The Study on Three Popular Optimization Methods Applied on Log-likelihood Function and The MaxLik Package in R

Xiran Wang
xiran@emails.edu
Drpartemnt of Statistics
12/04/2019

Outlines

- Part I: MaxLik(in R) Implementation
  - Functionalities /simulation comparison
- Part II: Three optimization methods for log-likelihood function
  - Newtown Raphson (N-R)
  - BHHH
  - Nelder-Mead (N-M)
  - Simulation results and comparisons

Motivation

```
################## gamma example
rm(list = ls()) # clear the workspace
ax=rgamma(100, 1, 2) # generate data

## log-likelihood of gamma sample, y is the data vector
loglkgam<- function(theta, y){ -length(y)*log(gamma(theta[1]))+
    length(y)*theta[1]*log(theta[2])+
    (theta[1]-1)*sum(log(y))-sum(y)*theta[2]}

## try maxLik for comparison purpose
res1<-maxLik(loglkgam, start=c(.5, 0.5), y=ax)
res1
```

```
## Maximum Likelihood estimation
## Newton-Raphson maximisation, 6 iterations
## Return code 2: successive function values within tolerance limit
## Log-Likelihood: -35.94528 (2 free parameter(s))
## Estimate(s): 1.19331 2.243339
```

```
res1$hess
```

```
##            [,1]      [,2]
## [1,] -127.71295  44.56524
## [2,]   44.56524 -23.73213
```

```
se1=sqrt(diag(solve(-res1$hess)))
```

```
logLikHess <- function(param) {
  alpha <- param[1]
  beta <- param[2]
  n <- length(ax)
  logLikHessValues <- matrix( 0, nrow = 2, ncol = 2)
  logLikHessValues[1, 1] <- -n*trigamma(alpha)
  logLikHessValues[1, 2] <- n/beta
  logLikHessValues[2, 1] <- n/beta
  logLikHessValues[2, 2] <- -n*alpha/beta^2
  return(logLikHessValues)}

logLikHess(c(res1$est[1],res1$est[2]))
```

```
##            [,1]      [,2]
## [1,] -123.77011  38.38228
## [2,]   38.38228 -17.98085
```

maxLik (in R)

- The maxLik package is designed in two layers.
  - The first (innermost) is the optimization (maximization) layer
  - The second layer is the likelihood maximization layer
- Output Comparisons:
  - R simulation vs. maxLik basic
  - maxLik basic vs. maxLik basic + gradients
  - maxLik basic + gradients vs. maxLik basic + gradients + hessian

```
maxLik(logLik, grad = NULL, hess = NULL, start, method,
constraints=NULL, ...)
```

Samples

All simulations(x's) are from $N(\mu = 1, \sigma = 2)$, n=100.

$$f(x_i) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x_i - \mu)^2}{2\sigma^2}\right)$$

$$L(x; \mu, \sigma) = \prod_{i=1}^{N} \frac{1}{\sigma\sqrt{2\pi}} exp(-\frac{(x_i - \mu)^2}{2\sigma^2})$$

$$log(L(x; \mu, \sigma)) = -\frac{1}{2} Nlog(2\pi) - Nlog(\sigma) - \frac{1}{2}\sum_{i=1}^{N} \frac{(x_i - \mu)^2}{\sigma^2}$$

R simulation vs. maxLik basic

- From theoretical maximum likelihood estimation method:

$$\mu = \frac{\sum_{i=1}^{N} x_i}{N} \qquad \sigma = \sqrt{\frac{\sum_{i=1}^{N} (X_i - \mu)^2}{N}}$$

```
mean(x)
```

```
## [1] 1.009759
```

```
sqrt( sum( ( x-mean(x) )^2 ) /length(x) )
```

```
## [1] 2.203676
```

