STAT6180
Advanced Topics in Statistics
Lecture 3

Optimization

Overview

- The NR Algorithm

Another Example with two parameters

R free software:

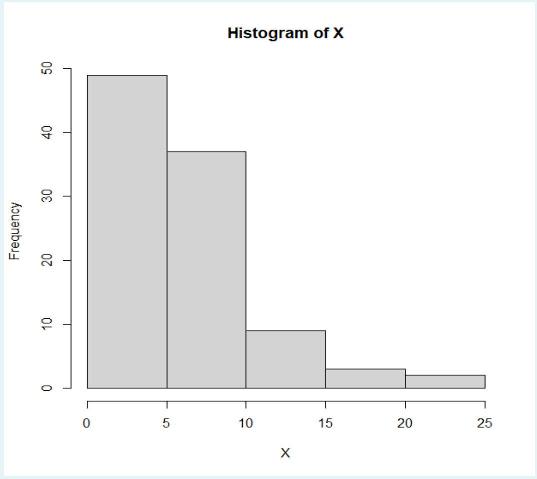
- We can run simulations
- We code ‘new’ statistical methods
- R is not particularly good at heavy computational problems
- R’s memory management is not the best in the world.

Now illustrate the NR method with two parameters

This is the approach; we first simulate 100 values from a known distribution with the parameters KNOWN (off course).

We will generate n=100 values from a Gamma(2,3). Then we will try to use the NR method to estimate those parameters.

```
n <- 100
X <- rgamma(n, shape=2, scale=3)
hist(X)
```



- We will need define the likelihood function
- Then we take the logs of that function
- We will see why it is the sum of logs
- It requires a vector t = c(scale, shape)

If $X \sim \text{Gamma}(\alpha, \beta)$, $f(x) = \frac{1}{\beta^{\alpha}\Gamma(\alpha)} x^{\alpha-1} e^{-\frac{x}{\beta}}$.

β is scale and α is shape parameter

Define the sum of the likelihood function

```
f <- function(t)
{
a <- t[1]; b <- t[2];
if( (a > 0) & (b > 0) )
{
return( sum( dgamma(X, a, scale=b, log=TRUE) ) )
} else
{
return(-Inf)
}
}
```

The vector of score functions – first differentials

```
 df <- function(t)
{
g=c(0,0)
a <- t[1]; b <- t[2];
if( (a > 0) & (b > 0) )
{
g[1] <- -n*digamma(a) - n*log(b) + sum(log(X))
g[2] <- -n*a/b + sum(X)/(b^2)
} else
{
g <- c(-Inf, -Inf)
}
return(g)
}
```

The Hessian matrix of second differentials

```
d2f <- function(t)
{
h = matrix(0,2,2)
a <- t[1]; b <- t[2];
if( (a>0) & (b>0) )
{
h[1,1] = -n*trigamma(a)
h[2,2] = n*a/(b^2) - 2*sum(X)/(b^3)
h[1,2] = -n/b
h[2,1] = -n/b
} else
{
h <- matrix(-Inf, 2, 2)
}
return(h)
}
```

Now apply the NR

```
### We simply apply newton's method now.

> newton( c(1,5), f, df, d2f)
         [,1]
[1,] 1.653766
[2,] 4.029083
```

Gamma Functions Notation

gamma*	the Gamma function,
lgamma*	the logarithm of the Gamma function,
digamma*	the first derivative of the Log Gamma function,
trigamma*	the second derivative of the Log Gamma function,

Using the NR for a Linear Model

Linear model has the form

$$\underset{\sim}{Y} = X\underset{\sim}{\beta} + \underset{\sim}{\epsilon}$$

Ex: $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \epsilon$

$$\epsilon \sim N(0, \sigma^2)$$

Now lets generate some fake data

```
# Generate some fake data with
# an intercept and 4 predictors
n <- 100
Beta <- c(.5, 1.2, -1.8, 3.6, 2.1)
X <- cbind( rep(1,n), rnorm(n), rnorm(n), rnorm(n), rnorm(n) )
Y <- rep(0,n)
for(i in 1:n) Y[i] <- sum( Beta * X[i,] ) + rnorm(1)
p <- ncol(X)
```

Full Sample log likelihood

```
L <- function(B)
{
like <- 0
for(i in 1:n)
{
# subject mean
mu_i <- sum( X[i,] * B )
like <- like + dnorm(Y[i], mean=mu_i, sd=1, log=TRUE)
}
return(like)
}
```

Vector of score functions

```
dL <- function(B)
{
grad <- rep(0, p)
for(i in 1:n)
{
mu_i <- sum( X[i,] * B )
for(j in 1:p)
{
grad[j] <- grad[j] + X[i,j]*(Y[i] - mu_i)
}
}
return(grad)
}
```

Hessian Matrix

```
d2L <- function(B)
{
H <- matrix(0, p, p)
for(i in 1:n)
{
for(j in 1:p)
{
for(k in 1:p)
{
H[j,k] <- H[j,k] - X[i,j]*X[i,k]
}
}
}
return(H)
}
```

Apply Newton Raphson Algorithm

```
> ### Test on our fake data and compare with lm() output
> newton(rep(0,p), L, dL, d2L)

          [,1]
[1,]   0.5690926
[2,]   1.2501343
[3,]  -1.7757474
[4,]   3.6766321
[5,]   2.0456357
```

These are the ORIGINAL betas:

```
Beta  <-  c(0.5,   1.2,    -1.8,    3.6,    2.1)
```

Lets estimate the betas from the lm function using the data we generated.

```
> lm.out = lm(Y ~ X[,2] + X[,3] + X[,4] + X[,5])
>
> summary(lm.out)

Call:
lm(formula = Y ~ X[, 2] + X[, 3] + X[, 4] + X[, 5])

Residuals:
     Min       1Q   Median       3Q      Max
-2.31301 -0.75329  0.05361  0.69407  2.33808

Coefficients:
             Estimate  Std. Error   t value   Pr(>|t|)
(Intercept)   0.5691    0.1065       5.344    6.21e-07 ***
X[, 2]        1.2501    0.1036      12.064    < 2e-16 ***
X[, 3]       -1.7757    0.1108     -16.033    < 2e-16 ***
X[, 4]        3.6766    0.1043      35.262    < 2e-16 ***
X[, 5]        2.0456    0.1091      18.749    < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.05 on 95 degrees of freedom
Multiple R-squared:  0.9578,    Adjusted R-squared:  0.956
F-statistic: 539.2 on 4 and 95 DF,  p-value: < 2.2e-16
```

Extracting the coefficients alone from the lm output.

```
>lm(Y ~ X[,2] + X[,3] + X[,4] + X[,5])$coef

 (Intercept)     X[, 2]      X[, 3]      X[, 4]      X[, 5]
 0.5690926   1.2501343  -1.7757474   3.6766321   2.0456357
>
```

Optim function in R

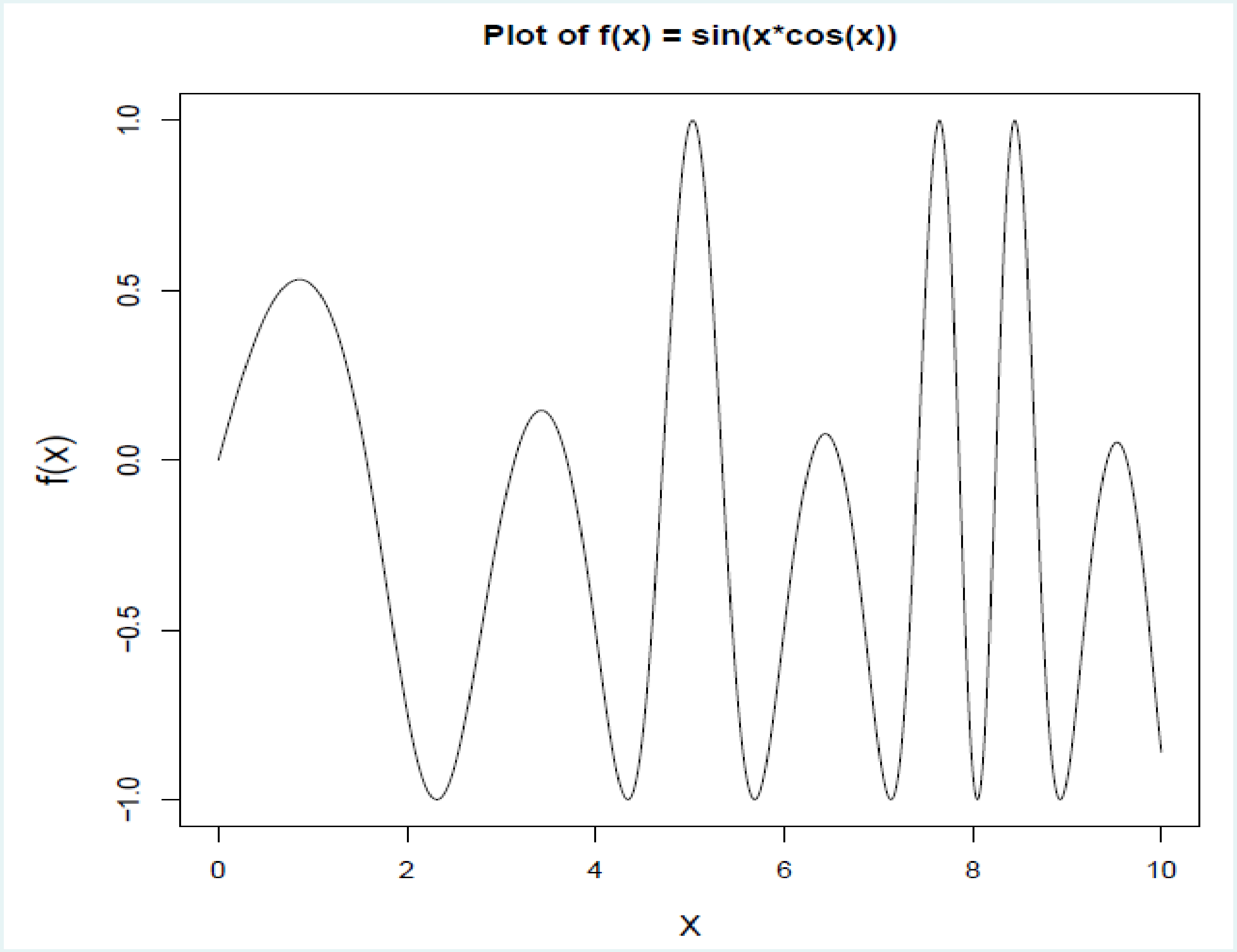
- **optim to solve minimization problems**
  - if you want to maximize you must supply the function multiplied by -1.
  - The default method for optim is a derivative-free optimization routine called the Nelder-Mead simplex algorithm.
  - The basic syntax is optim(init, f) where init is a vector of initial values you must specify and f is the objective function.
  - There are many optional argument (SEE HELP FILE!)
  - If you have also calculated the derivative and stored it in a function df, then the syntax is optim(init, f, df, method="CG") There are many choices for method, but CG is one of the better ones.
  - In many cases the derivative calculation itself is difficult, so the default choice will be preferred

