nlmrt-vignette

John C. Nash

September 18, 2017

Background

This vignette discusses the R package `nlmrt`, that aims to provide computationally robust tools for nonlinear least squares problems. Note that R already has the `nls()` function to solve nonlinear least squares problems, and this function has a large repertoire of tools for such problems. However, it is specifically NOT indicated for problems where the residuals are small or zero. Furthermore, it frequently fails to find a solution if starting parameters are provided that are not close enough to a solution. The tools of `nlmrt` are very much intended to cope with both these issues.

The functions are also intended to provide stronger support for bounds constraints and to introduce the capability for **masks**, that is, parameters that are fixed for a given run of the function.

`nlmrt` tools generally do not return the large nls-style object. However, we do provide a tool `wrapnls` that will run either `nlxb` followed by a call to `nls`. The call to `nls` is adjusted to use the `port` algorithm if there are bounds constraints.

1 An example problem and its solution

Let us try an example initially presented by [5] and developed by [2]. This is a model for the regrowth of pasture. We set up the computation by putting the data for the problem in a data frame, and specifying the formula for the model. This can be as a formula object, but I have found that saving it as a character string seems to give fewer difficulties. Note the ” ” that implies ”is modeled by”. There must be such an element in the formula for this package (and for `nls()`). We also specify two sets of starting parameters, that is, the `ones` which is a trivial (but possibly unsuitable) start with all parameters set to 1, and `huetstart` which was suggested in [2]. Finally we load the routines in the package `nlmrt`.

```
> options(width=60)
> pastured <- data.frame(
+ time=c(9, 14, 21, 28, 42, 57, 63, 70, 79),
```

```
+ yield= c(8.93, 10.8, 18.59, 22.33, 39.35,
+          56.11, 61.73, 64.62, 67.08))
> regmod <- "yield ˜ t1 - t2*exp(-exp(t3+t4*log(time)))"
> ones <- c(t1=1, t2=1, t3=1, t4=1) # all ones start
> huetstart <- c(t1=70, t2=60, t3=0, t4=1)
> require(nlmrt)
```

Let us now call the routine `nlsmnqb` (even though we are not specifying bounds). We try both starts.

```
> anmrt <- nlxb(regmod, start=ones, trace=FALSE, data=pastured)
> print(anmrt)
nlmrt class object: x
residual sumsquares =  4648.1  on  9 observations
    after  3    Jacobian and  4 function evaluations
  name            coeff          SE       tstat      pval      gradient    JSingval
t1               38.8378             NA         NA         NA  -3.039e-11           3
t2               1.00007             NA         NA         NA  -7.748e-10   1.437e-09
t3              0.998202             NA         NA         NA   1.889e-08   2.275e-16
t4              0.996049             NA         NA         NA    4.15e-08   4.933e-26
```

```
> anmrtx <- try(nlxb(regmod, start=huetstart, trace=FALSE, data=pastured))
> print(strwrap(anmrtx))
 [1] "c(0.480575683702448, 0.669264006079334,"
 [2] "-2.28426563497325, 0.843862687207341,"
 [3] "0.734652618487168, 0.0665106492947132,"
 [4] "-0.985862291968445, -0.0250879549069225,"
 [5] "0.500350456693326)"
 [6] "c(1, 1, 1, 1, 1, 1, 1, 1, 1, -0.981556726091092,"
 [7] "-0.948171282599527, -0.869750270888722,"
 [8] "-0.758399834057036, -0.484261107837448,"
 [9] "-0.223408951427337, -0.149363030476145,"
[10] "-0.086933293312168, -0.0385258954067725,"
[11] "1.12712321032759, 3.11275223693953, 7.48692917929397,"
[12] "12.9373484175607, 21.6609765596453, 20.654376815193,"
[13] "17.5183401160421, 13.0985419560081, 7.73883739451332,"
[14] "2.47654281941789, 8.21473160617155, 22.7941238760067,"
[15] "43.1098907467039, 80.9615739893344, 83.5067043689986,"
[16] "72.5808432835117, 55.6490931778844, 33.8144464340477)"
[17] "44"
[18] "32"
[19] "c(69.9553722026374, 61.6818319271118,"
[20] "-9.2088020481334, 2.37778402563407)"
[21] "8.37588360361953"
[22] "c(-Inf, -Inf, -Inf, -Inf)"
```

```
[23] "c(Inf, Inf, Inf, Inf)"
[24] "integer(0)"
```

Note that the standard `nls()` of R fails to find a solution from either start.

```
> anls <- try(nls(regmod, start=ones, trace=FALSE, data=pastured))
> print(strwrap(anls))
[1] "Error in nlsModel(formula, mf, start, wts) : singular"
[2] "gradient matrix at initial parameter estimates"
```

```
> anlsx <- try(nls(regmod, start=huetstart, trace=FALSE, data=pastured))
> print(strwrap(anlsx))
[1] "Error in nls(regmod, start = huetstart, trace ="
[2] "FALSE, data = pastured) : singular gradient"
```

In both cases, the `nls()` failed with a 'singular gradient'. This implies the Jacobian is effectively singular at some point. The Levenberg-Marquardt stabilization used in `nlxb` avoids this particular issue by augmenting the Jacobian until it is non-singular. The details of this common approach may be found elsewhere [4, Algorithm 23].

There are some other tools for R that aim to solve nonlinear least squares problems. We have not yet been able to successfully use the INRA package `nls2`. This is a quite complicated package and is not installable as a regular R package using `install.packages()`. Note that there is a very different package by the same name on CRAN by Gabor Grothendieck.

2 The `nls` solution

We can call `nls` after getting a potential nonlinear least squares solution using `nlxb`. Package `nlmrt` has function `wrapnls` to allow this to be carried out automatically. Thus,

```
> awnls <- wrapnls(regmod, start=ones, data=pastured, control=list(rofftest=FALSE))
> print(awnls)
Nonlinear regression model
  model: yield ~ t1 - t2 * exp(-exp(t3 + t4 * log(time)))
   data: data
   t1    t2    t3    t4 
69.96 61.68 -9.21  2.38 
 residual sum-of-squares: 8.38

Number of iterations to convergence: 0 
Achieved convergence tolerance: 7.15e-08
```

```
> cat("Note that the above is just the nls() summary result.\n")
Note that the above is just the nls() summary result.
```

3 Problems specified by residual functions

The model expressions in R , such as

`yield ~ t1 - t2*exp(-exp(t3+t4*log(time)))`

are an extremely helpful feature of the language. Moreover, they are used to compute symbolic or automatic derivatives, so we do not have to rely on numerical approximations for the Jacobian of the nonlinar least squares problem. However, there are many situations where the expression structure is not flexible enough to allow us to define our residuals, or where the construction of the residuals is simply too complicated. In such cases it is helpful to have tools that work with R functions.

Once we have an R function for the residuals, we can use the safeguarded Marquardt routine `nlfb` from package `nlmrt` or else the routine `nls.lm` from package `minpack.lm` [1]. The latter is built on the Minpack Fortran codes of [3] implemented by Kate Mullen. `nlfb` is written entirely in R , and is intended to be quite aggessive in ensuring it finds a good minimum. Thus these two approaches have somewhat different characteristics.

