nls handbook

John C. Nash

August 22, 2012

Background

Based on the nlmrt-vignette, this document is intended to show the various commands (and some failures) for different R functions that deal with nonlinear least squares problems. It is NOT aimed at being pretty, but a collection of notes to assist in developing other documents more quickly.

Essentially this is an annotated version of extended versions of the examples provided with different packages in the R repositories and elsewhere.

Comparisons between ways of doing things always force some thinking about which approaches are "best". In writing this I (JCN) caution that "best" is always within a particular context. I believe `nls()` was developed within a group of active researchers to allow them to conduct calculations that involved extended nonlinear regressions. Many of the present users of R may have totally different expectations and needs. While I would like to see `nls()` in a form that allows more transparent understanding of how it works, it is nonetheless a very powerful tool but a product of its time and place of creation as is all software.

1 nls

`nls()` is the base installation nonlinear least squares tool. It is coded in C with an R wrapper. I find it very difficult to comprehend. However, it does seem to work most of the time, though it has some weaknesses for certain types of problems.

Following are the examples in the nls.Rd file from the distribution (this one is from R-2.15.1). I have split the examples to provide comments.

1.1 A straightforward example

The first example, chunk nlsex1, uses the built-in data set DNase.

```r
od <- options(digits = 5)  # include in case needed
require(graphics)

DNase1 <- subset(DNase, Run == 1)

## using a selfStart model
```

```
fm1DNase1 <- nls(density ~ SSlogis(log(conc), Asym, xmid, scal), DNase1)
summary(fm1DNase1)

##
## Formula: density ~ SSlogis(log(conc), Asym, xmid, scal)
##
## Parameters:
##      Estimate Std. Error t value Pr(>|t|)
## Asym   2.3452     0.0782    30.0  2.2e-13 ***
## xmid   1.4831     0.0814    18.2  1.2e-10 ***
## scal   1.0415     0.0323    32.3  8.5e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0192 on 13 degrees of freedom
##
## Number of iterations to convergence: 0
## Achieved convergence tolerance: 3.22e-06
##

## the coefficients only:
coef(fm1DNase1)

##   Asym   xmid   scal
## 2.3452 1.4831 1.0415

## including their SE, etc:
coef(summary(fm1DNase1))

##      Estimate Std. Error t value   Pr(>|t|)
## Asym   2.3452   0.078154  30.007 2.1655e-13
## xmid   1.4831   0.081353  18.230 1.2185e-10
## scal   1.0415   0.032271  32.272 8.5069e-14


## using conditional linearity
fm2DNase1 <- nls(density ~ 1/(1 + exp((xmid - log(conc))/scal)), data = DNase1,
    start = list(xmid = 0, scal = 1), algorithm = "plinear")
summary(fm2DNase1)

##
## Formula: density ~ 1/(1 + exp((xmid - log(conc))/scal))
##
## Parameters:
##      Estimate Std. Error t value Pr(>|t|)
## xmid   1.4831     0.0814    18.2  1.2e-10 ***
## scal   1.0415     0.0323    32.3  8.5e-14 ***
## .lin   2.3452     0.0782    30.0  2.2e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0192 on 13 degrees of freedom
##
## Number of iterations to convergence: 5
## Achieved convergence tolerance: 1.08e-06
##


## without conditional linearity
fm3DNase1 <- nls(density ~ Asym/(1 + exp((xmid - log(conc))/scal)), data = DNase1,
    start = list(Asym = 3, xmid = 0, scal = 1))
summary(fm3DNase1)

##
## Formula: density ~ Asym/(1 + exp((xmid - log(conc))/scal))
##
```

```
## Parameters:
##      Estimate Std. Error t value Pr(>|t|)
## Asym   2.3452     0.0782    30.0  2.2e-13 ***
## xmid   1.4831     0.0814    18.2  1.2e-10 ***
## scal   1.0415     0.0323    32.3  8.5e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0192 on 13 degrees of freedom
##
## Number of iterations to convergence: 6
## Achieved convergence tolerance: 1.95e-06
##


## using Port's nl2sol algorithm
fm4DNase1 <- nls(density ~ Asym/(1 + exp((xmid - log(conc))/scal)), data = DNase1,
    start = list(Asym = 3, xmid = 0, scal = 1), algorithm = "port")
summary(fm4DNase1)

##
## Formula: density ~ Asym/(1 + exp((xmid - log(conc))/scal))
##
## Parameters:
##      Estimate Std. Error t value Pr(>|t|)
## Asym   2.3452     0.0782    30.0  2.2e-13 ***
## xmid   1.4831     0.0814    18.2  1.2e-10 ***
## scal   1.0415     0.0323    32.3  8.5e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0192 on 13 degrees of freedom
##
## Algorithm "port", convergence message: relative convergence (4)
##
```

1.2 A problem with a computationally singular Jacobian

`nls()` is fine for the problem above. But what happens when we supply a problem that is a bit nastier, the WEEDS problem (**?**, section 12.2).

```
traceval <- FALSE
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)  # for testing
tdat <- seq_along(ydat)  # for testing
start1 <- c(b1 = 1, b2 = 1, b3 = 1)
eunsc <- y ~ b1/(1 + b2 * exp(-b3 * tt))
weeddata1 <- data.frame(y = ydat, tt = tdat)
require(nlmrt)
## check using nlmrt function nlxb
anlxb1 <- try(nlxb(eunsc, start = start1, trace = traceval, data = weeddata1))
print(anlxb1)  # ?? need a summary function

## $resid
##  [1]  0.011900 -0.032755  0.092030  0.208782  0.392634 -0.057594 -1.105728
##  [8]  0.715786 -0.107648 -0.348396  0.652593 -0.287568
##
## $jacobian
##             b1       b2       b3
##  [1,] 0.027117 -0.10543   5.1756
##  [2,] 0.036737 -0.14142  13.8849
##  [3,] 0.049596 -0.18837  27.7424
##  [4,] 0.066645 -0.24858  48.8137
##  [5,] 0.089005 -0.32404  79.5373
##  [6,] 0.117921 -0.41568 122.4383
##  [7,] 0.154635 -0.52241 179.5225
##  [8,] 0.200186 -0.63986 251.2937
##  [9,] 0.255106 -0.75941 335.5263
## [10,] 0.319083 -0.86828 426.2517
## [11,] 0.390688 -0.95133 513.7254
## [12,] 0.467334 -0.99482 586.0466
##
## $feval
## [1] 36
##
## $jeval
## [1] 22
##
## $coeffs
## [1] 196.18626  49.09164   0.31357
##
## $ssquares
## [1] 2.5873
##

## try nls no fancies
anls1 <- try(nls(eunsc, start = start1, trace = traceval, data = weeddata1))
print(anls1)

## [1] "Error in nls(eunsc, start = start1, trace = traceval, data = weeddata1) : \n  singular gradient\n"
## attr(,"class")
## [1] "try-error"
## attr(,"condition")
## <simpleError in nls(eunsc, start = start1, trace = traceval, data = weeddata1): singular gradient>

## try nls with 'port' algorithm
anls1port <- try(nls(eunsc, start = start1, trace = traceval, data = weeddata1,
    algorithm = "port"))
print(anls1port)

## 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)

## try nls with 'plinear' algorithm
eunsclin <- y ~ 1/(1 + b2 * exp(-b3 * tt))
start1lin <- c(b2 = 1, b3 = 1)
anls1plin <- try(nls(eunsclin, start = start1lin, trace = traceval, data = weeddata1,
    algorithm = "plinear"))
print(anls1plin)

## [1] "Error in nls(eunsclin, start = start1lin, trace = traceval, data = weeddata1,  : \n  step factor 0.000488281 reduced belo
## attr(,"class")
## [1] "try-error"
## attr(,"condition")
## <simpleError in nls(eunsclin, start = start1lin, trace = traceval, data = weeddata1,     algorithm = "plinear"): step factor 0
```

For the WEEDS problem, the ”port” algorithm using the ’nl2sol’ code of (**?**) finds the solutionm, though the running output prints the sum of squares divided by 2. The ”plinear” method goes to a point where the Jacobian is essentially singular. Package `nlmrt` is helpful here to check this.

```
weedss <- model2ssfun(eunsc, start1)
y <- weeddata1$y
tt <- weeddata1$tt
print(weedss(c(1802.1, 60.38966, 0.04119948), y = y, tt = tt))

## [1] 6186.8

weedjac <- model2jacfun(eunsc, start1)
JJ <- (weedjac(c(1802.1, 60.38966, 0.04119948), y = y, tt = tt))
svd(JJ)$d

## [1] 1.0750e+03 9.0358e-01 1.4087e-05
```

1.3 Weighted nonlinear regression

As of 2012-8-17, package `nlmrt` does not provide for weighting, though it would not be difficult to add. (The code is all in R .)

```
## weighted nonlinear regression
Treated <- Puromycin[Puromycin$state == "treated", ]
weighted.MM <- function(resp, conc, Vm, K) {
    ## Purpose: exactly as white book p. 451 -- RHS for nls() Weighted version
    ## of Michaelis-Menten model
    ## ---------------------------------------------------------- Arguments:
    ## 'y', 'x' and the two parameters (see book)
    ## ---------------------------------------------------------- Author:
    ## Martin Maechler, Date: 23 Mar 2001

    pred <- (Vm * conc)/(K + conc)
    (resp - pred)/sqrt(pred)
}

Pur.wt <- nls(~weighted.MM(rate, conc, Vm, K), data = Treated, start = list(Vm = 200,
    K = 0.1))
summary(Pur.wt)

##
## Formula: 0 ~ weighted.MM(rate, conc, Vm, K)
##
## Parameters:
##    Estimate Std. Error t value Pr(>|t|)
## Vm 2.07e+02   9.22e+00   22.42  7.0e-10 ***
## K  5.46e-02   7.98e-03    6.84  4.5e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.21 on 10 degrees of freedom
##
## Number of iterations to convergence: 5
## Achieved convergence tolerance: 3.83e-06
##
```

This structure does not carry over to `nlxb()` from `nlmrt`, which is using rather more pedestrian code. Nor can we call `weighted.MM` in `nlfb()`. What we can do is define a new function `wMMx`, where we have changed the response name `resp` to match the name `rate` in the date frame `Treated`. These changes are a bit of an annoyance, and suggestions of how to make the routines more equivalent are welcome (to J Nash).

```
wMMx <- function(x, rate, conc) {
    Vm <- x[[1]]
    K <- x[[2]]
    pred <- (Vm * conc)/(K + conc)
    (rate - pred)/sqrt(pred)
}
anlfb2 <- nlfb(start = list(Vm = 200, K = 0.1), wMMx, jacfn = NULL, rate = Treated$rate,
    conc = Treated$conc)
```

1.4 A different passing mechanism

Why is this useful / important??

```
## Passing arguments using a list that can not be coerced to a data.frame
lisTreat <- with(Treated, list(conc1 = conc[1], conc.1 = conc[-1], rate = rate))

weighted.MM1 <- function(resp, conc1, conc.1, Vm, K) {
    conc <- c(conc1, conc.1)
    pred <- (Vm * conc)/(K + conc)
    (resp - pred)/sqrt(pred)
}
Pur.wt1 <- nls(~weighted.MM1(rate, conc1, conc.1, Vm, K), data = lisTreat, start = list(Vm = 200,
    K = 0.1))
stopifnot(all.equal(coef(Pur.wt), coef(Pur.wt1)))
```

1.5 Putting in a Jacobian

Unfortunately, for reasons that do not seem clear to me (JCN), R in the `nls()` function uses the term "gradient" for the **matrix** that is, arguably more commonly, called the **Jacobian**.

```
## Chambers and Hastie (1992) Statistical Models in S (p. 537): If the
## value of the right side [of formula] has an attribute called 'gradient'
## this should be a matrix with the number of rows equal to the length of
## the response and one column for each parameter.

weighted.MM.grad <- function(resp, conc1, conc.1, Vm, K) {
    conc <- c(conc1, conc.1)

    K.conc <- K + conc
    dy.dV <- conc/K.conc
    dy.dK <- -Vm * dy.dV/K.conc
    pred <- Vm * dy.dV
    pred.5 <- sqrt(pred)
    dev <- (resp - pred)/pred.5
    Ddev <- -0.5 * (resp + pred)/(pred.5 * pred)
    attr(dev, "gradient") <- Ddev * cbind(Vm = dy.dV, K = dy.dK)
    dev
}

Pur.wt.grad <- nls(~weighted.MM.grad(rate, conc1, conc.1, Vm, K), data = lisTreat,
    start = list(Vm = 200, K = 0.1))

rbind(coef(Pur.wt), coef(Pur.wt1), coef(Pur.wt.grad))

##         Vm       K
## [1,] 206.8 0.05461
## [2,] 206.8 0.05461
## [3,] 206.8 0.05461
## In this example, there seems no advantage to providing the gradient.
## In other cases, there might be.
```

1.6 Zero or small residual problems

Zero residual problems give difficulty to `nls()` for reasons that appear to be related to the choice of termination criteria. After all, they are in some ways ”perfect” problems.

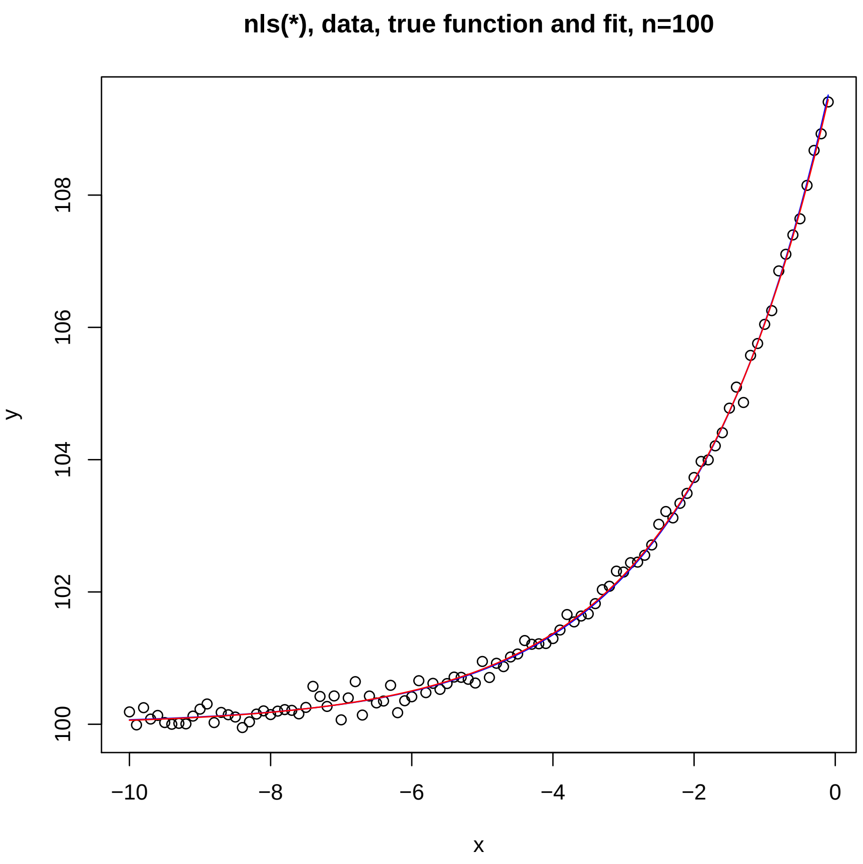
```
## The two examples below show that you can fit a model to artificial data
## with noise but not to artificial data without noise.
x <- 1:10
y <- 2 * x + 3  # perfect fit
yeps <- y + rnorm(length(y), sd = 0.01)  # added noise
test1 <- try(nls(yeps ~ a + b * x, start = list(a = 0.12345, b = 0.54321)))
print(test1)

## Nonlinear regression model
##   model:  yeps ~ a + b * x
##    data:  parent.frame()
## a b
## 3 2
##  residual sum-of-squares: 0.000384
##
## Number of iterations to convergence: 2
## Achieved convergence tolerance: 2.35e-08


## terminates in an error, because convergence cannot be confirmed:
err1 <- try(nls(y ~ a + b * x, start = list(a = 0.12345, b = 0.54321)))
test1port <- try(nls(y ~ a + b * x, start = list(a = 0.12345, b = 0.54321),
    algorithm = "port"))
print(test1port)

## Nonlinear regression model
##   model:  y ~ a + b * x
##    data:  parent.frame()
## a b
## 3 2
##  residual sum-of-squares: 0
##
## Algorithm "port", convergence message: X-convergence (3)

test1plinear <- try(nls(y ~ a + b * x, start = list(a = 0.12345, b = 0.54321),
    algorithm = "plinear"))
print(test1plinear)

## [1] "Error in qr.solve(QR.B, cc) : singular matrix 'a' in solve\n"
## attr(,"class")
## [1] "try-error"
## attr(,"condition")
## <simpleError in qr.solve(QR.B, cc): singular matrix 'a' in solve>


## Try nlmrt routine nlxb()
mydf <- data.frame(x = x, y = y, yeps = yeps)
test2 <- try(nlxb(y ~ a + b * x, start = list(a = 0.12345, b = 0.54321), data = mydf))
test2
```

```
## $resid
##  [1] 0 0 0 0 0 0 0 0 0 0
##
## $jacobian
##       a  b
##  [1,] 1  1
##  [2,] 1  2
##  [3,] 1  3
##  [4,] 1  4
##  [5,] 1  5
##  [6,] 1  6
##  [7,] 1  7
##  [8,] 1  8
##  [9,] 1  9
## [10,] 1 10
##
## $feval
## [1] 5
##
## $jeval
## [1] 5
##
## $coeffs
## [1] 3 2
##
## $ssquares
## [1] 0
##

test2eps <- try(nlxb(yeps ~ a + b * x, start = list(a = 0.12345, b = 0.54321),
    data = mydf))
test2eps

## $resid
##  [1] -0.0009035 -0.0079682  0.0085216  0.0115443 -0.0068470 -0.0069502
##  [7]  0.0027440 -0.0025939  0.0019516  0.0005013
##
## $jacobian
##       a  b
##  [1,] 1  1
##  [2,] 1  2
##  [3,] 1  3
##  [4,] 1  4
##  [5,] 1  5
##  [6,] 1  6
##  [7,] 1  7
##  [8,] 1  8
##  [9,] 1  9
## [10,] 1 10
##
## $feval
## [1] 5
##
## $jeval
## [1] 5
##
## $coeffs
## [1] 3.001 2.000
##
## $ssquares
## [1] 0.0003837
##
```

