Optimization

Feng Li
feng.li@cufe.edu.cn

School of Statistics and Mathematics
Central University of Finance and Economics

April 30, 2014

Revision: April 30, 2014

Today we are going to learn...

1. Maximum Likelihood Estimation
2. Newton-Raphson Algorithm
3. Newton-Raphson Algorithm: The Matrix Version
4. Diagnosing Convergence

Likelihood function

- Given that $x_i \sim N(\mu, \sigma)$ for $i = 1, ..., n$, the **likelihood function** is

$$\prod_{i=1}^{n} f(x_i, \mu, \sigma)$$

- However the **log likelihood function** is more often used

$$\sum_{i=1}^{n} \log f(x_i, \mu, \sigma)$$

- Do you know why?

Maximum likelihood estimation for linear regression

- Assume you want to make a regression model

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

  where $\epsilon_i \sim N(0, \sigma^2)$
- What is the (log) likelihood function?
- What are the unknown parameters?
- How do we estimate the parameters?
  - Write down a likelihood function with respect to the unknown parameters.
  - Use an optimization algorithm to find the estimates of the unknown parameters.

The Likelihood function

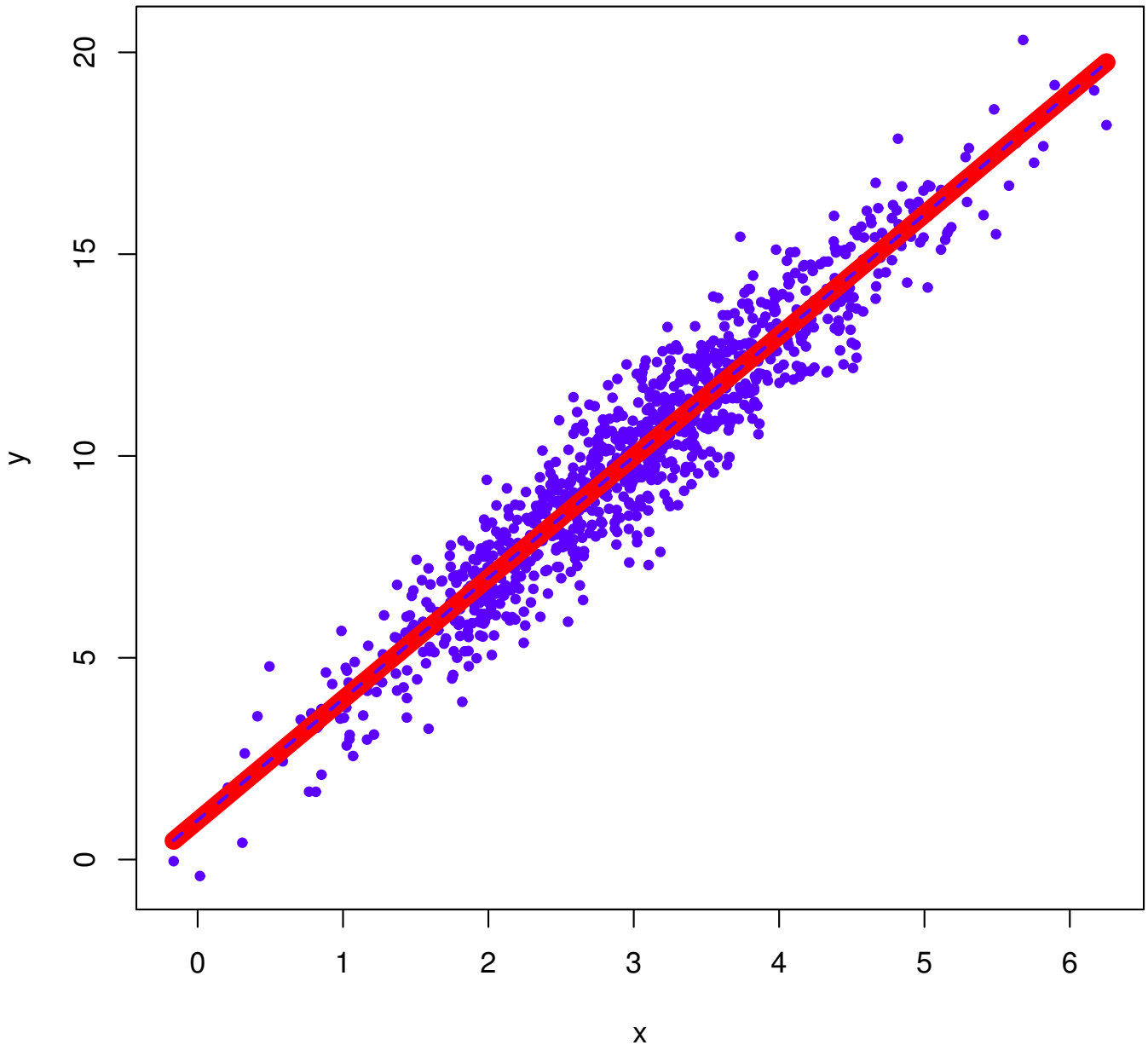
```
logNormLikelihood <- function(par, y, x)
    {
        beta0 <- par[1]
        beta1 <- par[2]
        sigma <- par[3]

        mean <- beta0 + x*beta1

        logDens <- dnorm(x = y, mean = mean,
                         sd = sigma, log = TRUE)
        loglikelihood <- sum(logDens)

        return(loglikelihood)
    }
```

