Bayesian Hierarchical Factor Model Tutorial

Mahbod Mehrvarz

August 2025

This is a tutorial document to accompany *Hierarchical Factor Models For Analysis in Experimental Designs* by Rouder, Mehrvarz, and Stevenson. It shows the steps needed to fit the hierarchical model:

Overview

Goal: Fit a simple factor model to the **illusions** dataset using JAGS, then summarize and visualize the factor structure. You will:

1. load and lightly standardize the data
2. specify a Bayesian factor model in JAGS
3. impose simple identification constraints
4. run MCMC and extract posterior samples
5. align and standardize loadings
6. tabulate and plot variance decomposition

Prerequisites: You need a working JAGS installation on your machine and the following R packages: `R2jags`, `abind`, `infinitefactor`, `rlist`, `RColorBrewer`.

```
set.seed(42)
options(width = 90)

# install if missing (no output if already installed)
need = c("R2jags", "abind", "infinitefactor", "rlist", "RColorBrewer")
to_install = setdiff(need, rownames(installed.packages()))
if (length(to_install) > 0) install.packages(to_install, quiet = TRUE)

library(R2jags)
library(abind)
library(infinitefactor)
library(rlist)
library(RColorBrewer)
```

Data

The Mehrvarz et al (2025) illusions data set is available online at `GitHub`. The following reads it into `R`.

```
url1 = "https://raw.githubusercontent.com/specl/"
url2 = "ind-illusions/refs/heads/public/_data/clean_data.csv"
url = paste0(url1, url2)
dat = read.csv(url)
```

The following data-processing steps create a **task** × **version** variable and standardize scores:

```
dat$S  = as.numeric(factor(dat$sub))   # subject id
dat$J  = as.numeric(factor(dat$task))  # task id (1..5)
dat$JV = (dat$J - 1) * 2 + dat$version # task×version id (1..10)

# variance-normalize by task×version
v1 = tapply(dat$y, list(dat$S, dat$JV), var)
divisor = sqrt(apply(v1, 2, mean))
dat$yc = dat$y / divisor[dat$JV]

I = length(unique(dat$S))   # subjects
J = length(unique(dat$JV))  # task×version = 10
D = 7                       # 5 task factors + 2 general factors
N = nrow(dat)

c(I = I, J = J, D = D, N = N)
```

```
##     I     J     D     N
##   138    10     7 20547
```

Model

Latent-variable measurement model per observation n with person i, task×version j, and replicate k:

$$
\begin{aligned}
y_{ijk} \mid \theta_{ij},\, \tau_j^2 &\sim \mathcal{N}\Big(\theta_{ij},\, \tau_j^2\Big) \\
\theta_{ij} \mid \boldsymbol{\eta}_i,\, \mu_j,\, \boldsymbol{\lambda}_j,\, \delta_j^2 &\sim \mathcal{N}\Big(\mu_j + \boldsymbol{\lambda}_j' \boldsymbol{\eta}_i,\, \delta_j^2\Big) \\
\boldsymbol{\eta}_i &\sim \mathcal{N}_D(\mathbf{0},\, \mathbf{I}_D) \\
\mu_j &\sim \mathcal{N}(\mu_m,\, \mu_s^2) \\
\tau_j^{-2} &\sim \text{Gamma}(0.5,\, 0.5) \\
\delta_j^{-2} &\sim \text{Gamma}\Big(0.5,\, 0.5\,\text{tuneDelta}^2\Big) \\
\lambda_{jd}^{+} &\sim \mathcal{N}^{+}\Big(0,\, \text{tuneLambda}^2\Big) \\
\lambda_{jd}^{*} &\sim \mathcal{N}\Big(0,\, \text{tuneLambda}^2\Big)
\end{aligned}
$$