2 A note on starting values

The examples in the .Rd file for `nls()` suggests that the internal "guess" in nls() can often work. I (JCN) have generally found that a Marquardt approach is very robust even to quite extreme starts, but that Gauss-Newton ones are much more temperamental.

```
## the nls() internal cheap guess for starting values can be sufficient:
x <- -(1:100)/10
y <- 100 + 10 * exp(x/2) + rnorm(x)/10
nlmod <- nls(y ~ Const + A * exp(B * x))

## Warning:  No starting values specified for some parameters.  Initializing
## 'Const', 'A', 'B' to '1.'.  Consider specifying 'start' or using a
## selfStart model


plot(x, y, main = "nls(*), data, true function and fit, n=100")
curve(100 + 10 * exp(x/2), col = 4, add = TRUE)
lines(x, predict(nlmod), col = 2)
```



3 A more complicated model

```
## The muscle dataset in MASS is from an experiment on muscle contraction
## on 21 animals.  The observed variables are Strip (identifier of
## muscle), Conc (Cacl concentration) and Length (resulting length of
## muscle section).
utils::data(muscle, package = "MASS")

## The non linear model considered is Length = alpha +
## beta*exp(-Conc/theta) + error where theta is constant but alpha and
## beta may vary with Strip.

with(muscle, table(Strip))  # 2,3 or 4 obs per strip

## Strip
## S01 S02 S03 S04 S05 S06 S07 S08 S09 S10 S11 S12 S13 S14 S15 S16 S17 S18
##   4   4   4   3   3   3   2   2   2   2   3   2   2   2   2   4   4   3
## S19 S20 S21
##   3   3   3


## We first use the plinear algorithm to fit an overall model, ignoring
## that alpha and beta might vary with Strip.

musc.1 <- nls(Length ~ cbind(1, exp(-Conc/th)), muscle, start = list(th = 1),
    algorithm = "plinear")
summary(musc.1)

##
## Formula: Length ~ cbind(1, exp(-Conc/th))
##
## Parameters:
##       Estimate Std. Error t value Pr(>|t|)
## th       0.608      0.115    5.31  1.9e-06 ***
## .lin1   28.963      1.230   23.55  < 2e-16 ***
## .lin2  -34.227      3.793   -9.02  1.4e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.67 on 57 degrees of freedom
##
## Number of iterations to convergence: 5
## Achieved convergence tolerance: 9.34e-07
##


## Then we use nls' indexing feature for parameters in non-linear models
## to use the conventional algorithm to fit a model in which alpha and
## beta vary with Strip.  The starting values are provided by the
## previously fitted model.  Note that with indexed parameters, the
## starting values must be given in a list (with names):
b <- coef(musc.1)
musc.2 <- nls(Length ~ a[Strip] + b[Strip] * exp(-Conc/th), muscle, start = list(a = rep(b[2],
    21), b = rep(b[3], 21), th = b[1]))
summary(musc.2)

##
## Formula: Length ~ a[Strip] + b[Strip] * exp(-Conc/th)
##
## Parameters:
##     Estimate Std. Error t value Pr(>|t|)
## a1    23.454      0.796   29.46  5.0e-16 ***
## a2    28.302      0.793   35.70  < 2e-16 ***
## a3    30.801      1.716   17.95  1.7e-12 ***
## a4    25.921      3.016    8.60  1.4e-07 ***
## a5    23.201      2.891    8.02  3.5e-07 ***
## a6    20.120      2.435    8.26  2.3e-07 ***
## a7    33.595      1.682   19.98  3.0e-13 ***
## a8    39.053      3.753   10.41  8.6e-09 ***
## a9    32.137      3.318    9.69  2.5e-08 ***
## a10   40.005      3.336   11.99  1.0e-09 ***
## a11   36.190      3.109   11.64  1.6e-09 ***
## a12   36.911      1.839   20.07  2.8e-13 ***
## a13   30.635      1.700   18.02  1.6e-12 ***
## a14   34.312      3.495    9.82  2.0e-08 ***
## a15   38.395      3.375   11.38  2.3e-09 ***
## a16   31.226      0.886   35.26  < 2e-16 ***
## a17   31.230      0.821   38.02  < 2e-16 ***
## a18   19.998      1.011   19.78  3.6e-13 ***
## a19   37.095      1.071   34.65  < 2e-16 ***
## a20   32.594      1.121   29.07  6.2e-16 ***
## a21   30.376      1.057   28.74  7.5e-16 ***
## b1   -27.300      6.873   -3.97  0.00099 ***
## b2   -26.270      6.754   -3.89  0.00118 **
## b3   -30.901      2.270  -13.61  1.4e-10 ***
## b4   -32.238      3.810   -8.46  1.7e-07 ***
## b5   -29.941      3.773   -7.94  4.1e-07 ***
## b6   -20.622      3.647   -5.65  2.9e-05 ***
## b7   -19.625      8.085   -2.43  0.02661 *
## b8   -45.780      4.113  -11.13  3.2e-09 ***
## b9   -31.345      6.352   -4.93  0.00013 ***
## b10  -38.599      3.955   -9.76  2.2e-08 ***
## b11  -33.921      3.839   -8.84  9.2e-08 ***
## b12  -38.268      8.992   -4.26  0.00053 ***
## b13  -22.568      8.194   -2.75  0.01355 *
## b14  -36.167      6.358   -5.69  2.7e-05 ***
## b15  -32.952      6.354   -5.19  7.4e-05 ***
## b16  -47.207      9.540   -4.95  0.00012 ***
## b17  -33.875      7.688   -4.41  0.00039 ***
## b18  -15.896      6.222   -2.55  0.02051 *
## b19  -28.969      7.235   -4.00  0.00092 ***
## b20  -36.917      8.033   -4.60  0.00026 ***
## b21  -26.508      7.012   -3.78  0.00149 **
## th     0.797      0.127    6.30  8.0e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.11 on 17 degrees of freedom
##
## Number of iterations to convergence: 8
## Achieved convergence tolerance: 2.17e-06
##
```

4 nls2 - Gabor Grothendieck

The CRAN package `nls2` is intended to assist in finding solutions when `nls()` has difficulties. It does this by offering multiple starts. As with `nlxb()` of `nlmrt` there are some minor differences in the syntax that may make it awkward to "just change the name", but overall this is a useful tool. ?? need to put in the example from nls2 and try with nlmrt??

