Overview

In this vignette, I demonstrate the FCD sampler for the σ_k^2 parameters in the covariance model with dense eigenvectors. If we consider all of the other parameters in the model as fixed, then in this case, the individual FCDs are essentially like marginal posterior distributions, which we can determine exactly. The sampler does reproduce these known posterior distributions. However, as I will show, the posterior distributions can be extremely biased away from the true parameter value. Thus, I am left with a mixed conclusion: the sampler "recovers the true parameter value" only in some cases, but it does appear to sample from the correct distribution.

Model description

Data model: Data matrices P_k are $P \times P$ Hermitian positive definite. n_k is the number of samples corresponding to data matrix P_k. M_k is a known $P \times P$ Hermitian positive definite matrix.

$$P_k|\sigma_k^2, M_k \sim \text{ComplexWishart}(n_k, \sigma_k^2 M_k)$$

Prior: The independence Jeffreys Prior is $p(\sigma_k^2) \propto 1/\sigma_k^2$.

FCD:

$$\begin{aligned}
p(\sigma_k^2|P_k) &\propto p(P_k|\sigma_k^2, M_k)p(\sigma_k^2) \\
&\propto |P_k|^{n_k-P}|\sigma_k^2 M_k|^{-n_k} etr[-(\sigma_k^2 M_k)^{-1}P_k](\sigma_k^2)^{-1} \\
&\propto (\sigma_k^2)^{-Pn_k-1}\exp\left[\frac{1}{\sigma_k^2}tr(M_k^{-1}P_k)\right] \\
&\Rightarrow \sigma_k^2|P_k \sim \text{InvGamma}[Pn_k, tr(M_k^{-1}P_k)]
\end{aligned}$$

Based on properties of the Inverse Gamma distribution, we know the **posterior mean** is $E[\sigma_k^2|P_k] = tr(M_k^{-1}P_k)/(Pn_k - 1)$. If we consider the posterior mean as an estimator of σ_k^2, then due to the randomness in the data, the expectation of this estimator is:

$$\begin{aligned}
E\left[tr(M_k^{-1}P_k)/(Pn_k-1)\right] &= (Pn_k-1)^{-1}tr[E(M_k^{-1}P_k)] \\
&= (Pn_k-1)^{-1}tr[M_k^{-1}n_k\sigma_k^2 M_k] \\
&= \frac{Pn_k}{Pn_k-1}\sigma_k^2.
\end{aligned}$$

Thus, the posterior mean becomes less biased as n_k increases, and does not appear to depend on M_k.

Compare sampler to known posterior

In this section, I will show that the sampler function does indeed sample from the correct distribution, which we know exactly. If we generate samples with the function, their empirical density matches the exact density that I expect.

A well-behaved case

The matrix M_k is generated in a similar way to that in our larger model, where the parameter to the Complex Wishart distribution is $\sigma_k^2 M_k = \sigma_k^2(U_k \Lambda_k U_k^H + I_p)$. In this case, the entries of the $d \times d$ matrix Λ_k primarily affect the eigenvalues of M_k.

In this first example, the randomly generated Lambda values are low, all below 2. Note that for, say, the M_1 matrix, the "Summary of eigenvalues" results shows that the eigenvalues are all on a similar order of magnitude. The same is true for the M_2 and M_3 matrices.

In the demonstration, a data matrix P_k is generated for each of the K groups. The function `sigmak2_gibbs_densCovar` is used to generate samples from the respective posteriors. After discarding half the generated samples, we can calculate the posterior mean from the samples, and also the exact posterior mean. We can compare these to the true σ_{k0}^2 values.

The exact and sampled posterior means are very close. They are all slightly positively biased compared to the true σ_{k0}^2 values. The sampled densities are all very close to the true Inverse Gamma densities.

