lbfgsb3c: Using the 2011 version of L-BFGSB

John C Nash Telfer School of Management, University of Ottawa, nashjc@uottawa.ca

July 10, 2019

Abstract

In 2011 the authors of the L-BFGSB program published a correction and update to their 1995 code. The latter is the basis of the L-BFGS-B method of the `optim()` function in base-R. The package `lbfgsb3` wrapped the updated code using a `.Fortran` call after removing a very large number of Fortran output statements. Matthew Fidler used this Fortran code and an `Rcpp` interface to produce package `lbfgsb3c` where the function `lbfgsb3c()` returns an object similar to that of base-R `optim()` and that of `optimx::optimr()`. Subsequently, in a fine example of the collaborations that have made **R** so useful, we have merged the functionality of package `lbfgsb3` into `lbfgsb3c`, as explained in this vignette. Note that this document is intended primarily to document our efforts to check the differences in variants of the code rather than be expository.

Provenance of the R optim::L-BFGS-B and related solvers

The base-R code lbfgsb.c (at writing in R-3.5.2/src/appl/) is commented:

```
/* l-bfgs-b.f -- translated by f2c (version 19991025).

  From ?optim:
  The code for method ‘"L-BFGS-B"’ is based on Fortran code by Zhu,
  Byrd, Lu-Chen and Nocedal obtained from Netlib (file 'opt/lbfgs_bcm.shar')

  The Fortran files contained no copyright information.

  Byrd, R. H., Lu, P., Nocedal, J. and Zhu, C.  (1995) A limited
  memory algorithm for bound constrained optimization.
  \emph{SIAM J. Scientific Computing}, \bold{16}, 1190--1208.
*/
```

The paper R. H. Byrd, Lu, Nocedal, and Zhu (1995a) builds on Lu et al. (1994). There have been a number of other workers who have followed-up on this work, but **R** code and packages seem to have largely stayed with codes derived from these original papers. Though the date of the paper is 1995, the ideas it embodies were around for a decade and a half at least, in particular in Nocedal80 and LiuN89. The definitive Fortran code was published as Zhu et al. (1997). This is available as `toms/778.zip` on www.netlib.org. A side-by-side comparison of the main subroutines in the two downloads from Netlib unfortunately shows a lot of differences. I have not tried to determine if these affect performance or are simply cosmetic.

More seriously perhaps, there were some deficiencies in the code(s), and in 2011 Nocedal's team published a Fortran code with some corrections (Morales and Nocedal (2011)). Since the **R** code predates this, I prepared package `lbfgsb3` (Nash et al. (2015)) to wrap the Fortran code. However, I did not discover any test cases where the `optim::L-BFGS-B` and `lbfgsb3` were different, though I confess to only running some limited tests. There are, in fact, more in this vignette.

In 2016, I was at a Fields Institute optimization conference in Toronto for the 70th birthday of Andy Conn. By sheer serendipity, Nocedal did not attend the conference, but sat down next to me at the conference dinner. When I asked him about the key changes, he said that the most important one was to fix the computation of the machine precision, which was not always correct in the 1995 code. Since **R** gets this number as `.Machine$double.eps`, the offending code is irrelevant.

Within Morales and Nocedal (2011), there is also reported an improvement in the subspace minimization that is applied in cases of bounds constraints. Since few of the tests I have applied impose such constraints, it is reasonable that I will not have observed performance differences between the base-R `optim` code and my `lbfsgb3` package. More appropriate tests are welcome, and on my agenda.

Besides the ACM TOMS code, there are two related codes from the Northwestern team on NETLIB: http://netlib.org/opt/lbfgs_um.shar is for unconstrained minimization, while http://netlib.org/opt/lbfgs_bcm.shar handles bounds constrained problems. To these are attached references Liu and Nocedal (1989) and R. H. Byrd, Lu, Nocedal, and Zhu (1995b) respectively, most likely reflecting the effort required to implement the constraints.

The unconstrained code has been converted to **C** under the leadership of Naoaki Okazaki (see http://www.chokkan.org/software/liblbfgs/, or the fork at https://github.com/MIRTK/LBFGS). This has been wrapped for **R** as Coppola, Stewart, and Okazaki (2014) as the `lbfgs` package. This can be called from `optimx::optimr()`.

Using Rcpp (see Eddelbuettel and François (2011)) and the Fortran code in package `lbfgs3`, Matthew Fidler developed package `lbfgsb3c` (Fidler et al. (2018)). As this provides a more standard call and return than `lbfgsb3` Fidler and I are unified the two packages and released them both under the same name `lbfgsb3c`.

Functions in package `lbfgsb3c`

There is really only one optimizer function in the package, but it may be called by four (4) names:

- `lbfgsb3c()` uses Rcpp (Eddelbuettel (2013), Eddelbuettel and François (2011), Eddelbuettel and Balamuta (2017)) to streamline the call to the underlying Fortran. This is the base function used.
- `lbfgsb3x()` is an alias of `lbfgsb3c()`. We were using this name for a while, and have kept the alias to avoid having to edit test scripts.
- `lbfgsb3`, which imitates a `.Fortran` call of the compiled 2011 Fortran code. The object returned by this routine is NOT equivalent to the object returned by base-R `optim()` or by `optimx::optimr()`. Instead, it includes a structure `info` which contains the detailed diagnostic information of the Fortran code. For most users, this is not of interest, and I only recommend use of this function for those needing to examine how the optimization has been carried out.
- `lbfgsb3f()` is an alias of `lbfsgb3()`.

We recommend using the `lbfsgb3c()` call for most uses.

Comparison with optim::L-BFGS-B

The new Fortran package claims better performance on bounds-constrained problems. Below we present two fairly simple tests, which unfortunately do not show any advantage. We welcome examples showing differences, either better or not. Note that we use the call that returns expanded information, but we do not interpret that here. See the documentation in the source Fortran for an explanation of the data returned in object `info`.

```
# ref BT.RES in Nash and Walker-Smith (1987)
library(lbfgsb3c)
require(optimx)
```

```
## Loading required package: optimx
```

```
sessionInfo()
```

```
## R version 3.6.1 (2019-07-05)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Linux Mint 19.1
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/openblas/libblas.so.3
## LAPACK: /usr/lib/x86_64-linux-gnu/libopenblasp-r0.2.20.so
##
## locale:
##  [1] LC_CTYPE=en_CA.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_CA.UTF-8        LC_COLLATE=en_CA.UTF-8
##  [5] LC_MONETARY=en_CA.UTF-8    LC_MESSAGES=en_CA.UTF-8
##  [7] LC_PAPER=en_CA.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_CA.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] optimx_2019-6.14   lbfgsb3c_2019-7.11
##
## loaded via a namespace (and not attached):
##  [1] compiler_3.6.1      magrittr_1.5        tools_3.6.1
##  [4] htmltools_0.3.6     yaml_2.2.0          Rcpp_1.0.1
##  [7] stringi_1.4.3       rmarkdown_1.13      knitr_1.23
## [10] stringr_1.4.0       xfun_0.8            digest_0.6.20
## [13] numDeriv_2016.8-1.1 evaluate_0.14
```

```r
bt.f<-function(x){
 sum(x*x)
}

bt.g<-function(x){
  gg<-2.0*x
}

bt.badsetup<-function(n){
   x<-rep(0,n)
   lo<-rep(0,n)
   up<-lo # to get arrays set
   bmsk<-rep(1,n)
   bmsk[(trunc(n/2)+1)]<-0
   for (i in 1:n) {
      x[i]<-2.2*i-n
      lo[i]<-1.0*(i-1)*(n-1)/n
      up[i]<-1.0*i*(n+1)/n
   }
   result<-list(x=x, lower=lo, upper=up, bdmsk=bmsk)
}

bt.setup0<-function(n){
   x<-rep(0,n)
   lo<-rep(0,n)
```

```
   up<-lo # to get arrays set
   bmsk<-rep(1,n)
   bmsk[(trunc(n/2)+1)]<-0
   for (i in 1:n) {
      lo[i]<-1.0*(i-1)*(n-1)/n
      up[i]<-1.0*i*(n+1)/n
   }
   x<-0.5*(lo+up)
   result<-list(x=x, lower=lo, upper=up, bdmsk=bmsk)
}
meths <- c("L-BFGS-B", "lbfgsb3c", "lbfgsb3", "Rvmmin", "Rcgmin", "Rtnmin")
nn <- 4
goody <- bt.setup0(nn)
lo <- goody$lower
up <- goody$upper
x0 <- goody$x
goody
```

```
## $x
## [1] 0.625 1.625 2.625 3.625
##
## $lower
## [1] 0.00 0.75 1.50 2.25
##
## $upper
## [1] 1.25 2.50 3.75 5.00
##
## $bdmsk
## [1] 1 1 0 1
```

```
## optim()
## Put trace=3 to get full details
solgood <- opm(x0, bt.f, bt.g, lower=lo, upper=up, method=meths, control=list(trace=0))
```

```
## trace= 0
```

```
print(summary(solgood, order=value))
```

```
##          p1   p2  p3   p4 value fevals gevals convergence  kkt1 kkt2
## L-BFGS-B  0 0.75 1.5 2.25 7.875      2      2           0 FALSE TRUE
## lbfgsb3c  0 0.75 1.5 2.25 7.875      2      2           0 FALSE TRUE
## lbfgsb3   0 0.75 1.5 2.25 7.875      2      2           0 FALSE TRUE
## Rvmmin    0 0.75 1.5 2.25 7.875      8      8           2 FALSE TRUE
## Rcgmin    0 0.75 1.5 2.25 7.875      9      8           0 FALSE TRUE
## Rtnmin    0 0.75 1.5 2.25 7.875      8      8           0 FALSE TRUE
##          xtimes
## L-BFGS-B  0.006
## lbfgsb3c  0.001
## lbfgsb3   0.001
## Rvmmin    0.003
## Rcgmin    0.001
## Rtnmin    0.003
```

```
bady <- bt.badsetup(nn)
lo <- bady$lower
up <- bady$upper
```

```
x0 <- bady$x
bady # This has x outside bounds, but method corrects.
```

```
## $x
## [1] -1.8  0.4  2.6  4.8
##
## $lower
## [1] 0.00 0.75 1.50 2.25
##
## $upper
## [1] 1.25 2.50 3.75 5.00
##
## $bdmsk
## [1] 1 1 0 1
```

```
## Should we notify user in a more aggressive fashion?
## optim()
## Put trace=3 to get full details
solbad0 <- opm(x0, bt.f, bt.g, lower=lo, upper=up, method=meths, control=list(trace=0))
```

```
## Warning in optimr(par, fn, gr, hess = hess, method = meth, lower = lower, : Parameter(s) changed to n

## Warning in optimr(par, fn, gr, hess = hess, method = meth, lower = lower, : Parameter(s) changed to n

## Warning in optimr(par, fn, gr, hess = hess, method = meth, lower = lower, : Parameter(s) changed to n

## Warning in optimr(par, fn, gr, hess = hess, method = meth, lower = lower, : Parameter(s) changed to n

## Warning in Rvmmin(par = spar, fn = efn, gr = egr, lower = slower, upper =
## supper, : Parameter out of bounds has been moved to nearest bound

## trace= 0

## Warning in optimr(par, fn, gr, hess = hess, method = meth, lower = lower, : Parameter(s) changed to n

## Warning in Rcgminb(par = spar, fn = efn, gr = egr, lower = slower, upper =
## supper, : x[1], set -1.8 to lower bound = 0

## Warning in Rcgminb(par = spar, fn = efn, gr = egr, lower = slower, upper =
## supper, : x[2], set 0.4 to lower bound = 0.75

## Warning in optimr(par, fn, gr, hess = hess, method = meth, lower = lower, : Parameter(s) changed to n
```

```
print(summary(solbad0, order=value))
```

```
##           p1   p2  p3   p4 value fevals gevals convergence  kkt1 kkt2
## L-BFGS-B  0 0.75 1.5 2.25 7.875      2      2           0 FALSE TRUE
## lbfgsb3c  0 0.75 1.5 2.25 7.875      2      2           0 FALSE TRUE
## lbfgsb3   0 0.75 1.5 2.25 7.875      2      2           0 FALSE TRUE
## Rvmmin    0 0.75 1.5 2.25 7.875      6      6           2 FALSE TRUE
## Rcgmin    0 0.75 1.5 2.25 7.875      5      4           0 FALSE TRUE
## Rtnmin    0 0.75 1.5 2.25 7.875      6      6           0 FALSE TRUE
##          xtimes
## L-BFGS-B  0.001
## lbfgsb3c  0.001
## lbfgsb3   0.001
## Rvmmin    0.001
## Rcgmin    0.001
```

```
## Rtnmin    0.001
```

Candlestick function

```r
# candlestick function
# J C Nash 2011-2-3
cstick.f<-function(x,alpha=100){
  x<-as.vector(x)
  r2<-crossprod(x)
  f<-as.double(r2+alpha/r2)
  return(f)
}

cstick.g<-function(x,alpha=100){
  x<-as.vector(x)
  r2<-as.numeric(crossprod(x))
  g1<-2*x
  g2 <- (-alpha)*2*x/(r2*r2)
  g<-as.double(g1+g2)
  return(g)
}
library(lbfgsb3c)
nn <- 2
x0 <- c(10,10)
lo <- c(1, 1)
up <- c(10,10)
print(x0)
```

```
## [1] 10 10
```

```r
c2o <- opm(x0, cstick.f, cstick.g, lower=lo, upper=up, method=meths, control=list(trace=0))
```

```
## trace= 0
```

```r
print(summary(c2o, order=value))
```

```
##               p1     p2 value fevals gevals convergence  kkt1  kkt2 xtimes
## lbfgsb3c 2.2361 2.2361    20     15     15           0  TRUE FALSE  0.001
## lbfgsb3  2.2361 2.2361    20     15     15           0  TRUE FALSE  0.001
## Rcgmin   2.2361 2.2361    20     10      6           0  TRUE FALSE  0.001
## L-BFGS-B 2.2361 2.2361    20     14     14           0  TRUE FALSE  0.009
## Rvmmin   1.0000 1.0000    52      2      2           2 FALSE FALSE  0.000
## Rtnmin   1.0000 1.0000    52      2      2           0 FALSE FALSE  0.001
```

```r
## meths <- c("L-BFGS-B", "lbfgsb3c", "Rvmmin", "Rcgmin", "Rtnmin")
## require(optimx)

## cstick2a <- opm(x0, cstick.f, cstick.g, method=meths, upper=up, lower=lo, control=list(kkt=FALSE))
## print(summary(cstick2a, par.select=1:2, order=value))
lo <- c(4, 4)
c2ob <- opm(x0, cstick.f, cstick.g, lower=lo, upper=up, method=meths, control=list(trace=0))
```

```
## trace= 0
```

```r
print(summary(c2ob, order=value))
```

```
##          p1 p2  value fevals gevals convergence  kkt1 kkt2 xtimes
## L-BFGS-B  4  4 35.125      2      2           0 FALSE TRUE  0.000
## lbfgsb3c  4  4 35.125      2      2           0 FALSE TRUE  0.000
## lbfgsb3   4  4 35.125      2      2           0 FALSE TRUE  0.001
## Rtnmin    4  4 35.125      2      2           0 FALSE TRUE  0.001
## Rvmmin    4  4 35.125      2      2           2 FALSE TRUE  0.000
## Rcgmin    4  4 35.125      3      2           0 FALSE TRUE  0.001
```

```
## cstick2b <- opm(x0, cstick.f, cstick.g, method=meths, upper=up, lower=lo, control=list(kkt=FALSE))
## print(summary(cstick2b, par.select=1:2, order=value))

nn <- 100
x0 <- rep(10, nn)
up <- rep(10, nn)
lo <- rep(1e-4, nn)
cco <- opm(x0, cstick.f, cstick.g, lower=lo, upper=up, method=meths, control=list(trace=0, kkt=FALSE))
```

```
## trace= 0
```

```
print(summary(cco, par.select=1:4, order=value))
```

```
##               p1      p2      p3      p4 value fevals gevals convergence
## L-BFGS-B 0.31623 0.31623 0.31623 0.31623    20     23     23           0
## lbfgsb3c 0.31623 0.31623 0.31623 0.31623    20     23     23           0
## lbfgsb3  0.31623 0.31623 0.31623 0.31623    20     23     23           0
## Rvmmin   0.31623 0.31623 0.31623 0.31623    20     46     23           0
## Rcgmin   0.31623 0.31623 0.31623 0.31623    20     24     18           0
## Rtnmin   0.31623 0.31623 0.31623 0.31623    20     18     18           0
##          kkt1 kkt2 xtimes
## L-BFGS-B   NA   NA  0.001
## lbfgsb3c   NA   NA  0.003
## lbfgsb3    NA   NA  0.002
## Rvmmin     NA   NA  0.050
## Rcgmin     NA   NA  0.013
## Rtnmin     NA   NA  0.002
```

Extended Rosenbrock function (from funconstrain)

```
# require(funconstrain) ## not in CRAN, so explicit inclusion of this function
# exrosen <- ex_rosen()
# exrosenf <- exrosen$fn
exrosenf <- function (par) {
    n <- length(par)
    if (n%%2 != 0) {
        stop("Extended Rosenbrock: n must be even")
    }
    fsum <- 0
    for (i in 1:(n/2)) {
        p2 <- 2 * i
        p1 <- p2 - 1
        f_p1 <- 10 * (par[p2] - par[p1]^2)
        f_p2 <- 1 - par[p1]
        fsum <- fsum + f_p1 * f_p1 + f_p2 * f_p2
    }
```

```
    fsum
}
# exroseng <- exrosen$gr
exroseng <- function (par) {
    n <- length(par)
    if (n%%2 != 0) {
        stop("Extended Rosenbrock: n must be even")
    }
    grad <- rep(0, n)
    for (i in 1:(n/2)) {
        p2 <- 2 * i
        p1 <- p2 - 1
        xx <- par[p1] * par[p1]
        yx <- par[p2] - xx
        f_p1 <- 10 * yx
        f_p2 <- 1 - par[p1]
        grad[p1] <- grad[p1] - 400 * par[p1] * yx - 2 * f_p2
        grad[p2] <- grad[p2] + 200 * yx
    }
    grad
}

exrosenx0 <- function (n = 20) {
    if (n%%2 != 0) {
        stop("Extended Rosenbrock: n must be even")
    }
    rep(c(-1.2, 1), n/2)
}


require(lbfgsb3c)
require(optimx)

## require(optimx)
for (n in seq(2,12, by=2)) {
  cat("ex_rosen try for n=",n,"\n")
  x0 <- exrosenx0(n)
  lo <- rep(-1.5, n)
  up <- rep(3, n)
##  cat("optim\n")
##  print(x0)
##  eo <- optim(x0, exrosenf, exroseng, lower=lo, upper=up, method="L-BFGS-B", control=list(trace=0))
##  print(eo)
##  cat("lbfgsb3c\n")
##  el <- lbfgsb3c(x0, exrosenf, exroseng, lower=lo, upper=up, control=list(trace=0))
##  print(el)
   erfg <- opm(x0, exrosenf, exroseng, method=meths, lower=lo, upper=up)
   print(summary(erfg, par.select=1:2, order=value))
}
```

```
## ex_rosen try for n= 2
## trace= 0
##          p1      p2      value fevals gevals convergence kkt1 kkt2 xtimes
## Rvmmin    1 1.00000 4.9797e-30     52     43           0 TRUE TRUE  0.003
```

```
## Rcgmin    1 1.00000 1.2897e-17    596    380           0 TRUE TRUE  0.010
## lbfgsb3c  1 1.00000 5.7131e-15     62     62           0 TRUE TRUE  0.003
## lbfgsb3   1 1.00000 5.7131e-15     62     62           0 TRUE TRUE  0.004
## L-BFGS-B  1 1.00000 3.8444e-14     51     51           0 TRUE TRUE  0.022
## Rtnmin    1 0.99999 1.2479e-11     49     49           0 TRUE TRUE  0.005
## ex_rosen try for n= 4
## trace= 0
##          p1      p2      value fevals gevals convergence kkt1 kkt2 xtimes
## Rvmmin    1 1.00000 0.0000e+00     67     54           2 TRUE TRUE  0.005
## Rcgmin    1 1.00000 1.2504e-16    714    459           0 TRUE TRUE  0.014
## lbfgsb3c  1 1.00000 1.1426e-14     62     62           0 TRUE TRUE  0.004
## lbfgsb3   1 1.00000 1.1426e-14     62     62           0 TRUE TRUE  0.004
## L-BFGS-B  1 1.00000 7.6888e-14     51     51           0 TRUE TRUE  0.001
## Rtnmin    1 0.99999 2.4845e-11     54     54           0 TRUE TRUE  0.005
## ex_rosen try for n= 6
## trace= 0
##          p1 p2      value fevals gevals convergence kkt1 kkt2 xtimes
## Rvmmin    1  1 2.4775e-30     90     71           0 TRUE TRUE  0.007
## Rtnmin    1  1 1.9703e-20     53     53           0 TRUE TRUE  0.005
## Rcgmin    1  1 1.3167e-15    160     80           0 TRUE TRUE  0.003
## lbfgsb3c  1  1 1.7139e-14     62     62           0 TRUE TRUE  0.004
## lbfgsb3   1  1 1.7139e-14     62     62           0 TRUE TRUE  0.004
## L-BFGS-B  1  1 1.1533e-13     51     51           0 TRUE TRUE  0.001
## ex_rosen try for n= 8
## trace= 0
##          p1 p2      value fevals gevals convergence kkt1 kkt2 xtimes
## Rvmmin    1  1 1.1155e-29    137     95           0 TRUE TRUE  0.010
## Rtnmin    1  1 8.6548e-17     58     58           0 TRUE TRUE  0.006
## Rcgmin    1  1 1.7654e-15    160     80           0 TRUE TRUE  0.004
## lbfgsb3c  1  1 2.2852e-14     62     62           0 TRUE TRUE  0.003
## lbfgsb3   1  1 2.2852e-14     62     62           0 TRUE TRUE  0.003
## L-BFGS-B  1  1 1.5378e-13     51     51           0 TRUE TRUE  0.001
## ex_rosen try for n= 10
## trace= 0
##          p1 p2      value fevals gevals convergence kkt1 kkt2 xtimes
## Rvmmin    1  1 6.2739e-30    140    106           0 TRUE TRUE  0.011
## Rcgmin    1  1 1.3182e-21    156     80           0 TRUE TRUE  0.005
## Rtnmin    1  1 5.1498e-20     53     53           0 TRUE TRUE  0.005
## lbfgsb3c  1  1 2.8566e-14     62     62           0 TRUE TRUE  0.004
## lbfgsb3   1  1 2.8566e-14     62     62           0 TRUE TRUE  0.004
## L-BFGS-B  1  1 1.9222e-13     51     51           0 TRUE TRUE  0.001
## ex_rosen try for n= 12
## trace= 0
##          p1 p2      value fevals gevals convergence kkt1 kkt2 xtimes
## Rvmmin    1  1 7.8640e-30    169    129           0 TRUE TRUE  0.014
## Rcgmin    1  1 1.1030e-21    156     80           0 TRUE TRUE  0.005
## Rtnmin    1  1 5.6832e-20     54     54           0 TRUE TRUE  0.006
## lbfgsb3c  1  1 3.4279e-14     62     62           0 TRUE TRUE  0.001
## lbfgsb3   1  1 3.4279e-14     62     62           0 TRUE TRUE  0.004
## L-BFGS-B  1  1 2.3067e-13     51     51           0 TRUE TRUE  0.001
```

Using compiled function code

While you may use the same interface as described in the writing R extensions to interface compiled code with this function, see L-BFGS-B, it is sometimes more convenient to use your own compiled code.

The following example shows how this is done using the file `jrosen.f`.

```
      subroutine rosen(n, x, fval)
      double precision x(n), fval, dx
      integer n, i
      fval = 0.0D0
      do 10 i=1,(n-1)
         dx = x(i + 1) - x(i) * x(i)
         fval = fval + 100.0 * dx * dx
         dx = 1.0 - x(i)
         fval = fval + dx * dx
 10   continue
      return
      end
```

Here is the example script. Note that we must have the file `jrosen.f` available.

```
# system("cd ~/temp")
system("R CMD SHLIB jrosen.f")
dyn.load("jrosen.so")
is.loaded("rosen")
```

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

```
x0 <- as.double(c(-1.2,1))
fv <- as.double(-999)
n <- as.double(2)
testf <- .Fortran("rosen", n=as.integer(n), x=as.double(x0), fval=as.double(fv))
testf
```

```
## $n
## [1] 2
##
## $x
## [1] -1.2  1.0
##
## $fval
## [1] 24.2
```

```
rrosen <- function(x) {
  fval <- 0.0
  for (i in 1:(n-1)) {
    dx <- x[i + 1] - x[i] * x[i]
    fval <- fval + 100.0 * dx * dx
    dx <- 1.0 - x[i]
    fval <- fval + dx * dx
  }
  fval
}

(rrosen(x0))
```

```
## [1] 24.2
```

```r
frosen <- function(x){
  nn <- length(x)
  if (nn > 100) { stop("max number of parameters is 100")}
  fv <- -999.0
  val <- .Fortran("rosen", n=as.integer(nn), x=as.double(x), fval=as.double(fv))
  val$fval # NOTE--need ONLY function value returned
}
# Test the funcion
tval <- frosen(x0)
str(tval)
```

```
##  num 24.2
```

```r
cat("Run with Nelder-Mead using R function\n")
```

```
## Run with Nelder-Mead using R function
```

```r
mynm <- optim(x0, rrosen, control=list(trace=0))
print(mynm)
```

```
## $par
## [1] 1.0003 1.0005
##
## $value
## [1] 8.8252e-08
##
## $counts
## function gradient
##      195       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```r
cat("\n\n Run with Nelder-Mead using Fortran function")
```

```
##
##
##  Run with Nelder-Mead using Fortran function
```

```r
mynmf <- optim(x0, frosen, control=list(trace=0))
print(mynmf)
```

```
## $par
## [1] 1.0003 1.0005
##
## $value
## [1] 8.8252e-08
##
## $counts
## function gradient
##      195       NA
##
## $convergence
## [1] 0
```

```
##
## $message
## NULL
```

```r
library(lbfgsb3c)
library(microbenchmark)
cat("try lbfgsb3c, no Gradient \n")
```

```
## try lbfgsb3c, no Gradient
```

```r
cat("R function\n")
```

```
## R function
```

```r
tlR<-microbenchmark(myopR <- lbfgsb3c(x0, rrosen, gr=NULL, control=list(trace=0)))
print(tlR)
```

```
## Unit: milliseconds
##                                                                   expr
##  myopR <- lbfgsb3c(x0, rrosen, gr = NULL, control = list(trace = 0))
##     min     lq   mean median     uq    max neval
##  14.188 14.603 15.499 14.885 16.288 18.945   100
```

```r
print(myopR)
```

```
## $par
## [1] 1.0000 1.0001
##
## $grad
## [1] -0.00034653  0.00020081
##
## $value
## [1] 8.5971e-10
##
## $counts
## [1] 74 74
##
## $convergence
## [1] 0
##
## $message
## [1] "CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH"
```

```r
cat("Fortran function\n")
```

```
## Fortran function
```

```r
tlF<-microbenchmark(myop <- lbfgsb3c(x0, frosen, gr=NULL, control=list(trace=0)))
print(tlF)
```

```
## Unit: milliseconds
##                                                                  expr    min
##  myop <- lbfgsb3c(x0, frosen, gr = NULL, control = list(trace = 0)) 16.514
##     lq   mean median     uq   max neval
##  18.26 18.961 18.484 20.269 23.86   100
```

```r
print(myop)
```

```
## $par
```

```
## [1] 1.0000 1.0001
##
## $grad
## [1] -0.00034653  0.00020081
##
## $value
## [1] 8.5971e-10
##
## $counts
## [1] 74 74
##
## $convergence
## [1] 0
##
## $message
## [1] "CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH"
```

In this example, Fortran is actually SLOWER than plain R.

References

Byrd, Richard H., Peihuang Lu, Jorge Nocedal, and Ci You Zhu. 1995a. "A Limited Memory Algorithm for Bound Constrained Optimization." *SIAM Journal on Scientific Computing* 16 (5): 1190–1208.

Byrd, Richard H., Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. 1995b. "A Limited Memory Algorithm for Bound Constrained Optimization." *SIAM J. Sci. Comput.* 16 (5). Philadelphia, PA, USA: Society for Industrial; Applied Mathematics: 1190–1208. https://doi.org/10.1137/0916069.

Coppola, Antonio, Brandon Stewart, and Naoaki Okazaki. 2014. *Lbfgs: Limited-Memory Bfgs Optimization*. https://CRAN.R-project.org/package=lbfgs.

Eddelbuettel, Dirk. 2013. *Seamless R and C++ Integration with Rcpp*. New York: Springer. https://doi.org/10.1007/978-1-4614-6868-4.

Eddelbuettel, Dirk, and James Joseph Balamuta. 2017. "Extending extitR with extitC++: A Brief Introduction to extitRcpp." *PeerJ Preprints* 5 (August): e3188v1. https://doi.org/10.7287/peerj.preprints.3188v1.

Eddelbuettel, Dirk, and Romain François. 2011. "Rcpp: Seamless R and C++ Integration." *Journal of Statistical Software* 40 (8): 1–18. https://doi.org/10.18637/jss.v040.i08.

Fidler, Matthew L, John C Nash, Ciyou Zhu, Richard Byrd, Jorge Nocedal, and Jose Luis Morales. 2018. *lbfgsb3c: Limited Memory Bfgs Minimizer with Bounds on Parameters with Optim() 'c' Interface*. https://CRAN.R-project.org/package=lbfgsb3c.

Liu, Dong C., and Jorge Nocedal. 1989. "On the Limited Memory BFGS Method for Large Scale Optimization." *Math. Program.* 45 (1-3): 503–28. https://doi.org/10.1007/BF01589116.

Lu, Peihuang, Jorge Nocedal, Ciyou Zhu, and Richard H. Byrd. 1994. "A Limited-Memory Algorithm for Bound Constrained Optimization." *SIAM Journal on Scientific Computing* 16: 1190–1208.

Morales, José Luis, and Jorge Nocedal. 2011. "Remark on Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound constrained optimization." *ACM Trans. Math. Softw.* 38 (1). New York, NY, USA: ACM: 7:1–7:4. http://doi.acm.org/10.1145/2049662.2049669.

Nash, John C, Ciyou Zhu, Richard Byrd, Jorge Nocedal, and Jose Luis Morales. 2015. *lbfgsb3: Limited Memory Bfgs Minimizer with Bounds on Parameters*. https://CRAN.R-project.org/package=lbfgsb3.

Zhu, Ciyou, Richard H. Byrd, Peihuang Lu, and Jorge Nocedal. 1997. "Algorithm 778: L-BFGS-B: Fortran Subroutines for Large-Scale Bound-Constrained Optimization." *ACM Trans. Math. Softw.* 23 (4). New York, NY, USA: ACM: 550–60. https://doi.org/10.1145/279232.279236.