Statistical Computing with R

Lecture 11: mixture models; the EM algorithm; how to deal with repetitive and time-consuming computations; `replicate( )`; parallelization and the `foreach` package

Mirko Signorelli
🏠: mirkosignorelli.github.io
✉: statcompr [at] gmail.com

Mathematical Institute
Leiden University

Master in Statistics and Data Science (2023-2024)

Announcements

- Third assignment will be published tomorrow (Nov. 25)
- Plan for the rest of the course
  1. new contents up to lecture 13
  2. lecture 14: more info about exam + question time + practice exam

Recap

Lecture 10:

- numeric optimization
- maximum likelihood
- confidence intervals
- hypothesis testing

Today:

- mixture models
- the EM algorithm
- dealing with repetitive and time-consuming computations
- `replicate( )`
- EM algorithm + `replicate`
- parallelization and the `foreach` package (*self-study*)

Mixture models

The EM algorithm

Parameter estimates, convervenge, classification and label switching

How to deal with repetitive and time-consuming computations

`replicate( )`

EM algorithm + `replicate( )`

parallelization and the `foreach` package (SELF-STUDY)

Mixture distribution

- A r.v. X follows a mixture distribution if

$$f_X(x) = \sum_{j=1}^{K} \pi_j f_{X_j}(x) = \pi_1 f_{X_1}(x) + \pi_2 f_{X_2}(x) + \ldots + \pi_K f_{X_K}(x),$$

  or, equivalently:

$$F_X(x) = \sum_{j=1}^{K} \pi_j F_{X_j}(x) = \pi_1 F_{X_1}(x) + \pi_2 F_{X_2}(x) + \ldots + \pi_K F_{X_K}(x),$$

  where:

1. $X_1, \ldots, X_K$ are K random variables, each with different f_{X_j} / F_{X_j}, called components of the mixture
2. $\pi_j > 0 \; \forall j = 1, \ldots, K$ and $\sum_{j=1}^{K} \pi_j = 1$
3. $\pi_1, \ldots, \pi_K$ are called mixing proportions or mixture weights. π_j represents the probability that a draw from $f_X(x)$ is picked from the j-th component X_j

Example: height

- Let S_i denote the sex[1] of subject i, and H_i their height in meters
- Suppose that

$$H_i | S_i = \text{female} \sim N(\mu = 1.69, \sigma = 0.04)$$
$$H_i | S_i = \text{male} \sim N(\mu = 1.83, \sigma = 0.06)$$

and assume, for simplicity, that $S_i \sim \text{Bernoulli}(\pi_F = 0.5)$.

- Then, the marginal distribution of H is a mixture of two normal distributions

$$f_H(x) = \pi_F f_{H|F}(x) + (1 - \pi_F) f_{H|M}(x)$$

[1] "Sex" versus "gender": what is the difference?

Example: height (cont'd)

- Let's simulate heights in a population of 100000 individuals:

```r
set.seed(2)
n = 1e5; height = rep(NA, n)
sex = sample(1:2, size = n, replace = T)
table(sex)
```

```
## sex
##     1     2
## 49909 50091
```

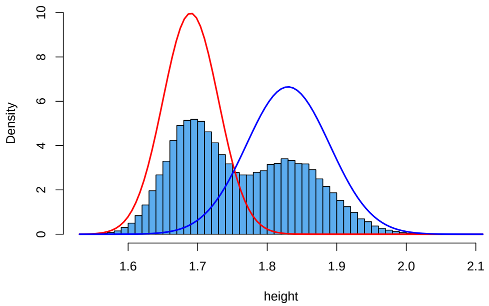
```r
# let's code 1 = female and 2 = male
n.female = sum(sex == 1)
height[which(sex == 1)] = rnorm(n = n.female,
                               mean = 1.69, sd = 0.04)
height[which(sex == 2)] = rnorm(n = n - n.female,
                               mean = 1.83, sd = 0.06)
summary(height)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.532   1.689   1.746   1.760   1.830   2.080
```

Example: height (cont'd)

- How does the density of H look like?

```r
hist(height, 50, prob = T, main = '',
     col = 'steelblue2', ylim = c(0, 10))
curve(dnorm(x, mean = 1.69, sd = 0.04),
      add = T, col = 'red', lwd = 2)
curve(dnorm(x, mean = 1.83, sd = 0.06),
      add = T, col = 'blue', lwd = 2)
```