Optimizing the likelihood function by using optim()

```
## Generate some data
beta0 <- 1
beta1 <- 3
sigma <- 1
n <- 1000
x <- rnorm(n, 3, 1)
y <- beta0 +x*beta1 + rnorm(n, mean = 0, sd = sigma)
plot(x, y, col = "blue", pch = 20)

## The optimization
optimOut <- optim(c(0, -1, 0.1), logNormLikelihood,
                  control = list(fnscale = -1),
                  x = x, y = y)
beta0Hat <- optimOut$par[1]
beta1Hat <- optimOut$par[2]
sigmaHat <- optimOut$par[3]
yHat <- beta0Hat + beta1Hat*x

plot(x, y, pch = 20, col = "blue")
points(sort(x), yHat[order(x)], type = "l", col = "red", lwd = 2)
```

Comparison with OLS

```
myLM <- lm(y~x)
myLMCoef <- myLM$coefficients
yHatOLS <- myLMCoef[1] + myLMCoef[2]*x

plot(x, y, pch = 20, col = "blue")
points(sort(x), yHat[order(x)], type = "l", col = "red", lwd = 10)
points(sort(x), yHatOLS[order(x)], type = "l",
       col = "blue", lty="dashed", lwd = 2, pch = 20)
```


y
0
5
10
15
20
0
1
2
3
4
5
6
x

Discussions

- How does the initial values help the optimization algorithm? Try with a few different numbers.
- Is there anything can be improved in the likelihood function?
- The importance of independence.

Lab 2

Submit your R code of this lab exercises online to the course page. The due date is **April 8 @ 24hrs**.

1. Generate 1000 random numbers ϵ_i from student t distribution with 3 degrees of freedom [`rt()`]. Generate a covariate x from $N(1, 1)$. Let $\beta_0 = 1, \beta_1 = -1$. Now generate y with the model

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

2. Use OLS method to estimate a linear model given y and x [`lm()`].
3. Estimate the model with the maximum likelihood method.
4. Compare the two results with a plot.

Newton-Raphson Algorithm

- When the goal is to solve an equation of the form $f(x) = 0$, a common approach is to use a **Newton-Raphson algorithm**, which produces a sequence such that

$$x_{n+1} = x_n - \left(\frac{\partial f(x)}{\partial x} \mid_{x=x_n} \right)^{-1} f(x_n)$$

- Note that $\frac{\partial f(x)}{\partial x}$ is the partial derivative for $f(x)$ with respect to x, also known as the **gradient**.
- x can be a scaler or a vector.