Priors: weakly informative normals on loadings and means; gammas on precisions. identification uses simple **constraint matrices**: each of the 5 task factors loads on its two versions, and two general factors are free across all tasks. we also use half-normal truncation for "positive" loadings where desired.

```
mod_bhfm = "
model {
  # priors on task-level params
  for (j in 1:J) {
    pTau2[j] ~ dgamma(0.5, 0.5)
    mu[j]    ~ dnorm(mu.m, pow(mu.s, -2))
    pDel2[j] ~ dgamma(0.5, 0.5 * pow(tuneDelta, 2))
    del2[j]  <- 1 / pDel2[j]

    for (d in 1:D) {
      lambda_pos[j,d]  ~ dnorm(0, pow(tuneLambda, -2)) T(0,)    # half-normal
      lambda_free[j,d] ~ dnorm(0, pow(tuneLambda, -2))          # normal
      aux_pos[j,d] <- constraint_zero[j,d] - constraint_pos[j,d]
      lambda[j,d] <- constraint_pos[j,d]*lambda_pos[j,d] + aux_pos[j,d]*lambda_free[j,d]
    }
  }

  # factor scores
  for (i in 1:I) {
    for (d in 1:D) {
      eta[i,d] ~ dnorm(0, 1)
    }
  }

  # latent effects
  for (i in 1:I) {
    for (j in 1:J) {
```

```
      center_theta[i,j] <- mu[j] + inprod(lambda[j,1:D], eta[i,1:D])
      theta[i,j] ~ dnorm(center_theta[i,j], pDel2[j])
    }
  }

  # likelihood
  for (n in 1:N) {
    center[n] <- theta[sub[n], task[n]]
    y[n] ~ dnorm(center[n], pTau2[task[n]])
  }
}
"
```

Identification constraints

Rows correspond to the 10 task×version indicators. first 5 columns: **task factors** (each loads on its two versions). last 2 columns: **general factors** (free on all).

```
constraint_matrix_zero = matrix(0, nrow = 10, ncol = 7)
for (i in 1:5) constraint_matrix_zero[(i*2-1):(i*2), i] = 1
constraint_matrix_zero[, 6:7] = 1

constraint_matrix_pos = matrix(0, nrow = 10, ncol = 7)
for (i in 1:5) constraint_matrix_pos[(i*2-1), i] = 1

constraint_matrix_zero
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7]
##  [1,]    1    0    0    0    0    1    1
##  [2,]    1    0    0    0    0    1    1
##  [3,]    0    1    0    0    0    1    1
##  [4,]    0    1    0    0    0    1    1
##  [5,]    0    0    1    0    0    1    1
##  [6,]    0    0    1    0    0    1    1
##  [7,]    0    0    0    1    0    1    1
##  [8,]    0    0    0    1    0    1    1
##  [9,]    0    0    0    0    1    1    1
## [10,]    0    0    0    0    1    1    1
```

```
constraint_matrix_pos
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7]
```

```
##  [1,]    1    0    0    0    0    0    0
##  [2,]    0    0    0    0    0    0    0
##  [3,]    0    1    0    0    0    0    0
##  [4,]    0    0    0    0    0    0    0
##  [5,]    0    0    1    0    0    0    0
##  [6,]    0    0    0    0    0    0    0
##  [7,]    0    0    0    1    0    0    0
##  [8,]    0    0    0    0    0    0    0
##  [9,]    0    0    0    0    1    0    0
## [10,]    0    0    0    0    0    0    0
```

Run JAGS

Thin wrapper for `R2jags::jags`, data list, and parameter set. default is 2 chains, 3k iters, 1k burn-in.

```
runJags = function(mod, dat, pars, nchains = 2, niter = 3000, nburnin = 1000) {
  jags(model.file = textConnection(mod),
       data = dat,
       n.chains = nchains,
       n.iter = niter,
       n.burnin = nburnin,
       n.thin = 1,
       parameters.to.save = pars)
}

data_list = list(
  y = dat$yc,
  task = dat$JV,
  sub = dat$S,
  I = I,
  J = J,
  N = N,
  D = D,
  constraint_zero = constraint_matrix_zero,
  constraint_pos  = constraint_matrix_pos,
  mu.m = 2,
  mu.s = 2,
  tuneDelta = 1,
  tuneLambda = 1
)
```

```r
parameters = c("lambda", "theta", "mu", "del2", "pTau2", "eta")

fit = runJags(mod_bhfm, data_list, parameters)
```

```
## module glm loaded
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 20547
##    Unobserved stochastic nodes: 2516
##    Total graph size: 67445
##
## Initializing model
```

```r
invisible(fit)
```

Sampling diagnostics (selected parameters)