Let us consider a slightly different problem, called WEEDS. Here the objective is to model a set of 12 data points (density y of weeds at annual time points tt) versus the time index. (A minor note: use of `t` rather than `tt` in R may encourage confusion with the transpose function `t()`, so I tend to avoid plain `t`.) The model suggested was a 3-parameter logistic function,

$y_{model} = b_1/(1 + b_2 exp(-b_3 tt))$

and while it is possible to use this formulation, a scaled version gives slightly better results

$y_{model} = 100b_1/(1 + 10b_2 exp(-0.1b_3 tt))$

The residuals for this latter model (in form "model" minus "data") are coded in R in the following code chunk in the function `shobbs.res`. We have also coded the Jacobian for this model as `shobbs.jac`

```
> shobbs.res <- function(x){ # scaled Hobbs weeds problem -- residual
+ # This variant uses looping
+     if(length(x) != 3) stop("hobbs.res -- parameter vector n!=3")
+     y <- c(5.308, 7.24, 9.638, 12.866, 17.069, 23.192, 31.443, 38.558, 50.156, 62.948,
+          75.995, 91.972)
+     tt <- 1:12
+     res <- 100.0*x[1]/(1+x[2]*10.*exp(-0.1*x[3]*tt)) - y
+ }
> shobbs.jac <- function(x) { # scaled Hobbs weeds problem -- Jacobian
+     jj <- matrix(0.0, 12, 3)
+     tt <- 1:12
+     yy <- exp(-0.1*x[3]*tt) # We don't need data for the Jacobian
+     zz <- 100.0/(1+10.*x[2]*yy)
+     jj[tt,1]  <-  zz
+     jj[tt,2]  <-  -0.1*x[1]*zz*zz*yy
+     jj[tt,3]  <-  0.01*x[1]*zz*zz*yy*x[2]*tt
```

```
  +     return(jj)
  + }
```

With package `nlmrt`, function `nlfb` can be used to estimate the parameters of the WEEDS problem as follows, where we use the naive starting point where all parameters are 1.

```
  > st <- c(b1=1, b2=1, b3=1)
  > ans1 <- nlfb(st, shobbs.res, shobbs.jac, trace=FALSE)
  > print(ans1)
nlmrt class object: x
residual sumsquares =  2.5873  on  12 observations
    after  10    Jacobian and  14 function evaluations
  name            coeff          SE       tstat      pval      gradient    JSingval
b1              1.96186        0.1131      17.35  3.166e-08  -7.327e-06       130.1
b2              4.90916        0.1688      29.08  3.282e-10   1.433e-07       6.165
b3               3.1357       0.06863      45.69  5.768e-12   1.717e-06       2.735
```

This works very well, with almost identical iterates as given by `nlxb`. (Since the algorithms are the same, this should be the case.) Note that we turn off the `trace` output. There is also the possibility of interrupting the iterations to `watch` the progress. Changing the value of `watch` in the call to `nlfb` below allows this. In this code chunk, we use an internal numerical approximation to the Jacobian.

```
  > cat("No jacobian function -- use internal approximation\n")
No jacobian function -- use internal approximation
  > ans1n <- nlfb(st, shobbs.res, trace=FALSE, control=list(watch=FALSE)) # NO jacfn
  > print(ans1n)
nlmrt class object: x
residual sumsquares =  2.5873  on  12 observations
    after  10    Jacobian and  14 function evaluations
  name            coeff          SE       tstat      pval      gradient    JSingval
b1              1.96186        0.1131      17.35  3.166e-08  -7.326e-06       130.1
b2              4.90916        0.1688      29.08  3.282e-10   1.428e-07       6.165
b3               3.1357       0.06863      45.69  5.768e-12   1.719e-06       2.735
```

Note that we could also form the sum of squares function and the gradient and use a function minimization code. The next code block shows how this is done, creating the sum of squares function and its gradient, then using the `optimx` package to call a number of minimizers simultaneously.

```
  > shobbs.f <- function(x){
  +    res <- shobbs.res(x)
  +    as.numeric(crossprod(res))
  + }
  > shobbs.g <- function(x){
  +    res <- shobbs.res(x) # This is NOT efficient -- we generally have res already calcula
  +    JJ <- shobbs.jac(x)
  +    2.0*as.vector(crossprod(JJ,res))
  + }
  > require(optimx)
  > aopx <- optimx(st, shobbs.f, shobbs.g, control=list(all.methods=TRUE))
```

```
step        nofc        fmin        xpar
  > summary(aopx)
                b1     b2     b3      value fevals gevals
BFGS        1.9619 4.9092 3.1357     2.5873    119     36
CG          1.9120 4.8246 3.1586     2.6679    427    101
Nelder-Mead 1.9645 4.9116 3.1340     2.5877    196     NA
L-BFGS-B    1.9619 4.9092 3.1357     2.5873     41     41
nlm         1.9619 4.9092 3.1357     2.5873     NA     NA
nlminb      1.9619 4.9092 3.1357     2.5873     31     29
spg         1.9618 4.9091 3.1357     2.5873    188     NA
ucminf      1.9619 4.9092 3.1357     2.5873     45     45
Rcgmin      1.9619 4.9092 3.1357     2.5873   1007    175
Rvmmin      1.9619 4.9092 3.1357     2.5873    148     57
newuoa      1.9619 4.9092 3.1357     2.5873    357     NA
bobyqa      1.9619 4.9092 3.1357     2.5873    626     NA
nmkb        1.9619 4.9092 3.1357     2.5873    195     NA
hjkb        1.0000 1.0000 1.0000 10685.2878      1     NA
            niter convcode  kkt1 kkt2 xtimes
BFGS           NA        0  TRUE TRUE  0.004
CG             NA        1 FALSE TRUE  0.004
Nelder-Mead    NA        0 FALSE TRUE  0.004
L-BFGS-B       NA        0  TRUE TRUE  0.004
nlm            50        0  TRUE TRUE  0.000
nlminb         28        0  TRUE TRUE  0.000
spg           150        0  TRUE TRUE  0.020
ucminf         NA        0  TRUE TRUE  0.004
Rcgmin         NA        1  TRUE TRUE  0.036
Rvmmin         NA        0  TRUE TRUE  0.016
newuoa         NA        0  TRUE TRUE  0.008
bobyqa         NA        0  TRUE TRUE  0.044
nmkb           NA        0 FALSE TRUE  0.020
hjkb            0     9999    NA   NA  0.004
  > cat("\nNow with numerical gradient approximation or derivative free methods\n")
Now with numerical gradient approximation or derivative free methods
  > aopxn <- optimx(st, shobbs.f, control=list(all.methods=TRUE))
step        nofc        fmin        xpar
  > summary(aopxn) # no file output
                b1     b2     b3       value fevals gevals
BFGS        1.9619 4.9092 3.1357  2.5873e+00    118     36
CG          1.7996 4.5967 3.2079  3.8329e+00    413    101
Nelder-Mead 1.9645 4.9116 3.1340  2.5877e+00    196     NA
L-BFGS-B    1.9619 4.9092 3.1357  2.5873e+00     45     45
nlm         1.9619 4.9092 3.1357  2.5873e+00     NA     NA
nlminb      1.9619 4.9092 3.1357  2.5873e+00     32     93
spg         1.9619 4.9091 3.1357  2.5873e+00    184     NA
ucminf      1.9619 4.9090 3.1356  2.5873e+00     45     45
Rcgmin          NA     NA     NA 8.9885e+307     NA     NA
Rvmmin          NA     NA     NA 8.9885e+307     NA     NA
newuoa      1.9619 4.9092 3.1357  2.5873e+00    357     NA
bobyqa      1.9619 4.9092 3.1357  2.5873e+00    626     NA
nmkb        1.9619 4.9092 3.1357  2.5873e+00    195     NA
hjkb        1.0000 1.0000 1.0000  1.0685e+04      1     NA
            niter convcode  kkt1 kkt2 xtimes
BFGS           NA        0  TRUE TRUE  0.004
CG             NA        1 FALSE TRUE  0.012
Nelder-Mead    NA        0 FALSE TRUE  0.004
L-BFGS-B       NA        0  TRUE TRUE  0.000
nlm            50        0  TRUE TRUE  0.004
nlminb         27        0  TRUE TRUE  0.004
spg           153        0  TRUE TRUE  0.024
ucminf         NA        0 FALSE TRUE  0.104
Rcgmin         NA     9999    NA   NA  0.188
Rvmmin         NA     9999    NA   NA  0.192
newuoa         NA        0  TRUE TRUE  0.032
bobyqa         NA        0  TRUE TRUE  0.076
```

```
nmkb           NA        0 FALSE TRUE  0.224
hjkb            0     9999    NA   NA  0.044
```

