# MTH210: Lab 11 Solutions

## EM algorithm for Gaussian Mixture Model

1. **Go through the `classDemonstration.R` file that runs the Gaussian Mixture Model EM algorithm for the `eruptions` on the Old Faithful dataset.**

   No solutions required.

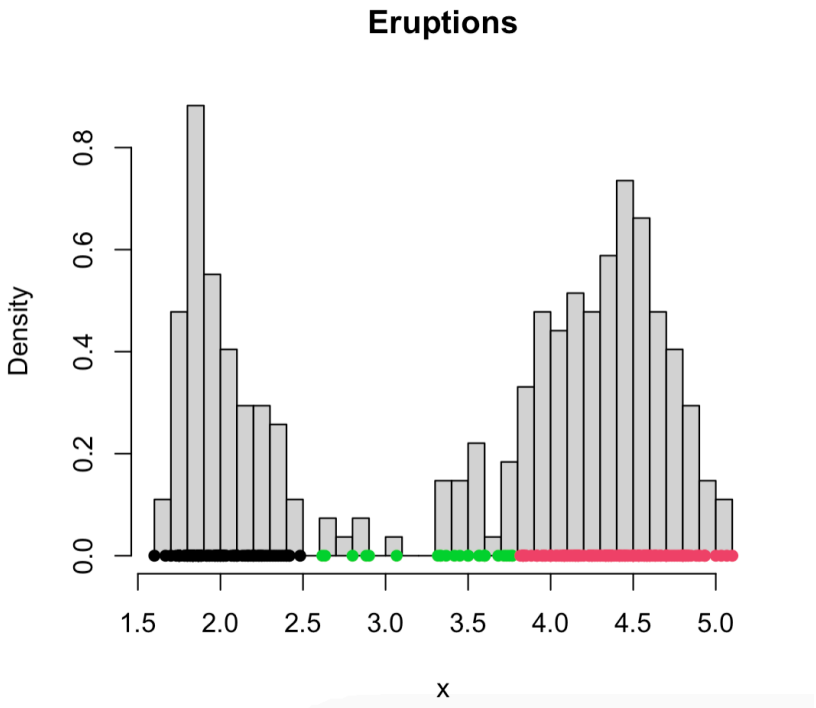
2. **Now, change the code so that it is a general function that takes any 1-dimensional data vector `x` and number fo desired clusters `C` as an argument. Your function should look like:**

```
# Function outputs estimated Zs (class allocations)
# and estimates of mu, sigma^2, and pis
GMMoneDim <- function(x, C = 2)
{
  ...
  rtn <- list(Zs, mu, sig2, pis)
}
```

   **Once done, call `GMMoneDim` for `eruptions`:**

```
erup2 <- GMMoneDim(faithpul$eruptions, C = 2)
erup3 <- GMMoneDim(faithpul$eruptions, C = 3)
```