We will not print the whole `fit`. instead, we check a small set of parameters and show convergence metrics and basic plots.

```r
sel = c("lambda[2,1]",   # Brentano 2 loading on its specific factor
        "lambda[4,2]",   # Ebbinghaus 2 loading on its specific factor
        "lambda[6,3]",   # Poggendorff 2 loading on its specific factor
        "del2[1]",       # residual variance, Brentano 1
        "pTau2[1]",      # within-subject precision for Brentano 1
        "mu[1]")         # intercept for Brentano 1

summ = fit$BUGSoutput$summary
keep = summ[sel, c("mean","sd","2.5%","50%","97.5%","Rhat","n.eff")]
knitr::kable(round(keep, 3), align = "c",
          caption = "Selected parameters: posterior summaries and diagnostics")
```

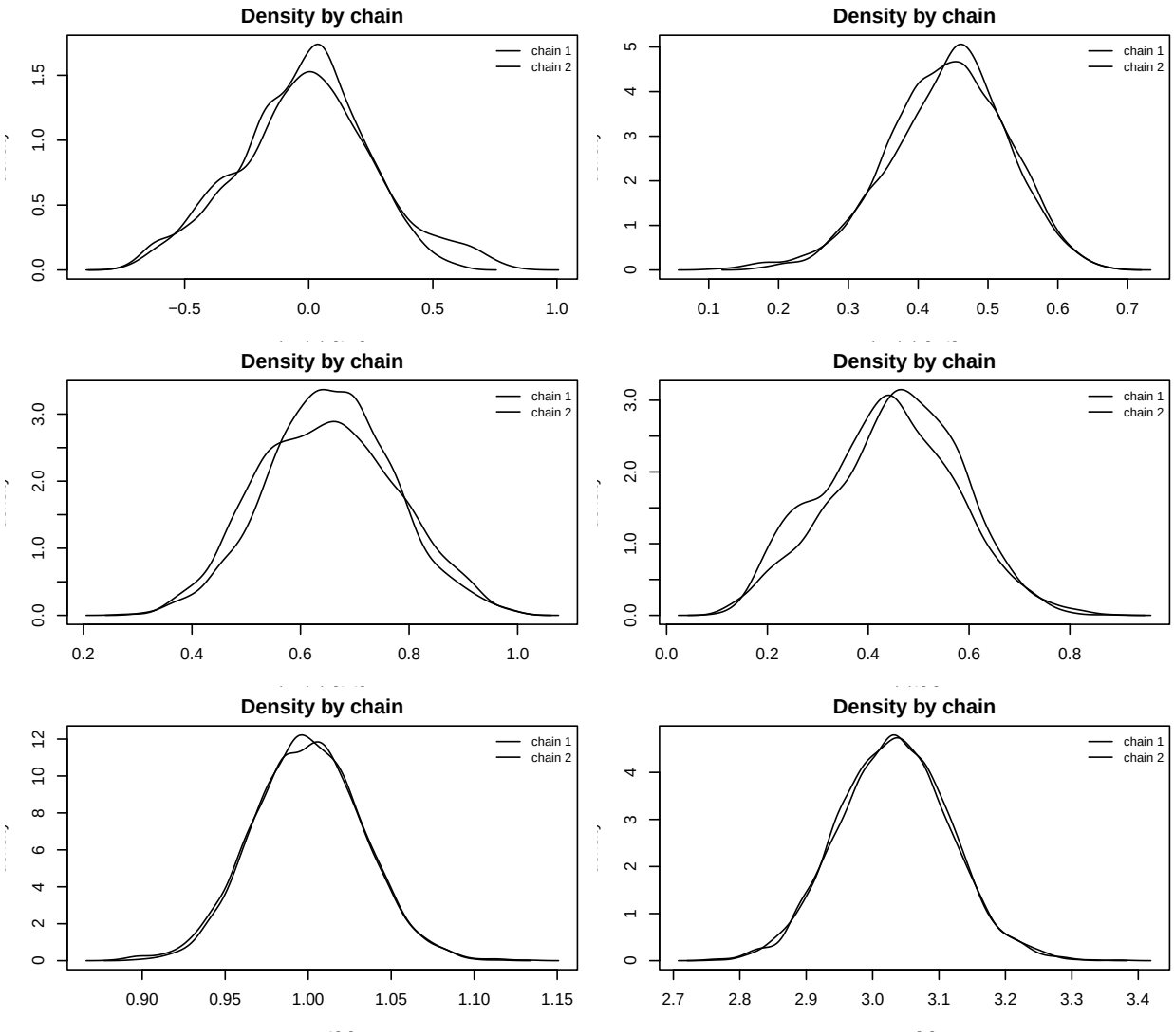
Table 1: Selected parameters: posterior summaries and diagnostics

