Rdonlp2 - an R interface to DONLP2

Ryuichi Tamura(`ry.tamura @ gmail.com`)

June 7, 2007(Version 0.3-1)

Contents

1 Abount This Package

Rdonlp2 is an R package to use Peter Spellucci's DONLP2 from R. DONLP2 is the copyrighted software written by Peter Spellucci for solving nonlinear programming problems.

DONLP2 is available from:

- `http://plato.la.asu.edu/donlp2.html`

Rdonlp2 is a wrapper for ANSI-C version of DONLP2(also called DONLP3):

- `http://plato.asu.edu/ftp/donlp2/donlp2_intv_dyn.tar.gz`

Current Rdonlp2 package is available from:

- `http://arumat.net/Rdonlp2/Rdonlp2_0.3-1.zip`(Windows Binary)
- `http://arumat.net/Rdonlp2/Rdonlp2_0.3-1_R_i386-apple-darwin8.8.1.tar.gz`(OSX Universal Binary)
- `http://arumat.net/Rdonlp2/Rdonlp2_0.3-1.tar.gz`(Source File)

Since Rdonlp2 is simply wrapper program, user is required to refer PDF manual included in `donlp2_intv_dyn.tar.gz` for the detail of the algorithm.

As for condition of use, Please refer Section8.

2 Problem Definition

R Package Rdonlp2 solves following nonlinear minimization problem with linear, nonlinear constraints as well as parameter bounds:

$$\begin{aligned}
\min_x f(x) \quad & \text{subject to} \quad x \in \mathcal{S} \\
\mathcal{S} \in \quad \{ \quad & x \in R^n, \\
& x_l \le x \le x_u, \\
& b_l \le Ax \le b_u, \\
& c_l \le c(x) \le c_u \quad \},
\end{aligned}$$

where, $f(x)$ is a continuous function, x_l, x_u are bounds for parameters(x), b_l, b_u are bounds for linear combinations Ax(linear constraints), and c_l, c_u are bounds for nonlinear function $c(x)$(nonlinear constraints). To describe equality constraint or parameter constancy, let lower and upper bounds for constraint be equal.

2.1 R function donlp2()

Rdonlp provides single function for this problem:

```
donlp2 <- function(par, fn,
                   par.upper=rep(+Inf, length(par)),
                   par.lower=rep(-Inf, length(par)),

                   A = NULL,
                   lin.upper=rep(+Inf, length(par)),
                   lin.lower=rep(-Inf, length(par)),

                   nlin = list(),
                   nlin.upper=rep(+Inf, length(nlin)),
                   nlin.lower=rep(-Inf, length(nlin)),

                   control=donlp2.control(),
                   control.fun=function(lst){return(TRUE)},
                   env=.GlobalEnv, name="Rdonlp2")
```

where,

`fn` - the objective function to be minimized. Currently, `fn` must take only one argument, and the parameter vector(`par`) will be passed to `fn` during the optimization. The first element of return value must be the evaluated value.

`par` - parameter vector(vector object)

`par.upper`, `par.lower` - upper and lower bounds for parameter vector, respectively. Their length must equal to `length(par)`. If some elements are unbounded, specify `+Inf` or `-Inf` explicitly.

`A` - the matrix object that represents linear constraints. Its columns must be equal to `length(par)`, and its rows must be equal to the number of linear constraints.

`lin.upper`, `lin.lower` - upper and lower bounds for linear constraints, respectively. Their length must equal to the number of linear constraints. If some elements are unbounded, specify `+Inf` or `-Inf` explicitly.

`nlin` - list object whose elements are functions that represents nonlinear constraints. rule for argument and return value is the same as `fn`, i.e., these functions take only one arugument(`par`), and return a vector object whose first element is the evaluated value.

`lin.upper`, `lin.lower` - upper and lower bounds for nonlinear constraints, respectively. Their length must equal to `length(nlin)`. If some elements are unbounded, specify `+Inf` or `-Inf` explicitly.

`control` - control parameters that defines the behavior of DONLP2. See below for details.