Mixture models

The EM algorithm

Parameter estimates, convervenge, classification and label switching

How to deal with repetitive and time-consuming computations

`replicate( )`

EM algorithm + `replicate( )`

parallelization and the `foreach` package (SELF-STUDY)

Motivation

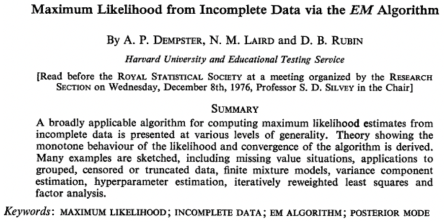
- In Lecture 10 we discussed maximum likelihood estimation when $X_1, ..., X_n$ are i.i.d. realizations from a single density $f_X(x)$
- Today: how can we perform maximum likelihood estimation for more complex problems, for example when $(x_1, ..., x_n)$ comes from a mixture model?
- A popular solution: use the EM algorithm!

The EM algorithm

- The Expectation-Maximization (EM) algorithm is a widely-used, multi-purpose optimization algorithm
- Typically applied to solve problems involving “incomplete” data:
    - latent (unobserved and unobservable) variable models $\rightarrow$ mixture models and model-based clustering, mixed-effects models, ...
    - missing data problems
- We are going to implement the EM algorithm to estimate mixture models

The EM algorithm (cont'd)

- Method first formalized by Dempster et al. (1977) (but the idea was already around in the 50s!):



Maximum Likelihood from Incomplete Data via the *EM* Algorithm

By A. P. Dempster, N. M. Laird and D. B. Rubin

Harvard University and Educational Testing Service

[Read before the Royal Statistical Society at a meeting organized by the Research Section on Wednesday, December 8th, 1976, Professor S. D. Silvey in the Chair]

Summary

A broadly applicable algorithm for computing maximum likelihood estimates from incomplete data is presented at various levels of generality. Theory showing the monotone behaviour of the likelihood and convergence of the algorithm is derived. Many examples are sketched, including missing value situations, applications to grouped, censored or truncated data, finite mixture models, variance component estimation, hyperparameter estimation, iteratively reweighted least squares and factor analysis.

Keywords: maximum likelihood; incomplete data; em algorithm; posterior mode

- **Q**: this sounds like pretty old stuff! Does a statistician nowadays ever need to implement an EM algorithm?!?
- **A**: Yes. The EM is still widely used[2] nowadays!

[2]Here's an example from my own research: Model-based clustering for populations of networks, but you can find plenty of recent publications where the EM is used

General formulation of the EM algorithm

- Let $\ell(\theta|x,z) = \log f(X,Z|\theta)$ be a log-likelihood that depends both on observed data x and on unobserved data z
- The EM algorithm iterates between two types of steps:
  1. E step: compute the conditional expectation of your objective function (here: ℓ) given the data and current (= latest) parameter estimates $\theta^{(t-1)}$:
  $$E_Z[\log f(Z|X,\theta^{(t-1)})]$$
  2. M step: update your parameter estimates by maximizing the conditional expectation obtained in the E step:
  $$\theta^{(t)} = \text{argmax}_\theta E_Z[\log f(Z|X,\theta^{(t-1)})]$$
- These two steps may sound very abstract for now. However, they become clearer and more concrete when you get to implement them on a mixture model! (Just wait for a few slides ☺)

Let's formalize the problem

Define the following quantities:

- K = number of components in the mixture model
- Z_i: a latent (unobserved) variable that takes value $z_i = j \in \{1, ..., K\}$ if subject i comes from component j
- a matrix $\hat{P}^{(t)}$ whose elements contain the estimated probability memberships at iteration t: $\hat{p}_{ij}^{(t)} = P(Z_i = j \text{ at iteration } t)$. Notice that $\sum_{j=1}^{K} \hat{p}_{ij}^{(t)} = 1 \, \forall i \in \{1, ..., n\}$!
- $\hat{\pi}_j^{(t)}$ = estimate of π_j at iteration t, $j = 1, ..., K$
- θ_j = parameters of the j-th component of the mixture $f_{X_j}(x) = f_{X_j}(x; \theta_j)$, $j = 1, ..., K$
- $\hat{\theta}_j^{(t)}$ = estimate of θ_j at iteration t, $j = 1, ..., K$