   **`erup2` should match your results from the previous exercise. Using `erup3` reproduce the following allocation plot:**



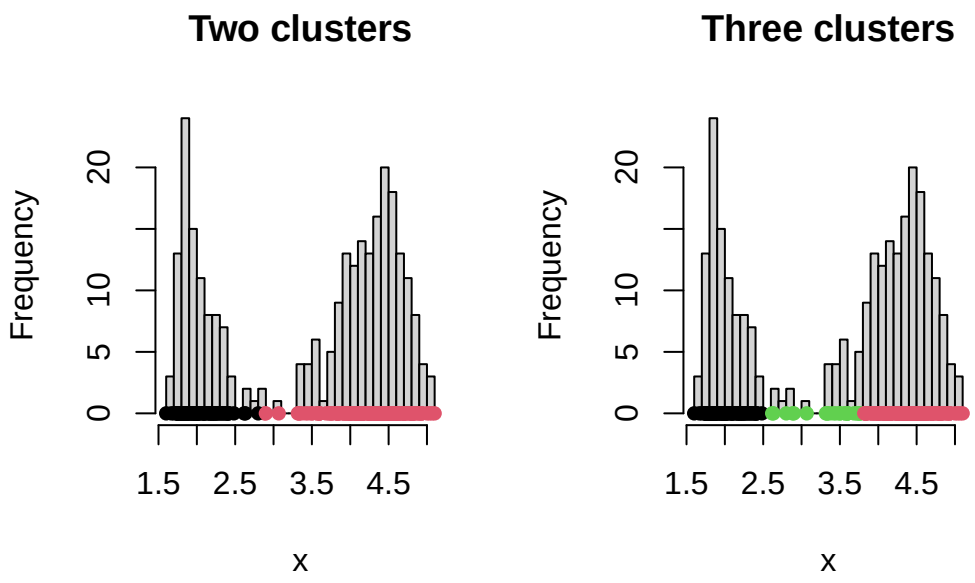

Below is the required function:

```
# Function that calculates the gamma_ick
gamma_ick <- function(x, mu, sig2, pis, C = 2)
{
  # making a full matrix gamma_prob
  gamma_prob <- matrix(0, nrow = length(x), ncol = C)
  for(c in 1:C)
  {
    gamma_prob[ ,c]  <- dnorm(x, mean = mu[c], sd = sqrt(sig2[c]))* pis[c]
  }

  # rows will sum to 1
  gamma_prob <- gamma_prob/(rowSums(gamma_prob))
  return(gamma_prob)
}

GMMoneDim <- function(x, C = 2)
{
  # Starting values
  pis <- rep(1/C, C)
  mu <- mean(x) + rnorm(C)
  sig2 <- rep(var(x), C)
```

```
  diff <- 100
  tol <- 1e-5
  iter <- 0

  # just for visualizing
  current <- c(pis, mu, sig2)
  while(diff > tol)
  {
    previous <- current
    iter <- iter + 1

    # E step
    Ep <- gamma_ick(x, mu, sig2, pis, C = C)

    # M-step
    pis <- colMeans(Ep)
    mu <- colSums(Ep*x) / colSums(Ep)
    for(c in 1:C)
    {
      sig2[c] <- sum(Ep[,c]*(x - mu[c])^2) / sum(Ep[,c])
    }
    current <- c(pis, mu, sig2)
    diff <- norm(previous - current, "2")
  }
  Ep <- gamma_ick(x, mu = mu, sig2 = sig2, pis = pis, C = C)
  Zs <- apply(Ep, 1, which.max)
  rtn <- list(Zs, mu, sig2, pis)
  return(rtn)
}
```

Now we will run this function for $C = 2$ and $C = 3$.

```
set.seed(1)
erup2 <- GMMoneDim(faithful$eruptions, C = 2)
erup3 <- GMMoneDim(faithful$eruptions, C = 3)
n <- length(faithful$eruptions)

par(mfrow = c(1,2))
hist(faithful$eruptions, breaks = 30, main = "Two clusters", xlab = "x")
points(faithful$eruptions, rep(0,n), col = erup2[[1]], pch = 16)
hist(faithful$eruptions, breaks = 30, main = "Three clusters", xlab = "x")
points(faithful$eruptions, rep(0,n), col = erup3[[1]], pch = 16)
```




3. **In class, we discussed how to choose $C$ using the Bayesian Information Criterion (BIC). Suppose we obtain a vector of estimates of $\theta = (\mu_1, \ldots, \mu_C, \sigma_1^2, \ldots, \sigma_C^2, \pi_1, \ldots, \pi_C)$. Denote these estimates by $\hat{\theta}$. Recall that the log likelihood for a given $C$ is**

$$\log f(x|\theta) = \log \left\{ \sum_{c=1}^{C} \pi_c f_c(x \mid \mu_c, \sigma_c^2) \right\}$$

**First, write a function that calculates the log-likelihood for the observed data, given a particular value of $\theta$:**

```
log.like <- function(x, mu, sig2, pis)
{
  n <- length(x)
  track <- 0
  for(i in 1:n)
  {
    track <- track + ...
  }
  ...
  return(...)
}
```

**Now, the BIC for an estimated $\hat{\theta}$ is**

$$\text{BIC}(\hat{\theta}) = -2 \log f(\mathbf{x}|\hat{\theta}) + K \log(n)\,,$$

**where $K$ is the number of parameters. In this 1-dimensional example $K = 3C - 1$. The choice of $C$ is the one that yields the lowest BIC. For the `eruptions` data, amongst $C =$**

**$2, 3, 4$, what value of $C$ is best?**

First, we complete the function to calculate the log-likelihood.

```
log.like <- function(x, mu, sig2, pis)
{
  n <- length(x)
  track <- 0
  # calculate for each data point
  for(i in 1:n)
  {
    track <- track + log(sum(dnorm(x[i], mean = mu, sd = sqrt(sig2))*pis))
  }
  return(track)
}
```

We will use this to calculate the BIC value. There was an error in the formula for BIC (oops):

$$\text{BIC}(\hat{\theta}) = -2 \log f(\mathbf{x}|\hat{\theta}) + K \log(n)\,,$$

Note that that $K$ is the number of parameters. In the one-dimensional problem, there are $3C - 1$ parameters; $-1$ because one of the $\pi$s need not be estimated.

```
# first finding the log-lielihood values for both models
x <- faithful$eruptions
loglike2 <- log.like(x, mu = erup2[[2]], sig2 = erup2[[3]], pis = erup2[[4]])
loglike3 <- log.like(x, mu = erup3[[2]], sig2 = erup3[[3]], pis = erup3[[4]])

# number of parameters
C <- 2
K2 <- 3*C - 1
C <- 3
K3 <- 3*C - 1
BIC.2 <- -2*loglike2 + K2*log(n)
BIC.3 <- -2*loglike3 + K3*log(n)

# comparing BICs
c(BIC.2, BIC.3)
```

