Fitting The Hierarchical Factor Model to a Synthetic Data Set

Jeff Rouder

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:

Step 1: Get the needed libraries

The following libraries need to be installed in R.

- R2jags (the JAGS API we use here)
- infinitefactor (for alignment)
- abind (for array manipulation)
- GPArotation (for rotations)

Step 2: Load Data.

```
dat=read.table("run2Dat.txt",head=T)
```

Feel free to explore the data. For example, the following code computes the observed effect for each person in each task and the correlations of these effects across tasks:

```
mns=tapply(dat$y,list(dat$sub,dat$task,dat$cond),mean)
effects=mns[,,2]-mns[,,1]
corMat=cor(effects)
```

Step 3: Specify The Model

We have written the core model as `facH.mod` in the file `facMod.R`. The JAGS specification is:

```
facH.mod="
model{
  for (j in 1:J){
    pTau2[j] ~ dgamma(.5, .5*pow(tuneTau,2))
    mu[j]~dnorm(mu.m, pow(mu.s, -2))
    pDelta2[j]~dgamma(.5,.5*pow(tuneDelta,2))}
  for (d in 1:D){
    for (j in 1:J){
      lambda[j,d] ~ dnorm(0, pow(tuneLambda, -2))}
    for (i in 1:I){
      eta[i,d] ~ dnorm(0,1)}}

  for (i in 1:I){
```

```
    for (j in 1:J){
      for (d in 1:D){
        temp[i,j,d] <- lambda[j,d]*eta[i,d]}
      centerTheta[i,j] <- mu[j] + sum(temp[i,j,1:D])
      theta[i,j] ~dnorm(centerTheta[i,j],pDelta2[j])
      alpha[i,j] ~dnorm(alpha.m,pow(alpha.s, -2))}}
  for (n in 1:N){
    center[n] = alpha[sub[n],task[n]]+(cond[n]-1.5)*theta[sub[n],task[n]]
    y[n] ~ dnorm(center[n], pTau2[task[n]])}
}"
```

All that is needed to load this model is to run the above chunk or load it from the file `facMod.R`:

Just make sure `facH.R` is in your working directory.

Step 4: Specify the priors

For RT tasks with contrasts, the following priors are informed but broad enough to include many treatment-control type subsecond RT experiments in psychology. They are entered as a list called `priorRT`.

```
priorRT=list(
    "mu.m"=70,
    "mu.s"=100,
    "alpha.m"=1000,
    "alpha.s"=1000,
    "tuneDelta"=25,
    "tuneLambda"=25,
    "tuneTau"=200)
```

Step 5: Run the model

This step will take several minutes on your computer as the data are comprised of 320,000 trials and there are about 4000 parameters. The function `facH.run` in `facMod.R` incorporates the prior and the data and calls the JAGS sampler. The arguments are the data (`dat`), the number of factors, the priors, and the number of iterations. Because the analysis takes several minutes, the output is saved.

```
#output=facH.run(dat,numFactors = 2,prior=priorRT,M=5000)
#saveRDS(file='synthExamp.RDS',object=output)
```

Here is the code for reading the saved output.

```
output=readRDS('synthExamp.RDS')
```

Step 5. Post sampling rotational alignment

As discussed, MCMC iterations often correspond to different rotations and need to be aligned. Poworoznek provides a R library `infinitefactor` that performed the alignment. We have written a wrapper in our `R` file `facExtra.R`. Here is how to do it:

```
source('facExtra.R')
```

```
## Loading required package: infinitefactor
```

```
## Loading required package: reshape2
```

```
## Loading required package: ggplot2
```