4.1 `nls2` examples

```
require(nls2)
y <- c(44, 36, 31, 39, 38, 26, 37, 33, 34, 48, 25, 22, 44, 5, 9, 13, 17, 15,
    21, 10, 16, 22, 13, 20, 9, 15, 14, 21, 23, 23, 32, 29, 20, 26, 31, 4, 20,
    25, 24, 32, 23, 33, 34, 23, 28, 30, 10, 29, 40, 10, 8, 12, 13, 14, 56, 47,
    44, 37, 27, 17, 32, 31, 26, 23, 31, 34, 37, 32, 26, 37, 28, 38, 35, 27,
    34, 35, 32, 27, 22, 23, 13, 28, 13, 22, 45, 33, 46, 37, 21, 28, 38, 21,
    18, 21, 18, 24, 18, 23, 22, 38, 40, 52, 31, 38, 15, 21)

x <- c(26.22, 20.45, 128.68, 117.24, 19.61, 295.21, 31.83, 30.36, 13.57, 60.47,
    205.3, 40.21, 7.99, 1.18, 5.4, 13.37, 4.51, 36.61, 7.56, 10.3, 7.29, 9.54,
    6.93, 12.6, 2.43, 18.89, 15.03, 14.49, 28.46, 36.03, 38.52, 45.16, 58.27,
    67.13, 92.33, 1.17, 29.52, 84.38, 87.57, 109.08, 72.28, 66.15, 142.27, 76.41,
    105.76, 73.47, 1.71, 305.75, 325.78, 3.71, 6.48, 19.26, 3.69, 6.27, 1689.67,
    95.23, 13.47, 8.6, 96, 436.97, 472.78, 441.01, 467.24, 1169.11, 1309.1,
    1905.16, 135.92, 438.25, 526.68, 88.88, 31.43, 21.22, 640.88, 14.09, 28.91,
    103.38, 178.99, 120.76, 161.15, 137.38, 158.31, 179.36, 214.36, 187.05,
    140.92, 258.42, 85.86, 47.7, 44.09, 18.04, 127.84, 1694.32, 34.27, 75.19,
    54.39, 79.88, 63.84, 82.24, 88.23, 202.66, 148.93, 641.76, 20.45, 145.31,
    27.52, 30.7)
```

```
## Example 1 brute force followed by nls optimization

fo <- y ~ Const + B * (x^A)

# pass our own set of starting values returning result of brute force
# search as nls object
st1 <- expand.grid(Const = seq(-100, 100, len = 4), B = seq(-100, 100, len = 4),
    A = seq(-1, 1, len = 4))
mod1 <- nls2(fo, start = st1, algorithm = "brute-force")

## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
##  -100  -100    -1
##  residual sum-of-squares: 1892244
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##  Const      B      A
##  -33.3 -100.0   -1.0
##  residual sum-of-squares: 483562
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##  Const      B      A
##   33.3 -100.0   -1.0
##  residual sum-of-squares: 17102
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
##   100  -100    -1
##  residual sum-of-squares: 492865
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##  Const      B      A
## -100.0  -33.3   -1.0
##  residual sum-of-squares: 1768740
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
## -33.3 -33.3  -1.0
##  residual sum-of-squares: 419031
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
##  33.3 -33.3  -1.0
##  residual sum-of-squares: 11545
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
## 100.0 -33.3  -1.0
##  residual sum-of-squares: 546281
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##  Const      B      A
## -100.0   33.3   -1.0
##  residual sum-of-squares: 1666758
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
## -33.3  33.3  -1.0
##  residual sum-of-squares: 376023
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
##  33.3  33.3  -1.0
##  residual sum-of-squares: 27509
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
## 100.0  33.3  -1.0
##  residual sum-of-squares: 621218
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
##  -100   100    -1
##  residual sum-of-squares: 1586298
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
## -33.3 100.0  -1.0
##  residual sum-of-squares: 354535
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
##  33.3 100.0  -1.0
##  residual sum-of-squares: 64995
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
##   100   100    -1
##  residual sum-of-squares: 717677
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##    Const        B        A
## -100.000 -100.000   -0.333
##  residual sum-of-squares: 2634134
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##    Const        B        A
##  -33.333 -100.000   -0.333
##  residual sum-of-squares: 886994
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##    Const        B        A
##   33.333 -100.000   -0.333
##  residual sum-of-squares: 82076
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##    Const        B        A
##  100.000 -100.000   -0.333
##  residual sum-of-squares: 219380
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##    Const        B        A
## -100.000  -33.333   -0.333
##  residual sum-of-squares: 1992912
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##   Const       B       A
## -33.333 -33.333  -0.333
##  residual sum-of-squares: 530384
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##   Const       B       A
##  33.333 -33.333  -0.333
##  residual sum-of-squares: 10078
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##   Const       B       A
## 100.000 -33.333  -0.333
##  residual sum-of-squares: 431994
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##    Const        B        A
## -100.000   33.333   -0.333
##  residual sum-of-squares: 1465711
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##   Const       B       A
## -33.333  33.333  -0.333
##  residual sum-of-squares: 287795
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##  Const      B      A
## 33.333 33.333 -0.333
##  residual sum-of-squares: 52101
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##   Const       B       A
## 100.000  33.333  -0.333
##  residual sum-of-squares: 758629
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##    Const        B        A
## -100.000  100.000   -0.333
##  residual sum-of-squares: 1052531
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##   Const       B       A
## -33.333 100.000  -0.333
##  residual sum-of-squares: 159227
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##   Const       B       A
##  33.333 100.000  -0.333
##  residual sum-of-squares: 208145
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##   Const       B       A
## 100.000 100.000  -0.333
##  residual sum-of-squares: 1199285
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##    Const        B        A
## -100.000 -100.000    0.333
##  residual sum-of-squares: 4e+07
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##    Const        B        A
##  -33.333 -100.000    0.333
##  residual sum-of-squares: 32468248
##
```

```
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##    Const        B        A
##   33.333 -100.000    0.333
##  residual sum-of-squares: 25905069
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##    Const        B        A
##  100.000 -100.000    0.333
##  residual sum-of-squares: 20284112
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##    Const        B        A
## -100.000  -33.333    0.333
##  residual sum-of-squares: 8626264
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##   Const       B       A
## -33.333 -33.333   0.333
##  residual sum-of-squares: 5244315
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##   Const       B       A
##  33.333 -33.333   0.333
##  residual sum-of-squares: 2804589
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##   Const       B       A
## 100.000 -33.333   0.333
##  residual sum-of-squares: 1307085
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##    Const        B        A
## -100.000   33.333    0.333
##  residual sum-of-squares: 645513
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##   Const       B       A
## -33.333  33.333   0.333
##  residual sum-of-squares: 1387017
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##  Const      B      A
## 33.333 33.333  0.333
##  residual sum-of-squares: 3070744
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##   Const       B       A
## 100.000  33.333   0.333
##  residual sum-of-squares: 5696692
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##    Const        B        A
## -100.000  100.000    0.333
##  residual sum-of-squares: 1.6e+07
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##   Const       B       A
## -33.333 100.000   0.333
##  residual sum-of-squares: 20896354
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##   Const       B       A
##  33.333 100.000   0.333
##  residual sum-of-squares: 26703533
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##   Const       B       A
## 100.000 100.000   0.333
##  residual sum-of-squares: 33452934
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
##  -100  -100     1
##  residual sum-of-squares: 1.56e+11
```

```
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##  Const      B      A
##  -33.3 -100.0    1.0
##  residual sum-of-squares: 1.56e+11
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##  Const      B      A
##   33.3 -100.0    1.0
##  residual sum-of-squares: 1.56e+11
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
##   100  -100     1
##  residual sum-of-squares: 1.55e+11
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##  Const      B      A
## -100.0  -33.3    1.0
##  residual sum-of-squares: 1.74e+10
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
## -33.3 -33.3   1.0
##  residual sum-of-squares: 1.74e+10
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
##  33.3 -33.3   1.0
##  residual sum-of-squares: 1.73e+10
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
## 100.0 -33.3   1.0
##  residual sum-of-squares: 1.72e+10
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##  Const      B      A
## -100.0   33.3    1.0
##  residual sum-of-squares: 1.71e+10
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
## -33.3  33.3   1.0
##  residual sum-of-squares: 1.72e+10
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
##  33.3  33.3   1.0
##  residual sum-of-squares: 1.73e+10
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
## 100.0  33.3   1.0
##  residual sum-of-squares: 1.74e+10
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
##  -100   100     1
##  residual sum-of-squares: 1.55e+11
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
## -33.3 100.0   1.0
##  residual sum-of-squares: 1.55e+11
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
##  33.3 100.0   1.0
##  residual sum-of-squares: 1.56e+11
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
##   100   100     1
##  residual sum-of-squares: 1.56e+11
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA

mod1

## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##   Const       B       A
##  33.333 -33.333  -0.333
##  residual sum-of-squares: 10078
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA

# use nls object mod1 just calculated as starting value for nls
# optimization.  Same as: nls(fo, start = coef(mod1))
nls2(fo, start = mod1)

## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##   Const       B       A
##  33.929 -33.459  -0.446
##  residual sum-of-squares: 8751
##
## Number of iterations to convergence: 3
## Achieved convergence tolerance: 3.05e-06
```

```
## Example 2

# pass a 2-row data frame and let nls2 calculate grid
st2 <- data.frame(Const = c(-100, 100), B = c(-100, 100), A = c(-1, 1))
mod2 <- nls2(fo, start = st2, algorithm = "brute-force")

## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
##  -100  -100    -1
##  residual sum-of-squares: 1892244
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##  Const      B      A
##  -33.3 -100.0   -1.0
##  residual sum-of-squares: 483562
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##  Const      B      A
##   33.3 -100.0   -1.0
##  residual sum-of-squares: 17102
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
##   100  -100    -1
##  residual sum-of-squares: 492865
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##  Const      B      A
## -100.0  -33.3   -1.0
##  residual sum-of-squares: 1768740
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
## -33.3 -33.3  -1.0
##  residual sum-of-squares: 419031
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
##  33.3 -33.3  -1.0
##  residual sum-of-squares: 11545
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
## 100.0 -33.3  -1.0
##  residual sum-of-squares: 546281
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##  Const      B      A
## -100.0   33.3   -1.0
##  residual sum-of-squares: 1666758
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
## -33.3  33.3  -1.0
##  residual sum-of-squares: 376023
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
##  33.3  33.3  -1.0
##  residual sum-of-squares: 27509
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
## 100.0  33.3  -1.0
##  residual sum-of-squares: 621218
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
##  -100   100    -1
##  residual sum-of-squares: 1586298
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
## -33.3 100.0  -1.0
##  residual sum-of-squares: 354535
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
##  33.3 100.0  -1.0
##  residual sum-of-squares: 64995
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
##   100   100    -1
##  residual sum-of-squares: 717677
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##    Const        B        A
## -100.000 -100.000   -0.333
##  residual sum-of-squares: 2634134
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##    Const        B        A
##  -33.333 -100.000   -0.333
##  residual sum-of-squares: 886994
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##    Const        B        A
##   33.333 -100.000   -0.333
##  residual sum-of-squares: 82076
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##    Const        B        A
##  100.000 -100.000   -0.333
##  residual sum-of-squares: 219380
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##    Const        B        A
## -100.000  -33.333   -0.333
##  residual sum-of-squares: 1992912
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##   Const       B       A
## -33.333 -33.333  -0.333
##  residual sum-of-squares: 530384
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##   Const       B       A
##  33.333 -33.333  -0.333
##  residual sum-of-squares: 10078
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##   Const       B       A
## 100.000 -33.333  -0.333
##  residual sum-of-squares: 431994
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##    Const        B        A
## -100.000   33.333   -0.333
##  residual sum-of-squares: 1465711
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##   Const       B       A
## -33.333  33.333  -0.333
##  residual sum-of-squares: 287795
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##  Const      B      A
## 33.333 33.333 -0.333
##  residual sum-of-squares: 52101
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##   Const       B       A
## 100.000  33.333  -0.333
##  residual sum-of-squares: 758629
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##    Const        B        A
## -100.000  100.000   -0.333
##  residual sum-of-squares: 1052531
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##   Const       B       A
## -33.333 100.000  -0.333
##  residual sum-of-squares: 159227
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##   Const       B       A
##  33.333 100.000  -0.333
##  residual sum-of-squares: 208145
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##   Const       B       A
## 100.000 100.000  -0.333
##  residual sum-of-squares: 1199285
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##    Const        B        A
## -100.000 -100.000    0.333
##  residual sum-of-squares: 4e+07
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##    Const        B        A
##  -33.333 -100.000    0.333
##  residual sum-of-squares: 32468248
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##    Const        B        A
##   33.333 -100.000    0.333
##  residual sum-of-squares: 25905069
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##    Const        B        A
##  100.000 -100.000    0.333
##  residual sum-of-squares: 20284112
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##    Const        B        A
## -100.000  -33.333    0.333
##  residual sum-of-squares: 8626264
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##   Const       B       A
## -33.333 -33.333   0.333
##  residual sum-of-squares: 5244315
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##   Const       B       A
##  33.333 -33.333   0.333
##  residual sum-of-squares: 2804589
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##   Const       B       A
## 100.000 -33.333   0.333
##  residual sum-of-squares: 1307085
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##    Const        B        A
## -100.000   33.333    0.333
##  residual sum-of-squares: 645513
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##   Const       B       A
## -33.333  33.333   0.333
##  residual sum-of-squares: 1387017
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##  Const      B      A
## 33.333 33.333  0.333
##  residual sum-of-squares: 3070744
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##   Const       B       A
## 100.000  33.333   0.333
##  residual sum-of-squares: 5696692
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##    Const        B        A
## -100.000  100.000    0.333
##  residual sum-of-squares: 1.6e+07
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##   Const       B       A
## -33.333 100.000   0.333
##  residual sum-of-squares: 20896354
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##   Const       B       A
##  33.333 100.000   0.333
##  residual sum-of-squares: 26703533
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##   Const       B       A
## 100.000 100.000   0.333
##  residual sum-of-squares: 33452934
##
```

```
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
##  -100  -100     1
##  residual sum-of-squares: 1.56e+11
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##  Const      B      A
##  -33.3 -100.0    1.0
##  residual sum-of-squares: 1.56e+11
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##  Const      B      A
##   33.3 -100.0    1.0
##  residual sum-of-squares: 1.56e+11
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
##   100  -100     1
##  residual sum-of-squares: 1.55e+11
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##  Const      B      A
## -100.0  -33.3    1.0
##  residual sum-of-squares: 1.74e+10
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
## -33.3 -33.3   1.0
##  residual sum-of-squares: 1.74e+10
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
##  33.3 -33.3   1.0
##  residual sum-of-squares: 1.73e+10
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
## 100.0 -33.3   1.0
##  residual sum-of-squares: 1.72e+10
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##  Const      B      A
## -100.0   33.3    1.0
##  residual sum-of-squares: 1.71e+10
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
## -33.3  33.3   1.0
##  residual sum-of-squares: 1.72e+10
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
##  33.3  33.3   1.0
##  residual sum-of-squares: 1.73e+10
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
## 100.0  33.3   1.0
##  residual sum-of-squares: 1.74e+10
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
##  -100   100     1
##  residual sum-of-squares: 1.55e+11
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
## -33.3 100.0   1.0
##  residual sum-of-squares: 1.55e+11
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
##  33.3 100.0   1.0
##  residual sum-of-squares: 1.56e+11
```

```
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
## Const     B     A
##   100   100     1
##  residual sum-of-squares: 1.56e+11
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA

mod2

## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  NULL
##   Const       B       A
##  33.333 -33.333  -0.333
##  residual sum-of-squares: 10078
##
## Number of iterations to convergence: 64
## Achieved convergence tolerance: NA

# use nls object mod1 just calculated as starting value for nls
# optimization.  Same as: nls(fo, start = coef(mod2))
nls2(fo, start = mod2)

## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##   Const       B       A
##  33.929 -33.459  -0.446
##  residual sum-of-squares: 8751
##
## Number of iterations to convergence: 3
## Achieved convergence tolerance: 3.05e-06
```

```
## Example 3

# Create same starting values as in Example 2 running an nls optimization
# from each one and picking best.  This one does an nls optimization for
# every random point generated whereas Example 2 only does a single nls
# optimization
nls2(fo, start = st2, control = nls.control(warnOnly = TRUE))

## Warning:  step factor 0.000488281 reduced below 'minFactor' of 0.000976562

## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##     Const         B         A
## -1.23e+03  1.23e+03 -4.38e-02
##  residual sum-of-squares: 5812544
##
## Number of iterations till stop: 1
## Achieved convergence tolerance: 2.72
## Reason stopped: step factor 0.000488281 reduced below 'minFactor' of 0.000976562

## Warning:  number of iterations exceeded maximum of 50

## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##     Const         B         A
## -184.9610  194.4494    0.0184
##  residual sum-of-squares: 10106
##
## Number of iterations till stop: 50
## Achieved convergence tolerance: 0.375
## Reason stopped: number of iterations exceeded maximum of 50
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##   Const       B       A
##  33.929 -33.460  -0.446
##  residual sum-of-squares: 8751
##
## Number of iterations to convergence: 4
## Achieved convergence tolerance: 1.84e-06

## Warning:  step factor 0.000488281 reduced below 'minFactor' of 0.000976562

## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##     Const         B         A
## -57.99368  67.12420   0.00586
##  residual sum-of-squares: 38758
##
## Number of iterations till stop: 3
## Achieved convergence tolerance: 1.82
## Reason stopped: step factor 0.000488281 reduced below 'minFactor' of 0.000976562
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##   Const       B       A
##  33.929 -33.459  -0.446
##  residual sum-of-squares: 8751
##
## Number of iterations to convergence: 4
## Achieved convergence tolerance: 9.39e-07
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##   Const       B       A
##  33.929 -33.459  -0.446
##  residual sum-of-squares: 8751
##
## Number of iterations to convergence: 4
## Achieved convergence tolerance: 3.83e-06

## Warning:  number of iterations exceeded maximum of 50

## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##    Const        B        A
## -47.4068  60.9894   0.0331
##  residual sum-of-squares: 11764
##
## Number of iterations till stop: 50
## Achieved convergence tolerance: 0.571
## Reason stopped: number of iterations exceeded maximum of 50
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##   Const       B       A
##  33.929 -33.459  -0.446
##  residual sum-of-squares: 8751
```

```
##
## Number of iterations to convergence: 5
## Achieved convergence tolerance: 6.77e-06

## Warning:  number of iterations exceeded maximum of 50

## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##     Const         B         A
## -161.3074  170.1414    0.0194
##  residual sum-of-squares: 11204
##
## Number of iterations till stop: 50
## Achieved convergence tolerance: 0.514
## Reason stopped: number of iterations exceeded maximum of 50

## Warning:  number of iterations exceeded maximum of 50

## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##    Const        B        A
## -59.0217  72.2264   0.0251
##  residual sum-of-squares: 13441
##
## Number of iterations till stop: 50
## Achieved convergence tolerance: 0.718
## Reason stopped: number of iterations exceeded maximum of 50

## Warning:  step factor 0.000488281 reduced below 'minFactor' of 0.000976562

## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##     Const         B         A
## -7.51e+02  7.27e+02  6.35e-03
##  residual sum-of-squares: 120778
##
## Number of iterations till stop: 28
## Achieved convergence tolerance: 3.56
## Reason stopped: step factor 0.000488281 reduced below 'minFactor' of 0.000976562
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##   Const       B       A
##  33.929 -33.459  -0.446
##  residual sum-of-squares: 8751
##
## Number of iterations to convergence: 5
## Achieved convergence tolerance: 1.66e-06

## Warning:  number of iterations exceeded maximum of 50

## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##     Const         B         A
## -142.9734  152.2533    0.0223
##  residual sum-of-squares: 10539
##
## Number of iterations till stop: 50
## Achieved convergence tolerance: 0.435
## Reason stopped: number of iterations exceeded maximum of 50

## Warning:  step factor 0.000488281 reduced below 'minFactor' of 0.000976562
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##     Const         B         A
## -256.0272  258.3887    0.0133
##  residual sum-of-squares: 20482
##
## Number of iterations till stop: 47
## Achieved convergence tolerance: 1.15
## Reason stopped: step factor 0.000488281 reduced below 'minFactor' of 0.000976562
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##   Const       B       A
##  33.929 -33.460  -0.446
##  residual sum-of-squares: 8751
##
## Number of iterations to convergence: 5
## Achieved convergence tolerance: 6.56e-07

## Warning:  step factor 0.000488281 reduced below 'minFactor' of 0.000976562

## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##     Const         B         A
## -254.4659  262.0286    0.0152
##  residual sum-of-squares: 10449
##
## Number of iterations till stop: 36
## Achieved convergence tolerance: 0.424
## Reason stopped: step factor 0.000488281 reduced below 'minFactor' of 0.000976562

## Warning:  step factor 0.000488281 reduced below 'minFactor' of 0.000976562

## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##    Const        B        A
## -36.9876  49.7021   0.0246
##  residual sum-of-squares: 18685
##
## Number of iterations till stop: 30
## Achieved convergence tolerance: 1.05
## Reason stopped: step factor 0.000488281 reduced below 'minFactor' of 0.000976562

## Warning:  number of iterations exceeded maximum of 50

## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##     Const         B         A
## -215.4769  219.8816    0.0178
##  residual sum-of-squares: 13411
##
## Number of iterations till stop: 50
## Achieved convergence tolerance: 0.717
## Reason stopped: number of iterations exceeded maximum of 50

## Warning:  step factor 0.000488281 reduced below 'minFactor' of 0.000976562

## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##    Const        B        A
## -40.1413  50.9259   0.0232
##  residual sum-of-squares: 23147
##
## Number of iterations till stop: 18
## Achieved convergence tolerance: 1.27
## Reason stopped: step factor 0.000488281 reduced below 'minFactor' of 0.000976562

## Warning:  number of iterations exceeded maximum of 50

## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##    Const        B        A
## -14.5346  31.5797   0.0485
##  residual sum-of-squares: 10830
##
## Number of iterations till stop: 50
## Achieved convergence tolerance: 0.468
## Reason stopped: number of iterations exceeded maximum of 50
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##   Const       B       A
##  33.929 -33.459  -0.446
##  residual sum-of-squares: 8751
##
## Number of iterations to convergence: 7
## Achieved convergence tolerance: 4.09e-07
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##   Const       B       A
##  33.929 -33.460  -0.446
##  residual sum-of-squares: 8751
##
## Number of iterations to convergence: 6
## Achieved convergence tolerance: 8.19e-07

## Warning:  number of iterations exceeded maximum of 50

## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##   Const       B       A
## -34.031  48.857   0.038
##  residual sum-of-squares: 11291
##
## Number of iterations till stop: 50
## Achieved convergence tolerance: 0.522
## Reason stopped: number of iterations exceeded maximum of 50

## Warning:  step factor 0.000488281 reduced below ’minFactor’ of 0.000976562

## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##     Const         B         A
## -7.14e+02  7.03e+02  7.96e-03
##  residual sum-of-squares: 35719
##
## Number of iterations till stop: 5
## Achieved convergence tolerance: 1.74
## Reason stopped: step factor 0.000488281 reduced below 'minFactor' of 0.000976562

## Warning:  number of iterations exceeded maximum of 50
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##    Const        B        A
## -19.6630  36.0402   0.0464
##  residual sum-of-squares: 10766
##
## Number of iterations till stop: 50
## Achieved convergence tolerance: 0.46
## Reason stopped: number of iterations exceeded maximum of 50

## Warning:  step factor 0.000488281 reduced below 'minFactor' of 0.000976562

## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##    Const        B        A
## -101.041  108.884    0.011
##  residual sum-of-squares: 31138
##
## Number of iterations till stop: 1
## Achieved convergence tolerance: 1.58
## Reason stopped: step factor 0.000488281 reduced below 'minFactor' of 0.000976562

## Warning:  step factor 0.000488281 reduced below 'minFactor' of 0.000976562

## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##    Const        B        A
## -93.3433 103.1961   0.0187
##  residual sum-of-squares: 17956
##
## Number of iterations till stop: 2
## Achieved convergence tolerance: 1.01
## Reason stopped: step factor 0.000488281 reduced below 'minFactor' of 0.000976562

## Warning:  step factor 0.000488281 reduced below 'minFactor' of 0.000976562

## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##     Const         B         A
## -198.3936  205.8454    0.0175
##  residual sum-of-squares: 11467
##
## Number of iterations till stop: 46
## Achieved convergence tolerance: 0.543
## Reason stopped: step factor 0.000488281 reduced below 'minFactor' of 0.000976562
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##   Const       B       A
##  33.929 -33.460  -0.446
##  residual sum-of-squares: 8751
##
## Number of iterations to convergence: 5
## Achieved convergence tolerance: 4.07e-07

## Warning:  number of iterations exceeded maximum of 50

## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##    Const        B        A
## -30.8103  45.8360   0.0426
##  residual sum-of-squares: 10697
##
## Number of iterations till stop: 50
## Achieved convergence tolerance: 0.452
## Reason stopped: number of iterations exceeded maximum of 50
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##   Const       B       A
##  33.929 -33.459  -0.446
##  residual sum-of-squares: 8751
##
## Number of iterations to convergence: 6
## Achieved convergence tolerance: 8.52e-06

## Warning:  number of iterations exceeded maximum of 50

## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##    Const        B        A
## -17.5262  33.8811   0.0499
##  residual sum-of-squares: 10671
##
## Number of iterations till stop: 50
## Achieved convergence tolerance: 0.448
## Reason stopped: number of iterations exceeded maximum of 50

## Warning:  number of iterations exceeded maximum of 50

## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##     Const         B         A
## -131.1733  141.2249    0.0233
##  residual sum-of-squares: 10298
##
## Number of iterations till stop: 50
## Achieved convergence tolerance: 0.403
## Reason stopped: number of iterations exceeded maximum of 50
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##   Const       B       A
##  33.929 -33.460  -0.446
##  residual sum-of-squares: 8751
##
## Number of iterations to convergence: 7
## Achieved convergence tolerance: 3.96e-07
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##   Const       B       A
##  33.929 -33.459  -0.446
##  residual sum-of-squares: 8751
##
## Number of iterations to convergence: 4
## Achieved convergence tolerance: 8.09e-06
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##   Const       B       A
##  33.929 -33.459  -0.446
##  residual sum-of-squares: 8751
##
## Number of iterations to convergence: 6
## Achieved convergence tolerance: 1.49e-06
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##   Const       B       A
##  33.929 -33.459  -0.446
##  residual sum-of-squares: 8751
##
## Number of iterations to convergence: 12
## Achieved convergence tolerance: 1.16e-06

## Warning:  number of iterations exceeded maximum of 50

## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##    Const        B        A
## -24.1355  39.9129   0.0447
##  residual sum-of-squares: 10768
##
## Number of iterations till stop: 50
## Achieved convergence tolerance: 0.461
## Reason stopped: number of iterations exceeded maximum of 50

## Warning:  step factor 0.000488281 reduced below 'minFactor' of 0.000976562

## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##     Const         B         A
## -300.9404  309.0298    0.0131
##  residual sum-of-squares: 10099
##
## Number of iterations till stop: 32
## Achieved convergence tolerance: 0.375
## Reason stopped: step factor 0.000488281 reduced below 'minFactor' of 0.000976562

## Warning:  step factor 0.000488281 reduced below 'minFactor' of 0.000976562

## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##    Const        B        A
## -19.4832  40.9415   0.0091
##  residual sum-of-squares: 13105
##
## Number of iterations till stop: 2
## Achieved convergence tolerance: 0.682
## Reason stopped: step factor 0.000488281 reduced below 'minFactor' of 0.000976562
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##   Const       B       A
##  33.929 -33.459  -0.446
##  residual sum-of-squares: 8751
##
## Number of iterations to convergence: 4
## Achieved convergence tolerance: 3.71e-06

## Warning:  number of iterations exceeded maximum of 50

## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##    Const        B        A
## -13.9098  30.7459   0.0523
##  residual sum-of-squares: 10636
```

```
##
## Number of iterations till stop: 50
## Achieved convergence tolerance: 0.443
## Reason stopped: number of iterations exceeded maximum of 50

## Warning:  step factor 0.000488281 reduced below 'minFactor' of 0.000976562

## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##   Const       B       A
## 24.2068  2.6453  0.0449
##  residual sum-of-squares: 12042
##
## Number of iterations till stop: 9
## Achieved convergence tolerance: 0.595
## Reason stopped: step factor 0.000488281 reduced below 'minFactor' of 0.000976562
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##   Const       B       A
##  33.929 -33.459  -0.446
##  residual sum-of-squares: 8751
##
## Number of iterations to convergence: 5
## Achieved convergence tolerance: 5.37e-07
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##   Const       B       A
##  33.929 -33.459  -0.446
##  residual sum-of-squares: 8751
##
## Number of iterations to convergence: 6
## Achieved convergence tolerance: 4.83e-06

## Warning:  step factor 0.000488281 reduced below 'minFactor' of 0.000976562

## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##    Const        B        A
## -65.1758  77.6498   0.0231
##  residual sum-of-squares: 14612
##
## Number of iterations till stop: 45
## Achieved convergence tolerance: 0.805
## Reason stopped: step factor 0.000488281 reduced below 'minFactor' of 0.000976562

## Warning:  step factor 0.000488281 reduced below 'minFactor' of 0.000976562

## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##     Const         B         A
## -225.5549  216.1429    0.0102
##  residual sum-of-squares: 87601
##
## Number of iterations till stop: 24
## Achieved convergence tolerance: 2.98
## Reason stopped: step factor 0.000488281 reduced below 'minFactor' of 0.000976562

## Warning:  step factor 0.000488281 reduced below 'minFactor' of 0.000976562

## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##     Const         B         A
## -6.17e+02  6.24e+02  5.27e-03
##  residual sum-of-squares: 12746
##
## Number of iterations till stop: 4
## Achieved convergence tolerance: 0.658
## Reason stopped: step factor 0.000488281 reduced below 'minFactor' of 0.000976562
## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##   Const       B       A
##  33.929 -33.460  -0.446
##  residual sum-of-squares: 8751
##
## Number of iterations to convergence: 3
## Achieved convergence tolerance: 8.96e-06

## Warning:  step factor 0.000488281 reduced below 'minFactor' of 0.000976562

## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##    Const        B        A
## -58.4618  71.1143   0.0235
##  residual sum-of-squares: 15170
##
## Number of iterations till stop: 43
## Achieved convergence tolerance: 0.843
## Reason stopped: step factor 0.000488281 reduced below 'minFactor' of 0.000976562

## Nonlinear regression model
##   model:  y ~ Const + B * (x^A)
##    data:  <environment>
##   Const       B       A
##  33.929 -33.460  -0.446
##  residual sum-of-squares: 8751
##
## Number of iterations to convergence: 5
## Achieved convergence tolerance: 4.07e-07
```

```
## Example 4

# Investigate singular gradient.  Note that this cannot be done with nls
# since the singular gradient at the initial conditions would stop it with
# an error.

DF1 <- data.frame(y = 1:9, one = rep(1, 9))
xx <- nls2(y ~ (a + 2 * b) * one, DF1, start = c(a = 1, b = 1), algorithm = "brute-force")
svd(xx$m$Rmat())[-2]

## $d
## [1] 6.708e+00 1.404e-16
##
## $v
##         [,1]    [,2]
## [1,] -0.4472  0.8944
## [2,] -0.8944 -0.4472
##
```

```
## Example 5

# Use plinear algorithm to reduce a 4 parameter model to a model with 2
# linear and 2 nonlinear parameters
```

```
## Fixed spelling error in example that is 'don't run' data(Ratkowsky,
## package = 'NISTnls') # Ratkowsky2 data set
data(Ratkowsky2, package = "NISTnls")  # Ratkowsky2 data set
# fo corresponds to the model on page 13 of Huet et al.
fo <- y ~ cbind(rep(1, 9), exp(-exp(p3 + p4 * log(x))))
st <- data.frame(p3 = c(-100, 100), p4 = c(-100, 100))
## Fixed spelling error in example that is 'don't run' rat.nls <- nls2(fo,
## Ratkwosky2, start = st, control = nls.control(maxiter = 200), algorithm
## = 'plinear')
rat.nls <- nls2(fo, Ratkowsky2, start = st, control = nls.control(maxiter = 200),
    algorithm = "plinear")

## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## Nonlinear regression model
##   model:  y ~ cbind(rep(1, 9), exp(-exp(p3 + p4 * log(x))))
##    data:  structure(list(y = c(8.93, 10.8, 18.59, 22.33, 39.35, 56.11,  61.73, 64.62, 67.08), x = c(9, 14, 21, 28, 42, 57, 63,
##     p3     p4  .lin1  .lin2
##  -9.21   2.38  69.96 -61.68
##  residual sum-of-squares: 8.38
##
## Number of iterations to convergence: 11
## Achieved convergence tolerance: 2.53e-06
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## Nonlinear regression model
##   model:  y ~ cbind(rep(1, 9), exp(-exp(p3 + p4 * log(x))))
##    data:  structure(list(y = c(8.93, 10.8, 18.59, 22.33, 39.35, 56.11,  61.73, 64.62, 67.08), x = c(9, 14, 21, 28, 42, 57, 63,
##     p3     p4  .lin1  .lin2
##  -9.21   2.38  69.96 -61.68
##  residual sum-of-squares: 8.38
##
## Number of iterations to convergence: 10
## Achieved convergence tolerance: 4.12e-06
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## Nonlinear regression model
##   model:  y ~ cbind(rep(1, 9), exp(-exp(p3 + p4 * log(x))))
##    data:  structure(list(y = c(8.93, 10.8, 18.59, 22.33, 39.35, 56.11,  61.73, 64.62, 67.08), x = c(9, 14, 21, 28, 42, 57, 63,
##        p3        p4     .lin1     .lin2
## -3.06e+13  1.02e+13  4.71e+01 -3.73e+01
##  residual sum-of-squares: 2490
##
## Number of iterations to convergence: 1
## Achieved convergence tolerance: 0
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## Nonlinear regression model
##   model:  y ~ cbind(rep(1, 9), exp(-exp(p3 + p4 * log(x))))
##    data:  structure(list(y = c(8.93, 10.8, 18.59, 22.33, 39.35, 56.11,  61.73, 64.62, 67.08), x = c(9, 14, 21, 28, 42, 57, 63,
##     p3     p4  .lin1  .lin2
##  -9.21   2.38  69.96 -61.68
##  residual sum-of-squares: 8.38
##
## Number of iterations to convergence: 8
## Achieved convergence tolerance: 3.04e-06
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
## [1] NA
```

```
rat.nls
```

```
## Nonlinear regression model
##   model:  y ~ cbind(rep(1, 9), exp(-exp(p3 + p4 * log(x))))
##    data:  structure(list(y = c(8.93, 10.8, 18.59, 22.33, 39.35, 56.11,  61.73, 64.62, 67.08), x = c(9, 14, 21, 28, 42, 57, 63,
##     p3     p4  .lin1  .lin2
##  -9.21   2.38  69.96 -61.68
##  residual sum-of-squares: 8.38
##
## Number of iterations to convergence: 11
## Achieved convergence tolerance: 2.53e-06
```

```
rat2.nls <- nls2(fo, Ratkowsky2, start = rat.nls, algorithm = "plinear")
rat2.nls
```

```
## Nonlinear regression model
##   model:  y ~ cbind(rep(1, 9), exp(-exp(p3 + p4 * log(x))))
##    data:  structure(list(y = c(8.93, 10.8, 18.59, 22.33, 39.35, 56.11,  61.73, 64.62, 67.08), x = c(9, 14, 21, 28, 42, 57, 63,
##     p3     p4  .lin1  .lin2
##  -9.21   2.38  69.96 -61.68
##  residual sum-of-squares: 8.38
##
## Number of iterations to convergence: 0
## Achieved convergence tolerance: 2.53e-06
```

