EM algorithm for missing data in multivariate normal data

Richard Wilkinson

26 April 2017

We will now implement the EM algorithm for the case where we have missing data in a multivariate normal distribution. We will work with fake data that we generate ourselves, so that we can test how we the approach works. Let's begin by generating 500 observations from a k dimensional MVN distribution with mean $\mu = (1, 2, \ldots, k)$ with covariance matrix

$$\Sigma_{ij} = \sqrt{ij}\left(1 - \frac{i-j}{k}\right)$$

which says let the variance increase as we go through the dimensions, and let adjacent values be most closely related. See data plot.

```
k <- 4  # dim y
mu <- seq(1:k)

Sigma <- matrix(nr=k,nc=k)
for(i in 1:k){
  for(j in 1:k){
    Sigma[i,j] = (1-abs(i-j)/k)*sqrt(i*j)
  }
}
Sigma
```

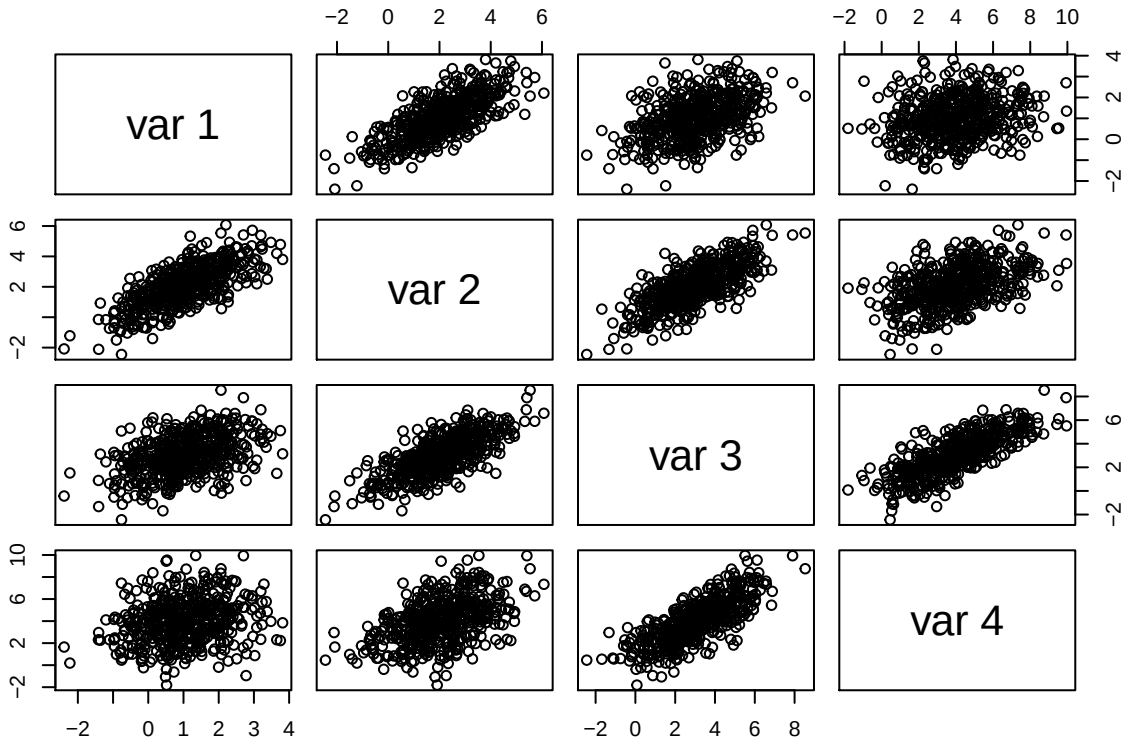
```
##           [,1]     [,2]      [,3]     [,4]
## [1,] 1.0000000 1.060660 0.8660254 0.500000
## [2,] 1.0606602 2.000000 1.8371173 1.414214
## [3,] 0.8660254 1.837117 3.0000000 2.598076
## [4,] 0.5000000 1.414214 2.5980762 4.000000
```

```
cov2cor(Sigma)
```

```
##      [,1] [,2] [,3] [,4]
## [1,] 1.00 0.75 0.50 0.25
## [2,] 0.75 1.00 0.75 0.50
## [3,] 0.50 0.75 1.00 0.75
## [4,] 0.25 0.50 0.75 1.00
```