A simple application of optim

$$f(x) = e^{-(x-2)^2}$$

```
> # we supply negative f, since we want to maximize.
> f <- function(x) -exp(-( (x-2)^2 ))
> # without derivative
> # I am using 1 at the initial value
> # $par extracts only the argmax and nothing else
> optim(1, f)$par
[1] 1.9
Warning message:
In optim(1, f) : one-dimensional optimization by Nelder-Mead is unreliable:
use "Brent" or optimize() directly
> # with derivative
> df <- function(x) -2*(x-2)*f(x)
> optim(1, f, df, method="CG")$par
[1] 2
>
```

What about a graph that looks like this?



Now we can prepare the function for Optim

```
f <- function(x) sin(x*cos(x))
optim(2, f)$par
optim(4, f)$par
optim(6, f)$par
optim(8, f)$par
```

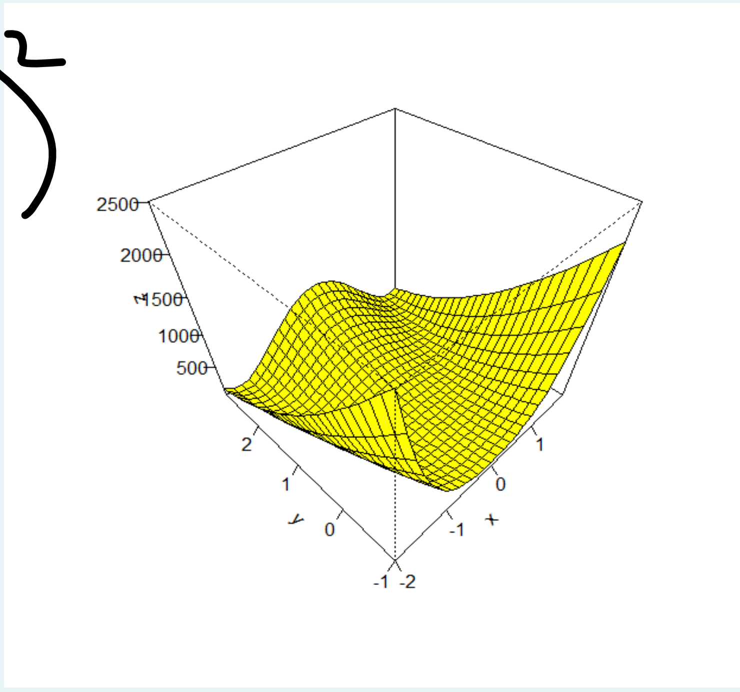
This is the output….

```
> f <- function(x) sin(x*cos(x))
> optim(2, f)$par
[1] 2.316016
Warning message:
In optim(2, f) : one-dimensional optimization by Nelder-Mead is unreliable:
use "Brent" or optimize() directly
> optim(4, f)$par
[1] 4.342236
Warning message:
In optim(4, f) : one-dimensional optimization by Nelder-Mead is unreliable:
use "Brent" or optimize() directly
> optim(6, f)$par
[1] 5.688647
Warning message:
In optim(6, f) : one-dimensional optimization by Nelder-Mead is unreliable:
use "Brent" or optimize() directly
> optim(8, f)$par
[1] 7.132227
Warning message:
In optim(8, f) : one-dimensional optimization by Nelder-Mead is unreliable:
use "Brent" or optimize() directly
```

Graphing a function with two variables

```
f <- function(x1,y1) (1-x1)^2 + 100*(y1 - x1^2)^2
x <- seq(-2,2,by=.15)
y <- seq(-1,3,by=.15)
z <- outer(x,y,f)
persp(x,y,z,phi=45,theta=-
45,col="yellow",shade=.00000001,ticktype="detailed")
```

$$f(x,y) = (1-x)^2 + 100(y-x^2)^2$$



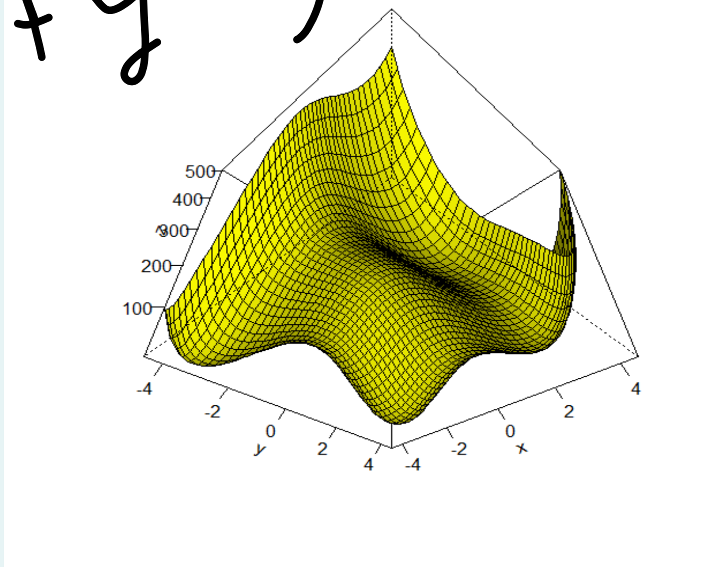
Now lets apply the optim function

```
> optim( c(0,0), f )$par
[1] 0.9999564 0.9999085
```

Another Example – Sketch the function

```
f <- function(x1,y1) (x1^2 + y1 - 11)^2 + (x1 + y1^2 - 7)^2
x <- seq(-4.5,4.5,by=.2)
y <- seq(-4.5,4.5,by=.2)
z <- outer(x,y,f)
persp(x,y,z,phi=-45,theta=45,col="yellow",shade=.65 ,ticktype="detailed")
```

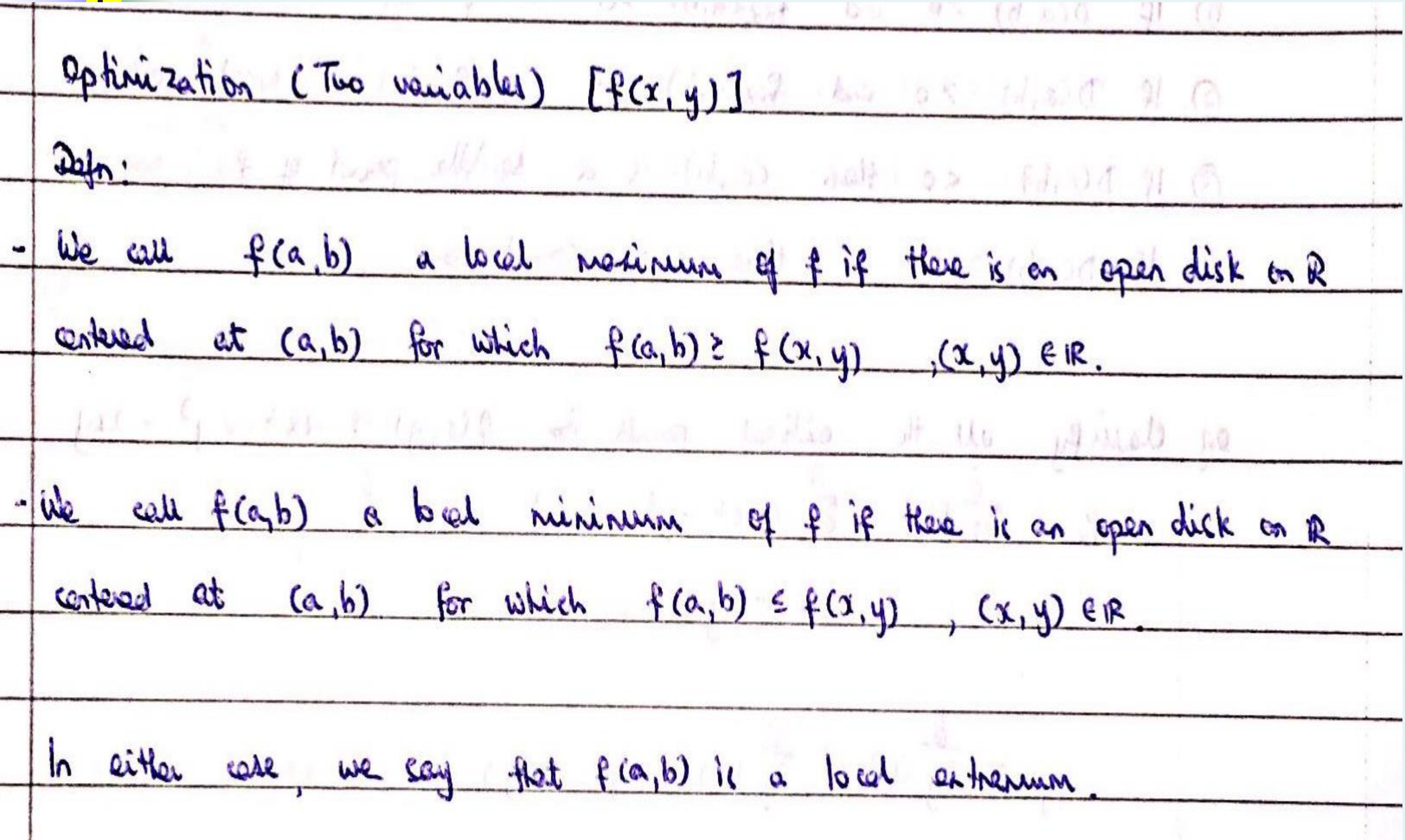
$$f(x,y) = (x^2+y-11)^2 + (x+y^2-7)^2$$



Use the Optim function

```
> f <- function(x) (x[1]^2 + x[2] - 11)^2 + (x[1] + x[2]^2 - 7)^2
> optim(c(-4,-4), f)$par
[1] -3.779347   -3.283172
> optim(c(2,-2), f)$par
[1]  3.584370   -1.848105
> optim(c(2,2), f)$par
[1] 3.000014     2.000032
> optim(c(-4,4),f)$par
[1] -2.805129    3.131435
```

