Introduction to `nloptr`: an R interface to NLopt *

Jelmer Ypma

July 28, 2011

Abstract

This document describes how to use `nloptr`, which is an R interface to NLopt. NLopt is a free/open-source library for nonlinear optimization started by Steven G. Johnson, providing a common interface for a number of different free optimization routines available online as well as original implementations of various other algorithms. The NLopt library is available under the GNU Lesser General Public License (LGPL), and the copyrights are owned by a variety of authors.

1 Introduction

NLopt addresses general nonlinear optimization problems of the form:

$$\begin{aligned} & \min_{x \in R^n} f(x) \\ s.t. \quad & g(x) \leq 0 \\ & h(x) = 0 \\ & x_L \leq x \leq x_U \end{aligned}$$

where $f(\cdot)$ is the objective function and x represents the n optimization parameters. This problem may optionally be subject to the bound constraints (also called box constraints), x_L and x_U. For partially or totally unconstrained problems the bounds can take values $-\infty$ or ∞. One may also optionally have m nonlinear inequality constraints (sometimes called a nonlinear programming problem), which can be specified in $g(\cdot)$, and equality constraints that can be specified in $h(\cdot)$. Note that not all of the algorithms in NLopt can handle constraints.

This vignette describes how to formulate minimization problems to be solved with the R interface to NLopt. If you want to use the C interface directly or are interested in the Matlab interface, there are other sources of documentation avialable. Some of the information here has been taken from the NLopt website[1], where more details are available. All credit for implementing the C code for the different algorithms availalbe in NLopt should go to the respective authors. See the website[2] for information on how to cite NLopt and the algorithms you use.

*This package should be considered in beta and comments about any aspect of the package are welcome. This document is an R vignette prepared with the aid of `Sweave`, Leisch(2002). Financial support of the UK Economic and Social Research Council through a grant (RES-589-28-0001) to the ESRC Centre for Microdata Methods and Practice (CeMMAP) is gratefully acknowledged.

[1] `http://ab-initio.mit.edu/nlopt`

[2] `http://ab-initio.mit.edu/wiki/index.php/Citing_NLopt`

2 Installation

This package is on CRAN and can be installed from within R using.

```
> install.packages("nloptr")
```

You should now be able to load the R interface to NLopt and read the help.

```
> library('nloptr')
> ?nloptr
```

3 Minimizing the Rosenbrock Banana function

As a first example we will solve an unconstrained minimization problem. The function we look at is the Rosenbrock Banana function

$$f(x) = 100 \left(x_2 - x_1^2\right)^2 + (1 - x_1)^2 ,$$

which is also used as an example in the documentation for the standard R optimizer `optim`. The gradient of the objective function is given by

$$\nabla f(x) = \begin{pmatrix} -400 \cdot x_1 \cdot (x_2 - x_1^2) - 2 \cdot (1 - x_1) \\ 200 \cdot (x_2 - x_1^2) \end{pmatrix} .$$

Not all of the algorithms in NLopt need gradients to be supplied by the user. We will show examples with and without supplying the gradient. After loading the library

```
> library(nloptr)
```

we start by specifying the objective function and its gradient

```
> ## Rosenbrock Banana function
> eval_f <- function(x) {
      return( 100 * (x[2] - x[1] * x[1])^2 + (1 - x[1])^2 )
  }
> ## Gradient of Rosenbrock Banana function
> eval_grad_f <- function(x) {
      return( c( -400 * x[1] * (x[2] - x[1] * x[1]) - 2 * (1 - x[1]),
                  200 * (x[2] - x[1] * x[1]) ) )
  }
```

We define initial values

```
> # initial values
> x0 <- c( -1.2, 1 )
```

and then minimize the function using the `nloptr` command. This command runs some checks on the supplied inputs and returns an object with the exit code of the solver, the optimal value of the objective function and the solution. Before we can minimize the function we need to specify which algorithm we want to use

```
> opts <- list("algorithm"="NLOPT_LD_LBFGS",
              "xtol_rel"=1.0e-8)
```