Example

- Assume we want to find the root of $x^2 - b = 0$ by using numerical method.
- First we have

$$f'(x) = 2x - b$$

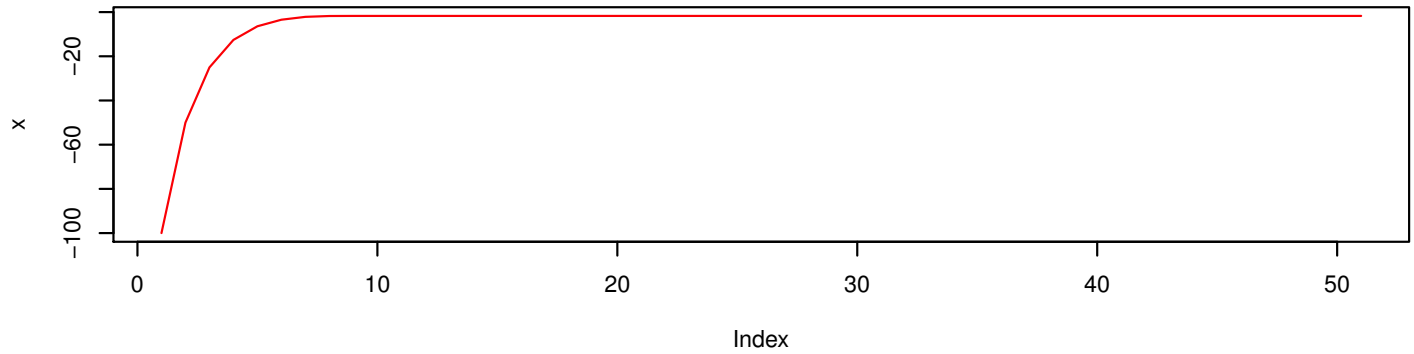
- Then

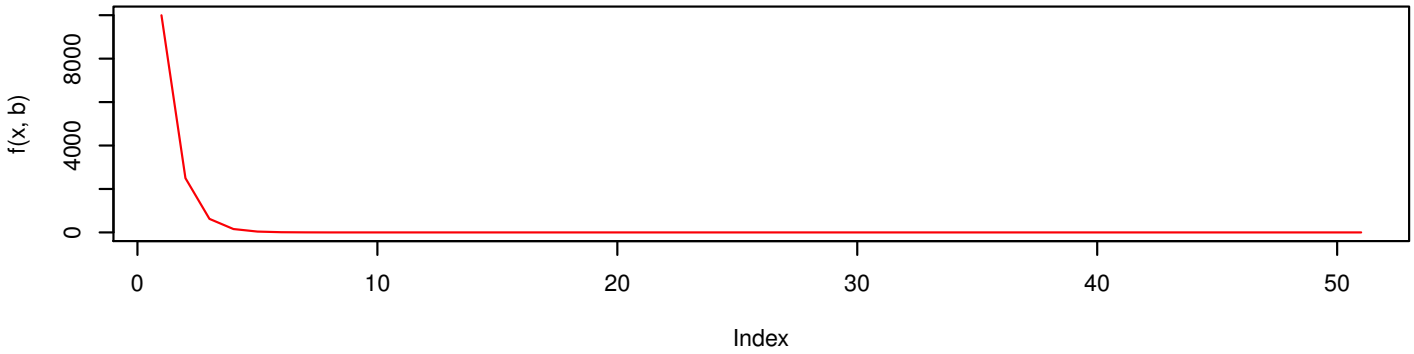
$$x_{n+1} = x_n - (2x)^{-1}(x_n^2 - b) = \frac{1}{2}(x_n + \frac{b}{x_n})$$