- Estimations of μ and σ from maxLik:

```
logLikFun <- function(param) {
  mu <- param[1]
  sigma <- param[2]
  sum(dnorm(x, mean = mu, sd = sigma, log = TRUE)) }

mle <- maxLik(logLik = logLikFun, start = c(mu = 0, sigma = 1))
summary(mle)
```

```
## --------------------------------------------
## Maximum Likelihood estimation
## Newton-Raphson maximisation, 7 iterations
## Return code 2: successive function values within tolerance limit
## Log-Likelihood: -220.9066
## 2  free parameters
## Estimates:
##       Estimate Std. error t value  Pr(> t)
## mu      1.0098     0.2204    4.58 4.64e-06 ***
## sigma   2.2037     0.1558   14.14  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## --------------------------------------------
```

maxLik basic vs. maxLik basic + gradients

```
logLikGrad <- function(param) {
  mu <- param[1]
  sigma <- param[2]
  N <- length(x)
  logLikGradValues <- numeric(2)
  logLikGradValues[1] <- sum((x - mu)/sigma^2)
  logLikGradValues[2] <- -N/sigma + sum((x - mu)^2/sigma^3)
  return(logLikGradValues)
   }
mleGrad <- maxLik(logLik = logLikFun,
                  grad = logLikGrad, start = c(mu = 0, sigma = 1))
summary(mleGrad)
```

```
## --------------------------------------------
## Maximum Likelihood estimation
## Newton-Raphson maximisation, 7 iterations
## Return code 2: successive function values within tolerance limit
## Log-Likelihood: -220.9066
## 2  free parameters
## Estimates:
##       Estimate Std. error t value Pr(> t)
## mu      1.0098     0.2204   4.582 4.6e-06 ***
## sigma   2.2037     0.1558  14.142 < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## --------------------------------------------
```

```
all.equal(logLik(mleGrad), logLik(mle))
```

```
## [1] TRUE
```

```
all.equal(coef(mleGrad), coef(mle))
```

```
## [1] TRUE
```

```
all.equal(vcov(mleGrad), vcov(mle)) #abs(a-b)/abs(a)
```

```
## [1] "Mean relative difference: 0.001079409"
```

```
vcov(mleGrad)
```

```
##                   mu        sigma
## mu    3.659469e-02 2.288827e-12
## sigma 2.288827e-12 1.829735e-02
```

```
vcov(mle)
```

```
##                    mu         sigma
## mu     3.657421e-02 -1.901935e-05
## sigma -1.901935e-05  1.829662e-02
```

maxLik basic + gradients vs. maxLik basic + gradients + hessian

```
logLikHess <- function(param) {
  mu <- param[1]
  sigma <- param[2]
  N <- length(x)
  logLikHessValues <- matrix(0, nrow = 2, ncol = 2)
  logLikHessValues[1, 1] <- -N/sigma^2
  logLikHessValues[1, 2] <- -2 * sum((x - mu)/sigma^3)
  logLikHessValues[2, 1] <- logLikHessValues[1, 2]
  logLikHessValues[2, 2] <- N/sigma^2 - 3 * sum((x - mu)^2/sigma^4)
  return(logLikHessValues)
}

mleHess <- maxLik(logLik = logLikFun, grad = logLikGrad,
                  hess = logLikHess, start = c(mu = 0, sigma = 1))
summary(mleHess)
```

```
## --------------------------------------------
## Maximum Likelihood estimation
## Newton-Raphson maximisation, 7 iterations
## Return code 1: gradient close to zero
## Log-Likelihood: -209.0059
## 2  free parameters
## Estimates:
##       Estimate Std. error t value  Pr(> t)
## mu      0.6447     0.1956   3.295 0.000983 ***
## sigma   1.9564     0.1383  14.142  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## --------------------------------------------
```

```
all.equal(list(logLik(mleHess), coef(mleHess),
               vcov(mleHess)),list(logLik(mleGrad), coef(mleGrad), vcov(mleGrad)))
```

```
## [1] TRUE
```

Optimization methods for log-likelihood function

- Numerical search algorithms
  - (1)Obtain initial values for the parameters, say θ_0
  - (2) Update the candidate value for θ_0 , and it is where the optimization method is applied.
  - (3) Determine when the optimum has been reached.