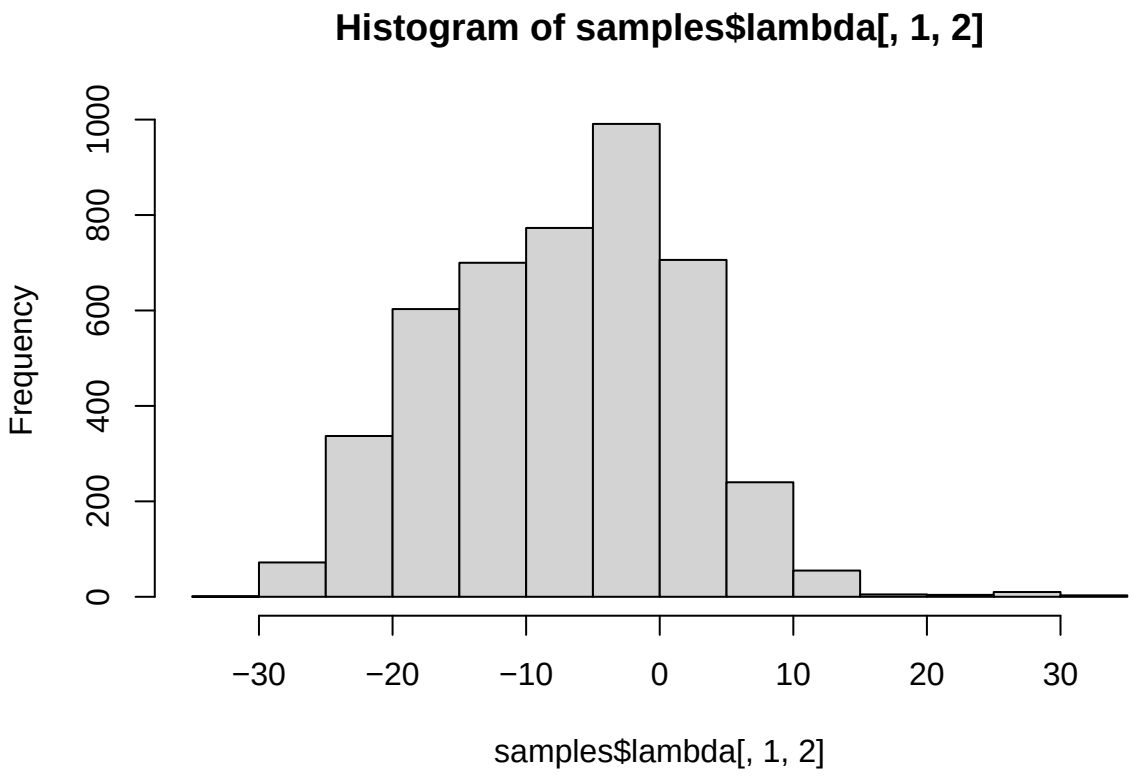
```
## Loading required package: abind
```

```
## Loading required package: GPArotation
```

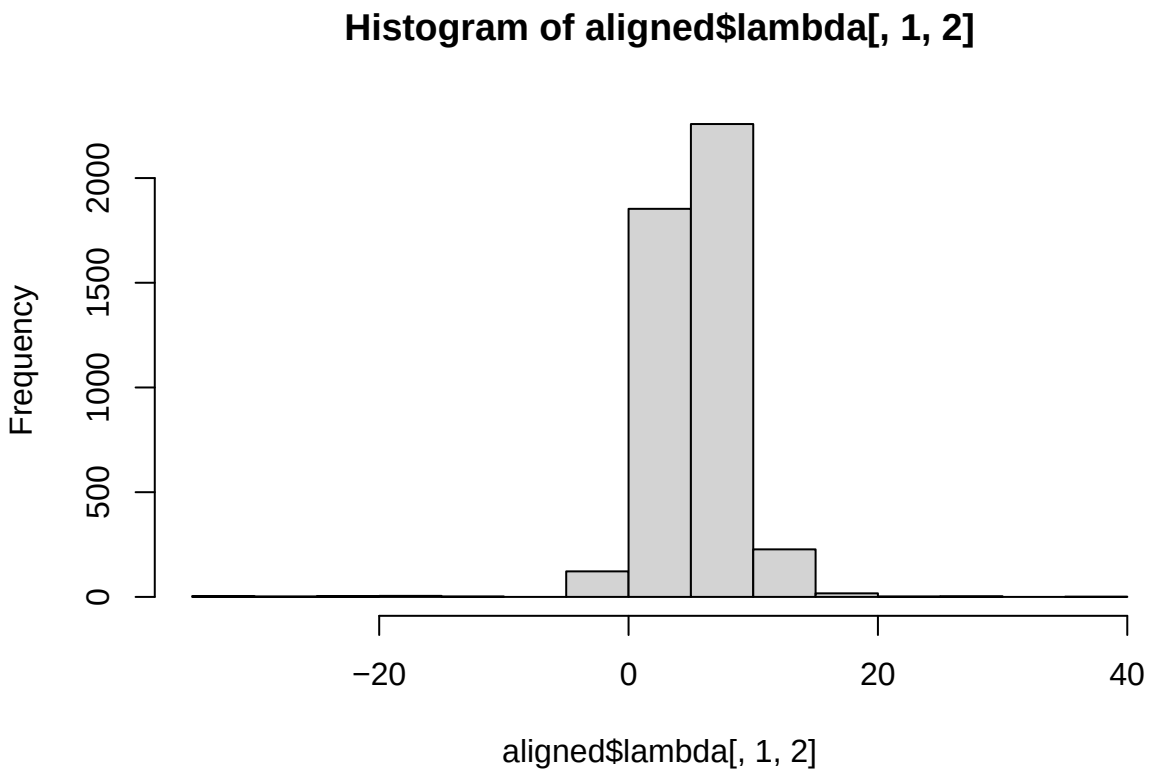
```
samples=output$BUGSoutput$sims.list  #get MCMC samples
aligned=align(samples)
```

