lab2_xiangnan_yue

R Markdown

Install and import packages

1. preliminaries:

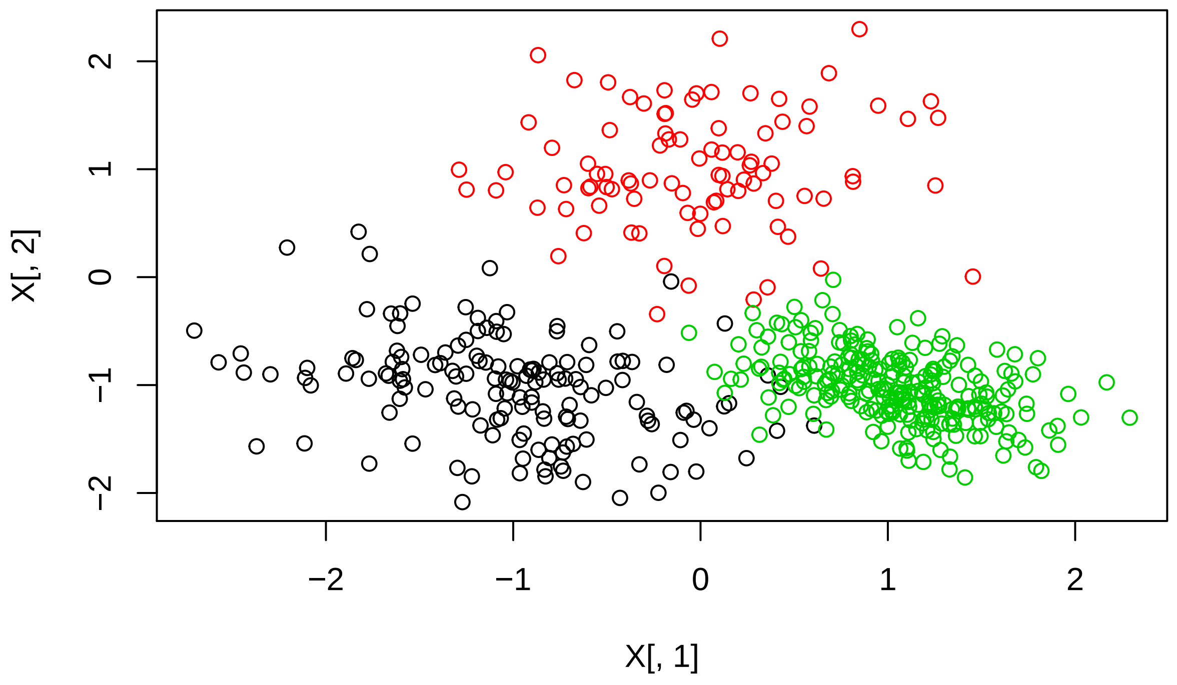
simulate a gaussian mixture

```
d=2
K <- 3
N <- 500
set.seed(1)
p <- c(3/10, 2/10, 5/10)
NN <- rmultinom(n = 1, size = N, prob = p)  # dimension 3 multinomial
Mu <- rbind(c(-1,-1), c(0,1), c(1, -1))  # the u
X <- matrix(ncol = d, nrow = 0)
Sigma <- array(dim = c(2,2,K))
for(j in 1:K){
    Sigma[,,j] <- rwishart(nu = 5, S = 0.05*diag(d))
    # Wishart distribution
}
for(j in 1:K){
    X <- rbind(X, mvrnorm(n=NN[j], mu = Mu[j,], Sigma=Sigma[,,j]))
    # multivariate normal distribution
}
```

```
#' labs: vector of labels
labs <- rep(0,N)
count=1
for(j in 1:K)
{
    labs[count:(count+NN[j]-1)] <- j
    count=count + NN[j]
}
```

Plot the labelled data

```
plot(X[,1], X[,2], col=labs)
```



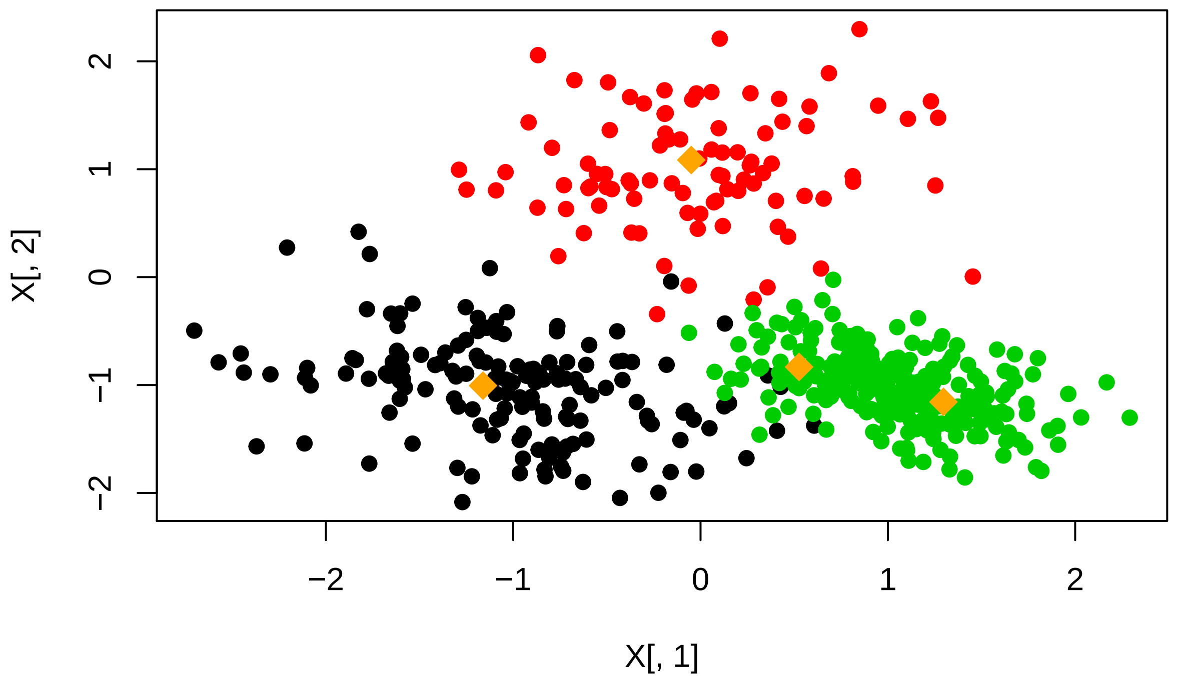
2 EM

```r
#' Run EM
Kfit <- 4 ## try with Kfit= 2,3,4,5 ...
outputem <- emalgo(x=X,k=Kfit, tol=1e-6)
```