Samples

All simulations(x's) are from $N(\mu = 1, \sigma = 2)$, n=100.

$$f(x_i) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x_i - \mu)^2}{2\sigma^2}\right)$$

$$L(x; \mu, \sigma) = \prod_{i=1}^{N} \frac{1}{\sigma\sqrt{2\pi}} exp(-\frac{(x_i - \mu)^2}{2\sigma^2})$$

$$log(L(x; \mu, \sigma)) = -\frac{1}{2} Nlog(2\pi) - Nlog(\sigma) - \frac{1}{2}\sum_{i=1}^{N} \frac{(x_i - \mu)^2}{\sigma^2}$$

Newtown Raphson(N-R)

- Let g(u) to be a target function need found its roots by performing the Newtown Raphson algorithm can be described as follows:

$$u_{n+1} = u_n - \frac{g(u_n)}{g'(u_n)}$$

- Similarly, when g’(u) is the target function, the Newtown Raphson algorithm becomes to be:

$$u_{n+1} = u_n - \frac{g'(u_n)}{g''(u_n)}$$

Multi-dimension Newtown Raphson algorithm – Log-likelihood

$$u^{(n+1)} = u^{(n)} - [Hg(u^{(n)})]^{-1} \nabla g(u^{(n)})$$

Where

$$\nabla g(u) = \left[\frac{\partial g(u)}{\partial u_1} \quad \frac{\partial g(u)}{\partial u_2} \quad \cdots \frac{\partial g(u)}{\partial u_n} \right]^T$$

* g(u) is Log-likelihood

And

$$Hg(u) = \frac{d\mathbf{g}}{d\mathbf{u}} = \left[\frac{\partial \mathbf{g}}{\partial u_1} \cdots \frac{\partial \mathbf{g}}{\partial u_n} \right] = \begin{bmatrix} \frac{\partial^2 g(u)}{\partial u_1^2} & \frac{\partial^2 g(u)}{\partial u_1 \partial u_2} & \cdots & \frac{\partial^2 g(u)}{\partial u_1 \partial u_n} \\ \frac{\partial^2 g(u)}{\partial u_2 \partial u_2} & \frac{\partial^2 g(u)}{\partial u_2 \partial u_2} & \cdots & \frac{\partial^2 g(u)}{\partial u_2 \partial u_n} \\ \vdots & & \ddots & \vdots \\ \frac{\partial g(u)}{\partial u_n \partial u_1} & \frac{\partial^2 g(u)}{\partial u_n \partial u_2} & \cdots & \frac{\partial g(u)}{\partial u_n^2} \end{bmatrix}$$

Application on normal distribution (N-R)

Where $\nabla g(x) = \left[\sum_{i=1}^{N} \frac{(x_i-\mu)}{\sigma^2}, -\frac{N}{\sigma} + \sum_{i=1}^{N} \frac{(x_i-\mu)^2}{\sigma^3}\right]$ $Hg(u) = \begin{bmatrix} -\frac{N}{\sigma^2} & -2\sum_{i=1}^{N} \frac{(x_i-\mu)}{\sigma^3} \\ -2\sum_{i=1}^{N} \frac{(x_i-\mu)}{\sigma^3} & \frac{N}{\sigma^2} - 3\sum_{i=1}^{N} \frac{(x_i-\mu)^2}{\sigma^4} \end{bmatrix}$

```
mleNR=maxLik(logLik = logLikFun, grad = logLikGrad,
                    hess = logLikHess, start = c(mu = 0, sigma = 1))
summary(mleNR)
```

```
## --------------------------------------------
## Maximum Likelihood estimation
## Newton-Raphson maximisation, 7 iterations
## Return code 1: gradient close to zero
## Log-Likelihood: -206.7598
## 2  free parameters
## Estimates:
##       Estimate Std. error t value  Pr(> t)
## mu      0.9823     0.1913   5.135 2.82e-07 ***
## sigma   1.9130     0.1353  14.142  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## --------------------------------------------
```

BHHH

$$u^{(n+1)} = u^{(n)} + [\nabla g(u^{(n)}) \times (\nabla g(u^{(n)})^t]^{-1} \nabla g(u^{(n)})$$

where

$$\nabla g(u) = \left[\frac{\partial g(u)}{\partial u_1} \quad \frac{\partial g(u)}{\partial u_2} \quad \dots \frac{\partial g(u)}{\partial u_n} \right]^T$$

- This algorithm follows the Newton-Raphson approach but replaces the negative of the Hessian $-Hg(u_n)$ by $\nabla g(u) * \nabla g(u)^T$
- $\nabla g(u) * \nabla g(u)^T$ is an approximation of - hessian matrix.

Why BHHH works?