We see that most of the minimizers work with either the analytic or approximated gradient. The 'CG' option of function `optim()` does not do very well in either case. As the author of the original step and description and then Turbo Pascal code, I can say I was never very happy with this method and replaced it recently with `Rcgmin` from the package of the same name, in the process adding the possibility of bounds or masks constraints.

4 Converting an expression to a function

Clearly if we have an expression, it would be nice to be able to automatically convert this to a function, if possible also getting the derivatives. Indeed, it is possible to convert an expression to a function, and there are several ways to do this (references??). In package `nlmrt` we provide the tools `model2grfun.R`, `model2jacfun.R`, `model2resfun.R`, and `model2ssfun.R` to convert a model expression to a function to compute the gradient, Jacobian, residuals or sum of squares functions respectively. We do not provide any tool for converting a function for the residuals back to an expression, as functions can use structures that are not easily expressed as R expressions.

Below are code chunks to illustrate the generation of the residual, sum of squares, Jacobian and gradient code for the Ratkowsky problem used earlier in the vignette. The commented-out first line shows how we would use one of these function generators to output the function to a file named "testresfn.R". However, it is not necessary to generate the file.

First, let us generate the residuals. We must supply the names of the parameters, and do this via the starting vector of parameters `ones`. The actual values are not needed by `model2resfun`, just the names. Other names are drawn from the variables used in the model expression `regmod`.

```
> # jres <- model2resfun(regmod, ones, funname="myxres", file="testresfn.R")
> jres <- model2resfun(regmod, ones)
> print(jres)
```

```
function (prm, yield = NULL, time = NULL)
{
    t1 <- prm[[1]]
    t2 <- prm[[2]]
    t3 <- prm[[3]]
    t4 <- prm[[4]]
    resids <- as.numeric(eval(t1 - t2 * exp(-exp(t3 + t4 * log(time))) -
        yield))
}
<environment: 0x611e688>
```

```
> valjres <- jres(ones, yield=pastured$yield, time=pastured$time)
> cat("valjres:")
```

```
valjres:
```

```
> print(valjres)
```

```
[1]  -7.93  -9.80 -17.59 -21.33 -38.35 -55.11 -60.73 -63.62
[9] -66.08
```

Now let us also generate the Jacobian and test it using the numerical approximations from package `numDeriv`.

```
  > jjac <- model2jacfun(regmod, ones)
  > print(jjac)
function (prm, yield = NULL, time = NULL)
{
    t1 <- prm[[1]]
    t2 <- prm[[2]]
    t3 <- prm[[3]]
    t4 <- prm[[4]]
    localdf <- data.frame(yield, time)
    jstruc <- with(localdf, eval({
        .expr1 <- log(time)
        .expr4 <- exp(t3 + t4 * .expr1)
        .expr6 <- exp(-.expr4)
        .value <- t1 - t2 * .expr6 - yield
        .grad <- array(0, c(length(.value), 4), list(NULL, c("t1",
            "t2", "t3", "t4")))
        .grad[, "t1"] <- 1
        .grad[, "t2"] <- -.expr6
        .grad[, "t3"] <- t2 * (.expr6 * .expr4)
        .grad[, "t4"] <- t2 * (.expr6 * (.expr4 * .expr1))
        attr(.value, "gradient") <- .grad
        .value
    }))
    jacmat <- attr(jstruc, "gradient")
    return(jacmat)
}
<environment: 0x61ed4d8>
  > # Note that we now need some data!
  > valjjac <- jjac(ones, yield=pastured$yield, time=pastured$time)
  > cat("valjac:")
valjac:
  > print(valjjac)
      t1          t2         t3         t4
 [1,]  1 -2.3724e-11 5.8040e-10 1.2753e-09
 [2,]  1 -2.9683e-17 1.1296e-15 2.9812e-15
 [3,]  1 -1.6172e-25 9.2317e-24 2.8106e-23
 [4,]  1 -8.8110e-34 6.7062e-32 2.2347e-31
 [5,]  1 -2.6154e-50 2.9859e-48 1.1160e-47
 [6,]  1 -5.1229e-68 7.9375e-66 3.2092e-65
 [7,]  1 -4.2297e-75 7.2434e-73 3.0010e-72
 [8,]  1 -2.3044e-83 4.3849e-81 1.8629e-80
 [9,]  1 -5.4670e-94 1.1740e-91 5.1298e-91
  > # Now compute the numerical approximation
  > require(numDeriv)
  > Jn <- jacobian(jres, ones, , yield=pastured$yield, time=pastured$time)
  > cat("maxabsdiff=",max(abs(Jn-valjjac)),"\n")
maxabsdiff= 3.7744e-10
```

As with the WEEDS problem, we can compute the sum of squares function and the gradient.

```
  > ssfn <- model2ssfun(regmod, ones) # problem getting the data attached!
  > print(ssfn)
function (prm, yield = NULL, time = NULL)
{
```

```
    t1 <- prm[[1]]
    t2 <- prm[[2]]
    t3 <- prm[[3]]
    t4 <- prm[[4]]
    resids <- as.numeric(eval(t1 - t2 * exp(-exp(t3 + t4 * log(time))) -
        yield))
    ss <- as.numeric(crossprod(resids))
}
<environment: 0x58a89b0>
```

```
> valss <- ssfn(ones, yield=pastured$yield, time=pastured$time)
> cat("valss: ",valss,"\n")
```

```
valss:  17533
```

```
> grfn <- model2grfun(regmod, ones) # problem getting the data attached!
> print(grfn)
```

```
function (prm, yield = NULL, time = NULL)
{
    t1 <- prm[[1]]
    t2 <- prm[[2]]
    t3 <- prm[[3]]
    t4 <- prm[[4]]
    localdf <- data.frame(yield, time)
    jstruc <- with(localdf, eval({
        .expr1 <- log(time)
        .expr4 <- exp(t3 + t4 * .expr1)
        .expr6 <- exp(-.expr4)
        .value <- t1 - t2 * .expr6 - yield
        .grad <- array(0, c(length(.value), 4), list(NULL, c("t1",
            "t2", "t3", "t4")))
        .grad[, "t1"] <- 1
        .grad[, "t2"] <- -.expr6
        .grad[, "t3"] <- t2 * (.expr6 * .expr4)
        .grad[, "t4"] <- t2 * (.expr6 * (.expr4 * .expr1))
        attr(.value, "gradient") <- .grad
        .value
    }))
    jacmat <- attr(jstruc, "gradient")
    resids <- as.numeric(eval(t1 - t2 * exp(-exp(t3 + t4 * log(time))) -
        yield))
    grj <- as.vector(2 * crossprod(jacmat, resids))
}
<environment: 0x5e84578>
```

```
> valgr <- grfn(ones, yield=pastured$yield, time=pastured$time)
> cat("valgr:")
```

```
valgr:
```

```
> print(valgr)
```

```
[1] -6.8108e+02  3.7626e-10 -9.2051e-09 -2.0226e-08
```

```
> gn <- grad(ssfn, ones, yield=pastured$yield, time=pastured$time)
> cat("maxabsdiff=",max(abs(gn-valgr)),"\n")
```

```
maxabsdiff= 1.4376e-07
```