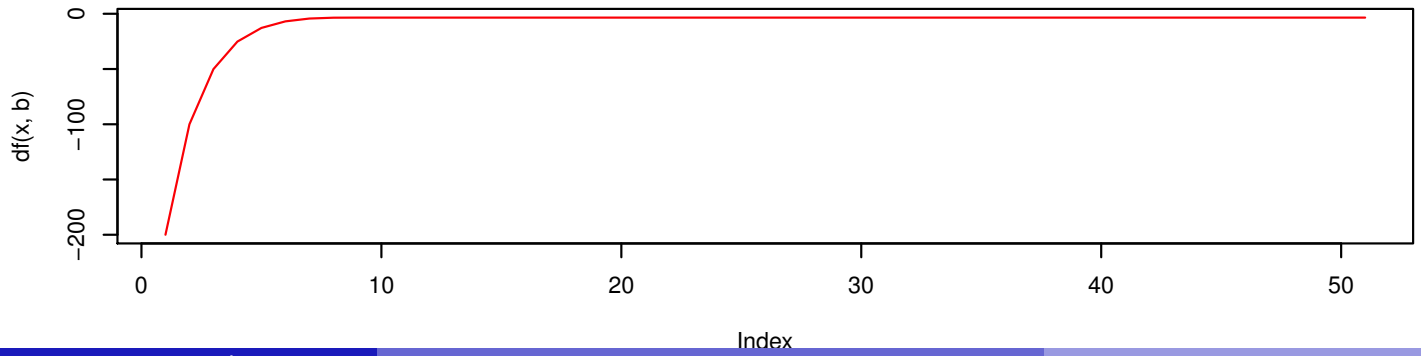
The realization in R

```
## The target function
f <- function(x, b) x^2 -b
## The gradient wrt x
df <- function(x, b) 2*x
## Loop to find the best x
n <- 50
x <- rep(NA, 1)
b <- 3
x[1] <- -100 # The initial value
for(i in 1:n)
{
    x[i+1] <- x[i]- (df(x[i], b))^(-1)*f(x[i], b)
}
## Check the convergence
par(mfrow = c(3, 1))
plot(x, type = "l", col = "red")
plot(f(x, b), type = "l", col = "red")
plot(df(x, b), type = "l", col = "red")
```

Diagnosis


x
−20
−60
−100
0
10
20
30
40
50
Index


f(x, b)
8000
4000
0
0
10
20
30
40
50
Index


df(x, b)
0
−100
−200
0
10
20
30
40
50
Index

Find the Maximum via Newton-Raphson Algorithm

- Find $\max f(x)$ can be viewed as to find the root of

$$\frac{\partial f(x)}{\partial x} = 0$$

- The algorithm is now as

$$x_{n+1} = x_n - \left(\frac{\partial^2 f(x)}{\partial x^2} \right)^{-1} \frac{\partial f(x)}{\partial x} \Big|_{x=x_n}$$

- The second order partial derivative is known as **Hessian**
- The Newton-Raphson algorithm cannot always find the maximum of $h(x)$, but rather converges to whatever mode is closest to the starting point.

Example

- Assume we want to find the maximum value of of $ax^2 + bx + c$ by using numerical method ($a < 0$).
- First we have