	mean	sd	2.5%	50%	97.5%	Rhat	n.eff
lambda[2,1]	-0.019	0.267	-0.567	-0.007	0.527	1.013	340
lambda[4,2]	0.443	0.085	0.270	0.447	0.598	1.005	4000
lambda[6,3]	0.653	0.121	0.421	0.652	0.896	1.006	890
del2[1]	0.448	0.130	0.194	0.452	0.693	1.013	130
pTau2[1]	1.000	0.032	0.938	1.000	1.065	1.002	1600
mu[1]	3.033	0.082	2.879	3.033	3.197	1.001	4000

```
op = par(mfrow = c(3, 2), mar = c(3.2, 3.2, 2, 1))
for (v in sel) {
  try(R2jags::traceplot(fit, varname = v), silent = TRUE)
}
par(op)

sa = fit$BUGSoutput$sims.array
pnames = dimnames(sa)[[3]]
plot_density = function(var) {
  idx = which(pnames == var)
  if (length(idx) != 1) return(invisible(NULL))
  dl = lapply(1:dim(sa)[2], function(c) density(sa[, c, idx]))
  xr = range(sapply(dl, function(d) d$x))
  yr = range(sapply(dl, function(d) d$y))
  plot(NA,
       xlim = xr,
       ylim = yr,
       xlab = var,
       ylab = "density",
       main = "Density by chain")
  for (c in seq_along(dl)) lines(dl[[c]])
  legend("topright",
         legend = paste("chain", seq_along(dl)),
         lty = 1,
         bty = "n",
         cex = 0.8)
}

op = par(mfrow = c(3, 2), mar = c(3.2, 3.2, 2, 1))
for (v in sel) plot_density(v)
```




```
par(op)
```

Post-processing helpers

We align and standardize loadings:

- `makePositive` flips factor signs so average loading per factor is positive
- `align` does a joint rotation across MCMC draws (requires `infinitefactor::jointRot`)
- `f2Eigenmax` rotates the **two general factors** to a convenient orientation
- `standardize_lambda` rescales so that each indicator has unit variance

```r
makePositive = function(lambda) {
  # lambda: M x J x D
  D = dim(lambda)[3]
  dimMean = apply(lambda, 3, mean)
  for (d in 1:D) {
    s = ifelse(dimMean[d] == 0, 1, sign(dimMean[d]))
    lambda[,,d] = s * lambda[,,d]
  }
  return(lambda)
}

align = function(lambda, I = 0) {
  # lambda: M x J x D
  M = dim(lambda)[1]
  numFactors = dim(lambda)[3]
  eta = array(0, dim = c(M, I, numFactors))  # placeholders
  lambdaList = lapply(1:M, function(x) lambda[x,,])
  etaList    = lapply(1:M, function(x) eta[x,,])
  aligned    = jointRot(lambda = lambdaList, eta = etaList)
  out        = aperm(abind(aligned$lambda, along = 3), c(3,1,2))
  return(out)
}

f2Eigenmax = function(lambda) {
  # only rotates last two dims when you pass lambda[,,6:7]
  euler2 = function(theta)
    matrix(c(cos(theta), -sin(theta),
             sin(theta),  cos(theta)), nrow = 2, byrow = TRUE)

  obj2 = function(theta, b) {
    r = b %*% euler2(theta)
    apply(r^2, 2, sum)[1]
  }

  M = dim(lambda)[1]
  pLam = apply(lambda, 2:3, mean)  # J x 2 average loadings
  out = lambda
  ang = optimize(obj2, c(-pi, pi), maximum = TRUE, b = pLam)$maximum
  for (m in 1:M) out[m,,] = lambda[m,,] %*% euler2(ang)
  return(out)
}

standardize_lambda = function(lambda, del2) {
  # lambda: M x J x D ; del2: M x J
```

```
  M = dim(lambda)[1]; J = dim(lambda)[2]; D = dim(lambda)[3]
  lam_std = lambda
  del2_std = del2
  Sigma = array(NA_real_, dim = c(M, J, J))
  Rho   = array(NA_real_, dim = c(M, J, J))

  for (m in 1:M) {
    S = tcrossprod(lambda[m,,]) + diag(del2[m,], J)
    Sigma[m,,] = S
    Rho[m,,]   = cov2cor(S)
    sigma = sqrt(diag(S))
    lam_std[m,,] = sweep(lambda[m,,], 1, sigma, "/")
    del2_std[m,] = del2[m,] / (sigma^2)
  }
  list(lambda = lam_std, del2 = del2_std, Sigma = Sigma, rho = Rho)
}

makeTable = function(lambda, del2) {
  dims = dim(lambda)
  M = dims[1]; J = dims[2]; D = dims[3]
  lam_sum = matrix(0, J, D)
  uni_prop_task_mat = matrix(0, M, J)
  fac_prop_total_mat = matrix(0, M, D)
  uni_prop_total_vec = numeric(M)

  for (m in 1:M) {
    tlam = lambda[m,,]
    tdel = del2[m,]

    tot_com_task = rowSums(tlam^2)   # communality per indicator
    tot_task_j   = tot_com_task + tdel
    tot_com_fac  = colSums(tlam^2)   # variance by factor
    tot_uni      = sum(tdel)
    tot_var      = sum(tot_task_j)

    lam_sum = lam_sum + tlam
    uni_prop_task_mat[m,] = tdel / tot_task_j
    fac_prop_total_mat[m,] = tot_com_fac / tot_var
    uni_prop_total_vec[m] = tot_uni / tot_var
  }

  lam_mean = round(lam_sum / M, 3)
  uni_prop_task = round(colMeans(uni_prop_task_mat), 3)
  fac_prop_total = round(colMeans(fac_prop_total_mat), 3)
```

```
  uni_prop_total = round(mean(uni_prop_total_vec), 3)

  out = cbind(lam_mean, uni_prop_task)
  out = rbind(out, c(fac_prop_total, uni_prop_total))

  longNames = c("Brentano 1","Brentano 2","Ebbinghaus 1","Ebbinghaus 2",
                "Poggendorf 1","Poggendorf 2","Ponzo 1","Ponzo 2",
                "Zoellner 1","Zoellner 2")
  rownames(out) = c(longNames, "Prop. Var.")
  colnames(out) = c(paste0("F", 1:D), "Unique Var.")
  out
}
```