Moreover, we can use the Huet starting parameters as a double check on our conversion of the expression to various optimization-style functions.

```
> cat("\n\nHuetstart:")
```

```
Huetstart:
```

```
> print(huetstart)
```

```
t1 t2 t3 t4
70 60  0  1
```

```
  > valjres <- jres(huetstart, yield=pastured$yield, time=pastured$time)
  > cat("valjres:")
valjres:
  > print(valjres)
[1] 61.063 59.200 51.410 47.670 30.650 13.890  8.270  5.380
[9]  2.920
  > valss <- ssfn(huetstart, yield=pastured$yield, time=pastured$time)
  > cat("valss:", valss, "\n")
valss: 13387
  > valjjac <- jjac(huetstart, yield=pastured$yield, time=pastured$time)
  > cat("valjac:")
valjac:
  > print(valjjac)
     t1         t2         t3         t4
 [1,]  1 -1.2341e-04 6.6641e-02 1.4643e-01
 [2,]  1 -8.3153e-07 6.9848e-04 1.8433e-03
 [3,]  1 -7.5826e-10 9.5540e-07 2.9087e-06
 [4,]  1 -6.9144e-13 1.1616e-09 3.8708e-09
 [5,]  1 -5.7495e-19 1.4489e-15 5.4154e-15
 [6,]  1 -1.7588e-25 6.0151e-22 2.4319e-21
 [7,]  1 -4.3596e-28 1.6479e-24 6.8276e-24
 [8,]  1 -3.9754e-31 1.6697e-27 7.0937e-27
 [9,]  1 -4.9061e-35 2.3255e-31 1.0161e-30
  > Jn <- jacobian(jres, huetstart, , yield=pastured$yield, time=pastured$time)
  > cat("maxabsdiff=",max(abs(Jn-valjjac)),"\n")
maxabsdiff= 5.3945e-10
  > valgr <- grfn(huetstart, yield=pastured$yield, time=pastured$time)
  > cat("valgr:")
valgr:
  > print(valgr)
[1] 560.90509  -0.01517   8.22138  18.10084
  > gn <- grad(ssfn, huetstart, yield=pastured$yield, time=pastured$time)
  > cat("maxabsdiff=",max(abs(gn-valgr)),"\n")
maxabsdiff= 4.7636e-08
```

Now that we have these functions, let us apply them with `nlfb`.

```
  > cat("All ones to start\n")
All ones to start
  > anlfb <- nlfb(ones, jres, jjac, trace=FALSE, yield=pastured$yield, time=pastured$time)
  > print(strwrap(anlfb))
 [1] "c(29.9077777777472, 28.0377777777778,"
 [2] "20.2477777777778, 16.5077777777778,"
 [3] "-0.512222222222185, -17.2722222222222,"
 [4] "-22.8922222222222, -25.7822222222222,"
 [5] "-28.2422222222222)"
 [6] "c(1, 1, 1, 1, 1, 1, 1, 1, 1, -2.5904803198541e-11,"
 [7] "-3.48177832682637e-17, -2.11977026263411e-25,"
 [8] "-1.30186504324169e-33, -5.00031799754126e-50,"
 [9] "-1.31593194786314e-67, -1.22799624106577e-74,"
[10] "-7.73807145709702e-83, -2.22164181475742e-93,"
[11] "6.31486798421711e-10, 1.31950211229055e-15,"
[12] "1.20434229540269e-23, 9.85816788938708e-32,"
[13] "5.67649212680981e-48, 2.02656873886703e-65,"
```

```
[14] "2.0899294411938e-72, 1.4630618491745e-80,"
[15] "4.73981859865953e-91, 1.38751831375555e-09,"
[16] "3.48224172088318e-15, 3.66664714105283e-23,"
[17] "3.284943150308e-31, 2.12168521608018e-47,"
[18] "8.1935213090302e-65, 8.65885924351809e-72,"
[19] "6.21581130504181e-80, 2.07103901969961e-90)"
[20] "4"
[21] "3"
[22] "c(38.8377777777778, 1.00007369903129,"
[23] "0.998201661261902, 0.996048644398237)"
[24] "4648.06335555373"
[25] "c(-Inf, -Inf, -Inf, -Inf)"
[26] "c(Inf, Inf, Inf, Inf)"
[27] "NULL"
```

```
  > cat("Huet start\n")
Huet start
  > anlfbh <- nlfb(huetstart, jres, jjac, trace=FALSE, yield=pastured$yield, time=pastured$t
  > print(strwrap(anlfbh))
 [1] "c(0.480575683702348, 0.669264006079271,"
 [2] "-2.28426563497321, 0.843862687207519,"
 [3] "0.734652618487608, 0.066510649295175,"
 [4] "-0.985862291968068, -0.0250879549066383,"
 [5] "0.500350456693454)"
 [6] "c(1, 1, 1, 1, 1, 1, 1, 1, 1, -0.981556726091092,"
 [7] "-0.948171282599528, -0.869750270888723,"
 [8] "-0.758399834057038, -0.484261107837453,"
 [9] "-0.223408951427342, -0.14936303047615,"
[10] "-0.0869332933121718, -0.0385258954067749,"
[11] "1.12712321032759, 3.11275223693951, 7.48692917929391,"
[12] "12.9373484175606, 21.6609765596452, 20.6543768151932,"
[13] "17.5183401160423, 13.0985419560085, 7.73883739451366,"
[14] "2.47654281941788, 8.2147316061715, 22.7941238760065,"
[15] "43.1098907467036, 80.9615739893341, 83.5067043689992,"
[16] "72.5808432835129, 55.6490931778859, 33.8144464340492)"
[17] "44"
[18] "32"
[19] "c(69.9553722026373, 61.6818319271118,"
[20] "-9.2088020481334, 2.37778402563408)"
[21] "8.37588360361957"
[22] "c(-Inf, -Inf, -Inf, -Inf)"
[23] "c(Inf, Inf, Inf, Inf)"
[24] "NULL"
```

5 Using bounds and masks

The manual for `nls()` tells us that bounds are restricted to the 'port' algorithm.

```
lower, upper: vectors of lower and upper bounds, replicated to be as
          long as 'start'.  If unspecified, all parameters are assumed
          to be unconstrained.  Bounds can only be used with the
          '"port"' algorithm.  They are ignored, with a warning, if
          given for other algorithms.
```

Later in the manual, there is the discomforting warning:

```
The 'algorithm = "port"' code appears unfinished, and does not
even check that the starting value is within the bounds.  Use with
caution, especially where bounds are supplied.
```

We will base the rest of this discussion on the examples in man/nlmrt-package.Rd, and use an unscaled version of the WEEDS problem.