How to implement the EM algorithm for a mixture model

- Iteration 1 (algorithm initialization):
    1. pick a random starting value for $\hat{\pi}^{(1)} = (\hat{\pi}_1^{(1)}, ..., \hat{\pi}_K^{(1)})$
    2. first "E step": draw random initial probability memberships $\hat{P}^{(1)}$ based on $\hat{\pi}^{(1)}$. Tip: you want to have $E(P_{ij}^{(1)}) = \pi_j \, \forall j \in \{1, .., K\}$!
    3. first M step: find $\hat{\theta}^{(1)}$ that maximizes the conditional log-likelihood that uses $\hat{P}^{(1)}$ and $\hat{\pi}_1^{(1)}$
- For $t = 2, 3, ...$:
    1. E step: conditionally on $\hat{\theta}^{(t-1)}$ and $\hat{\pi}^{(t-1)}$, compute new estimates for $\hat{P}^{(t)}$
    2. M step: conditionally on $\hat{P}^{(t)}$, compute new estimates for $\hat{\pi}^{(t)}$ and $\hat{\theta}^{(t)}$
- This iterative algorithm will create a sequence of estimates

$$\hat{\pi}^{(1)} \xrightarrow{E_1} \hat{P}^{(1)} \xrightarrow{M_1} \hat{\theta}^{(1)} \xrightarrow{E_2} \hat{P}^{(2)} \xrightarrow{M_2} \hat{\theta}^{(2)}, \hat{\pi}^{(2)} \xrightarrow{E_3} \hat{P}^{(3)} \xrightarrow{M_3} \hat{\theta}^{(3)}, \hat{\pi}^{(3)} \xrightarrow{E_4} ...$$

Stopping criteria

- In principle, you could iterate the algorithm as much as you want
- However, the idea is that at some point the algorithm will converge to a local maximum (and maybe even to the global one)
- So, when to stop? This is usually determined through one or more stopping criteria, for example:
  1. max number of iterations reached
  2. stop at $t : |\hat{\theta}^{(t)} - \hat{\theta}^{(t-1)}| < \varepsilon$
  3. stop at $t : |\ell^{(t)} - \ell^{(t-1)}| < \varepsilon$
  4. ... and more
- In the example that follows, I will simply use (1) and stop after 200 iterations, when I am sure that the EM has converged (I will show you how to check convergence later). Notice that this is not always the case!

Example: mixture of two normals

- Let's generate data from a mixture of $X_1 \sim N(\mu_1 = 1, \sigma_1 = 1)$ and $X_2 \sim N(\mu_2 = 1, \sigma_2 = 1)$, with $\pi_1 = 0.3$:

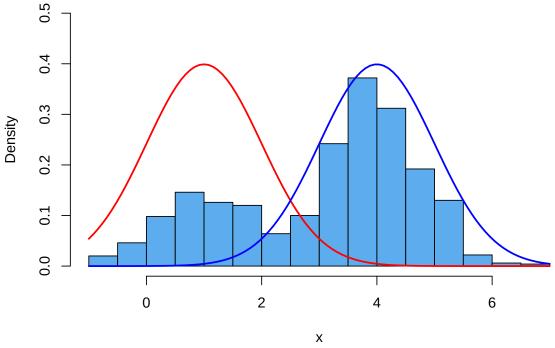
```r
set.seed(13)
n = 1000; pi1 = 0.3; mu1 = 1; mu2 = 4; sigma = 1
group = sample(1:2, n, replace = T, prob = c(pi1, 1-pi1))
table(group)
```

```
## group
##   1   2
## 303 697
```

```r
x = rep(NA, n)
x[group == 1] = rnorm(sum(group == 1), mu1, sigma)
x[group == 2] = rnorm(sum(group == 2), mu2, sigma)
```

Example: mixture of two normals (cont'd)

```r
hist(x, 20, prob = T, main = '',
     col = 'steelblue2', ylim = c(0, 0.5))
curve(dnorm(x, mu1, sigma), add = T, col = 'red', lwd = 2)
curve(dnorm(x, mu2, sigma), add = T, col = 'blue', lwd = 2)
```