Some calculus for two variable functions



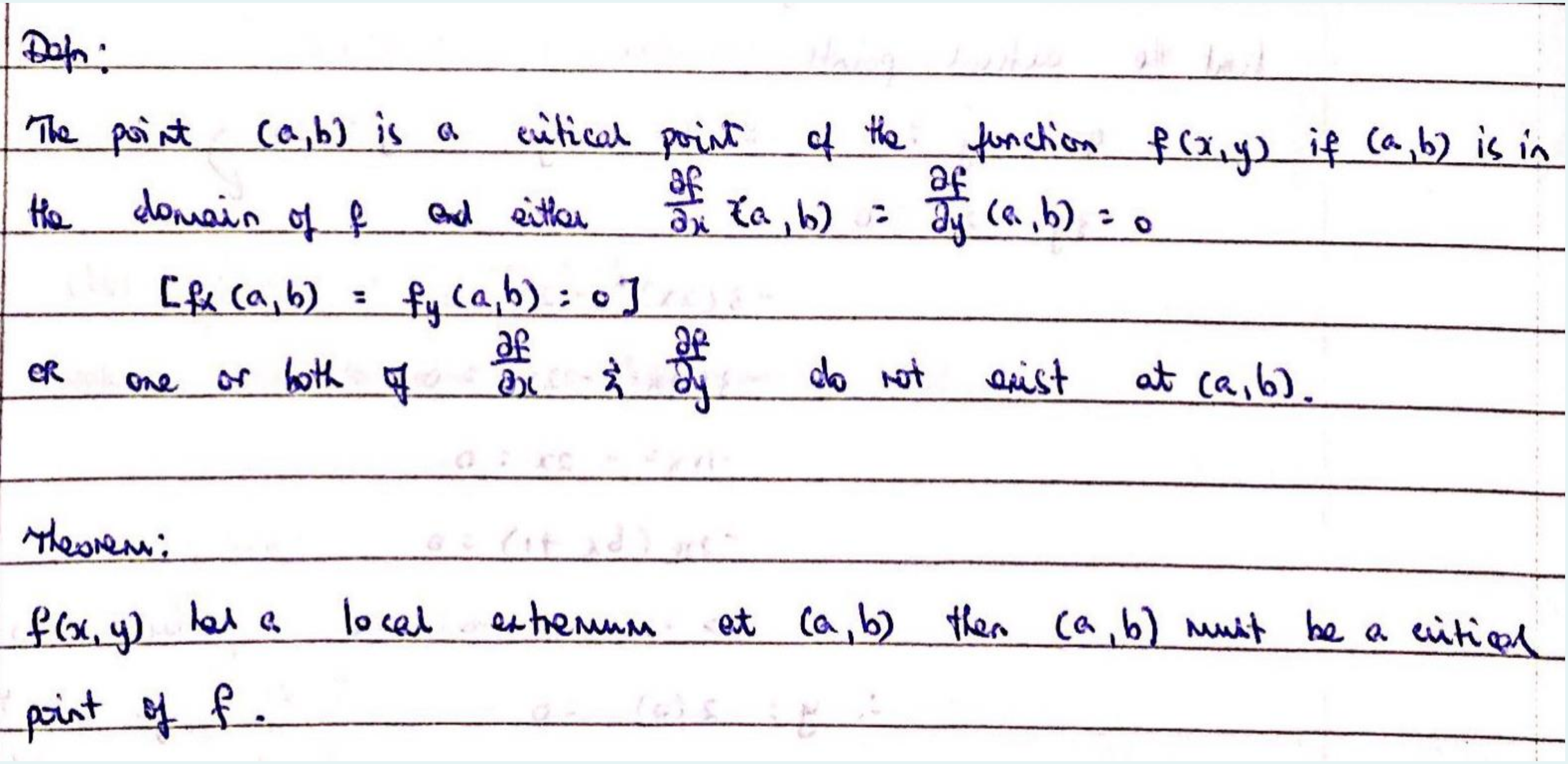
Optimization (Two variables) $[f(x,y)]$

Defn:

- We call $f(a,b)$ a local maximum of f if there is an open disk on R centered at (a,b) for which $f(a,b) \geq f(x,y)$, $(x,y) \in \mathbb{R}$.

- We call $f(a,b)$ a local minimum of f if there is an open disk on R centered at (a,b) for which $f(a,b) \leq f(x,y)$, $(x,y) \in \mathbb{R}$.

In either case, we say that $f(a,b)$ is a local extremum.



Defn:

The point (a,b) is a critical point of the function $f(x,y)$ if (a,b) is in the domain of f and either $\frac{\partial f}{\partial x}(a,b) = \frac{\partial f}{\partial y}(a,b) = 0$

$[f_x(a,b) = f_y(a,b) = 0]$

OR one or both of $\frac{\partial f}{\partial x}$ & $\frac{\partial f}{\partial y}$ do not exist at (a,b).

Theorem:

$f(x,y)$ has a local extremum at (a,b) then (a,b) must be a critical point of f.

Defn :

The point $P(a, b, f(a,b))$ is a saddle point of f if (a, b) is a critical point of f and every open disk centred (a,b) contains points (x, y) in the domain of f for which $f(x,y) > f(a,b)$.

Second Derivative Test

Theorem :

Suppose $f(x, y)$ for continuous 2nd order partial derivatives in some open disk containing the point (a, b) and $f_x(a,b) = f_y(a,b) = 0$

Define the discriminant D for the point (a, b) by

$D(a, b) = f_{xx}(a, b)\, f_{yy}(a, b) - (f_{xy}(a, b))^2$

① If $D(a,b) > 0$ and $f_{xx}(a,b) > 0 \rightarrow f$ has a local min.

② If $D(a,b) > 0$ and $f_{xx}(a,b) < 0 \rightarrow f$ has a local max.

③ If $D(a,b) < 0$ then (a, b) is a saddle point of f.

④ If $D(a,b) = 0$ ... there is no conclusion ...

classify all the critical points for $f(x,y) = 2x^2 - y^3 - 2xy$

↑ EXAMPLE

$f_x = \frac{\partial}{\partial x}(f) = \frac{\partial}{\partial x}(2x^2 - y^3 - 2xy)$

$= 4x - 2y$

$f_y = \frac{\partial}{\partial y}(f) = \frac{\partial}{\partial y}(2x^2 - y^3 - 2xy)$

$= -3y^2 - 2x$

find the critical points....