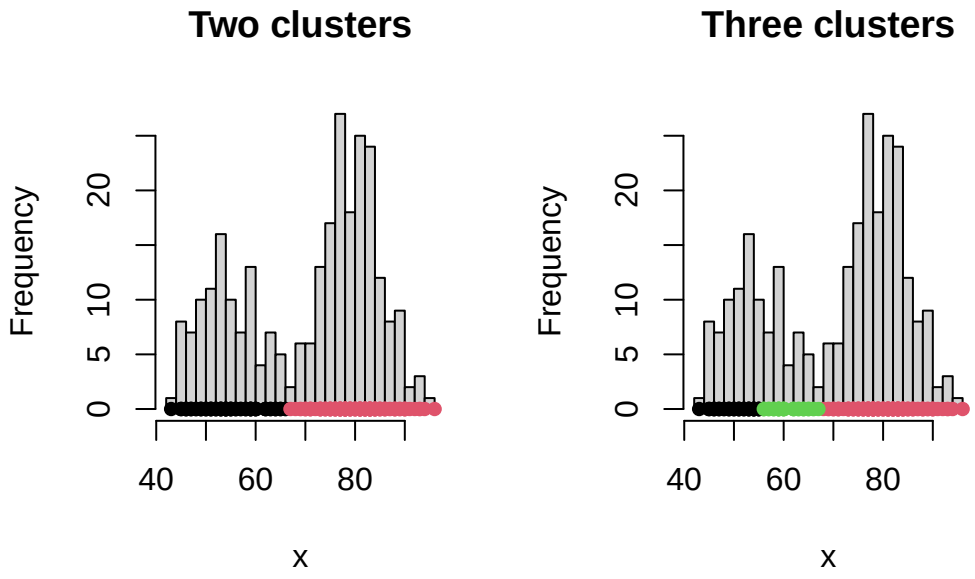
```
[1] 580.7491 580.6311
```

It looks like BIC for $C = 3$ is just marginally better. Since it is only just marginally better, we may choose either $C = 2$ or $C = 3$. (We need not always choose exactly the one that is largest). In this case, since $C = 2$ gives a simpler fit, and one that makes more sense, we may choose $C = 2$.

4. Repeat the above for the `waiting` component of `faithful` dataset.

   Let's go through the process again. Note that we have already set starting values in the `GMMoneDim` function so that they depend on the data, so we shouldn't need to change anything.

```
# Fitting the EM:
set.seed(1)
x <- faithful$waiting
erup2 <- GMMoneDim(x, C = 2)
erup3 <- GMMoneDim(x, C = 3)
n <- length(x)

par(mfrow = c(1,2))
hist(x, breaks = 30, main = "Two clusters", xlab = "x")
points(x, rep(0,n), col = erup2[[1]], pch = 16)
hist(x, breaks = 30, main = "Three clusters", xlab = "x")
points(x, rep(0,n), col = erup3[[1]], pch = 16)
```




```
loglike2 <- log.like(x, mu = erup2[[2]], sig2 = erup2[[3]], pis = erup2[[4]])
loglike3 <- log.like(x, mu = erup3[[2]], sig2 = erup3[[3]], pis = erup3[[4]])

# number of parameters
C <- 2
K2 <- 3*C - 1
C <- 3
K3 <- 3*C - 1
BIC.2 <- -2*loglike2 + K2*log(n)
BIC.3 <- -2*loglike3 + K3*log(n)
```

```r
# comparing BICs
c(BIC.2, BIC.3)
```

```
[1] 2096.033 2108.116
```

In this case, the BIC for $C = 2$ is clearly better, and we can confidently choose it in that case.

5. Now, we will implement EM algorithm for a general $p$-dimensional problem. Below is the theory for this. Go through the theory and the details carefully.

   Fundamentally, the steps are exactly the same, however, there are some additional nuances in the M-step and the practical implementations are important to consider.

   Suppose for some $C$ classes/clusters/components, we observe $X_1, \dots, X_n$ from a mixture of multivariate normal distributions. The joint distribution of the observed $\mathbf{x}$ is:

$$f(\mathbf{x}|\theta) = \sum_{c=1}^{C} \pi_c f_1(\mathbf{x}|\mu_c, \Sigma_c)$$

   where $\mu_c \in \mathbb{R}^p$, $\Sigma_c \in \mathbb{R}^{p\times p}$ and

$$f_c(\mathbf{x} \mid \mu_c, \sigma_c^2) = \left(\frac{1}{2\pi}\right)^{p/2} \frac{1}{|\Sigma_c|^{1/2}} \exp\left\{-\frac{(\mathbf{x}-\mu_c)^T \Sigma_c^{-1} (\mathbf{x}-\mu_c)}{2}\right\}.$$

   Similar to the one dimensional case, set up the EM algorithm for this $p$-dimensional case, and then implement this on the Old Faithful dataset.

   Introduce latent variable $Z_i$ such that

$$Z_i = c \text{ if } X_i \text{ is in class } c\,.$$

   The E-step is to find $q$:

$$q(\theta|\theta_{(k)}) = \mathbb{E}\left[\log f(\mathbf{x}, \mathbf{z}|\theta) \mid X = \mathbf{x}, \theta = \theta_{(k)}\right]$$

   Let's solve this a bit more:

$$
\begin{aligned}
q(\theta|\theta_{(k)}) &= \mathbb{E}_{Z|x}\left[\log f(x,z|\theta) \mid X=x,\theta_{(k)}\right] \\
&= \mathbb{E}_{Z|x}\left[\sum_{i=1}^{n}\log f(x_i,z_i|\theta) \mid X=x,\theta_{(k)}\right] \\
&= \sum_{i=1}^{n}\mathbb{E}_{Z_i|x_i}\left[\log f(x_i,z_i|\theta) \mid X=x_i,\theta_{(k)}\right] \\
&= \sum_{i=1}^{n}\sum_{c=1}^{C}\log\left\{f_c(x_i|\mu_c,\Sigma_c)\pi_c\right\}\ \frac{f_c(x_i|\mu_{c,k},\Sigma_{c,k})\pi_{c,k}}{\sum_{j=1,2}f_j(x_i|\mu_{j,k},\Sigma_{j,k})\pi_{j,k}}\,.
\end{aligned}
$$