Align, Standardize, Summarize

We (i) align the two general factors across draws, (ii) enforce positive direction, (iii) standardize loadings, and (iv) build a summary table.

```
samples = fit$BUGSoutput$sims.list
lambda = samples$lambda
del2   = samples$del2

lamb_g = lambda[,,6:7]
lamb_aligned = align(lamb_g, I = I)
lamb_aligned = f2Eigenmax(lamb_aligned)
lamb_aligned = makePositive(lamb_aligned)

lambda[,,6:7] = lamb_aligned

std = standardize_lambda(lambda, del2)
lam_std = std$lambda
del2_std = std$del2

tab = makeTable(lam_std, del2_std)
tcap = "Posterior mean loadings and variance proportions (standardized scale)."
knitr::kable(tab, align = "c",
             caption = tcap)
```

Table 2: Posterior mean loadings and variance proportions (standardized scale).

	F1	F2	F3	F4	F5	F6	F7	Unique Var.
Brentano 1	0.240	0.000	0.000	0.000	0.000	0.575	-0.113	0.522
Brentano 2	-0.022	0.000	0.000	0.000	0.000	0.482	0.053	0.617
Ebbinghaus 1	0.000	0.624	0.000	0.000	0.000	0.252	0.268	0.416
Ebbinghaus 2	0.000	0.616	0.000	0.000	0.000	0.373	-0.175	0.386
Poggendorf 1	0.000	0.000	0.665	0.000	0.000	0.507	0.021	0.237
Poggendorf 2	0.000	0.000	0.607	0.000	0.000	0.480	0.170	0.316
Ponzo 1	0.000	0.000	0.000	0.827	0.000	0.373	-0.108	0.125
Ponzo 2	0.000	0.000	0.000	0.833	0.000	0.283	0.156	0.162
Zoellner 1	0.000	0.000	0.000	0.000	0.589	0.583	0.002	0.253
Zoellner 2	0.000	0.000	0.000	0.000	0.561	0.620	-0.058	0.245
Prop. Var.	0.019	0.079	0.084	0.138	0.069	0.231	0.052	0.328