`control.fun` - `donlp2()` reports a group of optimization parameters in every iteration(see below for details). This (read-only) information can be available within `control.fun()`, in which user can decide whether the optimization should be iterrupted. Set its return value to `FALSE` to interrupt `donlp2()`.

`env` - the environment in which objective, constraint, control functions are evaluated.

`name` - an character object that specify file name(without extension) of output file. DONLP2 can output following 2 files(`name`.pro and `name`.mes) in working directory which contain detailed information during the optimization.

- `name`.pro: the results of optimization and other information during the optimazation will be written. the latter depends on the values of `te0`,`te1`,`te2`,`te3`.
- `name`.mes: other message from DONLP2 (warnings for ill-conditions, etc) will be written. This will be helpful if your R code does not work correctly.

3 Details

3.1 Initial Values

Although intial values(given to `par`) should be carefully determined by user, DONLP2 has the feature to correct initial values **automatically** that violate given constraints.

3.2 Objective Function and its Gradients

Objective function should be implemented so that parameter vector is the only argument and the first element of return value is numeric. Typically, it looks like:

```
objective.fun <- function(par){
  # calculation with par, and
  # results are stored to ans
  :
  ans # return value
}

:

ret <- donlp2(par=par, fn=objective.fun, ....)
```

User can implement gradient function to improve accuracy and efficiency. Gradient function should be implement such that parameter vector is the only argument and return value is vector of length equal to `length(par)`. Then, assign it to the attibute `gr` of objective function:

```
# par is vector of length n
grad.fun <- function(par){

  c(v1, v2, ..., vn)
}
# assign grad.fun to objective.fun's gr attribute
attr(objective.fn, "gr") <- grad.fun
```

3.3 Bounds and Equality Constraints

Bounds for parameter, linear and nonlinear constraints are given as vector of appropriate length(`par.upper`,`par.lower`,`lin.upper`,`lin.lower`,`nlin.upper`,`nlin.lower`). If some parameter or constraints are bounded from below (above), then specify `+Inf`(`-Inf`), respectively.

Set upper and lower bounds to be equal if equality constraint is needed.

```
# par[1]<0, 0<par[2]<1, par[3]>1
par.lower <- c(-Inf, 0,    1)
par.upper <- c(   0, 1, +Inf)

# two linear constraints on two parameters
# (1) par[1]+par[2]=0
# (2) par[1]-2*par[2]+10>0
lin.lower <- c(0,  -10)
lin.upper <- c(0, +Inf)
```

3.4 Linear Constraints

Bounds arguments and the coefficient matrix `A` represents the linear constraints. Every row of `A` stands for linear combination of parameters:

```
# two linear constraints on two parameters
# (1) par[1]+par[2]=0
# (2) par[1]-2*par[2]+10>0
lin.lower <- c(0,  -10)
lin.upper <- c(0, +Inf)

A = rbind( c(1, 1),   # 1st linear compination
           c(1,-2) )  # 2nd linear combination
```

3.5 Nonlinear Constraints and its Gradients

Nonlinear constraints are represented as their bounds given to `nlin.upper` and `nlin.lower`, and user-defined function(and gradients). The way to implement function and gradients are the same as objective function and its gradients(see Section 3.2):

```
# Nonlinear constraints: 1 constraint on 2 parameters
# par[1]*par[2] = 1
nlcon1 <- function(par){
  par[1]*par[2]
}
nlcon1.gr <- function(par){
  c(par[2], par[1])
}
attr(nlcon1, "gr")<-nlcon1.gr

nlin.upper = c(1)
nlin.lower = c(1)
:
ret <- donlp2(par, fn,
              nlin=list(nlcon1),
              nlin.upper=nlin.upper,nlin.lower=nlin.lower,.....)
```

All the nonlinear constraint function are collected in a list(`nlin`).

3.6 Numerical Gradients

User can omit the implementation of gradients. In this case one of 3 algorithm fro numerical differentiation provided by DONLP2 will be performed. If there are `n` parameters,

1. the forward difference: requires `n` additional evaluations of function(`difftype=1`).

