APA-L3

September 6, 2018

1 APA Laboratori 3 - Clustering

```
In [1]: # uncomment for installing missing packages
        # install.packages("cclust")
        # install.packages('ggplot2')
        # install.packages('RColorBrewer')
        # install.packages("Rmixmod")
```

```
In [2]: options(repr.plot.width=6, repr.plot.height=6)
```

```
In [3]: set.seed(222)
```

1.1 Example 1. Clustering easy artificial 2D data with k-means

First we create a simple data set:
the cclust library contains some clustering functions, including k-means

```
In [4]: library (cclust)
```

```
In [5]: N1 <- 30
        N2 <- 40
        N3 <- 50
```

create cluster 1

```
In [6]: x1 <- rnorm (N1,1,0.5)
        y1 <- rnorm (N1,1,0.5)
```

create cluster 2

```
In [7]: x2 <- rnorm (N2,2,0.5)
        y2 <- rnorm (N2,6,0.7)
```

create cluster 3

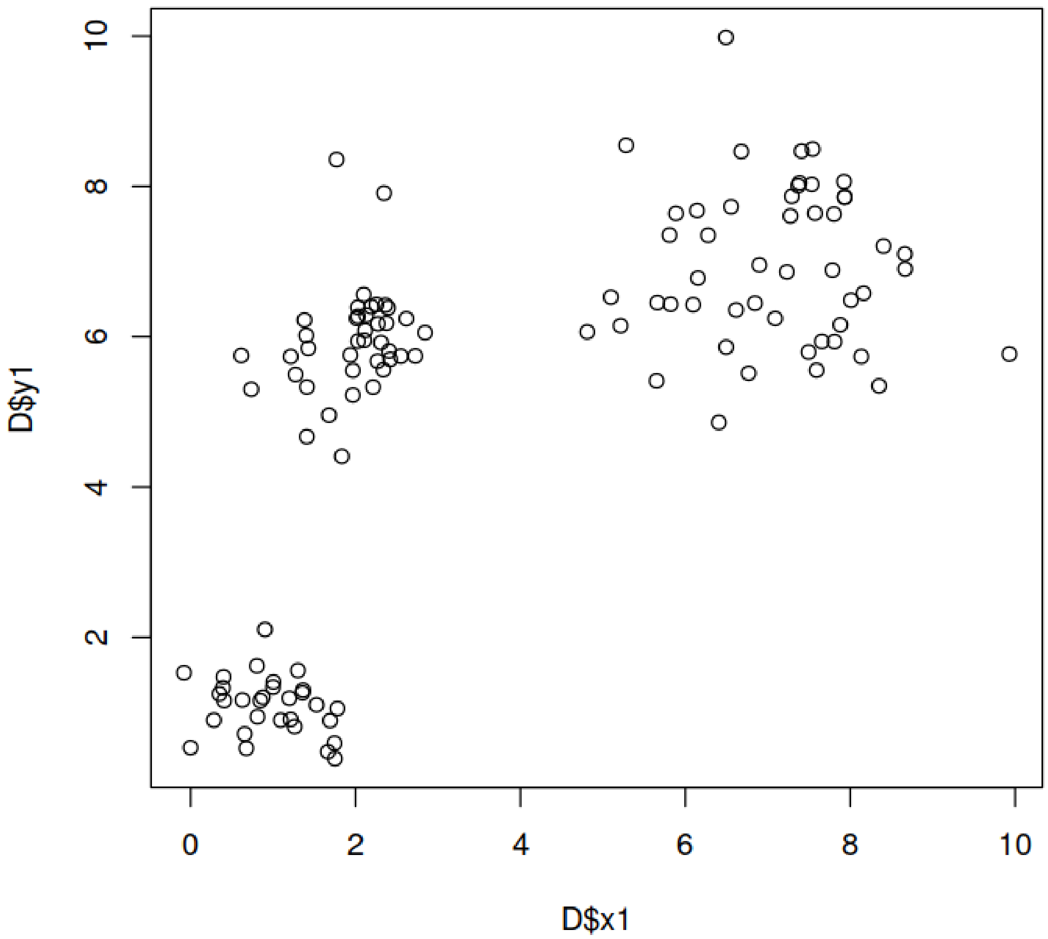
```
In [8]: x3 <- rnorm (N3,7,1)
        y3 <- rnorm (N3,7,1)
```

create the data

```
In [9]: x <- rbind (cbind(x1,y1), cbind(x2,y2), cbind(x3,y3))
        c <- c(rep("1", N1), rep("2", N2), rep("3", N3))
        D <- data.frame (x,color=c)
```

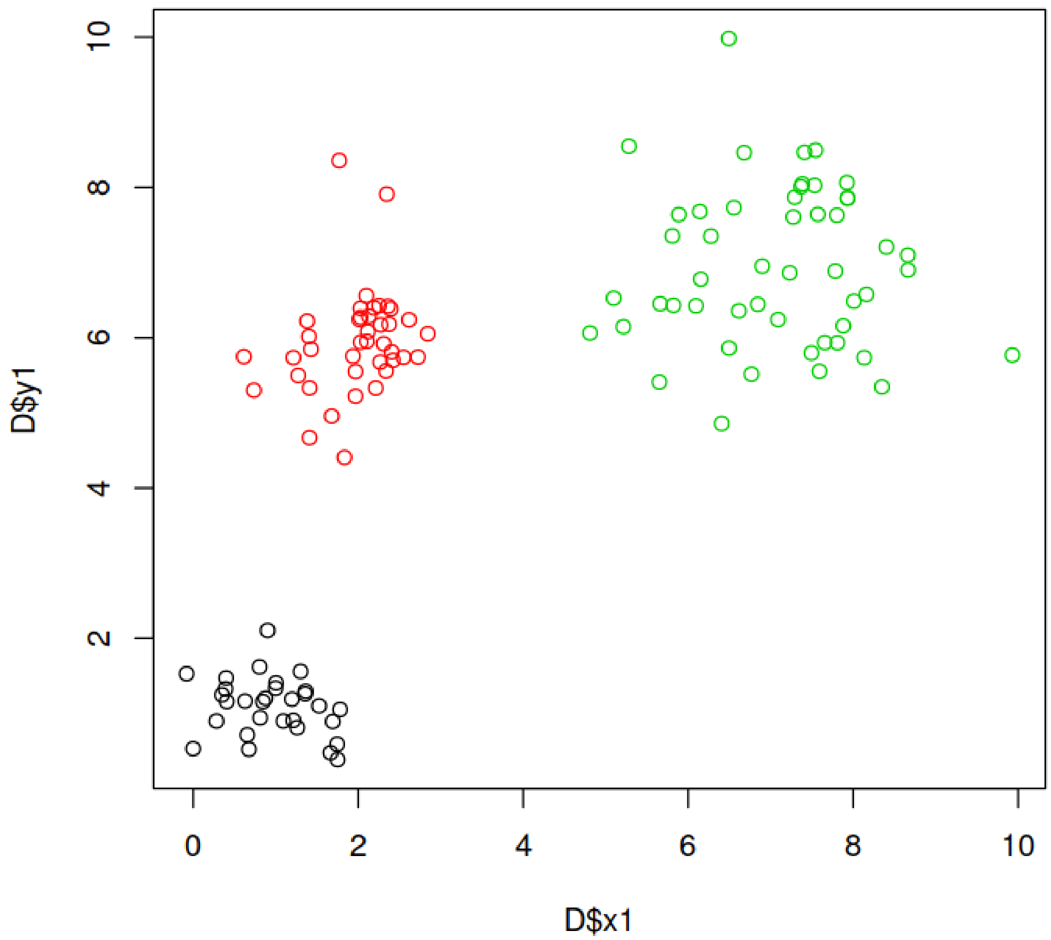
this is your data

```
In [10]: options(repr.plot.width=6, repr.plot.height=6)
         plot(D$x1,D$y1)
```



and these are the true clusters

```
In [11]: plot(D$x1,D$y1,col=as.vector(D$color))
```



so we have 3 very clean clusters ...
Let's execute k-means

```
In [12]: K <- 3 # yeah, this is tricky, why 3?
```

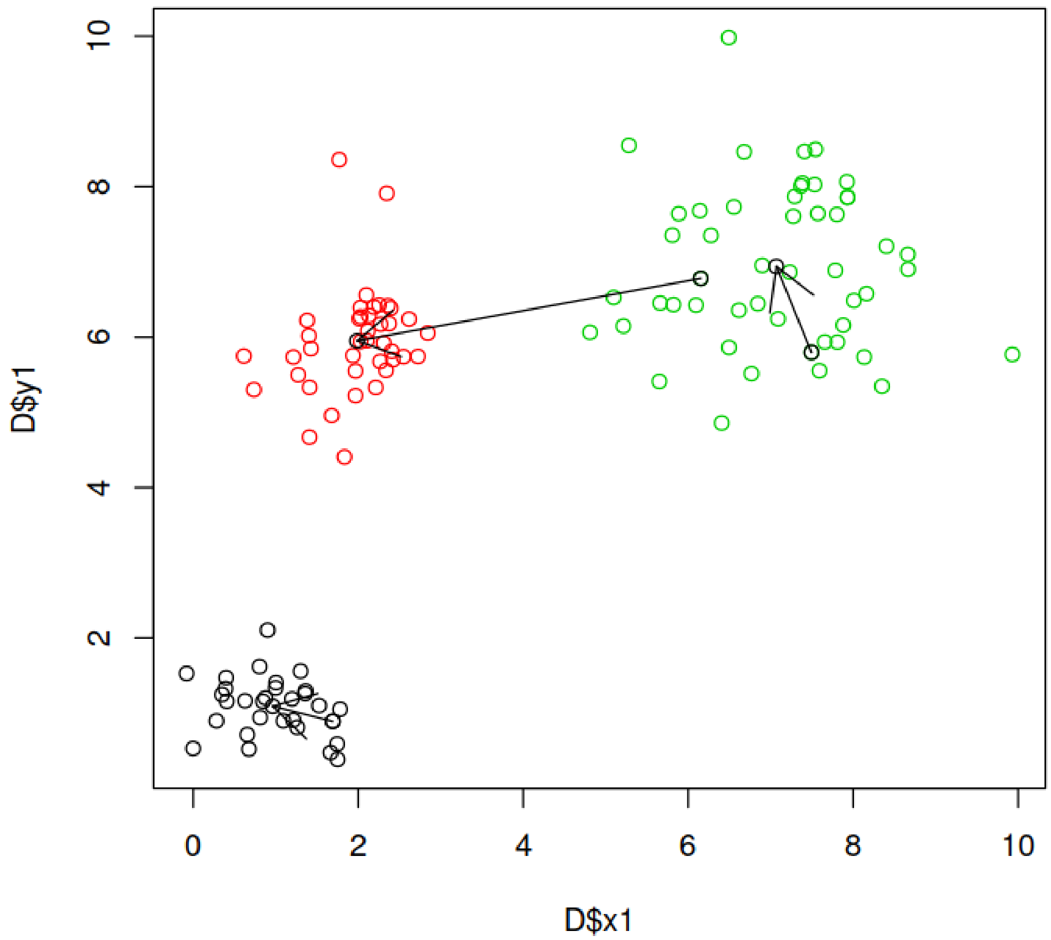
execute k-means with a maximum of 100 iterations

```
In [13]: kmeans.3 <- cclust (x,K,iter.max=100,method="kmeans",dist="euclidean")
```

plot initial and final prototypes (cluster centers) and draw arrows to see the process