For the E-step above, we need to find

$$
\Pr(Z=c|X=x_i,\theta=\theta_{(k)}) = \frac{f(x_i|Z=c,\theta=\theta_{(k)})\Pr(Z=c)}{f(x_i)} = \frac{f_c(x_i|\mu_{c,k},\Sigma_{c,k})\pi_{c,k}}{\sum_{j=1}^{C}f_j(x_i|\mu_{j,k},\Sigma_{j,k})\pi_{j,k}} := \gamma_{i,c,k}\,.
$$

And then in the M-step, we do the maximization step:

$$
q(\theta|\theta_{(k)}) = \text{const} - \frac{1}{2}\sum_{i=1}^{n}\sum_{c=1}^{C}\log|\Sigma_c|\gamma_{i,c,k} - \sum_{i=1}^{n}\sum_{c=1}^{C}\frac{(x_i-\mu_c)\Sigma_c^{-1}(x_i-\mu_c)^T}{2}\gamma_{i,c,k} + \sum_{i=1}^{n}\sum_{c=1}^{C}\log\pi_c\gamma_{i,c,k}\,.
$$

Taking derivative with respect to $\mu_c$

$$
\begin{aligned}
\frac{\partial q}{\partial\mu_c} &= -\sum_{i=1}^{n}\Sigma_c^{-1}(x_i-\mu_c)\gamma_{i,c,k} \overset{set}{=} 0 \\
\Rightarrow \mu_{c,(k+1)} &= \frac{\sum_{i=1}^{n}\gamma_{i,c,k}\ x_i}{\sum_{i=1}^{n}\gamma_{i,c,k}}
\end{aligned}
$$

Taking derivatives with respect to a matrix, $\Sigma_c$ is tricky. But the following properties will be useful to remember for compatible matrices $A, B, C$

- $\text{tr}(ACB) = \text{tr}(CAB) = \text{tr}(BC)$
- for scalar $x$, $x^TAx = \text{tr}(x^TAx) = \text{tr}(x^TxA)$
- $\dfrac{\partial}{\partial A}\text{tr}(AB) = B^T$
- $\dfrac{\partial}{\partial A}\log(|A|) = A^{-T}$

Using these properies and taking derivative w.r.t $\Sigma_c^{-1}$ (for convenience), we get

$$
\begin{aligned}
\frac{\partial q}{\partial \Sigma_c^{-1}} &= -\frac{n}{2}\Sigma_c \gamma_{i,,c,k} - \frac{\partial}{\partial \Sigma_c^{-1}}\left[\sum_{i=1}^{n} \frac{\text{tr}((x_i-\mu_c)^T(x_i-\mu_c)\Sigma_c^{-1})}{2}\gamma_{i,c,k}\right] \\
&= -\frac{n}{2}\Sigma_c \gamma_{i,,c,k} - \left[\sum_{i=1}^{n} \frac{(x_i-\mu_c)^T(x_i-\mu_c)\gamma_{i,c,k}}{2}\right] \stackrel{set}{=} 0 \\
\Rightarrow \Sigma_{c,(k+1)} &= \frac{\sum_{i=1}^{n} \gamma_{i,c,k}(x_i-\mu_{c,(k+1)})(x_i-\mu_{c,(k+1)})^T}{\sum_{i=1}^{n}\gamma_{i,c,k}}\,.
\end{aligned}
$$

The optimization for $\pi_c$ is the same as the univariate case. Hence, we get the following updates:

$$
\begin{aligned}
\mu_{c,(k+1)} &= \frac{\sum_{i=1}^{n} \gamma_{i,c,k}\ x_i}{\sum_{i=1}^{n}\gamma_{i,c,k}} \\
\Sigma_{c,(k+1)} &= \frac{\sum_{i=1}^{n} \gamma_{i,c,k}(x_i-\mu_{c,(k+1)})(x_i-\mu_{c,(k+1)})^T}{\sum_{i=1}^{n}\gamma_{i,c,k}} \\
\pi_{c,(k+1)} &= \frac{1}{n}\sum_{i=1}^{n}\gamma_{i,c,k}
\end{aligned}
$$

Here are some practical things to keep in mind:

- **Multivariate normal density**: Given a starting value, the algorithm updates the components using the above update equations. Notice that we need to evaluate the density of a multivariate normal. You can use `mvtnorm` package and `dmvnorm` function for that.
- **Local solutions**: Further note that the target log-likelihood is not concave. Thus, there will be multiple local maximas. The EM algorithm is an MM algorithm, so it does converge to a local maxima, but it need not converge to a global maxima. In order to try to ensure that we reach a reasonable maxima, it will be useful to start from multiple starting points and see where you converge. Then note down the log-likelihood and compare the log-likehood with other runs from different starting points. The optima with the largest log-likelihood should be the chosen maxima.
- **Starting values**: In addition to testing this on multiple starting values, one must also make sure the starting values are realistic. That is, if most of the data is centered near (50,100), then it is unreasonable to start at (0,0), and infact such starting values will lead to numerical inaccuracies. Additionally, if your starting values for the $\mu_c$ are the same for all $c$, then all normal distribution components are centered at the same spot and this leads to only one cluster, even if you have specified $C > 1$.
- **Lack of positive definiteness**: In the update for $\Sigma_c$, note that we are calculating a covariance matrix. Particularly when $C$ is chosen to be larger than the actual number of clusters, an iteration value of $\Sigma_{c,(k+1)}$ may yield singular matrices. To correct for this, a common solution is to set

$$
\Sigma_{c,(k+1)} = \frac{\sum_{i=1}^{n} \gamma_{i,c,k}(x_i-\mu_{c,(k+1)})(x_i-\mu_{c,(k+1)})^T}{\sum_{i=1}^{n}\gamma_{i,c,k}} + \text{diag}(\epsilon, p)\,.
$$

That is, we add a diagonal matrix with small entries $\epsilon$ in order to make sure it's positive definite.