Our goal

- For now, let's assume that $\sigma_1 = \sigma_2 = 1$ are known for simplicity (things are already complicated enough!). In the coding session, we will consider the more general case where σ_1 and σ_2 are also unknown and need to be estimated
- We observe `x` (= known), but we do not observe `group` (= latent / unknown)
- Our goal is to estimate:
  1. the unknown parameters $\theta = (\mu_1, \mu_2)$ and $\pi = (\pi_1, \pi_2)$
  2. the cluster membership probabilities $p_{ij} = P(Z_i = j)$, $i \in \{1, ..., n\}$, $j = \{1, 2\}$
- Notice that here $\pi_2 = 1 - \pi_1$ and $p_{i2} = 1 - p_{i1}\ \forall i$, so we actually need to estimate just (μ_1, μ_2), π_1 and p_{i1}

EM algorithm implementation

- Let's first initialize a few objects where we will store our estimates iteration after iteration:

```
n.iter = 200
# pi1hat: vector where 1 value = 1 iteration of the EM algorithm
pi1hat = rep(NA, n.iter) # estimates of \pi_1
# p1hat and muhat: matrices where 1 row = 1 iteration of the EM
p1hat = matrix(NA, n.iter, n) # estimates of Pr(Z_i = 1)
muhat = matrix(NA, n.iter, 2) # ML estimates of mu
```

- Let's set $\pi_1^{(1)} = 0.45$:

```
pi1hat[1] = 0.45
```

- E step, iteration 1: draw random $\hat{p}_{ij}^{(1)}$ while ensuring that $E(P_{ij}^{(1)}) = \pi_j \, \forall j \in \{1, .., K\}$

```
p1hat[1, ] = runif(n, pi1hat[1] - 0.35, pi1hat[1] + 0.35)
```

Log-likelihood function for $\theta = (\mu_1, \mu_2)$

- In theory, in the M step we would need to estimate both $\theta = (\mu_1, \mu_2)$ and π_1 numerically. However, for π_1 the problem can be solved analytically - we will see that in a few slides
- For θ, we need to maximize numerically the following negative log-likelihood[3]

```
neg.logl = function(theta, pi1, w1, x) {
  mu1 = theta[1]
  mu2 = theta[2]
  # density of the mixture model:
  f.x1 = pi1 * dnorm(x, mu1, sd = 1)
  f.x2 = (1-pi1) * dnorm(x, mu2, sd = 1)
  # negative log-likelihood:
  - sum(w1 * log(f.x1) + (1 - w1) * log(f.x2))
}
```

```
# let's check that the function works:
neg.logl(c(0.5, 2), pi1hat[1], p1hat[1,], x)
```

```
## [1] 4879.9
```

[3]this likelihood is commonly used with mixture models. You are not expected to know how to derive it - you can just take it as a given ☺

Maximization (M) step for θ

- M step, iteration 1: obtain the parameter estimates $\hat{\theta}^{(1)}$ conditionally on the starting values $\hat{\pi}_1^{(1)}$ and $\hat{p}_{11}^{(1)}, ..., \hat{p}_{n1}^{(1)}$:

```
muhat[1, ] = optim(c(2, 3), function(theta)
        neg.logl(theta, pi1hat[1], p1hat[1,], x))$par
```

⚠ M step = maximum likelihood estimation (see Lecture 10) ⚠

- 2 unknown parameters $\theta = (\mu_1, \mu_2) \Rightarrow$ use `optim( )`!
- $\theta = (\mu_1, \mu_2) \in \mathbb{R}^2 \Rightarrow$ unconstrained problem $\Rightarrow$ we can use `method = "Nelder-Mead"` (default)
- `par = c(2, 3)` is an arbitrary starting point that we need to supply to `optim( )`. At iteration 1 that's random, but from iteration 2 we can set `par =` $\hat{\theta}^{(t-1)}$!

E (expectation) step

In the E step we update estimates $\hat{p}_{ij}^{(t)}$ based on the latest parameter estimates $\hat{\theta}^{(t-1)}, \hat{\pi}^{(t-1)}$:

$$\hat{p}_{ij}^{(t)} = P(Z_i = j | \theta = \hat{\theta}^{(t-1)}) = \frac{\hat{\pi}_j^{(t-1)} f_{X_j}\left(x_i | \theta_j = \hat{\theta}_j^{(t-1)}\right)}{\sum_{j=1}^{K} \hat{\pi}_j^{(t-1)} f_{X_j}\left(x_i | \theta_j = \hat{\theta}_j^{(t-1)}\right)}$$

M (maximization) step

In the M step:

1. we can estimate $\hat{\pi}^{(t)}$ through the following closed-form expression (= no numeric optimization needed here):

$$\hat{\pi}_j^{(t)} = \frac{1}{n} \sum_{i=1}^{n} \hat{p}_{ij}^{(t)}, \; j = 1, ..., K$$

2. we can estimate $\hat{\theta}^{(t)}$ numerically by maximizing the `neg.logl` function that we previously defined

EM algorithm for the estimation of the mixture model

- Body of the EM algorithm:

```r
for (t in 2:n.iter) {
  # E step: update individual probability memberships
  p.temp = cbind(
    p1hat[t-1,] * dnorm(x, muhat[t-1, 1], sigma),
    (1 - p1hat[t-1,]) * dnorm(x, muhat[t-1, 2], sigma) )
  p1hat[t, ] = p.temp[ , 1]/rowSums(p.temp)
  # M step: update parameter estimates
  pi1hat[t] = mean(p1hat[t, ])
  muhat[t, ] = optim(muhat[t-1, ], function(theta)
    neg.logl(theta, pi1hat[t], p1hat[t,], x))$par
}
```

Mixture models

The EM algorithm

Parameter estimates, convervenge, classification and label switching

How to deal with repetitive and time-consuming computations

`replicate( )`

EM algorithm + `replicate( )`

parallelization and the `foreach` package (SELF-STUDY)

Final estimates

- Our ML estimates for θ and π_1 are:

```
muhat[n.iter, ]
```

```
## [1] 1.063929 4.114422
```

```
pi1hat[n.iter]
```

```
## [1] 0.334617
```

- The true values were:

```
c(mu1, mu2)
```

```
## [1] 1 4
```

```
pi1
```

```
## [1] 0.3
```

How do you know that 200 iterations would be enough?

- **Q**: previously you said that `n.iter = 200` would be enough. How do you know that?
- **A**: check if your algorithm has converged to a solution, i.e. if it has reached a point where parameter estimates don't change much iteration after iteration

Checking convergence (part 1)

- Estimates of θ at the last 10 iterations:

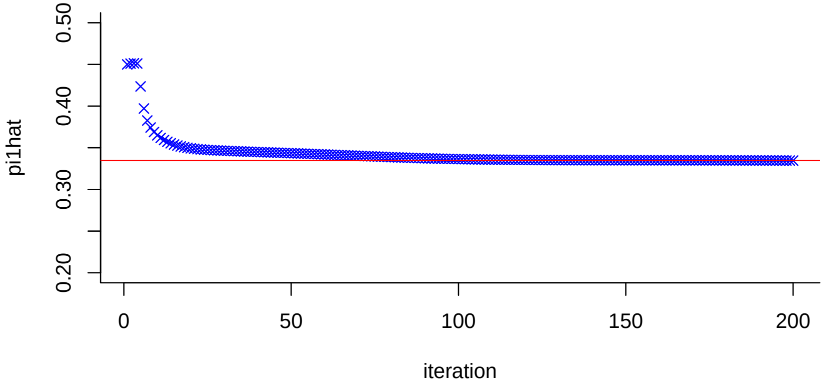
```r
tail(muhat, 10)
```

```
##               [,1]     [,2]
## [191,] 1.064568 4.114726
## [192,] 1.064568 4.114726
## [193,] 1.064568 4.114726
## [194,] 1.064568 4.114726
## [195,] 1.064568 4.114726
## [196,] 1.064203 4.114511
## [197,] 1.064203 4.114511
## [198,] 1.064203 4.114511
## [199,] 1.064203 4.114511
## [200,] 1.063929 4.114422
```

Checking convergence (part 2)

- Estimates of π_1 throughout the whole algorithm:

```r
par(bty = 'l')
plot(pi1hat, xlab = 'iteration', pch = 4, col = 'blue',
     ylim = c(0.2, 0.5))
abline(h = pi1hat[n.iter], col = 'red')
```



Classification accuracy

- Predicted vs true group memberships:

```
p1_last = p1hat[n.iter, ]
predicted_group = ifelse(p1_last > 0.5, 1, 2)
table(predicted_group, group)
```

```
##                  group
## predicted_group   1   2
##               1 290  45
##               2  13 652
```