2. the central difference: requires `2n` additional evaluations of function(`difftype=2`).
3. a sixth order approximation computing a Richardson extrapolation of the three symmetric difference: requires `6n` additional evaluations of function(`difftype=3`).

By default, Rdonlp2 uses 3rd algorithm(most acculate, but quite costly). User can change this by the control variable(below) `difftype`.

Currently, if user want to use analytical gradients, he **must** implement **all** of the gradients for **both** objective function and nonlinear constraint functions. If one of them are not implemented, Rdonlp2 gives up using any gradient function and uses numerical method instead.

3.7 Control Variables

User can control the behavior of `donlp2()` by `donlp2.control()`. `donlp2.control()`returns a group of default control parameters as list object, so user change some of them by giving `tag=value` pairs as arguments. Currently following tags(control variables) are available (values in () are the defaults).

- Settings
  - `iterma` (`4000`) - maximum number of iterations
  - `nstep` (`20`) - maximum number of tries in the backtracking allowed.
  - `fnscale`(`1`) - set `-1` for maximization instead of minimization. values other than `1,-1` are not recommended.
- Tunings and erfomance of the optmization
  - `tau0` (`1.0`) - the positive amount how much any constaint other than abound can be violated. A small `tau0` diminishes the efficiency of DONLP2, while a large `tau0` may degarde the reliability of the code.
  - `tau` (`0.1`) - gives a weight between descent for `fn` and infeasibility and is also used as a safety parameter for chosing the penalty weigths. It can be chosen larger zero at will, but useful values are between 0.1 and 1. The smaller tau, the more may `fn` be scaled down. Tau is also used as an additive increase for the penalty-weights. Therefore it should not be chosen too large, since that degrades the performance.
  - `del0` (`1.0`) - The positive amount by which constraints are considered binding. If too small, the indentification of correct sets of binding constraints may be delayed. If too large, DONLP2 will escape to the full regularlized SQP method(quite costly). Good values are in [0.01, 1.0]
- Termination criteria
  - `epsx`(`1e-5`) - successful temination is indicated if the Kuhn-Tucker criteria are satisfied within the value.
  - `delmin`(`0.1*del0`) - constraints are considered as sufficiently satisfied if absolute values of their violation are less than the value.
  - `epsdif`(`1e-8`) - relative precision in the gradients.

  - `nreset.multiplier` (1) - maximum number of steps (`nreset.multiplier` times `n`) until a "restart" of the accumulated quasi-newton-update is tried. Value should be integer between 1 and 4.

- Numerical differentiation
  - `difftype` (3) - See Section3.6.
  - `taubnd` (`0.1`) - The positive amount by which bounds may be violated if numerical differention is used.
  - `epsfcn` (`1e-16`) - relative precision of the function evaluation routine.
  - `hessian` (`FALSE`) - if `TRUE`, caliculate numeric Hessian matrix evaluated at the optimum by numerical differentiation specified in `difftype`.

- Ouputs on console or files
  - `report`(`FALSE`) - if `TRUE`, a list object which contains detailed information will be passed to `control.fun()`. See Section4.1.
  - `rep.freq`(1) - the frequency of report. the report will be passed to `control.fun` every `rep.freq` iterations.
  - `te0` (`TRUE`) - if `TRUE` one-line-output for every step is printed on R console.
  - `te1` (`FALSE`) - if `TRUE` post-mortem-dump of accumlated information is printed on R console. Note in Rdonlp2 the same information will be passed to `control.fun()`(See Section4).
  - `te2` (`FALSE`) - if `TRUE`, more detailed information is "pretty-printed" on R console.
  - `te3` (`FALSE`) - if `TRUE`, and output file is specified in `donlp2`, also print the gradients and Newton-Raphson update in upper trianglar matrix in the `.pro` file.
  - `silent` (`FALSE`) - if `TRUE`, `donlp2()` runs quietly, i.e. nothing is output to R console and `.pro, .mes` files are never created even if you specify file name in the `name` argument of `donlp2()`.
  - `intakt` (`TRUE`) - if `TRUE`, various information(depends on `te0,te1,te2`) from current iteration step is output to R console.

