Multinomial models and the EM-algorithm

Niels Richard Hansen

October 3, 2018

Peppered Moth

Three alleles, and six different genotypes (CC, CI, CT, II, IT and TT).

Three different phenotypes (black, mottled, light colored).

Allele frequencies: p_C, p_I, p_T with $p_C + p_I + p_T = 1$.

Peppered Moth

According to Hardy-Weinberg equilibrium the genotype frequencies are

$$p_C^2, 2p_Cp_I, 2p_Cp_T, p_I^2, 2p_Ip_T, p_T^2.$$

The complete multinomial log-likelihood is

$$2n_{CC}\log(p_C) + n_{CI}\log(2p_Cp_I) + n_{CT}\log(2p_Cp_I)$$
$$+2n_{II}\log(p_I) + n_{IT}\log(2p_Ip_T) + 2n_{TT}\log(p_T),$$

We only observe (n_C, n_I, n_T), where

$$n = n_{CC} + \underbrace{n_{CI}}_{=n_C} + n_{CT} + \underbrace{n_{IT}}_{=n_I} + n_{II} + \underbrace{n_{TT}}_{=n_T}$$

As a specific data example we have the observation $n_C = 85$, $n_I = 196$, and $n_T = 341.$

Multinomial cell collapsing

The Peppered Moth example is an example of cell collapsing in a multinomial model.

In general, let $A_1 \cup \ldots \cup A_{K_0} = \{1, \ldots, K\}$ be a partition and let

$$M : \mathrm{N}_0^K \to \mathrm{N}_0^{K_0}$$

be the map given by

$$M((n_1, \ldots, n_K))_j = \sum_{i \in A_j} n_i.$$

Multinomial cell collapsing

If $Y \sim \text{Mult}(p, n)$ with $p = (p_1, \ldots, p_K)$ then

$$X = M(Y) \sim \text{Mult}(M(p), n).$$

For the Peppered Moths, $K = 6$ corresponding to the six genotypes, $K_0 = 3$ and the partition corresponding to the phenotypes is

$$\{1, 2, 3\} \cup \{4, 5\} \cup \{6\} = \{1, \ldots, 6\},$$

and

$$M(n_1, \ldots, n_6) = (n_1 + n_2 + n_3, n_4 + n_5, n_6).$$

Multinomial cell collapsing

In terms of the (p_C, p_I) parametrization, $p_T = 1 - p_C - p_I$ and

$$p = (p_C^2, 2p_Cp_I, 2p_Cp_T, p_I^2, 2p_Ip_T, p_T^2).$$

Hence

$$M(p) = (p_C^2 + 2p_Cp_I + 2p_Cp_T, p_I^2 + 2p_Ip_T, p_T^2).$$

The log-likelihood is,

$$\begin{aligned}\ell(p_C, p_I) = n_C\log(p_C^2 + 2p_Cp_I + 2p_Cp_T)\\ +n_I\log(p_I^2 + 2p_Ip_T) + n_T\log(p_T^2)\end{aligned}$$

Peppered Moths negative log-likelihood

We can code a problem specific version of the negative log-likelihood and use `optim` to minimize it.

```r
## par = c(pC, pI), pT = 1 - pC - pI
## x is the data vector of length 3 of counts
loglik <- function(par, x) {
  pT <- 1 - par[1] - par[2]

  if (par[1] > 1 || par[1] < 0 || par[2] > 1
      || par[2] < 0 || pT < 0)
    return(Inf)

  PC <- par[1]^2 + 2 * par[1] * par[2] + 2 * par[1] * pT
  PI <- par[2]^2 + 2 * par[2] * pT
  PT <- pT^2

  - (x[1] * log(PC) + x[2] * log(PI) + x[3]* log(PT))
}
```

Note how parameter constraints are encoded via the return value ∞.

