1 Overview of BB

"BB" is a package intended for two purposes: (1) for solving a nonlinear system of equations, and (2) for finding a local optimum (can be minimum or maximum) of a scalar, objective function. An attractive feature of the package is that it has minimum memory requirements. Therefore, it is particularly well suited to solving high-dimensional problems with tens of thousands of parameters. However, *BB* can also be used to solve a single nonlinear equation or optimize a function with just one variable. The functions in this package are made available with:

```
> library("BB")
```

You can look at the basic information on the package, including all the available functions wtih

```
> help(package=BB)
```

The three basic functions are: *spg*, *dfsane*, and *sane*. You should *spg* for optimization, and either *dfsane* or *sane* for solving a nonlinear system of equations. We prefer *dfsane*, since it tends to perform slightly better than *sane*. There are also 3 higher level functions: *BBoptim*, *BBsolve*, and *multiStart*. *BBoptim* is a wrapper for *spg* in the sense that it calls *spg* repeatedly with different algorithmic options. It can be used when *spg* fails to find a local optimum, or it can be used in place of *spg*. Similarly, *BBsolve* is a wrapper for *dfsane* in the sense that it calls *dfsane* repeatedly with different algorithmic options. It can be used when *dfsane* (*sane*) fails to find a local optimum, or it can be used in place of *dfsane* (*sane*). The *multiStart* function can accept multiple starting values. It can be used for either solving a nonlinear system or for optimizing. It is useful for exploring sensitivity to starting values, and also for finding multiple solutions.

The package *setRNG* is not necessary, but if you want to exactly reproduce the examples in this guide then do this:

```
> require("setRNG")
> setRNG(list(kind="Wichmann-Hill", normal.kind="Box-Muller", seed=1236))
```

after which the example need to be run in the order here (or at least the parts that generate random numbers). For some examples the RNG is reset again so they can be reproduced more easily.

2 How to solve a nonlinear system of equations with BB?

The first two examples are from La Cruz and Raydan, Optim Methods and Software 2003, 18 (583-599).

```
> expo3 <- function(p) {
  #  From La Cruz and Raydan, Optim Methods and Software 2003, 18 (583-599)
  n <- length(p)
  f <- rep(NA, n)
  onm1 <- 1:(n-1)
  f[onm1] <- onm1/10 * (1 - p[onm1]^2 - exp(-p[onm1]^2))
  f[n] <- n/10 * (1 - exp(-p[n]^2))
  f
  }
> p0 <- runif(10)
> ans <- dfsane(par=p0, fn=expo3)

Iteration:  0  ||F(x0)||:  0.2024112
iteration:  10  ||F(xn)|| =   0.07536174
iteration:  20  ||F(xn)|| =   0.08777425
iteration:  30  ||F(xn)|| =   0.005029196
iteration:  40  ||F(xn)|| =   0.001517709
iteration:  50  ||F(xn)|| =   0.001769548
iteration:  60  ||F(xn)|| =   0.007896929
iteration:  70  ||F(xn)|| =   0.0001410588
iteration:  80  ||F(xn)|| =   2.002796e-06

> ans

$par
 [1]  3.819663e-02  3.031250e-02  2.647897e-02  2.404688e-02  2.233208e-02
 [6]  2.101498e-02  1.996221e-02  1.909301e-02  1.835779e-02 -7.493381e-06

$residual
[1] 6.645152e-08

$fn.reduction
[1] 0.6400804

$feval
[1] 96

$iter
[1] 85

$convergence
[1] 0

$message
[1] "Successful convergence"
```

Let us look at the output from *dfsane*. It is a list with 7 components. The most important components to focus on are the two named "*par*" and "*conver-*

gence". *ans$par* provides the solution from *dfsane*, but this is a root if and only if *ans$convergence* is equal to *0*, i.e. *ans$message* should say "Successful convergence". Otherwise, the algorithm has failed.

