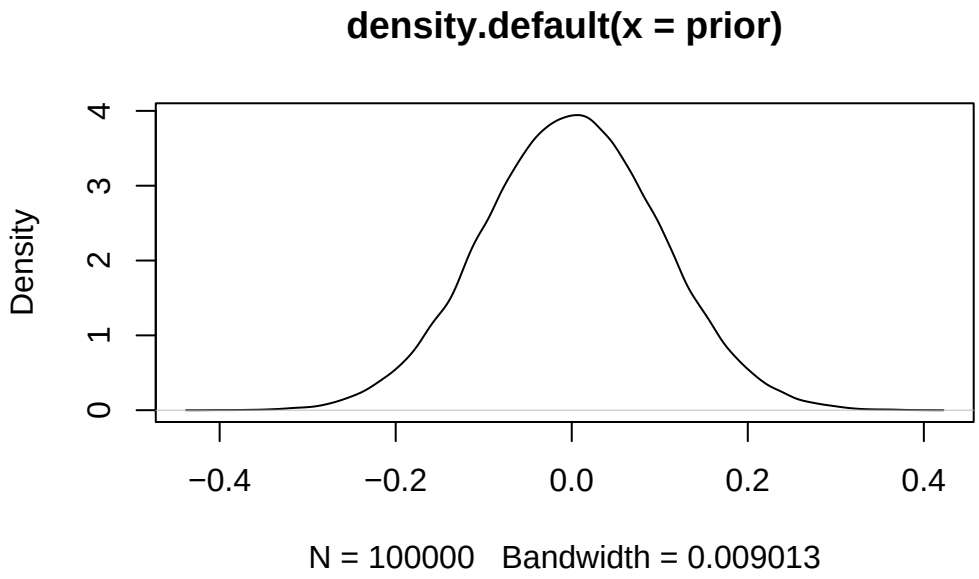
Before running (Bayesian) regularization, make sure the data is scaled so that the penalization affects each variable to the same extent.

```
dat <- scale(df, center = TRUE, scale = TRUE)
```

Prior visualization

You can visualize the prior densities by sampling from the corresponding distribution in `R`. For example, to visualize a normal or ridge shrinkage prior with a standard deviation of 0.1:

```r
prior <- rnorm(100000, mean = 0, sd = 0.1)
plot(density(prior))
```



To recreate the prior visualization in the manuscript, please see the code in *densityplots.R*.

Analysis with `lslx`

To run `lslx`, first a model needs to be specified which can be done using `lavaan` syntax. The only difference with regular `lavaan` syntax is that the user needs to specified which parameters are regularized. The code below specifies the factor model used in the empirical application.

```r
mod <- paste(c(paste("F1 =~ y", 1:18, " \n"), # main loadings: freely estimated
               paste("pen() * F1 =~ y", 19:36," \n"), # cross-loadings: penalized
               paste("F2 =~ y", 19:24," \n"),
               paste("pen() * F2 =~ y", 1:18," \n"),
               paste("pen() * F2 =~ y", 25:36," \n"),
               paste("F3 =~ y", 25:36," \n"),
               paste("pen() * F3 =~ y", 1:24," \n"),
               "F1  ~~  1 * F1 \n", # fix latent variances for identification
               "F2  ~~ 1 * F2 \n",
               "F3  ~~ 1 * F3")
)
```

Next, the model can be fit using different penalties, e.g., the lasso, elastic_net, or minimax concave penalty (mcp). In addition, a custom grid for cross-validation can be specified for the λ and δ parameters through `lambda_grid` and `delta_grid`.

```
fit <- plsem(model = mod, data = dat, penalty_method = "lasso")
```

The following function can be used to extract the results, where the `selector` argument is used to indicate which information criterion should be used to select the optimal penalty level.

```
fit$summarize(selector = "bic")
```

Analysis with `LAWBL`

To run the model with `LAWBL`, a design matrix first needs to be specified indicating which parameters are regularized (-1) or specified (1). In the empirical application, cross-loadings are regularized while main loadings are specified.

```
Q <- matrix(-1, nrow = ncol(dat), ncol = 3) # -1 for unspecified (regularized) and 1 for s
f1 <- which(colnames(dat) %in% paste0("y", 1:18))
f2 <- which(colnames(dat) %in% paste0("y", 19:24))
f3 <- which(colnames(dat) %in% paste0("y", 25:36))
Q[f1, 1] <- Q[f2, 2] <- Q[f3, 3] <- 1
```

Next, the model can be run using the `pcfa` function. The `LD` argument specifies whether local dependence is true or not. Here, we specify no local dependence, so no (regularized) residual covariances between items. `cati` can be used in case of categorical items. Since we are running a Bayesian analysis, we need to specify sufficient iterations to ensure convergence of the posterior distribution. `burn` specifies the number of burn-in iterations that are used to move away from the initial values and will not be included in the results, while `iter` specifies the actual number of posterior simulations.

```
fit <- pcfa(dat = dat,
            Q = Q,
            LD = FALSE,
            cati = NULL,
            burn = 2000,
            iter = 5000)
```

Usually, in a Bayesian analysis we would run multiple chains starting from different values and assess convergence by ensuring the chains coincide after burn-in. Unfortunately, the `pcfa` function does not include an argument to specify the number of chains so only one chain is

run. Multiple chains can be run by changing the random seed via the `rseed` argument and subsequently combining the posterior draws to assess convergence.