- In order to declare convergence, we will monitor the change in the log-likelihood values.

6. **Having understood the challenged and the theoretical framework from before, your task is to write a function `GMMforD` that takes a matrix input and the number of clusters $C$ as arguments:**

```r
# function that calculates the
# the log_likelihood
# for this multivariate setup
library(mvtnorm)
log_like <- function(X, pi.list, mu.list, Sigma.list, C)
{
  foo <- 0
  for(c in 1:C)
  {
    foo <- foo + pi.list[[c]]*dmvnorm(X, mean = mu.list[[c]], sigma = Sigma.list[[c]])
  }
  return(sum(log(foo)))
}

# Now I recommend the following
# mu is a list
# Sigma is a list
GMMforD <- function(X, C = 2, tol = 1e-5, maxit = 1e3)
{
  n <- dim(X)[1]
  p <- dim(X)[2]

  ######## Starting values ###################
  ## pi are equally split over C
  pi.list <- rep(1/C, C)

  mu <- list()  # list of all the means
  Sigma <- list()  # list of all variances

  # The means for each C cannot be the same,
  # since then the three distributions overlap
  # Hence adding random noise to colMeans(X)
  # the variance of the noise depends on the components
  for(i in 1:C)
  {
     mu[[i]] <-  rnorm(p) + colMeans(X)
```

```
    Sigma[[i]] <- var(X) # same covariance matrix
  }
  # Choosing good starting values is important since
  # The GMM likelihood is not concave, so the algorithm
  # may converge to a local optima.

######## EM algorithm steps ###################

  iter <- 0
  diff <- 100
  epsilon <- 1e-05  # postive-def of Sigma

  Ep <- matrix(0, nrow = n, ncol = C)  # gamma_{i,c}

  current <- c(unlist(mu), unlist(Sigma), pi.list)
  while((diff > tol) && (iter < maxit) )
  {
    iter <- iter + 1
    previous <- current
    ## E step: find gammas
    for(c in 1:C)
    {
      Ep[ ,c] <- pi.list[c]*apply(X, 1, dmvnorm , mu[[c]], Sigma[[c]])
    }
    Ep <- Ep/rowSums(Ep)

    ### M-step
    pi.list <- colMeans(Ep)
    for(c in 1:C)
    {
      mu[[c]] <- colSums(Ep[ ,c] * X )/sum(Ep[,c])
    }

    for(c in 1:C)
    {
      foo <- 0
      for(i in 1:n)
      {
        foo <- foo + (X[i, ] - mu[[c]]) %*% t(X[i, ] - mu[[c]]) * Ep[i,c]
      }
      Sigma[[c]] <- foo/sum(Ep[,c])
```

```
      if(min(eigen(Sigma[[c]])$values) <=0)
      {
        # To ensure the estimator is positive definite
        # otherwise next iteration gamma_i,c,k cannot be calculated
        Sigma[[c]] <- Sigma[[c]] + diag(epsilon, p)
        print("Matrix not positive-definite")
      }
    }

    # Difference in the log-likelihoods as the difference criterion
    current <- c(unlist(mu), unlist(Sigma), pi.list)
    diff <- norm(previous - current, "2")
  }

  # Final allocation updates
  for(c in 1:C)
  {
    Ep[ ,c] <- pi.list[c]*apply(X, 1, dmvnorm , mu[[c]], Sigma[[c]])
  }
  Ep <- Ep/rowSums(Ep)

  # calculate the loglikelihood of the final est
  save.loglike <- log_like(X = X, pi.list = pi.list, mu.list = mu, Sigma.list = Sigma,  C = C)

  return(list("pi" = pi.list, "mu" = mu, "Sigma" = Sigma, "Ep" = Ep,
              "log.like" = save.loglike))
}
```

7. **Run `GMMforD` for the bivariate Faithful dataset for $C = 2$. Plot the data, and color-code the clusters.**

```
X <- as.matrix(faithful)

C <- 2
class2.1 <- GMMforD(X = X, C = C)
class2.2 <- GMMforD(X = X, C = C)
class2.3 <- GMMforD(X = X, C = C)
class2.4 <- GMMforD(X = X, C = C)
```