First, let us estimate the model with no constraints.

```
> require(nlmrt)
> # Data for Hobbs problem
> ydat <- c(5.308, 7.24, 9.638, 12.866, 17.069, 23.192, 31.443,
+           38.558, 50.156, 62.948, 75.995, 91.972)
> tdat <- 1:length(ydat)
> weeddata1 <- data.frame(y=ydat, tt=tdat)
> start1 <- c(b1=1, b2=1, b3=1) # name parameters for nlxb, nls, wrapnls.
> eunsc <-  y ~ b1/(1+b2*exp(-b3*tt))
> anlxb1 <- try(nlxb(eunsc, start=start1, data=weeddata1))
> print(anlxb1)
nlmrt class object: x
residual sumsquares =  2.5873  on  12 observations
    after  18    Jacobian and  25 function evaluations
  name            coeff          SE       tstat      pval      gradient    JSingval
b1               196.186        11.31      17.35  3.164e-08  -2.663e-07        1011
b2               49.0916        1.688      29.08  3.281e-10    1.59e-07      0.4605
b3               0.31357     0.006863      45.69  5.768e-12  -5.531e-05     0.04715
```

Now let us see if we can apply bounds. Note that we name the parameters in the vectors for the bounds. First we apply bounds that are NOT active at the unconstrained solution.

```
> # WITH BOUNDS
> startf1 <- c(b1=1, b2=1, b3=.1) # a feasible start when b3 <= 0.25
> anlxb1 <- try(nlxb(eunsc, start=startf1, lower=c(b1=0, b2=0, b3=0),
+       upper=c(b1=500, b2=100, b3=5), data=weeddata1))
> print(anlxb1)
nlmrt class object: x
residual sumsquares =  2.5873  on  12 observations
    after  13    Jacobian and  18 function evaluations
  name            coeff          SE       tstat      pval      gradient    JSingval
b1               196.186        11.31      17.35  3.164e-08  -2.662e-07        1011
b2               49.0916        1.688      29.08  3.281e-10   1.648e-07      0.4605
b3               0.31357     0.006863      45.69  5.768e-12  -5.872e-05     0.04715
```

We note that `nls()` also solves this case.

```
> anlsb1 <- try(nls(eunsc, start=startf1, lower=c(b1=0, b2=0, b3=0),
+      upper=c(b1=500, b2=100, b3=5), data=weeddata1, algorithm='port'))
> print(anlsb1)
Nonlinear regression model
  model: y ~ b1/(1 + b2 * exp(-b3 * tt))
   data: weeddata1
     b1      b2      b3
196.186  49.092   0.314
 residual sum-of-squares: 2.59

Algorithm "port", convergence message: relative convergence (4)
```

Now we will change the bounds so the start is infeasible.

```
> ## Uncon solution has bounds ACTIVE. Infeasible start
> anlxb2i <- try(nlxb(eunsc, start=start1, lower=c(b1=0, b2=0, b3=0),
+            upper=c(b1=500, b2=100, b3=.25), data=weeddata1))
> print(anlxb2i)
[1] "Error in nlxb(eunsc, start = start1, lower = c(b1 = 0, b2 = 0, b3 = 0),  : \n  Infeasible start\n"
attr(,"class")
[1] "try-error"
attr(,"condition")
<simpleError in nlxb(eunsc, start = start1, lower = c(b1 = 0, b2 = 0, b3 = 0),     upper = c(b1 = 500, b2 = 100, b3 = 0.25), data
```

```
> anlsb2i <- try(nls(eunsc, start=start1, lower=c(b1=0, b2=0, b3=0),
+            upper=c(b1=500, b2=100, b3=.25), data=weeddata1, algorithm='port'))
> print(anlsb2i)
[1] "Error in nls(eunsc, start = start1, lower = c(b1 = 0, b2 = 0, b3 = 0),  : \n  Convergence failure: initial par violates cons
attr(,"class")
[1] "try-error"
attr(,"condition")
<simpleError in nls(eunsc, start = start1, lower = c(b1 = 0, b2 = 0, b3 = 0),     upper = c(b1 = 500, b2 = 100, b3 = 0.25), data =
```

Both `nlxb()` and `nls()` (with 'port') do the right thing and refuse to proceed. There is a minor "glitch" in the output processing of both `knitR` and `Sweave` here. Let us start them off properly and see what they accomplish.

```
> ## Uncon solution has bounds ACTIVE. Feasible start
> anlxb2f <- try(nlxb(eunsc, start=startf1, lower=c(b1=0, b2=0, b3=0),
+    upper=c(b1=500, b2=100, b3=.25), data=weeddata1))
> print(anlxb2f)
nlmrt class object: x
residual sumsquares =  29.993  on  12 observations
    after  13    Jacobian and  18 function evaluations
  name            coeff          SE       tstat      pval      gradient    JSingval
b1                  500U           NA         NA         NA             0       1.529
b2               87.9425           NA         NA         NA  -1.808e-10           0
b3                 0.25U           NA         NA         NA             0           0
```

```
> anlsb2f <- try(nls(eunsc, start=startf1, lower=c(b1=0, b2=0, b3=0),
+    upper=c(b1=500, b2=100, b3=.25), data=weeddata1, algorithm='port'))
> print(anlsb2f)
Nonlinear regression model
  model: y ~ b1/(1 + b2 * exp(-b3 * tt))
   data: weeddata1
    b1     b2     b3 
500.00  87.94   0.25 
 residual sum-of-squares: 30

Algorithm "port", convergence message: both X-convergence and relative convergence (5)
```

Both methods get essentially the same answer for the bounded problem, and this solution has parameters `b1` and `b3` at their upper bounds. The Jacobian elements for these parameters are zero as returned by `nlxb()`.

Let us now turn to **masks**, which functions from `nlmrt` are designed to handle. Masks are also available with packages `Rcgmin` and `Rvmmin`. I would like to hear if other packages offer this capability.

```
> ## TEST MASKS
> anlsmnqm <- try(nlxb(eunsc, start=start1, lower=c(b1=0, b2=0, b3=0),
```

```
  +    upper=c(b1=500, b2=100, b3=5), masked=c("b2"), data=weeddata1))
  > print(anlsmnqm) # b2 masked
nlmrt class object: x 
residual sumsquares =  6181.2  on  12 observations
    after  22    Jacobian and  35 function evaluations
  name            coeff          SE      tstat      pval      gradient    JSingval   
b1              50.4013            NA         NA         NA   -0.001511       162.1  
b2                    1  M         NA         NA         NA           0      0.4918  
b3              0.19862            NA         NA         NA     -0.0468           0  
```

```
  > an1qm3 <- try(nlxb(eunsc, start=start1, data=weeddata1, masked=c("b3")))
  > print(an1qm3) # b3 masked
nlmrt class object: x 
residual sumsquares =  1031  on  12 observations
    after  17    Jacobian and  18 function evaluations
  name            coeff          SE      tstat      pval      gradient    JSingval   
b1              78.5698            NA         NA         NA   8.489e-08       1.944  
b2              2293.71            NA         NA         NA  -1.757e-09     0.01097  
b3                    1  M         NA         NA         NA           0           0  
```

```
  > # Note that the parameters are put in out of order to test code.
  > an1qm123 <- try(nlxb(eunsc, start=start1, data=weeddata1, masked=c("b2","b1","b3")))
  > print(an1qm123) # ALL masked - fails!!
[1] "Error in nlxb(eunsc, start = start1, data = weeddata1, masked = c(\"b2\",  : \n  All parameters are masked\n"
attr(,"class")
[1] "try-error"
attr(,"condition")
<simpleError in nlxb(eunsc, start = start1, data = weeddata1, masked = c("b2",     "b1", "b3")): All parameters are masked>
```