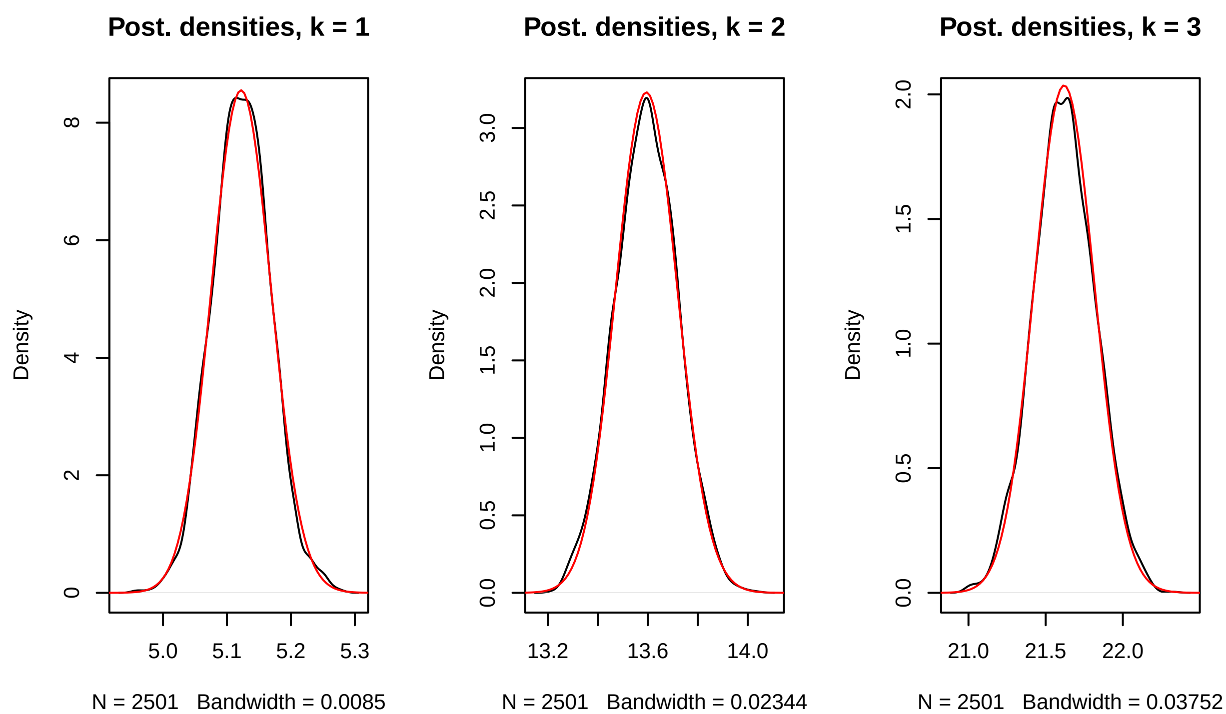
```
P = 12
d = 4
K = 3
nk_scale = 1000
gibbs_its = 5000

set.seed(13032025)
Lambda_ks <- array(NA, c(d, K))
# different Lambda values (eigenvalues) lead to different estimator quality
for(k in 1:K) {
    # Lambda_ks[, k] <- seq(4+k, -0.95+k, length.out = d)
    Lambda_ks[, k] <- rgamma(d, 1, 1) |> sort(decreasing = TRUE)
    # Lambda_ks[, k] <- seq(5, 10^(k/2), length.out = d) |> sort(decreasing = TRUE)
    # Lambda_ks[, k] <- seq(5, 10^(k), length.out = d) |> sort(decreasing = TRUE)
}

print(Lambda_ks)
```

```
##           [,1]      [,2]      [,3]
## [1,] 0.8475202 1.1770745 1.6116179
## [2,] 0.4274250 1.0677248 0.5785050
## [3,] 0.3464576 0.5847943 0.5508674
## [4,] 0.0021038 0.5117572 0.3574090
```

```
demo_FCD(Lambda_ks)
```

```
## [1] "Summary of eigenvalues for M_1 matrix"
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   1.000   1.000   1.135   1.088   1.848
## [1] "Summary of eigenvalues for M_2 matrix"
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   1.000   1.000   1.278   1.530   2.177
## [1] "Summary of eigenvalues for M_3 matrix"
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   1.000   1.000   1.258   1.406   2.612
## [1] "Exact posterior means"
```

```
## [1] 5.122912
## [1] 13.59639
## [1] 21.62307
## [1] "Sample posterior means"
## [1]  5.122898 13.594903 21.622620
## [1] "Known sigma_k2 values"
## [1]  5.0 12.5 20.0
```



A poorly-behaved case

In this case, we generate the Lambda values in a different way. The result is that the M_k matrix eigenvalues are spread across different orders of magnitude.

The result is that the exact and sample posterior means match, and the observed and exact densities match. However, we can see that the posterior means and distributions are very biased compared to the true values.