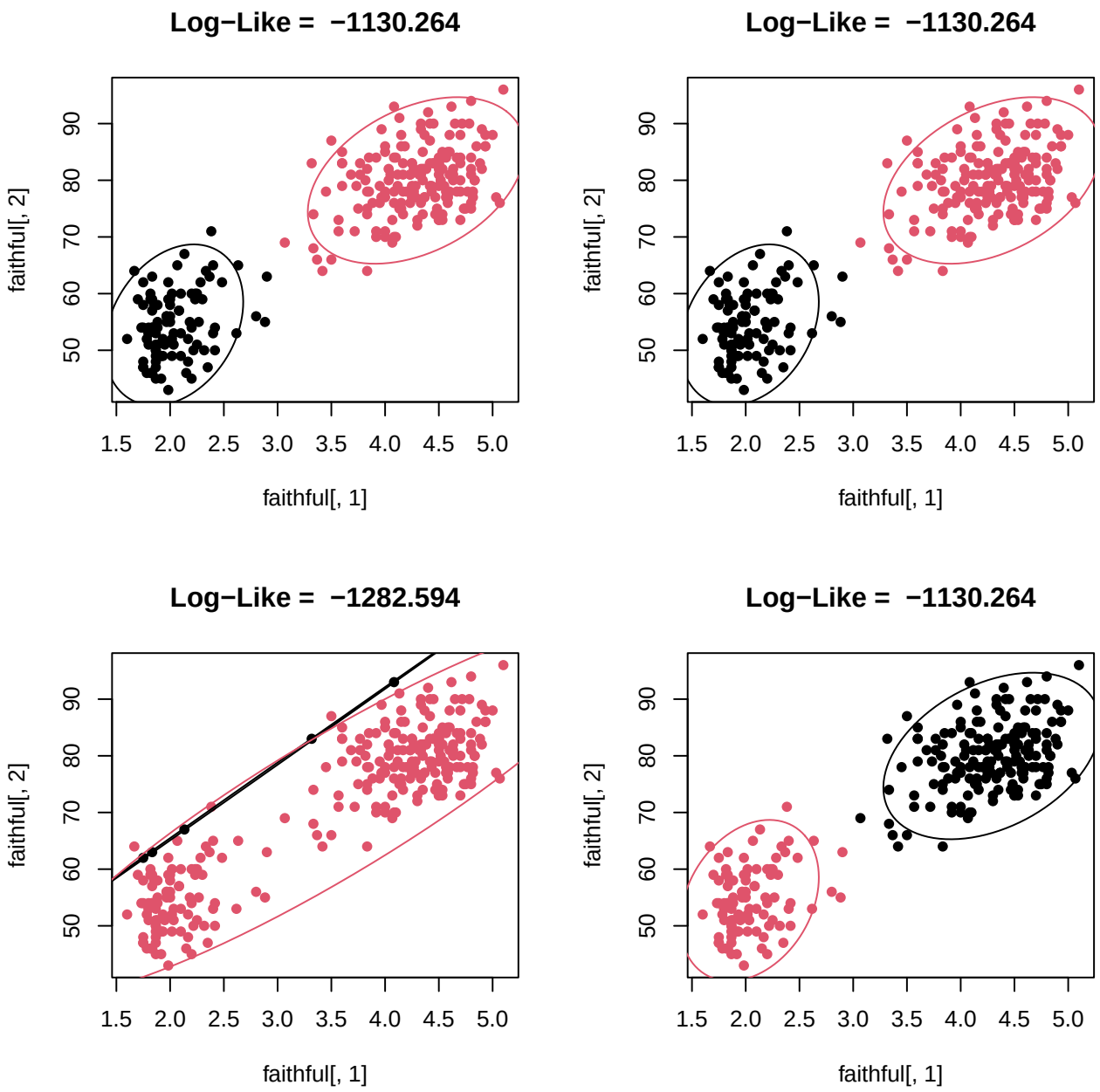
We will 4 different models to allow for the EM algorithm to converge to multiple local modes. Remember, that the GMM likelihood can be highly non-concave. Now we will color-code the data into the clusters we've found.

```
library(ellipse)
```

```
Attaching package: 'ellipse'
```

```
The following object is masked from 'package:graphics':

    pairs
```

```
# making 4 plots, one for each fit
par(mfrow = c(2,2))
for(i in 1:4)
{
  # get() allows to access the 4 different models
  model <- get(paste("class2.",i, sep = ""))

  # find which cluster is assigned to which data point
  allot <- apply(model$Ep, 1, which.max)

  # plot the data
  plot(faithful[,1], faithful[,2], col = allot, pch = 16,
       main = paste("Log-Like = ", round(model$log.like,3))) # plot allotment

  # make an ellipse for each cluster, to see the shape of clusters.
  ell <- list()
  for(c in 1:C)
  {
    ell[[c]] <- ellipse(model$Sigma[[c]], centre = as.numeric(model$mu[[c]]))
    lines(ell[[c]], col = c)
  }
}
```




The third model is different from the other 4, as we can clearly see the clsuters are different, and the log-likelihood value is must lower. Thus, models: `class2.1`, `class2.2`, `class2.4` are the same models, and we may choose any of these three as the chosen one.