4.2 as.lm.nls

```
# data is from ?nls
DNase1 <- subset(DNase, Run == 1)
fm1DNase1 <- nls(density ~ SSlogis(log(conc), Asym, xmid, scal), DNase1)

# these give same result
vcov(fm1DNase1)

##          Asym     xmid     scal
## Asym 0.006108 0.006274 0.002272
## xmid 0.006274 0.006618 0.002379
## scal 0.002272 0.002379 0.001041

## NOTE: had to change as.lm to as.lm.nls
vcov(as.lm.nls(fm1DNase1))

##          Asym     xmid     scal
## Asym 0.006108 0.006274 0.002272
## xmid 0.006274 0.006618 0.002379
## scal 0.002272 0.002379 0.001041


# nls confidence and prediction intervals based on asymptotic
# approximation are same as as.lm confidence intervals.  NOTE: had to
# change as.lm to as.lm.nls
predict(as.lm.nls(fm1DNase1), interval = "confidence")

##        fit     lwr     upr
## 1  0.03068 0.02442 0.03694
## 2  0.03068 0.02442 0.03694
## 3  0.11205 0.09892 0.12518
## 4  0.11205 0.09892 0.12518
## 5  0.20858 0.19246 0.22470
## 6  0.20858 0.19246 0.22470
## 7  0.37433 0.35768 0.39097
## 8  0.37433 0.35768 0.39097
## 9  0.63278 0.61640 0.64916
## 10 0.63278 0.61640 0.64916
## 11 0.98086 0.96082 1.00090
## 12 0.98086 0.96082 1.00090
## 13 1.36751 1.34799 1.38704
## 14 1.36751 1.34799 1.38704
## 15 1.71499 1.68707 1.74291
## 16 1.71499 1.68707 1.74291

## NOTE: had to change as.lm to as.lm.nls
predict(as.lm.nls(fm1DNase1), interval = "prediction")

## Warning:  Predictions on current data refer to _future_ responses

##        fit      lwr     upr
## 1  0.03068 -0.01126 0.07262
## 2  0.03068 -0.01126 0.07262
## 3  0.11205  0.06855 0.15555
## 4  0.11205  0.06855 0.15555
## 5  0.20858  0.16409 0.25307
## 6  0.20858  0.16409 0.25307
## 7  0.37433  0.32965 0.41901
## 8  0.37433  0.32965 0.41901
## 9  0.63278  0.58819 0.67736
## 10 0.63278  0.58819 0.67736
## 11 0.98086  0.93481 1.02692
## 12 0.98086  0.93481 1.02692
## 13 1.36751  1.32168 1.41335
## 14 1.36751  1.32168 1.41335
## 15 1.71499  1.66500 1.76498
## 16 1.71499  1.66500 1.76498
```

5 nlmrt

5.1 Problems using a model formula – `nlxb()`

```
rm(list = ls())
library(nlmrt)
# traceval set TRUE to debug or give full history
traceval <- FALSE
# 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)  # for testing
y <- ydat  # for testing
tdat <- seq_along(ydat)  # for testing
# WARNING -- using T can get confusion with TRUE
tt <- tdat
# A simple starting vector -- must have named parameters for nlxb, nls,
# wrapnls.
start1 <- c(b1 = 1, b2 = 1, b3 = 1)
startf1 <- c(b1 = 1, b2 = 1, b3 = 0.1)
eunsc <- y ~ b1/(1 + b2 * exp(-b3 * tt))
# set up data in data frames
weeddata1 <- data.frame(y = ydat, tt = tdat)
weeddata2 <- data.frame(y = 1.5 * ydat, tt = tdat)
```