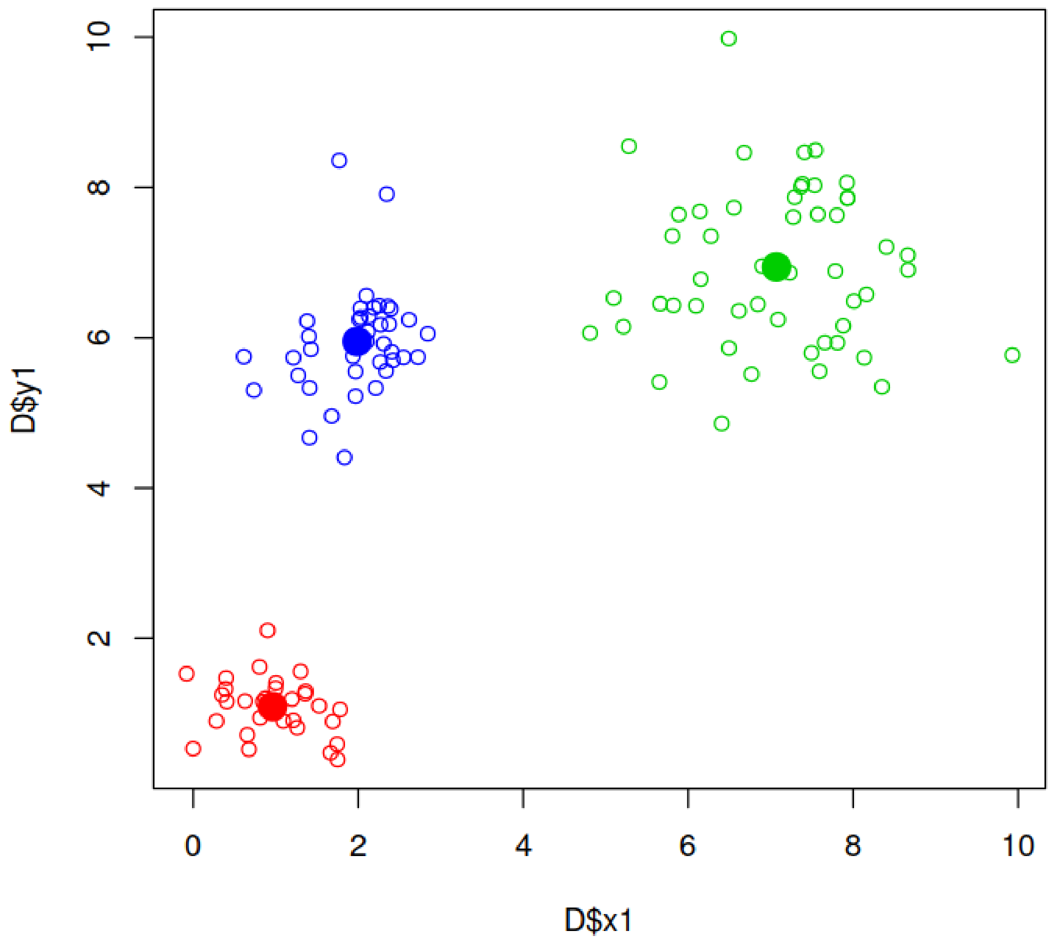
```
In [14]: plot(D$x1,D$y1,col=as.vector(D$color))
         points(kmeans.3$initcenters)
         points(kmeans.3$centers)
```

```
arrows (kmeans.3$initcenters[,1], kmeans.3$initcenters[,2],
        kmeans.3$centers[,1], kmeans.3$centers[,2])
```



plot and paint the clusters (according to the computed assignments) and plot the cluster centers

In [15]:
```
plot(D$x1,D$y1,col=(kmeans.3$cluster+1))

points(kmeans.3$centers,col=seq(1:kmeans.3$ncenters)+1,cex=2,pch=19)
```



clustering quality as measured by the Calinski-Harabasz index (recommended)

This index measures the dispersion of the data points within the clusters (SSW) and between the clusters (SSB)

A good clustering has small SSW (compact clusters) and large SSB (separated cluster centers)

There is also a correction for the number of clusters

The CH index is then:

$CH = (SSB/(K-1))/(SSW/(N-K))$

where N is the number of data points and K is the number of clusters

```
In [16]: (CH.3 <- clustIndex(kmeans.3,x, index="calinski"))
```

calinski: 611.073949634291

now let's not be tricky

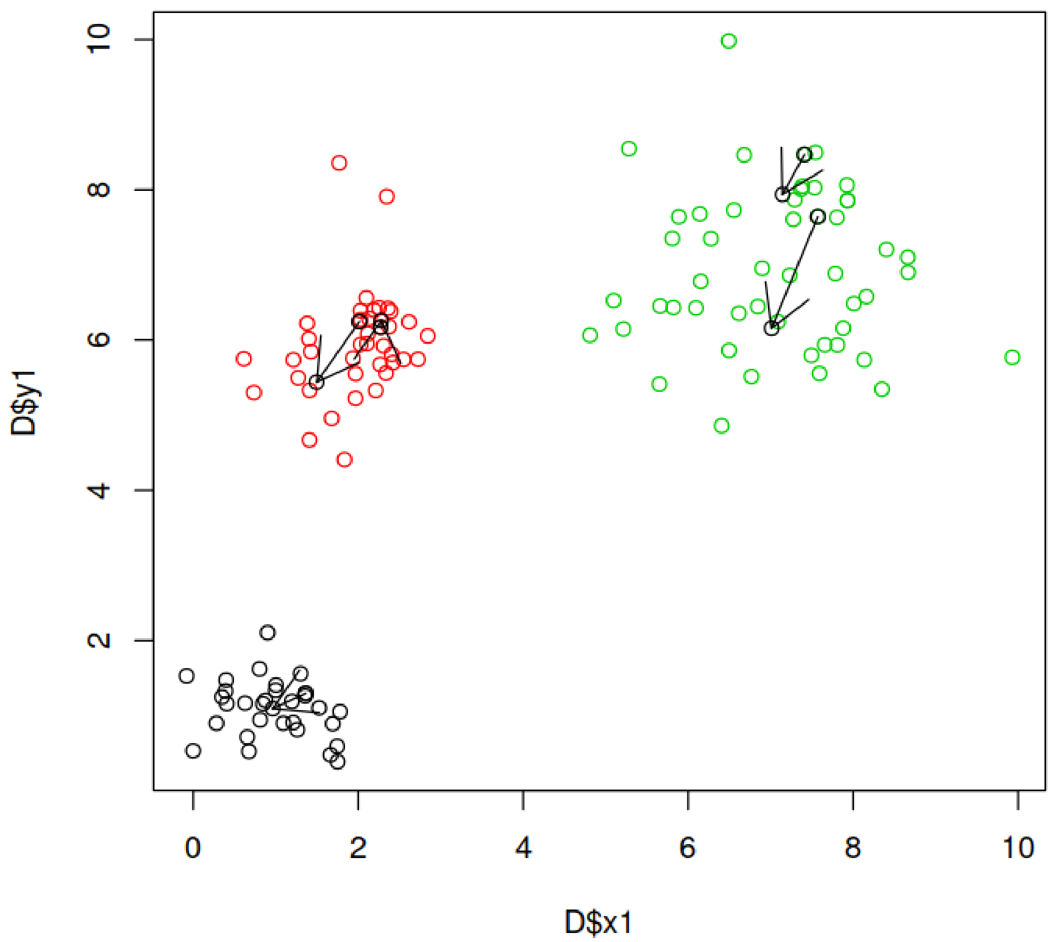
```
In [17]: K <- 5 # guess what is going to happen?
```

execute k-means with a maximum of 100 iterations

```
In [18]: kmeans.5 <- cclust (x,K,iter.max=100,method="kmeans",dist="euclidean")
```