Now, we show an example demonstrating the ability of BB to solve a large system of equations, N = 10000.

```
> trigexp <- function(x) {
  n <- length(x)
  F <- rep(NA, n)
  F[1] <- 3*x[1]^2 + 2*x[2] - 5 + sin(x[1] - x[2]) * sin(x[1] + x[2])
  tn1 <- 2:(n-1)
  F[tn1] <- -x[tn1-1] * exp(x[tn1-1] - x[tn1]) + x[tn1] * ( 4 + 3*x[tn1]^2) +
          2 * x[tn1 + 1] + sin(x[tn1] - x[tn1 + 1]) * sin(x[tn1] + x[tn1 + 1]) - 8
  F[n] <- -x[n-1] * exp(x[n-1] - x[n]) + 4*x[n] - 3
  F
  }
> n <- 10000
> p0 <- runif(n)
> ans <- dfsane(par=p0, fn=trigexp, control=list(trace=FALSE))
> ans$message

[1] "Successful convergence"

> ans$resid

[1] 5.725351e-08
```

The next example is from Freudenstein and Roth function (Broyden, Mathematics of Computation 1965, p. 577-593).

```
> froth <- function(p){
  f <- rep(NA,length(p))
  f[1] <- -13 + p[1] + (p[2]*(5 - p[2]) - 2) * p[2]
  f[2] <- -29 + p[1] + (p[2]*(1 + p[2]) - 14) * p[2]
  f
  }
```

Now, we introduce the function *BBsolve*. For the first starting value, *dfsane* used in the default manner does not find the zero, but *BBsolve*, which tries multiple control parameter settings, is able to successfully find the zero.

```
> p0 <- c(3,2)
> dfsane(par=p0, fn=froth, control=list(trace=FALSE))

$par
[1] -9.822061 -1.875381

$residual
```

```
[1] 11.63811

$fn.reduction
[1] 25.58882

$feval
[1] 137

$iter
[1] 114

$convergence
[1] 5

$message
[1] "Lack of improvement in objective function"
```

```
> BBsolve(par=p0, fn=froth)

  Successful convergence.
$par
[1] 5 4

$residual
[1] 3.659749e-10

$fn.reduction
[1] 0.001827326

$feval
[1] 100

$iter
[1] 10

$convergence
[1] 0

$message
[1] "Successful convergence"

$cpar
method      M     NM
     2     50      1
```

Note that the functions *dfsane*, *sane*, and *spg* produce a warning message if convergence fails. These warnings have been suppressed in this vignette.

For the next starting value, *BBsolve* finds the zero of the system, but *dfsane* (with defaults) fails.

```
> p0 <- c(1,1)
> BBsolve(par=p0, fn=froth)

  Successful convergence.
$par
[1] 5 4

$residual
[1] 9.579439e-08

$fn.reduction
[1] 6.998875

$feval
[1] 1165

$iter
[1] 247

$convergence
[1] 0

$message
[1] "Successful convergence"

$cpar
method      M     NM
     1     50      1

> dfsane(par=p0, fn=froth, control=list(trace=FALSE))

$par
[1] -9.674222 -1.984882

$residual
[1] 12.15994

$fn.reduction
[1] 24.03431

$feval
[1] 138

$iter
```

```
[1] 109

$convergence
[1] 5

$message
[1] "Lack of improvement in objective function"
```

Try random starting values. Run the following set of code many times. This shows that *BBsolve* is quite robust in finding the zero, whereas *dfsane* (with defaults) is sensitive to starting values. Admittedly, these are poor starting values, but still it would be nice to have a strategy that has a high likelihood of finding a zero of the nonlinear system.

```
> # two values generated independently from a poisson distribution with mean = 10
> p0 <- rpois(2,10)
> BBsolve(par=p0, fn=froth)

  Successful convergence.
$par
[1] 5 4

$residual
[1] 7.330654e-08

$fn.reduction
[1] 0.07273382

$feval
[1] 91

$iter
[1] 41

$convergence
[1] 0

$message
[1] "Successful convergence"

$cpar
method      M     NM
     2     50      1

> dfsane(par=p0, fn=froth, control=list(trace=FALSE))

$par
[1] 5 4
```

```
$residual
[1] 5.472171e-08

$fn.reduction
[1] 490.618

$feval
[1] 32

$iter
[1] 31

$convergence
[1] 0

$message
[1] "Successful convergence"
```