$4x - 2y = 0 \rightarrow 4x = 2y \rightarrow y = 2x$

$-3y^2 - 2x = 0$

$-3(2x)^2 - 2x = 0$

$-3(4x^2) - 2x = 0$

$-12x^2 - 2x = 0$

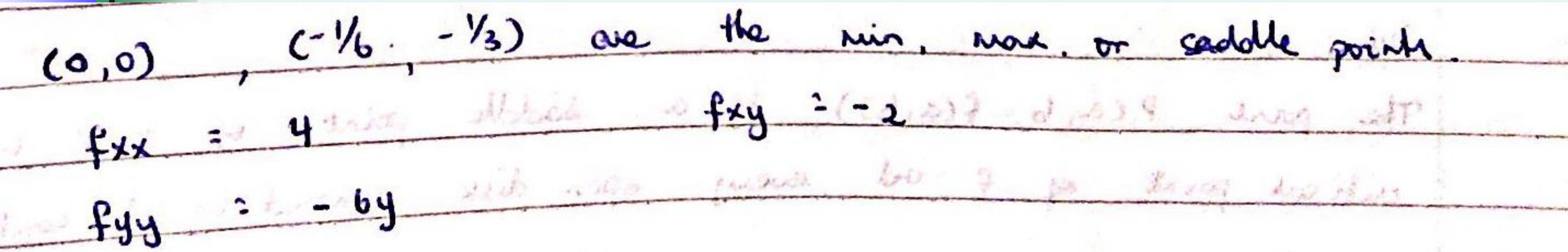
$-2x(6x+1) = 0$

$\rightarrow -2x = 0 \rightarrow x = 0$, $6x + 1 = 0 \rightarrow x = -1/6$

$\therefore y = 2(0) = 0$, $y = 2(-1/6)$

$= -2/6 = -1/3$

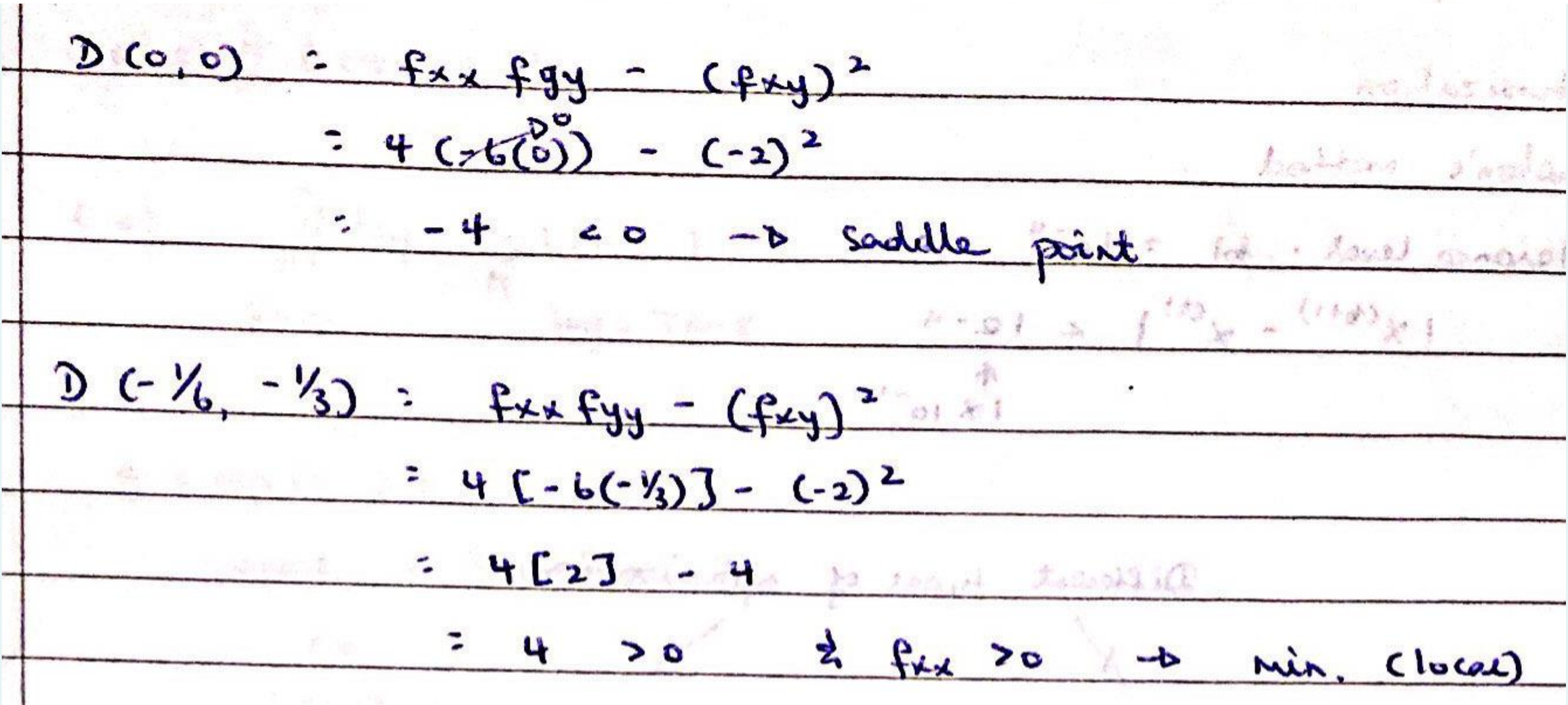
Continuation of Problem



$(0,0)$, $(-1/6, -1/3)$ are the min, max, or saddle points.

$f_{xx} = 4$ $\quad f_{xy} = -2$

$f_{yy} = -6y$



$$D(0,0) = f_{xx} f_{yy} - (f_{xy})^2$$

$$= 4(\overset{0}{\cancel{-6(0)}}) - (-2)^2$$

$$= -4 < 0 \rightarrow \text{saddle point}$$

$$D(-1/6, -1/3) = f_{xx} f_{yy} - (f_{xy})^2$$

$$= 4[-6(-1/3)] - (-2)^2$$

$$= 4[2] - 4$$

$$= 4 > 0 \ \& \ f_{xx} > 0 \rightarrow \text{min. (local)}$$

The EM Algorithm

- The EM algorithm is a very general iterative algorithm for parameter estimation by maximum likelihood when some of the random variables involved are not observed i.e., considered missing or incomplete. The EM algorithm formalizes an intuitive idea for obtaining parameter estimates when some of the data are missing:

i. replace missing values by estimated values,

ii. estimate parameters.

iii. Repeat step (i) using estimated parameter values as true values, and step (ii) using estimated values as \observed" values, iterating until convergence.

The EM Algorithm – the pioneers

- This idea has been in use for many years before Orchard and Woodbury (1972) in their missing information principle provided the theoretical foundation of the underlying idea.
- The term EM was introduced in Dempster, Laird, and Rubin (1977) where proof of general results about the behavior of the algorithm was first given as well as a large number of applications.

The EM Algorithm – to estimate parameters

The EM Algorithm consists of two steps:

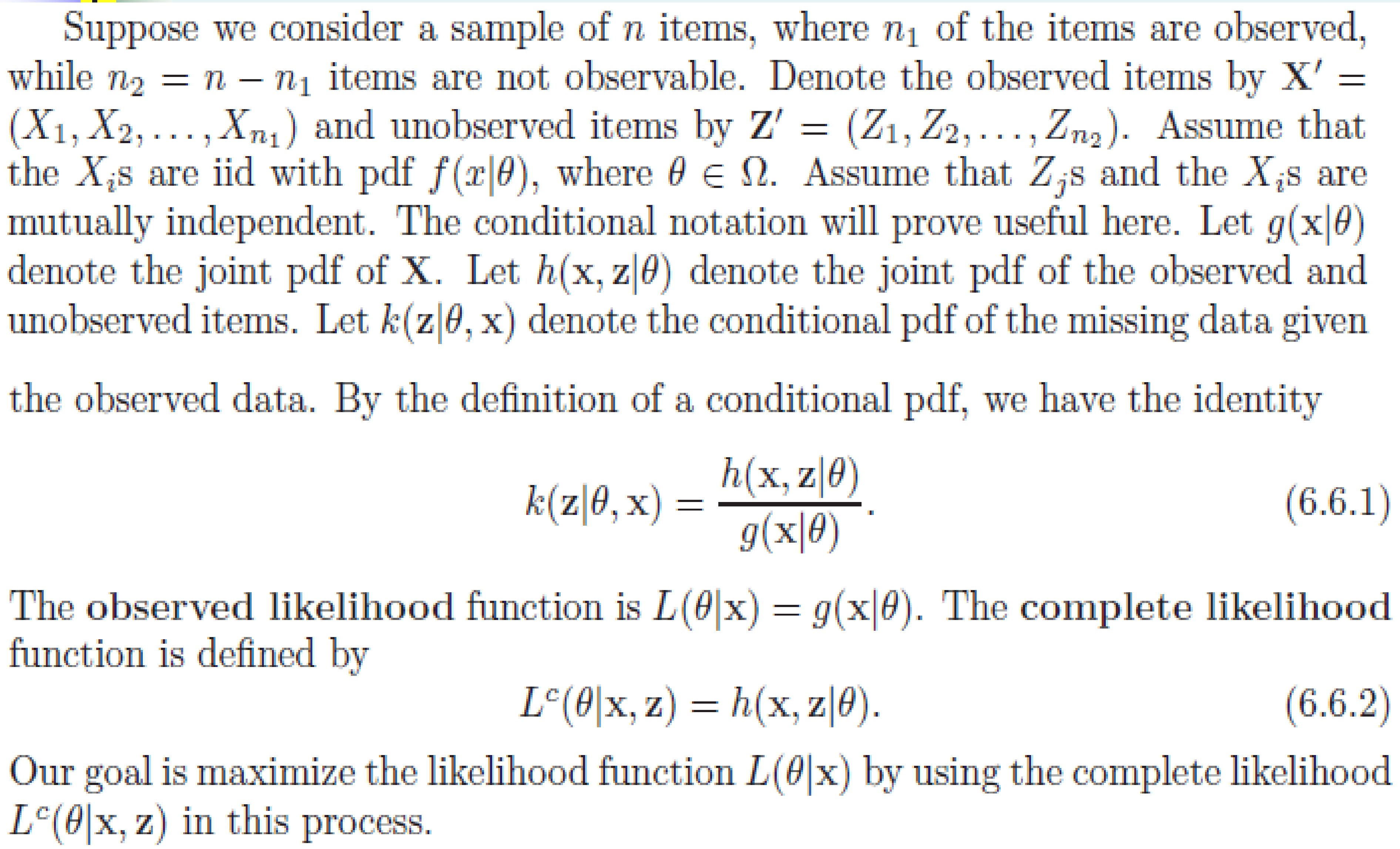
- An E step (Expectation Step)
- An M step (Maximization Step)

An estimation problem can be framed in terms of a missing data problem.

The Expectation Step “estimates” the missing data

Then, the missing data is used in the Maximization step to estimate the parameters.