Peppered Moths MLE

```r
## Default algorithm Nelder-Mead doesn't use derivatives
optim(c(0.3, 0.3), loglik, x = c(85, 196, 341))
```

```
$par
[1] 0.07084643 0.18871900

$value
[1] 600.481

$counts
function gradient 
      71       NA 

$convergence
[1] 0

$message
NULL
```

Peppered Moths MLE

Some thought has to go into the initial parameter choice.

```r
optim(c(0, 0), loglik, x = c(85, 196, 341))
```

```
Error in optim(c(0, 0), loglik, x = c(85, 196, 341)): function cannot be evaluated
```

Peppered Moths MLE

Using BFGS (or CG) is possible. Gradients are computed internally by numerical differentiation.

```r
optim(c(0.3, 0.3), loglik, x = c(85, 196, 341), method = "BFGS")
```

```
$par
[1] 0.07083646 0.18873621

$value
[1] 600.481

$counts
function gradient
      40        9

$convergence
[1] 0

$message
NULL
```

Peppered Moths negative log-likelihood

The computations can beneficially be implemented in greater generality.

The function `M` sums the cells that are collapsed, which has to be specified by the `group` argument.

```
M <- function(y, group)
  as.vector(tapply(y, group, sum))
```

Peppered Moths negative log-likelihood

The function `prob` maps the parameters to the multinomial probability vector. This function will have to be problem specific.

```
prob <- function(p) {
  p[3] <- 1 - p[1] - p[2]
  c(p[1]^2, 2 * p[1] * p[2], 2* p[1] * p[3],
    p[2]^2, 2 * p[2] * p[3], p[3]^2)
}
```

Peppered Moths MLE

```
loglik <- function(par, x) {
  pT <- 1 - par[1] - par[2]
  if (par[1] > 1 || par[1] < 0 || par[2] > 1
      || par[2] < 0 || pT < 0)
    return(Inf)

  - sum(x * log(M(prob(par), c(1, 1, 1, 2, 2, 3))))
}
```

Peppered Moths MLE

```r
pep_optim <- optim(c(0.3, 0.3), loglik, x = c(85, 196, 341))
pep_optim
```

```
$par
[1] 0.07084643 0.18871900

$value
[1] 600.481

$counts
function gradient 
      71       NA 

$convergence
[1] 0

$message
NULL
```

Peppered Moths summary

- The Peppered Moth example is very simple. The log-likelihood for the observed data can easily be computed.
- The full data model as well as the model of collapsed cells are curved exponential families.
- The example was used above to illustrate different ways of implementing a likelihood computation in R. One was problem specific and one was more abstract and general.
- Below the example is used to illustrate the EM-algorithm. The EM-algorithm does not require that we can compute the marginal likelihood. In (real) applications this will often not be possible, or it will be a numerically heavy computation.

Exercise: Gaussian mixtures

The following code simulates samples from a Gaussian mixture

```r
sigma1 <- 1
sigma2 <- 2
mu1 <- 0
mu2 <- 4
p <- 0.5
n <- 1000
x <- ifelse(sample(c(0, 1), n, replace = TRUE, prob = c(p, 1 - p)),
            rnorm(n, mu1, sigma1), rnorm(n, mu2, sigma2))
```

The density of the distribution of X is

$$p(2\pi\sigma_1^2)^{-1/2}e^{-(y-\mu_1)^2/(2\sigma_1^2)} + (1-p)(2\pi\sigma_2^2)^{-1/2}e^{-(y-\mu_2)^2/(2\sigma_2^2)}$$

`

- Implement an R function for computing the log-likelihood.
- Use `optim` to fit the parameters μ_1, μ_2, and p. You may assume σ_1 and σ_2 known. What about a gradient?

Incomplete data likelihood

Suppose that Y is a random variable and $X = M(Y)$. Suppose that Y has density $f(\cdot \mid \theta)$ and that X has marginal density $g(x \mid \theta)$.

The marginal density is typically of the form

$$g(x \mid \theta) = \int_{\{y : M(y) = x\}} f(y \mid \theta)\, \mu_x(\mathrm{d}y)$$

for a suitable measure μ_x depending on M and x but not θ.

The log-likelihood for observing $X = x$ is

$$\ell(\theta) = \log g(x \mid \theta).$$

Incomplete data likelihood

The marginal likelihood is often impossible to compute analytically and difficult and expensive to compute numerically.

The complete log-likelihood, $\log f(y \mid \theta)$, is often easy to compute, but we don't know Y, only that $M(Y) = x$.

In some cases it is possible to compute

$$Q(\theta \mid \theta') := E_{\theta'}(\log f(Y \mid \theta) \mid X = x),$$

which is the conditional expectation of the complete log-likelihood given the observed data and under the probability measure given by θ'.