Here we use the L-BFGS algorithm (Nocedal, 1980; Liu & Nocedal, 1989). The characters `LD` in the algorithm show that this algorithm looks for local minima (`L`) using a derivative-based (`D`) algorithm. Other algorithms look for global (`G`) minima, or they don't need derivatives (`N`). We also specified the termination criterium in terms of the relative x-tolerance. Other termination criteria are available. We then solve the minimization problem using

```
> # solve Rosenbrock Banana function
> res <- nloptr( x0=x0,
                 eval_f=eval_f,
                 eval_grad_f=eval_grad_f,
                 opts=opts)
```

We can see the results by printing the resulting object.

```
> print( res )
Call:

nloptr(x0 = x0, eval_f = eval_f, eval_grad_f = eval_grad_f, opts = opts)



Minimization using NLopt version 2.2.4

NLopt solver status: 1 ( NLOPT_SUCCESS: Generic success
return value. )

Number of Iterations....: 56
Termination conditions:  xtol_rel: 1e-08
Number of inequality constraints:  0
Number of equality constraints:    0
Current value of objective function:  7.35727226897802e-23
Current value of controls: 1 1
```

Sometimes the objective function and its gradient contain common terms. To economize on calculations, we can return the objective and its gradient in a list. For the Rosenbrock Banana function we have for instance

```
> ## Rosenbrock Banana function and gradient in one function
> eval_f_list <- function(x) {
     common_term <- x[2] - x[1] * x[1]
     return( list( "objective" = 100 * common_term^2 + (1 - x[1])^2,
                   "gradient"  = c( -400 * x[1] * common_term - 2 * (1 - x[1]),
                                    200 * common_term) ) )
 }
```

which we minimize using

```
> res <- nloptr( x0=x0,
                 eval_f=eval_f_list,
                 opts=opts)
> print( res )
Call:
nloptr(x0 = x0, eval_f = eval_f_list, opts = opts)


Minimization using NLopt version 2.2.4

NLopt solver status: 1 ( NLOPT_SUCCESS: Generic success
return value. )

Number of Iterations....: 56
Termination conditions:  xtol_rel: 1e-08
Number of inequality constraints:  0
Number of equality constraints:    0
Current value of objective function:  7.35727226897802e-23
Current value of controls: 1 1
```

This gives the same results as before.

4 Minimization with inequality constraints

This section shows how to minimize a function subject to inequality constraints. This example is the same as the one used in the tutorial on the NLopt website. The problem we want to solve is

$$\begin{aligned} & \min_{x \in R^n} \sqrt{x_2} \\ s.t. \quad & x_2 \geq 0 \\ & x_2 \geq (a_1 x_1 + b_1)^3 \\ & x_2 \geq (a_2 x_1 + b_2)^3, \end{aligned}$$

where $a_1 = 2$, $b_1 = 0$, $a_2 = -1$, and $b_2 = 1$. In order to solve this problem, we first have to re-formulate the constraints to be of the form $g(x) \leq 0$. Note that the first constraint is a bound on x_2, which we will add later. The other two constraints can be re-written as

$$\begin{aligned} (a_1 x_1 + b_1)^3 - x_2 &\leq 0 \\ (a_2 x_1 + b_2)^3 - x_2 &\leq 0. \end{aligned}$$

First we define R functions to calculate the objective function and its gradient

```
> # objective function
> eval_f0 <- function( x, a, b ){
     return( sqrt(x[2]) )
 }
> # gradient of objective function
```

```
> eval_grad_f0 <- function( x, a, b ){
     return( c( 0, .5/sqrt(x[2]) ) )
 }
```

If needed, these can of course be calculated in the same function as before. Then we define the two constraints and the jacobian of the constraints

```
> # constraint function
> eval_g0 <- function( x, a, b ) {
     return( (a*x[1] + b)^3 - x[2] )
 }
> # jacobian of constraint
> eval_jac_g0 <- function( x, a, b ) {
     return( rbind( c( 3*a[1]*(a[1]*x[1] + b[1])^2, -1.0 ),
                    c( 3*a[2]*(a[2]*x[1] + b[2])^2, -1.0 ) ) )
 }
```

Note that all of the functions above depend on additional parameters, `a` and `b`. We have to supply specific values for these when we invoke the optimization command. The constraint function `eval_g0` returns a vector with in this case the same length as the vectors `a` and `b`. The function calculating the jacobian of the constraint should return a matrix where the number of rows equal the number of constraints (in this case two). The number of columns should equal the number of control variables (two in this case as well).