Plots

(A) posterior mean loadings on the two general factors; (B) distribution of total variance share explained by each **task factor**; (C) distribution of variance share for the two **general factors** and **unique variance**.

```
dims = dim(lam_std)
M = dims[1]; J = dims[2]; D = dims[3]

fac_prop_total_mat = matrix(0, M, D)
uni_prop_total_vec = numeric(M)

for (m in 1:M) {
  tlam = lam_std[m,,]
  tdel = del2_std[m,]
  tot_com_fac = colSums(tlam^2)
  tot_uni = sum(tdel)
  tot_var = sum(tdel + rowSums(tlam^2))
  fac_prop_total_mat[m,] = tot_com_fac / tot_var
  uni_prop_total_vec[m]  = tot_uni / tot_var
```

```r
}

lam_mean = round(apply(lam_std, c(2,3), mean), 3)

task_names = c("Brentano 1","Brentano 2","Ebbinghaus 1","Ebbinghaus 2",
               "Poggendorf 1","Poggendorf 2","Ponzo 1","Ponzo 2",
               "Zoellner 1","Zoellner 2")

task_names2 = c("Brentano","Ebbinghaus","Poggendorf","Ponzo","Zoellner")

base_colors = RColorBrewer::brewer.pal(5, "Set1")
cols = unlist(lapply(base_colors,
                     function(col) c(adjustcolor(col, alpha.f = 0.6), col)))

par(mfrow = c(1,3), mar = c(5.2, 2.0, 2.5, 0.5))

# (A) mean loadings on general factors (F6,F7)
plot(NA, xlim = c(-1, 1), ylim = c(-1, 1), axes = FALSE, xlab = "", ylab = "")
grid(lwd = 2); box(); abline(h = 0, v = 0, lwd = 2)
for (i in 1:nrow(lam_mean)) {
  segments(0, 0, lam_mean[i, 6], lam_mean[i, 7], col = cols[i], lwd = 2)
}
points(lam_mean[, 6], lam_mean[, 7], col = cols, pch = 19, cex = 1.1, lwd = 2)
legend("bottomleft", legend = task_names, col = cols, pch = 19,
       cex = 0.75, bg = "white", ncol = 2)
mtext("(A)", side = 3, line = 0.4, cex = 0.85, font = 2, adj = 0)
mtext("General Factor 1", side = 1, line = 1.0, cex = 0.8)
mtext("General Factor 2", side = 2, line = 1.0, cex = 0.8)

# (B) variance share per task factor
plot(NA, xlim = c(0, 1), ylim = c(0, 50), axes = FALSE, xlab = "", ylab = "")
axis(1); box()
tcols = cols[c(1,3,5,7,9)]
for (j in 1:5) {
  dens = density(fac_prop_total_mat[, j])
  polygon(dens, col = adjustcolor(tcols[j], 0.4), border = tcols[j], lwd = 2)
}
legend("topright", legend = task_names2, fill = adjustcolor(tcols, 0.4),
       border = tcols, cex = 0.95)
mtext("(B)", side = 3, line = 0.4, cex = 0.85, font = 2, adj = 0)
mtext("Proportion of Total Variance (Task Factors)",
      side = 1, line = 2.6, cex = 0.8)
mtext("Posterior Density", side = 2, line = 1.0, cex = 0.8)
```

```r
# (C) general factors + unique variance
plot(NA, xlim = c(0, 1), ylim = c(0, 50), axes = FALSE, xlab = "", ylab = "")
axis(1); box()
tcols2 = c("black", "gray60", "brown")
for (d in 6:7) {
  dens = density(fac_prop_total_mat[, d])
  polygon(dens, col = adjustcolor(tcols2[d-5], 0.4),
          border = tcols2[d-5], lwd = 2)
}
dens = density(uni_prop_total_vec)
polygon(dens, col = adjustcolor(tcols2[3], 0.35), border = tcols2[3], lwd = 2)
legend("topright",
       legend = c("General Factor 1", "General Factor 2", "Unique Variance"),
       fill = adjustcolor(tcols2[1:3], 0.35),
       border = tcols2[1:3], cex = 0.95)
mtext("(C)", side = 3, line = 0.4, cex = 0.85, font = 2, adj = 0)
mtext("Proportion of Total Variance", side = 1, line = 2.6, cex = 0.8)
mtext("Posterior Density", side = 2, line = 1.0, cex = 0.8)
```