nlmrt is not intended to be used with global data i.e., data in the environment in which the user is working. (??should this be changed??) So the calls here should fail.

```
cat("GLOBAL DATA -- nls -- SHOULD WORK\n")

## GLOBAL DATA -- nls -- SHOULD WORK

anls1g <- try(nls(eunsc, start = start1, trace = traceval))
print(anls1g)

## [1] "Error in nls(eunsc, start = start1, trace = traceval) : singular gradient\n"
## attr(,"class")
## [1] "try-error"
## attr(,"condition")
## <simpleError in nls(eunsc, start = start1, trace = traceval): singular gradient>

cat("GLOBAL DATA -- nlxb -- SHOULD FAIL\n")

## GLOBAL DATA -- nlxb -- SHOULD FAIL

anlxb1g <- try(nlxb(eunsc, start = start1, trace = traceval))
print(anlxb1g)

## [1] "Error in nlxb(eunsc, start = start1, trace = traceval) : \n  'data' must be a list or an environment\n"
## attr(,"class")
## [1] "try-error"
## attr(,"condition")
## <simpleError in nlxb(eunsc, start = start1, trace = traceval): 'data' must be a list or an environment>

rm(y)

## Warning:  object 'y' not found

rm(tt)

## Warning:  object 'tt' not found
```

```
cat("LOCAL DATA IN DATA FRAMES\n")

## LOCAL DATA IN DATA FRAMES

anlxb1 <- try(nlxb(eunsc, start = start1, trace = traceval, data = weeddata1))
print(anlxb1)

## $resid
##  [1]  0.01190 -0.03276  0.09203  0.20878  0.39263 -0.05759 -1.10573
##  [8]  0.71579 -0.10765 -0.34840  0.65259 -0.28757
##
## $jacobian
##            b1      b2      b3
##  [1,] 0.02712 -0.1054   5.176
##  [2,] 0.03674 -0.1414  13.885
##  [3,] 0.04960 -0.1884  27.742
##  [4,] 0.06664 -0.2486  48.814
##  [5,] 0.08901 -0.3240  79.537
##  [6,] 0.11792 -0.4157 122.438
##  [7,] 0.15464 -0.5224 179.522
##  [8,] 0.20019 -0.6399 251.294
##  [9,] 0.25511 -0.7594 335.526
## [10,] 0.31908 -0.8683 426.252
## [11,] 0.39069 -0.9513 513.725
## [12,] 0.46733 -0.9948 586.047
##
## $feval
## [1] 36
##
## $jeval
## [1] 22
##
## $coeffs
## [1] 196.1863  49.0916   0.3136
##
## $ssquares
## [1] 2.587
##

anlxb2 <- try(nlxb(eunsc, start = start1, trace = traceval, data = weeddata2))
print(anlxb2)

## $resid
##  [1]  0.01785 -0.04913  0.13804  0.31317  0.58895 -0.08639 -1.65859
##  [8]  1.07368 -0.16147 -0.52259  0.97889 -0.43135
##
## $jacobian
##            b1      b2      b3
##  [1,] 0.02712 -0.1581   7.763
##  [2,] 0.03674 -0.2121  20.827
##  [3,] 0.04960 -0.2826  41.614
##  [4,] 0.06664 -0.3729  73.220
##  [5,] 0.08901 -0.4861 119.306
##  [6,] 0.11792 -0.6235 183.657
##  [7,] 0.15464 -0.7836 269.284
##  [8,] 0.20019 -0.9598 376.941
##  [9,] 0.25511 -1.1391 503.289
## [10,] 0.31908 -1.3024 639.377
## [11,] 0.39069 -1.4270 770.588
## [12,] 0.46733 -1.4922 879.070
##
## $feval
## [1] 40
##
## $jeval
## [1] 24
##
```

```
## $coeffs
## [1] 294.2794  49.0916   0.3136
##
## $ssquares
## [1] 5.821
##
```

```
## With BOUNDS
anlxb1 <- try(nlxb(eunsc, start = startf1, lower = c(b1 = 0, b2 = 0, b3 = 0),
    upper = c(b1 = 500, b2 = 100, b3 = 5), trace = traceval, data = weeddata1))
print(anlxb1)

## $resid
##  [1]  0.01190 -0.03276  0.09203  0.20878  0.39263 -0.05759 -1.10573
##  [8]  0.71579 -0.10765 -0.34840  0.65259 -0.28757
##
## $jacobian
##           b1      b2      b3
##  [1,] 0.02712 -0.1054   5.176
##  [2,] 0.03674 -0.1414  13.885
##  [3,] 0.04960 -0.1884  27.742
##  [4,] 0.06664 -0.2486  48.814
##  [5,] 0.08901 -0.3240  79.537
##  [6,] 0.11792 -0.4157 122.438
##  [7,] 0.15464 -0.5224 179.522
##  [8,] 0.20019 -0.6399 251.294
##  [9,] 0.25511 -0.7594 335.526
## [10,] 0.31908 -0.8683 426.252
## [11,] 0.39069 -0.9513 513.725
## [12,] 0.46733 -0.9948 586.047
##
## $feval
## [1] 29
##
## $jeval
## [1] 17
##
## $coeffs
## [1] 196.1863  49.0916   0.3136
##
## $ssquares
## [1] 2.587
##

# Check nls too
anlsb1 <- try(nls(eunsc, start = start1, lower = c(b1 = 0, b2 = 0, b3 = 0),
    upper = c(b1 = 500, b2 = 100, b3 = 5), trace = traceval, 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)

cat("Another case -- hard upper bound\n")

## Another case -- hard upper bound

anlxb2 <- try(nlxb(eunsc, start = start1, lower = c(b1 = 0, b2 = 0, b3 = 0),
    upper = c(b1 = 500, b2 = 100, b3 = 0.25), trace = traceval, data = weeddata1))
print(anlxb2)
```

```
## [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), t

anlsb2 <- try(nls(eunsc, start = start1, lower = c(b1 = 0, b2 = 0, b3 = 0),
    upper = c(b1 = 500, b2 = 100, b3 = 0.25), trace = traceval, data = weeddata1,
    algorithm = "port"))
print(anlsb2)

## [1] "Error in nls(eunsc, start = start1, lower = c(b1 = 0, b2 = 0, b3 = 0),  : \n  Convergence failure: initial par violates c
## 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), tr
```

```
cat("TEST MASKS\n")

## 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"), trace = traceval,
    data = weeddata1))
print(anlsmnqm)

## $resid
##  [1]  22.387  22.901  22.856  21.850  19.709  15.468   8.911   3.299
##  [9]  -6.981 -18.628 -30.690 -45.827
##
## $jacobian
##           b1 b2    b3
##  [1,] 0.5495  0 12.48
##  [2,] 0.5980  0 24.23
##  [3,] 0.6447  0 34.64
##  [4,] 0.6888  0 43.22
##  [5,] 0.7297  0 49.71
##  [6,] 0.7670  0 54.04
##  [7,] 0.8006  0 56.31
##  [8,] 0.8305  0 56.77
##  [9,] 0.8566  0 55.71
## [10,] 0.8793  0 53.48
## [11,] 0.8989  0 50.40
## [12,] 0.9156  0 46.76
##
## $feval
## [1] 57
##
## $jeval
## [1] 33
##
## $coeffs
## [1] 50.4018  1.0000  0.1986
##
## $ssquares
## [1] 6181
##


cat("UNCONSTRAINED\n")

## UNCONSTRAINED

an1q <- try(nlxb(eunsc, start = start1, trace = traceval, data = weeddata1))
print(an1q)
```

```
## $resid
##  [1]  0.01190 -0.03276  0.09203  0.20878  0.39263 -0.05759 -1.10573
##  [8]  0.71579 -0.10765 -0.34840  0.65259 -0.28757
##
## $jacobian
##            b1      b2      b3
##  [1,] 0.02712 -0.1054   5.176
##  [2,] 0.03674 -0.1414  13.885
##  [3,] 0.04960 -0.1884  27.742
##  [4,] 0.06664 -0.2486  48.814
##  [5,] 0.08901 -0.3240  79.537
##  [6,] 0.11792 -0.4157 122.438
##  [7,] 0.15464 -0.5224 179.522
##  [8,] 0.20019 -0.6399 251.294
##  [9,] 0.25511 -0.7594 335.526
## [10,] 0.31908 -0.8683 426.252
## [11,] 0.39069 -0.9513 513.725
## [12,] 0.46733 -0.9948 586.047
##
## $feval
## [1] 36
##
## $jeval
## [1] 22
##
## $coeffs
## [1] 196.1863  49.0916   0.3136
##
## $ssquares
## [1] 2.587
##
```

```
cat("MASKED\n")
```

```
## MASKED
```

```
an1qm3 <- try(nlxb(eunsc, start = start1, trace = traceval, data = weeddata1,
    masked = c("b3")))
print(an1qm3)
```

```
## $resid
##  [1]  -5.2150  -6.9877  -8.9560 -11.0394 -12.2945 -11.4407  -6.0304
##  [8]   5.8440  11.0794   8.2119  -0.3233 -14.4932
##
## $jacobian
##             b1         b2 b3
##  [1,] 0.001184 -4.049e-05  0
##  [2,] 0.003211 -1.096e-04  0
##  [3,] 0.008680 -2.947e-04  0
##  [4,] 0.023248 -7.778e-04  0
##  [5,] 0.060766 -1.955e-03  0
##  [6,] 0.149563 -4.357e-03  0
##  [7,] 0.323435 -7.495e-03  0
##  [8,] 0.565121 -8.418e-03  0
##  [9,] 0.779365 -5.890e-03  0
## [10,] 0.905678 -2.926e-03  0
## [11,] 0.963101 -1.217e-03  0
## [12,] 0.986101 -4.694e-04  0
##
## $feval
## [1] 48
##
## $jeval
## [1] 31
##
## $coeffs
## [1]   78.57 2293.95    1.00
##
## $ssquares
## [1] 1031
##
```

```
# Note that the parameters are put in out of order to test code.
an1qm123 <- try(nlxb(eunsc, start = start1, trace = traceval, data = weeddata1,
    masked = c("b2", "b1", "b3")))
print(an1qm123)

## [1] "Error in nlxb(eunsc, start = start1, trace = traceval, data = weeddata1,  : \n  All parameters are masked\n"
## attr(,"class")
## [1] "try-error"
## attr(,"condition")
## <simpleError in nlxb(eunsc, start = start1, trace = traceval, data = weeddata1,     masked = c("b2", "b1", "b3")): All paramete
```

```
cat("BOUNDS")

## BOUNDS

start2 <- c(b1 = 100, b2 = 10, b3 = 0.1)
an1qb1 <- try(nlxb(eunsc, start = start2, trace = traceval, data = weeddata1,
    lower = c(0, 0, 0), upper = c(200, 60, 0.3)))

## Warning:  NaNs produced

## Warning:  NaNs produced

print(an1qb1)

## $resid
##  [1]  0.6018  0.6557  0.8749  1.0687  1.2932  0.8231 -0.3330  1.2687
##  [9]  0.1030 -0.5880 -0.0972 -1.5286
##
## $jacobian
##       b1      b2 b3
##  [1,]  0 -0.1294  0
##  [2,]  0 -0.1711  0
##  [3,]  0 -0.2247  0
##  [4,]  0 -0.2924  0
##  [5,]  0 -0.3762  0
##  [6,]  0 -0.4767  0
##  [7,]  0 -0.5926  0
##  [8,]  0 -0.7195  0
##  [9,]  0 -0.8488  0
## [10,]  0 -0.9681  0
## [11,]  0 -1.0623  0
## [12,]  0 -1.1175  0
##
## $feval
## [1] 23
##
## $jeval
## [1] 18
##
## $coeffs
## [1] 200.00  44.33   0.30
##
## $ssquares
## [1] 9.473
##
```

```
cat("BOUNDS and MASK")

## BOUNDS and MASK

an1qbm2 <- try(nlxb(eunsc, start = start2, trace = traceval, data = weeddata1,
    lower = c(0, 0, 0), upper = c(200, 60, 0.3), masked = c("b2")))
print(an1qbm2)

## $resid
##  [1]   6.513   7.662   8.983  10.158  11.049  10.667   8.696   8.230
##  [9]   3.435  -2.638  -9.275 -19.336
##
## $jacobian
##           b1 b2     b3
##  [1,] 0.1154  0  10.46
##  [2,] 0.1455  0  25.47
##  [3,] 0.1818  0  45.70
##  [4,] 0.2248  0  71.39
##  [5,] 0.2746  0 101.99
##  [6,] 0.3307  0 135.98
##  [7,] 0.3920  0 170.84
##  [8,] 0.4569  0 203.28
##  [9,] 0.5233  0 229.90
## [10,] 0.5890  0 247.90
## [11,] 0.6516  0 255.73
## [12,] 0.7093  0 253.36
##
## $feval
## [1] 36
##
## $jeval
## [1] 19
##
## $coeffs
## [1] 102.4012  10.0000   0.2662
##
## $ssquares
## [1] 1143
##
```

```
cat("Try with scaled model\n")

## Try with scaled model

escal <- y ~ 100 * b1/(1 + 10 * b2 * exp(-0.1 * b3 * tt))
suneasy <- c(b1 = 200, b2 = 50, b3 = 0.3)
ssceasy <- c(b1 = 2, b2 = 5, b3 = 3)
st1scal <- c(b1 = 100, b2 = 10, b3 = 0.1)


cat("EASY start -- unscaled")

## EASY start -- unscaled

anls01 <- try(nls(eunsc, start = suneasy, trace = traceval, data = weeddata1))
print(anls01)

## 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
##
## Number of iterations to convergence: 4
## Achieved convergence tolerance: 1.92e-07
```

```
anlmrt01 <- try(nlxb(eunsc, start = ssceasy, trace = traceval, data = weeddata1))
print(anlmrt01)

## $resid
##  [1] -11.4817  31.9040  28.2676  25.0343  20.8313  14.7083   6.4573
##  [8]  -0.6577 -12.2557 -25.0477 -38.0947 -54.0717
##
## $jacobian
##            b1         b2         b3
##  [1,] -0.1629 -4.475e-03 -7.178e+00
##  [2,]  1.0328 -8.009e-04 -2.569e+00
##  [3,]  1.0001 -3.343e-06 -1.609e-02
##  [4,]  1.0000 -1.487e-08 -9.543e-05
##  [5,]  1.0000 -6.620e-11 -5.309e-07
##  [6,]  1.0000 -2.946e-13 -2.836e-09
##  [7,]  1.0000 -1.311e-15 -1.472e-11
##  [8,]  1.0000 -5.837e-18 -7.490e-14
##  [9,]  1.0000 -2.598e-20 -3.750e-16
## [10,]  1.0000 -1.156e-22 -1.855e-18
## [11,]  1.0000 -5.146e-25 -9.080e-21
## [12,]  1.0000 -2.290e-27 -4.409e-23
##
## $feval
## [1] 6996
##
## $jeval
## [1] 5001
##
## $coeffs
## [1]    37.900 -1604.128     5.415
##
## $ssquares
## [1] 8420
##


cat("All 1s start -- unscaled")

## All 1s start -- unscaled

anls02 <- try(nls(eunsc, start = start1, trace = traceval, data = weeddata1))
if (class(anls02) == "try-error") {
    cat("FAILED:")
    print(anls02)
} else {
    print(anls02)
}

## FAILED:[1] "Error in nls(eunsc, start = start1, trace = traceval, data = weeddata1) : \n  singular gradient\n"
## attr(,"class")
## [1] "try-error"
## attr(,"condition")
## <simpleError in nls(eunsc, start = start1, trace = traceval, data = weeddata1): singular gradient>

anlmrt02 <- nlxb(eunsc, start = start1, trace = traceval, data = weeddata1)
print(anlmrt02)

## $resid
##  [1]  0.01190 -0.03276  0.09203  0.20878  0.39263 -0.05759 -1.10573
##  [8]  0.71579 -0.10765 -0.34840  0.65259 -0.28757
##
## $jacobian
##            b1      b2      b3
##  [1,] 0.02712 -0.1054   5.176
##  [2,] 0.03674 -0.1414  13.885
##  [3,] 0.04960 -0.1884  27.742
##  [4,] 0.06664 -0.2486  48.814
##  [5,] 0.08901 -0.3240  79.537
##  [6,] 0.11792 -0.4157 122.438
##  [7,] 0.15464 -0.5224 179.522
##  [8,] 0.20019 -0.6399 251.294
##  [9,] 0.25511 -0.7594 335.526
## [10,] 0.31908 -0.8683 426.252
## [11,] 0.39069 -0.9513 513.725
## [12,] 0.46733 -0.9948 586.047
##
## $feval
## [1] 36
##
## $jeval
## [1] 22
##
## $coeffs
## [1] 196.1863  49.0916   0.3136
##
## $ssquares
## [1] 2.587
##
```

```
cat("ones start -- scaled")
```

```
## ones start -- scaled
```

```
anls03 <- try(nls(escal, start = start1, trace = traceval, data = weeddata1))
print(anls03)
```

```
## [1] "Error in nls(escal, start = start1, trace = traceval, data = weeddata1) : \n  singular gradient\n"
## attr(,"class")
## [1] "try-error"
## attr(,"condition")
## <simpleError in nls(escal, start = start1, trace = traceval, data = weeddata1): singular gradient>
```

```
anlmrt03 <- nlxb(escal, start = start1, trace = traceval, data = weeddata1)
print(anlmrt03)
```

```
## $resid
##  [1]  0.01190 -0.03276  0.09203  0.20878  0.39263 -0.05759 -1.10573
##  [8]  0.71579 -0.10765 -0.34840  0.65259 -0.28757
##
## $jacobian
##           b1     b2      b3
##  [1,]  2.712 -1.054  0.5176
##  [2,]  3.674 -1.414  1.3885
##  [3,]  4.960 -1.884  2.7742
##  [4,]  6.664 -2.486  4.8814
##  [5,]  8.901 -3.240  7.9537
##  [6,] 11.792 -4.157 12.2438
##  [7,] 15.464 -5.224 17.9522
##  [8,] 20.019 -6.399 25.1294
##  [9,] 25.511 -7.594 33.5526
## [10,] 31.908 -8.683 42.6252
## [11,] 39.069 -9.513 51.3725
## [12,] 46.733 -9.948 58.6047
##
## $feval
## [1] 26
##
## $jeval
## [1] 13
##
## $coeffs
## [1] 1.962 4.909 3.136
```

```
##
## $ssquares
## [1] 2.587
##
```

```
cat("HARD start -- scaled")
```

```
## HARD start -- scaled
```

```
anls04 <- try(nls(escal, start = st1scal, trace = traceval, data = weeddata1))
print(anls04)
```

```
## [1] "Error in nls(escal, start = st1scal, trace = traceval, data = weeddata1) : \n  singular gradient\n"
## attr(,"class")
## [1] "try-error"
## attr(,"condition")
## <simpleError in nls(escal, start = st1scal, trace = traceval, data = weeddata1): singular gradient>
```

```
anlmrt04 <- nlxb(escal, start = st1scal, trace = traceval, data = weeddata1)
print(anlmrt04)
```

```
## $resid
##  [1]  0.01190 -0.03276  0.09203  0.20878  0.39263 -0.05759 -1.10573
##  [8]  0.71579 -0.10765 -0.34840  0.65259 -0.28757
##
## $jacobian
##           b1     b2      b3
##  [1,]  2.712 -1.054  0.5176
##  [2,]  3.674 -1.414  1.3885
##  [3,]  4.960 -1.884  2.7742
##  [4,]  6.664 -2.486  4.8814
##  [5,]  8.901 -3.240  7.9537
##  [6,] 11.792 -4.157 12.2438
##  [7,] 15.464 -5.224 17.9522
##  [8,] 20.019 -6.399 25.1294
##  [9,] 25.511 -7.594 33.5526
## [10,] 31.908 -8.683 42.6252
## [11,] 39.069 -9.513 51.3725
## [12,] 46.733 -9.948 58.6047
##
## $feval
## [1] 36
##
## $jeval
## [1] 28
##
## $coeffs
## [1] 1.962 4.909 3.136
##
## $ssquares
## [1] 2.587
##
```

```
cat("EASY start -- scaled")
```

```
## EASY start -- scaled
```

```
anls05 <- try(nls(escal, start = ssceasy, trace = traceval, data = weeddata1))
print(anls05)
```

```
## Nonlinear regression model
##   model:  y ~ 100 * b1/(1 + 10 * b2 * exp(-0.1 * b3 * tt))
##    data:  weeddata1
##   b1   b2   b3
## 1.96 4.91 3.14
##  residual sum-of-squares: 2.59
##
## Number of iterations to convergence: 4
## Achieved convergence tolerance: 2.34e-07

anlmrt05 <- nlxb(escal, start = ssceasy, trace = traceval, data = weeddata1)
print(anlmrt03)

## $resid
##  [1]  0.01190 -0.03276  0.09203  0.20878  0.39263 -0.05759 -1.10573
##  [8]  0.71579 -0.10765 -0.34840  0.65259 -0.28757
##
## $jacobian
##           b1     b2      b3
##  [1,]  2.712 -1.054  0.5176
##  [2,]  3.674 -1.414  1.3885
##  [3,]  4.960 -1.884  2.7742
##  [4,]  6.664 -2.486  4.8814
##  [5,]  8.901 -3.240  7.9537
##  [6,] 11.792 -4.157 12.2438
##  [7,] 15.464 -5.224 17.9522
##  [8,] 20.019 -6.399 25.1294
##  [9,] 25.511 -7.594 33.5526
## [10,] 31.908 -8.683 42.6252
## [11,] 39.069 -9.513 51.3725
## [12,] 46.733 -9.948 58.6047
##
## $feval
## [1] 26
##
## $jeval
## [1] 13
##
## $coeffs
## [1] 1.962 4.909 3.136
##
## $ssquares
## [1] 2.587
##
```