`LAWBL` does provide plotting functions specifically for the eigenvalues of the factors. You can plot the trace, density, or estimated potential scale reduction (EPSR):

```
plot_lawbl(fit, what = "trace")
plot_lawbl(fit, what = "density")
plot_lawbl(fit, what = "EPSR")
```

There is also an `auto_stop` argument based on the EPSR, but such automatic criteria are not recommended. Instead, it is better to run the analysis with a specified number of iterations and rerun with, for example, double the number of iterations to ensure the results remain the same.

Results can be summarized with the `summary` function. See `?summary.lawbl` for the different types of summaries that can be obtained. For example, to get the loadings you can call:

```
summary(fit, what = "lambda")
```

Analysis with `blavaan`

When running the analysis in `blavaan`, we need to specify a model using `lavaan` syntax indicating which priors to use for which parameters. The code below specifies the model from the empirical application with a normal shrinkage prior with variance equal to 0.01 for the cross-loadings and a normal prior with standard deviation equal to 10 for the main loadings. Note that it is not possible to automatically change the distributional form of the prior on the loadings, this could be done by manually adapting the underlying Stan file (using `mcmcfile = TRUE`).

```
priorvar <- 0.01
priorsd <- sqrt(priorvar)
priorCrossL <- paste0("normal(0,", priorsd, ")")
priorMainL <- "normal(0,10)"

mod <- paste(c(paste0("F1 =~ prior(\"", priorMainL, "\")*", c(paste0("y", 1:18))," \n"),
                       paste0("F1 =~ prior(\"", priorCrossL, "\")*", c(paste0("y", 19:36)),
                       paste0("F2 =~ prior(\"", priorMainL, "\")*", c(paste0("y", 19:24))),"
                       paste0("F2 =~ prior(\"", priorCrossL, "\")*", c(paste0("y", 1:18)),"
                       paste0("F2 =~ prior(\"", priorCrossL, "\")*", c(paste0("y", 25:36)),
                       paste0("F3 =~ prior(\"", priorMainL, "\")*", c(paste0("y", 25:36)),"
                       paste0("F3 =~ prior(\"", priorCrossL, "\")*", c(paste0("y", 1:24))))
```

We can run the model for a specified number of MCMC chains and samples and extract the summary. This summary automatically reports the `Rhat` value for each parameter, which should be close to 1. `blavaan` will issue warnings if the model has certainly not converged based on `Rhat`. The effective sample size can be obtained as well. Finally, visual assessment of convergence can be achieved via the `plot.blavaan` function which relies on `bayesplot`. See the `bayesplot` documentation for the available plot types.

```
fit <- bcfa(mod, data = dat, n.chains = 4, burnin = 2000, sample = 5000, target = "mcmc")
summary(fit)
blavInspect(fit, 'neff')
plot(fit, pars = 1:3, plot.type = "trace")
plot(fit, pars = 1:3, plot.type = "areas")
```

Analysis with `rstan`

To run the analysis directly with `rstan`, a Stan model needs to be specified. The Stan models used for the empirical applications are available in the *code/stanmodels* directory on Github. Note that these models are specific for the empirical application and should be adapted manually for other models.

To run the model, we need to specify the data corresponding to the first block of the Stan model. With large models, it is also useful to specify a subset of parameters of interest to store in the results. Here, we focus on parameters from the `generated_quantities` block instead of original parameters, because these are adapted to solve sign-switching problems. We can then extract a summary, which includes numerical convergence diagnostics, or we can use `shinystan` to visualize the model and its convergence.

```
standat <- list(N = nrow(dat),
                P = ncol(dat),
                Q = 3,
                y = dat,
                nC = 72)
stanmod <- stan_model("./code/stanmodels/adapt_ridge.stan")
parsel <- c("psi", "L_main_C", "L_cross_C", "phi_C", "sd0")
fit <- sampling(stanmod, data = standat, chains = 4, iter = 5000, pars = parsel)
summary(fit)$summary
launch_shinystan(fit)
```

Results

To recreate the plots containing the results for the empirical application shown in the manuscript, see the file *adapt_results.R* on Github.

Original computing environment

This is the computing environment that was used to create this document as well as run the empirical application in the manuscript.

```
devtools::session_info()
```