2.1 Finding multiple roots of a nonlinear system of equations

Now, we introduce the function *multiStart*. This accepts a matrix of starting values, where each row is a single starting value. *multiStart* calls *BBsolve* for each starting value. Here is a system of 3 non-linear equations, where each equation is a high-degree polynomial. This system has 12 real-valued roots and 126 complex-valued roots. Here we will demonstrate how to identify all the 12 real roots using *multiStart*. Note that we specify the '*action*' argument in the following call to *multiStart* only to highlight that *multiStart* can be used for both solving a system of equations and for optimization. The default is '*action = "solve"*', so it is really not needed in this call.

```
> # Example
> # A high-degree polynomial system (R.B. Kearfott, ACM 1987)
> # There are 12 real roots (and 126 complex roots to this system!)
> #
> hdp <- function(x) {
    f <- rep(NA, length(x))
    f[1] <- 5 * x[1]^9 - 6 * x[1]^5 * x[2]^2 + x[1] * x[2]^4 + 2 * x[1] * x[3]
    f[2] <- -2 * x[1]^6 * x[2] + 2 * x[1]^2 * x[2]^3 + 2 * x[2] * x[3]
    f[3] <- x[1]^2 + x[2]^2 - 0.265625
    f
    }
```

We generate 100 randomly generated starting values, each a vector of length equal to 3. (Setting the seed is only necessary to reproduce the result shown here.)

```
> setRNG(list(kind="Wichmann-Hill", normal.kind="Box-Muller", seed=123))
> p0 <- matrix(runif(300), 100, 3)  # 100 starting values, each of length 3
> ans <- multiStart(par=p0, fn=hdp, action="solve")
> sum(ans$conv)  # number of successful runs = 99
> pmat <- ans$par[ans$conv, ] # selecting only converged solutions
```

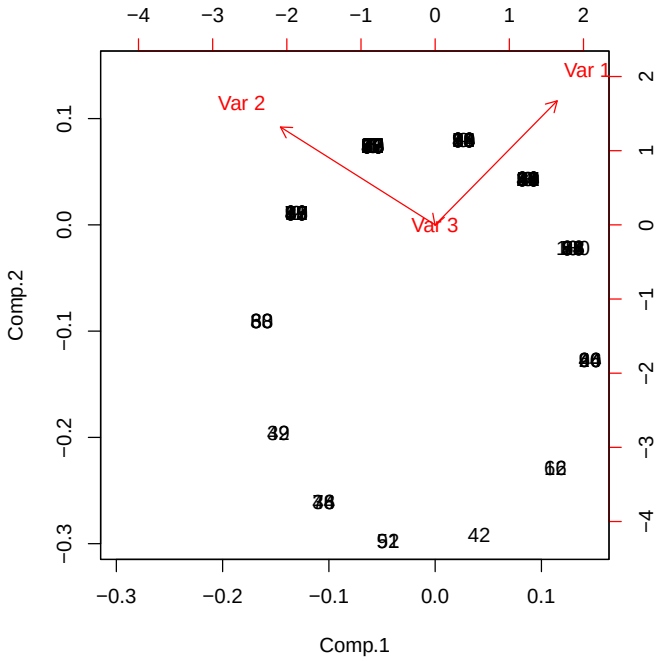
Now, we display the unique real solutions.

```
> ans <- round(pmat, 4)
> ans[!duplicated(ans), ]
```

```
         [,1]    [,2]    [,3]
 [1,]  0.2799  0.4328 -0.0142
 [2,]  0.0000  0.5154  0.0000
 [3,]  0.5154  0.0000 -0.0124
 [4,]  0.4670 -0.2181  0.0000
 [5,]  0.4670  0.2181  0.0000
 [6,]  0.0000 -0.5154  0.0000
 [7,]  0.2799 -0.4328 -0.0142
 [8,] -0.4670  0.2181  0.0000
 [9,] -0.2799  0.4328 -0.0142
[10,] -0.5154  0.0000 -0.0124
[11,] -0.2799 -0.4328 -0.0142
[12,] -0.4670 -0.2181  0.0000
```

We can also visualize these 12 solutions beautifully using a 'biplot' based on the first 2 principal components of the converged parameter matrix.

```
> pc <- princomp(pmat)
> biplot(pc)  # you can see all 12 solutions beautifully like on a clock!
```