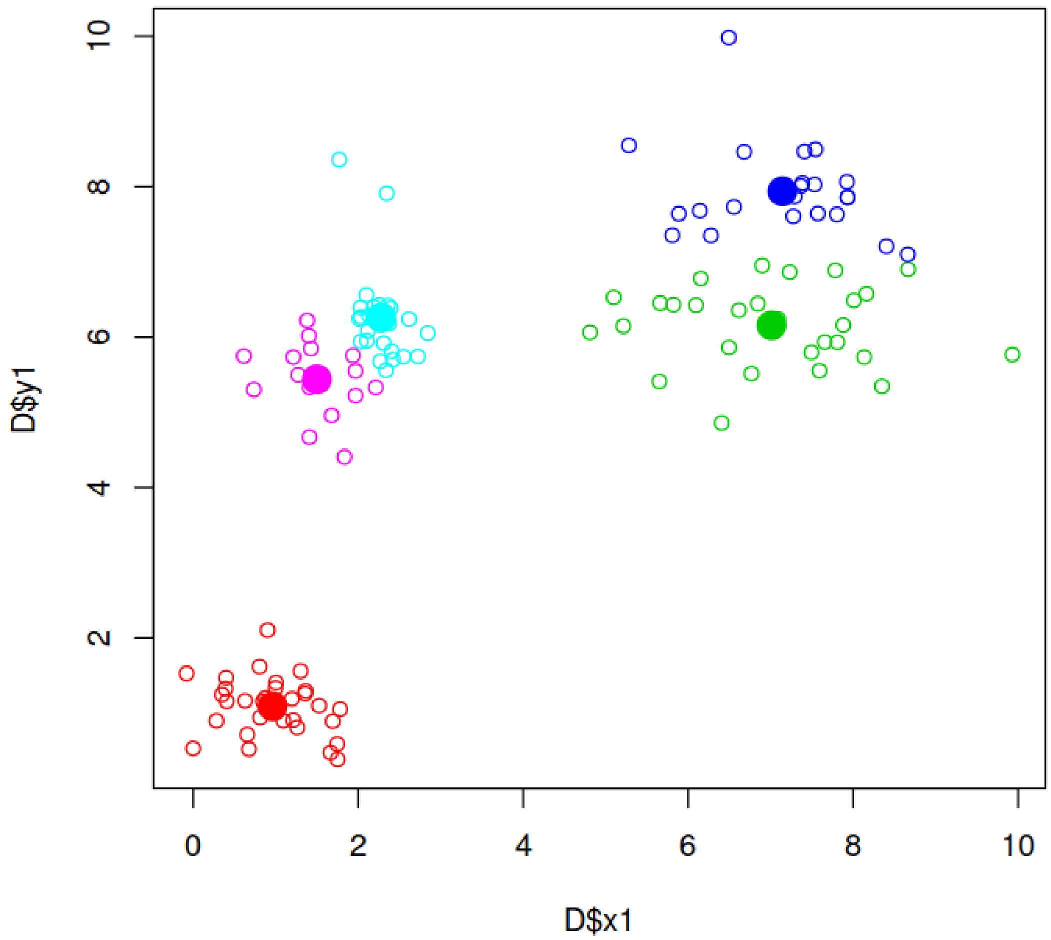
this is your data again, plot initial and final prototypes (centers) and draw arrows to see the process

```
In [19]: plot(D$x1,D$y1,col=as.vector(D$color))

         points(kmeans.5$initcenters)
         points(kmeans.5$centers)

         arrows (kmeans.5$initcenters[,1], kmeans.5$initcenters[,2],
                 kmeans.5$centers[,1], kmeans.5$centers[,2])
```



plot and paint the clusters (according to the computed assignments), plot the cluster centers

```
In [20]: plot(D$x1,D$y1,col=(kmeans.5$cluster+1))

         points(kmeans.5$centers,col=seq(1:kmeans.5$ncenters)+1,cex=2,pch=19)
```



clustering quality as measured by the Calinski-Harabasz index

```
In [21]: (CH.5 <- clustIndex(kmeans.5,x, index="calinski"))
```

calinski: 467.706370270364
notice CH.3 > CH.5, so K=3 is better according to C-H

1.2 Example 2. Clustering not-so-easy artificial 2D data with k-means and E-M

the MASS library contains the multivariate gaussian

In [22]:
```
library(MASS)
```

the ggplot2 library contains functions for making nice plots

In [23]:
```
library(ggplot2)
set.seed(333)
```

First we need some auxiliary functions

In [24]:
```
#GENERATE DATA FROM A MIXTURE OF 2D GAUSSIANS
generate.data <- function(N, K, prior.mean, prior.var)
{
  p <- length(prior.mean)

  # generate random mixture centres from the prior
  mu_k <- mvrnorm(K, mu=prior.mean, Sigma=diag(prior.var, 2))

  # generate mixture coefficients
  pi_k <- runif(K)
  pi_k <- pi_k/sum(pi_k)

  # generate the data
  obs <- matrix(0, nrow=N, ncol=p)
  z <- numeric(N)
  sigma_k <- matrix(0, nrow=K, ncol=p)

  for (i in 1:K)
    sigma_k[i,] <- runif(p)

  for (i in 1:N)
  {
    # draw the observation from a component according to coefficient
    z[i] <- sample(1:K, 1, prob=pi_k)
    # draw the observation from the corresponding mixture location
    obs[i,] <- mvrnorm(1, mu=mu_k[z[i],], Sigma=diag(sigma_k[z[i],],p))
  }
  list(locs=mu_k, z=z, obs=obs, coefs=pi_k)
}
```

In [25]:
```
# plot 2d data from a mixture
plot.mixture <- function(locs, z, obs)
{
  stopifnot(dim(obs)[2]==2)
  z <- as.factor(z)
  df1 <- data.frame(x=obs[,1], y=obs[,2], z=z)
```

```
  df2 <- data.frame(x=locs[,1], y=locs[,2])
  p <- ggplot()
  p <- p + geom_point(data=df1, aes(x=x, y=y, colour=z), shape=16,
                      size=2, alpha=0.75)
  p <- p + geom_point(data=df2, aes(x=x, y=y), shape=16, size=3)
  p <- p + theme(legend.position="none")
  p
}
```

```
In [26]: # plot 2D data as a scatter plot
         plot.data <- function(dat)
         {
           stopifnot(dim(dat)[2]==2)
           df1 <- data.frame(x=dat[,1], y=dat[,2])
           p <- ggplot()
           p <- p + geom_point(data=df1, aes(x=x, y=y), size=2, alpha=0.75)
           p
         }
```

Let us generate the data

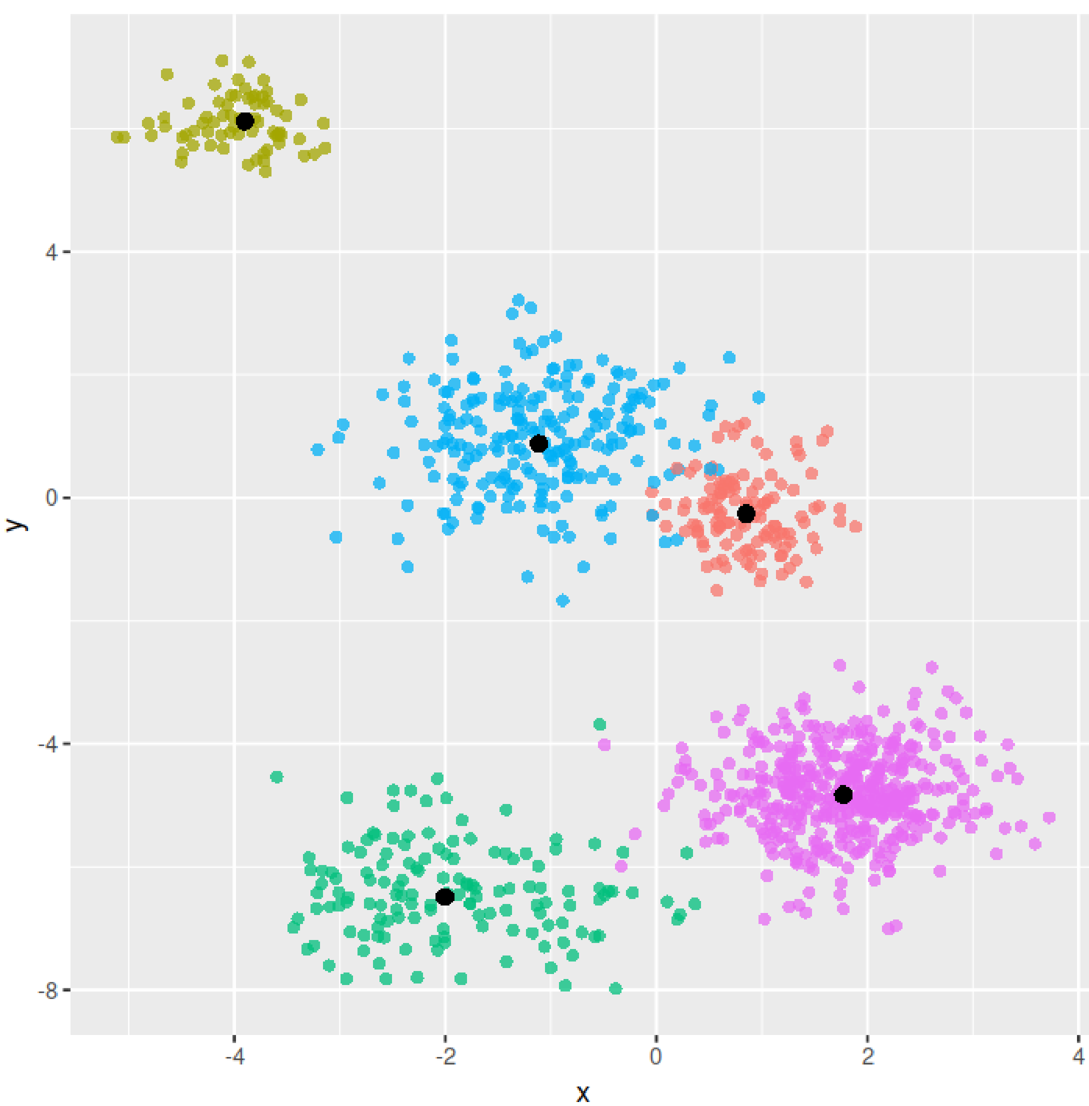
```
In [27]: N <- 1000
         K <- 5
         centre <- c(0,0)
         dispersion <- 10
```

generate 2D data as a mixture of 5 Gaussians, each axis-aligned (therefore the two variables are independent) with different variances the centers and coefficients of the mixture are chosen randomly

```
In [28]: d <- generate.data (N,K,centre,dispersion)
```

these are the components of the mixture

```
In [29]: plot.mixture(d$locs, d$z, d$obs)
```



may be we want to have a look at the unconditional density p(x)
compute 2D kernel density

```
In [30]: z <- kde2d(d$obs[,1], d$obs[,2], n=50)
```