5.2 Problems using an objective or residual function – nlfb()

```
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 * 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, 12, 3)
    tt <- 1:12
    yy <- exp(-0.1 * x[3] * tt)
    zz <- 100/(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)
}

cat("try nlfb\n")

## try nlfb

st <- c(b1 = 1, b2 = 1, b3 = 1)
low <- -Inf
up <- Inf

ans1 <- nlfb(st, shobbs.res, shobbs.jac, trace = traceval)
ans1

## $resid
##  [1]  0.01190 -0.03276  0.09203  0.20878  0.39263 -0.05759 -1.10573
##  [8]  0.71579 -0.10765 -0.34840  0.65259 -0.28757
##
## $jacobian
##         [,1]   [,2]    [,3]
##  [1,]  2.712 -1.054  0.5176
##  [2,]  3.674 -1.414  1.3885
##  [3,]  4.960 -1.884  2.7742
##  [4,]  6.664 -2.486  4.8814
##  [5,]  8.901 -3.240  7.9537
##  [6,] 11.792 -4.157 12.2438
##  [7,] 15.464 -5.224 17.9522
##  [8,] 20.019 -6.399 25.1294
##  [9,] 25.511 -7.594 33.5526
## [10,] 31.908 -8.683 42.6252
## [11,] 39.069 -9.513 51.3725
## [12,] 46.733 -9.948 58.6047
##
## $feval
## [1] 24
##
## $jeval
## [1] 15
##
## $coeffs
## [1] 1.962 4.909 3.136
##
## $ssquares
## [1] 2.587
##

cat("No jacobian function -- use internal approximation\n")

## No jacobian function -- use internal approximation

ans1n <- nlfb(st, shobbs.res, trace = TRUE, control = list(watch = TRUE))  # NO jacfn

## lower:[1] -Inf -Inf -Inf
## upper:[1] Inf Inf Inf
## Using default jacobian approximation
## Start:lamda: 1e-04  SS= 10685  at  b1 = 1  b2 = 1  b3 = 1  1 / 0
## Continue
## bdmsk:[1] 1 1 1
## JJ
##          [,1]     [,2]    [,3]
##  [1,]  9.9519  -8.9615  0.8961
##  [2,] 10.8846  -9.6998  1.9400
##  [3,] 11.8932 -10.4787  3.1436
##  [4,] 12.9816 -11.2964  4.5186
##  [5,] 14.1537 -12.1504  6.0752
##  [6,] 15.4128 -13.0373  7.8224
##  [7,] 16.7621 -13.9524  9.7667
##  [8,] 18.2040 -14.8902 11.9121
##  [9,] 19.7406 -15.8437 14.2593
## [10,] 21.3730 -16.8050 16.8050
## [11,] 23.1016 -17.7647 19.5412
## [12,] 24.9256 -18.7127 22.4553
## [13,]  0.5983   0.0000  0.0000
## [14,]  0.0000   0.4842  0.0000
## [15,]  0.0000   0.0000  0.4174
## [16,]  0.0100   0.0000  0.0000
## [17,]  0.0000   0.0100  0.0000
## [18,]  0.0000   0.0000  0.0100
## gradient projection =  -10578  g-delta-angle= 97.58
## delta:[1] 6.138 8.116 2.769
## Stepsize= 1
## lamda: 0.001  SS= 177800  at  b1 = 7.138  b2 = 9.116  b3 = 3.769  2 / 1
## Cycle
## JJ
##           [,1]      [,2]     [,3]
##  [1,]  9.95186  -8.96146  0.89615
##  [2,] 10.88458  -9.69984  1.93997
##  [3,] 11.89318 -10.47870  3.14361
##  [4,] 12.98162 -11.29639  4.51856
##  [5,] 14.15367 -12.15040  6.07520
##  [6,] 15.41279 -13.03725  7.82235
##  [7,] 16.76206 -13.95239  9.76668
##  [8,] 18.20403 -14.89016 11.91213
##  [9,] 19.74062 -15.84370 14.25933
## [10,] 21.37303 -16.80496 16.80496
## [11,] 23.10157 -17.76474 19.54122
## [12,] 24.92558 -18.71274 22.45528
## [13,]  1.89187   0.00000  0.00000
## [14,]  0.00000   1.53133  0.00000
## [15,]  0.00000   0.00000  1.31988
## [16,]  0.03162   0.00000  0.00000
## [17,]  0.00000   0.03162  0.00000
## [18,]  0.00000   0.00000  0.03162
## gradient projection =  -10516  g-delta-angle= 108.1
## delta:[1] 0.4903 2.0720 3.9249
## Stepsize= 1
## lamda: 0.01  SS= 19087  at  b1 = 1.49  b2 = 3.072  b3 = 4.925  3 / 1
## Cycle
## JJ
##         [,1]    [,2]    [,3]
##  [1,]  9.952  -8.961  0.8961
##  [2,] 10.885  -9.700  1.9400
##  [3,] 11.893 -10.479  3.1436
##  [4,] 12.982 -11.296  4.5186
##  [5,] 14.154 -12.150  6.0752
##  [6,] 15.413 -13.037  7.8224
##  [7,] 16.762 -13.952  9.7667
##  [8,] 18.204 -14.890 11.9121
##  [9,] 19.741 -15.844 14.2593
## [10,] 21.373 -16.805 16.8050
## [11,] 23.102 -17.765 19.5412
## [12,] 24.926 -18.713 22.4553
## [13,]  5.983   0.000  0.0000
## [14,]  0.000   4.842  0.0000
## [15,]  0.000   0.000  4.1738
## [16,]  0.100   0.000  0.0000
## [17,]  0.000   0.100  0.0000
## [18,]  0.000   0.000  0.1000
## gradient projection =  -10241  g-delta-angle= 110.1
## delta:[1] -0.2051  1.0743  3.7610
## Stepsize= 1
## <<lamda: 0.004  SS= 1274  at  b1 = 0.7949  b2 = 2.074  b3 = 4.761  4 / 1
## Cycle
## bdmsk:[1] 1 1 1
## JJ
##           [,1]     [,2]     [,3]
##  [1,]  7.20181 -2.56102  0.53122
##  [2,] 11.10561 -3.78310  1.56943
##  [3,] 16.74370 -5.34195  3.32418
##  [4,] 24.45663 -7.07985  5.87419
##  [5,] 34.26045 -8.63080  8.95127
##  [6,] 45.62093 -9.50664 11.83156
##  [7,] 57.45607 -9.36709 13.60086
##  [8,] 68.49425 -8.26942 13.72236
##  [9,] 77.77628 -6.62362 12.36521
## [10,] 84.92553 -4.90582 10.17595
## [11,] 90.06859 -3.42780  7.82117
## [12,] 93.58940 -2.29909  5.72270
## [13,] 13.02831  0.00000  0.00000
## [14,]  0.00000  1.42197  0.00000
## [15,]  0.00000  0.00000  1.99205
## [16,]  0.06325  0.00000  0.00000
## [17,]  0.00000  0.06325  0.00000
## [18,]  0.00000  0.00000  0.06325
## gradient projection =  -1158  g-delta-angle= 100.8
## delta:[1]  0.2676 -0.1075 -2.4141
## Stepsize= 1
## lamda: 0.04  SS= 4265  at  b1 = 1.062  b2 = 1.967  b3 = 2.347  5 / 2
## Cycle
## JJ
##         [,1]   [,2]    [,3]
##  [1,]  7.202 -2.561  0.5312
##  [2,] 11.106 -3.783  1.5694
##  [3,] 16.744 -5.342  3.3242
##  [4,] 24.457 -7.080  5.8742
##  [5,] 34.260 -8.631  8.9513
##  [6,] 45.621 -9.507 11.8316
##  [7,] 57.456 -9.367 13.6009
##  [8,] 68.494 -8.269 13.7224
##  [9,] 77.776 -6.624 12.3652
## [10,] 84.926 -4.906 10.1760
## [11,] 90.069 -3.428  7.8212
## [12,] 93.589 -2.299  5.7227
## [13,] 41.199  0.000  0.0000
## [14,]  0.000  4.497  0.0000
## [15,]  0.000  0.000  6.2994
## [16,]  0.200  0.000  0.0000
## [17,]  0.000  0.200  0.0000
## [18,]  0.000  0.000  0.2000
## gradient projection =  -981.6  g-delta-angle= 105.5
## delta:[1]  0.1714  0.7143 -1.2522
## Stepsize= 1
## <<lamda: 0.016  SS= 946.6  at  b1 = 0.9663  b2 = 2.789  b3 = 3.509  6 / 2
## Cycle
## bdmsk:[1] 1 1 1
## JJ
##          [,1]    [,2]    [,3]
##  [1,]  4.8465 -1.5980  0.4456
##  [2,]  6.7461 -2.1800  1.2158
##  [3,]  9.3174 -2.9279  2.4494
##  [4,] 12.7349 -3.8510  4.2955
##  [5,] 17.1686 -4.9279  6.8709
##  [6,] 22.7436 -6.0887 10.1873
##  [7,] 29.4845 -7.2046 14.0635
##  [8,] 37.2598 -8.1006 18.0714
##  [9,] 45.7550 -8.6006 21.5853
## [10,] 54.5045 -8.5928 23.9617
## [11,] 62.9844 -8.0789 24.7815
## [12,] 70.7325 -7.1736 24.0050
## [13,] 16.6959  0.0000  0.0000
## [14,]  0.0000  2.7543  0.0000
## [15,]  0.0000  0.0000  6.8491
## [16,]  0.1265  0.0000  0.0000
## [17,]  0.0000  0.1265  0.0000
## [18,]  0.0000  0.0000  0.1265
## gradient projection =  -862.5  g-delta-angle= 99.78
## delta:[1] 0.37510 1.21512 0.04998
## Stepsize= 1
## <<lamda: 0.0064  SS= 43.28  at  b1 = 1.341  b2 = 4.004  b3 = 3.559  7 / 3
## Cycle
## bdmsk:[1] 1 1 1
## JJ
##         [,1]   [,2]    [,3]
##  [1,]  3.443 -1.114  0.4459
##  [2,]  4.843 -1.544  1.2363
##  [3,]  6.772 -2.115  2.5408
##  [4,]  9.395 -2.852  4.5675
##  [5,] 12.893 -3.763  7.5326
##  [6,] 17.443 -4.825 11.5901
##  [7,] 23.171 -5.964 16.7160
##  [8,] 30.095 -7.049 22.5762
##  [9,] 38.062 -7.899 28.4611
## [10,] 46.729 -8.340 33.3916
## [11,] 55.598 -8.271 36.4263
## [12,] 64.123 -7.708 37.0313
## [13,]  9.092  0.000  0.0000
## [14,]  0.000  1.597  0.0000
## [15,]  0.000  0.000  6.0040
## [16,]  0.080  0.000  0.0000
## [17,]  0.000  0.080  0.0000
## [18,]  0.000  0.000  0.0800
## gradient projection =  -29.81  g-delta-angle= 96.73
## delta:[1]  0.2217  0.4813 -0.1692
## Stepsize= 1
## <<lamda: 0.00256  SS= 17.39  at  b1 = 1.563  b2 = 4.485  b3 = 3.39  8 / 4
## Cycle
## bdmsk:[1] 1 1 1
## JJ
##          [,1]    [,2]    [,3]
##  [1,]  3.0343 -1.0254  0.4599
##  [2,]  4.2071 -1.4045  1.2599
##  [3,]  5.8060 -1.9060  2.5645
##  [4,]  7.9621 -2.5540  4.5818
##  [5,] 10.8268 -3.3648  7.5456
##  [6,] 14.5591 -4.3353 11.6665
##  [7,] 19.2997 -5.4281 17.0417
##  [8,] 25.1299 -6.5572 23.5276
##  [9,] 32.0224 -7.5865 30.6232
## [10,] 39.8004 -8.3503 37.4515
## [11,] 48.1300 -8.7007 42.9252
## [12,] 56.5649 -8.5627 46.0847
## [13,]  4.9530  0.0000  0.0000
## [14,]  0.0000  1.0018  0.0000
## [15,]  0.0000  0.0000  4.3446
## [16,]  0.0506  0.0000  0.0000
## [17,]  0.0000  0.0506  0.0000
## [18,]  0.0000  0.0000  0.0506
## gradient projection =  -12.51  g-delta-angle= 95.82
## delta:[1]  0.1959  0.2088 -0.1423
## Stepsize= 1
## <<lamda: 0.001024  SS= 6.637  at  b1 = 1.759  b2 = 4.694  b3 = 3.247  9 / 5
## Cycle
## bdmsk:[1] 1 1 1
## JJ
##         [,1]    [,2]    [,3]
##  [1,]  2.863 -1.0423  0.4892
##  [2,]  3.919 -1.4110  1.3246
##  [3,]  5.342 -1.8949  2.6684
##  [4,]  7.243 -2.5177  4.7270
##  [5,]  9.751 -3.2977  7.7395
##  [6,] 13.005 -4.2398 11.9405
##  [7,] 17.139 -5.3220 17.4865
##  [8,] 22.252 -6.4832 24.3447
##  [9,] 28.367 -7.6148 32.1683
## [10,] 35.397 -8.5695 40.2237
## [11,] 43.121 -9.1913 47.4566
## [12,] 51.195 -9.3632 52.7394
## [13,]  2.807  0.0000  0.0000
## [14,]  0.000  0.6525  0.0000
## [15,]  0.000  0.0000  3.0051
## [16,]  0.032  0.0000  0.0000
## [17,]  0.000  0.0320  0.0000
## [18,]  0.000  0.0000  0.0320
## gradient projection =  -3.759  g-delta-angle= 94.64
## delta:[1]  0.13563  0.13826 -0.07907
## Stepsize= 1
## <<lamda: 0.0004096  SS= 3.072  at  b1 = 1.895  b2 = 4.832  b3 = 3.168  10 / 6
## Cycle
## bdmsk:[1] 1 1 1
## JJ
##           [,1]     [,2]     [,3]
##  [1,]  2.76242 -1.05319  0.50891
##  [2,]  3.75345 -1.41645  1.36888
##  [3,]  5.08145 -1.89114  2.74145
##  [4,]  6.84588 -2.50044  4.83295
##  [5,]  9.16383 -3.26378  7.88544
##  [6,] 12.16412 -4.18926 12.14575
##  [7,] 15.97399 -5.26274 17.80104
##  [8,] 20.69596 -6.43525 24.87661
##  [9,] 26.37562 -7.61394 33.11213
## [10,] 32.96603 -8.66456 41.86797
## [11,] 40.30175 -9.43345 50.14164
## [12,] 48.09858 -9.78805 56.75617
## [13,]  1.66030  0.00000  0.00000
## [14,]  0.00000  0.41875  0.00000
## [15,]  0.00000  0.00000  1.99975
## [16,]  0.02024  0.00000  0.00000
## [17,]  0.00000  0.02024  0.00000
## [18,]  0.00000  0.00000  0.02024
## gradient projection =  -0.4697  g-delta-angle= 93.78
## delta:[1]  0.05569  0.06363 -0.02689
## Stepsize= 1
## <<lamda: 0.0001638  SS= 2.598  at  b1 = 1.95  b2 = 4.896  b3 = 3.141  11 / 7
## Cycle
## bdmsk:[1] 1 1 1
## JJ
##          [,1]    [,2]    [,3]
##  [1,]  2.7204 -1.0542  0.5161
##  [2,]  3.6873 -1.4148  1.3852
##  [3,]  4.9804 -1.8852  2.7689
##  [4,]  6.6954 -2.4887  4.8735
##  [5,]  8.9454 -3.2448  7.9428
##  [6,] 11.8554 -4.1629 12.2283
##  [7,] 15.5504 -5.2315 17.9283
##  [8,] 20.1340 -6.4058 25.0890
##  [9,] 25.6581 -7.5988 33.4813
## [10,] 32.0889 -8.6812 42.5008
## [11,] 39.2799 -9.5014 51.1678
## [12,] 46.9678 -9.9226 58.2938
## [13,]  1.0236  0.0000  0.0000
## [14,]  0.0000  0.2657  0.0000
## [15,]  0.0000  0.0000  1.2890
## [16,]  0.0128  0.0000  0.0000
## [17,]  0.0000  0.0128  0.0000
## [18,]  0.0000  0.0000  0.0128
## gradient projection =  -0.01017  g-delta-angle= 93.01
## delta:[1]  0.010674  0.012430 -0.005095
## Stepsize= 1
## <<lamda: 6.554e-05  SS= 2.587  at  b1 = 1.961  b2 = 4.908  b3 = 3.136  12 / 8
## Cycle
## bdmsk:[1] 1 1 1
## JJ
##             [,1]      [,2]      [,3]
##  [1,]  2.712325 -1.054273  0.517453
##  [2,]  3.674734 -1.414229  1.388250
##  [3,]  4.961219 -1.883835  2.773844
##  [4,]  6.666916 -2.486075  4.880813
##  [5,]  8.904099 -3.240727  7.952990
##  [6,] 11.797114 -4.157308 12.242818
##  [7,] 15.470462 -5.224749 17.950701
##  [8,] 20.027889 -6.399225 25.126691
##  [9,] 25.522513 -7.594550 33.547684
## [10,] 31.922847 -8.682737 42.616186
## [11,] 39.085887 -9.512431 51.357296
## [12,] 46.752399 -9.946209 58.580996
## [13,]  0.644225  0.000000  0.000000
## [14,]  0.000000  0.168135  0.000000
## [15,]  0.000000  0.000000  0.818052
## [16,]  0.008095  0.000000  0.000000
## [17,]  0.000000  0.008095  0.000000
## [18,]  0.000000  0.000000  0.008095
## gradient projection =  -2.606e-05  g-delta-angle= 92.72
## delta:[1]  0.0008584  0.0009765 -0.0004482
## Stepsize= 1
## <<lamda: 2.621e-05  SS= 2.587  at  b1 = 1.962  b2 = 4.909  b3 = 3.136  13 / 9
## Cycle
## bdmsk:[1] 1 1 1
## JJ
##            [,1]     [,2]     [,3]
##  [1,]  2.71168 -1.05428  0.51756
##  [2,]  3.67371 -1.41419  1.38849
##  [3,]  4.95965 -1.88372  2.77423
##  [4,]  6.66456 -2.48585  4.88135
##  [5,]  8.90067 -3.24037  7.95371
##  [6,] 11.79225 -4.15681 12.24380
##  [7,] 15.46376 -5.22415 17.95220
##  [8,] 20.01896 -6.39861 25.12929
##  [9,] 25.51107 -7.59412 33.55247
## [10,] 31.90879 -8.68278 42.62486
## [11,] 39.06941 -9.51326 51.37201
## [12,] 46.73406 -9.94811 58.60383
## [13,]  0.40727  0.00000  0.00000
## [14,]  0.00000  0.10634  0.00000
## [15,]  0.00000  0.00000  0.51752
## [16,]  0.00512  0.00000  0.00000
## [17,]  0.00000  0.00512  0.00000
## [18,]  0.00000  0.00000  0.00512
## gradient projection =  -2.219e-08  g-delta-angle= 99.5
## delta:[1]  3.091e-05  3.591e-05 -1.687e-05
## Stepsize= 1
## <<lamda: 1.049e-05  SS= 2.587  at  b1 = 1.962  b2 = 4.909  b3 = 3.136  14 / 10
## Cycle
## bdmsk:[1] 1 1 1
## JJ
##             [,1]      [,2]      [,3]
##  [1,]  2.711658 -1.054282  0.517564
##  [2,]  3.673675 -1.414187  1.388495
##  [3,]  4.959589 -1.883714  2.774238
##  [4,]  6.664475 -2.485844  4.881367
##  [5,]  8.900541 -3.240359  7.953727
##  [6,] 11.792066 -4.156794 12.243830
##  [7,] 15.463509 -5.224121 17.952247
##  [8,] 20.018628 -6.398588 25.129372
##  [9,] 25.510639 -7.594104 33.552631
## [10,] 31.908260 -8.682774 42.625162
## [11,] 39.068799 -9.513291 51.372531
## [12,] 46.733373 -9.948172 58.604645
## [13,]  0.257578  0.000000  0.000000
## [14,]  0.000000  0.067256  0.000000
## [15,]  0.000000  0.000000  0.327312
## [16,]  0.003238  0.000000  0.000000
## [17,]  0.000000  0.003238  0.000000
## [18,]  0.000000  0.000000  0.003238
## gradient projection =  -8.01e-12  g-delta-angle= 153.1
## delta:[1]  5.841e-07  7.483e-07 -3.083e-07
## Stepsize= 1
## <<lamda: 4.194e-06  SS= 2.587  at  b1 = 1.962  b2 = 4.909  b3 = 3.136  15 / 11
## Cycle
## bdmsk:[1] 1 1 1
## JJ
##            [,1]      [,2]      [,3]
##  [1,]  2.711658 -1.054282  0.517564
##  [2,]  3.673674 -1.414186  1.388495
##  [3,]  4.959588 -1.883714  2.774238
##  [4,]  6.664474 -2.485844  4.881367
##  [5,]  8.900539 -3.240359  7.953728
##  [6,] 11.792062 -4.156793 12.243830
##  [7,] 15.463505 -5.224121 17.952248
##  [8,] 20.018622 -6.398587 25.129374
##  [9,] 25.510631 -7.594104 33.552634
## [10,] 31.908251 -8.682774 42.625168
## [11,] 39.068787 -9.513291 51.372540
## [12,] 46.733360 -9.948173 58.604660
## [13,]  0.162907  0.000000  0.000000
## [14,]  0.000000  0.042536  0.000000
## [15,]  0.000000  0.000000  0.207010
## [16,]  0.002048  0.000000  0.000000
## [17,]  0.000000  0.002048  0.000000
## [18,]  0.000000  0.000000  0.002048
## gradient projection =  -1.963e-15  g-delta-angle= 172.3
## delta:[1]  8.547e-09  1.308e-08 -4.083e-09
## Stepsize= 1
## <<lamda: 1.678e-06  SS= 2.587  at  b1 = 1.962  b2 = 4.909  b3 = 3.136  16 / 12
## Cycle
## bdmsk:[1] 1 1 1
## JJ
##            [,1]      [,2]      [,3]
##  [1,]  2.711658 -1.054282  0.517564
##  [2,]  3.673674 -1.414186  1.388495
##  [3,]  4.959588 -1.883714  2.774238
##  [4,]  6.664474 -2.485844  4.881367
##  [5,]  8.900539 -3.240359  7.953728
##  [6,] 11.792062 -4.156793 12.243830
##  [7,] 15.463505 -5.224121 17.952248
##  [8,] 20.018622 -6.398587 25.129374
##  [9,] 25.510631 -7.594104 33.552634
## [10,] 31.908250 -8.682774 42.625168
## [11,] 39.068787 -9.513291 51.372541
## [12,] 46.733360 -9.948173 58.604660
## [13,]  0.103031  0.000000  0.000000
## [14,]  0.000000  0.026902  0.000000
## [15,]  0.000000  0.000000  0.130925
## [16,]  0.001295  0.000000  0.000000
## [17,]  0.000000  0.001295  0.000000
## [18,]  0.000000  0.000000  0.001295
## gradient projection =  -3.592e-17  g-delta-angle= 143.3
## delta:[1]  5.187e-10 -6.323e-10 -5.376e-10
## Stepsize= 1
## lamda: 1.678e-05  SS= 2.587  at  b1 = 1.962  b2 = 4.909  b3 = 3.136  17 / 13
## Cycle
## JJ
##            [,1]      [,2]      [,3]
##  [1,]  2.711658 -1.054282  0.517564
##  [2,]  3.673674 -1.414186  1.388495
##  [3,]  4.959588 -1.883714  2.774238
##  [4,]  6.664474 -2.485844  4.881367
##  [5,]  8.900539 -3.240359  7.953728
##  [6,] 11.792062 -4.156793 12.243830
##  [7,] 15.463505 -5.224121 17.952248
##  [8,] 20.018622 -6.398587 25.129374
##  [9,] 25.510631 -7.594104 33.552634
## [10,] 31.908250 -8.682774 42.625168
## [11,] 39.068787 -9.513291 51.372541
## [12,] 46.733360 -9.948173 58.604660
## [13,]  0.325814  0.000000  0.000000
## [14,]  0.000000  0.085073  0.000000
## [15,]  0.000000  0.000000  0.414020
## [16,]  0.004096  0.000000  0.000000
## [17,]  0.000000  0.004096  0.000000
## [18,]  0.000000  0.000000  0.004096
## gradient projection =  -3.585e-17  g-delta-angle= 143.2
## delta:[1]  5.147e-10 -6.357e-10 -5.351e-10
## Stepsize= 1
## lamda: 0.0001678  SS= 2.587  at  b1 = 1.962  b2 = 4.909  b3 = 3.136  18 / 13
## Cycle
## JJ
##           [,1]     [,2]     [,3]
##  [1,]  2.71166 -1.05428  0.51756
##  [2,]  3.67367 -1.41419  1.38849
##  [3,]  4.95959 -1.88371  2.77424
##  [4,]  6.66447 -2.48584  4.88137
##  [5,]  8.90054 -3.24036  7.95373
##  [6,] 11.79206 -4.15679 12.24383
##  [7,] 15.46350 -5.22412 17.95225
##  [8,] 20.01862 -6.39859 25.12937
##  [9,] 25.51063 -7.59410 33.55263
## [10,] 31.90825 -8.68277 42.62517
## [11,] 39.06879 -9.51329 51.37254
## [12,] 46.73336 -9.94817 58.60466
## [13,]  1.03031  0.00000  0.00000
## [14,]  0.00000  0.26902  0.00000
## [15,]  0.00000  0.00000  1.30925
## [16,]  0.01295  0.00000  0.00000
## [17,]  0.00000  0.01295  0.00000
## [18,]  0.00000  0.00000  0.01295
## gradient projection =  -3.517e-17  g-delta-angle= 142.6
## delta:[1]  4.768e-10 -6.668e-10 -5.115e-10
## Stepsize= 1
## lamda: 0.001678  SS= 2.587  at  b1 = 1.962  b2 = 4.909  b3 = 3.136  19 / 13
## Cycle
## JJ
##           [,1]     [,2]     [,3]
##  [1,]  2.71166 -1.05428  0.51756
##  [2,]  3.67367 -1.41419  1.38849
##  [3,]  4.95959 -1.88371  2.77424
##  [4,]  6.66447 -2.48584  4.88137
##  [5,]  8.90054 -3.24036  7.95373
##  [6,] 11.79206 -4.15679 12.24383
##  [7,] 15.46350 -5.22412 17.95225
##  [8,] 20.01862 -6.39859 25.12937
##  [9,] 25.51063 -7.59410 33.55263
## [10,] 31.90825 -8.68277 42.62517
## [11,] 39.06879 -9.51329 51.37254
## [12,] 46.73336 -9.94817 58.60466
## [13,]  3.25814  0.00000  0.00000
## [14,]  0.00000  0.85073  0.00000
## [15,]  0.00000  0.00000  4.14020
## [16,]  0.04096  0.00000  0.00000
## [17,]  0.00000  0.04096  0.00000
## [18,]  0.00000  0.00000  0.04096
## gradient projection =  -3.067e-17  g-delta-angle= 135.6
## delta:[1]  2.621e-10 -8.189e-10 -3.732e-10
## Stepsize= 1
## lamda: 0.01678  SS= 2.587  at  b1 = 1.962  b2 = 4.909  b3 = 3.136  20 / 13
## Cycle
## JJ
##          [,1]    [,2]    [,3]
##  [1,]  2.7117 -1.0543  0.5176
##  [2,]  3.6737 -1.4142  1.3885
##  [3,]  4.9596 -1.8837  2.7742
##  [4,]  6.6645 -2.4858  4.8814
##  [5,]  8.9005 -3.2404  7.9537
##  [6,] 11.7921 -4.1568 12.2438
##  [7,] 15.4635 -5.2241 17.9522
##  [8,] 20.0186 -6.3986 25.1294
##  [9,] 25.5106 -7.5941 33.5526
## [10,] 31.9083 -8.6828 42.6252
## [11,] 39.0688 -9.5133 51.3725
## [12,] 46.7334 -9.9482 58.6047
## [13,] 10.3031  0.0000  0.0000
## [14,]  0.0000  2.6902  0.0000
## [15,]  0.0000  0.0000 13.0925
## [16,]  0.1295  0.0000  0.0000
## [17,]  0.0000  0.1295  0.0000
## [18,]  0.0000  0.0000  0.1295
## gradient projection =  -1.782e-17  g-delta-angle= 123.6
## delta:[1]  1.698e-11 -6.853e-10 -1.523e-10
## Stepsize= 1
## lamda: 0.1678  SS= 2.587  at  b1 = 1.962  b2 = 4.909  b3 = 3.136  21 / 13
## Cycle
## JJ
##          [,1]    [,2]    [,3]
##  [1,]  2.7117 -1.0543  0.5176
##  [2,]  3.6737 -1.4142  1.3885
##  [3,]  4.9596 -1.8837  2.7742
##  [4,]  6.6645 -2.4858  4.8814
##  [5,]  8.9005 -3.2404  7.9537
##  [6,] 11.7921 -4.1568 12.2438
##  [7,] 15.4635 -5.2241 17.9522
##  [8,] 20.0186 -6.3986 25.1294
##  [9,] 25.5106 -7.5941 33.5526
## [10,] 31.9083 -8.6828 42.6252
## [11,] 39.0688 -9.5133 51.3725
## [12,] 46.7334 -9.9482 58.6047
## [13,] 32.5814  0.0000  0.0000
## [14,]  0.0000  8.5073  0.0000
## [15,]  0.0000  0.0000 41.4020
## [16,]  0.4096  0.0000  0.0000
## [17,]  0.0000  0.4096  0.0000
## [18,]  0.0000  0.0000  0.4096
## gradient projection =  -3.863e-18  g-delta-angle= 120.9
## delta:[1] -3.569e-12 -1.615e-10 -2.888e-11
## Stepsize= 1
## <<lamda: 0.06711  SS= 2.587  at  b1 = 1.962  b2 = 4.909  b3 = 3.136  22 / 13
## Cycle
## bdmsk:[1] 1 1 1
## JJ
##          [,1]    [,2]    [,3]
##  [1,]  2.7117 -1.0543  0.5176
##  [2,]  3.6737 -1.4142  1.3885
##  [3,]  4.9596 -1.8837  2.7742
##  [4,]  6.6645 -2.4858  4.8814
##  [5,]  8.9005 -3.2404  7.9537
##  [6,] 11.7921 -4.1568 12.2438
##  [7,] 15.4635 -5.2241 17.9522
##  [8,] 20.0186 -6.3986 25.1294
##  [9,] 25.5106 -7.5941 33.5526
## [10,] 31.9083 -8.6828 42.6252
## [11,] 39.0688 -9.5133 51.3725
## [12,] 46.7334 -9.9482 58.6047
## [13,] 20.6063  0.0000  0.0000
## [14,]  0.0000  5.3805  0.0000
## [15,]  0.0000  0.0000 26.1849
## [16,]  0.2591  0.0000  0.0000
## [17,]  0.0000  0.2591  0.0000
## [18,]  0.0000  0.0000  0.2591
## gradient projection =  -1.139e-17  g-delta-angle= 123.1
## delta:[1] -5.784e-11 -4.397e-10 -4.341e-11
## Stepsize= 1
## lamda: 0.6711  SS= 2.587  at  b1 = 1.962  b2 = 4.909  b3 = 3.136  23 / 14
## Cycle
## JJ
##          [,1]    [,2]    [,3]
##  [1,]  2.7117 -1.0543  0.5176
##  [2,]  3.6737 -1.4142  1.3885
##  [3,]  4.9596 -1.8837  2.7742
##  [4,]  6.6645 -2.4858  4.8814
##  [5,]  8.9005 -3.2404  7.9537
##  [6,] 11.7921 -4.1568 12.2438
##  [7,] 15.4635 -5.2241 17.9522
##  [8,] 20.0186 -6.3986 25.1294
##  [9,] 25.5106 -7.5941 33.5526
## [10,] 31.9083 -8.6828 42.6252
## [11,] 39.0688 -9.5133 51.3725
## [12,] 46.7334 -9.9482 58.6047
## [13,] 65.1627  0.0000  0.0000
## [14,]  0.0000 17.0146  0.0000
## [15,]  0.0000  0.0000 82.8040
## [16,]  0.8192  0.0000  0.0000
## [17,]  0.0000  0.8192  0.0000
## [18,]  0.0000  0.0000  0.8192
## gradient projection =  -1.513e-18  g-delta-angle= 122.7
## delta:[1] -6.579e-12 -5.905e-11 -6.120e-12
## Stepsize= 1
## lamda: 6.711  SS= 2.587  at  b1 = 1.962  b2 = 4.909  b3 = 3.136  24 / 14
## Cycle
## JJ
##          [,1]   [,2]     [,3]
##  [1,]   2.712 -1.054   0.5176
##  [2,]   3.674 -1.414   1.3885
##  [3,]   4.960 -1.884   2.7742
##  [4,]   6.664 -2.486   4.8814
##  [5,]   8.901 -3.240   7.9537
##  [6,]  11.792 -4.157  12.2438
##  [7,]  15.464 -5.224  17.9522
##  [8,]  20.019 -6.399  25.1294
##  [9,]  25.511 -7.594  33.5526
## [10,]  31.908 -8.683  42.6252
## [11,]  39.069 -9.513  51.3725
## [12,]  46.733 -9.948  58.6047
## [13,] 206.063  0.000   0.0000
## [14,]   0.000 53.805   0.0000
## [15,]   0.000  0.000 261.8493
## [16,]   2.591  0.000   0.0000
## [17,]   0.000  2.591   0.0000
## [18,]   0.000  0.000   2.5905
## gradient projection =  -1.585e-19  g-delta-angle= 121.4
## delta:[1] -5.818e-13 -6.444e-12 -5.687e-13
## Stepsize= 1
## lamda: 67.11  SS= 2.587  at  b1 = 1.962  b2 = 4.909  b3 = 3.136  25 / 14
## Cycle
## JJ
##          [,1]    [,2]     [,3]
##  [1,]   2.712  -1.054   0.5176
##  [2,]   3.674  -1.414   1.3885
##  [3,]   4.960  -1.884   2.7742
##  [4,]   6.664  -2.486   4.8814
##  [5,]   8.901  -3.240   7.9537
##  [6,]  11.792  -4.157  12.2438
##  [7,]  15.464  -5.224  17.9522
##  [8,]  20.019  -6.399  25.1294
##  [9,]  25.511  -7.594  33.5526
## [10,]  31.908  -8.683  42.6252
## [11,]  39.069  -9.513  51.3725
## [12,]  46.733  -9.948  58.6047
## [13,] 651.627   0.000   0.0000
## [14,]   0.000 170.146   0.0000
## [15,]   0.000   0.000 828.0402
## [16,]   8.192   0.000   0.0000
## [17,]   0.000   8.192   0.0000
## [18,]   0.000   0.000   8.1920
## gradient projection =  -1.601e-20  g-delta-angle= 120.7
## delta:[1] -5.361e-14 -6.643e-13 -5.346e-14
## Stepsize= 1
## lamda: 671.1  SS= 2.587  at  b1 = 1.962  b2 = 4.909  b3 = 3.136  26 / 14
## Cycle
## JJ
##           [,1]    [,2]      [,3]
##  [1,]    2.712  -1.054    0.5176
##  [2,]    3.674  -1.414    1.3885
##  [3,]    4.960  -1.884    2.7742
##  [4,]    6.664  -2.486    4.8814
##  [5,]    8.901  -3.240    7.9537
##  [6,]   11.792  -4.157   12.2438
##  [7,]   15.464  -5.224   17.9522
##  [8,]   20.019  -6.399   25.1294
##  [9,]   25.511  -7.594   33.5526
## [10,]   31.908  -8.683   42.6252
## [11,]   39.069  -9.513   51.3725
## [12,]   46.733  -9.948   58.6047
## [13,] 2060.627   0.000    0.0000
## [14,]    0.000 538.047    0.0000
## [15,]    0.000   0.000 2618.4931
## [16,]   25.905   0.000    0.0000
## [17,]    0.000  25.905    0.0000
## [18,]    0.000   0.000   25.9054
## gradient projection =  -1.603e-21  g-delta-angle= 120.6
## delta:[1] -5.295e-15 -6.671e-14 -5.296e-15
## Stepsize= 1
## <<lamda: 268.4  SS= 2.587  at  b1 = 1.962  b2 = 4.909  b3 = 3.136  27 / 14
## Cycle
## bdmsk:[1] 1 1 1
## JJ
##           [,1]    [,2]      [,3]
##  [1,]    2.712  -1.054    0.5176
##  [2,]    3.674  -1.414    1.3885
##  [3,]    4.960  -1.884    2.7742
##  [4,]    6.664  -2.486    4.8814
##  [5,]    8.901  -3.240    7.9537
##  [6,]   11.792  -4.157   12.2438
##  [7,]   15.464  -5.224   17.9522
##  [8,]   20.019  -6.399   25.1294
##  [9,]   25.511  -7.594   33.5526
## [10,]   31.908  -8.683   42.6252
## [11,]   39.069  -9.513   51.3725
## [12,]   46.733  -9.948   58.6047
## [13,] 1303.255   0.000    0.0000
## [14,]    0.000 340.291    0.0000
## [15,]    0.000   0.000 1656.0804
## [16,]   16.384   0.000    0.0000
## [17,]    0.000  16.384    0.0000
## [18,]    0.000   0.000   16.3840
## gradient projection =  -1.067e-20  g-delta-angle= 119.2
## delta:[1]  2.158e-14 -2.670e-13 -2.402e-14
## Stepsize= 1
## lamda: 2684  SS= 2.587  at  b1 = 1.962  b2 = 4.909  b3 = 3.136  28 / 15
## Cycle
## JJ
##           [,1]     [,2]      [,3]
##  [1,]    2.712   -1.054    0.5176
##  [2,]    3.674   -1.414    1.3885
##  [3,]    4.960   -1.884    2.7742
##  [4,]    6.664   -2.486    4.8814
##  [5,]    8.901   -3.240    7.9537
##  [6,]   11.792   -4.157   12.2438
##  [7,]   15.464   -5.224   17.9522
##  [8,]   20.019   -6.399   25.1294
##  [9,]   25.511   -7.594   33.5526
## [10,]   31.908   -8.683   42.6252
## [11,]   39.069   -9.513   51.3725
## [12,]   46.733   -9.948   58.6047
## [13,] 4121.253    0.000    0.0000
## [14,]    0.000 1076.095    0.0000
## [15,]    0.000    0.000 5236.9862
## [16,]   51.811    0.000    0.0000
## [17,]    0.000   51.811    0.0000
## [18,]    0.000    0.000   51.8108
## gradient projection =  -1.07e-21  g-delta-angle= 119.2
## delta:[1]  2.178e-15 -2.678e-14 -2.387e-15
## Stepsize= 1
## lamda: 26844  SS= 2.587  at  b1 = 1.962  b2 = 4.909  b3 = 3.136  29 / 15
## Cycle
## JJ
##            [,1]     [,2]      [,3]
##  [1,]     2.712   -1.054 5.176e-01
##  [2,]     3.674   -1.414 1.388e+00
##  [3,]     4.960   -1.884 2.774e+00
##  [4,]     6.664   -2.486 4.881e+00
##  [5,]     8.901   -3.240 7.954e+00
##  [6,]    11.792   -4.157 1.224e+01
##  [7,]    15.464   -5.224 1.795e+01
##  [8,]    20.019   -6.399 2.513e+01
##  [9,]    25.511   -7.594 3.355e+01
## [10,]    31.908   -8.683 4.263e+01
## [11,]    39.069   -9.513 5.137e+01
## [12,]    46.733   -9.948 5.860e+01
## [13,] 13032.548    0.000 0.000e+00
## [14,]     0.000 3402.911 0.000e+00
## [15,]     0.000    0.000 1.656e+04
## [16,]   163.840    0.000 0.000e+00
## [17,]     0.000  163.840 0.000e+00
## [18,]     0.000    0.000 1.638e+02
## gradient projection =  -1.07e-22  g-delta-angle= 119.2
## delta:[1]  2.180e-16 -2.679e-15 -2.385e-16
## Stepsize= 1
## No parameter change
## Cycle
```

```
ans1n
```

```
## $resid
##  [1]  0.01190 -0.03276  0.09203  0.20878  0.39263 -0.05759 -1.10573
##  [8]  0.71579 -0.10765 -0.34840  0.65259 -0.28757
##
## $jacobian
##         [,1]   [,2]    [,3]
##  [1,]  2.712 -1.054  0.5176
##  [2,]  3.674 -1.414  1.3885
##  [3,]  4.960 -1.884  2.7742
##  [4,]  6.664 -2.486  4.8814
##  [5,]  8.901 -3.240  7.9537
##  [6,] 11.792 -4.157 12.2438
##  [7,] 15.464 -5.224 17.9522
##  [8,] 20.019 -6.399 25.1294
##  [9,] 25.511 -7.594 33.5526
## [10,] 31.908 -8.683 42.6252
## [11,] 39.069 -9.513 51.3725
## [12,] 46.733 -9.948 58.6047
##
## $feval
## [1] 29
##
## $jeval
## [1] 15
##
## $coeffs
## [1] 1.962 4.909 3.136
##
## $ssquares
## [1] 2.587
##


# tmp <- readline('Try with bounds at 2')
time2 <- system.time(ans2 <- nlfb(st, shobbs.res, shobbs.jac, upper = c(2, 2,
    2), trace = traceval))

## Warning:  NaNs produced

## Warning:  NaNs produced

## Warning:  NaNs produced

## Warning:  NaNs produced

## Warning:  NaNs produced

## Warning:  NaNs produced

ans2

## $resid
##  [1]   7.4916   8.1751   8.8741   9.2907   9.3453   8.1540   5.5591
##  [8]   4.8580   0.4407  -4.4269  -8.8661 -15.6521
##
## $jacobian
##       [,1]    [,2] [,3]
##  [1,]    0  -6.707    0
##  [2,]    0  -7.964    0
##  [3,]    0  -9.404    0
##  [4,]    0 -11.029    0
##  [5,]    0 -12.834    0
##  [6,]    0 -14.797    0
##  [7,]    0 -16.881    0
##  [8,]    0 -19.028    0
##  [9,]    0 -21.158    0
## [10,]    0 -23.174    0
## [11,]    0 -24.966    0
## [12,]    0 -26.420    0
##
## $feval
## [1] 20
##
## $jeval
## [1] 11
##
## $coeffs
## [1] 2.000 1.786 2.000
##
## $ssquares
## [1] 839.7
##

time2

##    user  system elapsed
##   0.008   0.000   0.010
```

```
cat("BOUNDS")

## BOUNDS

st2s <- c(b1 = 1, b2 = 1, b3 = 1)
an1qb1 <- try(nlxb(escal, start = st2s, trace = traceval, data = weeddata1,
    lower = c(0, 0, 0), upper = c(2, 6, 3), control = list(watch = FALSE)))
print(an1qb1)

## $resid
##  [1]  0.6018  0.6557  0.8749  1.0687  1.2932  0.8231 -0.3330  1.2687
##  [9]  0.1030 -0.5880 -0.0972 -1.5286
##
## $jacobian
##       b1      b2 b3
##  [1,]  0  -1.294  0
##  [2,]  0  -1.711  0
##  [3,]  0  -2.247  0
##  [4,]  0  -2.924  0
##  [5,]  0  -3.762  0
##  [6,]  0  -4.767  0
##  [7,]  0  -5.926  0
##  [8,]  0  -7.195  0
##  [9,]  0  -8.488  0
## [10,]  0  -9.681  0
## [11,]  0 -10.623  0
## [12,]  0 -11.175  0
##
## $feval
## [1] 25
##
## $jeval
## [1] 17
##
## $coeffs
## [1] 2.000 4.433 3.000
##
## $ssquares
## [1] 9.473
##

tmp <- readline("next")

## next
```

```
ans2 <- nlfb(st2s, shobbs.res, shobbs.jac, lower = c(0, 0, 0), upper = c(2,
    6, 3), trace = traceval, control = list(watch = FALSE))

## Warning:  NaNs produced

print(ans2)

## $resid
##  [1]  0.6018  0.6557  0.8749  1.0687  1.2932  0.8231 -0.3330  1.2687
##  [9]  0.1030 -0.5880 -0.0972 -1.5286
##
## $jacobian
##       [,1]    [,2] [,3]
##  [1,]    0  -1.294    0
##  [2,]    0  -1.711    0
##  [3,]    0  -2.247    0
##  [4,]    0  -2.924    0
##  [5,]    0  -3.762    0
##  [6,]    0  -4.767    0
##  [7,]    0  -5.926    0
##  [8,]    0  -7.195    0
##  [9,]    0  -8.488    0
## [10,]    0  -9.681    0
## [11,]    0 -10.623    0
## [12,]    0 -11.175    0
##
## $feval
## [1] 18
##
## $jeval
## [1] 18
##
## $coeffs
## [1] 2.000 4.433 3.000
##
## $ssquares
## [1] 9.473
##


cat("BUT ... nls() seems to do better from the TRACE information\n")

## BUT ... nls() seems to do better from the TRACE information

anlsb <- nls(escal, start = st2s, trace = traceval, data = weeddata1, lower = c(0,
    0, 0), upper = c(2, 6, 3), algorithm = "port")
cat("However, let us check the answer\n")

## However, let us check the answer

print(anlsb)

## Nonlinear regression model
##   model:  y ~ 100 * b1/(1 + 10 * b2 * exp(-0.1 * b3 * tt))
##    data:  weeddata1
##   b1   b2   b3
## 2.00 4.43 3.00
##  residual sum-of-squares: 9.47
##
## Algorithm "port", convergence message: both X-convergence and relative convergence (5)

cat("BUT...crossprod(resid(anlsb))=", crossprod(resid(anlsb)), "\n")

## BUT...crossprod(resid(anlsb))= 9.473
```

```
cat("Try wrapnls\n")

## Try wrapnls

traceval <- FALSE
# 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)  # for testing
tdat <- seq_along(ydat)  # for testing
start1 <- c(b1 = 1, b2 = 1, b3 = 1)
escal <- y ~ 100 * b1/(1 + 10 * b2 * exp(-0.1 * b3 * tt))
up1 <- c(2, 6, 3)
up2 <- c(1, 5, 9)
## weeddata1 <- data.frame(y=ydat, tt=tdat)
an1w <- try(wrapnls(escal, start = start1, trace = traceval, data = weeddata1))
print(an1w)

## Nonlinear regression model
##   model:  y ~ 100 * b1/(1 + 10 * b2 * exp(-0.1 * b3 * tt))
##    data:  data
##   b1   b2   b3
## 1.96 4.91 3.14
##  residual sum-of-squares: 2.59
##
## Number of iterations to convergence: 0
## Achieved convergence tolerance: 3.03e-08


cat("BOUNDED wrapnls\n")

## BOUNDED wrapnls

an1wb <- try(wrapnls(escal, start = start1, trace = traceval, data = weeddata1,
    upper = up1))
print(an1wb)

## Nonlinear regression model
##   model:  y ~ 100 * b1/(1 + 10 * b2 * exp(-0.1 * b3 * tt))
##    data:  data
##   b1   b2   b3
## 2.00 4.43 3.00
##  residual sum-of-squares: 9.47
##
## Algorithm "port", convergence message: both X-convergence and relative convergence (5)


cat("BOUNDED wrapnls\n")

## BOUNDED wrapnls

an2wb <- try(wrapnls(escal, start = start1, trace = traceval, data = weeddata1,
    upper = up2))

## Warning:  NaNs produced

## Warning:  NaNs produced

print(an2wb)

## Nonlinear regression model
##   model:  y ~ 100 * b1/(1 + 10 * b2 * exp(-0.1 * b3 * tt))
##    data:  data
##   b1   b2   b3
## 1.00 5.00 4.53
##  residual sum-of-squares: 160
##
## Algorithm "port", convergence message: both X-convergence and relative convergence (5)
```

```
cat("TRY MASKS ONLY\n")

## TRY MASKS ONLY

an1xm3 <- try(nlxb(escal, start1, trace = traceval, data = weeddata1, masked = c("b3")))
print(an1xm3)

## $resid
##  [1]  16.569  16.938  17.083  16.665  15.568  12.878   8.420   5.498
##  [9]  -1.467  -9.138 -16.526 -26.249
##
## $jacobian
##              b1         b2 b3
##  [1,] 1.403e-12 -2.777e-12  0
##  [2,] 1.550e-12 -3.069e-12  0
##  [3,] 1.713e-12 -3.392e-12  0
##  [4,] 1.894e-12 -3.749e-12  0
##  [5,] 2.093e-12 -4.143e-12  0
##  [6,] 2.313e-12 -4.579e-12  0
##  [7,] 2.556e-12 -5.060e-12  0
##  [8,] 2.825e-12 -5.592e-12  0
##  [9,] 3.122e-12 -6.180e-12  0
## [10,] 3.450e-12 -6.830e-12  0
## [11,] 3.813e-12 -7.549e-12  0
## [12,] 4.214e-12 -8.343e-12  0
##
## $feval
## [1] 82
##
## $jeval
## [1] 82
##
## $coeffs
## [1] 1.559e+13 7.878e+12 1.000e+00
##
## $ssquares
## [1] 2688
##

an1fm3 <- try(nlfb(start1, shobbs.res, shobbs.jac, trace = traceval, data = weeddata1,
    maskidx = c(3)))
print(an1fm3)

## [1] "Error in resfn(pnum, ...) : \n  unused argument(s) (data = list(y = c(5.308, 7.24, 9.638, 12.866, 17.069, 23.192, 31.443,
## attr(,"class")
## [1] "try-error"
## attr(,"condition")
## <simpleError in resfn(pnum, ...): unused argument(s) (data = list(y = c(5.308, 7.24, 9.638, 12.866, 17.069, 23.192, 31.443, 38


an1xm1 <- try(nlxb(escal, start1, trace = traceval, data = weeddata1, masked = c("b1")))
print(an1xm1)

## $resid
##  [1] -2.5652 -2.9375 -2.9500 -2.6115 -1.6602 -0.6877  0.2019  3.9057
##  [9]  3.9002  2.2778 -1.0562 -9.3119
##
## $jacobian
##       b1      b2      b3
##  [1,]  0 -0.4719  0.2668
##  [2,]  0 -0.7284  0.8235
##  [3,]  0 -1.1040  1.8722
##  [4,]  0 -1.6280  3.6812
##  [5,]  0 -2.3058  6.5173
##  [6,]  0 -3.0851 10.4639
##  [7,]  0 -3.8265 15.1416
##  [8,]  0 -4.3220 19.5456
##  [9,]  0 -4.3934 22.3519
## [10,]  0 -4.0124 22.6818
## [11,]  0 -3.3223 20.6586
## [12,]  0 -2.5355 17.1998
##
## $feval
## [1] 30
##
## $jeval
## [1] 15
##
## $coeffs
## [1] 1.000 5.653 4.664
##
## $ssquares
## [1] 157.5
##

an1fm1 <- try(nlfb(start1, shobbs.res, shobbs.jac, trace = traceval, data = weeddata1,
    maskidx = c(1)))
print(an1fm1)

## [1] "Error in resfn(pnum, ...) : \n  unused argument(s) (data = list(y = c(5.308, 7.24, 9.638, 12.866, 17.069, 23.192, 31.443,
## attr(,"class")
## [1] "try-error"
## attr(,"condition")
## <simpleError in resfn(pnum, ...): unused argument(s) (data = list(y = c(5.308, 7.24, 9.638, 12.866, 17.069, 23.192, 31.443, 38


# Need to check when all parameters masked.??
```