Some of parameteters listed above are not well documented in this tutorial. Please refer the original PDF manual included in DONLP distribution for details.

4 Information during the Optimization

User may want to know what happens during the optimation, and if some parameters are 'undesirable' he may stop the execution. Rdonlp2 provides the way to access the information via `control.fun(lst)`. On each iterantion, 35 parameters that tell us how the optimization is working are passed to `control.fun(lst)`. The last four parameters are not reported until the optimization has finished. Parameters are collected into single list object, so user can easily access some of them by specifying the `tags`. Complete tag list is given Section 4.1.

`control.fun()` should return `TRUE` if user want to continue and `FALSE` if user want to interrupt.

```
## Example 1
## keep track of current lagrange multiplier values
mycontrol <- function(lst){
 print(lst$u)
 TRUE # return TRUE to continue execution
}

# tell donlp2 to use mycontrol()
donlp2(.....,control.fun=mycontrol,....)

## Example 2(useless example)
## force to terminate optimization after 10 iterations
mycontrol2 <- function(lst){
  lst$step.nr <= 10 # return FALSE when step.nr>11
}
# tell donlp2 to use mycontrol2()
donlp2(.....,control.fun=mycontrol2,....)
```

4.1 Tag list

Some of tags listed here are not well documented in this tutorial. Please refer the original PDF manual included in DONLP distribution for details.

- `par` - current value of parameter vector.
- `u` - current value of langrange multipliers.
- `w` - current value of penalty terms.
- `gradf` - current gradient vector.
- `step.nr` - step number(total number of iterations when finished).
- `fx` - current value of `fn`.
- `scf` - scaling of `fn`.
- `psi` - the weighted penalty term.
- `upsi` - the unweighted penalty term(L1 norm of constraint vector).
- `del.k.1` - bound for currently active constraints.
- `b2n0` - weighted L2 norm of projected gradients.
- `b2n` - L2 norm of gradients based on `del.k.1`
- `nr` - number of binding constraints.
- `sing` - value other than `-1` indicates working set is singular

- `umin` - infinity norm of negative part of inequalities multipliers.
- `not.used` - always `0`(curretnly not used).
- `cond.r` - condition number of diagonal part of qr decomposition of normalized gradients of binding constraints.
- `cond.h` - condition number of diagonal of cholesky factor of updated full Hessian.
- `scf0` - the relative damping of tangential component if `upsi > tau0/2`.
- `xnorm` - L2 norm of `par`.
- `dnorm` - unscaled L2 norm of `d`, correction from eqp/qp subproblem.
- `phase` -
  - `-1`: infeasibility improvement phase.
  - `0`: initial optimization.
  - `1`: binding constraints unchanged
  - `2`: `d` small, maratos correction in use
- `c.k` - number of decreases of penalty weights
- `wmax` - infinity norm of weights
- `sig.k` - stepsize from unidimensional minimization(backtracking).
- `cfincr` - number of objective function evaluations for stepsize algorithm.
- `dirder` - scaled directional derivative of penalty function along `d`.
- `dscal` - scaling factor for `d`.
- `cosphi` - cosine of arc between `d` and previous `d`.
- `violis` - number of constraints not binding at `par` but hit during line search.
- `hesstype` - type of update for Hessian:
  - `1`: normal P&M-BFGS update
  - `0`: update suppressed
  - `-1`: restart with scaled unit matrix
  - `2`: standard BFGS
  - `3`: BFGS modified by Powell's Device
- `modbfgs` - modification factor for damping the projector int the BFGS or pantoja-mayne update.
- `modnr` -modification factor for damping the quasi-newton-relation in BFGS.
- `qpterm` -

  - `0`: if `sing=-1`: temination indicator of the QP solver
  - `1`: successful
  - `-1`: `tau` becomes larger than `tauqp` without slack variables becoming sufficiently small.
  - `-2`: infeasible QP problem(theoretically impossible).