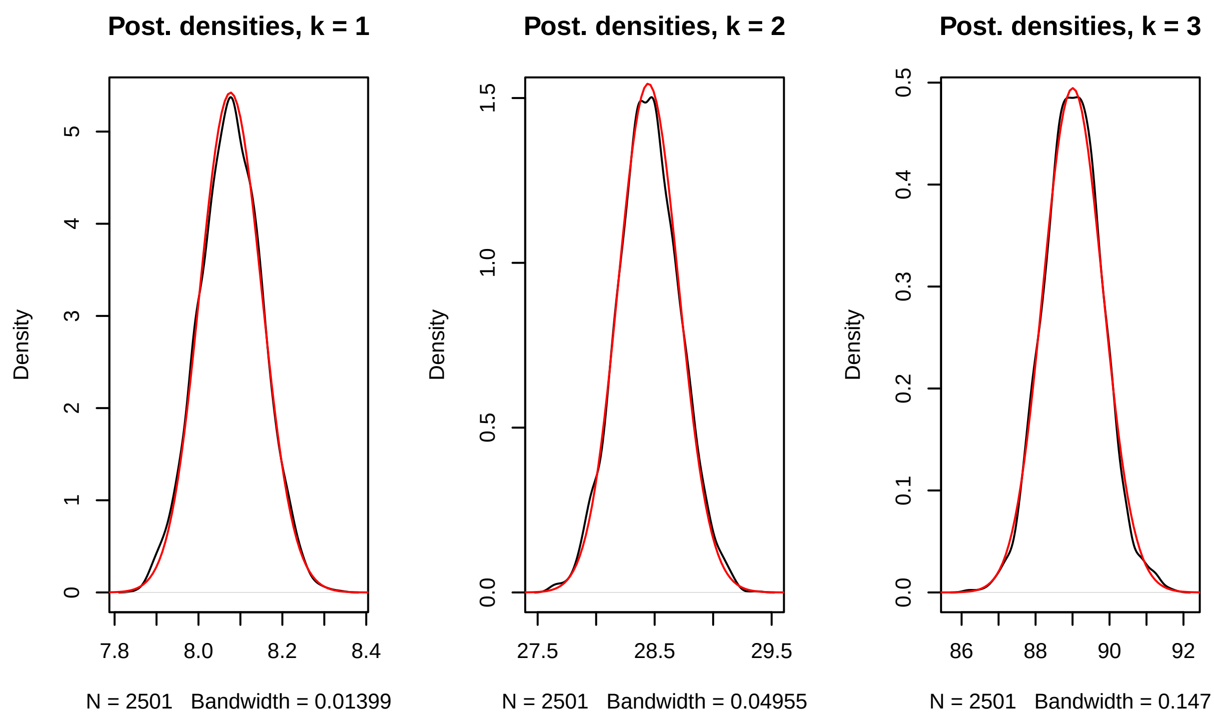
```r
set.seed(13032025)
Lambda_ks <- array(NA, c(d, K))
# different Lambda values (eigenvalues) lead to different estimator quality
for(k in 1:K) {
    Lambda_ks[, k] <- seq(5, 10^(k/2), length.out = d) |> sort(decreasing = TRUE)
}

print(Lambda_ks)
```

```
##          [,1]      [,2]     [,3]
## [1,] 5.000000 10.000000 31.62278
## [2,] 4.387426  8.333333 22.74852
```

```
## [3,] 3.774852  6.666667 13.87426
## [4,] 3.162278  5.000000  5.00000
```

```r
demo_FCD(Lambda_ks)
```

```
## [1] "Summary of eigenvalues for M_1 matrix"
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   1.000   1.000   2.360   4.315   6.000
## [1] "Summary of eigenvalues for M_2 matrix"
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   1.000   1.000   3.500   6.417  11.000
## [1] "Summary of eigenvalues for M_3 matrix"
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   1.000   1.000   7.104   8.219  32.623
## [1] "Exact posterior means"
## [1] 8.077764
## [1] 28.45015
## [1] 89.01993
## [1] "Sample posterior means"
## [1]  8.076842 28.449468 89.019321
## [1] "Known sigma_k2 values"
## [1]  5.0 12.5 20.0
```



Evaluating the distribution of the posterior mean

In this section, we consider whether the posterior distributions which result from randomly observed data will "recover" a known true σ_k^2 value. That is, if we start with a known σ_k^2 value and generate multiple observations of P_k using that value, how do the resulting posterior distributions behave? Are the posterior means, $E[\sigma_k^2|P_k] = tr(M_k^{-1}P_k)/(Pn_k - 1)$, close to the true parameter value? Do posterior credible intervals often contain the true parameter value?