The EM Algorithm - intro



Suppose we consider a sample of n items, where n_1 of the items are observed, while $n_2 = n - n_1$ items are not observable. Denote the observed items by $\mathbf{X}' = (X_1, X_2, \ldots, X_{n_1})$ and unobserved items by $\mathbf{Z}' = (Z_1, Z_2, \ldots, Z_{n_2})$. Assume that the X_is are iid with pdf $f(x|\theta)$, where $\theta \in \Omega$. Assume that Z_js and the X_is are mutually independent. The conditional notation will prove useful here. Let $g(\mathbf{x}|\theta)$ denote the joint pdf of $\mathbf{X}$. Let $h(\mathbf{x}, \mathbf{z}|\theta)$ denote the joint pdf of the observed and unobserved items. Let $k(\mathbf{z}|\theta, \mathbf{x})$ denote the conditional pdf of the missing data given

the observed data. By the definition of a conditional pdf, we have the identity

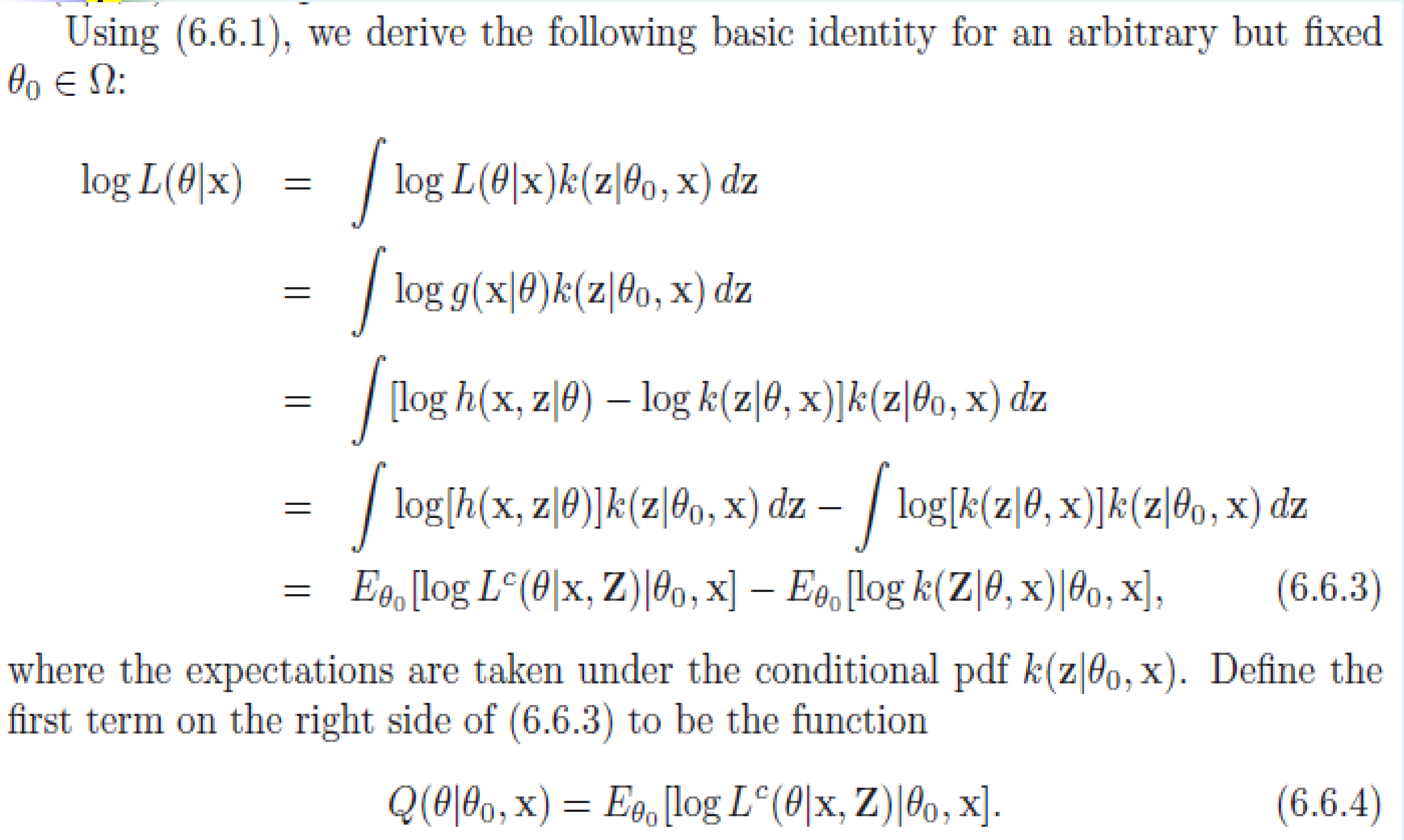
$$k(\mathbf{z}|\theta, \mathbf{x}) = \frac{h(\mathbf{x}, \mathbf{z}|\theta)}{g(\mathbf{x}|\theta)}. \tag{6.6.1}$$

The observed likelihood function is $L(\theta|\mathbf{x}) = g(\mathbf{x}|\theta)$. The complete likelihood function is defined by

$$L^c(\theta|\mathbf{x}, \mathbf{z}) = h(\mathbf{x}, \mathbf{z}|\theta). \tag{6.6.2}$$

Our goal is maximize the likelihood function $L(\theta|\mathbf{x})$ by using the complete likelihood $L^c(\theta|\mathbf{x}, \mathbf{z})$ in this process.

The EM Algorithm - intro



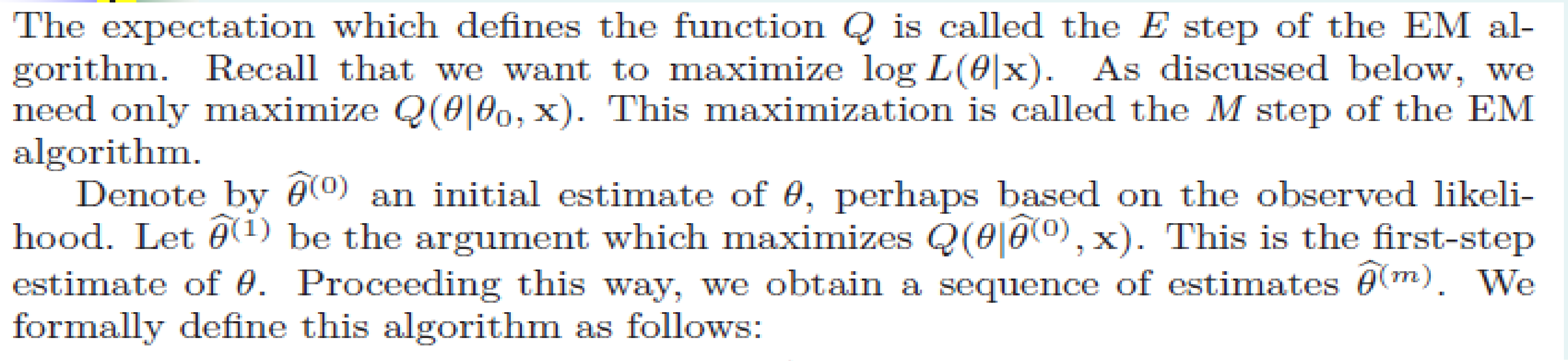
Using (6.6.1), we derive the following basic identity for an arbitrary but fixed $\theta_0 \in \Omega$:

$$\begin{aligned}
\log L(\theta|\mathbf{x}) &= \int \log L(\theta|\mathbf{x}) k(\mathbf{z}|\theta_0, \mathbf{x})\, d\mathbf{z} \\
&= \int \log g(\mathbf{x}|\theta) k(\mathbf{z}|\theta_0, \mathbf{x})\, d\mathbf{z} \\
&= \int [\log h(\mathbf{x}, \mathbf{z}|\theta) - \log k(\mathbf{z}|\theta, \mathbf{x})] k(\mathbf{z}|\theta_0, \mathbf{x})\, d\mathbf{z} \\
&= \int \log[h(\mathbf{x}, \mathbf{z}|\theta)] k(\mathbf{z}|\theta_0, \mathbf{x})\, d\mathbf{z} - \int \log[k(\mathbf{z}|\theta, \mathbf{x})] k(\mathbf{z}|\theta_0, \mathbf{x})\, d\mathbf{z} \\
&= E_{\theta_0}[\log L^c(\theta|\mathbf{x}, \mathbf{Z})|\theta_0, \mathbf{x}] - E_{\theta_0}[\log k(\mathbf{Z}|\theta, \mathbf{x})|\theta_0, \mathbf{x}], \qquad (6.6.3)
\end{aligned}$$

where the expectations are taken under the conditional pdf $k(\mathbf{z}|\theta_0, \mathbf{x})$. Define the first term on the right side of (6.6.3) to be the function

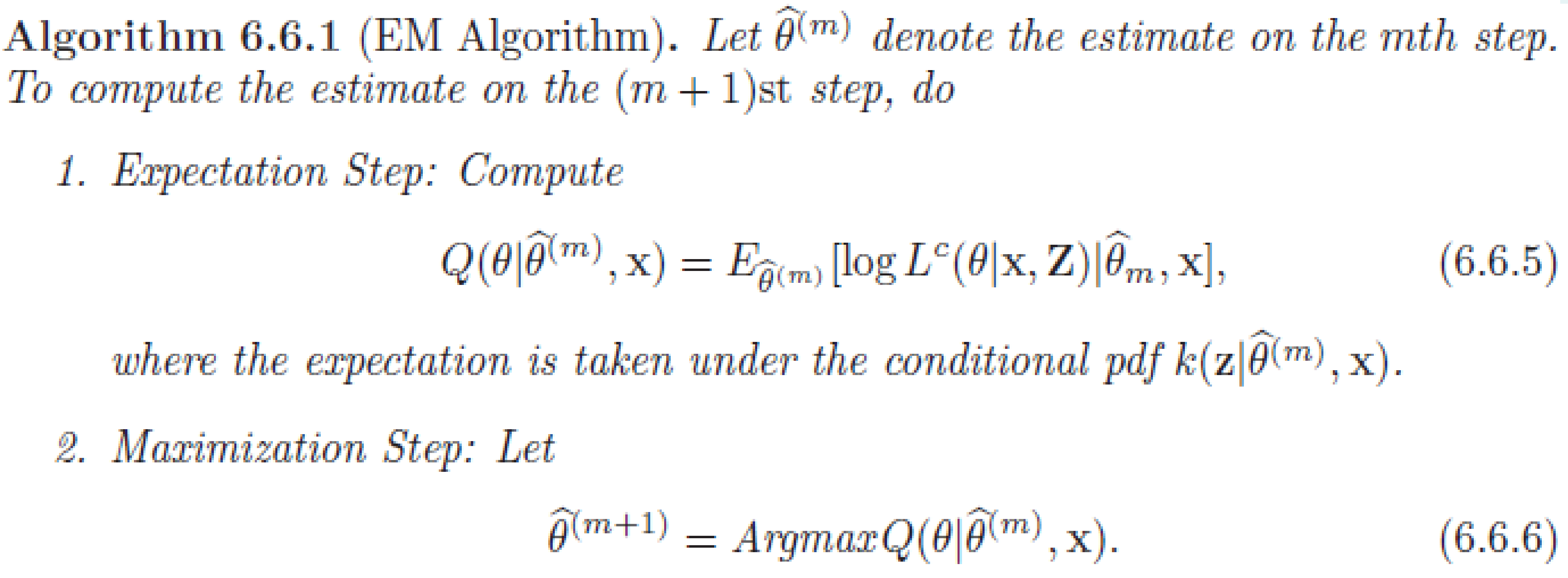
$$Q(\theta|\theta_0, \mathbf{x}) = E_{\theta_0}[\log L^c(\theta|\mathbf{x}, \mathbf{Z})|\theta_0, \mathbf{x}]. \qquad (6.6.4)$$

The EM Algorithm - intro



The expectation which defines the function Q is called the E step of the EM algorithm. Recall that we want to maximize $\log L(\theta|\mathrm{x})$. As discussed below, we need only maximize $Q(\theta|\theta_0, \mathrm{x})$. This maximization is called the M step of the EM algorithm.