- `tauqp` - weight of slack variables in QP solver
- `infeas` - L1 norm of slack variables in QP solver

5 Value from donlp2()

The return value from `donlp2()` is the list object with 38(35+3; if `hessian=FALSE`(default)) or 39(35+4; if `hessian=TRUE`) elements with specified tags. The 35 pareameters are identical to those listed in Section 4.1, and the rest parameters are:

- `nr.update`: the approximated newton-raphson updates in upper triangular matrix.
- `hessian`(if `hessian=TRUE` in `donlp2.control()`): numeric Hessian matrix evaluated at the final value `par`.
- `runtime`: the elapsed time for the optimization.
- `message`: the termination message. See 5.1.

5.1 The termination Reason

When the optimization finishes, DONLP2 returns one of 19 messages listed below. They are classified to following 3 groups, last of which user need to decide the result is 'reasonable' and accestable.

- 1.-10. : irregular case
- 11.-13.: successful
- 14.-19.: successful, but the precision may be very poor.

1. `"constraint evaluation returns error with current point"`
2. `"objective evaluation returns error with current point"`
3. `"qpsolver:  working set singular in dual extended qp "`
4. `"qpsolver:  extended qp-problem seemingly infeasible "`
5. `"qpsolver:  no descent for infeas from qp for tau=tau_max"`
6. `"qpsolver:  on exit correction small, infeasible point"`
7. `"stepsizeselection:  computed d from qp not a dir.  of descent"`
8. `"more than maxit iteration steps"`

9. `"stepsizeselection:  no acceptable stepsize in [sigsm,sigla]"`
10. `"small correction from qp, infeasible point"`
11. `"kt-conditions satisfied, no further correction computed"`
12. `"computed correction small, regular case "`
13. `"stepsizeselection:  x almost feasible, dir.  deriv.  very small"`
14. `"kt-conditions (relaxed) satisfied, singular point"`
15. `"very slow primal progress, singular or illconditoned problem"`
16. `"more than nreset small corrections in x "`
17. `"correction from qp very small, almost feasible, singular "`
18. `"numsm small differences in penalty function,terminate"`
19. `"user required termination"`

Some of messages listed above are not well documented in this tutorial. Please refer the original PDF manual included in DONLP distribution for details.

6 Examples

Example C source files are included in the original DONLP2 distribution. In this section, we rewrite some of them in R and show you how to code constrained optimization problem with Rdonlp2.

6.1 examples/simple.c: linear constraint

$$\min_{x,y} x^2 + y^2 \quad \text{subject to} \quad 0 < x, y < 100, \quad x + y = 1 \tag{1}$$

with intial value: $(x, y) = (-10, 10)$. Note that initial value is infeasible because $x = -10 \notin (0, 100)$.

This problem has 2 parameters, 2 parameter bounds, 1 linear equality constraint. R script looks like:

```
p <- c(-10,10)
par.l <- c(0,0); par.u <- c(100,100)

lin.u <- 1; lin.l <- 1
A <- t(c(1,1))

fn <- function(x){
  x[1]^2+x[2]^2
}
ret <- donlp2(p, fn, par.lower=par.l, par.upper=par.u,
              A=A, lin.u=lin.u, lin.l=lin.l, name="simple")
```

Note that `A` must be represented as row vector(`1x2`) with single linear constraint.
Also we use numerical gradients.