Finally (for `nlxb`) we combine the bounds and mask.

```
  > ## BOUNDS and MASK
  > an1qbm2 <- try(nlxb(eunsc, start=startf1, data=weeddata1,
  +     lower=c(0,0,0), upper=c(200, 60, .3), masked=c("b2")))
  > print(an1qbm2)
nlmrt class object: x 
residual sumsquares =  6181.2  on  12 observations
    after  17    Jacobian and  28 function evaluations
  name            coeff          SE      tstat      pval      gradient    JSingval   
b1              50.4016            NA         NA         NA   0.0004618       162.2  
b2                    1  M         NA         NA         NA           0      0.4918  
b3             0.198618            NA         NA         NA     -0.0746           0  
```

```
  > an1qbm2x <- try(nlxb(eunsc, start=startf1, data=weeddata1,
  +     lower=c(0,0,0), upper=c(48, 60, .3), masked=c("b2")))
  > print(an1qbm2x)
nlmrt class object: x 
residual sumsquares =  6206.1  on  12 observations
    after  11    Jacobian and  20 function evaluations
  name            coeff          SE      tstat      pval      gradient    JSingval   
b1                   48U           NA         NA         NA           0       141.2  
b2                    1  M         NA         NA         NA           0           0  
b3             0.215971            NA         NA         NA     -0.1502           0  
```

Turning to the function-based `nlfb`,

```
  > hobbs.res <- function(x){ # Hobbs weeds problem -- residual
  +     if(length(x) != 3) stop("hobbs.res -- parameter vector n!=3")
  +     y <- c(5.308, 7.24, 9.638, 12.866, 17.069, 23.192, 31.443, 38.558, 50.156, 62.948,
```

```
  +         75.995, 91.972)
  +     tt <- 1:12
  +     res <- x[1]/(1+x[2]*exp(-x[3]*tt)) - y
  + }
  > hobbs.jac <- function(x) { # Hobbs weeds problem -- Jacobian
  +     jj <- matrix(0.0, 12, 3)
  +     tt <- 1:12
  +     yy <- exp(-x[3]*tt)
  +     zz <- 1.0/(1+x[2]*yy)
  +     jj[tt,1]  <-  zz
  +     jj[tt,2]  <-  -x[1]*zz*zz*yy
  +     jj[tt,3]  <-  x[1]*zz*zz*yy*x[2]*tt
  +     return(jj)
  + }
  > # Check unconstrained
  > ans1 <- nlfb(start1, hobbs.res, hobbs.jac)
  > ans1
nlmrt class object: x
residual sumsquares =  2.5873  on  12 observations
    after  18    Jacobian and  25 function evaluations
  name            coeff          SE       tstat      pval      gradient    JSingval
b1               196.186         11.31      17.35  3.164e-08  -2.663e-07        1011
b2               49.0916         1.688      29.08  3.281e-10    1.59e-07      0.4605
b3               0.31357      0.006863      45.69  5.768e-12  -5.531e-05     0.04715
  > ## No jacobian - use internal approximation
  > ans1n <- nlfb(start1, hobbs.res)
  > ans1n
nlmrt class object: x
residual sumsquares =  2.5873  on  12 observations
    after  18    Jacobian and  25 function evaluations
  name            coeff          SE       tstat      pval      gradient    JSingval
b1               196.186         11.31      17.35  3.164e-08  -2.662e-07        1011
b2               49.0916         1.688      29.08  3.281e-10   1.589e-07      0.4605
b3               0.31357      0.006863      45.69  5.768e-12  -5.526e-05     0.04715
  > # Bounds -- infeasible start
  > ans2i <- try(nlfb(start1, hobbs.res, hobbs.jac,
  +    lower=c(b1=0, b2=0, b3=0), upper=c(b1=500, b2=100, b3=.25)))
  > ans2i
[1] "Error in nlfb(start1, hobbs.res, hobbs.jac, lower = c(b1 = 0, b2 = 0,  : \n  Infeasible start\n"
attr(,"class")
[1] "try-error"
attr(,"condition")
<simpleError in nlfb(start1, hobbs.res, hobbs.jac, lower = c(b1 = 0, b2 = 0,     b3 = 0), upper = c(b1 = 500, b2 = 100, b3 = 0.25
  > # Bounds -- feasible start
  > ans2f <- nlfb(startf1, hobbs.res, hobbs.jac,
  +    lower=c(b1=0, b2=0, b3=0), upper=c(b1=500, b2=100, b3=.25))
  > ans2f
nlmrt class object: x
residual sumsquares =  29.993  on  12 observations
    after  13    Jacobian and  18 function evaluations
  name            coeff          SE       tstat      pval      gradient    JSingval
b1                   500U           NA         NA         NA           0       1.529
b2               87.9425            NA         NA         NA  -1.809e-10           0
b3                  0.25U           NA         NA         NA           0           0
```

```
> # Mask b2
> ansm2 <- nlfb(start1, hobbs.res, hobbs.jac, maskidx=c(2))
> ansm2
nlmrt class object: x
residual sumsquares =  6181.2  on  12 observations
    after  24    Jacobian and  38 function evaluations
  name            coeff          SE       tstat      pval      gradient    JSingval
b1               50.4022            NA         NA         NA    0.001528       162.2
b2                     1  M         NA         NA         NA           0      0.4918
b3              0.198611            NA         NA         NA     0.04544           0
> # Mask b3
> ansm3 <- nlfb(start1, hobbs.res, hobbs.jac, maskidx=c(3))
> ansm3
nlmrt class object: x
residual sumsquares =  1031  on  12 observations
    after  17    Jacobian and  18 function evaluations
  name            coeff          SE       tstat      pval      gradient    JSingval
b1               78.5698            NA         NA         NA   8.489e-08       1.944
b2               2293.71            NA         NA         NA  -1.757e-09     0.01097
b3                     1  M         NA         NA         NA           0           0
> # Mask all -- should fail
> ansma <- try(nlfb(start1, hobbs.res, hobbs.jac, maskidx=c(3,1,2)))
> ansma
[1] "Error in nlfb(start1, hobbs.res, hobbs.jac, maskidx = c(3, 1, 2)) : \n  All parameters are masked\n"
attr(,"class")
[1] "try-error"
attr(,"condition")
<simpleError in nlfb(start1, hobbs.res, hobbs.jac, maskidx = c(3, 1, 2)): All parameters are masked>
> # Bounds and mask
> ansmbm2 <- nlfb(startf1, hobbs.res, hobbs.jac, maskidx=c(2),
+       lower=c(0,0,0), upper=c(200, 60, .3))
> ansmbm2
nlmrt class object: x
residual sumsquares =  6181.2  on  12 observations
    after  17    Jacobian and  28 function evaluations
  name            coeff          SE       tstat      pval      gradient    JSingval
b1               50.4016            NA         NA         NA   0.0004618       162.2
b2                     1  M         NA         NA         NA           0      0.4918
b3              0.198618            NA         NA         NA     -0.0746           0
> # Active bound
> ansmbm2x <- nlfb(startf1, hobbs.res, hobbs.jac, maskidx=c(2),
+       lower=c(0,0,0), upper=c(48, 60, .3))
> ansmbm2x
nlmrt class object: x
residual sumsquares =  6206.1  on  12 observations
    after  11    Jacobian and  20 function evaluations
  name            coeff          SE       tstat      pval      gradient    JSingval
b1                    48U           NA         NA         NA           0       141.2
b2                     1  M         NA         NA         NA           0           0
b3              0.215971            NA         NA         NA     -0.1502           0
```