After defining values for the parameters

```
> # define parameters
> a <- c(2,-1)
> b <- c(0, 1)
```

we can minimize the function subject to the constraints with the following command

```
> # Solve using NLOPT_LD_MMA with gradient information supplied in separate function
> res0 <- nloptr( x0=c(1.234,5.678),
                  eval_f=eval_f0,
                  eval_grad_f=eval_grad_f0,
                  lb = c(-Inf,0),
                  ub = c(Inf,Inf),
                  eval_g_ineq = eval_g0,
                  eval_jac_g_ineq = eval_jac_g0,
                  opts = list("algorithm" = "NLOPT_LD_MMA",
                              "print_level" = 2,
                              "check_derivatives" = TRUE,
                              "check_derivatives_print" = "all"),
                  a = a,
                  b = b )
[1] "No termination criterium specified, using default (relative x-tolerance = 1e-4)"
Checking gradients of objective function.
Derivative checker results: 0 error(s) detected.

  eval_grad_f[ 1 ] = 0.000000e+00 ~ 0.000000e+00   [0.000000e+00]
```

```
  eval_grad_f[ 2 ] = 2.098323e-01 ~ 2.098323e-01   [1.422937e-09]

Checking gradients of inequality constraints.
Derivative checker results: 0 error(s) detected.

  eval_jac_g_ineq[ 1, 1 ] =  3.654614e+01 ~  3.654614e+01   [1.667794e-08]
  eval_jac_g_ineq[ 2, 1 ] = -1.642680e-01 ~ -1.642680e-01   [2.103453e-07]
  eval_jac_g_ineq[ 1, 2 ] = -1.000000e+00 ~ -1.000000e+00   [0.000000e+00]
  eval_jac_g_ineq[ 2, 2 ] = -1.000000e+00 ~ -1.000000e+00   [0.000000e+00]

iteration: 1
        f(x) = 2.382855
        g(x) = ( 9.354647,-5.690813 )
iteration: 2
        f(x) = 2.356135
        g(x) = ( -0.122989,-5.549587 )
iteration: 3
        f(x) = 2.245864
        g(x) = ( -0.531886,-5.038655 )
iteration: 4
        f(x) = 2.019102
        g(x) = ( -3.225104,-3.931194 )
iteration: 5
        f(x) = 1.740934
        g(x) = ( -2.676260,-2.761137 )
iteration: 6
        f(x) = 1.404206
        g(x) = ( -1.674056,-1.676216 )
iteration: 7
        f(x) = 1.022295
        g(x) = ( -0.748790,-0.748792 )
iteration: 8
        f(x) = 0.685203
        g(x) = ( -0.173206,-0.173206 )
iteration: 9
        f(x) = 0.552985
        g(x) = ( -0.009495,-0.009496 )
iteration: 10
        f(x) = 0.544354
        g(x) = ( -0.000026,-0.000025 )
iteration: 11
        f(x) = 0.544331
        g(x) = ( -0.000000,0.000000 )
> print( res0 )
Call:

nloptr(x0 = c(1.234, 5.678), eval_f = eval_f0, eval_grad_f = eval_grad_f0,
    lb = c(-Inf, 0), ub = c(Inf, Inf), eval_g_ineq = eval_g0,
    eval_jac_g_ineq = eval_jac_g0, opts = list(algorithm = "NLOPT_LD_MMA",
```

```
        print_level = 2, check_derivatives = TRUE, check_derivatives_print = "all"),
    a = a, b = b)


Minimization using NLopt version 2.2.4

NLopt solver status: 4 ( NLOPT_XTOL_REACHED: Optimization
stopped because xtol_rel or xtol_abs (above) was reached.
)

Number of Iterations....: 11
Termination conditions:  relative x-tolerance = 1e-4 (DEFAULT)
Number of inequality constraints:  2
Number of equality constraints:    0
Current value of objective function:  0.544331050645946
Current value of controls: 0.3333333 0.2962963
```

Here we supplied lower bounds for x_2 in `lb`. There are no upper bounds for both control variables, so we supply `Inf` values. If we don't supply lower or upper bounds, plus or minus infinity is chosen by default. The inequality constraints and its jacobian are defined using `eval_g_ineq` and `eval_jac_g_ineq`. Not all algorithms can handle inequality constraints, so we have to specifiy one that does, `NLOPT_LD_MMA` (Svanberg, 2002).