Since control variables are all default values(specifically `te0=TRUE` and `silent=FALSE`), running the script outputs detailed information on console:

```
    1 fx=   0.000000e+00 upsi=  5.9e+01 b2n= -1.0e+00 umi=  0.0e+00 nr   1 si-1
    2 fx=   0.000000e+00 upsi=  2.0e+01 b2n= -1.0e+00 umi=  0.0e+00 nr   2 si-1
    3 fx=   1.000000e+00 upsi=  0.0e+00 b2n=  4.4e-16 umi=  0.0e+00 nr   2 si-1
simple

     n=         2    nlin=         1    nonlin=         0

  epsx= 1.000e-05 sigsm= 3.293e-10

startvalue
  5.0000000e+01   1.0000000e+01

  eps=  1.08e-19  tol=  0.00e+00 del0=  1.00e+00 delm=  1.00e-06 tau0=  1.00e+00
  tau=  1.00e-01   sd=  1.00e-01   sw=  2.27e-13  rho=  1.00e-06 rho1=  1.00e-10
 scfm=  1.00e+04  c1d=  1.00e-02 epdi=  1.00e-16
  nre=         4 anal=        0
taubnd=  1.00e+00 epsfcn=  1.00e-16 difftype=3

 termination reason:
 KT-conditions satisfied, no further correction computed
 evaluations of f                         3
 evaluations of grad f                    2
 evaluations of constraints               6
 evaluations of grads of constraints      0
 final scaling of objective           1.000000e+00
 norm of grad(f)                      1.414214e+00
 lagrangian violation                 4.718134e-14
 feasibility violation                0.000000e+00
 dual feasibility violation           0.000000e+00
 optimizer runtime sec's              0.000000e+00


 optimal value of f =   5.00000000000000e-01

 optimal solution  x =
  4.99999999999991e-01  5.00000000000009e-01

  multipliers are relativ to scf=1
  nr.    constraint     multiplier norm(grad) or 1
    1   5.0000000e-01    0.0000000e+00
    2   9.9500000e+01    0.0000000e+00
    3   5.0000000e-01    0.0000000e+00
    4   9.9500000e+01    0.0000000e+00
    5   0.0000000e+00    1.0000000e+00   1.3906925e-309
    6   0.0000000e+00    0.0000000e+00
 evaluations of restrictions and their gradients
 (     6,     0)
 last estimate of cond.nr. of active gradients   1.414e+00

 last estimate of cond.nr. of approx.  hessian   1.000e+00
iterative steps total              3
# of restarts                      0
# of full regular updates          1
# of updates                       1
# of regularized full SQP-steps    0
```

If user want to access parameter values, simply

```
> ret$par
[1] 0.5 0.5
```

6.2 examples/simple2.c: nonlinear constraint

Second example shows optimization problem with parameter bounds and nonlinear constraint.

$$\min_{x,y} x^2 + y^2 \quad \text{subject to} \quad -100 < x, y < 100, \quad xy = 2 \tag{2}$$

with intial value: $(x, y) = (10, 10)$. We give gradient functions for objective and nonlinear constraint(`dfn()` and `dnlcon()`, respectively).

```
p <- c(10,10)
par.l <- c(-100,-100); par.u <- c(100,100)
nlin.l <- nlin.u <- 2

fn <- function(x){
  x[1]^2+x[2]^2
}
dfn <- function(x){
  c(2*x[1], 2*x[2])
}
attr(fn, "gr") <- dfn

nlcon <- function(x){
  x[1]*x[2]
}
dnlcon <- function(x){
  c(x[2], x[1])
}
attr(nlcon, "gr") <- dnlcon

ret<-donlp2(p, fn, par.u=par.u, par.l=par.l,
            nlin=list(nlcon), nlin.u=nlin.u, nlin.l=nlin.l)
```

6.3 example/hs211.c

This problem uses all types of constraints:

$$\min_{x_i, i=1\ldots10} 5.04x_1 + 0.035x_2 + 10x_3 + 3.36x_5 - 0.063x_4x_7$$

subject to:

$$\begin{aligned}
h_1(x) &= 1.22x_4 - x_1 - x_5 = 0\\
h_2(x) &= 98000x_3/(x_4x_9 + 1000x_3) - x_6 = 0\\
h_3(x) &= (x_2 + x_5)/x_1 - x_8 = 0\\
g_1(x) &= 35.82 - 0.222x_{10} - bx_9 \geq 0, b = 0.9\\
g_2(x) &= -133 + 3x_7 - ax_10 \geq 0, a = 0.99\\
g_3(x) &= -g_1(x) + x_9(1/b - b) \geq 0\\
g_4(x) &= -g_2(x) + (1/a - a)x_{10} \geq 0\\
g_5(x) &= 1.12x_1 + 0.13167x_1x_8 - 0.00667x_1x_8^2 - ax_4 \geq 0\\
g_6(x) &= 57.425 + 1.098x_8 - 0.038x_8^2 + 0.325x_6 - ax_7 \geq 0\\
g_7(x) &= -g_5(x) + (1/a - a)x_4 \geq 0\\
g_8(x) &= -g_6(x) + (1/a - a)x_7 \geq 0
\end{aligned}$$