2.2 Power polynomial method: Fleishman system of equations

Fleishman (Psychometrika 1978, p.521-532) developed an approach for simulating random numbers from non-normal distrinbutions, with specified values of skewness and kurtosis. This approach involves the solution of a system of polynomial equations. This system is also discussed in the paper by Demirtas and Hedeker (Communications in Statistics 2008, p. 1682-1695; Equations on p. 1684) and is given as follows:

```
> fleishman <- function(x, r1, r2) {
  b <- x[1]
  c <- x[2]
  d <- x[3]
  f <- rep(NA, 3)
  f[1] <- b^2 + 6 * b * d + 2 * c^2 + 15 * d^2 - 1
  f[2] <- 2*c * (b^2 + 24*b*d + 105*d^2 + 2) - r1
  f[3] <- b*d + c^2 * (1 + b^2 + 28 * b * d) + d^2 * (12 + 48 * b* d +
                141 * c^2 + 225 * d^2) - r2/24
  f
  }
```

We only use 3 equations, since 1st equation is trivially solved by a = -c.

Here we describe an experiment based on Fleishman (Psychometrika 1978, p.521-532), and is reproduced as follows. We randomly picked 10 scenarios (more or less randomly) from Table 1 of Fleishman (1978):

```
> rmat <- matrix(NA, 10, 2)
> rmat[1,] <- c(1.75, 3.75)
> rmat[2,] <- c(1.25, 2.00)
> rmat[3,] <- c(1.00, 1.75)
> rmat[4,] <- c(1.00, 0.50)
> rmat[5,] <- c(0.75, 0.25)
> rmat[6,] <- c(0.50, 3.00)
> rmat[7,] <- c(0.50, -0.50)
> rmat[8,] <- c(0.25, -1.00)
> rmat[9,] <- c(0.0, -0.75)
> rmat[10,] <- c(-0.25, 3.75)
```

