Package 'n1qn1'

May 9, 2026

Title Port of the 'Scilab' 'n1qn1' Module for Unconstrained BFGS Optimization

Version 6.0.1-14

Maintainer Matthew Fidler <matthew.fidler@gmail.com>

Description Provides 'Scilab' 'n1qn1'. This takes more memory than traditional L-BFGS. The n1qn1 routine is useful since it allows prespecification of a Hessian. If the Hessian is near enough the truth in optimization it can speed up the optimization problem. The algorithm is described in the 'Scilab' optimization documentation located at <https://www.scilab.org/sites/default/files/optimization_in_scilab.pdf>. This version uses manually modified code from 'f2c' to make this a C only binary.

URL https://github.com/nlmixr2/n1qn1c, https://nlmixr2.github.io/n1qn1c/

BugReports https://github.com/nlmixr2/n1qn1c/issues

Depends R (>= 3.2)

Imports Rcpp (>= 0.12.3)

Suggests testthat (>= 3.0.0), covr

License CeCILL-2

Biarch true

NeedsCompilation yes

LinkingTo RcppArmadillo (>= 0.5.600.2.0), Rcpp (>= 0.12.3)

Encoding UTF-8

RoxygenNote 7.3.3

Config/testthat/edition 3

Author Matthew Fidler [aut, cre],
Wenping Wang [aut],
Claude Lemarechal [aut, ctb],
Joseph Bonnans [ctb],
Bill Denney [ctb] (ORCID: <https://orcid.org/0000-0002-5759-428X>),
Jean-Charles Gilbert [ctb],

Claudia Sagastizabal [ctb],
Stephen L. Campbell, [ctb],
Jean-Philippe Chancelier [ctb],
Ramine Nikoukhah [ctb],
Dirk Eddelbuettel [ctb],
Bruno Jofret [ctb],
INRIA [cph]

Repository CRAN

Date/Publication 2026-04-06 05:10:29 UTC

Contents

`.n1qn1ptr` *This gives the function pointers in the n1qn1 library*

Description

Using this will allow C-level linking by function pointers instead of abi.

Usage

```
.n1qn1ptr()
```

Value

list of pointers to the n1qn1 functions

Author(s)

Matthew L. Fidler

Examples

```
.n1qn1ptr()
```

n1qn1 *n1qn1 optimization*

Description

This is an R port of the n1qn1 optimization procedure in scilab.

Usage

```
n1qn1(
  call_eval,
  call_grad,
  vars,
  environment = parent.frame(1),
  ...,
  epsilon = .Machine$double.eps,
  max_iterations = 100,
  nsim = 100,
  imp = 0,
  invisible = NULL,
  zm = NULL,
  restart = FALSE,
  assign = FALSE,
  print.functions = FALSE
)
```

Arguments

call_eval	Objective function
call_grad	Gradient Function
vars	Initial starting point for line search
environment	Environment where call_eval/call_grad are evaluated.
...	Ignored additional parameters.
epsilon	Precision of estimate
max_iterations	Number of iterations
nsim	Number of function evaluations
imp	Verbosity of messages.
invisible	boolean to control if the output of the minimizer is suppressed.
zm	Prior Hessian (in compressed format; This format is output in c.hess).
restart	Is this an estimation restart?
assign	Assign hessian to c.hess in environment environment? (Default FALSE)
print.functions	Boolean to control if the function value and parameter estimates are echoed every time a function is called.

Value

The return value is a list with the following elements:

- `value` The value at the minimized function.
- `par` The parameter value that minimized the function.
- `H` The estimated Hessian at the final parameter estimate.
- `c.hess` Compressed Hessian for saving curvature.
- `n.fn` Number of function evaluations
- `n.gr` Number of gradient evaluations

Author(s)

C. Lemarechal, Stephen L. Campbell, Jean-Philippe Chancelier, Ramine Nikoukhah, Wenping Wang & Matthew L. Fidler

Examples

```
## Rosenbrock's banana function
n=3; p=100

fr = function(x)
{
    f=1.0
    for(i in 2:n) {
        f=f+p*(x[i]-x[i-1]**2)**2+(1.0-x[i])**2
    }
    f
}

grr = function(x)
{
    g = double(n)
    g[1]=-4.0*p*(x[2]-x[1]**2)*x[1]
    if(n>2) {
        for(i in 2:(n-1)) {
            g[i]=2.0*p*(x[i]-x[i-1]**2)-4.0*p*(x[i+1]-x[i]**2)*x[i]-2.0*(1.0-x[i])
        }
    }
    g[n]=2.0*p*(x[n]-x[n-1]**2)-2.0*(1.0-x[n])
    g
}

x = c(1.02,1.02,1.02)
eps=1e-3
n=length(x); niter=100L; nsim=100L; imp=3L;
nzm=as.integer(n*(n+13)/2)
zm=double(nzm)

(op1 <- n1qn1(fr, grr, x, imp=3))
```

```
## Note there are 40 function calls and 40 gradient calls in the above optimization

## Now assume we know something about the Hessian:
c.hess <- c(797.861115,
            -393.801473,
            -2.795134,
            991.271179,
            -395.382900,
            200.024349)
c.hess <- c(c.hess, rep(0, 24 - length(c.hess)))

(op2 <- n1qn1(fr, grr, x,imp=3, zm=c.hess))

## Note with this knowledge, there were only 29 function/gradient calls

(op3 <- n1qn1(fr, grr, x, imp=3, zm=op1$c.hess))

## The number of function evaluations is still reduced because the Hessian
## is closer to what it should be than the initial guess.

## With certain optimization procedures this can be helpful in reducing the
## Optimization time.
```

Index