$$\begin{aligned}
0.00001 \leq x_1 &\leq 2000\\
0.00001 \leq x_2 &\leq 16000\\
0.00001 \leq x_3 &\leq 120\\
0.00001 \leq x_4 &\leq 5000\\
0.00001 \leq x_5 &\leq 2000\\
85 \leq x_6 &\leq 93\\
90 \leq x_7 &\leq 95\\
3 \leq x_8 &\leq 12\\
1.2 \leq x_9 &\leq 4\\
145 \leq x_{10} &\leq 162
\end{aligned}$$

We have 10 parameter bounds, 5 linear constraints(arranging terms):

$$\begin{aligned}
h_1 &\rightarrow 1.22x_4 - x_1 - x_5 = 0\\
g_1 &\rightarrow -0.222x_10 - bx_9 \geq -35.82, b = 0.9\\
g_2 &\rightarrow 3x_7 - ax_10 \geq 133, a = 0.99\\
g_3 &\rightarrow x_9(1/b - b + b) + 0.222x_{10} \geq 35.82\\
g_4 &\rightarrow -3x_7 + (1/a - a + a)x_{10} \geq -133,
\end{aligned}$$

1 of which(h_1) is equality constraint, and 6 nonlinear constraints:

$$\begin{aligned}
h_2 &\rightarrow 98000x_3/(x_4x_9 + 1000x_3) - x_6 = 0\\
h_3 &\rightarrow (x_2 + x_5)/x_1 - x_8 = 0\\
g_5 &\rightarrow 1.12x_1 + 0.13167x_1x_8 - 0.00667x_1x_8^2 - ax_4 \geq 0\\
g_6 &\rightarrow 57.425 + 1.098x_8 - 0.038x_8^2 + 0.325x_6 - ax_7 \geq 0\\
g_7 &\rightarrow -g_5(x) + (1/a - a)x_4 \geq 0\\
g_8 &\rightarrow -g_6(x) + (1/a - a)x_7 \geq 0,
\end{aligned}$$

2 of which(h_2, h_3) are equality constraint.