Denote by $\widehat{\theta}^{(0)}$ an initial estimate of θ, perhaps based on the observed likelihood. Let $\widehat{\theta}^{(1)}$ be the argument which maximizes $Q(\theta|\widehat{\theta}^{(0)}, \mathrm{x})$. This is the first-step estimate of θ. Proceeding this way, we obtain a sequence of estimates $\widehat{\theta}^{(m)}$. We formally define this algorithm as follows:



Algorithm 6.6.1 (EM Algorithm). *Let $\widehat{\theta}^{(m)}$ denote the estimate on the mth step. To compute the estimate on the $(m+1)$st step, do*

1. *Expectation Step: Compute*

$$Q(\theta|\widehat{\theta}^{(m)}, \mathrm{x}) = E_{\widehat{\theta}^{(m)}}[\log L^c(\theta|\mathrm{x}, \mathbf{Z})|\widehat{\theta}_m, \mathrm{x}], \tag{6.6.5}$$

   where the expectation is taken under the conditional pdf $k(\mathbf{z}|\widehat{\theta}^{(m)}, \mathrm{x})$.

2. *Maximization Step: Let*

$$\widehat{\theta}^{(m+1)} = Argmax Q(\theta|\widehat{\theta}^{(m)}, \mathrm{x}). \tag{6.6.6}$$

The EM Algorithm

Under strong assumptions, it can be shown that $\widehat{\theta}^{(m)}$ converges in probability to the maximum likelihood estimate, as $m \to \infty$. We will not show these results, but as the next theorem shows, $\widehat{\theta}^{(m+1)}$ always increases the likelihood over $\widehat{\theta}^{(m)}$.

Theorem 6.6.1. *The sequence of estimates $\widehat{\theta}^{(m)}$, defined by Algorithm 6.6.1, satisfies*

$$L(\widehat{\theta}^{(m+1)}|\mathbf{x}) \geq L(\widehat{\theta}^{(m)}|\mathbf{x}). \tag{6.6.7}$$

Proof: Because $\widehat{\theta}^{(m+1)}$ maximizes $Q(\theta|\widehat{\theta}^{(m)}, \mathbf{x})$, we have

$$Q(\widehat{\theta}^{(m+1)}|\widehat{\theta}^{(m)}, \mathbf{x}) \geq Q(\widehat{\theta}^{(m)}|\widehat{\theta}^{(m)}, \mathbf{x});$$

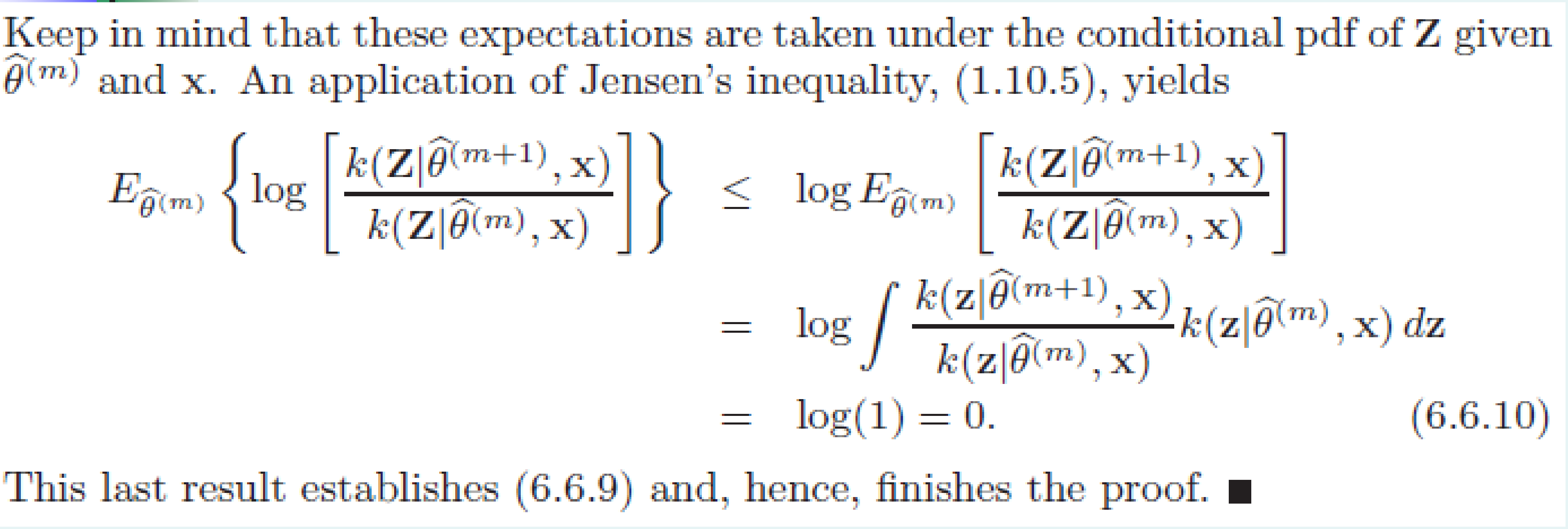
that is,

$$E_{\widehat{\theta}^{(m)}}[\log L^c(\widehat{\theta}^{(m+1)}|\mathbf{x}, \mathbf{Z})] \geq E_{\widehat{\theta}^{(m)}}[\log L^c(\widehat{\theta}^{(m)}|\mathbf{x}, \mathbf{Z})], \tag{6.6.8}$$

where the expectation is taken under the pdf $k(\mathbf{z}|\widehat{\theta}^{(m)}, \mathbf{x})$. By expression (6.6.3), we can complete the proof by showing that

$$E_{\widehat{\theta}^{(m)}}[\log k(\mathbf{Z}|\widehat{\theta}^{(m+1)}, \mathbf{x})] \leq E_{\widehat{\theta}^{(m)}}[\log k(\mathbf{Z}|\widehat{\theta}^{(m)}, \mathbf{x})]. \tag{6.6.9}$$

The EM Algorithm – continued and Jensen's Inequality

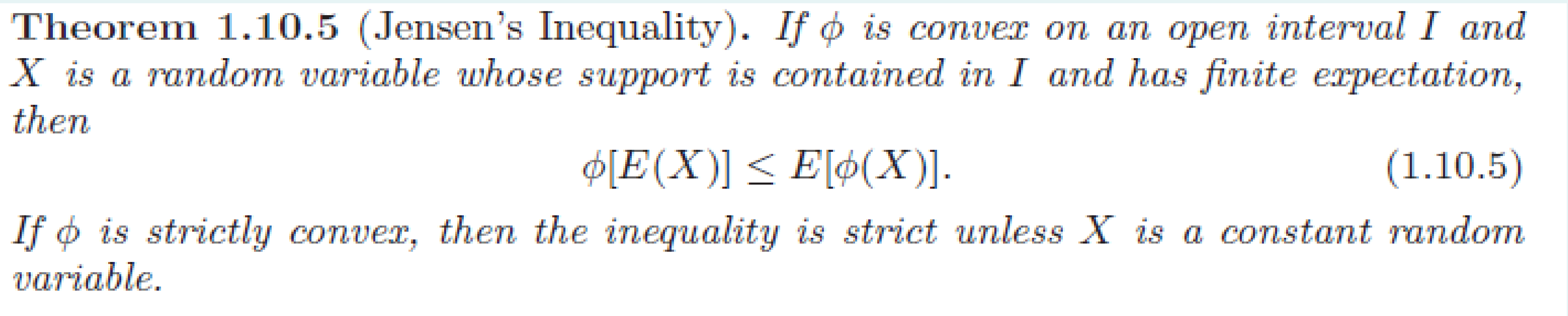


Keep in mind that these expectations are taken under the conditional pdf of $\mathbf{Z}$ given $\widehat{\theta}^{(m)}$ and x. An application of Jensen's inequality, (1.10.5), yields

$$\begin{aligned} E_{\widehat{\theta}^{(m)}}\left\{\log\left[\frac{k(\mathbf{Z}|\widehat{\theta}^{(m+1)},\mathbf{x})}{k(\mathbf{Z}|\widehat{\theta}^{(m)},\mathbf{x})}\right]\right\} &\leq \log E_{\widehat{\theta}^{(m)}}\left[\frac{k(\mathbf{Z}|\widehat{\theta}^{(m+1)},\mathbf{x})}{k(\mathbf{Z}|\widehat{\theta}^{(m)},\mathbf{x})}\right] \\ &= \log\int\frac{k(\mathbf{z}|\widehat{\theta}^{(m+1)},\mathbf{x})}{k(\mathbf{z}|\widehat{\theta}^{(m)},\mathbf{x})}k(\mathbf{z}|\widehat{\theta}^{(m)},\mathbf{x})\,d\mathbf{z} \\ &= \log(1) = 0. \qquad (6.6.10) \end{aligned}$$

This last result establishes (6.6.9) and, hence, finishes the proof. ■

Now, lets review Jensen's Inequality



Theorem 1.10.5 (Jensen's Inequality). *If ϕ is convex on an open interval I and X is a random variable whose support is contained in I and has finite expectation, then*

$$\phi[E(X)] \leq E[\phi(X)]. \qquad (1.10.5)$$

If ϕ is strictly convex, then the inequality is strict unless X is a constant random variable.

