Homework 2: Numerical Optimization and the E-M Algorithm

Runyan Xin

2025-09-23

Instructions

- Please submit your R Markdown file (`.Rmd`) and the knitted PDF.
- Write clear code, comments, and short written explanations of your results.
- You may use external packages for **data handling and visualization**, but for optimization and EM you should **write your own functions** (except `optim()` for Problem 1).

Problem 1: Ridge Regression with `optim`

In this problem, you will practice writing custom loss functions and using `optim()` to solve regression problems with regularization. Use the **Boston** dataset from the `MASS` package, where the response is median house value (`medv`) and predictors are the other variables.

(a) Load the `Boston` dataset and standardize the predictors and response (center and scale them).

```
library(MASS)
data("Boston")
stan_Boston<-scale(Boston,center = T,scale = T)
X_Boston<-stan_Boston[ ,c(1:13)]
y_Boston<-stan_Boston[ ,14]
```

(b) Write an R function that computes the **ridge regression objective function**:

$$L(\beta) = \sum_{i=1}^{n} (y_i - x_i^T \beta)^2 + \lambda \sum_{j=1}^{p} \beta_j^2,$$

where $\lambda \geq 0$ is a penalty parameter. Test your function on a toy example to make sure it works.

```
ridge_regression<- function(beta, x, y, lambda) {
  l<-sum((y - x %*% beta)^2) + lambda * sum(beta^2)
  return(l)
}

# toy data
X_toy<- matrix(c(1,2,3,4,5,6,7,8,9), nrow=3, ncol=3)  # 3x3 matrix
beta_toy<- c(10,20,30)
y_toy<-X_toy%*%beta_toy
```

```r
ridge_regression(beta_toy,X_toy,y_toy,lambda=0)
```

```
## [1] 0
```

```r
ridge_regression(beta_toy,X_toy,y_toy,lambda=5)
```

```
## [1] 7000
```

(c) Use `optim()` to minimize the ridge objective function for several values of λ (e.g., $\lambda = 0.01, 0.1, 1, 10$). Compare the solutions and comment on how the coefficients shrink as λ increases.

```r
lambda_<-c(0,0.1,1,10,100,200,300,400,500,600,700)
opt_results<-matrix(NA,nrow=length(lambda_),ncol=3)
for(i in 1:11){
  opt<-optim(par=rep(0,3),fn=ridge_regression,x=X_toy,y=y_toy,lambda=lambda_[i],method="BFGS")
  opt_results[i, ]<-opt$par  #the value of loss

}
opt_results
```

```
##            [,1]      [,2]      [,3]
##  [1,] 10.000000 20.000000 30.000000
##  [2,]  9.838019 19.948370 30.058722
##  [3,]  9.049179 19.672131 30.295083
##  [4,]  7.953572 18.839343 29.725105
##  [5,]  5.956111 14.384913 22.813725
##  [6,]  4.718672 11.410134 18.101599
##  [7,]  3.908680  9.455367 15.002048
##  [8,]  3.336345  8.072491 12.808647
##  [9,]  2.910309  7.042536 11.174772
## [10,]  2.580795  6.245672  9.910560
## [11,]  2.318326  5.610813  8.903311
```

```r
#The coefficients become smaller as  increases; the larger the , the stronger the shrinkage effect.
```

(d) Based on your results, which predictors seem most important for explaining median house value?

```r
lambda_<-c(0,0.01,0.1,1,10)
opt_results<-matrix(NA,nrow=length(lambda_),ncol=13)
for(i in 1:5){
  opt<-optim(par=rep(0,13),fn=ridge_regression,x=X_Boston,y=y_Boston,lambda=lambda_[i],method="BFGS")
  opt_results[i, ]<-opt$par  #the value of loss

}
opt_results
```

```
##             [,1]      [,2]        [,3]       [,4]       [,5]      [,6]
## [1,] -0.10101707 0.1177152  0.01533523 0.07419883 -0.2238480 0.2910565
## [2,] -0.10100784 0.1176984  0.01530875 0.07420265 -0.2238174 0.2910660
## [3,] -0.10092496 0.1175471  0.01507120 0.07423689 -0.2235417 0.2911517
```

```
## [4,] -0.10011472 0.1160678  0.01277099 0.07456819 -0.2208265 0.2919859
## [5,] -0.09348469 0.1039146 -0.00452721 0.07703709 -0.1972343 0.2984810
##              [,7]       [,8]      [,9]      [,10]      [,11]      [,12]
## [1,]  0.002118647 -0.3378363 0.2897490 -0.2260317 -0.2242712 0.09243223
## [2,]  0.002111597 -0.3378070 0.2896697 -0.2259571 -0.2242621 0.09243178
## [3,]  0.002048313 -0.3375428 0.2889580 -0.2252878 -0.2241803 0.09242768
## [4,]  0.001430819 -0.3349242 0.2820416 -0.2188102 -0.2233746 0.09238643
## [5,] -0.003533435 -0.3108733 0.2282271 -0.1702916 -0.2163280 0.09193492
##           [,13]
## [1,] -0.4074469
## [2,] -0.4074327
## [3,] -0.4073043
## [4,] -0.4060336
## [5,] -0.3943969
```

```r
# lasta seems most important for explaining median house value
```

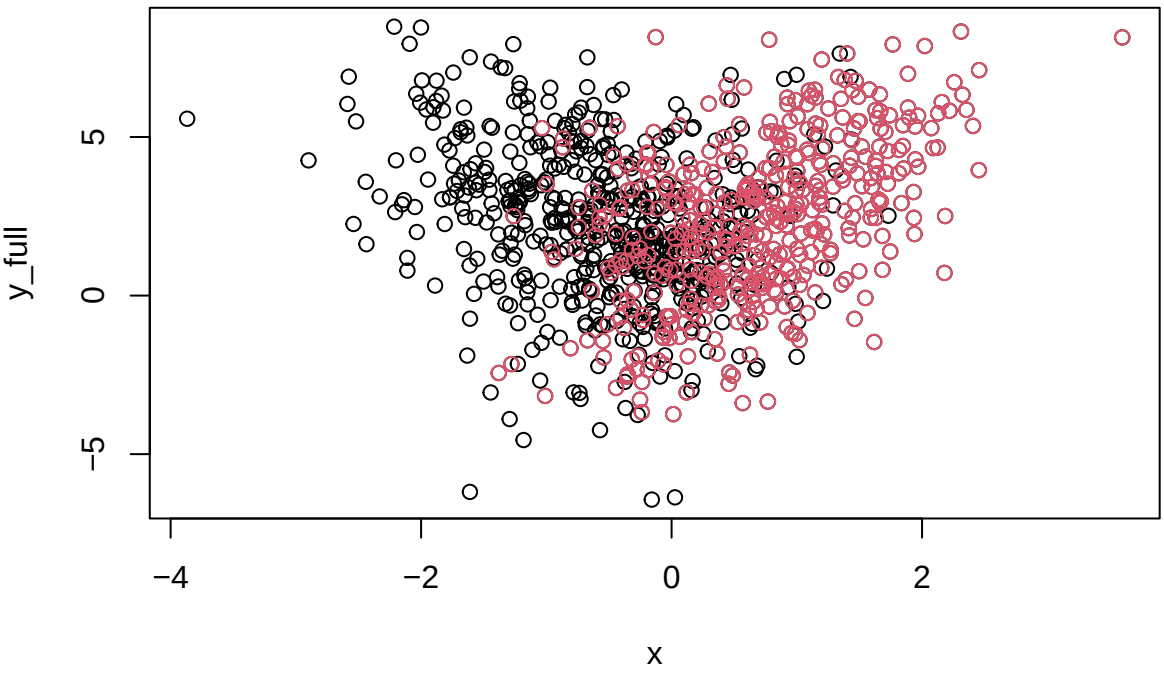
Problem 2: Estimating Regression Coefficients under Missing Data

Consider the data generating model, where there is missing-at-random data among the response.

```r
set.seed(134)
# simulate data
x<-rnorm(1000)
t<-rbinom(1000,1,pnorm(-x))
y_full<- rnorm(1000, 1+ 2*x + t - 3*t*x,2) #ground truth model

#simulate missingness
y_obs = y_full
r= rbinom(1000,1,pnorm(-x + .5*t))
y_obs[which(r ==1)] = NA


plot(x,y_full) # fully observed data
points(x,y_obs, col = 2)# complete cases
```



(a) Consider complete-case regression, of y on (x, t) (i.e. you only use observations for which you have completely observed response variables and covariates. In repeated simulation – i.e. generate data from the above model, and fit a complete-case regression in each simulated data set, saving the results from each simulated data set (perhaps using parallel computing) – how is inference on the regression coefficients affected by the missing data?

```
library(foreach)
library(doParallel)
```

```
## Loading required package: iterators
```

```
## Loading required package: parallel
```

```
num_cores<-parallel::detectCores()
cl<-makeCluster(num_cores)
registerDoParallel(cl)
n_iter=1000
results<-foreach(1:num_cores)%dopar%{

# simulate data

  sum_full_coef<-c(0,0,0,0)
  sum_cc_coef<-c(0,0,0,0)
  n_foreach<-ceiling(n_iter/num_cores)
  for(i in 1:n_foreach){

x<-rnorm(1000)
t<-rbinom(1000,1,pnorm(-x))
y_full<- rnorm(1000, 1+ 2*x + t - 3*t*x,2) #ground truth model
#simulate missingness
y_obs = y_full
```

```
r= rbinom(1000,1,pnorm(-x + .5*t))
y_obs[which(r ==1)] = NA

  fit_full<-lm(y_full~x+t+x:t)
  fit_cc<-lm(y_obs~x+t+x:t)
  sum_full_coef<-fit_full$coefficients+sum_full_coef
  sum_cc_coef<-fit_cc$coefficients+sum_cc_coef
  }
  mean_full<-sum_full_coef/n_foreach
  mean_cc<-sum_cc_coef/n_foreach
  return(c(mean_full,mean_cc))
}
stopCluster(cl)
results_mat <- do.call(rbind, results)
mean_coef_full<- colMeans(results_mat[,1:4])
mean_coef_cc<- colMeans(results_mat[,5:8])
coef_diff<- mean_coef_full - mean_coef_cc
coef_diff
```

```
##  (Intercept)            x            t          x:t
## -0.005293219  0.003596263  0.002075609  0.003809997
```

```
# the regression coefficients is almost not affected by missing data, it is
#probably due to a large number of repeated simulations
```

(b) In a repeated sampling study, implement an E-M algorithm to infer the regression coefficients. Derive the E- and the M-steps (Hint: use the normal-unknown mean and variance example from class). How does accounting for the missing data change the inferences?

```
mis_idx<- which(is.na(y_obs))
obs_idx<- which(!is.na(y_obs))


beta_1<-1
beta_2<-1
beta_3<-1
beta_4<-1
beta<-c(beta_1,beta_2,beta_3,beta_4)
for(i in 1:100){
  #step E:
  Y_mis_imputed <- beta_1 + beta_2*x[mis_idx] + beta_3*t[mis_idx] + beta_4*(t[mis_idx]*x[mis_idx])
  Y_complete <- y_obs
  Y_complete[mis_idx] <- Y_mis_imputed
  #step M:
  X_mat <- cbind(1, x, t, x*t)
  beta_new <- solve(t(X_mat) %*% X_mat) %*% t(X_mat) %*% Y_complete

  if(max(abs(beta-beta_new)) < 1e-6) break
  beta<-beta_new
}
beta
```

```
##         [,1]
```

```
##    1.0199571
## x  1.8557743
## t  0.6126675
##   -0.3280953
```

```
#beta_1 and beta_2 are not largely affected by the missing data. But the
#latter two coefficients are largely affected, because these coefficients are #associated with t, and t
```

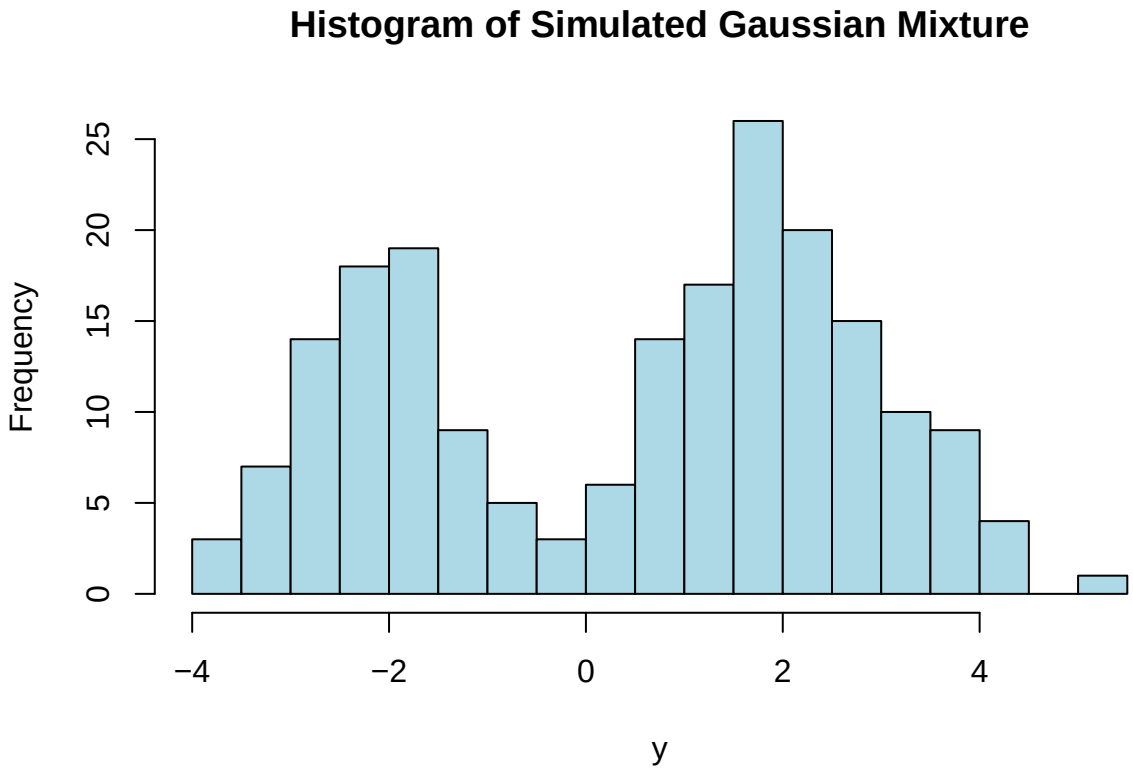
Problem 3: EM Algorithm for a Finite Gaussian Mixture

In this problem, you will implement the EM algorithm for a finite mixture of Gaussians.

We assume data $y_1, \dots, y_n \sim \sum_{k=1}^{K} \pi_k \mathcal{N}(\mu_k, \sigma^2)$, where: - π_k are mixture weights ($\sum_k \pi_k = 1, \pi_k \geq 0$), - μ_k are cluster means, - σ^2 is a shared variance across components.

(a) Simulate data with $n = 200$, $K = 2$, means $\mu = (-2, 2)$, weights $\pi = (0.4, 0.6)$, and variance $\sigma^2 = 1$. Plot a histogram of the data.

```
set.seed(123)
n_3<- 200
K_n<- 2
mu<- c(-2, 2)
pi<- c(0.4,0.6)
sigma<-1
z<- sample(1:K_n, size=200, replace = TRUE, prob = pi)
y <- rnorm(n_3, mean = mu[z], sd = sigma)
hist(y,breaks = 30, col = "lightblue", main = "Histogram of Simulated Gaussian Mixture",xlab ="y", ylab
```



(b) Write an R function to perform the **E-step**, which computes the responsibilities:

$$\gamma_{ik} = \frac{\pi_k \phi(y_i; \mu_k, \sigma^2)}{\sum_{j=1}^{K} \pi_j \phi(y_i; \mu_j, \sigma^2)},$$

where $\phi(\cdot)$ is the Gaussian density.

```
finite_Gaussian_Mixture_E<-function(y,pi_k,mu_k,sigma_k){
  n_ <- length(y)
  K_ <- length(mu_k)
  r<-matrix(NA,nrow = n_3,ncol = K_n)
  for(i in 1:n_){
  for(k in 1:K_){
    Gaussian_value<-dnorm(y[i],mean=mu_k[k],sd=sigma_k)
    up<-pi_k[k]*Gaussian_value
    low <- sum(pi_k * dnorm(y[i], mean = mu_k, sd = sigma_k))
    r[i,k]<-up/low
  }
  }
  return(r)
}
```

(c) Using the responsibilities γ_{ik}, implement the **M-step updates** for the parameters π_k, μ_k, σ^2.

```
#M step
pi_k_new<-colMeans(r)
    mu_k_new<-colSums(r*y)/colSums(r)
    sigma_k_new<-sqrt(sum(r*(y-matrix(mu_k_new, nrow=n_,ncol=K_, byrow=TRUE))^2)/n_)

if(max(abs(sigma_k-sigma_k_new),abs(mu_k-mu_k_new),abs(pi_k-pi_k_new))<1e-6) break

#update
  pi_k<-pi_k_new
  mu_k<-mu_k_new
  sigma_k<-sigma_k_new
```

(d) Combine your E-step and M-step into a full EM algorithm function. Run it until convergence (e.g., until parameter updates are smaller than 10^{-6}). Compare your estimated parameters with the true parameters used to generate the data.

```
Finite_Gaussian_Mixture_EM<-function(y){
  n_<-length(y)
  K_<-2
  initial_mu<-c(-1,1)
  initial_pi<-c(0.5,0.5)
  initial_sigma<-1
  pi_k<-initial_pi
  mu_k<-initial_mu
  sigma_k<-initial_sigma
  for(iter in 1:100){
    #E step:
    r<-finite_Gaussian_Mixture_E(y,pi_k,mu_k,sigma_k)
    #M step:
```

```
    pi_k_new<-colMeans(r)
    mu_k_new<-colSums(r*y)/colSums(r)
    sigma_k_new<-sqrt(sum(r*(y-matrix(mu_k_new, nrow=n_,ncol=K_, byrow=TRUE))^2)/n_)

if(max(abs(sigma_k-sigma_k_new),abs(mu_k-mu_k_new),abs(pi_k-pi_k_new))<1e-6) break

#update
  pi_k<-pi_k_new
  mu_k<-mu_k_new
  sigma_k<-sigma_k_new
  }

return(list(pi=pi_k, mu=mu_k, sigma=sigma_k))

}
Finite_Gaussian_Mixture_EM(y)
```

```
## $pi
## [1] 0.3854983 0.6145017
##
## $mu
## [1] -2.100576  2.044303
##
## $sigma
## [1] 0.9758479
```

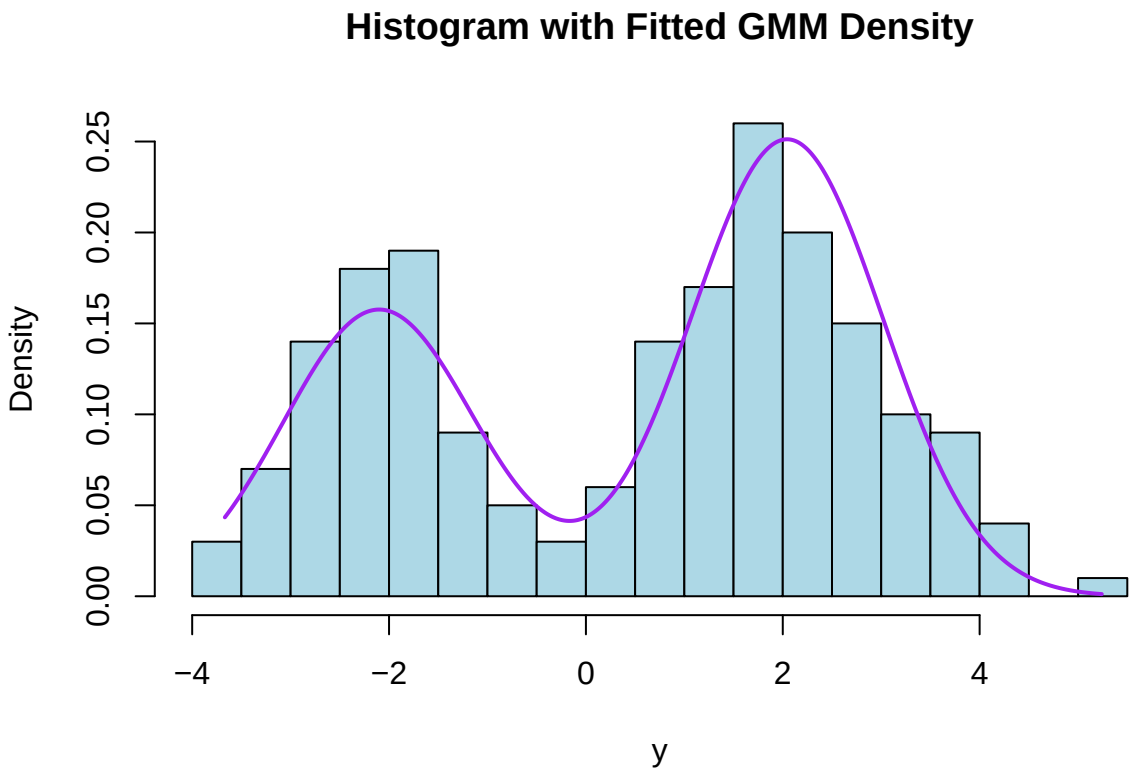
```
#comment: my estimated parameters is very close to the true parameters used to generate the data.
```

(e) Make a plot showing the fitted mixture density overlaid on the histogram of your data. Comment: does the fitted mixture capture the two subpopulations?

```
result <- Finite_Gaussian_Mixture_EM(y)
pi_k<-result$pi
mu_k<-result$mu
sigma_k<-result$sigma

hist(y, breaks=30, probability=TRUE,
     col="lightblue", border="black",
     main="Histogram with Fitted GMM Density",
     xlab="y")

mixture_density<-function(x,pi_k,mu_k,sigma_k) {
  pi_k[1]*dnorm(x, mean=mu_k[1], sd=sigma_k) + pi_k[2]*dnorm(x, mean=mu_k[2], sd=sigma_k)
}
x_<-seq(min(y),max(y),length.out=1000)
lines(x_,mixture_density(x_,pi_k,mu_k,sigma_k),col="purple",lwd=2)
```



(f) Using the NFL data below, fit a finite Gaussian mixture model for different choices of k, the number of clusters. Choose an evaluative criteria (you may search on the internet how to evaluate clustering models) and select a "best" number of clusters. Is there any interpretation for the clusters?

```
#install.packages('nflreadr')
#install.packages('lubridate')
library(nflreadr)
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.2     v tibble    3.3.0
## v lubridate 1.9.4     v tidyr     1.3.1
## v purrr     1.1.0
## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x purrr::accumulate() masks foreach::accumulate()
## x dplyr::filter()     masks stats::filter()
## x dplyr::lag()        masks stats::lag()
## x dplyr::select()     masks MASS::select()
## x purrr::when()       masks foreach::when()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
dat_NFL <-load_player_stats(2023) %>%
  filter(position == "QB") %>%
  group_by(player_name) %>%
  summarise(passing_yards = sum(passing_yards),
            passing_tds = sum(passing_tds),
            interceptions = sum(passing_interceptions),
```

```
            rushing_yards = sum(rushing_yards),
            rushing_tds = sum(rushing_tds)
            )
```

#After many attempts, it still didn't work—here's the final version

```
finite_Gaussian_Mixture_E<-function(y,pi_k,mu_k,sigma_k){
  n_ <- dim(y)[1]
  K_ <- length(mu_k)
  r<-matrix(NA,nrow = n_,ncol = K_)
   for(i in 1:n_){
    denom <- sum(sapply(1:K_, function(j){
      pi_k[j] * dmvnorm(y[i, ], mean = mu_k[[j]], sigma = sigma_k)
    }))
    for(k in 1:K_){
      numer <- pi_k[k] * dmvnorm(y[i, ], mean = mu_k[[k]], sigma = sigma_k)
      r[i, k] <- numer / denom
    }
  }
  return(r)
}

Finite_Gaussian_Mixture_EM<-function(y,K){
  y <- as.matrix(y)
  n_ <- nrow(y)
  d <- ncol(y)
  pi_k <- rep(1/K, K)
  mu_k <- lapply(1:K, function(k) colMeans(y) + rnorm(d))
  Sigma_k <-diag(d)
  for(iter in 1:100){
    #E step:
    r<-finite_Gaussian_Mixture_E(y,pi_k,mu_k,sigma_k)
    #M step:
    Nk <- colSums(r)
    pi_k_new <- Nk / n_
    mu_k_new <- lapply(1:K, function(k) colSums(r[,k] * y) / Nk[k])
    Sigma_new <- matrix(0, nrow = d, ncol = d)
    for(k in 1:K){
      diff <- sweep(y, 2, mu_k_new[[k]], "-")
      Sigma_new <- Sigma_new + t(diff) %*% (diff * r[,k])
    }
   mu_diff <- max(sapply(1:K, function(k) max(abs(mu_k_new[[k]] - mu_k[[k]]))))
    if(max(abs(pi_k_new - pi_k), mu_diff, max(abs(Sigma_new - Sigma))) < 1e-6) break

#update
  pi_k<-pi_k_new
  mu_k<-mu_k_new
  sigma_k<-sigma_k_new
  }

return(list(pi=pi_k, mu=mu_k, sigma=sigma_k))

}
```

```
for(i in 1:6){
result<-Finite_Gaussian_Mixture_EM(dat_NFL[,-1],i)
result
}
```