- $\nabla g(u) * \nabla g(u)^T$ an approximation of $-Hg(u_n)$, refer to asymptotic normality for MLEs:

Suppose $X_1, ..., X_n$ are iid from $L(x|\theta)$ (log-likeliohood):

$$\sqrt{n}(\theta_{mle} - \theta) \xrightarrow{d} N(0, v(\theta))$$

as n $\rightarrow$ inf, where the asymptotic variance

$$v(\theta) = \frac{1}{nI_n(\theta)}$$

Knowing that $I_n(\theta)$, the Fisher Information based on n observations, is given by

$$I_n(\theta) = E_\theta\{ [\frac{\partial}{\partial\theta} logL_n(X|\theta)]^2 \} = -E_\theta[\frac{\partial^2}{\partial\theta^2} logL_n(X|\theta)]$$

Application on normal distribution (BHHH)

$$\nabla g(x) = \left[\sum_{i=1}^{N} \frac{(x_i - \mu)}{\sigma^2}, -\frac{N}{\sigma} + \sum_{i=1}^{N} \frac{(x_i - \mu)^2}{\sigma^3} \right]^T$$

```
mleBHHH <- maxLik(logLik = logLikFun, grad = logLikGradInd,
                  start = c(mu = 0, sigma = 1), method = "BHHH")
summary(mleBHHH)
```

```
## --------------------------------------------
## Maximum Likelihood estimation
## BHHH maximisation, 11 iterations
## Return code 2: successive function values within tolerance limit
## Log-Likelihood: -206.7598
## 2  free parameters
## Estimates:
##       Estimate Std. error t value  Pr(> t)
## mu      0.9823     0.1913   5.135 2.82e-07 ***
## sigma   1.9130     0.1445  13.235  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## --------------------------------------------
```

Comparison: N-R vs. BHHH

- maxLik output comparison:

```
all.equal(logLik(mleBHHH), logLik(mleNR))
```

```
## [1] TRUE
```

```
all.equal(coef(mleBHHH), coef(mleNR))
```

```
## [1] "Mean relative difference: 2.1456e-06"
```

```
all.equal(vcov(mleBHHH), vcov(mleNR))
```

```
## [1] "Mean relative difference: 0.04879283"
```

```
vcov(mleBHHH)
```

```
##                    mu          sigma
## mu     0.0365957264 -0.0001098873
## sigma -0.0001098873  0.0208922627
```

```
vcov(mleNR)
```

```
##                    mu          sigma
## mu     3.659469e-02 -4.803543e-13
## sigma -4.803543e-13  1.829735e-02
```

- Variance Estimators for two methods:

N-R: $$var(\theta) = -E[Hg(u^{(n)})]^{-1}$$

BHHH: $$var(\theta) = E[\nabla g(u^{(n)}) \times (\nabla g(u^{(n)})^t]^{-1}$$

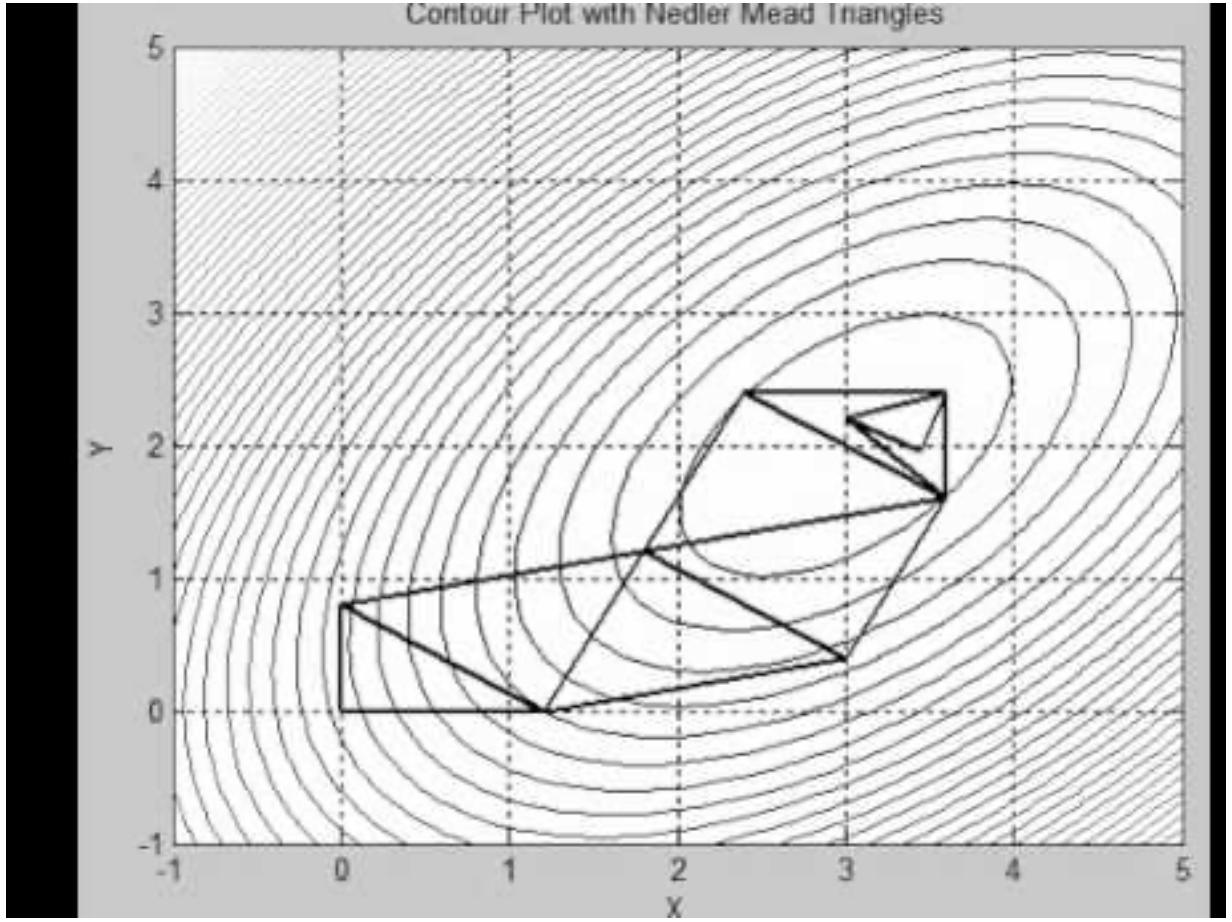
N-R or BHHH?

- N-R
- Advantages :
  - Converge quickly
  - The estimation is robust
- Disadvantages:
  - Computationally intensive as it requires to compute and invert the Hessian matrix at each step.
  - Information matrix might not be well-defined

- BHHH
- Advantages :
  - Only needs 1st derivatives;
  - The outer product is necessarily positive definite.
- Disadvantages:
  - The BHHH method is likely to have a slower convergence rate since the information matrix is only an approximation of the Hessian, more iterations may be needed.
  - This approximation can be quite bad, especially if $\theta_0$is very far from the optimal value θ.

Nelder-Mead (N-M)

- The Nelder-Mead method uses a geometrical shape(triangle) called a simplex to obtain optimal value .
- The simplex is like a “vehicle” of sorts to search the domain. This is why the technique is also called the Simplex search method.
- Three main steps:
  1. Ordering
  2. Centroid
  3. Transformation: reflect, expand contract, and shrink.

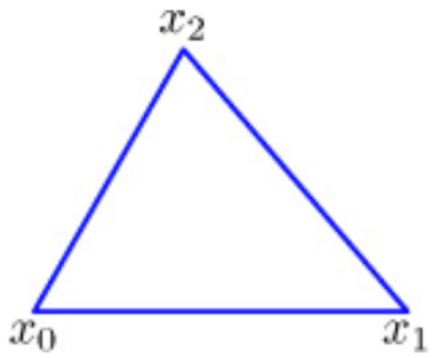


Nelder-Mead (N-M)

- **Ordering**: determine the indices h,s,l of the worst, second worst and the best vertex, respectively, in the current working simplex S

$$f_h = min_j f_j, \quad f_s = min_{j \neq h} f_j, \quad f_l = max_{j \neq h} f_j$$

- **Centroid**: Calculate the centroid c of the best side—this is the one opposite the worst vertex x_h, $c = \frac{1}{n} \sum_{j \neq h} x_j$