A review of the main idea of Convexity

Definition 1.10.1. *A function ϕ defined on an interval (a, b), $-\infty \leq a < b \leq \infty$, is said to be a* **convex** *function if for all x, y in (a, b) and for all $0 < \gamma < 1$,*

$$\phi[\gamma x + (1-\gamma)y] \leq \gamma\phi(x) + (1-\gamma)\phi(y). \tag{1.10.4}$$

We say ϕ is **strictly convex** *if the above inequality is strict.*

Depending on the existence of first or second derivatives of ϕ, the following theorem can be proved.

Theorem 1.10.4. *If ϕ is differentiable on (a, b), then*

(a) ϕ is convex if and only if $\phi'(x) \leq \phi'(y)$, for all $a < x < y < b$,

(b) ϕ is strictly convex if and only if $\phi'(x) < \phi'(y)$, for all $a < x < y < b$.

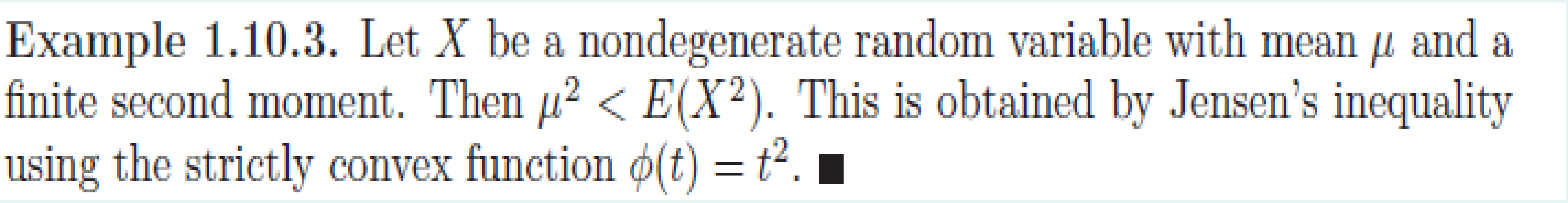
If ϕ is twice differentiable on (a, b), then

(a) ϕ is convex if and only if $\phi''(x) \geq 0$, for all $a < x < b$,

(b) ϕ is strictly convex if $\phi''(x) > 0$, for all $a < x < b$.

Two simple examples

Example 1

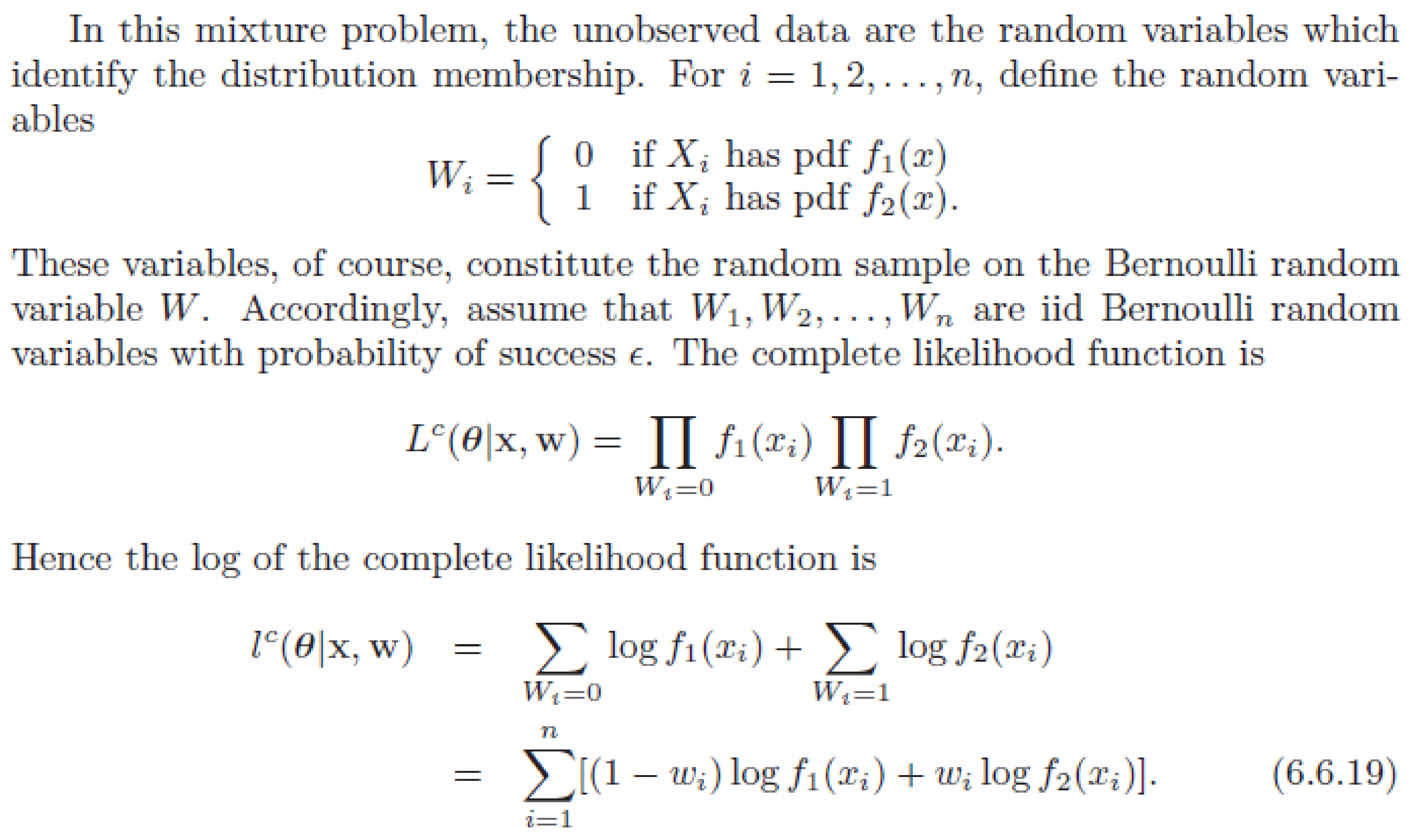


Example 1.10.3. Let X be a nondegenerate random variable with mean μ and a finite second moment. Then $\mu^2 < E(X^2)$. This is obtained by Jensen's inequality using the strictly convex function $\phi(t) = t^2$. ■

Example 2

If f(t) = log(t), then log is concave.

The EM Algorithm – the mixture model



In this mixture problem, the unobserved data are the random variables which identify the distribution membership. For $i = 1, 2, \ldots, n$, define the random variables

$$W_i = \begin{cases} 0 & \text{if } X_i \text{ has pdf } f_1(x) \\ 1 & \text{if } X_i \text{ has pdf } f_2(x). \end{cases}$$

These variables, of course, constitute the random sample on the Bernoulli random variable W. Accordingly, assume that $W_1, W_2, \ldots, W_n$ are iid Bernoulli random variables with probability of success ϵ. The complete likelihood function is

$$L^c(\boldsymbol{\theta}|\mathbf{x}, \mathbf{w}) = \prod_{W_i=0} f_1(x_i) \prod_{W_i=1} f_2(x_i).$$

Hence the log of the complete likelihood function is

$$\begin{aligned} l^c(\boldsymbol{\theta}|\mathbf{x}, \mathbf{w}) &= \sum_{W_i=0} \log f_1(x_i) + \sum_{W_i=1} \log f_2(x_i) \\ &= \sum_{i=1}^{n} [(1 - w_i) \log f_1(x_i) + w_i \log f_2(x_i)]. \qquad (6.6.19) \end{aligned}$$

The EM Algorithm - intro

For the E step of the algorithm, we need the conditional expectation of W_i given $\mathbf{x}$ under $\boldsymbol{\theta}_0$; that is,

$$E_{\boldsymbol{\theta}_0}[W_i|\boldsymbol{\theta}_0, \mathbf{x}] = P[W_i = 1|\boldsymbol{\theta}_0, \mathbf{x}].$$

An estimate of this expectation is the likelihood of x_i being drawn from distribution $f_2(x)$, which is given by

$$\gamma_i = \frac{\widehat{\epsilon} f_{2,0}(x_i)}{(1-\widehat{\epsilon}) f_{1,0}(x_i) + \widehat{\epsilon} f_{2,0}(x_i)}, \tag{6.6.20}$$

where the subscript 0 signifies that the parameters at $\boldsymbol{\theta}_0$ are being used. Expression (6.6.20) is intuitively evident; see McLachlan and Krishnan (1997) for more discussion. Replacing w_i by γ_i in expression (6.6.19), the M step of the algorithm is to maximize

$$Q(\theta|\theta_0, \mathbf{x}) = \sum_{i=1}^{n} [(1-\gamma_i) \log f_1(x_i) + \gamma_i \log f_2(x_i)]. \tag{6.6.21}$$

The EM Algorithm

This maximization is easy to obtain by taking partial derivatives of $Q(\boldsymbol{\theta}|\boldsymbol{\theta}_0, \mathbf{x})$ with respect to the parameters. For example,

$$\frac{\partial Q}{\partial \mu_1} = \sum_{i=1}^{n}(1-\gamma_i)(-1/2\sigma_1^2)(-2)(x_i-\mu_1).$$

Setting this to 0 and solving for μ_1 yields the estimate of μ_1. The estimates of the other mean and the variances can be obtained similarly. These estimates are

$$\begin{aligned}
\widehat{\mu}_1 &= \frac{\sum_{i=1}^n (1-\gamma_i)x_i}{\sum_{i=1}^n (1-\gamma_i)} \\
\widehat{\sigma}_1^2 &= \frac{\sum_{i=1}^n (1-\gamma_i)(x_i-\widehat{\mu}_1)^2}{\sum_{i=1}^n (1-\gamma_i)} \\
\widehat{\mu}_2 &= \frac{\sum_{i=1}^n \gamma_i x_i}{\sum_{i=1}^n \gamma_i} \\
\widehat{\sigma}_2^2 &= \frac{\sum_{i=1}^n \gamma_i (x_i-\widehat{\mu}_2)^2}{\sum_{i=1}^n \gamma_i}.
\end{aligned}$$

Since γ_i is an estimate of $P[W_i = 1|\boldsymbol{\theta}_0, \mathbf{x}]$, the average $n^{-1}\sum_{i=1}^n \gamma_i$ is an estimate of $\epsilon = P[W_i = 1]$. This average is our estimate of $\widehat{\epsilon}$.