Appendix

```r
#####
# sigmak2_gibbs_densCovar
#####

sigmak2_gibbs_densCovar
```

```
## function (data_list, param_list)
## {
##     result_sigmas <- vector(mode = "numeric", K)
##     IP <- diag(param_list$P)
##     for (k in 1:K) {
##         Pk <- data_list[[k]]
##         Uk <- param_list$U_ks[, , k]
##         Lambdak <- diag(param_list$Lambda_ks[, k])
##         Mk <- Uk %*% Lambdak %*% t(Conj(Uk)) + IP
##         ak <- .ak_sigmak2_densCovar(P, param_list$n_k[k])
##         bk <- .bk_sigmak2_densCovar(Mk, Pk)
##         result_sigmas[k] <- 1/rgamma(1, ak, rate = bk)
##     }
##     result_sigmas
## }
## <bytecode: 0x55ea577fd1e0>
```

```r
#####
# Posterior demonstration function
#
# Generates true parameter values and data using those values.
# Draws samples from posterior using the sampling function to be validated.
# Calculates observed and exact posterior means, and plots observed and exact posterior
↪  densities.
#
#####
demo_FCD <- function(Lambda_ks, P = 12, d = 4, K = 3, nk_scale = 1000,
                     gibbs_its = 5000) {

    n_k <- nk_scale + 5*(1:K)
    # number of Gibbs iterations, for testing
    gibbs_its <- 5000

    data_list <- list()

    U_ks <- array(NA, c(P, d, K))

    sigma_k20 <- seq(5, 20, length.out = K)

    for (k in 1:K) {
        U_ks[, , k] <- runitary(P, d)

        # temp matrix
        M_k <- diag(P) +
```

```r
        U_ks[, , k] %*% diag(Lambda_ks[, k]) %*% t(Conj(U_ks[, , k]))
    M_k <- round(M_k, 9)
    print(paste0("Summary of eigenvalues for M_",k," matrix"))
    print(summary(eigen(M_k)$values))

    data_list[[k]] <- rcwis(n_k[k], sigma_k20[k] * M_k)
}

# stand-in for sigma_k2, the parameters to be sampled
sigma_k2s <- vector("numeric", K)

##### Parameters list
param_list <- list(P = P,
                   d = d,
                   K = K,
                   n_k = n_k,
                   U_ks = U_ks,
                   Lambda_ks = Lambda_ks,
                   sigma_k2s = sigma_k2s)

# test function
sigmak2_gibbs_densCovar(data_list, param_list)

# test the function within a sampling loop
# initialize an array to track the sigma_k2 values
sigma_k2_S <- array(NA, c(K, gibbs_its))

for (s in 1:gibbs_its) {
    sigma_k2s <- sigmak2_gibbs_densCovar(data_list, param_list)
    param_list$sigma_k2s <- sigma_k2s
    sigma_k2_S[, s] <- sigma_k2s
}

# in this case, we know that the distribution should be certain Inv Gammas

# Exact posterior mean
print("Exact posterior means")
for (k in 1:K) {
    # temp matrix
    M_k <- diag(P) +
        U_ks[, , k] %*% diag(Lambda_ks[, k]) %*% t(Conj(U_ks[, , k]))
    print(Re(sum(diag(solve(M_k) %*% data_list[[k]]))) / (P * n_k[k] - 1))
}

post_samples <- round(gibbs_its/2):gibbs_its

# Sample posterior mean
print("Sample posterior means")
print(rowMeans(sigma_k2_S[, post_samples]))
# Known sigmak2 values
print("Known sigma_k2 values")
print(sigma_k20)

par(mfrow = c(1, 3))
```

```
    # plot observed and true densities
    for (k in 1:K) {
        xmin <- min(sigma_k2_S[k, post_samples]) * .8
        xmax <- max(sigma_k2_S[k, post_samples]) * 1.2
        plot(density(sigma_k2_S[k, post_samples]), type = "l",
             main = paste0("Post. densities, k = ",k))

        M_k <- diag(P) +
            U_ks[, , k] %*% diag(Lambda_ks[, k]) %*% t(Conj(U_ks[, , k]))

        ak <- P*n_k[k]
        bk <- Re(sum(diag(solve(M_k) %*% data_list[[k]])))
        # TODO note the transformation required to plot inverse gamma, i.e. /x^2,
        curve(dgamma(1/x, shape = ak, rate = bk) / x^2,
          from = xmin, to = xmax, n = 501, add = TRUE,
          col = "red", ylab = "density")
    }
    par(mfrow = c(1, 1))
}
```