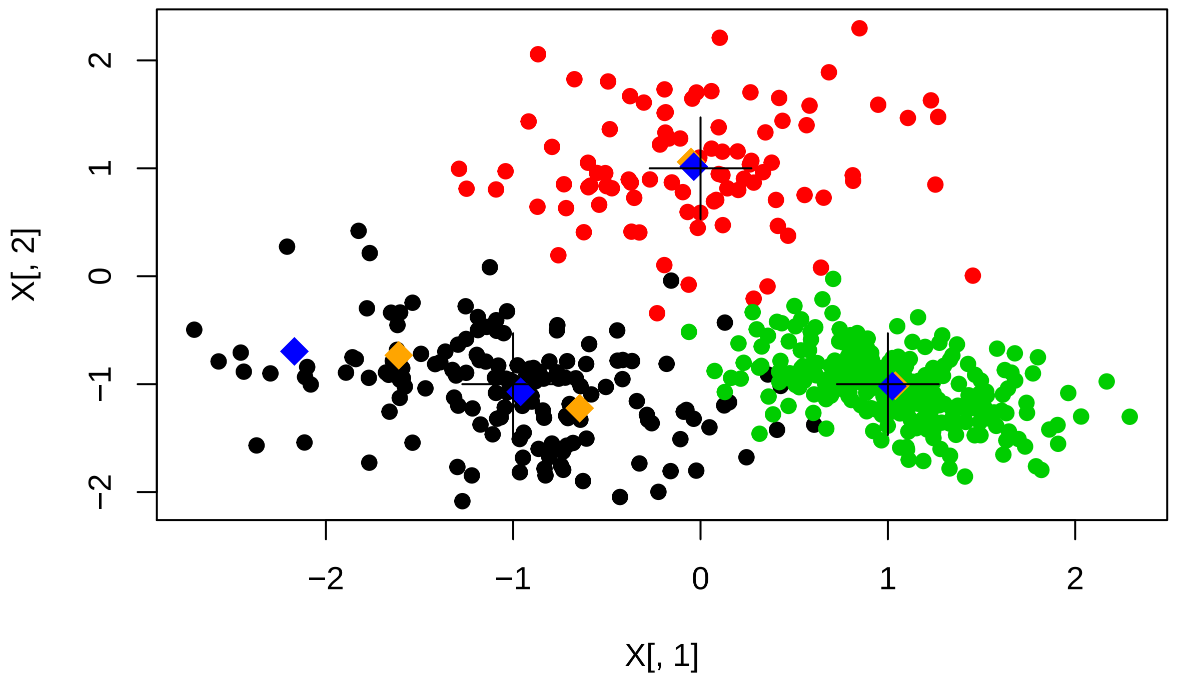
```r
#' inspect the objective function (stopping criterion)
length(outputem$objective)
```

```
## [1] 262
```

```r
#' Plot the (labelled) data
plot(X[,1], X[,2], col = labs, pch = 19)

#' Add the starting points (from kmeans) to the plot
Init <-  initem(X,Kfit)
points(Init$Mu[,1],Init$Mu[,2], col="orange",pch=18,cex=2)
```



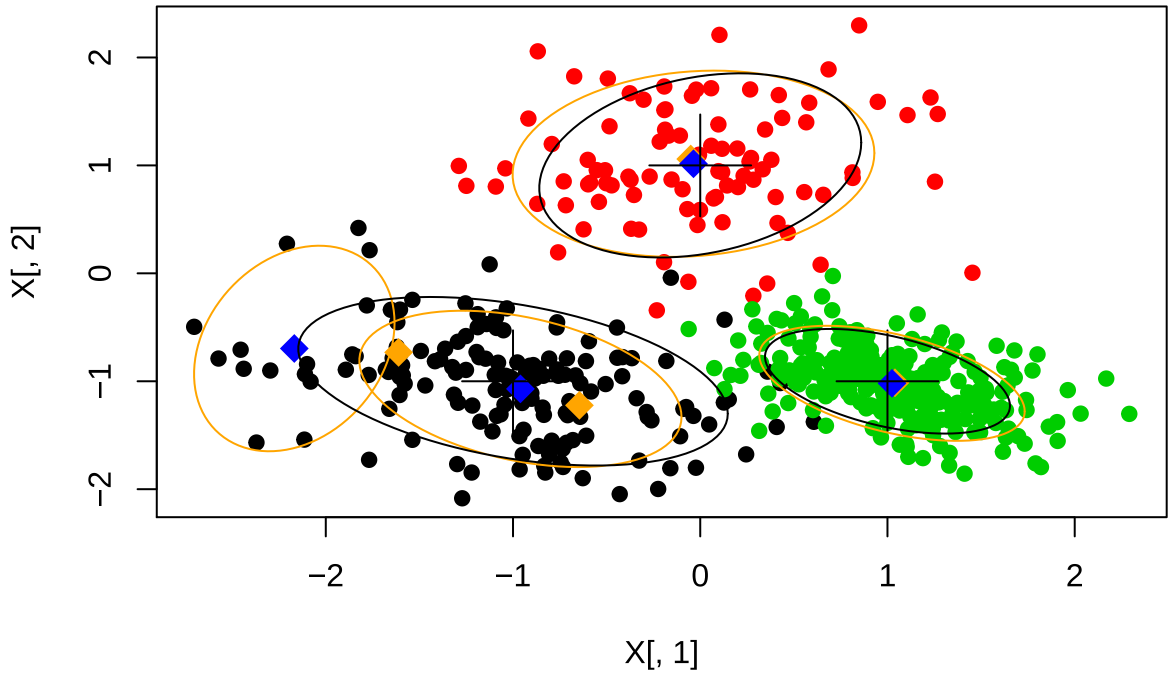
```r
#' Plot the (labelled) data
plot(X[,1], X[,2], col = labs, pch = 19)

#' Add the starting points (from kmeans) to the plot
Init <-  initem(X,Kfit)
points(Init$Mu[,1],Init$Mu[,2], col="orange",pch=18,cex=2)
#' Add the centers from EM
points(outputem$last$Mu[,1],outputem$last$Mu[,2], col="blue",pch=18,cex=2)
#' Add the true centers
points(Mu[,1],Mu[,2], col="black",pch=3,cex=5)
```



where

the blue points are the centres returned by EM algorithm, and the black cross represents the real centres.

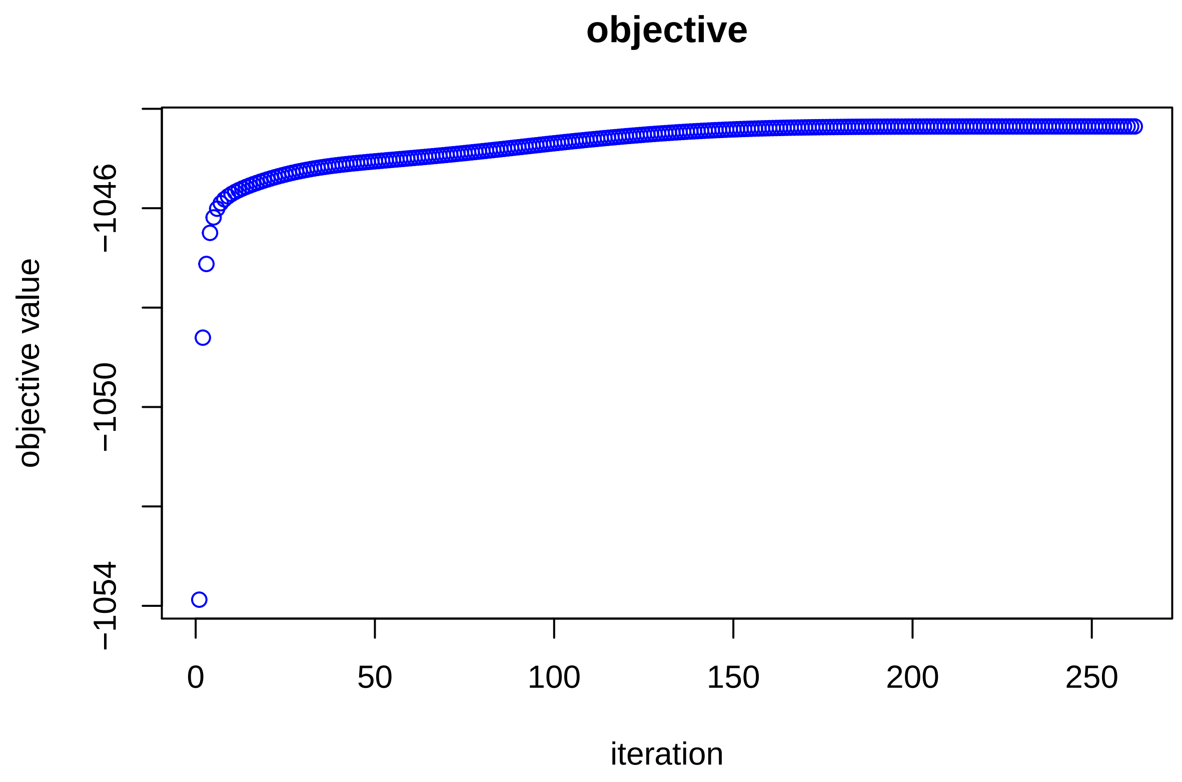
```
#' Plot the (labelled) data
plot(X[,1], X[,2], col = labs, pch = 19)

#' Add the starting points (from kmeans) to the plot
Init <-  initem(X,Kfit)
points(Init$Mu[,1],Init$Mu[,2], col="orange",pch=18,cex=2)
#' Add the centers from EM
points(outputem$last$Mu[,1],outputem$last$Mu[,2], col="blue",pch=18,cex=2)
#' Add the true centers
points(Mu[,1],Mu[,2], col="black",pch=3,cex=5)

#' Draw 1.64 sd level sets
for(j in 1:Kfit){
    ellips <- draw_sd(outputem$last$Mu[j,], outputem$last$Sigma[,,j])
    lines(ellips[1,], ellips[2,], col='orange')
}

#' add the real level sets
for(j in 1:K){
    ellips <- draw_sd(Mu[j,], Sigma[,,j])
    lines(ellips[1,], ellips[2,], col='black')
}
```