Simple R function for the EM Algorithm for a mixture of two normal distributions

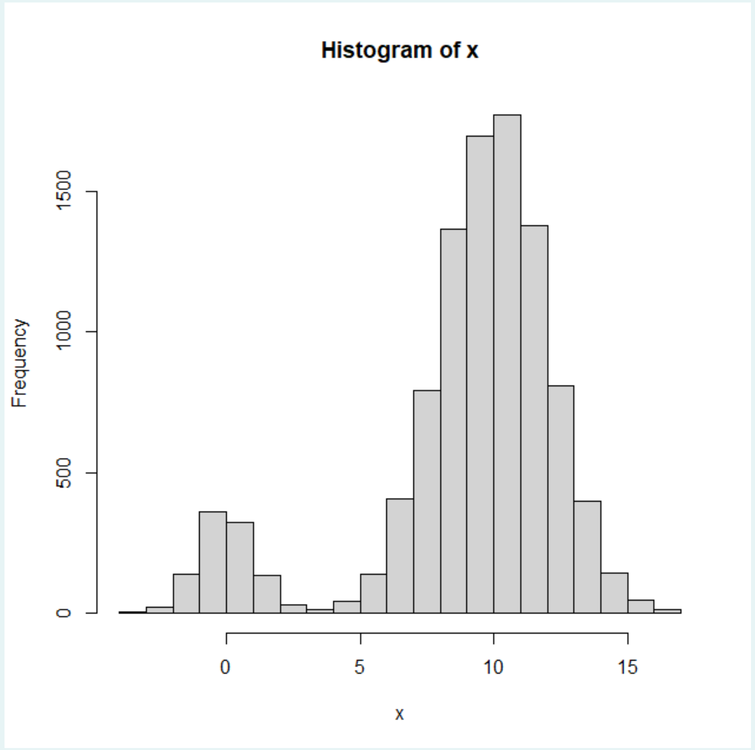
```
mixnormal = function(x,theta0)
{
part1=(1-theta0[5])*dnorm(x,theta0[1],theta0[3])
part2=theta0[5]*dnorm(x,theta0[2],theta0[4])
gam = part2/(part1+part2)
denom1 = sum(1 - gam)
denom2 = sum(gam)
mu1 = sum((1-gam)*x)/denom1
sig1 = sqrt(sum((1-gam)*((x-mu1)^2))/denom1)
mu2 = sum(gam*x)/denom2
sig2 = sqrt(sum(gam*((x-mu2)^2))/denom2)
p = mean(gam)
mixnormal = c(mu1,mu2,sig1,sig2,p)
mixnormal
}
```

Lets see how good this function actually is…

- Generate some data from a mixture (in other words, we need to supply the parameters).
  - Generate 1000 values from a normal distribution with mean 0 and variance 1
  - Generate 9000 values from a normal distribution with mean 10 and variance 4
  - Combine these two columns
  - So the parameters are $\mu_1 = 0, \mu_2 = 10, \sigma_1^2 = 1, \sigma_2^2 = 1, \pi = 0.1$
  - Now, run the code:

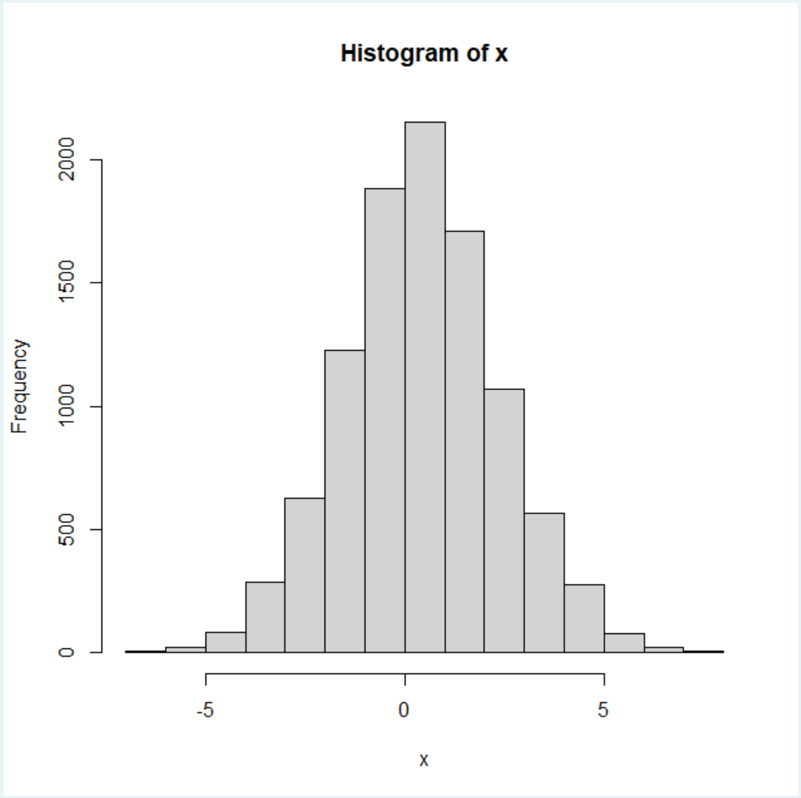
The code and Output

```
> x1 = rnorm(1000,0,1)
> x2 = rnorm(9000, 10, 2)
>
> x = c(x1,x2)
>
> hist(x)
>
> mixnormal = function(x,theta0)
+ {
+ part1=(1-theta0[5])*dnorm(x,theta0[1],theta0[3])
+ part2=theta0[5]*dnorm(x,theta0[2],theta0[4])
+ gam = part2/(part1+part2)
+ denom1 = sum(1 - gam)
+ denom2 = sum(gam)
+ mu1 = sum((1-gam)*x)/denom1
+ sig1 = sqrt(sum((1-gam)*((x-mu1)^2))/denom1)
+ mu2 = sum(gam*x)/denom2
+ sig2 = sqrt(sum(gam*((x-mu2)^2))/denom2)
+ p = mean(gam)
+ mixnormal = c(mu1,mu2,sig1,sig2,p)
+ mixnormal
+ }
>
> theta0 = c(1,2,3,4,0.5)
>
> mixnormal(x, theta0)
[1] 5.5855287 9.6965456 4.3915141 2.9211951 0.8336376
```



Simple R function for the EM Algorithm for a mixture of two normal distributions

```
> x1 = rnorm(1000,0,1)
> x2 = rnorm(9000, 0.5, 2)
> x = c(x1,x2)
>
> hist(x)
>
> mixnormal = function(x,theta0)
+ {
+ part1=(1-theta0[5])*dnorm(x,theta0[1],theta0[3])
+ part2=theta0[5]*dnorm(x,theta0[2],theta0[4])
+ gam = part2/(part1+part2)
+ denom1 = sum(1 - gam)
+ denom2 = sum(gam)
+ mu1 = sum((1-gam)*x)/denom1
+ sig1 = sqrt(sum((1-gam)*((x-mu1)^2))/denom1)
+ mu2 = sum(gam*x)/denom2
+ sig2 = sqrt(sum(gam*((x-mu2)^2))/denom2)
+ p = mean(gam)
+ mixnormal = c(mu1,mu2,sig1,sig2,p)
+ mixnormal
+ }
>
> theta0 = c(1,2,3,4,0.5)
> mixnormal(x, theta0)
[1] 0.3587278   0.4988542   1.8507168   2.0398650   0.4365510
```



Are there any good online libraries?

```
> install.packages("mclust")
> library(mclust)
> mod1 <- Mclust(X)
fitting ...                                                                                      |   0%
  |
  |==============================================================================| 100%
> summary(mod1, parameters = TRUE)
----------------------------------------------------
Gaussian finite mixture model fitted by EM algorithm
----------------------------------------------------
Mclust X (univariate normal) model with 1 component:
 log-likelihood       n        df             BIC             ICL
      -20799.82   10000        2       -41618.05       -41618.05
Clustering table:
    1
10000
Mixing probabilities:
1
1
Means:
[1] 0.4199001
Variances:
[1] 3.751239
>
```

How does the previous data we generated stack up?

```
> x1 = rnorm(1000,  0,  1)
> x2 = rnorm(9000, 10, 2)
> x = c(x1,x2)
> X = x
> mod1 <- Mclust(X)
fitting ...
  |==========================================================================| 100%
> summary(mod1, parameters = TRUE)
----------------------------------------------------
Gaussian finite mixture model fitted by EM algorithm
----------------------------------------------------
Mclust V (univariate, unequal variance) model with 2 components:
 log-likelihood     n       df      BIC          ICL
      -23635.92  10000      5    -47317.9    -47322.59
Clustering table:
   1       2
 999    9001
Mixing probabilities:
        1                  2
0.0999233         0.9000767
Means:
          1                  2
-0.02879448     10.03611897
Variances:
        1                  2
0.9393556          3.9966452
```