The results match those of `nlxb()`

Finally, let us check the results above with `Rvmmin` and `Rcgmin`. Note that this vignette cannot be created on systems that lack these codes.

```
> require(Rcgmin)
> require(Rvmmin)
> hobbs.f <- function(x) {
+    res<-hobbs.res(x)
+    as.numeric(crossprod(res))
+ }
> hobbs.g <- function(x) {
+    res <- hobbs.res(x) # Probably already available
+    JJ <- hobbs.jac(x)
+    2.0*as.numeric(crossprod(JJ, res))
+ }
> # Check unconstrained
> a1cg <- Rcgmin(start1, hobbs.f, hobbs.g)
> a1cg
$par
       b1        b2        b3
196.18475  49.09147   0.31357

$value
[1] 2.5873

$counts
[1] 1011  197

$convergence
[1] 1

$message
[1] "Too many function evaluations (> 1000) "

> a1vm <- Rvmmin(start1, hobbs.f, hobbs.g)
> a1vm
$par
       b1        b2        b3
196.18626  49.09164   0.31357

$value
[1] 2.5873

$counts
function gradient
     215       55

$convergence
[1] 3

$message
[1] "Rvmminu appears to have converged"

> ## No jacobian - use internal approximation
> a1cgn <- try(Rcgmin(start1, hobbs.f))
> a1cgn
$par
       b1        b2        b3
196.19292  49.09238   0.31357

$value
[1] 2.5873

$counts
```

```
[1] 1009  238

$convergence
[1] 1

$message
[1] "Too many function evaluations (> 1000) "
  > a1vmn <- try(Rvmmin(start1, hobbs.f))
  > a1vmn
$par
       b1        b2        b3 
196.18634  49.09163   0.31357 

$value
[1] 2.5873

$counts
function gradient 
     206       50 

$convergence
[1] 3

$message
[1] "Rvmminu appears to have converged"
  > # But
  > grfwd <- function(par, userfn, fbase=NULL, eps=1.0e-7, ...) {
  +    # Forward different gradient approximation
  +    if (is.null(fbase)) fbase <- userfn(par, ...)  # ensure we function value at par
  +    df <- rep(NA, length(par))
  +    teps <- eps * (abs(par) + eps)
  +    for (i in 1:length(par)) {
  +       dx <- par
  +       dx[i] <- dx[i] + teps[i]
  +       df[i] <- (userfn(dx, ...) - fbase)/teps[i]
  +    }
  +    df
  + }
  > a1vmn <- try(Rvmmin(start1, hobbs.f, gr="grfwd"))
  > a1vmn
$par
       b1        b2        b3 
196.18634  49.09163   0.31357 

$value
[1] 2.5873

$counts
function gradient 
     206       50 

$convergence
[1] 3

$message
[1] "Rvmminu appears to have converged"
  > # Bounds -- infeasible start
  > # Note: These codes move start to nearest bound
```

```
> a1cg2i <- Rcgmin(start1, hobbs.f, hobbs.g,
+      lower=c(b1=0, b2=0, b3=0), upper=c(b1=500, b2=100, b3=.25))
> a1cg2i
$par
     b1      b2      b3 
500.000  87.942   0.250 

$value
[1] 29.993

$counts
[1] 87 45

$convergence
[1] 0

$message
[1] "Rcgmin seems to have converged"

$bdmsk
[1] -1  1 -1
> a1vm2i <- Rvmmin(start1, hobbs.f, hobbs.g,
+      lower=c(b1=0, b2=0, b3=0), upper=c(b1=500, b2=100, b3=.25))
> a1vm2i # Fails to get to solution!
$par
    b1     b2     b3 
35.532  0.000  0.250 

$value
[1] 9205.4

$counts
function gradient 
       6        4 

$convergence
[1] 2

$message
[1] "Rvmminb appears to have converged"

$bdmsk
[1]  1 -3 -1
> # Bounds -- feasible start
> a1cg2f <- Rcgmin(startf1, hobbs.f, hobbs.g,
+      lower=c(b1=0, b2=0, b3=0), upper=c(b1=500, b2=100, b3=.25))
> a1cg2f
$par
     b1      b2      b3 
500.000  87.942   0.250 

$value
[1] 29.993

$counts
[1] 67 34

$convergence
[1] 0

$message
[1] "Rcgmin seems to have converged"
```

```
$bdmsk
[1] -1  1 -1

> a1vm2f <- Rvmmin(startf1, hobbs.f, hobbs.g,
+     lower=c(b1=0, b2=0, b3=0), upper=c(b1=500, b2=100, b3=.25))
> a1vm2f # Gets there, but only just!
$par
        b1         b2         b3 
3.5532e+01 1.6777e-15 2.5000e-01 

$value
[1] 9205.4

$counts
function gradient 
      31        5 

$convergence
[1] 2

$message
[1] "Rvmminb appears to have converged"

$bdmsk
[1]  1 -3 -1

> # Mask b2
> a1cgm2 <- Rcgmin(start1, hobbs.f, hobbs.g, bdmsk=c(1,0,1))
> a1cgm2
$par
      b1       b2       b3 
50.40179  1.00000  0.19861 

$value
[1] 6181.2

$counts
[1] 1006  129

$convergence
[1] 1

$message
[1] "Too many function evaluations (> 1000) "

$bdmsk
[1] 1 0 1

> a1vmm2 <- Rvmmin(start1, hobbs.f, hobbs.g, bdmsk=c(1,0,1))
> a1vmm2
$par
      b1       b2       b3 
50.40179  1.00000  0.19861 

$value
[1] 6181.2

$counts
function gradient 
     290       28 

$convergence
[1] 3

$message
[1] "Rvmminb appears to have converged"
```

```
$bdmsk
[1] 1 0 1

> # Mask b3
> a1cgm3 <- Rcgmin(start1, hobbs.f, hobbs.g, bdmsk=c(1,1,0))
> a1cgm3
$par
      b1       b2       b3
  78.571 2293.937    1.000

$value
[1] 1031

$counts
[1] 172  71

$convergence
[1] 0

$message
[1] "Rcgmin seems to have converged"

$bdmsk
[1] 1 1 0

> a1vmm3 <- Rvmmin(start1, hobbs.f, hobbs.g, bdmsk=c(1,1,0))
> a1vmm3
$par
      b1       b2       b3
  78.571 2293.947    1.000

$value
[1] 1031

$counts
function gradient
      88       30

$convergence
[1] 0

$message
[1] "Rvmminb appears to have converged"

$bdmsk
[1] 1 1 0

> # Mask all -- should fail
> a1cgma <- Rcgmin(start1, hobbs.f, hobbs.g, bdmsk=c(0,0,0))
> a1cgma
$par
b1 b2 b3
 1  1  1

$value
[1] 23521

$counts
[1] 1 1

$convergence
[1] 0

$message
[1] "Rcgmin seems to have converged"
```

```
$bdmsk
[1] 0 0 0
  > a1vmma <- Rvmmin(start1, hobbs.f, hobbs.g, bdmsk=c(0,0,0))
  > a1vmma
$par
b1 b2 b3
 1  1  1

$value
[1] 23521

$counts
function gradient
       1        1

$convergence
[1] 0

$message
[1] "Rvmminb appears to have converged"

$bdmsk
[1] 0 0 0
  > # Bounds and mask
  > ansmbm2 <- nlfb(startf1, hobbs.res, hobbs.jac, maskidx=c(2),
  +       lower=c(0,0,0), upper=c(200, 60, .3))
  > ansmbm2
nlmrt class object: x
residual sumsquares =  6181.2  on  12 observations
    after  17    Jacobian and  28 function evaluations
  name            coeff          SE       tstat      pval      gradient    JSingval
b1               50.4016           NA         NA         NA   0.0004618       162.2
b2                     1  M        NA         NA         NA           0      0.4918
b3              0.198618           NA         NA         NA     -0.0746           0
  > a1cgbm2 <- Rcgmin(start1, hobbs.f, hobbs.g, bdmsk=c(1,0,1),
  +        lower=c(0,0,0), upper=c(200, 60, .3))
  > a1cgbm2
$par
      b1       b2       b3
50.40179  1.00000  0.19861

$value
[1] 6181.2

$counts
[1] 1004  118

$convergence
[1] 1

$message
[1] "Too many function evaluations (> 1000) "

$bdmsk
[1] 1 0 1
  > a1vmbm2 <- Rvmmin(start1, hobbs.f, hobbs.g, bdmsk=c(1,0,1),
  +        lower=c(0,0,0), upper=c(200, 60, .3))
  > a1vmbm2
$par
      b1       b2       b3
```

```
50.40179  1.00000  0.19861

$value
[1] 6181.2

$counts
function gradient
      31       12

$convergence
[1] 0

$message
[1] "Rvmminb appears to have converged"

$bdmsk
[1] 1 0 1
```

```
> # Active bound
> a1cgm2x <- Rcgmin(start1, hobbs.f, hobbs.g, bdmsk=c(1,0,1),
+         lower=c(0,0,0), upper=c(48, 60, .3))
> a1cgm2x
```

```
$par
      b1       b2       b3
48.00000  1.00000  0.21597

$value
[1] 6206.1

$counts
[1] 1005  115

$convergence
[1] 1

$message
[1] "Too many function evaluations (> 1000) "

$bdmsk
[1] -1  0  1
```

```
> a1vmm2x <- Rvmmin(start1, hobbs.f, hobbs.g, bdmsk=c(1,0,1),
+         lower=c(0,0,0), upper=c(48, 60, .3))
> a1vmm2x
```

```
$par
  b1   b2   b3
48.0  1.0  0.3

$value
[1] 6463.3

$counts
function gradient
       2        2

$convergence
[1] 2

$message
[1] "Rvmminb appears to have converged"

$bdmsk
[1] -1  0 -1
```