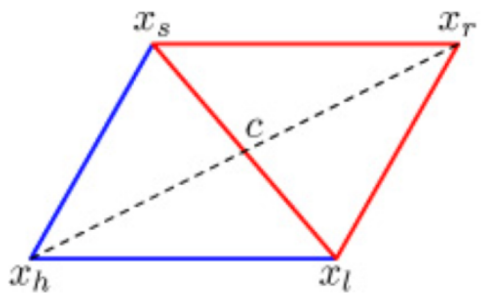


Nelder-Mead (N-M)

- **Transformation**: Compute the new working simplex from the current one. First, try to replace only the worst vertex x_h with a better point by using reflection, expansion or contraction with respect to the best side.Simplex transformations in the Nelder-Mead method are controlled by four parameters,which are $\alpha > 0$, $0 < \beta < 1$, $\gamma > 1$, $0 < \delta < 1$. For most of implementations, $\alpha = 1$, $\beta = 1/2$, $\gamma = 2$, $delta = 1/2$
  - Reflect:

    Compute the reflection point $x_r = c + \alpha(c - x_h)$ and $f_r = f(x_r)$.

    If $f_s \leq f_r$, accept x_r,terminate the iteration and move to the expansion interation.



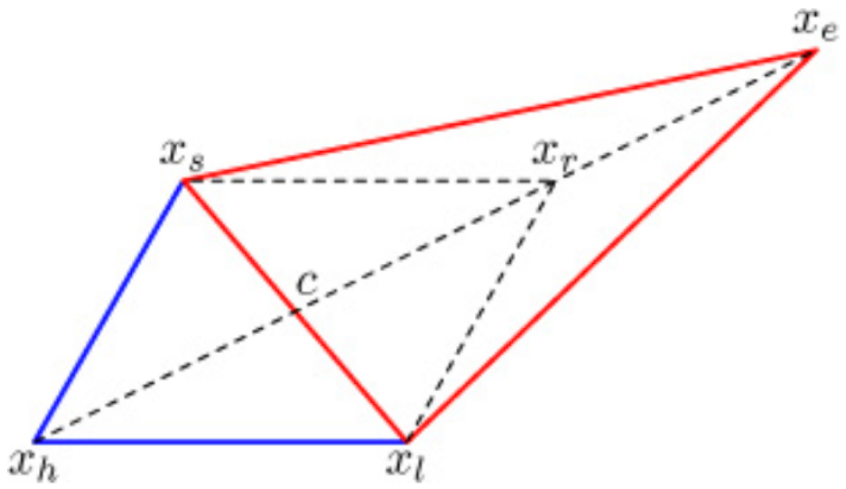
Nelder-Mead (N-M)

– Expand:

If $f_r > f_l$ compute the expansion point, $x_e = c + \gamma(x_r - c)$ and $f_e = f(x_e)$.

If $f_e > f_r$, accept x_e and terminate the iteration and move to contraction interation.

Otherwise $f_e \leq f_r$, accept x_r, terminate the iteration and move to contraction interation.



Nelder-Mead (N-M)

– Contraction: If $f_r \leq f_s$, compute the contraction point x_c by using the better of the two points x_h and x_r.

  * Outside:

    If $f_h \leq f_r < f_s$, compute $x_c = c + \beta(x_r - c)$ and $f_c = f(x_c)$.

    If $f_c \geq f_r$, accept x_c and terminate the iteration.

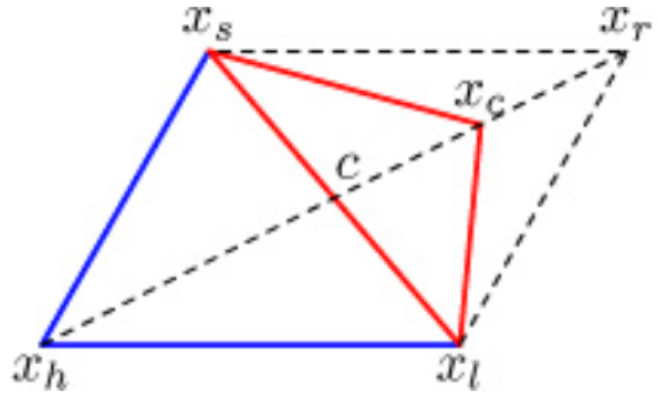
    Otherwise, perform a shrink transformation.