We solve the system of equations for the above 10 specifications of skewness and kurtosis 3 times, each time with a different random starting seed.

```
> # 1
> setRNG(list(kind="Mersenne-Twister", normal.kind="Inversion", seed=13579))
> ans1 <- matrix(NA, nrow(rmat), 3)
> for (i in 1:nrow(rmat)) {
    x0 <- rnorm(3)  # random starting value
    temp <- BBsolve(par=x0, fn=fleishman, r1=rmat[i,1], r2=rmat[i,2])
    if (temp$conv == 0) ans1[i, ] <- temp$par
    }

  Successful convergence.
  Successful convergence.
  Successful convergence.
  Successful convergence.
  Successful convergence.
  Successful convergence.
  Successful convergence.
  Successful convergence.
  Successful convergence.
  Successful convergence.

> ans1 <- cbind(rmat, ans1)
> colnames(ans1) <- c("skew", "kurtosis", "B", "C", "D")
> ans1

       skew kurtosis          B             C            D
 [1,]  1.75     3.75 -0.9296606  3.994967e-01  0.036466986
 [2,]  1.25     2.00 -0.9664061  2.230888e-01  0.005862543
```

```
 [3,]  1.00     1.75  0.9274664  1.543072e-01  0.015885481
 [4,]  1.00     0.50  1.1146549  2.585245e-01 -0.066013188
 [5,]  0.75     0.25 -1.2977959  2.727191e-01  0.150766137
 [6,]  0.50     3.00 -0.7933810  5.859729e-02 -0.063637596
 [7,]  0.50    -0.50 -1.3482151  1.886967e-01  0.153679396
 [8,]  0.25    -1.00 -1.3628960  9.474017e-02  0.146337538
 [9,]  0.00    -0.75  1.1336220 -6.936031e-13 -0.046731705
[10,] -0.25     3.75  1.5483100 -6.610187e-02 -0.263217996
```

```
> # 2
> setRNG(list(kind="Mersenne-Twister", normal.kind="Inversion", seed=91357))
> ans2 <- matrix(NA, nrow(rmat), 3)
> for (i in 1:nrow(rmat)) {
    x0 <- rnorm(3)  # random starting value
    temp <- BBsolve(par=x0, fn=fleishman, r1=rmat[i,1], r2=rmat[i,2])
    if (temp$conv == 0) ans2[i, ] <- temp$par
    }

  Successful convergence.
  Successful convergence.
  Successful convergence.
  Successful convergence.
  Successful convergence.
  Successful convergence.
  Successful convergence.
  Successful convergence.
  Successful convergence.
  Successful convergence.
```

```
> ans2 <- cbind(rmat, ans2)
> colnames(ans2) <- c("skew", "kurtosis", "B", "C", "D")
> ans2

       skew kurtosis          B             C           D
 [1,]  1.75     3.75 -0.9296606  3.994967e-01  0.03646699
 [2,]  1.25     2.00  0.9664061  2.230888e-01 -0.00586255
 [3,]  1.00     1.75 -0.9274663  1.543073e-01 -0.01588548
 [4,]  1.00     0.50 -1.1146552  2.585249e-01  0.06601337
 [5,]  0.75     0.25 -1.2977961  2.727192e-01  0.15076629
 [6,]  0.50     3.00  0.7933810  5.859729e-02  0.06363759
 [7,]  0.50    -0.50 -1.3482151  1.886967e-01  0.15367938
 [8,]  0.25    -1.00  1.3628963  9.474021e-02 -0.14633771
 [9,]  0.00    -0.75  1.1336221 -2.520587e-13 -0.04673174
[10,] -0.25     3.75  0.7503153 -2.734120e-02  0.07699283
```

```
> # 3
> setRNG(list(kind="Mersenne-Twister", normal.kind="Inversion", seed=79135))
```

```
> ans3 <- matrix(NA, nrow(rmat), 3)
> for (i in 1:nrow(rmat)) {
    x0 <- rnorm(3)  # random starting value
    temp <- BBsolve(par=x0, fn=fleishman, r1=rmat[i,1], r2=rmat[i,2])
    if (temp$conv == 0) ans3[i, ] <- temp$par
    }

  Successful convergence.
  Successful convergence.
  Successful convergence.
  Successful convergence.
  Successful convergence.
  Successful convergence.
  Successful convergence.
  Successful convergence.
  Successful convergence.
  Successful convergence.

> ans3 <- cbind(rmat, ans3)
> colnames(ans3) <- c("skew", "kurtosis", "B", "C", "D")
> ans3

       skew kurtosis          B             C           D
 [1,]  1.75     3.75  0.9207619  4.868014e-01 -0.07251973
 [2,]  1.25     2.00  1.1312711  4.094915e-01 -0.12535043
 [3,]  1.00     1.75  0.9274666  1.543073e-01  0.01588543
 [4,]  1.00     0.50 -1.1146554  2.585250e-01  0.06601345
 [5,]  0.75     0.25  1.0591737  1.506888e-01 -0.02819626
 [6,]  0.50     3.00 -0.7933811  5.859729e-02 -0.06363759
 [7,]  0.50    -0.50  1.1478497  1.201563e-01 -0.05750376
 [8,]  0.25    -1.00 -1.3628959  9.474013e-02  0.14633745
 [9,]  0.00    -0.75 -1.1336219 -1.156859e-16  0.04673169
[10,] -0.25     3.75  1.5483099 -6.610187e-02 -0.26321798

>
```

This usually finds an accurate root of the Fleishman system successfully in all 50 cases (but may occassionaly fail with different seeds).

An interesting aspect of this exercise is the existence of multiple roots to the Fleishman system. There are 4 valid roots for any "feasible" combination of skewness and kurtosis. These 4 roots can be denoted as: $(b_1, c_1, -d_1)$, $(-b_1, c_1, d_1)$, $(b_2, c_2, -d_2)$, $(-b_2, c_2, d_2)$, where $b_1, c_1, d_1, b_2, c_2, d_2$ are all positive (except for the coefficient c which is zero when skewness is zero). Fleishman only reports the first root, whereas we can locate the other roots using BBsolve.

The experiments demonstrate quite convincingly that the wrapper function *BBsolve* can successfully solve the system of equations associated with the power polynomial method of Fleishman.