The function `align` manipulated factor loadings and factor scores and passes all other parts of the chain. Here is a histogram of the original samples and the aligned ones for λ_{12}:

```
hist(samples$lambda[,1,2])
```




```
hist(aligned$lambda[,1,2])
```



As a final courtesey, we sign-transform each dimension of factor loadings to be positive. The function `makePositive()` in `facExtra` does this:

```
alignPos=makePositive(aligned)
```

Step 6. Standardize the components

You may choose to standardize your variance components. The function `standardize()` in `facExtra.R` does the standardization:

```
std=standardize(alignPos)
```

Step 7. Explore the posteriors.

You can use `names()` to see what parameters are sampled and reported. Of note is that `pDelta2` and `pTau2` are precisions not variances. That is what the `p` is for.

```
names(std)
```

```
##  [1] "deviance" "eta"      "lambda"   "mu"       "pDelta2"  "pTau2"
##  [7] "theta"    "delta2"   "Sigma"    "rho"
```

Lets look at the posterior factor loadings. The dimensions are $4500 \times 8 \times 2$ for 4500 iterations (after burn in), 8 tasks, and 2 dimensions. The posterior means are had by averaging across iterations:

```
dim(std$lambda)
```

```
## [1] 4500    8    2
```

```
pmLambda=apply(std$lambda,2:3,mean)
pmLambda
```

```
##            [,1]      [,2]
## [1,] 0.81170870 0.1477149
## [2,] 0.77196912 0.2230042
## [3,] 0.69313852 0.4779561
## [4,] 0.61905662 0.4868523
## [5,] 0.44498752 0.5745266
## [6,] 0.36373499 0.6635393
## [7,] 0.39961732 0.7970480
## [8,] 0.08571148 0.8754926
```

The standardized residuals (standard deviations) are

```
pmDelta=apply(sqrt(std$delta2),2,mean)
pmDelta
```

```
## [1] 0.5336395 0.5671893 0.5080603 0.5913498 0.6645235 0.6202261 0.4235911
## [8] 0.4425916
```

And the correlation matrix resulting from adding the components is:

```
pmRho=apply(std$rho,2:3,mean)
round(pmRho,2)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [1,] 1.00 0.65 0.63 0.57 0.45 0.40 0.44 0.20
## [2,] 0.65 1.00 0.65 0.59 0.47 0.43 0.48 0.26
## [3,] 0.63 0.65 1.00 0.67 0.58 0.57 0.66 0.48
## [4,] 0.57 0.59 0.67 1.00 0.56 0.55 0.64 0.48
## [5,] 0.45 0.47 0.58 0.56 1.00 0.55 0.64 0.55
## [6,] 0.40 0.43 0.57 0.55 0.55 1.00 0.68 0.62
## [7,] 0.44 0.48 0.66 0.64 0.64 0.68 1.00 0.74
## [8,] 0.20 0.26 0.48 0.48 0.55 0.62 0.74 1.00
```