- We have classified correctly 94.2 % of the subjects! ☺

Label switching and identifiability

- EM solutions are ⚠ identical up to label switching ⚠
- In other words, two mixture models with swapped components ($f_{X_1} \to f_{X_2}$, $f_{X_2} \to f_{X_1}$, $w_1 \to w_2$ and $w_2 \to w_1$) are equivalent and not identifiable (they yield the very same likelihood!)
- For simple problems with 2 groups (in the data) and 2 components (in the mixture model), this creates two possible scenarios:
  1. Component 1 $=$ group 1, comp. 2 $=$ group 2, like in previous example
  2. Component 1 $=$ group 2, comp. 2 $=$ group 1
- When (2) happens, simply switch your predicted groups using

```
predicted_group = ifelse(p1.last > 0.5, 2, 1)
```

- More label-switching combinations possible with $K > 2$
- Compare final EM estimates to input data to try to understand which component better represents each group in your data ☺

Using multiple starting points

- We applied the EM to a mixture of 2 normals with unknown means
  1. this is an "easy" optimization problem; problems involving mixtures usually more complicated than that (unknown σs, $K > 2$, ...)
  2. in general, with numeric optimization ⚠ where your algorithm converges can depend on the random starting point ⚠
- How to address problem (2)? Optimize from different starting points:
  1. consider Q (e.g., 10) random initializations for the EM
  2. for each $q = 1, ..., Q$, (re-)run the EM algorithm
- Q repetitions $\Rightarrow$ repetitive task, and possibly time-consuming
  1. how to quickly run a repetitive algorithm several times? $\Rightarrow$ `replicate( )`
  2. how to make its execution faster using parallel computing? $\Rightarrow$ the `foreach` package

Mixture models

The EM algorithm

Parameter estimates, convervenge, classification and label switching

How to deal with repetitive and time-consuming computations

`replicate( )`

EM algorithm + `replicate( )`

parallelization and the `foreach` package (SELF-STUDY)

replicate( )

- The `replicate( )` function is designed to repeat a given computation a fixed number of times

```r
mean_var_gamma = function(n, alpha, beta) {
  # sample n observations from X \sim Gamma(alpha, beta)
  x = rgamma(n = n, shape = alpha, rate = beta)
  # return the mean and variance of the sampled vector
  c('mean' = mean(x), 'variance' = var(x))
}
set.seed(13)
mc_experiment = replicate(100, mean_var_gamma(10, 4, 3))
is(mc_experiment)
```

```
## [1] "matrix"    "array"     "structure" "vector"
```

- See `?replicate` for details. We will use this function a lot during *Computational Statistics*!

The output of `replicate( )`

- In this example, we repeatedly executed (100 times) a function that returned 2 values
- R returns the output of this replicates in a matrix with 2 rows and 100 columns

```
dim(mc_experiment)
```

```
## [1]   2 100
```

```
mc_experiment[ , 1:4]
```

```
##               [,1]      [,2]      [,3]      [,4]
## mean     1.3186678 1.4373730 0.9748045 1.6362815
## variance 0.2930247 0.3014649 0.2005424 0.3651998
```

- First replicate: $\hat{\mu} = 1.318, \hat{\sigma} = 0.293$; second replicate: $\hat{\mu} = 1.437, \hat{\sigma} = 0.301$; ...

Mixture models

The EM algorithm

Parameter estimates, convervenge, classification and label switching

How to deal with repetitive and time-consuming computations

`replicate( )`

EM algorithm + `replicate( )`

parallelization and the `foreach` package (SELF-STUDY)

Applying `replicate( )` to the EM estimation problem

- We can use `replicate( )` to run the EM from multiple starting points
- To do so we need to:
  1. Create a function that runs the whole EM from a given starting point
  2. Call `replicate(n_reps, EM_function() )`

Function that runs the whole EM

- I have created a script called `L11_EMfunction4replicate.R` that contains:
    1. a function to evaluate the negative loglikelihood
    2. a function that runs the whole EM from a random starting point
- To make use of such functions:
    1. Download `L11_EMfunction4replicate.R` from Brightspace
    2. Move the file to the folder in which you are working
    3. In `R`, source the script using:

```r
source('L11_EMfunction4replicate.R')
```

Running the EM 20 times with `replicate( )`

- To run the EM from 20 different starting points, we can use

```r
set.seed(19931101)
EMruns = replicate(20, EM_function(x = x, n.iter = 200),
                                 simplify = F)
is(EMruns)
```

```
## [1] "list"   "vector"
```

```r
length(EMruns)
```

```
## [1] 20
```