To generate data, we can use the rmvnorm package.

```
library(mvtnorm)
nsamps <- 500
Y <- rmvnorm(n = nsamps, mean=mu, sigma = Sigma)
pairs(Y)
```



To get an idea of what the mean and covriance matrix would be if we estimated them using the full data, let's look at

```
ybar <- apply(Y,2, mean)
#Ycent <- sweep(Y, 2, ybar, '-') # remiove the mean from each Y
#1/nsamps *t(Ycent)%*%Ycent # for the MLE we divide by n not n-1
cov(Y)*(nsamps-1)/nsamps
```

```
##          [,1]     [,2]     [,3]     [,4]
## [1,] 1.046583 1.026421 0.745278 0.420459
## [2,] 1.026421 1.875073 1.694474 1.320436
## [3,] 0.745278 1.694474 2.835715 2.615298
## [4,] 0.420459 1.320436 2.615298 4.030878
```

```
1/nsamps* t(Y)%*%Y - ybar%*%t(ybar) # check it agrees with cov(Y)
```

```
##          [,1]     [,2]     [,3]     [,4]
## [1,] 1.046583 1.026421 0.745278 0.420459
## [2,] 1.026421 1.875073 1.694474 1.320436
## [3,] 0.745278 1.694474 2.835715 2.615298
## [4,] 0.420459 1.320436 2.615298 4.030878
```

Now finally let's remove some data to give us a missing-dataset.

```
missingfreq <- 0.3
M <- matrix(as.logical(rbinom(nsamps*k,1,missingfreq)), nr=nsamps)
Ymis <- Y
Ymis[M] <- NA
```

EM algorithm

We will now only work with the missing data. We need to implement functions to calculate the expected values described in section 6.5 of the notes. To begin with, let's calculate

$$E(y|y_{obs}, \theta)$$

```
expectY <- function(y, mu, Sigma){
  #
  # function to calculate
  # E(y|y_obs, theta^{m})
  #
  miss.pat <- is.na(y) # find pattern of missingness
  if(sum(miss.pat) ==0) return(y) ## i.e. no missing data,
                      #so just return observation
  if(sum(miss.pat)==k) return(mu) ## both observations missing, so return mean
  else{
    # only if we have a mixture of observed and unobserved data do we need the conditional
    # Gaussian formalue
    tmp <- y
    tmp[is.na(tmp)] <- mu[miss.pat] + Sigma[miss.pat,!miss.pat]%*%
      solve(Sigma[!miss.pat, !miss.pat])%*%(y[!miss.pat]- mu[!miss.pat])
    return(tmp)
  }
}
```

This function only works on a single observation at a time. We can then loop over all the rows in the data matrix using

```
Yexp <- t(apply(Ymis,1, expectY, mu=mu, Sigma=Sigma))
```

Next, we need a function to calculate the variance. We can do this as follows

```
varY <- function(y, mu, Sigma){
  #
  # function to calculate
  # V(y | y_obs, theta)
  #

  miss.pat <- is.na(y) # find pattern of missingness
  if(sum(miss.pat) ==0) return(matrix(rep(0,k^2), nr=k)) ## i.e. no missing data,
                      #### so just return observation
  if(sum(miss.pat)==k) return(Sigma) ## both observations missing, so return mean
  else{
    # only if we have a mixture of observed and unobserved data do we need the conditional
    # Gaussian formalue
    tmp <- matrix(rep(0,k^2),nr=k)
    tmp[miss.pat, miss.pat] <-  Sigma[miss.pat, miss.pat] - Sigma[miss.pat,!miss.pat]%*%
      solve(Sigma[!miss.pat, !miss.pat]) %*%Sigma[!miss.pat, miss.pat]
    return(tmp)
  }
}
```

Then we calculate calculate

$$E(yy^\top|y_{obs}, \theta)$$

using

```
expYYt <- function(y, mu, Sigma){
  #
  # this function calculate E(y y^t | y_obs, theta)
  #

  tmp <- expectY(y,mu,Sigma)
  varY(y,mu, Sigma) + tmp%*%t(tmp)
}
```

and again, we need to loop over all the rows of Ymis

```
S <- matrix(nr=k,rep(0,k^2))
for(i in 1:nsamps){
  S <- S+expYYt(Ymis[i,], mu=mu, Sigma=Sigma)
}
```