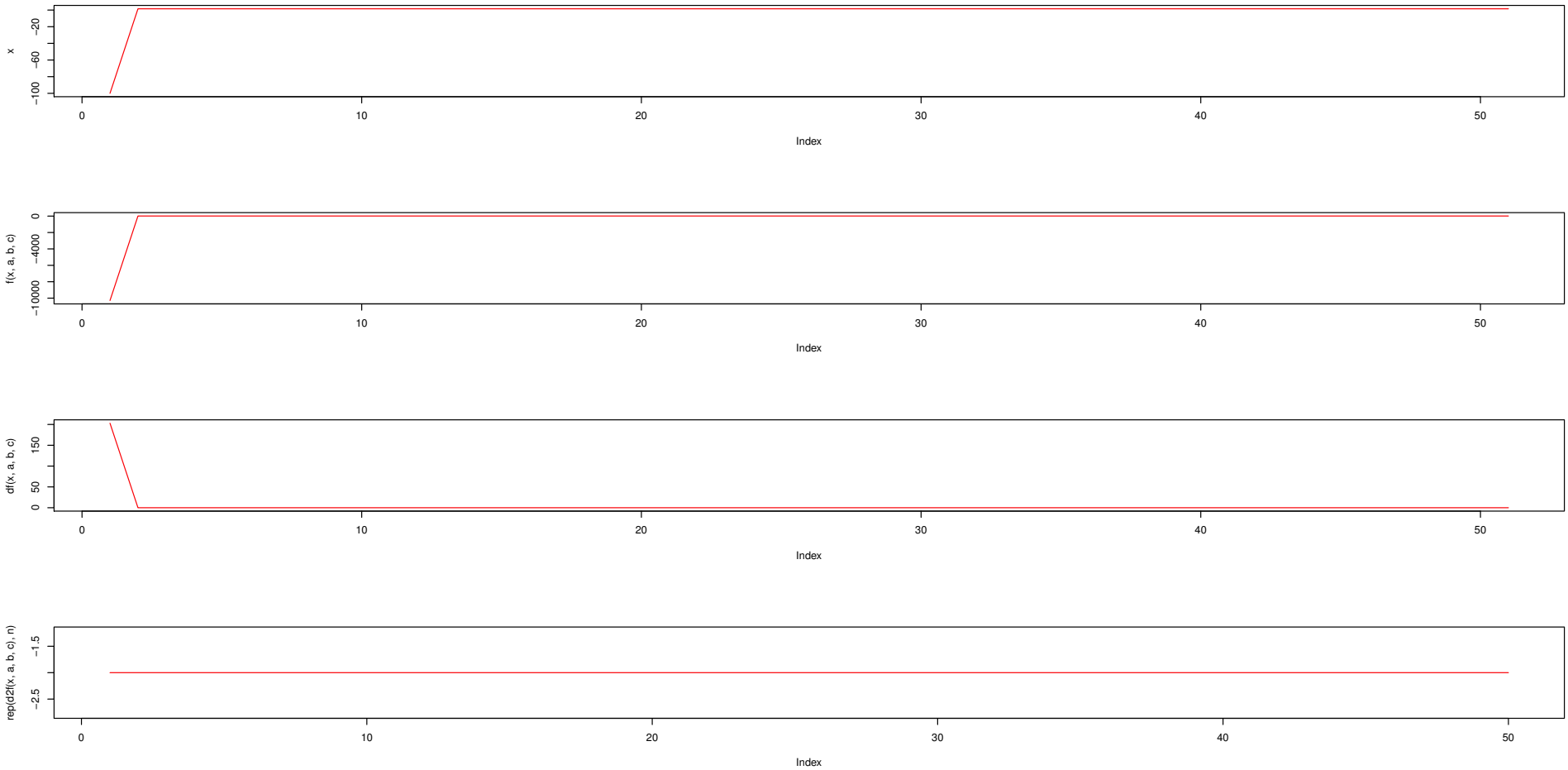
$$f'(x) = 2ax + b$$
$$f''(x) = 2a$$

- Then

$$x_{n+1} = x_n - (2a)^{-1}(2ax_n + b)$$

```
## The target function
f <- function(x, a, b, c) a*x^2 +b*x +c
## The gradient wrt x
df <- function(x, a, b, c) 2*a*x +b
## The Hessian wrt x
d2f <- function(x, a, b, c) 2*a
## Loop to find the best x
n <- 50
x <- rep(NA, 1)
a <- -1
b <- 3
c <- 10
x[1] <- -100 # The initial value
for(i in 1:n)
{
    x[i+1] <- x[i]- (d2f(x[i], a, b, c))^(-1)*df(x[i], a, b, c)
}
```

Diagnosis


x
−100
−60
−20
0
10
20
30
40
50
Index
f(x, a, b, c)
−10000
−4000
0
0
10
20
30
40
50
Index
df(x, a, b, c)
0
50
150
0
10
20
30
40
50
Index
rep(d2f(x, a, b, c), n)
−2.5
−1.5
0
10
20
30
40
50
Index

Newton-Raphson Algorithm: The Matrix Version

- The Newton-Raphson algorithm for finding the maximum value of $f(\mathbf{x})$ where x now is a $p \times 1$ vector can be naturally extended as

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \left(\frac{\partial^2 f(\mathbf{x})}{\partial \mathbf{x} \partial \mathbf{x}'} \right)^{-1} \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} \mid_{\mathbf{x}=\mathbf{x}_n}$$

- Note the dimensions of each elements.