- The output is a list where each element is the output of 1 replication

⚠ Don't forget the `simplify = F` here! ⚠

Choosing the optimal solution

- We have run the algorithm 20 times
- **Q**: which run should you select as solution?
- **A**: choose (one of) the run(s) with the highest likelihood!

```
logl_values = sapply(EMruns, function(x) x$logl)
logl_values
```

```
##  [1] -1970.205 -1971.786 -1968.924 -1968.829 -1973.920
##  [6] -1968.612 -1974.796 -1967.471 -1974.955 -1967.429
## [11] -1970.438 -1967.581 -1974.814 -1968.457 -1969.054
## [16] -1976.433 -1973.381 -1967.314 -1967.446 -1970.928
```

```
which(logl_values == max(logl_values))
```

```
## [1] 18
```

Conclusion

- Run 18 contains our final estimates:

```r
final_est = EMruns[[18]]
final_est$logl # loglikelihood
```

```
## [1] -1967.314
```

```r
final_est$muhat # means
```

```
## [1] 1.060269 4.112692
```

```r
final_est$pi1hat # pi1hat
```

```
## [1] 0.3338128
```

```r
# p1hat for the first 15 observations:
final_est$p1hat[1:15]
```

```
##  [1]  1.000000e+00  0.000000e+00  0.000000e+00
##  [4]  0.000000e+00  1.000000e+00  0.000000e+00
##  [7]  0.000000e+00  1.000000e+00  8.557842e-65
## [10]  0.000000e+00 7.229517e-155  1.000000e+00
## [13]  1.000000e+00  0.000000e+00  0.000000e+00
```

Mixture models

The EM algorithm

Parameter estimates, convervenge, classification and label switching

How to deal with repetitive and time-consuming computations

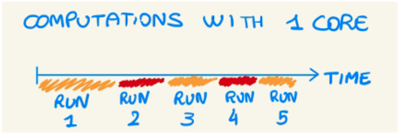
`replicate( )`

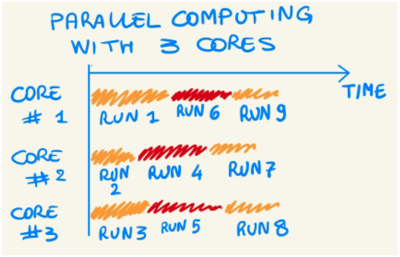
EM algorithm + `replicate( )`

parallelization and the `foreach` package (SELF-STUDY)

Parallel computing

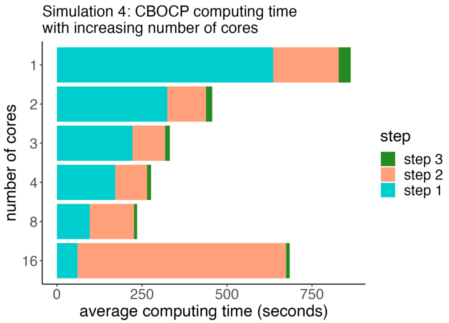
- We have seen how we can use `replicate( )` to run a repetitive computation several times
- We have used a single core, and this was enough for our example
- However, sometimes single-core computing may be too slow
- Solution: parallel computing





Example of the effect of parallelization on computing time

- Increasing the number of cores can reduce computing time, but it can also increase it!
- A recent example from my own work[4]:



[4]for a broader explanation, see page 16 of pencal: an R Package for the Dynamic Prediction of Survival with Many Longitudinal Predictors. Preprint: arXiv.2309.15600

The foreach package

- Handy way to parallelize computations in R: `foreach` package
- Available on CRAN
- If you want to learn how to use `foreach`, make sure to read:
  1. the vignette of the package
  2. this tutorial

Do I need access to a HPC cluster?

A last remark:

- parallelization typically done using an high-performance computing (HPC) cluster (e.g., Leiden University's *Alice* cluster)
- BUT: ⚠ you don't need an HPC cluster to parallelize - your own laptop can already parallelize computations! ⚠
- How? By spreading your computations over the different cores and threads of your processor ☺

Useful readings

- Mixtures: Section 3.5 of Rizzo (2019)
- Statistical and mathematical background behind the EM algorithm for mixture models: Sections 2.1, 2.2 and 2.4.4 of Frühwirth-Schnatter, S. (2006). *Finite mixture and Markov switching models*. Springer (pdf can be downloaded from the library's website)
- `foreach` package:
  1. vignette
  2. this tutorial