Idea

With an initial guess of $\theta' = \theta^{(0)}$ compute iteratively

$$\theta^{(n+1)} = \arg\max Q(\theta \mid \theta^{(n)})$$

for $n = 0, 1, 2, \ldots$.

This is the EM-algorithm:

- E-step: Compute the conditional expectation $Q(\theta \mid \theta^{(n)})$.
- M-step: Maximize $\theta \mapsto Q(\theta \mid \theta^{(n)})$.

The EM-algorithm

For some nice models (e.g. exponential families) the conditional expectation is easy to compute, the complete log-likelihood is easy to maximize, and this transfers to easy maximization of Q.

We prove below that the algorithm is an ascent algorithm; it (weakly) increases the marginal likelihood in every step.

But first some theory about conditional distributions.

Conditional distributions

It holds in great generality that the conditional distribution of Y given $X = x$ has density

$$h(y \mid x, \theta) = \frac{f(y \mid \theta)}{g(x \mid \theta)}$$

w.r.t. a suitable measure μ_x that does not depend upon θ.

You can verify this for discrete distributions and when $Y = (Z, X)$ with joint density w.r.t. a product measure $\mu \otimes \nu$ that does not depend upon θ.

Conditional distributions

In the latter case, $f(y \mid \theta) = f(z, x \mid \theta)$ and

$$g(x \mid \theta) = \int f(z, x \mid \theta)\, \mu(\mathrm{d}z)$$

is the marginal density w.r.t. ν.

In general

$$\log g(x \mid \theta) = \log f(y \mid \theta) - \log h(y \mid x, \theta),$$

and under some integrability conditions, this decomposition is used to show that the EM-algorithm increases the log-likelihood, $\ell(\theta)$, in each iteration.

Multinomial conditional distributions

The conditional distribution of $Y_{A_j} = (Y_i)_{i \in A_j}$ conditionally on X can be found too.

$$Y_{A_j} \mid X = x \sim \mathrm{Mult}\left(\frac{p_{A_j}}{M(p)_j}, x_j\right).$$

The probability parameters are simply p restricted to A_j and renormalized to a probability vector. Hence

$$E(Y_k \mid X = x) = \frac{x_j p_k}{M(p)_j}$$

for $k \in A_j$.

Abstract E-step

The EM-algorithm can be implemented by two simple functions that compute the conditional expectations above (the E-step) and then maximization of the complete observation log-likelihood.

```
EStep0 <- function(p, x, group) {
  x[group] * p / M(p, group)[group]
}
```

Multinomial MLE

With $y = (n_{CC}, n_{CI}, n_{CT}, n_{II}, n_{IT}, n_{TT})^T$ a complete observation, it can be shown that the MLE is

$$\hat{p}_C = n_{CC} + (n_{CI} + n_{CT})/2$$
$$\hat{p}_I = (n_{CI} + n_{IT})/2 + n_{II}$$

Which is $\hat{p} = \mathbf{X}y$ for

```
 X <- matrix(
  c(2, 1, 1, 0, 0, 0,
    0, 1, 0, 2, 1, 0) / 2,
  2, 6, byrow = TRUE)
X
```

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1  0.5  0.5    0  0.0    0
[2,]    0  0.5  0.0    1  0.5    0
```

Abstract M-step

The MLE of the complete log-likelihood is a linear estimator, as is the case in many examples with explicit MLEs.

```
MStep0 <- function(n, X)
  as.vector(X %*% n / (sum(n)))
```

The `EStep0` and `MStep0` functions are abstract implementations. They require specification of the arguments `group` and `X`, respectively, to become concrete.

The M-step is only implemented in the case where the complete-data MLE is a linear estimator, that is, a linear map of the complete data vector y that can be expressed in terms of a matrix $\mathbf{X}$.

Peppered Moths E- and M-steps

Concrete functions for the E- and M-steps are implemented for the particular example.