R code will be follows:

```
a <- 0.99; b <- 0.9
##
## Objective Function
##
fn <- function(par){
  x1 <- par[1]; x2 <- par[2]; x3 <- par[3]; x4 <- par[4]; x5 <- par[5]
  x6 <- par[6]; x7 <- par[7]; x8 <- par[8]; x9 <- par[9]; x10 <- par[10]

  5.04*x1 + 0.035*x2 + 10*x3 +3.36*x5 - 0.063*x4*x7
}
##
## Parameter Bounds
##
par.l <- c(rep(1e-5, 5), 85, 90, 3, 1.2, 145)
par.u <- c(2000, 16000, 120, 5000, 2000, 93, 95, 12, 4, 162)

##
## Constraints
##
linbd  <- matrix(0, nr=5, nc=2)
nlinbd <- matrix(0, nr=6, nc=2)

## linear equality
linbd[1,] <- c(0,0)          # h1
linbd[2,] <- c(-35.82, Inf)  # g1
linbd[3,] <- c(133, Inf)     # g2
linbd[4,] <- c(35.82,Inf)    # g3
linbd[5,] <- c(-133, Inf)    # g4

## nonlinear equality
h2 <- function(par){
  x1 <- par[1]; x2 <- par[2]; x3 <- par[3]; x4 <- par[4]; x5 <- par[5]
  x6 <- par[6]; x7 <- par[7]; x8 <- par[8]; x9 <- par[9]; x10 <- par[10]

  98000*x3/(x4*x9+1000*x3)-x6
}
nlinbd[1,] <- c(0,0)

## nonlinear equality
h3 <- function(par){
  x1 <- par[1]; x2 <- par[2]; x3 <- par[3]; x4 <- par[4]; x5 <- par[5]
  x6 <- par[6]; x7 <- par[7]; x8 <- par[8]; x9 <- par[9]; x10 <- par[10]

  (x2+x5)/x1 - x8
}
nlinbd[2,] <- c(0,0)
```

```
## nonlinear inequality
g5 <- function(par){
  x1 <- par[1]; x2 <- par[2]; x3 <- par[3]; x4 <- par[4]; x5 <- par[5]
  x6 <- par[6]; x7 <- par[7]; x8 <- par[8]; x9 <- par[9]; x10 <- par[10]

  1.12*x1 + 0.13167*x1*x8 - 0.00667*x1*x8^2 - a*x4
}
nlinbd[3,] <- c(0,Inf)

## nonlinear inequality
g6 <- function(par){
  x1 <- par[1]; x2 <- par[2]; x3 <- par[3]; x4 <- par[4]; x5 <- par[5]
  x6 <- par[6]; x7 <- par[7]; x8 <- par[8]; x9 <- par[9]; x10 <- par[10]

  57.425 + 1.098*x8 - 0.038*x8^2 + 0.325*x6 - a*x7
}
nlinbd[4,] <- c(0,Inf)

## nonlinear inequality
g7 <- function(par){
  x1 <- par[1]; x2 <- par[2]; x3 <- par[3]; x4 <- par[4]; x5 <- par[5]
  x6 <- par[6]; x7 <- par[7]; x8 <- par[8]; x9 <- par[9]; x10 <- par[10]

  -g5(par) + (1/a-a)*x4
}
nlinbd[5,] <- c(0,Inf)

## nonlinear inequality
g8 <- function(par){
  x1 <- par[1]; x2 <- par[2]; x3 <- par[3]; x4 <- par[4]; x5 <- par[5]
  x6 <- par[6]; x7 <- par[7]; x8 <- par[8]; x9 <- par[9]; x10 <- par[10]

  -g6(par) + (1/a-a)*x7
}
nlinbd[6,] <- c(0,Inf)

## A is 5(linear constraints) x 10(params) matrix
A <- rbind(c(-1, 0, 0, 1.22,-1, 0,  0, 0,        0,         0), #h1
           c( 0, 0, 0,    0, 0, 0,  0, 0,       -b,    -0.222), #g1
           c( 0, 0, 0,    0, 0, 0,  3, 0,        0,        -a), #g2
           c( 0, 0, 0,    0, 0, 0,  0, 0,(1/b-b)+b,     0.222), #g3
           c( 0, 0, 0,    0, 0, 0, -3, 0,        0,(1/a-a)+a))  #g4

## initial values
p0 <- c(1745, 12e3, 11e1, 3048, 1974, 89.2, 92.8, 8, 3.6, 145)

## control variables
cntl <- donlp2.control(del0=0.2, tau0=1.0, tau=0.1, taubnd=5e-6)

## start constrained optimization
```

```
ret <- donlp2(par=p0, fn=fn,
              par.u=par.u, par.l=par.l,
              A=A,
              lin.u=linbd[,2], lin.l=linbd[,1],
              nlin=list(h2,h3,g5,g6,g7,g8),
              nlin.upper=nlinbd[,2], nlin.lower=nlinbd[,1], name="alkylation",
              control=cntl)
```

Since gradient functions are not implemented in the program and the numerical differentiation algorithm is `difftype=3`(default), it takes about 0.90 sec to finish the computation on my machine(Intel CoreDuo 2G, Memory 2G), while original C version does within 0.1 sec.

7 Bugs

DONLP2 provides much more 'fine-tuning' parameters than those exported to Rdonlp2. So if you want other parameters that should be exported, please e-mail me. Also, any comments of suggestions are highly welcome.

8 Copyright

Original DONLP2: