Package ‘roptim’

October 14, 2022

Type Package

Title General Purpose Optimization in R using C++

Version 0.1.6

Author Yi Pan [aut, cre]

Maintainer Yi Pan <ypan1988@gmail.com>

Description Perform general purpose optimization in R using C++. A unified wrapper interface is provided to call C functions of the five optimization algorithms ('Nelder-Mead', 'BFGS', 'CG', 'L-BFGS-B' and 'SANN') underlying optim().

License GPL (>= 2)

Encoding UTF-8

SystemRequirements C++11

Imports Rcpp (>= 0.12.14)

LinkingTo Rcpp, RcppArmadillo

RoxygenNote 7.1.1

URL https://github.com/ypan1988/roptim/

BugReports https://github.com/ypan1988/roptim/issues

Suggests R.rsp, testthat (>= 3.0.0)

VignetteBuilder R.rsp

Config/testthat/edition 3

NeedsCompilation yes

Repository CRAN

Date/Publication 2022-08-06 10:30:02 UTC

R topics documented:

`example1_rosen_bfgs` *Example 1: Minimize Rosenbrock function using BFGS*

Description

Minimize Rosenbrock function using BFGS.

Usage

```
example1_rosen_bfgs(print = TRUE)
```

Arguments

`print`	whether the results should be printed.

Examples

```
fr <- function(x) {   ## Rosenbrock Banana function
 x1 <- x[1]
 x2 <- x[2]
 100 * (x2 - x1 * x1)^2 + (1 - x1)^2
}
grr <- function(x) { ## Gradient of 'fr'
 x1 <- x[1]
 x2 <- x[2]
 c(-400 * x1 * (x2 - x1 * x1) - 2 * (1 - x1),
   200 *      (x2 - x1 * x1))
}
res <- optim(c(-1.2,1), fr, grr, method = "BFGS", control = list(trace=TRUE), hessian = TRUE)
res

## corresponding C++ implementation:
example1_rosen_bfgs()
```

`example1_rosen_grad_hess_check`
Example 1: Gradient/Hessian checks for the implemented C++ class of Rosenbrock function

Description

Gradient/Hessian checks for the implemented C++ class of Rosenbrock function.

Usage

```
example1_rosen_grad_hess_check()
```

`example1_rosen_nograd_bfgs`
Example 1: Minimize Rosenbrock function (with numerical gradient) using BFGS

Description

Minimize Rosenbrock function (with numerical gradient) using BFGS.

Usage

```
example1_rosen_nograd_bfgs()
```

Examples

```
fr <- function(x) {   ## Rosenbrock Banana function
 x1 <- x[1]
 x2 <- x[2]
 100 * (x2 - x1 * x1)^2 + (1 - x1)^2
}

optim(c(-1.2,1), fr, NULL, method = "BFGS")

## corresponding C++ implementation:
example1_rosen_nograd_bfgs()
```

`example1_rosen_other_methods` *Example 1: Minimize Rosenbrock function using other methods*

Description

Minimize Rosenbrock function using other methods ("Nelder-Mead"/"CG"/ "L-BFGS-B"/"SANN").

Usage

```
example1_rosen_other_methods()
```

Examples

```
fr <- function(x) {   ## Rosenbrock Banana function
 x1 <- x[1]
 x2 <- x[2]
 100 * (x2 - x1 * x1)^2 + (1 - x1)^2
}
grr <- function(x) { ## Gradient of 'fr'
 x1 <- x[1]
 x2 <- x[2]
 c(-400 * x1 * (x2 - x1 * x1) - 2 * (1 - x1),
   200 *      (x2 - x1 * x1))
}

optim(c(-1.2,1), fr)

## These do not converge in the default number of steps
optim(c(-1.2,1), fr, grr, method = "CG")
optim(c(-1.2,1), fr, grr, method = "CG", control = list(type = 2))

optim(c(-1.2,1), fr, grr, method = "L-BFGS-B")

optim(c(-1.2,1), fr, method = "SANN")

## corresponding C++ implementation:
example1_rosen_other_methods()
```

`example2_tsp_sann` *Example 2: Solve Travelling Salesman Problem (TSP) using SANN*

Description

Solve Travelling Salesman Problem (TSP) using SANN.

Usage

```
example2_tsp_sann(distmat, x)
```

Arguments

distmat	a distance matrix for storing all pair of locations.
x	initial route.

Examples

```
## Combinatorial optimization: Traveling salesman problem
library(stats) # normally loaded

eurodistmat <- as.matrix(eurodist)

distance <- function(sq) {  # Target function
 sq2 <- embed(sq, 2)
 sum(eurodistmat[cbind(sq2[,2], sq2[,1])])
}

genseq <- function(sq) {  # Generate new candidate sequence
 idx <- seq(2, NROW(eurodistmat)-1)
 changepoints <- sample(idx, size = 2, replace = FALSE)
 tmp <- sq[changepoints[1]]
 sq[changepoints[1]] <- sq[changepoints[2]]
 sq[changepoints[2]] <- tmp
 sq
}

sq <- c(1:nrow(eurodistmat), 1)  # Initial sequence: alphabetic
distance(sq)
# rotate for conventional orientation
loc <- -cmdscale(eurodist, add = TRUE)$points
x <- loc[,1]; y <- loc[,2]
s <- seq_len(nrow(eurodistmat))
tspinit <- loc[sq,]

plot(x, y, type = "n", asp = 1, xlab = "", ylab = "",
    main = "initial solution of traveling salesman problem", axes = FALSE)
arrows(tspinit[s,1], tspinit[s,2], tspinit[s+1,1], tspinit[s+1,2],
      angle = 10, col = "green")
text(x, y, labels(eurodist), cex = 0.8)

## The original R optimization:
## set.seed(123) # chosen to get a good soln relatively quickly
## res <- optim(sq, distance, genseq, method = "SANN",
##              control = list(maxit = 30000, temp = 2000, trace = TRUE,
##              REPORT = 500))
## res  # Near optimum distance around 12842

## corresponding C++ implementation:
set.seed(4)  # chosen to get a good soln relatively quickly
```

```
res <- example2_tsp_sann(eurodistmat, sq)

tspres <- loc[res$par,]
plot(x, y, type = "n", asp = 1, xlab = "", ylab = "",
    main = "optim() 'solving' traveling salesman problem", axes = FALSE)
arrows(tspres[s,1], tspres[s,2], tspres[s+1,1], tspres[s+1,2],
      angle = 10, col = "red")
text(x, y, labels(eurodist), cex = 0.8)
```

example3_flb_25_dims_box_con

Example 3: Minimize a function using L-BFGS-B with 25-dimensional box constrained

Description

Minimize a function using L-BFGS-B with 25-dimensional box constrained.

Usage

```
example3_flb_25_dims_box_con()
```

Examples

```
flb <- function(x)
{ p <- length(x); sum(c(1, rep(4, p-1)) * (x - c(1, x[-p])^2)^2) }
## 25-dimensional box constrained
optim(rep(3, 25), flb, NULL, method = "L-BFGS-B",
     lower = rep(2, 25), upper = rep(4, 25)) # par[24] is *not* at boundary

## corresponding C++ implementation:
example3_flb_25_dims_box_con()
```

example4_wild_fun

Example 4: Minimize a "wild" function using SANN and BFGS

Description

Minimize a "wild" function using SANN and BFGS.

Usage

```
example4_wild_fun()
```

Examples

```
## "wild" function , global minimum at about -15.81515
fw <- function (x)
 10*sin(0.3*x)*sin(1.3*x^2) + 0.00001*x^4 + 0.2*x+80
plot(fw, -50, 50, n = 1000, main = "optim() minimising 'wild function'")

res <- optim(50, fw, method = "SANN",
            control = list(maxit = 20000, temp = 20, parscale = 20))
res
## Now improve locally {typically only by a small bit}:
(r2 <- optim(res$par, fw, method = "BFGS"))
points(r2$par,  r2$value,  pch = 8, col = "red", cex = 2)

## corresponding C++ implementation:
example4_wild_fun()
```

---

roptim *roptim*

Description

Perform general purpose optimization in R using C++. A unified wrapper interface is provided to call C functions of the five optimization algorithms ('Nelder-Mead', 'BFGS', 'CG', 'L-BFGS-B' and 'SANN') underlying optim().

Author(s)

Yi Pan

Index