# 1 Functions for Barzilai-Borwein Optimization

The functions in this package are made available with

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

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

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

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

```
> expo1 <- function(x) {
      n <- length(x)
      f <- rep(NA, n)
      f[1] <- exp(x[1] - 1) - 1
      f[2:n] <- (2:n) * (exp(x[2:n] - 1) - x[2:n])
      f
  }
> p0 <- runif(100)
> sane(par = p0, fn = expo1)
> dfsane(par = p0, fn = expo1)
```

```
> expo3 <- function(p) {
      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
  }
> n <- 100
> p0 <- (1:n)/(4 * n^2)
> sane(par = p0, fn = expo3)
> dfsane(par = p0, fn = expo3)
```

This 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
  }
```

Here p0 gives the zero of the system

```
> p0 <- c(3, 2)
> sane(par = p0, fn = froth)
> dfsane(par = p0, fn = froth)
```

Here p0 gives the local minimum that is not the zero of the system.

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

Trying random starts

```
> p0 <- rpois(2, 10)
> sane(par = p0, fn = froth)
> dfsane(par = p0, fn = froth)
```

This example is from

```
> trig <- function(x) {
      n <- length(x)
      suma <- sum(cos(x))
      o2n <- 1:n
      F <- 2 * (n + o2n * (1 - cos(x)) - sin(x) - sum(cos(x))) *
          (2 * sin(x) - cos(x))
      F
  }
> p0 <- runif(100)
> sane(par = p0, fn = trig)
> dfsane(par = p0, fn = trig)
```

This example is from

```
> 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
  }
> sane(par = p0, fn = trigexp)
> dfsane(par = p0, fn = trigexp)
```

This example is from

```
> chen <- function(x) {
      v <- log(x) + exp(x)
      f <- (v - sqrt(v^2 + 0.005))/2
      f
  }
> sane(par = p0, fn = chen)
> dfsane(par = p0, fn = chen)
> valley <- function(x) {
      c1 <- 1.00334448160535
      c2 <- -0.000334448160535117
      n <- length(x)
      f <- rep(NA, n)
      j <- 3 * (1:(n/3))
      jm2 <- j - 2
      jm1 <- j - 1
      f[jm2] <- (c2 * x[jm2]^3 + c1 * x[jm2]) * exp(-(x[jm2]^2)/100) -
          1
      f[jm1] <- 10 * (sin(x[jm2]) - x[jm1])
      f[j] <- 10 * (cos(x[jm2]) - x[j])
      f
  }
```

Here the number of variables must be a multiple of 3

```
> p102 <- runif(102)
> sane(par = p102, fn = valley)
> dfsane(par = p102, fn = valley)
> broydt <- function(x) {
      n <- length(x)
      f <- rep(NA, n)
      f[1] <- ((3 - 0.5 * x[1]) * x[1]) - 2 * x[2] + 1
      tnm1 <- 2:(n - 1)
      f[tnm1] <- ((3 - 0.5 * x[tnm1]) * x[tnm1]) - x[tnm1 - 1] -
          2 * x[tnm1 + 1] + 1
      f[n] <- ((3 - 0.5 * x[n]) * x[n]) - x[n - 1] + 1
      f
  }
> sane(par = p0, fn = broydt)
> dfsane(par = p0, fn = broydt)
```