Finally, we need to loop over all of these steps until convergence. We will use available-case analysis to estimate initial values of the parameters.

```
EM <- function(Ymis){
  k = dim(Ymis)[2]
  n=dim(Ymis)[1]
  mu0 <- apply(Ymis,2, mean, na.rm=T)  # available-case estimate of the mean
  Sigma0 <- cov(Ymis, use='pairwise.complete.obs')   # available-case estimate of the variance
  mu.list <- list()
  mu.list[[1]] <- mu0

  Sigma.list <- list()
  Sigma.list[[1]]<- Sigma0

  change <-1

  i<-1
  while(change>10^-6 && i<=500){
    Yexp <- t(apply(Ymis,1, expectY, mu=mu.list[[i]], Sigma=Sigma.list[[i]]))
    mu.list[[i+1]] <- apply(Yexp,2,mean)
    S <- matrix(nr=k,rep(0,k^2))
    for(j in 1:n){
      S <- S+expYYt(Yexp[j,], mu=mu.list[[i+1]], Sigma=Sigma.list[[i]])
    }
    Sigma.list[[i+1]] <- S/n - mu.list[[i+1]]%*%t(mu.list[[i+1]])

    change <- max(max(abs(mu.list[[i+1]]-mu.list[[i]])), max(abs(Sigma.list[[i+1]]-Sigma.list[[i]])))
    i<-i+1
  }

  return(list(mu=mu.list[[i]], Sigma = Sigma.list[[i]]))
}
```

Finally, we can run the algorithm.

```
out <- EM(Ymis)
out
```

```
## $mu
## [1] 0.9706331 2.1042672 3.0375640 3.9360238
```

```
##
## $Sigma
##           [,1]      [,2]      [,3]      [,4]
## [1,] 0.8534720 0.9903538 0.7031473 0.3522325
## [2,] 0.9903538 1.7711936 1.7906137 1.3312792
## [3,] 0.7031473 1.7906137 2.7749341 2.6999195
## [4,] 0.3522325 1.3312792 2.6999195 3.5746964
```

We can compare these with the true values

```
mu
```

```
## [1] 1 2 3 4
```

```
Sigma
```

```
##           [,1]     [,2]      [,3]     [,4]
## [1,] 1.0000000 1.060660 0.8660254 0.500000
## [2,] 1.0606602 2.000000 1.8371173 1.414214
## [3,] 0.8660254 1.837117 3.0000000 2.598076
## [4,] 0.5000000 1.414214 2.5980762 4.000000
```

Complete case analysis

Complete case analysis works here by deleting any y_i which has a single missing entry. If the rate of missingness is large compared to k, then we can find we have very few cases to work with.

```
Ycomplete <- Ymis[complete.cases(Ymis),]
colMeans(Ycomplete)
```

```
## [1] 0.9214329 2.0165419 2.9712046 3.9533356
```

```
cov(Ycomplete)
```

```
##           [,1]      [,2]      [,3]      [,4]
## [1,] 0.9805785 0.9691321 0.6982316 0.3484698
## [2,] 0.9691321 1.8354219 1.7403658 1.3051399
## [3,] 0.6982316 1.7403658 2.7910236 2.4656986
## [4,] 0.3484698 1.3051399 2.4656986 3.8392444
```

```
print(paste('Complete case analysis is based on only', dim(Ycomplete)[1],'samples!'))
```

```
## [1] "Complete case analysis is based on only 104 samples!"
```

Available case analysis works reasonably well in this case.

```
apply(Ymis,2, mean, na.rm=T)
```

```
## [1] 0.9767894 2.0814951 3.0882066 3.9409223
```

```
cov(Ymis, use='pairwise.complete.obs')
```

```
##           [,1]      [,2]      [,3]      [,4]
## [1,] 0.9679472 0.9487884 0.7312128 0.3279645
## [2,] 0.9487884 1.8588385 1.8351488 1.2528432
## [3,] 0.7312128 1.8351488 3.0383503 2.7640900
## [4,] 0.3279645 1.2528432 2.7640900 4.0488058
```