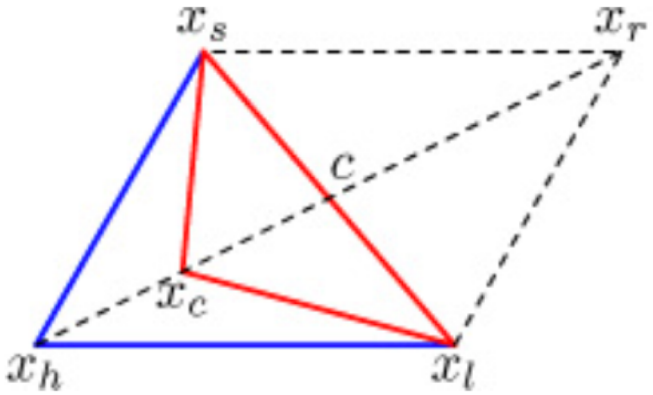
  * Inside:

    If $f_h \geq f_r$, compute $x_c = c + \beta(x_h - c)$ and $f_c = f(x_c)$.

    If $f_c > f_h$, accept x_c and terminate the iteration.

    Otherwise, perform a shrink transformation.



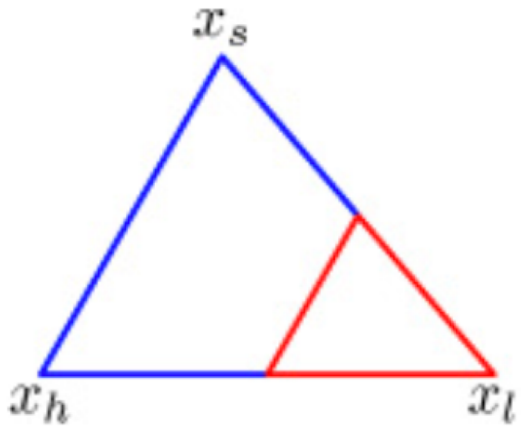


Nelder-Mead (N-M)

– Shrink:

Compute n new vertices $x_j = x_l + \delta(x_j - x_l)$ and $f_j = f(x_j)$, for j=0,. . . ,n, with $j \neq l$.

To be noticed, The shrink takes place when the contraction causes the reflected point to move away from the the peak instead of towards it. Further contractions are then useless. The action proposed contracts the simplex towards the highest point, and will eventually bring all points nearby the peak.



Application on normal distribution (N-M)

```r
mleNM <- maxLik(logLik = logLikFun, start = c(mu = 0,  sigma = 1), method = "NM")
summary(mleNM)
```

```
## --------------------------------------------
## Maximum Likelihood estimation
## Nelder-Mead maximization, 61 iterations
## Return code 0: successful convergence
## Log-Likelihood: -206.7598
## 2  free parameters
## Estimates:
##       Estimate Std. error t value  Pr(> t)
## mu      0.9825     0.1911    5.14 2.74e-07 ***
## sigma   1.9130     0.1352   14.15  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Nelder-Mead (N-M)

- Advantages:
  - 1. Gives significant improvements in the first few iterations and quickly produces quite satisfactory results. It is very important in applications where each function evaluation is very expensive or time-consuming.
  - 2. The method typically requires only one or two function evaluations per iteration.
- Disadvantage:
  - 1. The lack of convergence theory is often reflected in practice as a numerical breakdown of the algorithm, even for smooth and well-behaved functions.
  - 2. The method can take an enormous amount of iterations with negligible improvement in function value, despite being nowhere near to the optimum.

Summary and simulation results for N-R, BHHH, and N-R

Table 1: Estimations for 3 methods

	N-R	BHHH	N-M
Loglikeloood	-206.7597594	-206.7597594	-206.7597600
Mu	0.9823451	0.9823449	0.9825471
Sigma	1.9129739	1.9129799	1.9129516

Table 2: Covariance matrix for 3 methods

	mu	sigma
N-R		
mu	0.0365947	0.0000000
sigma	0.0000000	0.0182973
BHHH		
mu	0.0365957	-0.0001099
sigma	-0.0001099	0.0208923
N-M		
mu	0.0365362	-0.0000190
sigma	-0.0000190	0.0182871