```
EStep <- function(p, x)
  EStep0(prob(p), x, c(1, 1, 1, 2, 2, 3))

MStep <- function(n) {
  X <- matrix(
  c(2, 1, 1, 0, 0, 0,
    0, 1, 0, 2, 1, 0) / 2,
  2, 6, byrow = TRUE)

  MStep0(n, X)
}
```

Peppered Moths EM

```r
EM <- function(par, x, epsilon = 1e-6, trace = NULL) {
  repeat{
     par0 <- par
     par <- MStep(EStep(par, x))
     if(!is.null(trace)) trace()
     if(sum((par - par0)^2) <= epsilon * (sum(par^2) + epsilon))
        break
  }
  par  ## Remember to return the parameter estimate
}

EM(c(0.3, 0.3), c(85, 196, 341))
pep_optim$par
```

```
[1] 0.07083693 0.18877365
[1] 0.07084643 0.18871900
```

Inside the EM

Check what is going on in each step of the EM-algorithm.

```r
source("Debugging_and_tracing.R")
```

```r
EM_tracer <- tracer("par")
EM(c(0.3, 0.3), c(85, 196, 341), trace = EM_tracer$trace)
```

```
n = 1: par = 0.08038585, 0.22464192;
n = 2: par = 0.07118928, 0.19546961;
n = 3: par = 0.07084985, 0.18993393;
n = 4: par = 0.07083738, 0.18894757;
n = 5: par = 0.07083693, 0.18877365;
```

```
[1] 0.07083693 0.18877365
```

Inside the EM

```
summary(EM_tracer)
```

```
       par.1     par.2 .time
1 0.08038585 0.2246419 0.000
2 0.07118928 0.1954696 0.000
3 0.07084985 0.1899339 0.000
4 0.07083738 0.1889476 0.001
5 0.07083693 0.1887737 0.001
```

Inside the EM

```r
EM_tracer <- tracer(c("par0", "par"), N = 0)
phat <- EM(c(0.3, 0.3), c(85, 196, 341), epsilon = 1e-20, trace = EM_trac
phat
```

```
[1] 0.07083691 0.18873652
```

```r
EM_trace <- summary(EM_tracer)
tail(EM_trace)
```

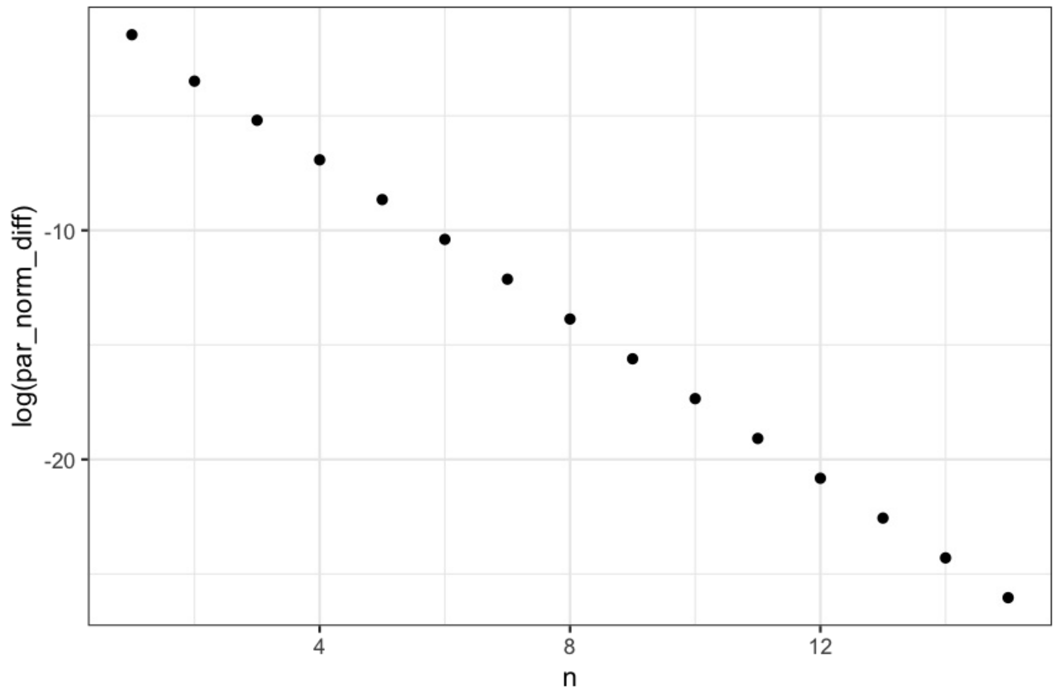
```
       par0.1    par0.2      par.1     par.2 .time
10 0.07083691 0.1887366 0.07083691 0.1887365 0.002
11 0.07083691 0.1887365 0.07083691 0.1887365 0.002
12 0.07083691 0.1887365 0.07083691 0.1887365 0.002
13 0.07083691 0.1887365 0.07083691 0.1887365 0.002
14 0.07083691 0.1887365 0.07083691 0.1887365 0.003
15 0.07083691 0.1887365 0.07083691 0.1887365 0.003
```

Adding computed values

