# STATS 606 - Project code

Tim White

Due April 14th, 2023

## Contents

## Functions

### generate_data()

```
generate_data = function(n, p, K, sepVal) {
  # Inputs:
  #   n = number of rows
  #   p = number of columns
  #   K = number of mixture components
  #   sepVal = index of separation between components
  # Outputs:
  #   X = n by p data matrix
  #   cluster = n by 1 vector of true component assignments
  #   pi_star = K by 1 vector of true mixing proportions
  #   mu_star = K by p matrix of true component means

  # Generate clustered data
  cluster_data = genRandomClust(numClust = K, sepVal = sepVal, numNonNoisy = p,
                                numNoisy = 0, numOutlier = 0, numReplicate = 1,
                                covMethod = "eigen", ratioLambda = 1,
                                clustszind = 1, clustSizeEq = n/K + 25,
                                fileName = "cluster_data")

  points_and_clusters_orig = cbind(cluster_data$datList$cluster_data_1,
                                   cluster_data$memList$cluster_data_1)
  points_and_clusters = points_and_clusters_orig[sample(1:nrow(points_and_clusters_orig), n),]

  X = points_and_clusters[,1:p]

  cluster = points_and_clusters[,p+1]

  # Compute true pi and mu
  pi_star = sapply(1:K, function(k) {return(mean(cluster == k))})
  mu_star = t(sapply(1:K, function(k) {return(apply(X[cluster == k,], 2, mean))}))

  return(list(X = X, cluster = cluster, pi_star = pi_star, mu_star = mu_star))
}
```

**initialize_random()**

```
initialize_random = function(X, K) {
  # Inputs:
  #   X = n by p data matrix
  #   K = number of mixture components
  # Outputs:
  #   pi_init = initialized vector of mixing proportions
  #   mu_init = initialized matrix of component means
  #   sigma_init = initialized list of component covariance matrices

  n = nrow(X)
  p = ncol(X)

  pi_init = rep(1/K, K)
  mu_init = X[sample(1:n, K),]
  sigma_init = replicate(K, diag(p), simplify = FALSE)

  return(list(pi_init = pi_init, mu_init = mu_init, sigma_init = sigma_init))
}
```

## gmmEM()

```
gmmEM = function(data, K, tol, max_iter, pi_init, mu_init, sigma_init) {
  # Inputs:
  #   data = objected produced by generate_data()
  #   K = number of mixture components
  #   tol = convergence tolerance
  #   max_iter = maximum number of iterations allowed
  #   pi_init = initialized vector of mixing proportions
  #   mu_init = initialized matrix of component means
  #   sigma_init = initialized list of component covariance matrices
  # Outputs:
  #   num_iterations = number of iterations before convergence
  #   pi_hat = estimate of pi
  #   mu_hat = estimate of mu
  #   sigma_hat = estimate of sigma
  #   posterior_prob_hat = estimated posterior probability that each observation
  #                        belongs to each component
  #   cluster_pred = predicted component assignments
  #   loglikelihood = vector of loglikelihood values from each iteration
  #   abs_diff_norm_mu = absolute difference between Frobenius norms of
  #                      initialized mu and true mu

  # Identify dimensions of X
  n = nrow(data$X)
  p = ncol(data$X)

  # Compute absolute difference between Frobenius norms of initialized mu and true mu
  abs_diff_norm_mu = abs(sqrt(sum(diag(t(mu_init) %*% mu_init))) -
                         sqrt(sum(diag(t(data$mu_star) %*% data$mu_star))))

  # Set relative difference in loglikelihoods equal to tol for start of loop
  rel_diff_loglikelihood = tol

  # Set number of iterations equal to zero
  iter = 0

  # Initialize pi, mu, and sigma for iteration t
  pi_t = pi_init
  mu_t = mu_init
  sigma_t = sigma_init

  # Initialize vector to store loglikelihood
  loglikelihood = numeric()

  while (rel_diff_loglikelihood >= tol & iter < max_iter) {

    # Increment number of iterations
    iter = iter + 1



    ### E step
```

```r
# Compute posterior probabilities
compute_kth_density = function(k, pi, mu, sigma) {

  covmat = if (!matrixcalc::is.symmetric.matrix(sigma[[k]])) {
              if (matrixcalc::is.positive.definite(
                              as.matrix(symmpart(Matrix(sigma[[k]],
                                                        sparse = TRUE))))) {
                symmpart(Matrix(sigma[[k]], sparse = TRUE))
              } else {
                Matrix(diag(nrow(sigma[[k]])), sparse = TRUE)
              }
            } else {
              if (matrixcalc::is.positive.definite(sigma[[k]])) {
                Matrix(sigma[[k]], sparse = TRUE)
              } else {
                Matrix(diag(nrow(sigma[[k]])), sparse = TRUE)
              }
            }

  return(pi[k] * sparseMVN::dmvn.sparse(data$X,
                             mu[k,],
                             CH = Cholesky(covmat),
                             prec = FALSE, log = FALSE))
}

numerator_t = lapply(1:K, compute_kth_density,
                     pi = pi_t, mu = mu_t, sigma = sigma_t)
denominator_t = Reduce("+", numerator_t) + 0.000000001

compute_kth_posterior_probs = function(k, numerator, denominator) {
  return(numerator[[k]]/denominator)
}

posterior_probs_t1 = sapply(1:K, compute_kth_posterior_probs, numerator_t, denominator_t)



### M step
# Update pi
pi_t1 = colMeans(posterior_probs_t1)

# Update mu
mu_t1 = (t(posterior_probs_t1) %*% data$X)/colSums(posterior_probs_t1)

# Update sigma
compute_kth_sigma = function(k, X, mu, posterior_probs) {
  X_centered = X - rep(1,n) %*% t(mu[k,])
  num = t(X_centered) %*% (posterior_probs[,k] * (X_centered))
  denom = sum(posterior_probs[,k]) + 0.000000001
  return(num/denom)
}

sigma_t1 = lapply(1:K, compute_kth_sigma, data$X, mu_t1, posterior_probs_t1)
```

```
    ### Compute metrics for the current iteration
    # Compute loglikelihood
    loglikelihood = append(loglikelihood,
                           sum(log(Reduce("+",
                                          lapply(1:K, compute_kth_density,
                                                 pi = pi_t1, mu = mu_t1, sigma = sigma_t1)))))

    rel_diff_loglikelihood = (loglikelihood[iter] -
                                ifelse(iter == 1,
                                       sum(log(Reduce("+",
                                              lapply(1:K, compute_kth_density,
                                                     pi = pi_t, mu = mu_t,
                                                     sigma = sigma_t)) + 0.000000001)),
                                       loglikelihood[iter - 1]))/
                              (loglikelihood[iter] -
                                 ifelse(iter == 1,
                                        sum(log(Reduce("+",
                                               lapply(1:K, compute_kth_density,
                                                      pi = pi_t, mu = mu_t,
                                                      sigma = sigma_t)) + 0.000000001)),
                                        loglikelihood[1]) + 0.000000001)

    # Reset parameter estimates for next iteration
    pi_t = pi_t1
    mu_t = mu_t1
    sigma_t = sigma_t1
  }

  # Predict clusters based on posterior_probs_t1
  cluster_pred = apply(posterior_probs_t1, 1, function(row) {which.max(row)})

  return(list(num_iterations = iter, pi_hat = pi_t, mu_hat = mu_t, sigma_hat = sigma_t,
              posterior_prob_hat = posterior_probs_t1, cluster_pred = cluster_pred,
              loglikelihood = loglikelihood, abs_diff_norm_mu = abs_diff_norm_mu))
}
```