3 How to optimize a nonlinear objective function with BB?

The basic function for optimization is *spg*. It can solve smooth, nonlinear optimization problems with box-constraints, and also other types of constraints using projection. We would like to direct the user to the help page for many examples of how to use *spg*. Here we discuss an example involving estimation of parameters maximizing a log-likelihood function for a binary Poisson mixture distribution.

```
> poissmix.loglik <- function(p,y) {
  # Log-likelihood for a binary Poisson mixture distribution
  i <- 0:(length(y)-1)
  loglik <- y * log(p[1] * exp(-p[2]) * p[2]^i / exp(lgamma(i+1)) +
          (1 - p[1]) * exp(-p[3]) * p[3]^i / exp(lgamma(i+1)))
  return (sum(loglik) )
  }
> # Data from Hasselblad (JASA 1969)
> poissmix.dat <- data.frame(death=0:9,
            freq=c(162,267,271,185,111,61,27,8,3,1))
```

There are 3 model parameters, which have restricted domains. So, we define these constraints as follows:

```
> lo <- c(0,0,0)  # lower limits for parameters
> hi <- c(1, Inf, Inf) # upper limits for parameters
```

Now, we maximize the log-likelihood function using both *spg* and *BBoptim*, with a randomly generated starting value for the 3 parameters:

```
> p0 <- runif(3,c(0.2,1,1),c(0.8,5,8))  # a randomly generated vector of length 3
> y <- c(162,267,271,185,111,61,27,8,3,1)
> ans1 <- spg(par=p0, fn=poissmix.loglik, y=y,
            lower=lo, upper=hi, control=list(maximize=TRUE, trace=FALSE))
> ans1

$par
[1] 0.640094 2.663430 1.256131

$value
[1] -1989.946

$gradient
[1] 0.0001523404

$fn.reduction
[1] -209.0405
```

```
$iter
[1] 42

$feval
[1] 44

$convergence
[1] 0

$message
[1] "Successful convergence"
```

```
> ans2 <- BBoptim(par=p0, fn=poissmix.loglik, y=y,
           lower=lo, upper=hi, control=list(maximize=TRUE))

iter:  0  f-value:  -2198.986  pgrad:  360.6254
iter:  10  f-value:  -1991.173  pgrad:  3.212342
iter:  20  f-value:  -1990.47  pgrad:  1.571746
iter:  30  f-value:  -1990.053  pgrad:  0.6429582
iter:  40  f-value:  -1989.946  pgrad:  0.3574752
iter:  50  f-value:  -1989.946  pgrad:  0.01283524
iter:  60  f-value:  -1989.946  pgrad:  0.0009822543
  Successful convergence.
```

```
> ans2

$par
[1] 0.6401248 2.6633909 1.2560779

$value
[1] -1989.946

$gradient
[1] 0.0001591616

$fn.reduction
[1] -209.0405

$iter
[1] 65

$feval
[1] 169

$convergence
[1] 0
```

```
$message
[1] "Successful convergence"

$cpar
method      M
     2     50
```

Note that we had to specify the '*maximize*' option inside the *control* list to let the algorithm know that we are maximizing the objective function, since the default is to minimize the objective function. Also note how we pass the data vector '*y*' to the log-likelihood function, *possmix.loglik*.

Now, we illustrate how to compute the Hessian of the log-likelihood at the MLE, and then how to use the Hessian to compute the standard errors for the parameters. To compute the Hessian we require the package "*numDeriv*."

```
> require(numDeriv)
> hess <- hessian(x=ans2$par, func=poissmix.loglik, y=y)
> # Note that we have to supplied data vector `y'
> hess

          [,1]       [,2]       [,3]
[1,] -907.1186 -341.25895 -270.22619
[2,] -341.2590 -192.78641  -61.68141
[3,] -270.2262  -61.68141 -113.47653

> se <- sqrt(diag(solve(-hess)))
> se

[1] 0.1946797 0.2504706 0.3500305
```

Now, we explore the use of multiple starting values to see if we can identify multiple local maxima. We have to make sure that we specify '*action = "optimize"*', because the default option in *multiStart* is *"solve"*.

```
> # 3 randomly generated starting values
> p0 <- matrix(runif(30, c(0.2,1,1), c(0.8,8,8)), 10, 3, byrow=TRUE)
> ans <- multiStart(par=p0, fn=poissmix.loglik, action="optimize",
        y=y, lower=lo, upper=hi, control=list(maximize=TRUE))

Parameter set :  1 ...
iter:  0  f-value:  -2629.616  pgrad:  5.149479
iter:  10  f-value:  -2001.398  pgrad:  0.01419494
  Successful convergence.
Parameter set :  2 ...
iter:  0  f-value:  -2046.752  pgrad:  172.2726
iter:  10  f-value:  -1990.065  pgrad:  0.8918596
```

```
iter:  20  f-value:  -1990.031  pgrad:  4.181468
iter:  30  f-value:  -1990.291  pgrad:  8.707709
  Successful convergence.
Parameter set :  3 ...
iter:  0  f-value:  -2722.155  pgrad:  7.534183
iter:  10  f-value:  -1991.544  pgrad:  3.31378
iter:  20  f-value:  -1990.761  pgrad:  8.096627
iter:  30  f-value:  -1989.949  pgrad:  0.5093102
iter:  40  f-value:  -1989.946  pgrad:  0.0177306
  Successful convergence.
Parameter set :  4 ...
iter:  0  f-value:  -3692.509  pgrad:  6.213669
iter:  10  f-value:  -1990.145  pgrad:  2.718772
iter:  20  f-value:  -1990.188  pgrad:  7.280146
iter:  30  f-value:  -1989.946  pgrad:  0.0004024514
  Successful convergence.
Parameter set :  5 ...
iter:  0  f-value:  -2996.35  pgrad:  7.452469
iter:  10  f-value:  -1997.898  pgrad:  2.430247
iter:  20  f-value:  -1989.959  pgrad:  0.3875152
iter:  30  f-value:  -1989.949  pgrad:  0.5795755
iter:  40  f-value:  -1989.946  pgrad:  0.01441776
  Successful convergence.
Parameter set :  6 ...
iter:  0  f-value:  -4492.74  pgrad:  6.965384
iter:  10  f-value:  -2001.472  pgrad:  8.750483
  Successful convergence.
Parameter set :  7 ...
iter:  0  f-value:  -3357.482  pgrad:  6.954945
iter:  10  f-value:  -1991.658  pgrad:  2.799363
iter:  20  f-value:  -1989.997  pgrad:  0.6908181
iter:  30  f-value:  -1989.959  pgrad:  1.203134
iter:  40  f-value:  -1989.946  pgrad:  0.001996341
iter:  50  f-value:  -1989.946  pgrad:  0.001468834
  Successful convergence.
Parameter set :  8 ...
iter:  0  f-value:  -3172.301  pgrad:  5.470799
iter:  10  f-value:  -2007.457  pgrad:  2.315072
  Successful convergence.
Parameter set :  9 ...
iter:  0  f-value:  -4019.753  pgrad:  6.606661
iter:  10  f-value:  -1993.303  pgrad:  32.41122
iter:  20  f-value:  -1990.292  pgrad:  2.832038
iter:  30  f-value:  -1989.956  pgrad:  1.914161
iter:  40  f-value:  -1989.946  pgrad:  0.02872412
  Successful convergence.
```

```
Parameter set :  10 ...
iter:  0  f-value:  -2045.64  pgrad:  3.808228
iter:  10  f-value:  -1991.291  pgrad:  2.49011
iter:  20  f-value:  -1990.413  pgrad:  1.413719
iter:  30  f-value:  -1989.946  pgrad:  0.03627974
iter:  40  f-value:  -1989.946  pgrad:  0.01119133
iter:  50  f-value:  -1989.946  pgrad:  0.0002614797
  Successful convergence.

> # selecting only converged solutions
> pmat <-  round(cbind(ans$fvalue[ans$conv], ans$par[ans$conv, ]), 4)
> dimnames(pmat) <- list(NULL, c("fvalue","parameter 1","parameter 2","parameter 3"))
> pmat[!duplicated(pmat), ]

        fvalue parameter 1 parameter 2 parameter 3
[1,] -1996.689      0.3095      2.6448      1.9311
[2,] -1989.946      0.3599      1.2561      2.6634
[3,] -1989.946      0.6401      2.6634      1.2561
[4,] -1995.572      0.4053      2.6018      1.8435
[5,] -1997.205      0.7042      1.9525      2.6274

>
```

Here *multiStart* is able to identifies many solutions. Two of these, the 2nd and 3rd rows, appear to be global maxima with different parameter values. Actually, there is only one global maximum. It is due to the 'label switching' problem that we see 2 solutions. The multiStart algorithm also identifies three local maxima with inferior values.