```r
loglik_pep <- Vectorize(function(p1, p2) loglik(c(p1, p2),  c(85, 196, 34
EM_trace <- transform(
  EM_trace,
  n = 1:nrow(EM_trace),
  par_norm_diff = sqrt((par0.1 - par.1)^2 + (par0.2 - par.2)^2),
  loglik = loglik_pep(par.1, par.2)
)
```

Inside the EM

```
qplot(n, log(par_norm_diff), data = EM_trace)
```



Note the log-axis. The EM-algorithm converges linearly (this is the terminology, see Algorithms and Convergence).

Linear convergence

The log-rate of the convergence can be estimated by least-squares.

```r
log_rate <- coefficients(lm(log(par_norm_diff) ~ n,
                            data = EM_trace))["n"]
exp(log_rate)
```

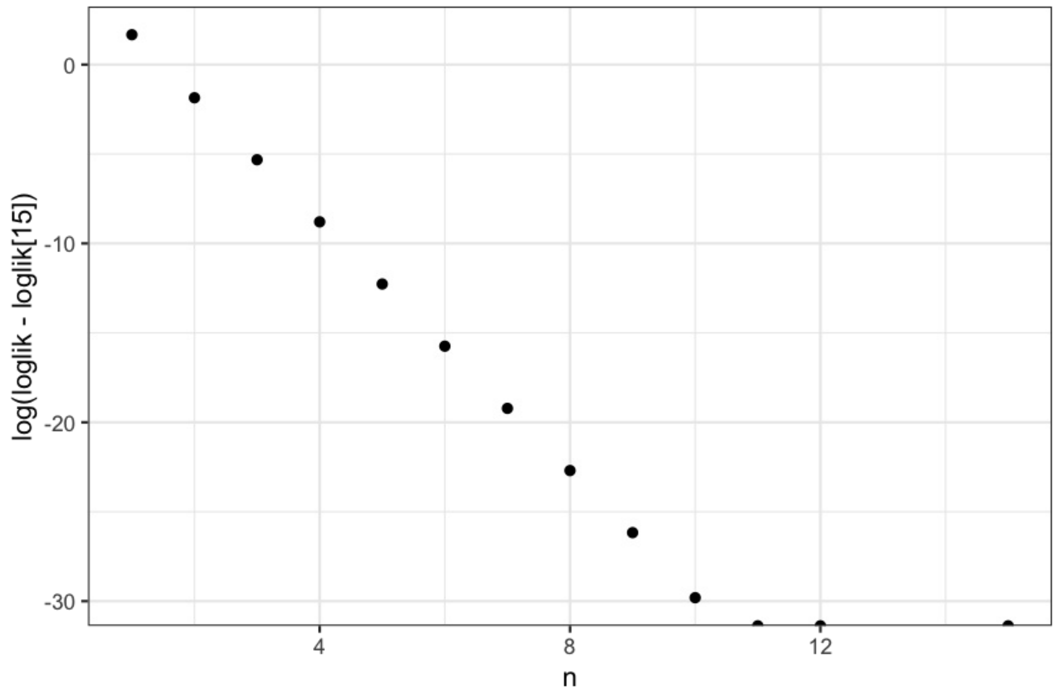
```
        n
0.1750251
```

It is very small in this case implying fast convergence.

This is not always the case. If the log-likelihood is flat, the EM-algorithm can become quite slow with a rate close to 1.

Convergence of the log-likelihood

```
qplot(n, log(loglik- loglik[15]), data = EM_trace)
```



Exercise: Gaussian mixtures

- Implement the EM-algorithm for the Gaussian mixture.
- Can you generalize to more than two groups?
- Can you also estimate the variance parameters?

Object oriented implementation

An (S3) object oriented implementation is useful when

- some parts of an algorithm can be implemented abstractly
- other parts require details specific to a particular problem

Details can be data, dimensions and parametrizations. These are logically related, and they can be stored in a list and given a label.

That's all there is to an S3 object.

Pep. Moths object

```r
pepMoths <- structure(
  list(
    x = c(85, 196, 341),

    X = matrix(
      c(2, 1, 1, 0, 0, 0,
        0, 1, 0, 2, 1, 0) / 2,
      2, 6, byrow = TRUE),

    group = c(1, 1, 1, 2, 2, 3),

    par = c(0.3, 0.3),

    prob = function(p) {
      p[3] <- 1 - p[1] - p[2]
      c(p[1]^2, 2 * p[1] * p[2], 2* p[1] * p[3],
        p[2]^2, 2 * p[2] * p[3], p[3]^2)
    }
  ),
  class = "MultCollapse"
)
```

EM generic function

```
EM <- function(x, ...)  ## ... allows for additional arg.
  UseMethod("EM")
```

An EM method

```r
EM.MultCollapse <- function(x, epsilon = 1e-6) {
    par <- x$par
   repeat{
     par0 <- par
     par <- MStep0(EStep0(x$prob(par), x$x, x$group), x$X)
     if(sum((par - par0)^2) <= epsilon * (sum(par^2) + epsilon))
        break
  }
  x$par <- par
  x ## Returns the entire object
}
```

Pep. Moths EM

```
pepMoths <- EM(pepMoths)
pepMoths$par
```

```
[1] 0.07083693 0.18877365
```

```
phat
```

```
[1] 0.07083691 0.18873652
```

```
str(pepMoths)
```

```
List of 5
 $ x    : num [1:3] 85 196 341
 $ X    : num [1:2, 1:6] 1 0 0.5 0.5 0.5 0 0 1 0 0.5 ...
 $ group: num [1:6] 1 1 1 2 2 3
 $ par  : num [1:2] 0.0708 0.1888
 $ prob :function (p)
  ..- attr(*, "srcref")= 'srcref' int [1:8] 14 12 18 5 12 5 14 18
  .. ..- attr(*, "srcfile")=Classes 'srcfilecopy', 'srcfile' <environment: 0x7fcf1e
 - attr(*, "class")= chr "MultCollapse"
```

Optimization and statistics

The EM-algorithm is a general algorithm for numerical optimization of a log-likelihood function. It works by iteratively optimizing

$$Q(\theta \mid \theta^{(n)}) = E_{\theta^{(n)}}(\log f(Y \mid \theta) \mid X = x).$$

For numerical optimization of Q or variants of EM (like EM gradient or acceleration methods) the gradient and Hessian of Q can be useful.

For statistics we need the observed Fisher information (Hessian of the negative log-likelihood for the observed data).

There are interesting and useful relations between these different derivatives that we will cover in this lecture.

Gradient

Note that with $p = p(\theta)$ in some parametrization of the cell probabilities,

$$Q(\theta \mid \theta') = \sum_i \frac{x_{j(i)} p_i(\theta')}{M(p(\theta'))_{j(i)}} \log p_i(\theta),$$

where $j(i)$ is defined by $i \in A_{j(i)}$.

The gradient of Q w.r.t. θ and evaluated in θ' is

$$\nabla_\theta Q(\theta' \mid \theta') = \sum_i \frac{x_{j(i)}}{M(p(\theta'))_{j(i)}} \nabla p_i(\theta').$$

Gradient

```
Dprob <- function(p) {
   matrix(
    c(2 * p[1],                      0,
      2 * p[2],                 2 * p[1],
      2* p[3] - 2 * p[1],  -2 * p[1],
      0,                        2 * p[2],
      -2 * p[2], 2 * p[3] - 2 * p[2],
      -2 * p[3],               -2 * p[3]),
    ncol = 2, nrow = 6, byrow = TRUE)
}

gradQ <- function(p, x) {
  p[3] <- 1 - p[1] - p[2]
  group <- c(1, 1, 1, 2, 2, 3)
  - (x[group] / M(prob(p), group)[group]) %*% Dprob(p)
}
```

Gradient identity

Though computed as the gradient of Q,

$$\nabla_\theta Q(\theta' \mid \theta') = \nabla_\theta \ell(\theta')$$

from the fact that θ' maximizes

$$\theta \mapsto Q(\theta \mid \theta') - \ell(\theta).$$

This can also be verified by direct computation.

We can use the gradient for optimization of the log-likelihood.

Gradient used for optimization

```r
optim(c(0.3, 0.3), loglik, gr = gradQ, x = c(85, 196, 341),
      method = "BFGS")[1:3]
```

```
$par
[1] 0.07083689 0.18873652

$value
[1] 600.481

$counts
function gradient
      46        9
```

```r
phat  ## Optimum computed by EM algorithm
```

```
[1] 0.07083691 0.18873652
```

Empirical Fisher information

Note that a multinomial observation with size parameter n can be regarded as n i.i.d. samples.

For i.i.d. samples the Fisher information (for one sample) can be estimated as the empirical variance of the gradient of the log-likelihood. By the identity

$$\nabla_\theta Q(\theta' \mid \theta') = \nabla_\theta \ell(\theta')$$

holding for each sample, we can compute the empirical variance.

Pep. Moths empirical Fisher

```
empFisher <- function(p, x, center = FALSE) {
  gradQ <- 0 ## is supposed to be 0 in the MLE
  if (center)
     gradQ <-  gradQ(p, x) / sum(x)
   grad1 <- gradQ(p, c(1, 0, 0)) - gradQ
   grad2 <- gradQ(p, c(0, 1, 0)) - gradQ
   grad3 <- gradQ(p, c(0, 0, 1)) - gradQ
   x[1] * t(grad1) %*% grad1 +
     x[2] * t(grad2) %*% grad2 +
     x[3] * t(grad3) %*% grad3
}
```

Note that the gradient is 0 in the maximum of $Q(\cdot \mid \hat{\theta})$, but the code above allows for empirically centering the gradient before computing the estimate of the Fisher information.

Pep. Moths empirical Fisher

```
empFisher(phat, c(85, 196, 341))
```

```
          [,1]     [,2]
[1,] 18487.558 1384.626
[2,]  1384.626 6816.612
```

```
empFisher(phat, c(85, 196, 341), center = TRUE)
```

```
          [,1]     [,2]
[1,] 18487.558 1384.626
[2,]  1384.626 6816.612
```

Pep. Moths numerical Fisher information

We can compute the observed Fisher information using `optim`.

```
ihat <- optim(c(0.3, 0.3), loglik, x = c(85, 196, 341), hessian = TRUE)$h
ihat
```

```
          [,1]     [,2]
[1,] 18489.734 1384.604
[2,]  1384.604 6817.921
```

Note that this Hessian and the empirical Fisher informations above are different estimates of the same quantity. Thus they are not supposed to be identical on a given data set.

Fisher information

Computing an estimate of the Fisher information is important for computing standard errors of MLEs.

The empirical Fisher information can be computed for i.i.d. samples using the gradient of Q, if that gradient can be computed or estimated.

SEM is a general alternative that relies only on EM-steps and a numerical differentiation scheme.

Bootstrap is a another alternative for estimating standard errors; in the i.i.d. case it can be done nonparametrically , which means that the inference is correct even if the model is wrong; for other models, parametric bootstrapping can be implemented, which then hinges on model assumptions just like the Fisher information. Bootstrapping is computationally expensive. It will not be covered in this course (but in Regression).

The EM-mapping

Define $\Phi : \Theta \mapsto \Theta$ by

$$\Phi(\theta^{'}) = \arg\max Q(\theta \mid \theta^{'}).$$

- A global maximum of the likelihood is a fixed point of Φ.
- The EM-algorithm searches for a fixed point for Φ, that is, a solution to
$$\Phi(\theta) = \theta.$$
- Variations of the EM-algorithm can often be seen as other ways to find a fixed point for Φ.

Information identity

From

$$\ell(\theta) = Q(\theta \mid \theta') - H(\theta \mid \theta')$$

it follows that the observed Fisher information equals

$$\hat{i}_X := -D_\theta^2 \ell(\hat{\theta}) = -D_\theta^2 \underbrace{Q(\hat{\theta} \mid \theta')}_{=\hat{i}_Y(\theta')} + D_\theta^2 \underbrace{H(\hat{\theta} \mid \theta')}_{=-\hat{i}_{Y|X}(\theta')} .$$

It is possible to compute $\hat{i}_Y := \hat{i}_Y(\hat{\theta})$. For Pep. Moths (and exponential families) it is as difficult as computing the Fisher information for complete observations.

We want to compute $\hat{i}_X$ and $\hat{i}_{Y|X} := \hat{i}_{Y|X}(\hat{\theta})$ is not computable either.

Supplemented EM

It can be shown that

$$D_\theta \Phi(\hat{\theta})^T = \hat{i}_{Y|X}\left(\hat{i}_Y\right)^{-1}.$$

Hence

$$\begin{aligned}\hat{i}_X &= \left(I - \hat{i}_{Y|X}\left(\hat{i}_Y\right)^{-1}\right)\hat{i}_Y \\ &= \left(I - D_\theta \Phi(\hat{\theta})^T\right)\hat{i}_Y.\end{aligned}$$

$D_\theta \Phi(\hat{\theta})$ can be computed via numerical differentiation.

Numerical hessian of Q

First we implement the map Q as an R function.

```
Q <- function(p, pp, x = c(85, 196, 341)) {
  p[3] <- 1 - p[1] - p[2]
  pp[3] <- 1 - pp[1] - pp[2]
  group <- c(1, 1, 1, 2, 2, 3)
  - (x[group] * prob(pp) / M(prob(pp), group)[group]) %*% log(prob(p))
}
```

The R package numDeriv contains functions that compute numerical derivatives.

```
library(numDeriv)
iY <- hessian(Q, phat, pp = phat)
```

Supplemented EM

```
Phi <- function(pp) MStep(EStep(pp, x = c(85, 196, 341)))
DPhi <- jacobian(Phi, phat)  ## Using numDeriv function 'jacobian'
iX <- (diag(1, 2) - t(DPhi)) %*% iY
iX
```

```
          [,1]     [,2]
[1,] 18487.558 1384.626
[2,]  1384.626 6816.612
```

```
ihat
```

```
          [,1]     [,2]
[1,] 18489.734 1384.604
[2,]  1384.604 6817.921
```

Supplemented EM

The inverse Fisher information is

$$\begin{aligned}\hat{i}_X^{-1} &= \hat{i}_Y^{-1}\left(I - D_\theta\Phi(\hat{\theta})^T\right)^{-1}\\ &= \hat{i}_Y^{-1}\left(I + \sum_{n=1}^{\infty}\left(D_\theta\Phi(\hat{\theta})^T\right)^n\right)\\ &= \hat{i}_Y^{-1} + \hat{i}_Y^{-1}D_\theta\Phi(\hat{\theta})^T\left(I - D_\theta\Phi(\hat{\theta})^T\right)^{-1}\end{aligned}$$

where the second identity follows by the Neumann series.

The last formula above explicitly gives the asymptotic variance for the incomplete observation X as the asymptotic variance for the complete observation Y plus a correction term.

Supplemented EM

```r
iYinv <- solve(iY)
iYinv %*% solve(diag(1, 2) - t(DPhi))
```

```
              [,1]          [,2]
[1,]  5.492602e-05 -1.115686e-05
[2,] -1.115686e-05  1.489667e-04
```

```r
iYinv + iYinv %*% t(solve(diag(1, 2) - DPhi, DPhi))
```

```
              [,1]          [,2]
[1,]  5.492602e-05 -1.115686e-05
[2,] -1.115686e-05  1.489667e-04
```

Compare these with previous computations

```
solve(iX) ## SEM-based, but different use of inversion
```

```
              [,1]          [,2]
[1,]  5.492602e-05 -1.115686e-05
[2,] -1.115686e-05  1.489667e-04
```

```
solve(ihat) ## Numerical hessian
```

```
              [,1]          [,2]
[1,]  5.491927e-05 -1.115318e-05
[2,] -1.115318e-05  1.489373e-04
```

Supplemented EM

The SEM implementation above relies on the `hessian` and `jacobian` functions from the numDeriv package for numerical differentiation.

It is possible to implement the computation of the hessian of Q analytically.

Variants on the strategy for computing $D_\theta \Phi(\hat{\theta})$ via numerical differentiation suggest using difference approximations along the s equence of EM-steps. I don't see the point of doing this over standard numerical differentiation (besides that it explicitly uses that $\hat{\theta}$ is a fixed point). On the contrary, I don't believe such difference approximations will converge. When the algorithm gets sufficiently close to the MLE, the numerical errors will dominate.

For Monday

- Implement SEM for Gaussian mixtures.
- Experiment with using Rcpp for implementing the computation of the log-likelihood.