8. **Use BIC to obtain the best value of $C$ among $2, 3, 4$ for this dataset.**

   First, let's make a function to find the number of parameters for a given cluster. There are $(C-1)$ of the $\pi$s, $Cp$ of the $\mu$s, and $\Sigma$ are $p \times p$ covariance matrices, with $p(p+1)/2$ number of parameters in each cluster.

```
Kpar <- function(C, p)
{
  (C-1) + C*p + p*(p+1)/2*C
}
```

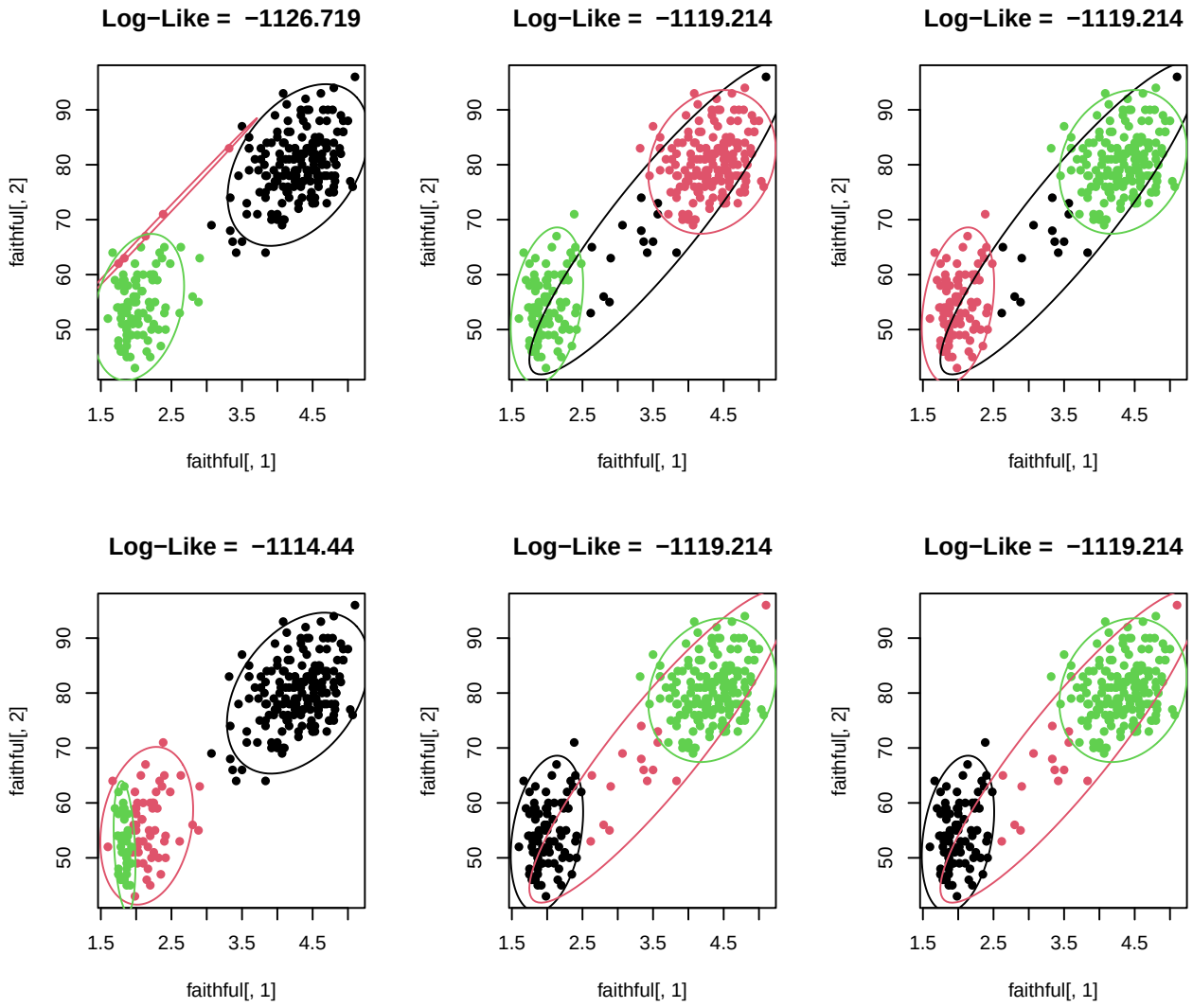
Next, we have to fit $C = 3, 4$ as well, with different starting values.

First, we fit $C = 3$:

```
class3.1 <- GMMforD(X = X, C = 3)
class3.2 <- GMMforD(X = X, C = 3)
class3.3 <- GMMforD(X = X, C = 3)
class3.4 <- GMMforD(X = X, C = 3)
class3.5 <- GMMforD(X = X, C = 3)
class3.6 <- GMMforD(X = X, C = 3)
```

Plotting them:

```
C <- 3
par(mfrow = c(2,3))
for(i in 1:6)
{
  model <- get(paste("class3.",i, sep = ""))
  allot <- apply(model$Ep, 1, which.max)  ## Final allotment of classification
  plot(faithful[,1], faithful[,2], col = allot, pch = 16,
       main = paste("Log-Like = ", round(model$log.like,3))) # plot allotment

  ell <- list()
  for(c in 1:C)
  {
    ell[[c]] <- ellipse(model$Sigma[[c]], centre = as.numeric(model$mu[[c]]))
    lines(ell[[c]], col = c)
  }
}
```