where the orange ellipson represents the region of 1.64 sigma (95% quantile) given by the EM algorithm, while the black ellipson reprensents the real level set.

check the objective function

```
plot(outputem$objective, type = 'p',
     main = 'objective', xlab = 'iteration',
     ylab = 'objective value', col='blue')
```



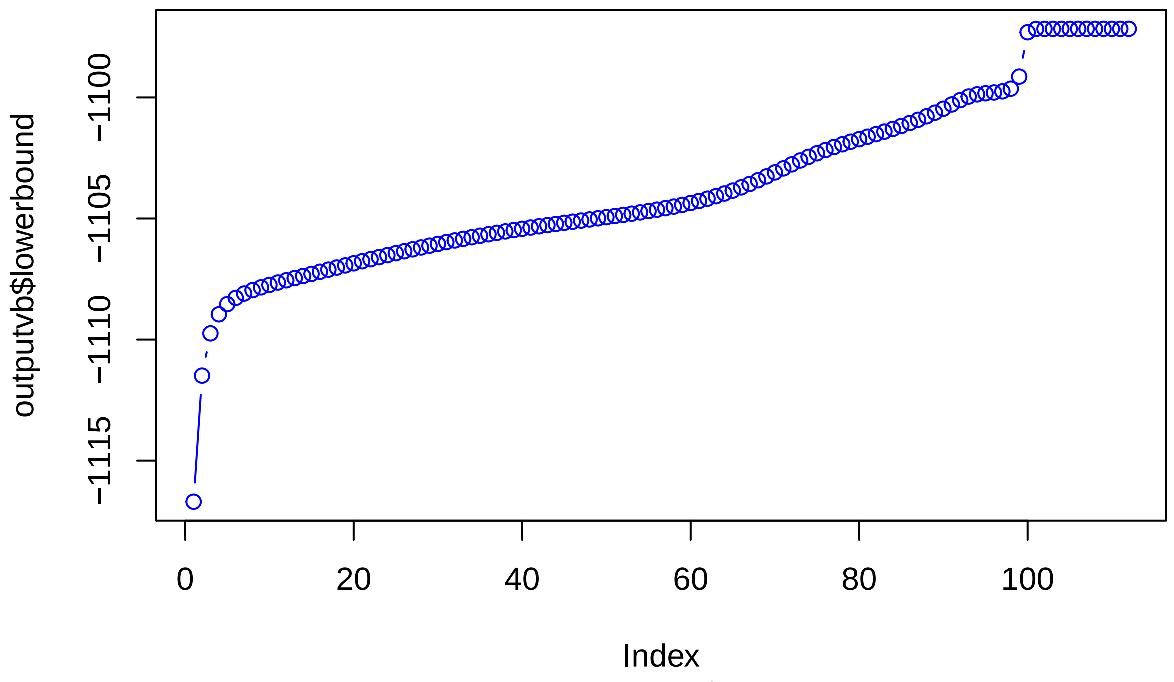
3 VB

Initialization

```
#' Bayesian model:
#' p ~ dirichlet(alpha);  alpha = (alpha0, ... , alpha0)
#' [ xi | p ] ~ Multinomial(p)
#' [ mu_j | Lambda_j ] ~ Normal(m0, beta0 Lambda_j^(-1))
#' Lambda_j ~ Wishart(W0, nu0)
#' [ X| xi=j, mu, Lambda ] ~ Normal (mu_j, Lambda_j^(-1))

#' hyper-parameters : to be varied
alpha0 <- 0.1
m0 <- rep(0,2)
beta0 <- 0.1
W0 <- 1*diag(2)
nu0 <- 10
```

```
#' Run VB
#'
seed <- 10
set.seed(seed)
outputvb <- vbalgo(x=X,k=Kfit, alpha0 = alpha0, W0inv = solve(W0),
                   nu0 = nu0, m0 = m0, beta0=beta0, tol=1e-6)
```

```
#' plot the lowerbound over iterations
plot(outputvb$lowerbound, col='blue', type='b')
```



the lower bound < maximum likelyhood and the gap is the KL function

```
##' show a summary of VB's output
T <- ncol(outputvb$alphamat)
outputvb$alphamat[,T]
```

```
## [1] 262.67386  90.67938 146.94676   0.10000
```

```
outputvb$Marray[,,T]
```

```
##               [,1]          [,2]
## [1,]  1.027284e+00 -1.016041e+00
## [2,] -4.395968e-02  1.028035e+00
## [3,] -1.038371e+00 -1.039631e+00
## [4,]  3.877258e-10 -4.759042e-10
```

```
#' Visual summary of VB's output :
#' posterior expectancy of each parameter
p_vb <- outputvb$alphamat[,T] / sum(outputvb$alphamat[,T]) ## complete the code
    ## (variational posterior expectancy of mixture weights)
Mu_vb <- outputvb$Marray[,,T] ## complete the code
    ## (variational posterior expectancy of mixture centers)
Sigma_vb <- array(dim=c(d,d,Kfit))
for(j in 1:Kfit){
    Sigma_vb[,,j] <- 1/outputvb$Numat[,T][j]* (outputvb$Winvarray[,,j,T]) ## complete the code
    ## (variational posterior expectancy of mixture covariances)
}

## show the data, true centers and initial positions from K-means
graphics.off()
plot(X[,1], X[,2], col=labs)
points(Mu[,1],Mu[,2], col="black",pch=8,cex=10*p)
```

```r
set.seed(seed)
Init <-  initem(X,Kfit)
points(Init$Mu[,1],Init$Mu[,2], col="orange",pch=18,cex = 10*Init$p)
```

```r
## show the data, true centers and initial positions from K-means
graphics.off()
plot(X[,1], X[,2], col=labs)
points(Mu[,1],Mu[,2], col="black",pch=8,cex=10*p)
set.seed(seed)
Init <-  initem(X,Kfit)
points(Init$Mu[,1],Init$Mu[,2], col="orange",pch=18,cex = 10*Init$p)

## Add a  summary of the VB solution
nonneg <- which(p_vb>0.001)
for(j in nonneg){
    points(Mu_vb[j,1], Mu_vb[j,2], col="blue",
           pch=18,cex= 10 * p_vb[j])
    ellips <- draw_sd(mu = Mu_vb[j,],
                      sigma = Sigma_vb[,,j])
    lines(ellips[1,], ellips[2,], col='blue')
}

#' add the real level sets
for(j in 1:K){
    ellips <- draw_sd(Mu[j,], Sigma[,,j])
    lines(ellips[1,], ellips[2,], col='black')
}
```

where the blue ellipson is the VB Algorithm results and the black ones are generated by the real parametres. We saw that the VB eliminated the overfitting. compared with the EM or Kmeans

Study the influence of the hyperparameter alpha

```r
alpha0 <- 10
outputvb <- vbalgo(x=X,k=Kfit, alpha0 = alpha0, W0inv = solve(W0),
                 nu0 = nu0, m0 = m0, beta0=beta0, tol=1e-6)
#' posterior expectancy of each parameter
p_vb <- outputvb$alphamat[,T] / sum(outputvb$alphamat[,T]) ## complete the code
    ## (variational posterior expectancy of mixture weights)
Mu_vb <- outputvb$Marray[,,T] ## complete the code
    ## (variational posterior expectancy of mixture centers)
Sigma_vb <- array(dim=c(d,d,Kfit))
for(j in 1:Kfit){
    Sigma_vb[,,j] <- 1/outputvb$Numat[,T][j]* (outputvb$Winvarray[,,j,T]) ## complete the code
    ## (variational posterior expectancy of mixture covariances)
}
graphics.off()
plot(X[,1], X[,2], col=labs)
points(Mu[,1],Mu[,2], col="black",pch=8,cex=10*p)
set.seed(seed)
Init <-  initem(X,Kfit)
points(Init$Mu[,1],Init$Mu[,2], col="orange",pch=18,cex = 10*Init$p)
```

```r
## Add a  summary of the VB solution
nonneg <- which(p_vb>0.001)
for(j in nonneg){
    points(Mu_vb[j,1], Mu_vb[j,2], col="blue",
           pch=18,cex= 10 * p_vb[j])
    ellips <- draw_sd(mu = Mu_vb[j,],
                      sigma = Sigma_vb[,,j])
    lines(ellips[1,], ellips[2,], col='blue')
}
#' add the real level sets
for(j in 1:K){
    ellips <- draw_sd(Mu[j,], Sigma[,,j])
    lines(ellips[1,], ellips[2,], col='black')
}
```

we see that when alpha0 is large (eg. 10) the number of cluster given by VB becomes 4. This means that VB approaches the EM results.

Study the influence of the other hyper-parameters.

```r
#' still use alpha0 = 0.1
alpha0 <- 0.1
m0 <- rep(0,2)
beta0 <- 0.1
W0 <- 1*diag(2)
#' use a large value
# nu0 <- 1000
nu0 <- 2

outputvb <- vbalgo(x=X,k=Kfit, alpha0 = alpha0, W0inv = solve(W0),
                 nu0 = nu0, m0 = m0, beta0=beta0, tol=1e-6)
T <- ncol(outputvb$alphamat)
#' posterior expectancy of each parameter
p_vb <- outputvb$alphamat[,T] / sum(outputvb$alphamat[,T]) ## complete the code
    ## (variational posterior expectancy of mixture weights)
Mu_vb <- outputvb$Marray[,,T] ## complete the code
    ## (variational posterior expectancy of mixture centers)
Sigma_vb <- array(dim=c(d,d,Kfit))
for(j in 1:Kfit){
    Sigma_vb[,,j] <- 1/outputvb$Numat[,T][j]* (outputvb$Winvarray[,,j,T]) ## complete the code
    ## (variational posterior expectancy of mixture covariances)
}
graphics.off()
plot(X[,1], X[,2], col=labs)
points(Mu[,1],Mu[,2], col="black",pch=8,cex=10*p)
set.seed(seed)
Init <-  initem(X,Kfit)
points(Init$Mu[,1],Init$Mu[,2], col="orange",pch=18,cex = 10*Init$p)

## Add a  summary of the VB solution
nonneg <- which(p_vb>0.001)
for(j in nonneg){
    points(Mu_vb[j,1], Mu_vb[j,2], col="blue",
           pch=18,cex= 10 * p_vb[j])
    ellips <- draw_sd(mu = Mu_vb[j,],
                      sigma = Sigma_vb[,,j])
    lines(ellips[1,], ellips[2,], col='blue')
}
#' add the real level sets
for(j in 1:K){
    ellips <- draw_sd(Mu[j,], Sigma[,,j])
    lines(ellips[1,], ellips[2,], col='black')
}
```

we conclude that… when nu is large, the number of freedom increase and the sigma becomes small. so the level set becomes small . when beta is large (eg. 1000), the distribution approaches a normal distribution, the cluster number becomes 2 and sigma becomes large while the center moves to the center of three cluster

```r
#' still use iriginal values
alpha0 <- 0.1
m0 <- rep(0,2)
beta0 <- 0.1
W0 <- 1*diag(2)
nu0 <- 10

outputvb <- vbalgo(x=X,k=Kfit, alpha0 = alpha0, W0inv = solve(W0),
                   nu0 = nu0, m0 = m0, beta0=beta0, tol=1e-6)
T <- ncol(outputvb$alphamat)
```

4 Metropolis-Hastings

Basic testing

```r
#' Basic testing for the MH sampler

Kmc <- Kfit ## try with different values, here Kfit = 4
init <- initem(x=X, k=Kmc)

hpar <- list( alpha0=rep(alpha0, Kmc),
           m0 = rep(0, d), beta0 = beta0,
           W0 = W0, nu0 = nu0)

ppar <- list(var_Mu = 0.001,
            nu_Sigma = 500,
            alpha_p = 500)


set.seed(1)
pct <- proc.time()
outputmh <- MHsample(x=X, k=Kmc, nsample= 10000,
                    init=init, hpar=hpar, ppar=ppar)
newpct <- proc.time()
elapsed <- newpct - pct
elapsed
```

```
##    user  system elapsed
##  81.680   0.717  85.216
```

```
outputmh$naccept ## should not be ridiculously low.
```

```
## [1] 1052
```

Heidelberger and Welch's convergence diagnostic

```
y_0 <- mcmc(data=cdfTrace(c(0,0), sample = outputmh, burnin = 1000, thin = 5))
heidel.diag(y_0)
```

```
##
##      Stationarity start     p-value
##      test         iteration
## var1 passed       1         0.294
##
##      Halfwidth Mean  Halfwidth
##      test
## var1 passed    0.283 0.0033
```

```
y_1 <- mcmc(data=cdfTrace(c(1,1), sample = outputmh, burnin = 1000, thin = 5))
heidel.diag(y_1)
```

```
##
##      Stationarity start     p-value
##      test         iteration
## var1 passed       541       0.255
##
##      Halfwidth Mean  Halfwidth
##      test
## var1 passed    0.626 0.00487
```

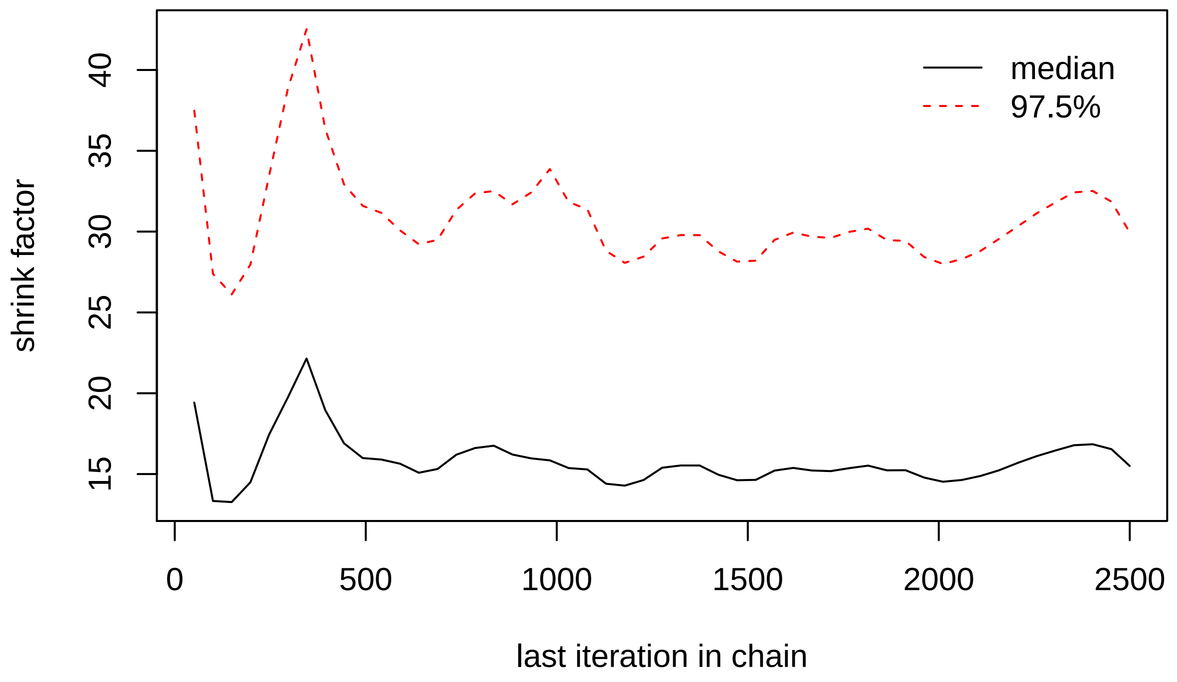
the test passed with about 10% datas discarded (burnin), so we say that we cannot reject the null hypothesis that the time series is stationary, and we accept that within 1000 iterations the time series converges

Gelman and Rubin's diagnostic

```
mcObject_0 <- mcmc(data=cdfTrace(c(1,1), sample = outputmh, burnin = 5000, thin = 2))
mcObject_1 <- mcmc(data=cdfTrace(c(0,1), sample = outputmh, burnin = 5000, thin = 2))
mcObject_2 <- mcmc(data=cdfTrace(c(0,0), sample = outputmh, burnin = 5000, thin = 2))
mcList <- mcmc.list(mcObject_0, mcObject_1, mcObject_2)
gelman.diag(mcList)
```

```
## Potential scale reduction factors:
##
##      Point est. Upper C.I.
## [1,]       15.5         30
```

```
gelman.plot(mcList)
```



rerun for several times to get the shrink factor close to 1~1.1

```
set.seed(3)
pct <- proc.time()
outputmh <- MHsample(x=X, k=Kmc, nsample= 5000,
                    init=init, hpar=hpar, ppar=ppar)
newpct <- proc.time()
elapsed <- newpct - pct
elapsed
```

```
##    user  system elapsed
##  36.848   0.095  37.025
```

```
outputmh$naccept ## should not be ridiculously low.
```

```
## [1] 584
```

```
y_0 <- mcmc(data=cdfTrace(c(0,0), sample = outputmh, burnin = 1000, thin = 5))
heidel.diag(y_0)
```

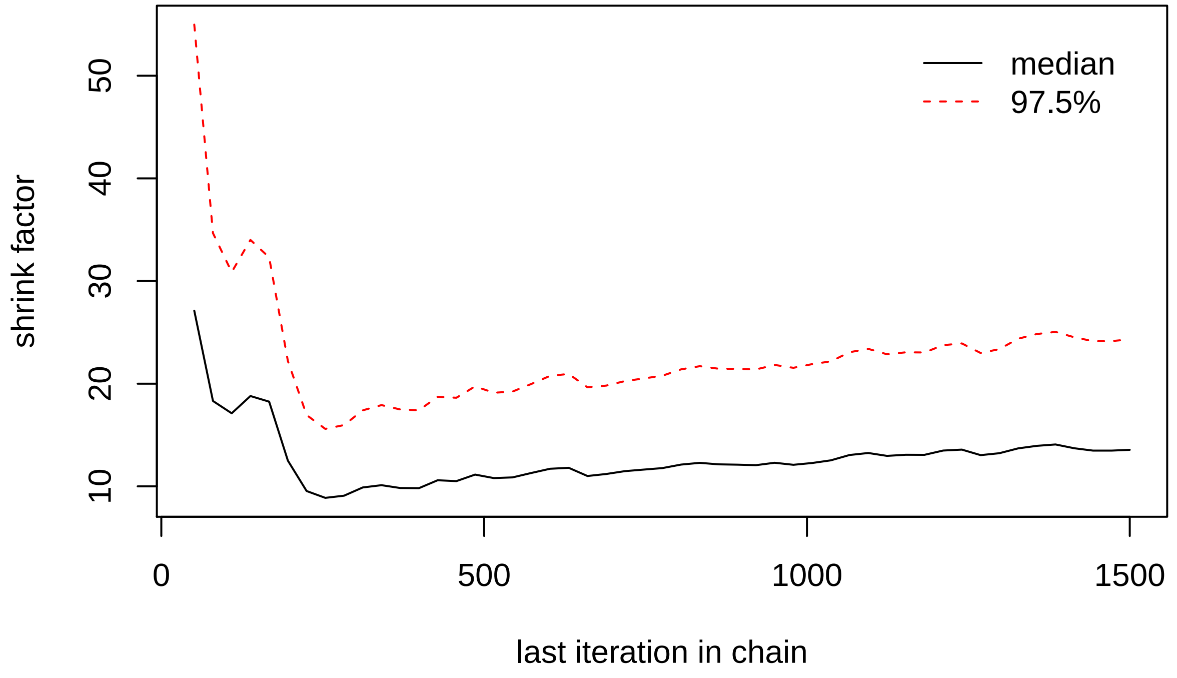
```
##
##      Stationarity start     p-value
##      test         iteration
## var1 passed       1         0.201
##
##      Halfwidth Mean  Halfwidth
##      test
## var1 passed    0.287 0.00419
```

```
mcObject_0 <- mcmc(data=cdfTrace(c(1,1), sample = outputmh, burnin = 2000, thin = 2))
mcObject_1 <- mcmc(data=cdfTrace(c(0,1), sample = outputmh, burnin = 2000, thin = 2))
mcObject_2 <- mcmc(data=cdfTrace(c(0,0), sample = outputmh, burnin = 2000, thin = 2))
mcObject_3 <- mcmc(data=cdfTrace(c(1,0), sample = outputmh, burnin = 2000, thin = 2))

mcList <- mcmc.list(mcObject_0, mcObject_1, mcObject_2, mcObject_3)
```

```
gelman.diag(mcList)
```

```
## Potential scale reduction factors:
##
##      Point est. Upper C.I.
## [1,]       13.6       24.3
```

```
gelman.plot(mcList)
```



Predictive density

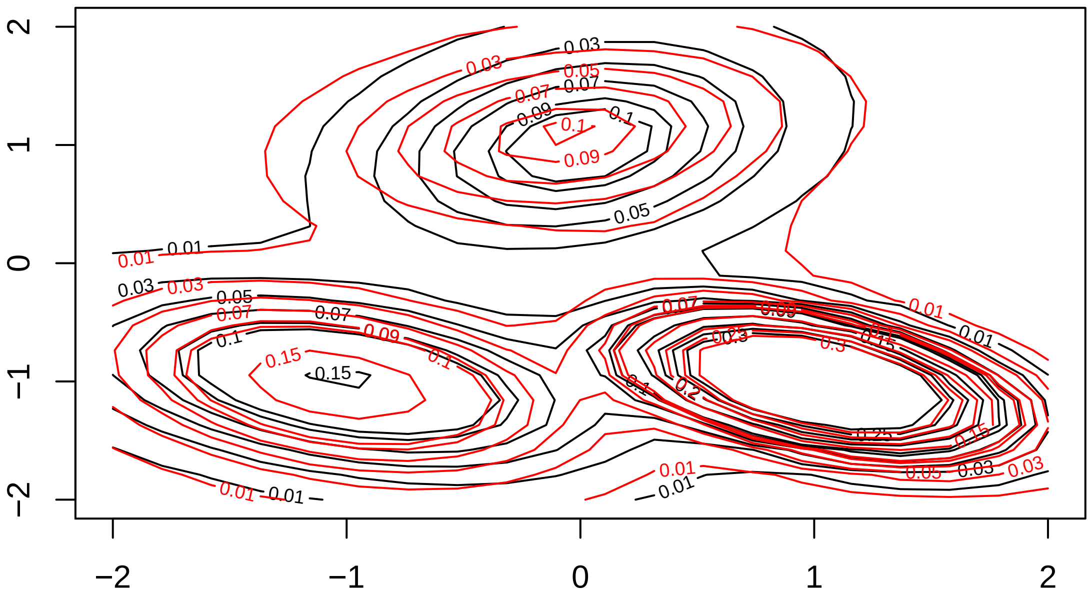
```
xx <- seq(-2,2,length.out=20)
yy <- xx
dtrue <- outer(X= xx, Y=yy,
            FUN = function(x,y){
                wrapper(x=x, y=y,
                        FUN=function(u,v){
                          exp(gmllk(x = c(u,v), Mu = Mu,
                          Sigma = Sigma, p = p))
                        })
            })
```

dtrue is the value generated by the real distribution

```
dpredmh <-  outer(X= xx, Y=yy,
            FUN = function(x,y){
                wrapper(x = x, y = y,
                        FUN =function(u,v){
                          ## complete the code
                          MHpredictive(c(u, v), outputmh, burnin = 2000, thin = 5)
                          })
            })
```

warning: the burnin cannot be too small otherwise it will take long time to run

```r
#' plot the graph grids
breaks <- c(seq(0.01,0.09, length.out=5),seq(0.1,0.3,length.out=5))
nbreaks <- length(breaks)
contour(xx,yy, z = dtrue, nlevels=nbreaks, levels = breaks)
contour(xx,yy, z = dpredmh,  nlevels=nbreaks, levels = breaks,
          add=TRUE, col='red')
```

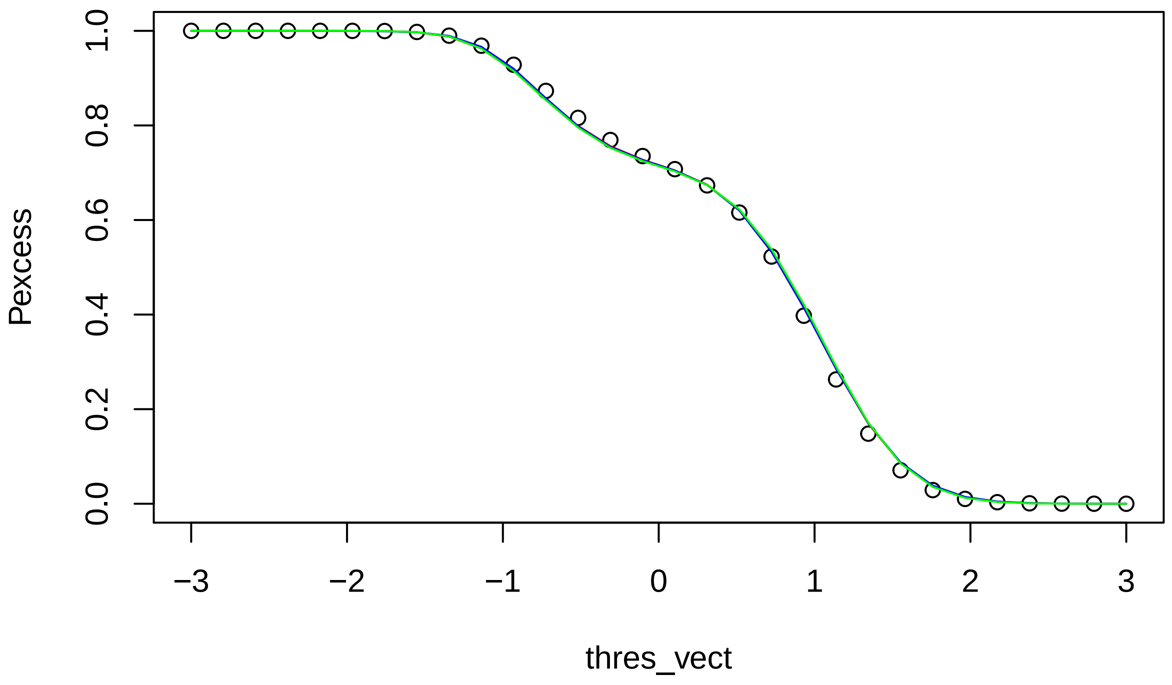


as a test set we didn't conclude that our algo is well converged, yet it captures the clusters of the real distribution.

5 predictive Cdf's

```r
Pexcess <- rep(0,10)
Pexcess_em <- Pexcess; Pexcess_vb <- Pexcess; Pexcess_mh <- Pexcess
thres_vect <-  seq(-3, 3, length.out=30)
#seq(1, 5, length.out=30)
Tem <- length(outputem$objective)
T <- ncol(outputvb$alphamat)

for(i in seq_along(thres_vect)){
threshold <- rep(thres_vect[i], 2)
Pexcess[i] <- 1 - gmcdf(x = threshold, Mu = Mu, Sigma=Sigma, p=p)
Pexcess_em[i] <- 1 - gmcdf(x = threshold, Mu = outputem$Muarray[,,Tem],
                           Sigma = outputem$Sigmaarray[,,,Tem], p = outputem$parray[,-1] )
                  ## complete the code:
                 ##maximum likelihood estimator using EM output

Pexcess_vb[i] <- 1 - vbPredictiveCdf(x = threshold,
                                     alpha = outputvb$alphamat[,T],
                                     Beta = outputvb$Betamat[, T],
                                     M = outputvb$Marray[,,T],
                                     Winv = outputvb$Winvarray[,,,T],
                                     Nu = outputvb$Numat[,T] )  ## complete the code:
    ## posterior predictive  estimator using VB output:
```

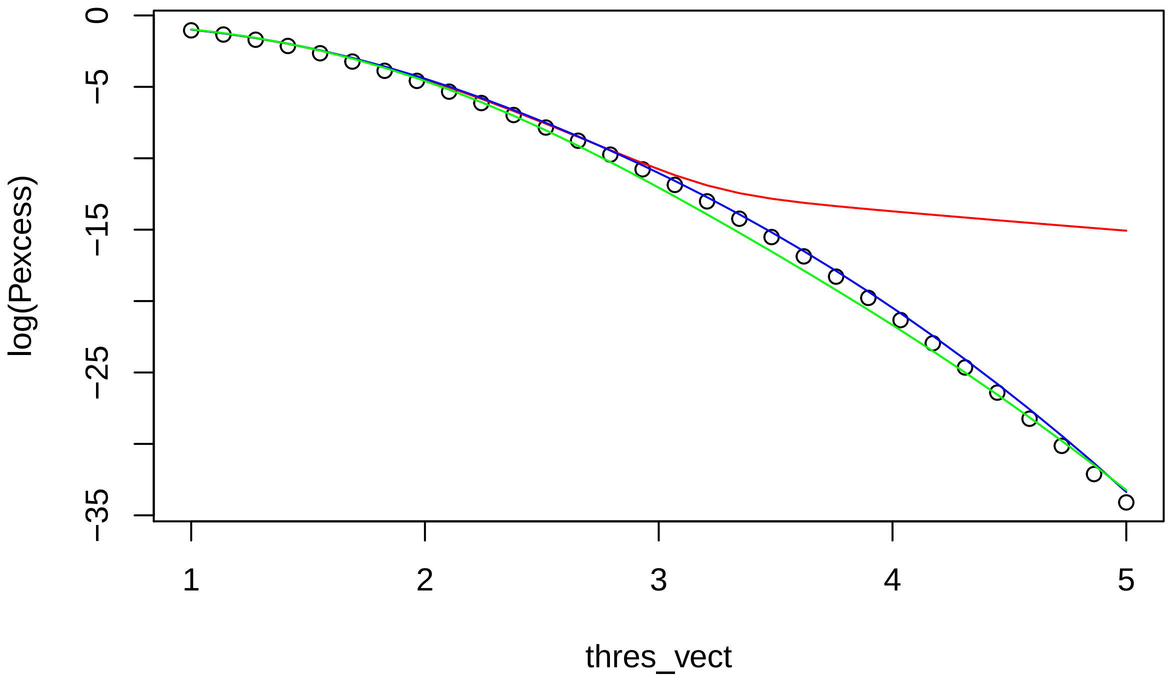
```
    ## use vbPredictiveCdf

Pexcess_mh[i] <- 1 - MHpredictiveCdf(x = threshold, sample = outputmh, burnin = 2900, thin = 1)
    ## complete the code:
    ## posterior predictive  estimator using MH output:
    ## use MHpredictiveCdf.
}
ylim <- range(Pexcess, Pexcess_em,Pexcess_vb)
plot(thres_vect,Pexcess, ylim = ylim)
lines(thres_vect, Pexcess_vb, col='red')
lines(thres_vect, Pexcess_em, col='blue')
lines(thres_vect, Pexcess_mh, col='green')
```



seen from the threshold vector from -3 to 3, the cummulative distribution function is well coincided.

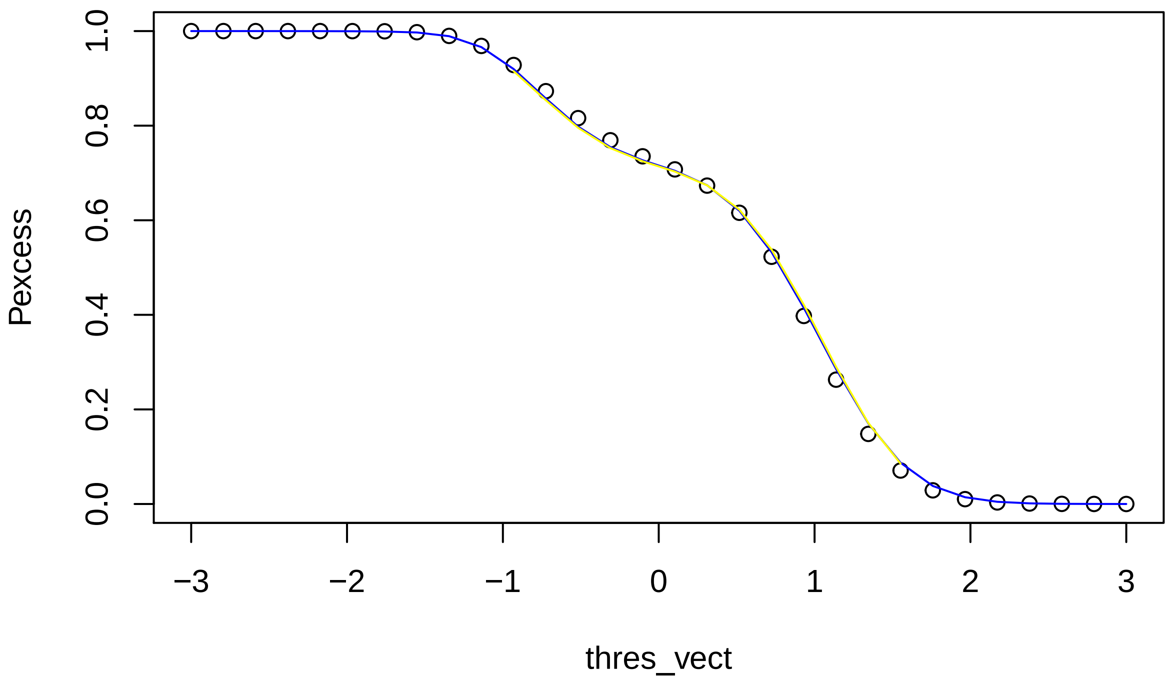
```
Pexcess <- rep(0,10)
Pexcess_em <- Pexcess; Pexcess_vb <- Pexcess; Pexcess_mh <- Pexcess
thres_vect <-  seq(1, 5, length.out=30)
Tem <- length(outputem$objective)
T <- ncol(outputvb$alphamat)

for(i in seq_along(thres_vect)){
  threshold <- rep(thres_vect[i], 2)
  Pexcess[i] <- 1 - gmcdf(x = threshold, Mu = Mu, Sigma=Sigma, p=p)
  Pexcess_em[i] <- 1 - gmcdf(x = threshold, Mu = outputem$Muarray[,,Tem],
                          Sigma = outputem$Sigmaarray[,,,Tem], p = outputem$parray[,-1] )
  ## complete the code:
  ##maximum likelihood estimator using EM output

  Pexcess_vb[i] <- 1 - vbPredictiveCdf(x = threshold,
                                     alpha = outputvb$alphamat[,T],
                                     Beta = outputvb$Betamat[, T],
                                     M = outputvb$Marray[,,T],
```

```r
                                        Winv = outputvb$Winvarray[,,,T],
                                        Nu = outputvb$Numat[,T] )  ## complete the code:
  ## posterior predictive  estimator using VB output:
  ## use vbPredictiveCdf

  Pexcess_mh[i] <- 1 - MHpredictiveCdf(x = threshold, sample = outputmh, burnin = 2900, thin = 1)
  ## complete the code:
  ## posterior predictive  estimator using MH output:
  ## use MHpredictiveCdf.
}
ylim <- range(log(Pexcess), log(Pexcess_em),log(Pexcess_vb))
plot(thres_vect,log(Pexcess), ylim = ylim)
lines(thres_vect, log(Pexcess_vb), col='red')
lines(thres_vect, log(Pexcess_em), col='blue')
lines(thres_vect, log(Pexcess_mh), col='green')
```



at the tail value the VB and MH algorithm seems to underestimate the CDF

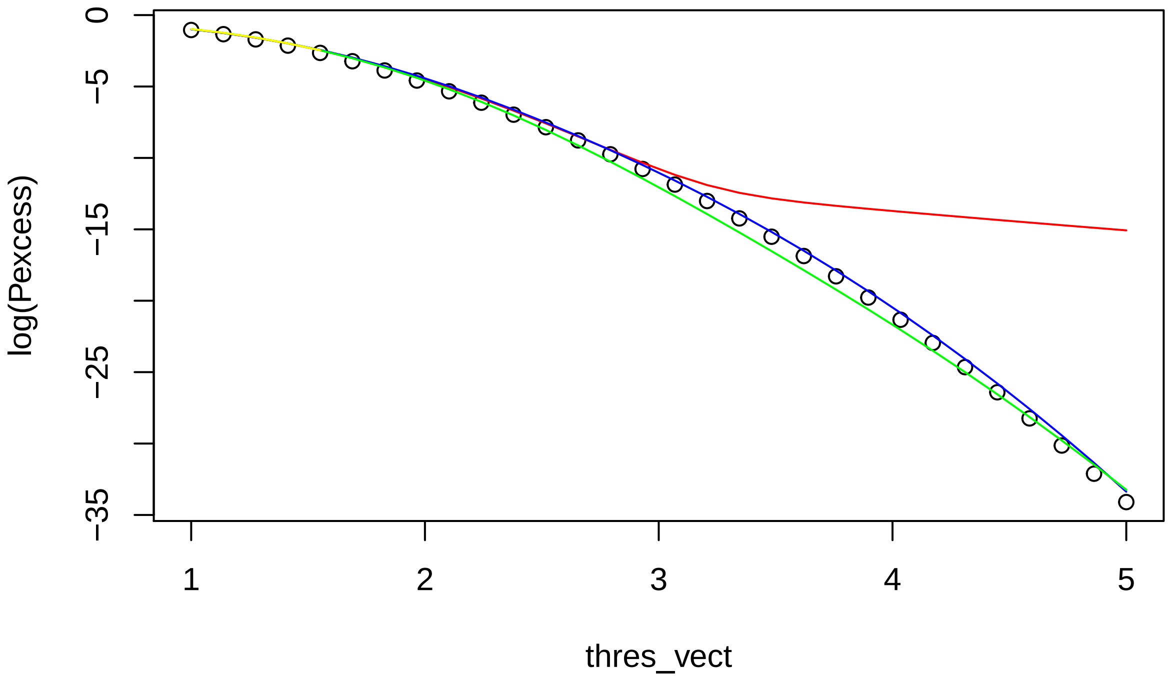
```r
Pexcess <- rep(0,10)
Pexcess_em <- Pexcess; Pexcess_vb <- Pexcess; Pexcess_mh <- Pexcess
thres_vect <-  seq(-3, 3, length.out=30)
#seq(1, 5, length.out=30)
Tem <- length(outputem$objective)
T <- ncol(outputvb$alphamat)

for(i in seq_along(thres_vect)){
threshold <- rep(thres_vect[i], 2)
Pexcess[i] <- 1 - gmcdf(x = threshold, Mu = Mu, Sigma=Sigma, p=p)
Pexcess_em[i] <- 1 - gmcdf(x = threshold, Mu = outputem$Muarray[,,Tem],
                           Sigma = outputem$Sigmaarray[,,,Tem], p = outputem$parray[,-1] )
                  ## complete the code:
                 ##maximum likelihood estimator using EM output
Pexcess_vb[i] <- 1 - vbPredictiveCdf(x = threshold,
                                     alpha = outputvb$alphamat[,T],
                                     Beta = outputvb$Betamat[, T],
                                     M = outputvb$Marray[,,T],
                                     Winv = outputvb$Winvarray[,,,T],
                                     Nu = outputvb$Numat[,T] )  ## complete the code:
    ## posterior predictive  estimator using VB output:
    ## use vbPredictiveCdf

Pexcess_mh[i] <- 1 - MHpredictiveCdf(x = threshold, sample = outputmh, burnin = 2900, thin = 1)
    ## complete the code:
    ## posterior predictive  estimator using MH output:
    ## use MHpredictiveCdf.
}
ylim <- range(Pexcess, Pexcess_em,Pexcess_vb)
plot(thres_vect, Pexcess, ylim = ylim)
lines(thres_vect, Pexcess_em, col='blue')

credit <- Pexcess_mh < 0.95 & Pexcess_mh >0.05
credit_thres <- sapply(1:length(credit), function(j){
  if(credit[j] == TRUE) thres_vect[j] else NaN
})
credit_Pexcess_mh <- sapply(1:length(credit), function(j){
  if(credit[j]==TRUE) Pexcess_mh[j] else NaN
})

lines(credit_thres, credit_Pexcess_mh, col='yellow')
```



we show that the green value from 0.05 to 0.95 is well fitted, the same for the log graph:

```
Pexcess <- rep(0,10)
Pexcess_em <- Pexcess; Pexcess_vb <- Pexcess; Pexcess_mh <- Pexcess
thres_vect <-  seq(1, 5, length.out=30)
```

```r
Tem <- length(outputem$objective)
T <- ncol(outputvb$alphamat)

for(i in seq_along(thres_vect)){
  threshold <- rep(thres_vect[i], 2)
  Pexcess[i] <- 1 - gmcdf(x = threshold, Mu = Mu, Sigma=Sigma, p=p)
  Pexcess_em[i] <- 1 - gmcdf(x = threshold, Mu = outputem$Muarray[,,Tem],
                          Sigma = outputem$Sigmaarray[,,,Tem], p = outputem$parray[,-1] )
  ## complete the code:
  ##maximum likelihood estimator using EM output

  Pexcess_vb[i] <- 1 - vbPredictiveCdf(x = threshold,
                                    alpha = outputvb$alphamat[,T],
                                    Beta = outputvb$Betamat[, T],
                                    M = outputvb$Marray[,,T],
                                    Winv = outputvb$Winvarray[,,,T],
                                    Nu = outputvb$Numat[,T] )  ## complete the code:
  ## posterior predictive  estimator using VB output:
  ## use vbPredictiveCdf

  Pexcess_mh[i] <- 1 - MHpredictiveCdf(x = threshold, sample = outputmh, burnin = 2900, thin = 1)
  ## complete the code:
  ## posterior predictive  estimator using MH output:
  ## use MHpredictiveCdf.
}
ylim <- range(log(Pexcess), log(Pexcess_em),log(Pexcess_vb))
plot(thres_vect,log(Pexcess), ylim = ylim)
lines(thres_vect, log(Pexcess_vb), col='red')
lines(thres_vect, log(Pexcess_em), col='blue')
lines(thres_vect, log(Pexcess_mh), col='green')


credit <- Pexcess_mh < 0.95 & Pexcess_mh >0.05
credit_thres <- sapply(1:length(credit), function(j){
  if(credit[j] == TRUE) thres_vect[j] else NaN
})
credit_Pexcess_mh <- sapply(1:length(credit), function(j){
  if(credit[j]==TRUE) Pexcess_mh[j] else NaN
})

lines(credit_thres, log(credit_Pexcess_mh), col='yellow')
```



we show that the credit interval (yellow part) avoid the tail values.