We also specify the option `print_level` to obtain output during the optimization process. For the available `print_level` values, see `?nloptr`. Setting the `check_derivatives` option to `TRUE`, compares the gradients supplied by the user with a finite difference approximation in the initial point (`x0`). When this check is run, the option `check_derivatives_print` can be used to print all values of the derivative checker (`all` (default)), only those values that result in an error (`errors`) or no output (`none`), in which case only the number of errors is shown. The tolerance that determines if a difference between the analytic gradient and the finite difference approximation results in an error can be set using the option `check_derivatives_tol` (default = 1e-04). The first column shows the value of the analytic gradient, the second column shows the value of the finite difference approximation, and the third column shows the relative error. Stars are added at the front of a line if the relative error is larger than the specified tolerance.

Finally, we add all the parameters that have to be passed on to the objective and constraint functions, `a` and `b`.

We can also use a different algorithm to solve the same minimization problem. The only thing we have to change is the algorithm that we want to use, in this case `NLOPT_LN_COBYLA`, which is an algorithm that doesn't need gradient information (Powell, 1994, 1998).

```
> # Solve using NLOPT_LN_COBYLA without gradient information
> res1 <- nloptr( x0=c(1.234,5.678),
                  eval_f=eval_f0,
                  lb = c(-Inf,0),
                  ub = c(Inf,Inf),
                  eval_g_ineq = eval_g0,
```

```
                opts = list("algorithm"="NLOPT_LN_COBYLA"),
                a = a,
                b = b )
[1] "No termination criterium specified, using default (relative x-tolerance = 1e-4)"
> print( res1 )
Call:

nloptr(x0 = c(1.234, 5.678), eval_f = eval_f0, lb = c(-Inf, 0),
    ub = c(Inf, Inf), eval_g_ineq = eval_g0, opts = list(algorithm = "NLOPT_LN_COBYLA"),
    a = a, b = b)


Minimization using NLopt version 2.2.4

NLopt solver status: 4 ( NLOPT_XTOL_REACHED: Optimization
stopped because xtol_rel or xtol_abs (above) was reached.
)

Number of Iterations....: 31
Termination conditions:  relative x-tolerance = 1e-4 (DEFAULT)
Number of inequality constraints:  2
Number of equality constraints:    0
Current value of objective function:  0.544242301658176
Current value of controls: 0.3333292 0.2961997
```

5 Notes

The `.R` scripts in the `tests` directory contain more examples. For instance, `hs071.R` and `systemofeq.R` show how to solve problems with equality constraints. See also `http://ab-initio.mit.edu/wiki/index.php/NLopt_Algorithms#Augmented_Lagrangia` for more details. Please let me know if you're missing any of the features that are implemented in NLopt.

Sometimes the optimization procedure terminates with a message `maxtime was reached` without evaluating the objective function. Submitting the same problem again usually solves this problem.

References

Johnson, S. G. (n.d.). *The NLopt nonlinear-optimization package.* (`http://ab-initio.mit.edu/nlopt`)

Leisch, F. (2002). Sweave: Dynamic generation of statistical reports using literate data analysis. In W. Härdle & B. Rönz (Eds.), *Compstat 2002 — proceedings in computational statistics* (pp. 575–580). Physica Verlag, Heidelberg. Available from `http://www.stat.uni-muenchen.de/ leisch/Sweave` (ISBN 3-7908-1517-9)

Liu, D. C., & Nocedal, J. (1989). On the limited memory BFGS method for large scale optimization. *Math. Programming*, *45*, 503–528.

Nocedal, J. (1980). Updating quasi-Newton matrices with limited storage. *Math. Comput.*, *35*, 773–782.

Powell, M. J. D. (1994). A direct search optimization method that models the objective and constraint functions by linear interpolation. In S. Gomez & J.-P. Hennart (Eds.), *Advances in optimization and numerical analysis* (pp. 51–67). Kluwer Academic, Dordrecht.

Powell, M. J. D. (1998). Direct search algorithms for optimization calculations. *Acta Numerica*, *7*, 287–336.

Svanberg, K. (2002). A class of globally convergent optimization methods based on conservative convex separable approximations. *SIAM J. Optim.*, *12*(2), 555–573.