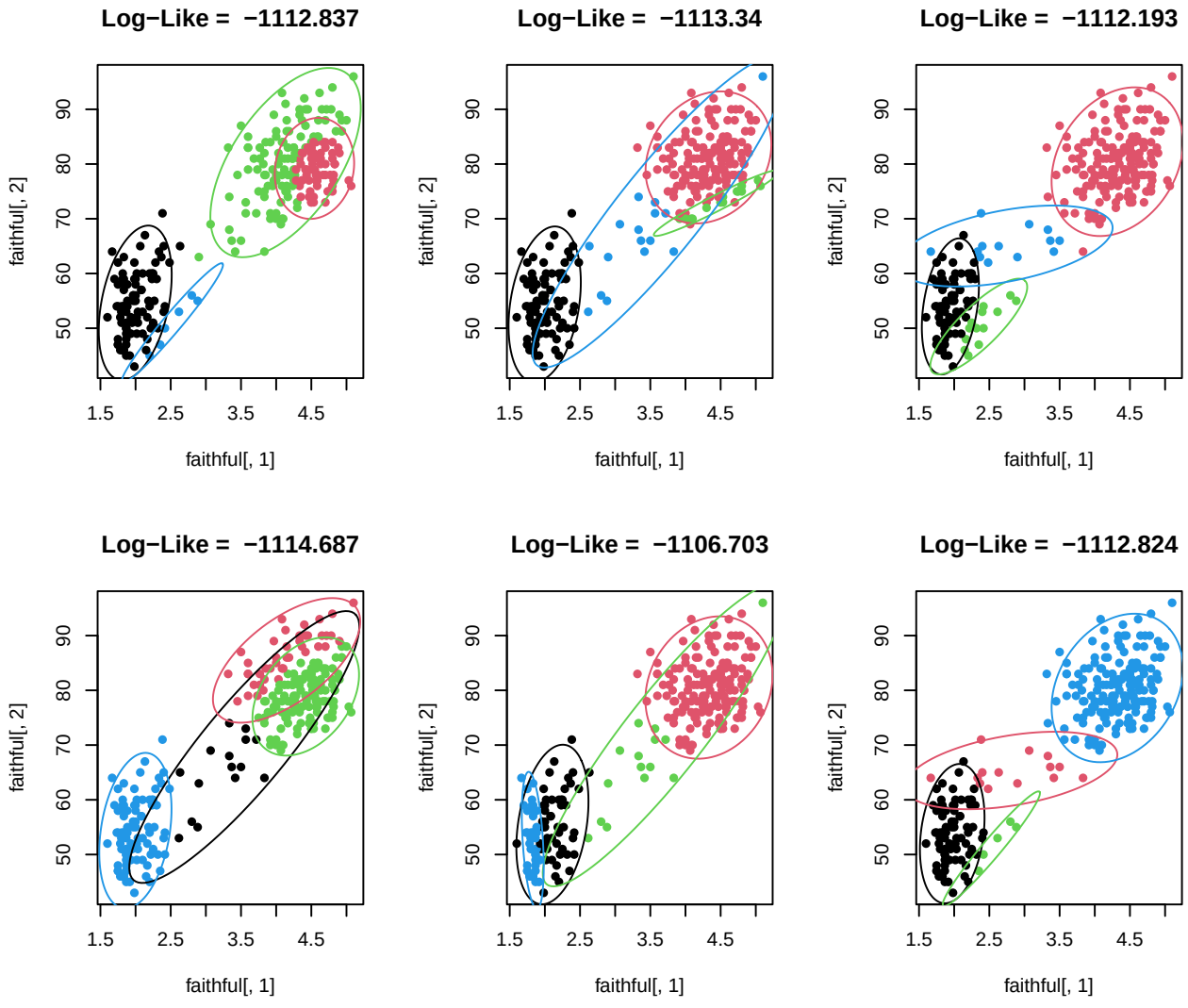

Model 4 in the above has the best log-likelihood, so that will be the chosen model. Now fitting for $C = 4$.

```
class4.1 <- GMMforD(X = X, C = 4)
class4.2 <- GMMforD(X = X, C = 4)
class4.3 <- GMMforD(X = X, C = 4)
class4.4 <- GMMforD(X = X, C = 4)
class4.5 <- GMMforD(X = X, C = 4)
class4.6 <- GMMforD(X = X, C = 4)
```

Again, we can look at the plot:

```
C <- 4
par(mfrow = c(2,3))
for(i in 1:6)
{
```

```
  model <- get(paste("class4.",i, sep = ""))
  allot <- apply(model$Ep, 1, which.max)  ## Final allotment of classification
  plot(faithful[,1], faithful[,2], col = allot, pch = 16,
       main = paste("Log-Like = ", round(model$log.like,3))) # plot allotment

  ell <- list()
  for(c in 1:C)
  {
    ell[[c]] <- ellipse(model$Sigma[[c]], centre = as.numeric(model$mu[[c]]))
    lines(ell[[c]], col = c)
  }
}
```




Model 5 is the largest log-likelihood in the above. Note that there is no guarantee that the solution we have converged to is the global solution. So Model 5 could still be a local maxima, and there is another

model out there that is better. We may run the function a few more times to see if the model converges somewhere better, but for now I've stopped at 6 different runs. Now we are ready to compare BICs to see which $C$ is better:

```
BIC.2 <- -2*class2.1$log.like + log(n)*Kpar(C = 2, p = 2)
BIC.3 <- -2*class3.4$log.like + log(n)*Kpar(C = 3, p = 2)
BIC.4 <- -2*class4.5$log.like + log(n)*Kpar(C = 4, p = 2)

c(BIC.2, BIC.3, BIC.4)
```

```
[1] 2322.192 2324.178 2342.340
```

Of the above, $C = 2$ has the best BIC value, and thus we will choose two clusters in this dataset.