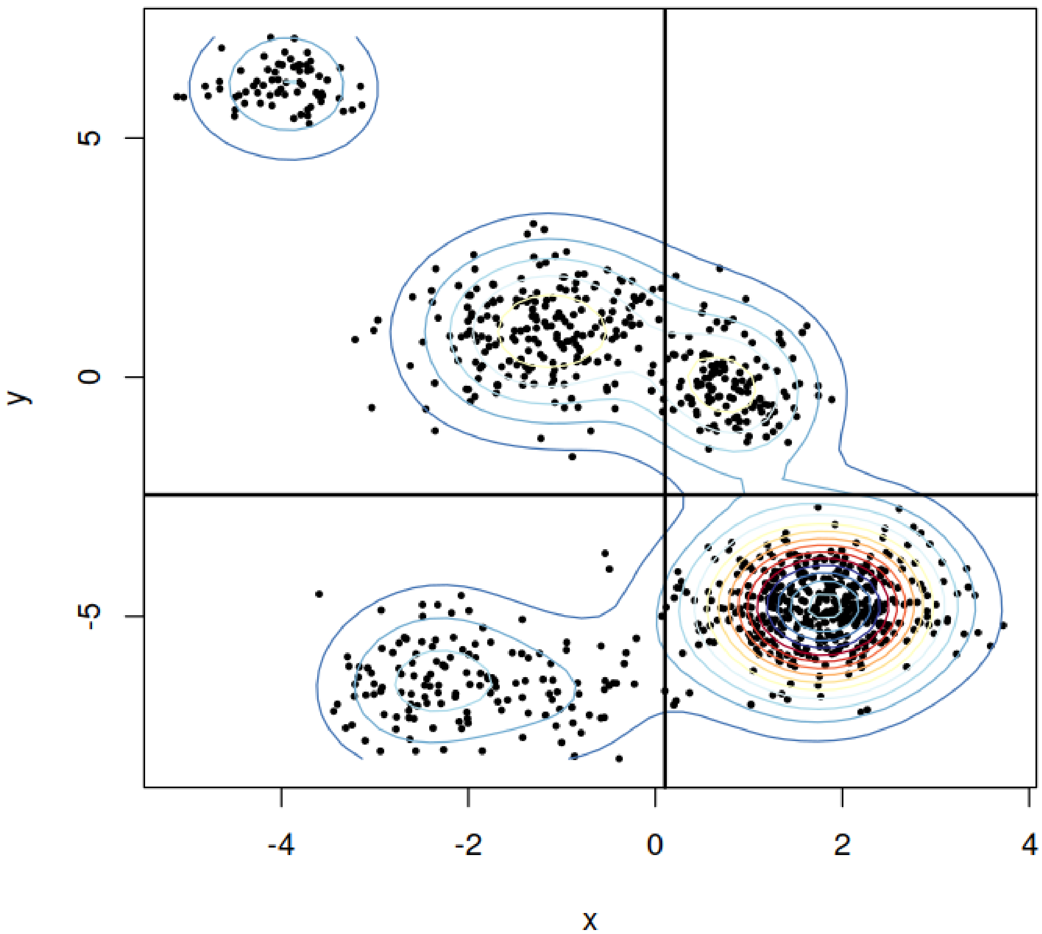
some pretty colors

```
In [31]: library(RColorBrewer)
         colorets <- rev(brewer.pal(11, "RdYlBu"))
```

this is the raw data (what the clustering method sees) and a contour plot of the unconditional density

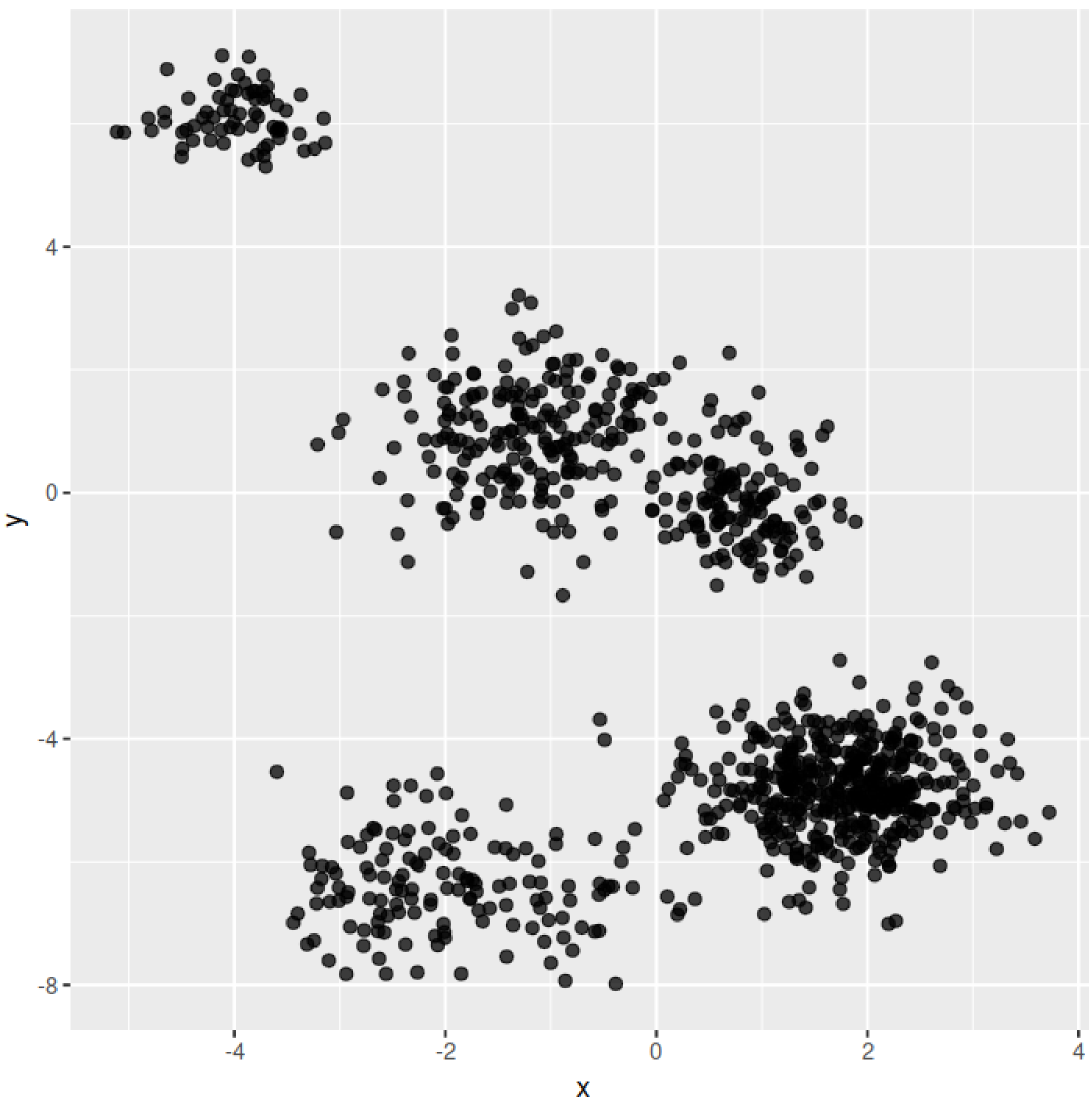
```
In [32]: plot(d$obs, xlab="x", ylab="y", pch=19, cex=.4)
```

```
contour(z, drawlabels=FALSE, nlevels=22, col=colorets, add=TRUE)
abline(h=mean(d$obs[,2]), v=mean(d$obs[,1]), lwd=2)
```



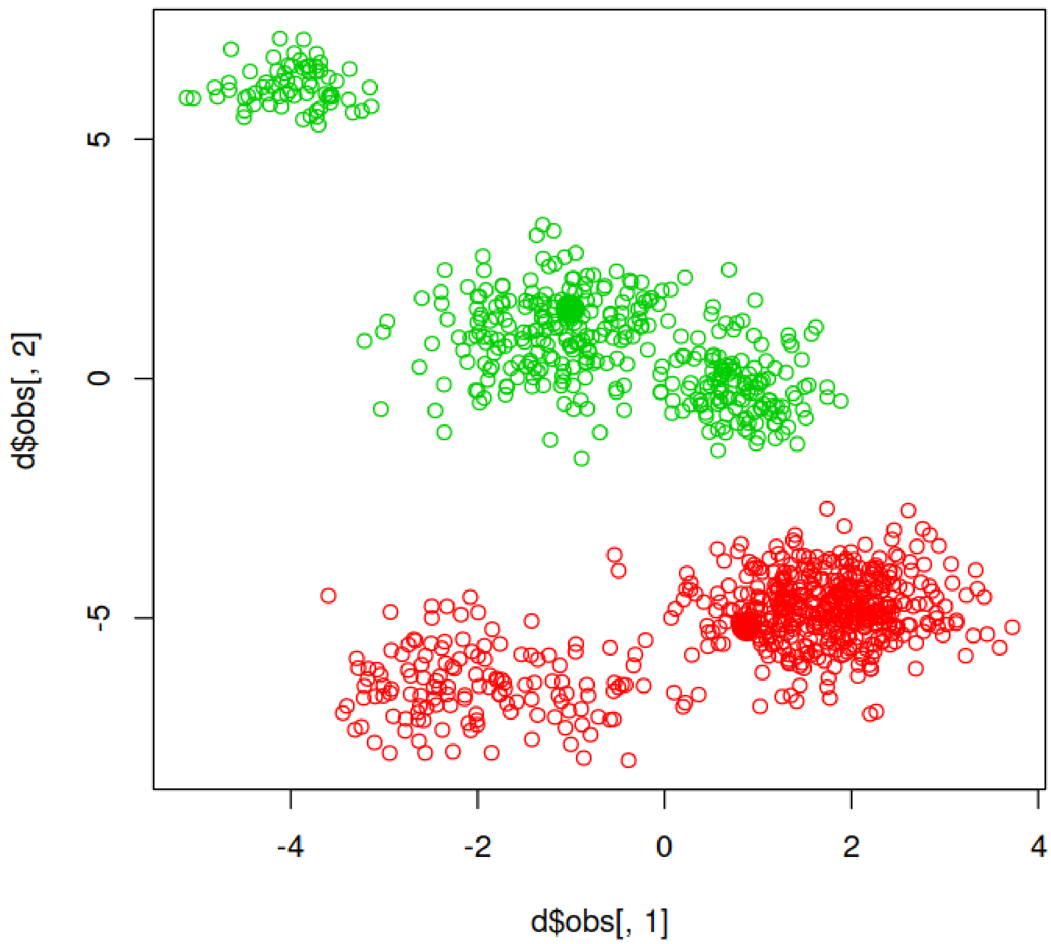
a simpler way of plotting the data

In [33]:
```
plot.data(d$obs)
```



let us try first with k-means (K=2)

```
In [34]: K <- 2

         kmeans2.2 <- cclust (d$obs,K,iter.max=100,method="kmeans",dist="euclidean")

         plot(d$obs[,1],d$obs[,2],col=(kmeans2.2$cluster+1))
         points(kmeans2.2$centers,col=seq(1:kmeans2.2$ncenters)+1,cex=2,pch=19)
```

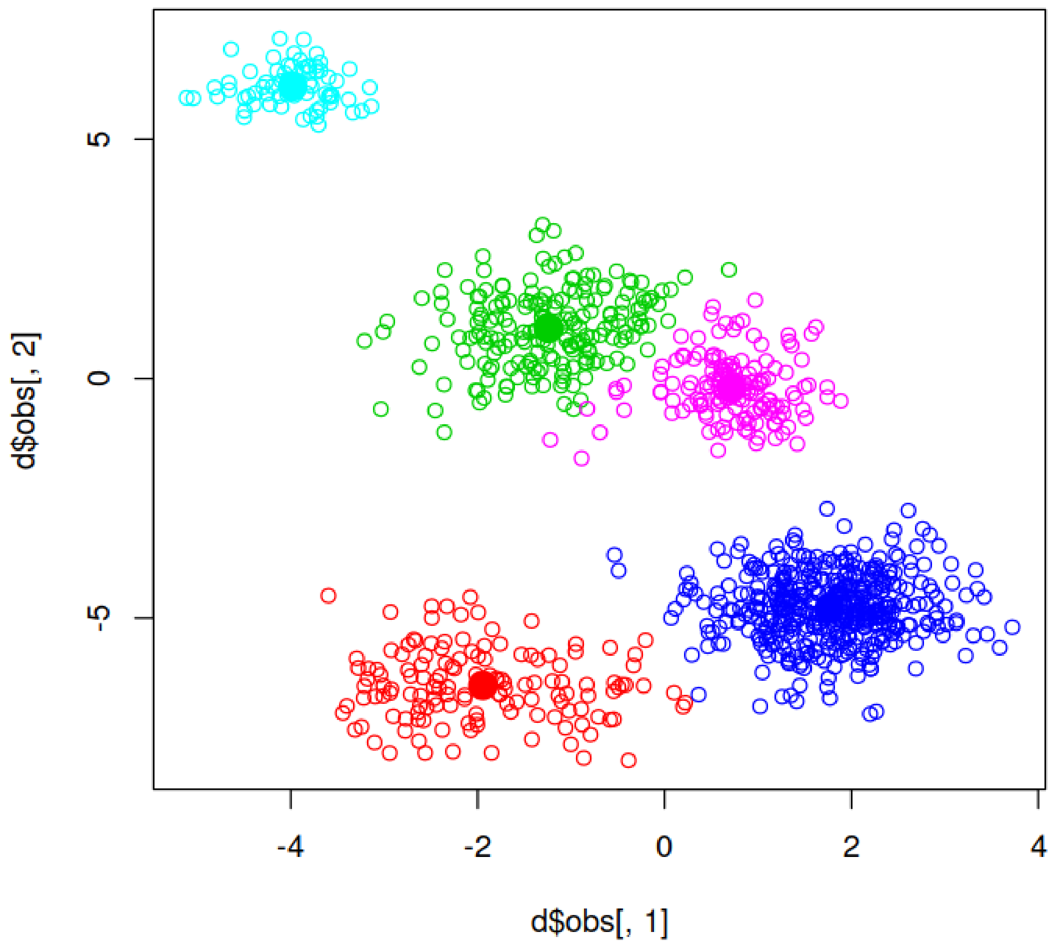


Can we be indulgent with the result? we know the truth is there are 5 clusters,
Is this is a reasonable result if we ask for 2?
clustering quality as measured by the Calinski-Harabasz index

```
In [35]: (CH2.2 <- clustIndex(kmeans2.2,d$obs, index="calinski"))
```

calinski: 2041.23491010702
let us try now with k-means (K=5)

```
In [36]: K <- 5

         kmeans2.5 <- cclust (d$obs,K,iter.max=100,method="kmeans",dist="euclidean")

         plot(d$obs[,1],d$obs[,2],col=(kmeans2.5$cluster+1))
         points(kmeans2.5$centers,col=seq(1:kmeans2.5$ncenters)+1,cex=2,pch=19)
```



This time the result has even more chances of being largely incorrect because there are more ways of getting a wrong solution

clustering quality as measured by the Calinski-Harabasz index

```
In [37]: (CH2.5 <- clustIndex(kmeans2.5,d$obs, index="calinski"))
```

calinski: 4251.74389464729

at least CH2.5 >> CH2.2 ... so C-H does a good job

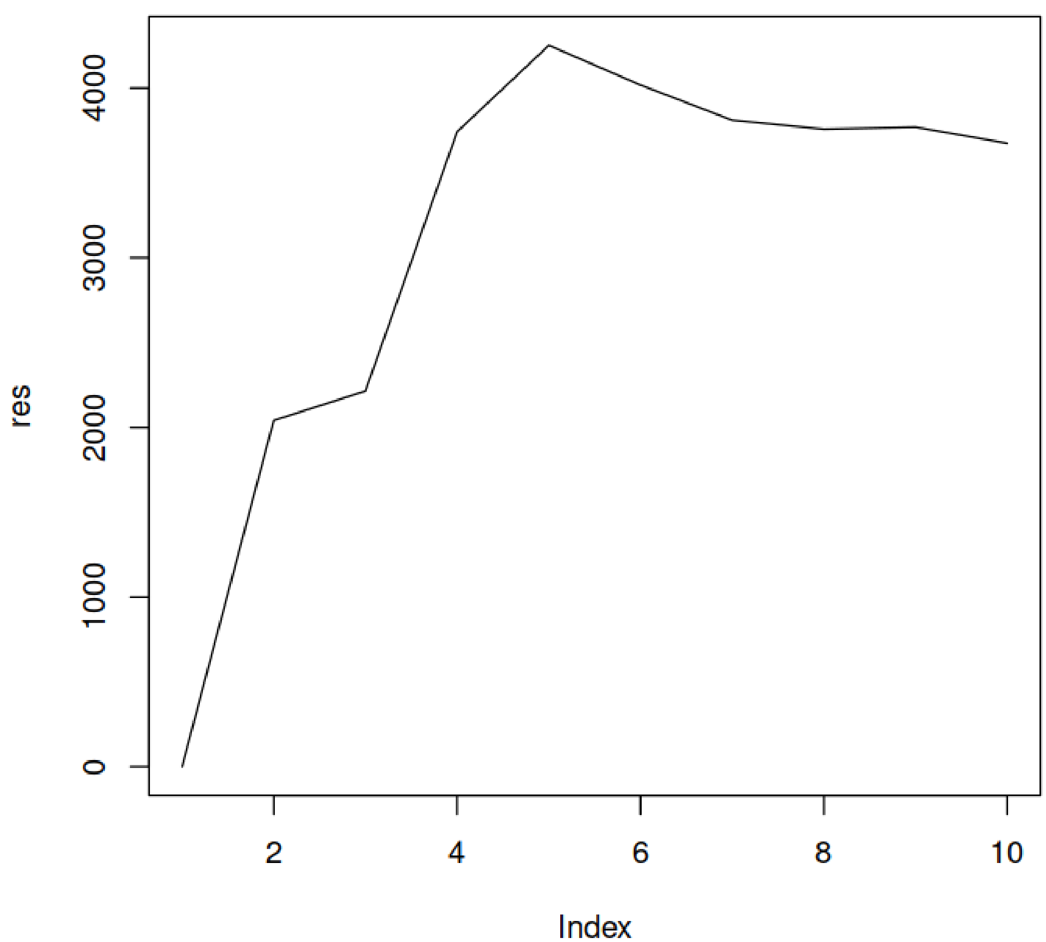
In class we saw that k-means is usually re-run several times

```
In [38]: do.kmeans <- function (whatK)
         {
           r <- cclust (d$obs,whatK,iter.max=100,method="kmeans",dist="euclidean")
           (clustIndex(r,d$obs, index="calinski"))
```

```
}

max (r <- replicate (100, do.kmeans(5)))
```

4251.74389464729
so it is not a matter of wrong initialization this is really the best k-means can do here
this may take a while

```
In [39]: res <- vector("numeric",10)
         for (K in 2:10)
           res[K] <- max (r <- replicate (100, do.kmeans(K)))

         plot(res, type="l")
```



the conclusion is that k-means + C-H bet for 5 clusters ... not bad, not bad ...

but the real *shape* of the clusters cannot be captured, because k-means only "sees" spherical clusters and these are ellipsoidal

let us try now E-M

```
In [40]: library(Rmixmod)
```

```
Loading required package: Rcpp
Rmixmod v. 2.1.2 / URI: www.mixmod.org
```

This method performs E-M for mixture densities, including mixtures of Gaussians we can specify which family of gaussians we intend to fit:

- "general" for the general family, "diagonal" for the diagonal family,
- "spherical" for the spherical family and "all" for all families (meaning the union)

  WARNING: default is "general".

suppose first that we know the truth and specify axis-aligned densities (i.e., independent variables)

```
In [41]: fammodel <- mixmodGaussianModel (family="diagonal", equal.proportions=FALSE)

         z <- mixmodCluster (data.frame(d$obs), models = fammodel, nbCluster = 5)

         summary(z)
```

```
**************************************************************
* Number of samples    =  1000
* Problem dimension    =  2
**************************************************************
*       Number of cluster =  5
*              Model Type =  Gaussian_pk_Lk_Bk
*               Criterion =  BIC(7016.0809)
*              Parameters =  list by cluster
*                  Cluster  1 :
                        Proportion =  0.0680
                             Means =  -3.9795 6.1025
                         Variances = |     0.1841     0.0000 |
                                     |     0.0000     0.1673 |
*                  Cluster  2 :
                        Proportion =  0.2189
                             Means =  -1.1349 0.9525
                         Variances = |     0.5681     0.0000 |
                                     |     0.0000     0.7267 |
*                  Cluster  3 :
                        Proportion =  0.4478
                             Means =  1.7672 -4.7968
```

```
                         Variances = |     0.4383     0.0000 |
                                     |     0.0000     0.4819 |
*                 Cluster  4 :
                        Proportion =  0.1231
                             Means =  0.8438 -0.2265
                         Variances = |     0.1573     0.0000 |
                                     |     0.0000     0.3783 |
*                 Cluster  5 :
                        Proportion =  0.1422
                             Means =  -1.9238 -6.4042
                         Variances = |     0.8494     0.0000 |
                                     |     0.0000     0.5556 |
*         Log-likelihood =  -3425.1474
*************************************************************
```

the final centers

```
In [42]: (means <- z@bestResult@parameters@mean)
```

-3.9795341	6.1025470
-1.1348664	0.9525007
1.7672169	-4.7968487
0.8438481	-0.2265113
-1.9238207	-6.4042320

if you want hard assignments

```
In [43]: (found.clusters <- z@bestResult@partition)
```

1. 2 2. 3 3. 3 4. 5 5. 3 6. 3 7. 3 8. 3 9. 3 10. 2 11. 5 12. 1 13. 3 14. 3 15. 3 16. 2 17. 4 18. 3 19. 3 20. 3 21. 4 22. 3 23. 3 24. 3 25. 2 26. 2 27. 1 28. 2 29. 3 30. 4 31. 1 32. 2 33. 3 34. 3 35. 1 36. 3 37. 3 38. 2 39. 4 40. 5 41. 3 42. 2 43. 3 44. 4 45. 2 46. 3 47. 4 48. 5 49. 2 50. 2 51. 3 52. 4 53. 3 54. 4 55. 5 56. 3 57. 2 58. 4 59. 5 60. 3 61. 5 62. 2 63. 3 64. 2 65. 3 66. 3 67. 2 68. 3 69. 2 70. 3 71. 3 72. 4 73. 5 74. 3 75. 5 76. 5 77. 4 78. 3 79. 3 80. 3 81. 4 82. 2 83. 3 84. 5 85. 2 86. 3 87. 1 88. 5 89. 5 90. 3 91. 3 92. 3 93. 3 94. 4 95. 4 96. 2 97. 3 98. 4 99. 3 100. 2 101. 3 102. 2 103. 3 104. 3 105. 3 106. 3 107. 4 108. 2 109. 2 110. 5 111. 3 112. 3 113. 4 114. 3 115. 3 116. 4 117. 2 118. 3 119. 1 120. 5 121. 2 122. 3 123. 4 124. 3 125. 3 126. 3 127. 2 128. 4 129. 3 130. 3 131. 3 132. 5 133. 4 134. 2 135. 4 136. 3 137. 2 138. 1 139. 3 140. 3 141. 5 142. 4 143. 5 144. 3 145. 3 146. 1 147. 5 148. 3 149. 3 150. 3 151. 3 152. 3 153. 3 154. 2 155. 3 156. 3 157. 3 158. 3 159. 5 160. 3 161. 3 162. 3 163. 3 164. 3 165. 3 166. 3 167. 3 168. 2 169. 1 170. 5 171. 3 172. 3 173. 5 174. 1 175. 4 176. 3 177. 5 178. 2 179. 2 180. 3 181. 3 182. 2 183. 5 184. 2 185. 4 186. 5 187. 3 188. 5 189. 1 190. 3 191. 2 192. 4 193. 5 194. 3 195. 2 196. 4 197. 2 198. 3 199. 5 200. 2 201. 4 202. 4 203. 1 204. 2 205. 2 206. 1 207. 4 208. 2 209. 3 210. 4 211. 3 212. 3 213. 5 214. 2 215. 2 216. 2 217. 2 218. 3 219. 1 220. 3 221. 4 222. 5 223. 2 224. 5 225. 5 226. 3 227. 2 228. 2 229. 4 230. 2 231. 3 232. 3 233. 3 234. 3 235. 3 236. 2 237. 5 238. 3 239. 3 240. 2 241. 3 242. 5 243. 5 244. 3 245. 3 246. 5 247. 3 248. 3 249. 2 250. 1 251. 3 252. 2 253. 2 254. 2 255. 5 256. 3 257. 4 258. 2 259. 5 260. 3 261. 1 262. 3 263. 2 264. 5 265. 5 266. 3 267. 3 268. 4 269. 4 270. 3 271. 2 272. 5 273. 3 274. 1 275. 4 276. 3 277. 5 278. 5 279. 2 280. 2 281. 2 282. 2 283. 3 284. 4 285. 2 286. 4 287. 3 288. 4 289. 5 290. 3 291. 3 292. 2 293. 3 294. 3 295. 5 296. 3 297. 3 298. 4 299. 5 300. 5 301. 1 302. 3 303. 3 304. 5 305. 2 306. 4 307. 4 308. 2 309. 3 310. 2 311. 3 312. 4 313. 4 314. 5 315. 3 316. 4 317. 1 318. 3 319. 3 320. 3 321. 3 322. 2 323. 2 324. 3 325. 2

326. 3 327. 3 328. 4 329. 3 330. 3 331. 3 332. 3 333. 2 334. 1 335. 1 336. 3 337. 2 338. 2 339. 1 340. 3 341. 2 342. 4 343. 2 344. 5 345. 5 346. 3 347. 3 348. 3 349. 5 350. 3 351. 4 352. 3 353. 1 354. 3 355. 3 356. 3 357. 2 358. 3 359. 4 360. 3 361. 2 362. 2 363. 2 364. 3 365. 3 366. 3 367. 3 368. 5 369. 1 370. 2 371. 5 372. 4 373. 2 374. 5 375. 5 376. 5 377. 3 378. 5 379. 5 380. 3 381. 2 382. 5 383. 5 384. 3 385. 3 386. 3 387. 2 388. 2 389. 3 390. 3 391. 4 392. 2 393. 2 394. 2 395. 3 396. 5 397. 2 398. 3 399. 5 400. 3 401. 3 402. 4 403. 3 404. 5 405. 3 406. 3 407. 3 408. 2 409. 3 410. 3 411. 3 412. 3 413. 3 414. 3 415. 5 416. 2 417. 3 418. 3 419. 5 420. 3 421. 4 422. 3 423. 2 424. 2 425. 4 426. 2 427. 2 428. 5 429. 1 430. 3 431. 4 432. 3 433. 2 434. 2 435. 5 436. 2 437. 3 438. 5 439. 1 440. 1 441. 1 442. 5 443. 3 444. 5 445. 3 446. 3 447. 3 448. 1 449. 2 450. 2 451. 3 452. 3 453. 3 454. 2 455. 3 456. 3 457. 3 458. 4 459. 3 460. 3 461. 3 462. 3 463. 2 464. 2 465. 4 466. 2 467. 3 468. 2 469. 5 470. 3 471. 4 472. 3 473. 3 474. 3 475. 4 476. 1 477. 3 478. 3 479. 2 480. 3 481. 3 482. 2 483. 5 484. 3 485. 3 486. 4 487. 3 488. 4 489. 3 490. 2 491. 5 492. 4 493. 2 494. 5 495. 3 496. 2 497. 5 498. 3 499. 1 500. 2 501. 3 502. 3 503. 3 504. 3 505. 5 506. 2 507. 3 508. 4 509. 2 510. 3 511. 4 512. 2 513. 3 514. 3 515. 4 516. 2 517. 3 518. 3 519. 2 520. 3 521. 1 522. 1 523. 5 524. 5 525. 3 526. 4 527. 3 528. 3 529. 3 530. 5 531. 4 532. 3 533. 2 534. 4 535. 1 536. 2 537. 2 538. 3 539. 2 540. 4 541. 3 542. 3 543. 3 544. 5 545. 4 546. 3 547. 3 548. 3 549. 3 550. 1 551. 3 552. 5 553. 3 554. 1 555. 3 556. 5 557. 5 558. 3 559. 3 560. 3 561. 2 562. 1 563. 3 564. 1 565. 5 566. 3 567. 5 568. 3 569. 4 570. 2 571. 3 572. 4 573. 5 574. 2 575. 3 576. 2 577. 2 578. 3 579. 3 580. 1 581. 5 582. 3 583. 2 584. 3 585. 1 586. 3 587. 2 588. 3 589. 2 590. 4 591. 3 592. 2 593. 1 594. 3 595. 5 596. 3 597. 3 598. 3 599. 3 600. 3 601. 2 602. 5 603. 1 604. 3 605. 3 606. 3 607. 5 608. 4 609. 5 610. 3 611. 2 612. 2 613. 4 614. 3 615. 3 616. 3 617. 3 618. 2 619. 2 620. 3 621. 4 622. 4 623. 5 624. 3 625. 3 626. 3 627. 1 628. 2 629. 4 630. 3 631. 2 632. 3 633. 3 634. 2 635. 2 636. 3 637. 3 638. 2 639. 2 640. 5 641. 3 642. 3 643. 2 644. 2 645. 5 646. 1 647. 3 648. 3 649. 5 650. 3 651. 3 652. 5 653. 2 654. 4 655. 5 656. 4 657. 3 658. 3 659. 3 660. 4 661. 4 662. 3 663. 1 664. 5 665. 3 666. 3 667. 3 668. 2 669. 4 670. 3 671. 2 672. 3 673. 2 674. 1 675. 3 676. 3 677. 3 678. 3 679. 3 680. 3 681. 4 682. 3 683. 3 684. 3 685. 4 686. 4 687. 5 688. 2 689. 4 690. 3 691. 3 692. 5 693. 5 694. 2 695. 3 696. 2 697. 3 698. 1 699. 3 700. 5 701. 2 702. 3 703. 3 704. 2 705. 3 706. 2 707. 2 708. 4 709. 3 710. 3 711. 3 712. 2 713. 4 714. 3 715. 3 716. 2 717. 2 718. 3 719. 1 720. 3 721. 3 722. 2 723. 3 724. 5 725. 1 726. 4 727. 3 728. 5 729. 5 730. 3 731. 3 732. 2 733. 2 734. 4 735. 3 736. 3 737. 5 738. 2 739. 5 740. 3 741. 3 742. 3 743. 2 744. 2 745. 3 746. 3 747. 4 748. 4 749. 2 750. 2 751. 4 752. 3 753. 4 754. 3 755. 2 756. 3 757. 2 758. 5 759. 3 760. 3 761. 2 762. 2 763. 2 764. 5 765. 1 766. 5 767. 1 768. 4 769. 3 770. 4 771. 3 772. 3 773. 4 774. 3 775. 3 776. 3 777. 3 778. 3 779. 5 780. 3 781. 3 782. 4 783. 5 784. 5 785. 3 786. 3 787. 1 788. 3 789. 2 790. 5 791. 1 792. 3 793. 4 794. 1 795. 3 796. 2 797. 2 798. 3 799. 4 800. 3 801. 3 802. 1 803. 3 804. 3 805. 3 806. 3 807. 2 808. 4 809. 3 810. 5 811. 2 812. 4 813. 3 814. 3 815. 2 816. 3 817. 2 818. 1 819. 5 820. 3 821. 4 822. 5 823. 5 824. 2 825. 2 826. 4 827. 5 828. 4 829. 4 830. 2 831. 4 832. 3 833. 3 834. 3 835. 3 836. 3 837. 5 838. 5 839. 2 840. 2 841. 3 842. 3 843. 5 844. 3 845. 3 846. 3 847. 2 848. 3 849. 3 850. 5 851. 2 852. 3 853. 3 854. 5 855. 5 856. 3 857. 5 858. 3 859. 3 860. 4 861. 3 862. 1 863. 4 864. 3 865. 2 866. 4 867. 1 868. 3 869. 3 870. 5 871. 4 872. 1 873. 2 874. 3 875. 5 876. 1 877. 3 878. 3 879. 3 880. 2 881. 3 882. 5 883. 2 884. 1 885. 1 886. 1 887. 4 888. 3 889. 4 890. 2 891. 2 892. 2 893. 4 894. 4 895. 3 896. 4 897. 3 898. 3 899. 3 900. 2 901. 1 902. 3 903. 3 904. 3 905. 3 906. 5 907. 2 908. 5 909. 3 910. 3 911. 4 912. 4 913. 3 914. 5 915. 5 916. 3 917. 3 918. 3 919. 3 920. 2 921. 5 922. 3 923. 3 924. 3 925. 3 926. 2 927. 5 928. 4 929. 3 930. 3 931. 2 932. 2 933. 1 934. 3 935. 5 936. 5 937. 2 938. 2 939. 2 940. 3 941. 3 942. 3 943. 5 944. 3 945. 3 946. 4 947. 3 948. 2 949. 1 950. 4 951. 3 952. 1 953. 2 954. 3 955. 4 956. 3 957. 2 958. 2 959. 3 960. 4 961. 2 962. 4 963. 3 964. 2 965. 3 966. 3 967. 3 968. 4 969. 3 970. 3 971. 3 972. 2 973. 2 974. 3 975. 3 976. 3 977. 3 978. 5 979. 3 980. 2 981. 3 982. 2 983. 4 984. 3 985. 3 986. 2 987. 3 988. 3 989. 3 990. 5 991. 2 992. 3 993. 2 994. 3 995. 2 996. 2 997. 3 998. 1 999. 5 1000. 3

other interesting outcomes are:

the estimated covariance matrices for each cluster

```
In [44]: z@bestResult@parameters@variance
```

1. | | |
   |---|---|
   | 0.1840552 | 0.000000 |
   | 0.0000000 | 0.167305 |

2. | | |
   |---|---|
   | 0.5681031 | 0.0000 |
   | 0.0000000 | 0.7267 |

3. | | |
   |---|---|
   | 0.43829 | 0.0000000 |
   | 0.00000 | 0.4818727 |

4. | | |
   |---|---|
   | 0.1572885 | 0.0000000 |
   | 0.0000000 | 0.3783127 |

5. | | |
   |---|---|
   | 0.8493765 | 0.0000000 |
   | 0.0000000 | 0.5555805 |

self-explained

```
In [45]: z@bestResult@likelihood
```

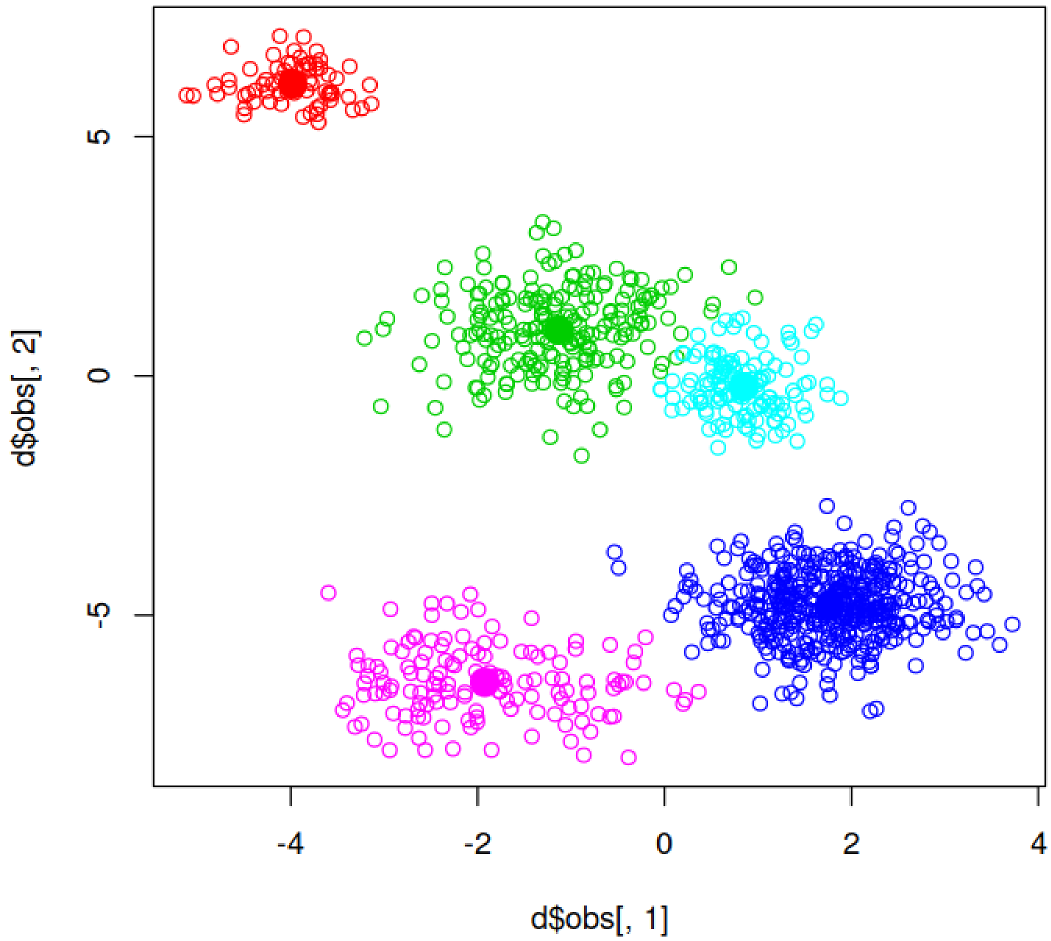
-3425.14736462991
the posterior probabilities = soft assignments = the gamma_k(x_n) in class

```
In [46]: (z@bestResult@proba)
```

5.378244e-34	9.999994e-01	5.515228e-24	5.938099e-07	2.446627e-28
2.790386e-164	2.333798e-10	9.999998e-01	1.387041e-08	1.817091e-07
1.774785e-198	7.118278e-15	9.999957e-01	5.549683e-15	4.265477e-06
9.474941e-222	2.921002e-18	2.954085e-07	7.659677e-33	9.999997e-01
4.488956e-208	2.363658e-16	1.000000e+00	2.068831e-18	1.791060e-08
2.568679e-203	1.848492e-15	9.999937e-01	6.960039e-16	6.270896e-06
2.049372e-206	6.366546e-16	9.999970e-01	1.521195e-16	3.033571e-06
6.510808e-200	2.539839e-15	1.000000e+00	3.630641e-16	4.101595e-08
4.711541e-222	4.957860e-18	9.999998e-01	1.472532e-20	1.911063e-07
5.122163e-40	1.000000e+00	2.677774e-22	1.109343e-15	3.221830e-20
2.743707e-215	4.222983e-16	2.774001e-03	5.070827e-24	9.972260e-01
1.000000e+00	9.207671e-13	2.230509e-73	7.670122e-59	5.956955e-66
2.999289e-145	5.776977e-07	9.998988e-01	4.355664e-06	9.626922e-05
8.165306e-235	4.714164e-19	9.999557e-01	8.439122e-22	4.431761e-05
1.369330e-195	2.063755e-14	9.999899e-01	2.016107e-14	1.007998e-05
3.489337e-38	9.998066e-01	1.549163e-22	1.934008e-04	1.038241e-28
5.539821e-76	1.650330e-02	3.657689e-11	9.834967e-01	8.865224e-19
4.250076e-160	9.405263e-10	1.000000e+00	3.902589e-08	5.741134e-09
5.042478e-185	8.616563e-13	9.999342e-01	1.160803e-12	6.576544e-05
1.100850e-189	5.924764e-14	9.999999e-01	1.085249e-13	1.106313e-07
1.136005e-100	1.107357e-03	7.353875e-06	9.988853e-01	8.364261e-14
2.022404e-218	6.550540e-17	9.997928e-01	1.216618e-18	2.071743e-04
3.301088e-197	7.268042e-15	9.999994e-01	6.338650e-15	6.084932e-07
2.220260e-195	2.507473e-14	9.999829e-01	2.156865e-14	1.711000e-05
3.009891e-35	1.000000e+00	2.844312e-23	4.701969e-09	1.011965e-25
1.245661e-61	9.999068e-01	2.264888e-13	9.318752e-05	4.149642e-16
1.000000e+00	7.873739e-09	3.938246e-62	2.285327e-47	2.501995e-56
9.784397e-29	1.000000e+00	2.104395e-26	4.936636e-13	4.291587e-27
6.514401e-206	4.786805e-16	9.999998e-01	5.645316e-17	1.954348e-07
1.520895e-94	1.286166e-03	1.441994e-07	9.987137e-01	2.319870e-16
2.405977e-178	2.117193e-12	9.999999e-01	2.202396e-11	1.058939e-07
4.659067e-66	9.859530e-01	8.983143e-12	1.404704e-02	7.215671e-16
1.935517e-70	9.989831e-01	1.205887e-10	1.016883e-03	1.688192e-13
9.645170e-172	4.041920e-11	9.999857e-01	3.636814e-10	1.430721e-05
3.572818e-188	2.013685e-12	9.847558e-01	2.637532e-14	1.524417e-02
1.198490e-190	1.221201e-13	9.999702e-01	1.435781e-13	2.984506e-05
5.776804e-170	3.365973e-11	9.999999e-01	1.071311e-09	1.377697e-07
2.263976e-228	1.415219e-20	2.788158e-12	3.559399e-45	1.000000e+00
6.121465e-223	3.780605e-18	9.999999e-01	6.273066e-21	1.322899e-07
1.750790e-39	1.000000e+00	1.922463e-21	4.853997e-08	4.708330e-24
5.216380e-218	1.606470e-17	9.999997e-01	1.918446e-19	3.267061e-07
5.504045e-60	1.000000e+00	1.412003e-14	6.966302e-09	9.950387e-15
2.641499e-81	5.622235e-03	2.864077e-10	9.943778e-01	2.556893e-18
6.416354e-189	2.611764e-13	9.999314e-01	2.422532e-13	6.859156e-05
8.614630e-176	4.308530e-11	9.983587e-01	9.484107e-12	1.641251e-03
4.992997e-46	9.999969e-01	8.767555e-19	3.095791e-06	6.228731e-22
7.416405e-188	2.257153e-13	9.999886e-01	5.196733e-13	1.136108e-05
1.434922e-168	7.717655e-11	9.999979e-01	2.002655e-09	2.091063e-06
1.819343e-221	1.625735e-17	9.999655e-01	3.074042e-19	3.448507e-05
2.802252e-172	1.255009e-11	4.542838e-06	3.338230e-24	9.999955e-01
1.753418e-46	1.000000e+00	3.058242e-20	2.904533e-16	2.576071e-17
7.010374e-206	5.556057e-16	9.999994e-01	1.078744e-16	5.982140e-07

This is a graphical summary of the clustering

```
In [47]: plot(d$obs[,1],d$obs[,2],col=(found.clusters+1))
         points(means,col=seq(1:5)+1,cex=2,pch=19)
```



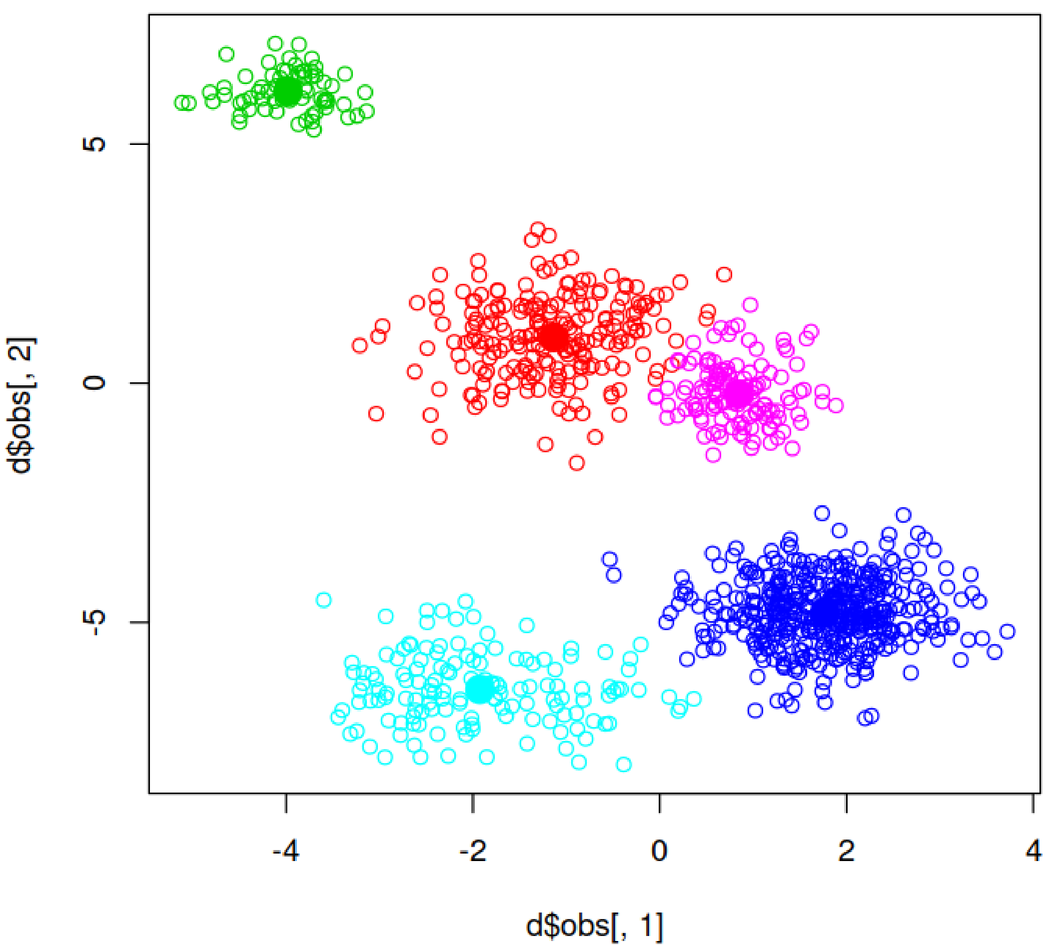
it was very likely that E-M performed extremely well why? because we knew the truth (cluster form and number)

suppose now we do not the know the truth but we still wish to fit general gaussians

```
In [48]: fammodel <- mixmodGaussianModel (family="general", equal.proportions=FALSE)

         z <- mixmodCluster (data.frame(d$obs),models = fammodel, nbCluster = 5)

         summary(z)
```

```
****************************************************************
* Number of samples    =  1000
* Problem dimension    =  2
****************************************************************
*       Number of cluster =  5
*              Model Type =  Gaussian_pk_Lk_D_Ak_D
*               Criterion =  BIC(7022.9405)
*              Parameters =  list by cluster
*                  Cluster  1 :
                        Proportion =  0.2187
                             Means =  -1.1366 0.9537
                         Variances = |     0.5638     0.0035 |
                                     |     0.0035     0.7279 |
*                  Cluster  2 :
                        Proportion =  0.0680
                             Means =  -3.9795 6.1025
                         Variances = |     0.1843    -0.0004 |
                                     |    -0.0004     0.1671 |
*                  Cluster  3 :
                        Proportion =  0.4479
                             Means =  1.7669 -4.7969
                         Variances = |     0.4385     0.0009 |
                                     |     0.0009     0.4822 |
*                  Cluster  4 :
                        Proportion =  0.1421
                             Means =  -1.9247 -6.4049
                         Variances = |     0.8502    -0.0063 |
                                     |    -0.0063     0.5523 |
*                  Cluster  5 :
                        Proportion =  0.1233
                             Means =  0.8430 -0.2263
                         Variances = |     0.1587     0.0046 |
                                     |     0.0046     0.3771 |
*          Log-likelihood =  -3425.1233
****************************************************************
```

```
In [49]: means <- z@bestResult@parameters@mean
         found.clusters <- z@bestResult@partition

         plot(d$obs[,1],d$obs[,2],col=(found.clusters+1))
         points(means,col=seq(1:5)+1,cex=2,pch=19)
```



the method works also very smoothly why? because the data *is* gaussian
compare the estimated centers

```
In [50]: means
```

-1.1365764	0.9536966
-3.9795341	6.1025470
1.7669497	-4.7968709
-1.9246850	-6.4049024
0.8430279	-0.2263335

with the truth (note the clusters may appear in a different order)

```
In [51]: d$locs
```

0.8511701	-0.2618734
-3.8992587	6.1179985
-2.0036436	-6.4867479
-1.1153905	0.8782877
1.7724012	-4.8255111

or the estimated coefficients

```
In [52]: sort(z@bestResult@parameters@proportions)
```

1. 0.0679999997741943 2. 0.123340290968537 3. 0.14208960863035 4. 0.218660412233532 5. 0.447909688393386

with the truth

```
In [53]: sort(d$coefs)
```

1. 0.0606961177019854 2. 0.110630466458976 3. 0.162127999479225 4. 0.227224024271389 5. 0.439321392088424