Preliminaries for matrix derivatives

The linear regression model, a revisit

- Consider the linear regression model with multiple covariates,

$$y_i = \beta_0 + \beta_1 x_1 + ... + \beta_p x_p + \epsilon_i$$

  where $\epsilon_i \sim N(0, \sigma^2)$

- What is the gradient and Hessian matrix for the log likelihood ($\mathcal{L}$) with respect to the parameter vector $\boldsymbol{\beta} = (\beta_0, ..., \beta_p)$?

$$\frac{\partial log\mathcal{L}}{\partial \boldsymbol{\beta}} = ?$$

$$\frac{\partial^2 log\mathcal{L}}{\partial \boldsymbol{\beta} \boldsymbol{\beta}'} = ?$$

Vector Newton-Raphson Algorithm: The logit model

↛ Estimate logit model with ungrouped (individual) data

- **The idea**: using maximum likelihood method with binomial distribution.
- One owns a house ($Y = 1$) or do not own a house ($Y = 0$) can be represented with **Bernoulli distribution**

$$\Pr(y; p) = p^y (1-p)^{1-y} \quad \text{for } y \in \{0, 1\}.$$

- The log likelihood function is as follows

$$l(\beta) = \sum_{n=1}^{N} \{y_i \log P_i + (1 - y_i) \log(1 - P_i)\}$$

where

$$P_i = \frac{1}{1 + \exp(-(\beta_1 + \beta_2 X_{2i} + ... + \beta_p X_{pi}))}$$

- Note that the sum of n Bernoulli samples will be **binomial** distributed.
- To obtain $\hat{\beta}$, use Newton-Raphson algorithm

$$\beta^{new} = \beta^{old} - \left(\frac{\partial^2 l(\beta)}{\partial \beta \partial \beta'} \right)^{-1} \frac{\partial l(\beta)}{\partial \beta} |_{\beta = \beta^{old}}$$

Diagnosing Convergence

- We need a **stopping rule** to guarantee that the number of iterations is sufficient.
- Criteria
  - Convergence to the Stationary Distribution
  - Convergence of Averages
  - Convergence to iid Sampling

Convergence in multiple chains

- Many multiple-chain convergence diagnostics are quite elaborate.
- The performances of these parallel methods require a degree of a priori knowledge on the distribution in order to construct an initial distribution.
  - An initial distribution which is too concentrated around a local mode does not contribute significantly more than a single chain to the exploration
  - Moreover, slow algorithms, Gibbs sampling used in highly nonlinear setups, usually favor single chains.
- It is somewhat of an illusion to think we can control the flow of a Markov chain and assess its convergence behavior from a few realizations of this chain.

Monitoring Convergence of Distribution

- A natural empirical approach to convergence control is to draw pictures of the output of simulated chains, in order to detect deviant or non-stationary behaviors. However, this plot is only useful for strong non-stationarities of the chain.
- Tests for non-stationary checking.
  - Autocorrelation functions.
- Another approach to convergence monitoring is to assess how much of the support of the target distribution has been explored by the chain via an evaluation of

$$\int_A f(x)dx \approx \sum_{t=1}^{T-1} (\theta(t+1) - \theta_t) f(\theta_t)$$

when $f(x)$ is a one-dimensional density, the above converges to 1.

Monitoring Convergence of Average

- Graphical outputs can detect obvious problems of convergence of the empirical average.
- One may use cumulative sums (CUSUM), graphing the partial differences

$$D_T^* \sum_{t=1}^{i} (h(\theta_t) - \frac{1}{T} \sum_{t=1}^{T} h(\theta_t))$$

Effective Sample Size

- The standard approach to restore to the **effective sample size** which gives the size of an iid sample with the same variance as the current sample and thus indicates the loss in efficiency due to the use of a Markov chain. This value is computed as

$$\frac{T}{1 + 2\sum_{t=1}^{\infty} \mathrm{Corr}(h(\theta_0), h(\theta_t))}$$

where the denominator is the measurement of efficiency (**inefficiency factor**)

Further Suggested Reading

Monte Carlo Statistical Methods Book by Christian P Robert and George Casella. (2004 edition)