6 Brief example of `minpack.lm`

Recently Kate Mullen provided some capability for the package `minpack.lm` to include bounds constraints. I am particularly happy that this effort is proceeding, as there are significant differences in how `minpack.lm` and `nlmrt` are built and implemented. They can be expected to have different performance characteristics on different problems. A lively dialogue between developers, and the opportunity to compare and check results can only improve the tools.

The examples below are a very quick attempt to show how to run the Ratkowsky-Huet problem with `nls.lm` from `minpack.lm`.

```
> require(minpack.lm)
> anlslm <- nls.lm(ones, lower=rep(-1000,4), upper=rep(1000,4), jres, jjac, yield=pastured
> cat("anlslm from ones\n")
```

```
anlslm from ones
```

```
> print(strwrap(anlslm))
```

```
 [1] "c(NaN, NaN, NaN, NaN)"
 [2] "c(NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN,"
 [3] "NaN, NaN, NaN, NaN, NaN, NaN)"
 [4] "c(NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN)"
 [5] "4"
 [6] "The cosine of the angle between `fvec' and any column"
 [7] "of the Jacobian is at most `gtol' in absolute value."
 [8] "list(t1 = 3, t2 = 2.3723939879224e-11, t3 ="
 [9] "5.8039519205899e-10, t4 = 1.27525858056086e-09)"
[10] "3"
[11] "c(17533.3402000004, 16864.5616372991, NaN,"
[12] "9.88131291682493e-324)"
[13] "NaN"
```

```
> anlslmh <- nls.lm(huetstart, lower=rep(-1000,4), upper=rep(1000,4), jres, jjac, yield=pa
> cat("anlslmh from huetstart\n")
```

```
anlslmh from huetstart
```

```
> print(strwrap(anlslmh))
```

```
 [1] "c(69.9551973916736, 61.6814877170941,"
 [2] "-9.20891880263443, 2.37781455978467)"
 [3] "c(9, -4.54037977686007, 105.318033221555,"
 [4] "403.043210394646, -4.54037977686007,"
 [5] "3.51002837648689, -39.5314537948583,"
 [6] "-137.559566823766, 105.318033221555,"
 [7] "-39.5314537948583, 1668.11894086464,"
 [8] "6495.67702199831, 403.043210394646,"
 [9] "-137.559566823766, 6495.67702199831,"
[10] "25481.4530263827)"
[11] "c(0.480682793156291, 0.669303022602289,"
[12] "-2.28431914156848, 0.843754801653787,"
[13] "0.734587578832198, 0.0665510313004489,"
[14] "-0.985814877917491, -0.0250630130722556,"
[15] "0.500317790294602)"
[16] "1"
[17] "Relative error in the sum of squares is at most"
[18] "`ftol'."
[19] "list(t1 = 3, t2 = 2.35105755434962, t3 ="
[20] "231.250186433367, t4 = 834.778914353851)"
[21] "42"
[22] "c(13386.9099465603, 13365.3097414383,"
[23] "13351.1970260154, 13321.6478455192, 13260.1135652244,"
[24] "13133.6391318145, 12877.8542053848, 12373.5432344283,"
[25] "11428.8257706578, 9832.87890178625, 7138.12187613237,"
[26] "3904.51162830831, 2286.64875980737, 1978.18149980306,"
```

```
[27] "1620.89081508973, 1140.58638304326, 775.173148616758,"
[28] "635.256627921479, 383.73614705125, 309.341249993346,"
[29] "219.735856060244, 177.398738179149, 156.718991828473,"
[30] "135.51359456819, 93.4016394568234, 72.8219383036211,"
[31] "66.3315609834918, 56.2809616213409, 54.9453021619838,"
[32] "53.6227655715768, 51.9760950696957, 50.1418078879665,"
[33] "48.1307021647518, 44.7097757109306, 42.8838792615115,"
[34] "32.3474231559263, 26.5253835687508, 15.352821554109,"
[35] "14.7215507012923, 8.37980617628203, 8.37589765770215,"
[36] "8.3758836534811, 8.37588355972578)"
[37] "8.37588355972578"
```

References

[1] Timur V. Elzhov, Katharine M. Mullen, Andrej-Nikolai Spiess, and Ben Bolker, *minpack.lm: R interface to the levenberg-marquardt nonlinear least-squares algorithm found in minpack, plus support for bounds*, R Project for Statistical Computing, 2012, R package version 1.1-6.

[2] S. (Sylvie) Huet et al., *Statistical tools for nonlinear regression: a practical guide with S-PLUS examples*, Springer series in statistics, pub-SV, pub-SV:adr, 1996.

[3] J. J. Moré, B. S. Garbow, and K. E. Hillstrom, *ANL-80-74, User Guide for MINPACK-1*, Tech. report, Argonne National Laboratory, 1980.

[4] J. C. Nash, *Compact numerical methods for computers : linear algebra and function minimisation*, Hilger, Bristol :, 1979 (English).

[5] David A. Ratkowsky, *Nonlinear regression modeling: A unified practical approach*, Marcel Dekker Inc., New York and Basel, 1983.