**randomEM()**

```
randomEM = function(data, K, tol, max_iter) {
  # Inputs:
  #   data = objected produced by generate_data()
  #   K = number of mixture components
  #   tol = convergence tolerance
  #   max_iter = maximum number of iterations allowed
  # Outputs:
  #   EM = objected produced by gmmEM()
  #   time = runtime of algorithm

  # Record start time
  start = Sys.time()

  # Initialize pi, mu, and sigma randomly
  init = initialize_random(data$X, K)

  # Run EM with randomly initialized pi, mu, and sigma
  EM = gmmEM(data, K, tol, max_iter,
             pi_init = init$pi_init,
             mu_init = init$mu_init,
             sigma_init = init$sigma_init)

  # Record end time
  end = Sys.time()

  # Compute runtime
  time = difftime(end, start, units = "secs")

  return(list(EM = EM, time = time))
}
```

**fit_randomEM()**

```
fit_randomEM = function(data, K_seq, tol, max_iter) {
  # Inputs:
  #   data = object produced by generate_data()
  #   K_seq = vector of values of number of mixture components
  #   tol = convergence tolerance
  #   max_iter = maximum number of iterations allowed
  # Outputs:
  #   EM = objected produced by randomEM()
  #   BIC = BIC of selected fit (maximizes 2*loglik - nparams*log(n))
  #   K = number of mixture components in selected fit
  #   time = runtime of selected fit

  # Run randomEM() for all values in K_seq
  fits = foreach(K = K_seq) %dopar% randomEM(data, K, tol, max_iter)

  # Compute BIC for each fit in fits
  BICs = lapply(seq(1, length(K_seq)),
                function(i) {
                  bic("VVV", fits[[i]]$EM$loglikelihood[fits[[i]]$EM$num_iterations],
                      n = nrow(data$X), d = ncol(data$X), G = K_seq[i])
                })

  # Choose the fit that maximizes BIC
  fit = fits[[which.max(BICs)]]

  return(list(EM = fit$EM, BIC = max(unlist(BICs)),
              K = K_seq[which.max(BICs)], time = as.numeric(fit$time)))
}
```

**emEM()**

```
emEM = function(data, K, tol, max_iter, num_start = 3) {
  # Inputs:
  #   data = objected produced by generate_data()
  #   K = number of mixture components
  #   tol = convergence tolerance
  #   max_iter = maximum number of iterations allowed
  #   num_start = number of candidate starting values for mu
  # Outputs:
  #   EM = objected produced by gmmEM()
  #   time = runtime of algorithm

  # Record start time
  start = Sys.time()

  # Create empty sequences for pi, mu, sigma, and loglikelihood
  pi_seq = list()
  mu_seq = list()
  sigma_seq = list()
  loglikelihood_seq = numeric(num_start)

  for (i in 1:num_start) {
    # Initialize randomly
    init = initialize_random(data$X, K)

    # Short run of EM with random initialization
    EM = gmmEM(data, K, tol = tol^(1/4), max_iter = nrow(data$X),
               pi_init = init$pi_init, mu_init = init$mu_init, sigma_init = init$sigma_init)

    # Store ith estimates of pi, mu, and sigma
    pi_seq[[i]] = EM$pi_hat
    mu_seq[[i]] = EM$mu_hat
    sigma_seq[[i]] = EM$sigma_hat

    # Store ith loglikelihood
    loglikelihood_seq[i] = EM$loglikelihood[EM$num_iterations]
  }

  # Long run of EM initialized with pi, mu, and sigma that maximize loglikelihood
  EM = gmmEM(data, K, tol, max_iter,
             pi_init = pi_seq[[which.max(loglikelihood_seq)]],
             mu_init = mu_seq[[which.max(loglikelihood_seq)]],
             sigma_init = sigma_seq[[which.max(loglikelihood_seq)]])

  # Record end time
  end = Sys.time()

  # Compute runtime
  time = difftime(end, start, units = "secs")

  return(list(EM = EM, time = time))
}
```

**fit_emEM()**

```
fit_emEM = function(data, K_seq, tol, max_iter, num_start = 3) {
  # Inputs:
  #   data = object produced by generate_data()
  #   K_seq = vector of values of number of mixture components
  #   tol = convergence tolerance
  #   max_iter = maximum number of iterations allowed
  #   num_start = number of candidate starting values for mu
  # Outputs:
  #   EM = objected produced by emEM()
  #   BIC = BIC of selected fit (maximizes 2*loglik - nparams*log(n))
  #   K = number of mixture components in selected fit
  #   time = runtime of selected fit

  # Run emEM() for all values in K_seq
  fits = foreach(K = K_seq) %dopar% emEM(data, K, tol, max_iter, num_start)

  # Compute BIC for each fit in fits
  BICs = lapply(seq(1, length(K_seq)),
                function(i) {
                  bic("VVV", fits[[i]]$EM$loglikelihood[fits[[i]]$EM$num_iterations],
                      n = nrow(data$X), d = ncol(data$X), G = K_seq[i])
                })

  # Choose the fit that maximizes BIC
  fit = fits[[which.max(BICs)]]

  return(list(EM = fit$EM, BIC = max(unlist(BICs)),
              K = K_seq[which.max(BICs)], time = as.numeric(fit$time)))
}
```

**svdEM()**

```
svdEM = function(data, K, tol, max_iter) {
  # Inputs:
  #   data = objected produced by generate_data()
  #   K = number of mixture components
  #   tol = convergence tolerance
  #   max_iter = maximum number of iterations allowed
  # Outputs:
  #   EM = objected produced by gmmEM()
  #   time = runtime of algorithm

  # Record start time
  start = Sys.time()

  # Identify number of columns of X
  p = ncol(data$X)

  # Initialize pi and mu with SVD
  init = starts.via.svd(data$X, nclass = K, method = "em")

  # Initialize sigma with SVD
  sigma_init = lapply(1:K, function(k) {
                 sigma = matrix(0, nrow = p, ncol = p)
                 sigma[upper.tri(sigma, diag = TRUE)] = init$LTSigma[k,]
                 sigma_init = sigma + t(sigma) - diag(diag(sigma))
                 return(sigma_init)
               })

  # Run EM with SVD-initialized parameters
  EM = gmmEM(data, K, tol, max_iter,
             pi_init = init$pi,
             mu_init = init$Mu,
             sigma_init = sigma_init)

  # Record end time
  end = Sys.time()

  # Compute runtime
  time = difftime(end, start, units = "secs")

  return(list(EM = EM, time = time))
}
```

**fit_svdEM()**

```
fit_svdEM = function(data, K_seq, tol, max_iter) {
  # Inputs:
  #   data = object produced by generate_data()
  #   K_seq = vector of values of number of mixture components
  #   tol = convergence tolerance
  #   max_iter = maximum number of iterations allowed
  # Outputs:
  #   EM = objected produced by svdEM()
  #   BIC = BIC of selected fit (maximizes 2*loglik - nparams*log(n))
  #   K = number of mixture components in selected fit
  #   time = runtime of selected fit

  # Run svdEM() for all values in K_seq
  fits = foreach(K = K_seq) %dopar% svdEM(data, K, tol, max_iter)

  # Compute BIC for each fit in fits
  BICs = lapply(seq(1, length(K_seq)),
                function(i) {
                  bic("VVV", fits[[i]]$EM$loglikelihood[fits[[i]]$EM$num_iterations],
                      n = nrow(data$X), d = ncol(data$X), G = K_seq[i])
                })

  # Choose the fit that maximizes BIC
  fit = fits[[which.max(BICs)]]

  return(list(EM = fit$EM, BIC = max(unlist(BICs)),
              K = K_seq[which.max(BICs)], time = as.numeric(fit$time)))
}
```

**kmEM()**

```r
kmEM = function(data, K, tol, max_iter) {
  # Inputs:
  #   data = objected produced by generate_data()
  #   K = number of mixture components
  #   tol = convergence tolerance
  #   max_iter = maximum number of iterations allowed
  # Outputs:
  #   EM = objected produced by gmmEM()
  #   time = runtime of algorithm

  # Record start time
  start = Sys.time()

  # Initialize pi, mu, and sigma using K-means
  init = kmeans(data$X, centers = K, nstart = nrow(data$X))
  pi_init = init$size/sum(init$size)
  mu_init = init$centers
  sigma_init = initialize_random(data$X, K)$sigma_init

  # Run EM with K-means initialized parameters
  EM = gmmEM(data, K, tol, max_iter,
             pi_init = pi_init,
             mu_init = mu_init,
             sigma_init = sigma_init)

  # Record end time
  end = Sys.time()

  # Compute runtime
  time = difftime(end, start, units = "secs")

  return(list(EM = EM, time = time))
}
```

**fit_kmEM()**

```
fit_kmEM = function(data, K_seq, tol, max_iter) {
  # Inputs:
  #   data = object produced by generate_data()
  #   K_seq = vector of values of number of mixture components
  #   tol = convergence tolerance
  #   max_iter = maximum number of iterations allowed
  # Outputs:
  #   EM = objected produced by kmEM()
  #   BIC = BIC of selected fit (maximizes 2*loglik - nparams*log(n))
  #   K = number of mixture components in selected fit
  #   time = runtime of selected fit

  # Run kmEM() for all values in K_seq
  fits = foreach(K = K_seq) %dopar% kmEM(data, K, tol, max_iter)

  # Compute BIC for each fit in fits
  BICs = lapply(seq(1, length(K_seq)),
                function(i) {
                  bic("VVV", fits[[i]]$EM$loglikelihood[fits[[i]]$EM$num_iterations],
                      n = nrow(data$X), d = ncol(data$X), G = K_seq[i])
                })

  # Choose the fit that maximizes BIC
  fit = fits[[which.max(BICs)]]

  return(list(EM = fit$EM, BIC = max(unlist(BICs)),
              K = K_seq[which.max(BICs)], time = as.numeric(fit$time)))
}
```

**plot_results()**

```
plot_results = function(data, K, EM) {
  # Inputs:
  #   data = objected produced by generate_data()
  #   K = number of mixture components
  #   EM = object produced by gmmEM(), randomEM(), emEM(), svdEM(), or kmEM()
  # Outputs:
  #   plot_clusters = function that plots X1 vs X2, colored by true cluster with
  #                       predicted clusters overlaid using ellipses (only for p = 2)
  #   plot_loglikelihood = plot of loglikelihood for each iteration

  palette(c("dodgerblue3", "indianred3", "gray70", "olivedrab3", "purple3", "hotpink1",
            "darkorange1", "goldenrod1", "navajowhite3", "cyan3", "green4", "rosybrown3"))

  # If p = 2, plot X1 vs X2 colored by cluster with predicted clusters overlaid
  if (ncol(EM$mu_hat) == 2) {
    plot_clusters = function() {
      plot(data$X, col = data$cluster, pch = 19, xlab = "X1", ylab = "X2")
      invisible(lapply(1:K, function(k) {ellipse(mu = as.vector(EM$mu_hat[k,]),
                                                 sigma = as.matrix(EM$sigma_hat[[k]]),
                                                 alpha = 0.05, lwd = 3)}))
    }
  }
  else if (ncol(EM$mu_hat) != 2) {
    plot_clusters = NULL
  }

  # Create data frame for plots
  plot_data = tibble(iterations = seq(from = 1, to = EM$num_iterations, by = 1),
                     loglikelihood = EM$loglikelihood)

  # Plot loglikelihood by number of iterations
  plot_loglikelihood = plot_data %>%
                       ggplot(aes(x = iterations, y = loglikelihood)) +
                         geom_line(col = "darkslategray", linewidth = 1) +
                         labs(x = "Number of iterations", y = "Loglikelihood") +
                         theme_classic()

  return(list(plot_clusters = plot_clusters,
              plot_loglikelihood = plot_loglikelihood))
}
```

**compare_methods()**

```
compare_methods = function(n, p, true_K, sepVal, K_seq, tol, max_iter) {
  # Inputs:
  #   n = number of rows in data matrix
  #   p = number of columns in data matrix
  #   true_K = true number of mixture components
  #   sepVal = index of separation between mixture components
  #   K_seq = vector of values of number of mixture components
  #   tol = convergence tolerance
  #   max_iter = maximum number of iterations allowed
  # Outputs:
  #   results = table summarizing performance of randomEM, emEM, svdEM, and
  #             kmEM; columns record name of method, adjusted Rand index,
  #             estimated number of mixture components, difference between
  #             Frobenius norm of estimated mu and true mu, number of iterations
  #             before convergence, and runtime (in seconds)

  # Generate data
  data = generate_data(n = n, p = p, K = true_K, sepVal = sepVal)

  # Apply the four methods
  random = fit_randomEM(data, K_seq, tol, max_iter)
  em = fit_emEM(data, K_seq, tol, max_iter, num_start = 3)
  svd = fit_svdEM(data, K_seq, tol, max_iter)
  km = fit_kmEM(data, K_seq, tol, max_iter)

  # Summarize performance of the four methods
  results = tibble(
    method = c("randomEM", "emEM", "svdEM", "kmEM"),
    BIC_rank = order(c(random$BIC, em$BIC, svd$BIC, km$BIC), decreasing = TRUE),
    rand = c(adjustedRandIndex(data$cluster, random$EM$cluster_pred),
             adjustedRandIndex(data$cluster, em$EM$cluster_pred),
             adjustedRandIndex(data$cluster, svd$EM$cluster_pred),
             adjustedRandIndex(data$cluster, km$EM$cluster_pred)),
    K = c(random$K, em$K, svd$K, km$K),
    diff_norm_mu = c(random$EM$abs_diff_norm_mu,
                     em$EM$abs_diff_norm_mu,
                     svd$EM$abs_diff_norm_mu,
                     km$EM$abs_diff_norm_mu),
    num_iter = c(random$EM$num_iterations,
                 em$EM$num_iterations,
                 svd$EM$num_iterations,
                 km$EM$num_iterations),
    time = c(random$time, em$time, svd$time, km$time)
  )

  return(results)
}
```

## Demonstration of initialization strategies

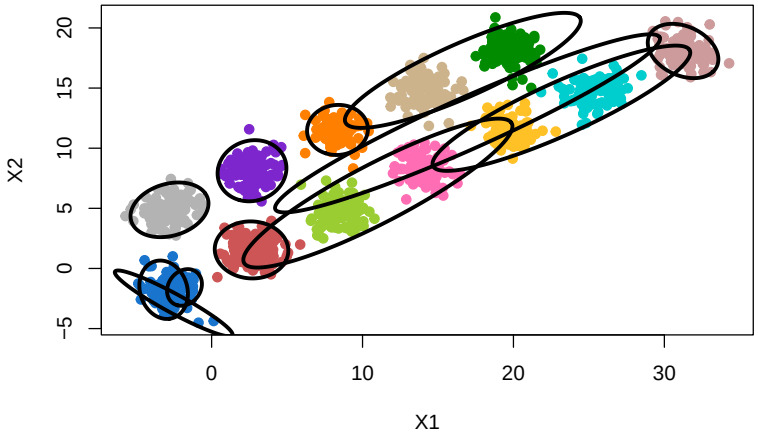
```
set.seed(606)

# Generate example data
example_data = generate_data(n = 1200, p = 2, K = 12, sepVal = 0.25)



# Run randomEM three times
example_randomEM = list(); example_plots = list()

for (i in 1:3) {
  example_randomEM[[i]] = randomEM(data = example_data, K = 12,
                                   tol = 1e-4, max_iter = 1000)
  example_plots[[i]] = plot_results(data = example_data, K = 12,
                                    EM = example_randomEM[[i]]$EM)
}



# Report maximum loglikelihood for each run
knitr::kable(t(sapply(1:3,
                  function(i) {
                    round(max(example_randomEM[[i]]$EM$loglikelihood), 3)})),
                  align = "c", col.names = c("Run 1", "Run 2", "Run 3"))
```

| Run 1 | Run 2 | Run 3 |
|---|---|---|
| -6598.999 | -6499.256 | -6511.704 |

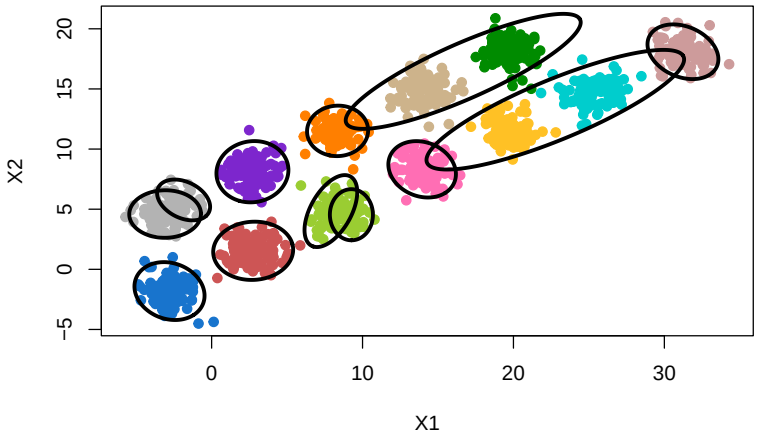
```
# Report adjusted Rand index for each run
knitr::kable(t(sapply(1:3,
                  function(i) {
                    round(adjustedRandIndex(
                             example_data$cluster,
                             example_randomEM[[i]]$EM$cluster_pred), 3)})),
              align = "c", col.names = c("Run 1", "Run 2", "Run 3"))
```

| Run 1 | Run 2 | Run 3 |
|---|---|---|
| 0.731 | 0.78 | 0.788 |

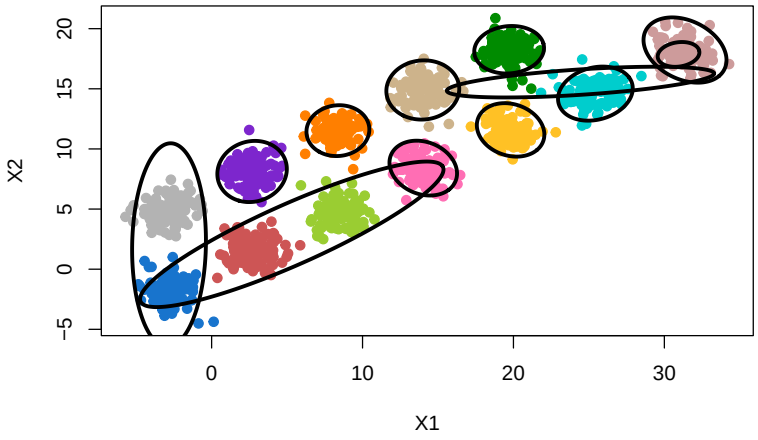
```
# Plot clusters for each run
example_plots[[1]]$plot_clusters()
```




```
example_plots[[2]]$plot_clusters()
```




```
example_plots[[3]]$plot_clusters()
```



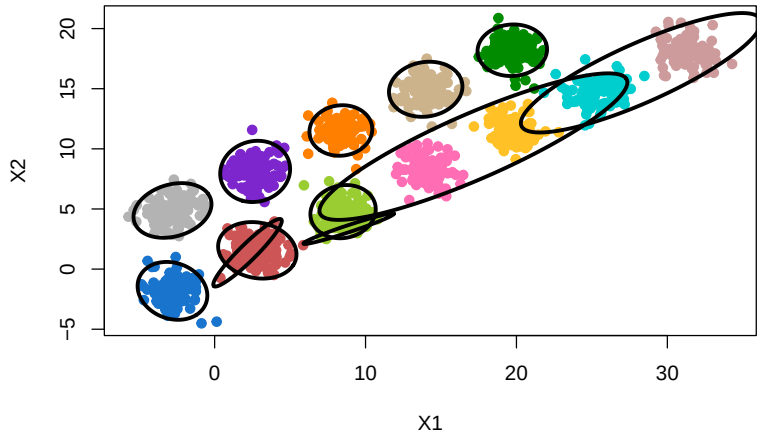
```
set.seed(606)

# Run emEM, svdEM, and kmEM
example_emEM = emEM(data = example_data, K = 12,
                    tol = 1e-4, max_iter = 1000)
example_svdEM = svdEM(data = example_data, K = 12,
                      tol = 1e-4, max_iter = 1000)
example_kmEM = kmEM(data = example_data, K = 12,
                    tol = 1e-4, max_iter = 1000)



# Report maximum loglikelihood for each method
knitr::kable(round(t(c(max(example_emEM$EM$loglikelihood),
               max(example_svdEM$EM$loglikelihood),
               max(example_kmEM$EM$loglikelihood))), 3),
             align = "c", col.names = c("emEM", "svdEM", "kmEM"))
```

| emEM | svdEM | kmEM |
|---|---|---|
| -6514.254 | -6307.834 | -6307.834 |

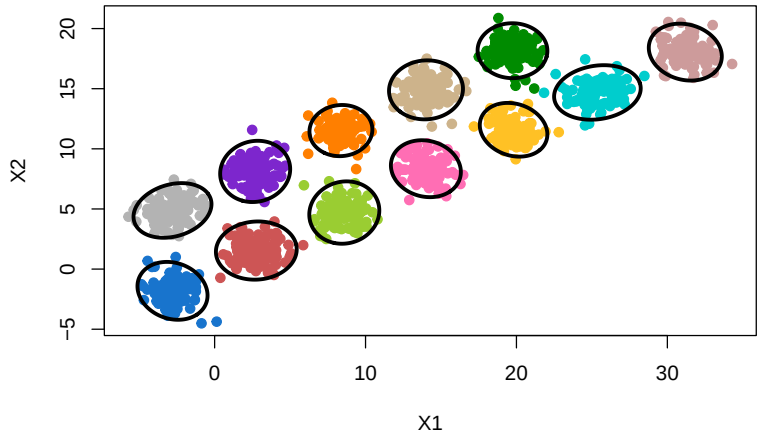
```
# Report adjusted Rand index for each method
knitr::kable(round(t(c(adjustedRandIndex(example_data$cluster,
                                 example_emEM$EM$cluster_pred),
               adjustedRandIndex(example_data$cluster,
                                 example_svdEM$EM$cluster_pred),
               adjustedRandIndex(example_data$cluster,
                                 example_kmEM$EM$cluster_pred))), 3),
             align = "c", col.names = c("emEM", "svdEM", "kmEM"))
```

| emEM | svdEM | kmEM |
|---|---|---|
| 0.8 | 0.998 | 0.998 |

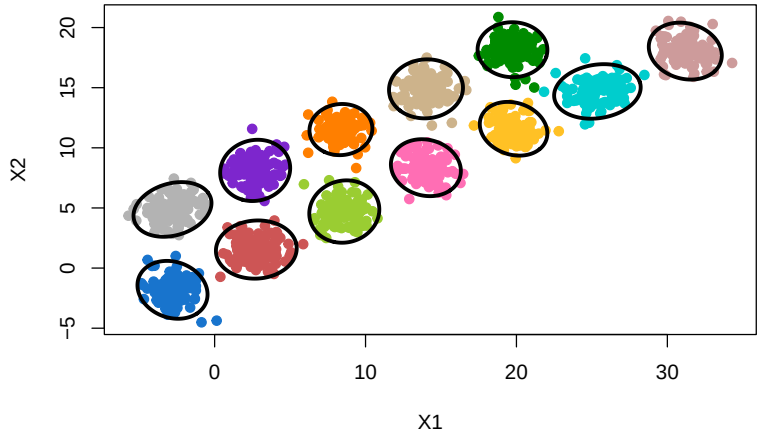
```
# Plot clusters for each method
plot_results(data = example_data,
             K = 12, EM = example_emEM$EM)$plot_clusters()
```




```
plot_results(data = example_data,
             K = 12, EM = example_svdEM$EM)$plot_clusters()
```




```
plot_results(data = example_data,
             K = 12, EM = example_kmEM$EM)$plot_clusters()
```



## Simulation studies for $K = 5$

```
# Set sample size and number of data sets to simulate
n = 500
num_datasets = 20

# Set true number of mixture components and values to try
true_K = 5
K_seq = c(4, 5, 6)
```

### $p = 2$, **sepVal** $= 0.01$

```
set.seed(606)

results_K5_1 = foreach(k = 1:num_datasets) %dorng%
                  compare_methods(n = n, p = 2,
                                  true_K = true_K,
                                  sepVal = 0.01,
                                  K_seq = K_seq,
                                  tol = 1e-4, max_iter = 1000) %>%
                  bind_rows() %>%
                  group_by(method) %>%
                  summarize(across(BIC_rank:time,
                                   ~ mean(.x, na.rm = TRUE)))
```

```
knitr::kable(results_K5_1 %>% mutate(BIC_rank = round(BIC_rank, 1),
                                     rand = round(rand, 3),
                                     K = format(round(K, 1), nsmall = 1),
                                     diff_norm_mu = round(diff_norm_mu, 3),
                                     num_iter = round(num_iter, 1),
                                     time = round(time, 1)),
             align = "c", col.names = c("Method", "BIC rank", "Adj. Rand index",
                                        "Detected K", "Diff norm mu",
                                        "# iterations", "Runtime"))
```

| Method | BIC rank | Adj. Rand index | Detected K | Diff norm mu | # iterations | Runtime |
|---|---|---|---|---|---|---|
| emEM | 3.7 | 0.785 | 4.8 | 0.709 | 64.5 | 2.9 |
| kmEM | 1.2 | 0.840 | 5.0 | 0.116 | 27.3 | 1.3 |
| randomEM | 3.2 | 0.766 | 4.8 | 1.206 | 57.4 | 2.5 |
| svdEM | 1.8 | 0.839 | 5.0 | 0.168 | 30.2 | 1.1 |

$p = 2$, **sepVal** $= 0.15$

```
set.seed(606)

results_K5_2 = foreach(k = 1:num_datasets) %dorng%
                  compare_methods(n = n, p = 2,
                                  true_K = true_K,
                                  sepVal = 0.15,
                                  K_seq = K_seq,
                                  tol = 1e-4, max_iter = 1000) %>%
                  bind_rows() %>%
                  group_by(method) %>%
                  summarize(across(BIC_rank:time,
                                   ~ mean(.x, na.rm = TRUE)))
```

```
knitr::kable(results_K5_2 %>% mutate(BIC_rank = round(BIC_rank, 1),
                                     rand = round(rand, 3),
                                     K = format(round(K, 1), nsmall = 1),
                                     diff_norm_mu = round(diff_norm_mu, 3),
                                     num_iter = round(num_iter, 1),
                                     time = round(time, 1)),
             align = "c", col.names = c("Method", "BIC rank", "Adj. Rand index",
                                        "Detected K", "Diff norm mu",
                                        "# iterations", "Runtime"))
```

| Method | BIC rank | Adj. Rand index | Detected K | Diff norm mu | # iterations | Runtime |
|---|---|---|---|---|---|---|
| emEM | 3.9 | 0.967 | 5.2 | 0.641 | 31.0 | 1.5 |
| kmEM | 1.1 | 0.972 | 5.0 | 0.020 | 9.5 | 0.5 |
| randomEM | 3.0 | 0.930 | 5.3 | 1.325 | 38.0 | 1.3 |
| svdEM | 2.1 | 0.972 | 5.0 | 0.024 | 9.8 | 0.4 |

$p = 2$, **sepVal** $= 0.3$

```
set.seed(606)

results_K5_3 = foreach(k = 1:num_datasets) %dorng%
                  compare_methods(n = n, p = 2,
                                  true_K = true_K,
                                  sepVal = 0.3,
                                  K_seq = K_seq,
                                  tol = 1e-4, max_iter = 1000) %>%
                  bind_rows() %>%
                  group_by(method) %>%
                  summarize(across(BIC_rank:time,
                                   ~ mean(.x, na.rm = TRUE)))
```

```
knitr::kable(results_K5_3 %>% mutate(BIC_rank = round(BIC_rank, 1),
                                     rand = round(rand, 3),
                                     K = format(round(K, 1), nsmall = 1),
                                     diff_norm_mu = round(diff_norm_mu, 3),
                                     num_iter = round(num_iter, 1),
                                     time = round(time, 1)),
             align = "c", col.names = c("Method", "BIC rank", "Adj. Rand index",
                                        "Detected K", "Diff norm mu",
                                        "# iterations", "Runtime"))
```

| Method | BIC rank | Adj. Rand index | Detected K | Diff norm mu | # iterations | Runtime |
|---|---|---|---|---|---|---|
| emEM | 3.2 | 0.986 | 5.4 | 1.196 | 37.4 | 1.5 |
| kmEM | 1.5 | 0.999 | 5.0 | 0.001 | 5.0 | 0.3 |
| randomEM | 2.9 | 0.946 | 5.4 | 2.282 | 25.4 | 0.9 |
| svdEM | 2.4 | 0.999 | 5.0 | 0.002 | 4.8 | 0.2 |

$p = 4$, **sepVal** $= 0.01$

```
set.seed(606)

results_K5_4 = foreach(k = 1:num_datasets) %dorng%
                  compare_methods(n = n, p = 4,
                                  true_K = true_K,
                                  sepVal = 0.01,
                                  K_seq = K_seq,
                                  tol = 1e-4, max_iter = 1000) %>%
                  bind_rows() %>%
                  group_by(method) %>%
                  summarize(across(BIC_rank:time,
                                   ~ mean(.x, na.rm = TRUE)))
```

```
knitr::kable(results_K5_4 %>% mutate(BIC_rank = round(BIC_rank, 1),
                                     rand = round(rand, 3),
                                     K = format(round(K, 1), nsmall = 1),
                                     diff_norm_mu = round(diff_norm_mu, 3),
                                     num_iter = round(num_iter, 1),
                                     time = round(time, 1)),
             align = "c", col.names = c("Method", "BIC rank", "Adj. Rand index",
                                        "Detected K", "Diff norm mu",
                                        "# iterations", "Runtime"))
```

| Method | BIC rank | Adj. Rand index | Detected K | Diff norm mu | # iterations | Runtime |
|---|---|---|---|---|---|---|
| emEM | 2.7 | 0.598 | 4.0 | 1.124 | 39.5 | 2.5 |
| kmEM | 1.6 | 0.613 | 4.0 | 0.962 | 22.5 | 1.2 |
| randomEM | 3.0 | 0.562 | 4.0 | 1.249 | 42.6 | 2.1 |
| svdEM | 2.8 | 0.607 | 4.0 | 0.959 | 24.9 | 0.9 |

$p = 4$, **sepVal** $= 0.15$

```
set.seed(606)

results_K5_5 = foreach(k = 1:num_datasets) %dorng%
                  compare_methods(n = n, p = 4,
                                  true_K = true_K,
                                  sepVal = 0.15,
                                  K_seq = K_seq,
                                  tol = 1e-4, max_iter = 1000) %>%
                  bind_rows() %>%
                  group_by(method) %>%
                  summarize(across(BIC_rank:time,
                                   ~ mean(.x, na.rm = TRUE)))
```

```
knitr::kable(results_K5_5 %>% mutate(BIC_rank = round(BIC_rank, 1),
                                     rand = round(rand, 3),
                                     K = format(round(K, 1), nsmall = 1),
                                     diff_norm_mu = round(diff_norm_mu, 3),
                                     num_iter = round(num_iter, 1),
                                     time = round(time, 1)),
             align = "c", col.names = c("Method", "BIC rank", "Adj. Rand index",
                                        "Detected K", "Diff norm mu",
                                        "# iterations", "Runtime"))
```

| Method | BIC rank | Adj. Rand index | Detected K | Diff norm mu | # iterations | Runtime |
|---|---|---|---|---|---|---|
| emEM | 3.7 | 0.925 | 5.0 | 0.612 | 24.0 | 1.9 |
| kmEM | 1.1 | 0.963 | 5.0 | 0.011 | 12.3 | 0.8 |
| randomEM | 3.0 | 0.895 | 5.0 | 0.873 | 35.6 | 1.7 |
| svdEM | 2.1 | 0.963 | 5.0 | 0.013 | 12.3 | 0.6 |

$p = 4$, **sepVal** $= 0.3$

```
set.seed(606)

results_K5_6 = foreach(k = 1:num_datasets) %dorng%
                  compare_methods(n = n, p = 4,
                                  true_K = true_K,
                                  sepVal = 0.3,
                                  K_seq = K_seq,
                                  tol = 1e-4, max_iter = 1000) %>%
                  bind_rows() %>%
                  group_by(method) %>%
                  summarize(across(BIC_rank:time,
                                   ~ mean(.x, na.rm = TRUE)))
```

```
knitr::kable(results_K5_6 %>% mutate(BIC_rank = round(BIC_rank, 1),
                                     rand = round(rand, 3),
                                     K = format(round(K, 1), nsmall = 1),
                                     diff_norm_mu = round(diff_norm_mu, 3),
                                     num_iter = round(num_iter, 1),
                                     time = round(time, 1)),
             align = "c", col.names = c("Method", "BIC rank", "Adj. Rand index",
                                        "Detected K", "Diff norm mu",
                                        "# iterations", "Runtime"))
```

| Method | BIC rank | Adj. Rand index | Detected K | Diff norm mu | # iterations | Runtime |
|---|---|---|---|---|---|---|
| emEM | 3.4 | 0.984 | 5.3 | 0.598 | 25.0 | 1.7 |
| kmEM | 1.4 | 0.998 | 5.0 | 0.001 | 6.2 | 0.5 |
| randomEM | 3.0 | 0.941 | 5.0 | 1.594 | 16.8 | 0.9 |
| svdEM | 2.2 | 0.998 | 5.0 | 0.001 | 6.2 | 0.4 |

## Simulation studies for $K = 10$

```
# Set sample size and number of data sets to simulate
n = 1000
num_datasets = 20

# Set true number of mixture components and values to try
true_K = 10
K_seq = c(9, 10, 11)
```

### $p = 2$, **sepVal** $= 0.01$

```
set.seed(606)

results_K10_1 = foreach(k = 1:num_datasets) %dorng%
                  compare_methods(n = n, p = 2,
                                  true_K = true_K,
                                  sepVal = 0.01,
                                  K_seq = K_seq,
                                  tol = 1e-4, max_iter = 1000) %>%
                  bind_rows() %>%
                  group_by(method) %>%
                  summarize(across(BIC_rank:time,
                                   ~ mean(.x, na.rm = TRUE)))
```

```
knitr::kable(results_K10_1 %>% mutate(BIC_rank = round(BIC_rank, 1),
                                      rand = round(rand, 3),
                                      K = format(round(K, 1), nsmall = 1),
                                      diff_norm_mu = round(diff_norm_mu, 3),
                                      num_iter = round(num_iter, 1),
                                      time = round(time, 1)),
             align = "c", col.names = c("Method", "BIC rank", "Adj. Rand index",
                                        "Detected K", "Diff norm mu",
                                        "# iterations", "Runtime"))
```

| Method | BIC rank | Adj. Rand index | Detected K | Diff norm mu | # iterations | Runtime |
|---|---|---|---|---|---|---|
| emEM | 2.5 | 0.725 | 9.7 | 1.836 | 118.0 | 10.0 |
| kmEM | 1.5 | 0.742 | 9.2 | 1.054 | 64.3 | 5.8 |
| randomEM | 3.9 | 0.705 | 9.7 | 3.046 | 117.0 | 9.3 |
| svdEM | 2.1 | 0.771 | 9.6 | 0.710 | 106.6 | 7.4 |

$p = 2$, **sepVal** $= 0.15$

```
set.seed(606)

results_K10_2 = foreach(k = 1:num_datasets) %dorng%
                  compare_methods(n = n, p = 2,
                                  true_K = true_K,
                                  sepVal = 0.15,
                                  K_seq = K_seq,
                                  tol = 1e-4, max_iter = 1000) %>%
                  bind_rows() %>%
                  group_by(method) %>%
                  summarize(across(BIC_rank:time,
                                   ~ mean(.x, na.rm = TRUE)))
```

```
knitr::kable(results_K10_2 %>% mutate(BIC_rank = round(BIC_rank, 1),
                                      rand = round(rand, 3),
                                      K = format(round(K, 1), nsmall = 1),
                                      diff_norm_mu = round(diff_norm_mu, 3),
                                      num_iter = round(num_iter, 1),
                                      time = round(time, 1)),
             align = "c", col.names = c("Method", "BIC rank", "Adj. Rand index",
                                        "Detected K", "Diff norm mu",
                                        "# iterations", "Runtime"))
```

| Method | BIC rank | Adj. Rand index | Detected K | Diff norm mu | # iterations | Runtime |
|---|---|---|---|---|---|---|
| emEM | 3.2 | 0.945 | 10.3 | 2.057 | 48.0 | 5.7 |
| kmEM | 1.2 | 0.970 | 10.0 | 0.044 | 10.9 | 1.8 |
| randomEM | 3.6 | 0.882 | 10.2 | 3.733 | 58.5 | 4.7 |
| svdEM | 1.9 | 0.956 | 9.9 | 0.279 | 23.4 | 1.9 |

$p = 2$, **sepVal** $= 0.3$

```
set.seed(606)

results_K10_3 = foreach(k = 1:num_datasets) %dorng%
                  compare_methods(n = n, p = 2,
                                  true_K = true_K,
                                  sepVal = 0.3,
                                  K_seq = K_seq,
                                  tol = 1e-4, max_iter = 1000) %>%
                  bind_rows() %>%
                  group_by(method) %>%
                  summarize(across(BIC_rank:time,
                                   ~ mean(.x, na.rm = TRUE)))
```

```
knitr::kable(results_K10_3 %>% mutate(BIC_rank = round(BIC_rank, 1),
                                      rand = round(rand, 3),
                                      K = format(round(K, 1), nsmall = 1),
                                      diff_norm_mu = round(diff_norm_mu, 3),
                                      num_iter = round(num_iter, 1),
                                      time = round(time, 1)),
             align = "c", col.names = c("Method", "BIC rank", "Adj. Rand index",
                                        "Detected K", "Diff norm mu",
                                        "# iterations", "Runtime"))
```

| Method | BIC rank | Adj. Rand index | Detected K | Diff norm mu | # iterations | Runtime |
|---|---|---|---|---|---|---|
| emEM | 3.5 | 0.928 | 10.4 | 3.404 | 23.4 | 2.8 |
| kmEM | 1.2 | 0.998 | 10.0 | 0.006 | 6.3 | 1.2 |
| randomEM | 3.5 | 0.890 | 10.2 | 4.518 | 21.6 | 1.7 |
| svdEM | 1.9 | 0.998 | 10.0 | 0.028 | 6.0 | 0.6 |

$p = 4$, **sepVal** $= 0.01$

```
set.seed(606)

results_K10_4 = foreach(k = 1:num_datasets) %dorng%
                  compare_methods(n = n, p = 4,
                                  true_K = true_K,
                                  sepVal = 0.01,
                                  K_seq = K_seq,
                                  tol = 1e-4, max_iter = 1000) %>%
                  bind_rows() %>%
                  group_by(method) %>%
                  summarize(across(BIC_rank:time,
                                   ~ mean(.x, na.rm = TRUE)))
```

```
knitr::kable(results_K10_4 %>% mutate(BIC_rank = round(BIC_rank, 1),
                                      rand = round(rand, 3),
                                      K = format(round(K, 1), nsmall = 1),
                                      diff_norm_mu = round(diff_norm_mu, 3),
                                      num_iter = round(num_iter, 1),
                                      time = round(time, 1)),
             align = "c", col.names = c("Method", "BIC rank", "Adj. Rand index",
                                        "Detected K", "Diff norm mu",
                                        "# iterations", "Runtime"))
```

| Method | BIC rank | Adj. Rand index | Detected K | Diff norm mu | # iterations | Runtime |
|---|---|---|---|---|---|---|
| emEM | 2.6 | 0.599 | 9.1 | 1.386 | 104.0 | 12.4 |
| kmEM | 1.4 | 0.672 | 9.0 | 0.800 | 64.6 | 8.5 |
| randomEM | 3.9 | 0.538 | 9.0 | 2.015 | 87.8 | 9.0 |
| svdEM | 2.1 | 0.644 | 9.0 | 0.869 | 81.2 | 7.7 |

$p = 4$, **sepVal** $= 0.15$

```
set.seed(606)

results_K10_5 = foreach(k = 1:num_datasets) %dorng%
                  compare_methods(n = n, p = 4,
                                  true_K = true_K,
                                  sepVal = 0.15,
                                  K_seq = K_seq,
                                  tol = 1e-4, max_iter = 1000) %>%
                  bind_rows() %>%
                  group_by(method) %>%
                  summarize(across(BIC_rank:time,
                                   ~ mean(.x, na.rm = TRUE)))
```

```
knitr::kable(results_K10_5 %>% mutate(BIC_rank = round(BIC_rank, 1),
                                      rand = round(rand, 3),
                                      K = format(round(K, 1), nsmall = 1),
                                      diff_norm_mu = round(diff_norm_mu, 3),
                                      num_iter = round(num_iter, 1),
                                      time = round(time, 1)),
             align = "c", col.names = c("Method", "BIC rank", "Adj. Rand index",
                                        "Detected K", "Diff norm mu",
                                        "# iterations", "Runtime"))
```

| Method | BIC rank | Adj. Rand index | Detected K | Diff norm mu | # iterations | Runtime |
|---|---|---|---|---|---|---|
| emEM | 3.5 | 0.878 | 9.9 | 1.351 | 37.1 | 5.4 |
| kmEM | 1.4 | 0.941 | 9.9 | 0.136 | 16.0 | 3.3 |
| randomEM | 3.5 | 0.843 | 9.7 | 2.385 | 58.0 | 5.7 |
| svdEM | 1.6 | 0.945 | 9.9 | 0.101 | 13.1 | 1.4 |

$p = 4$, **sepVal** $= 0.3$

```
set.seed(606)

results_K10_6 = foreach(k = 1:num_datasets) %dorng%
                  compare_methods(n = n, p = 4,
                                  true_K = true_K,
                                  sepVal = 0.3,
                                  K_seq = K_seq,
                                  tol = 1e-4, max_iter = 1000) %>%
                  bind_rows() %>%
                  group_by(method) %>%
                  summarize(across(BIC_rank:time,
                                   ~ mean(.x, na.rm = TRUE)))
```

```
knitr::kable(results_K10_6 %>% mutate(BIC_rank = round(BIC_rank, 1),
                                      rand = round(rand, 3),
                                      K = format(round(K, 1), nsmall = 1),
                                      diff_norm_mu = round(diff_norm_mu, 3),
                                      num_iter = round(num_iter, 1),
                                      time = round(time, 1)),
             align = "c", col.names = c("Method", "BIC rank", "Adj. Rand index",
                                        "Detected K", "Diff norm mu",
                                        "# iterations", "Runtime"))
```

| Method | BIC rank | Adj. Rand index | Detected K | Diff norm mu | # iterations | Runtime |
|---|---|---|---|---|---|---|
| emEM | 3.6 | 0.894 | 10.2 | 2.346 | 34.1 | 3.6 |
| kmEM | 1.4 | 0.999 | 10.0 | 0.002 | 6.8 | 1.7 |
| randomEM | 3.4 | 0.881 | 10.2 | 3.671 | 21.8 | 1.9 |
| svdEM | 1.6 | 0.999 | 10.0 | 0.015 | 6.7 | 0.7 |