5.3 Transforming a model formula to objective function form

```
cat("\n\n Now check conversion of expression to function\n\n")

##
##
##  Now check conversion of expression to function
##

cat("K Vandepoel function\n")

## K Vandepoel function


x <- c(1, 3, 5, 7)  # data
y <- c(37.98, 11.68, 3.65, 3.93)
penetrationks28 <- data.frame(x = x, y = y)

cat("Try nls() -- note the try() function!\n")

## Try nls() -- note the try() function!


fit0 <- try(nls(y ~ (a + b * exp(1)^(-c * x)), data = penetrationks28, start = c(a = 0,
    b = 1, c = 1), trace = TRUE))

## 1579 :  0 1 1
```

```
print(fit0)

## [1] "Error in nls(y ~ (a + b * exp(1)^(-c * x)), data = penetrationks28, start = c(a = 0,  : \n  singular gradient\n"
## attr(,"class")
## [1] "try-error"
## attr(,"condition")
## <simpleError in nls(y ~ (a + b * exp(1)^(-c * x)), data = penetrationks28, start = c(a = 0,     b = 1, c = 1), trace = TRUE):


cat("\n\n")
```

```

fit1 <- nlxb(y ~ (a + b * exp(-c * x)), data = penetrationks28, start = c(a = 0,
    b = 1, c = 1), trace = TRUE)

## formula: y ~ (a + b * exp(-c * x))
## <environment: 0xa247b14>
## lower:[1] -Inf -Inf -Inf
## upper:[1] Inf Inf Inf
## $watch
## [1] FALSE
##
## $phi
## [1] 1
##
## $lamda
## [1] 1e-04
##
## $offset
## [1] 100
##
## $laminc
## [1] 10
##
## $lamdec
## [1] 4
##
## $femax
## [1] 10000
##
## $jemax
## [1] 5000
##
## [[9]]
## [1] 1e+60
##
## Data variable  y :[1] 37.98 11.68  3.65  3.93
## Data variable  x :[1] 1 3 5 7
## Start:lamda: 1e-04  SS= 1579  at  a = 0  b = 1  c = 1  1 / 0
## gradient projection =  -1577  g-delta-angle= 112.5
## Stepsize= 1
## lamda: 0.001  SS= 9.826e+231  at  a = 2.899  b = 56.76  c = -37.58  2 / 1
## gradient projection =  -1572  g-delta-angle= 112.7
## Stepsize= 1
## lamda: 0.01  SS= 1.283e+241  at  a = 2.892  b = 54.83  c = -39.08  3 / 1
## gradient projection =  -1528  g-delta-angle= 113.5
## Stepsize= 1
## lamda: 0.1  SS= 1.404e+249  at  a = 3.329  b = 48.6  c = -40.42  4 / 1
## gradient projection =  -1245  g-delta-angle= 118.9
## Stepsize= 1
## lamda: 1  SS= 8.042e+171  at  a = 6.167  b = 31.62  c = -27.78  5 / 1
## gradient projection =  -565.8  g-delta-angle= 134.3
## Stepsize= 1
## lamda: 10  SS= 9.502e+48  at  a = 5.368  b = 9.554  c = -7.733  6 / 1
## gradient projection =  -95.87  g-delta-angle= 141
## Stepsize= 1
## <<lamda: 4  SS= 1376  at  a = 1.03  b = 2.2  c = -0.2749  7 / 1
## gradient projection =  -59.98  g-delta-angle= 92.23
## Stepsize= 1
## <<lamda: 1.6  SS= 1268  at  a = 2.151  b = 2.07  c = -0.2533  8 / 2
## gradient projection =  -112.5  g-delta-angle= 92.57
## Stepsize= 1
## <<lamda: 0.64  SS= 1079  at  a = 4.791  b = 1.87  c = -0.2019  9 / 3
## gradient projection =  -175  g-delta-angle= 93.96
## Stepsize= 1
## <<lamda: 0.256  SS= 837.4  at  a = 9.704  b = 1.669  c = -0.05151  10 / 4
## gradient projection =  -231.4  g-delta-angle= 102.1
## Stepsize= 1
## <<lamda: 0.1024  SS= 719.2  at  a = 15.3  b = 4.14  c = 0.8473  11 / 5
## gradient projection =  -491  g-delta-angle= 133.9
## Stepsize= 1
## lamda: 1.024  SS= 1.432e+55  at  a = 5.228  b = 22.59  c = -8.626  12 / 6
## gradient projection =  -148.2  g-delta-angle= 141.4
## Stepsize= 1
## lamda: 10.24  SS= 5.482e+16  at  a = 13.22  b = 8.947  c = -2.44  13 / 6
## gradient projection =  -20.88  g-delta-angle= 144.4
## Stepsize= 1
## <<lamda: 4.096  SS= 692.7  at  a = 15.15  b = 4.795  c = 0.3465  14 / 6
## gradient projection =  -22.16  g-delta-angle= 108.9
## Stepsize= 1
## <<lamda: 1.638  SS= 656.5  at  a = 14.78  b = 6.127  c = 0.4925  15 / 7
## gradient projection =  -43.22  g-delta-angle= 139.5
## Stepsize= 1
## <<lamda: 0.6554  SS= 575  at  a = 13.79  b = 9.802  c = 0.4238  16 / 8
## gradient projection =  -84.48  g-delta-angle= 110
## Stepsize= 1
## <<lamda: 0.2621  SS= 431.6  at  a = 11.95  b = 17.41  c = 0.5834  17 / 9
## gradient projection =  -144.9  g-delta-angle= 98.94
## Stepsize= 1
## <<lamda: 0.1049  SS= 406.7  at  a = 7.507  b = 30.26  c = 0.241  18 / 10
## gradient projection =  -288.1  g-delta-angle= 91.29
## Stepsize= 1
## <<lamda: 0.04194  SS= 113  at  a = 6.129  b = 40.36  c = 0.4866  19 / 11
## gradient projection =  -78.42  g-delta-angle= 93.44
## Stepsize= 1
## <<lamda: 0.01678  SS= 17.08  at  a = 2.907  b = 57.94  c = 0.6089  20 / 12
## gradient projection =  -12.36  g-delta-angle= 91.09
## Stepsize= 1
## <<lamda: 0.006711  SS= 2.875  at  a = 2.343  b = 67.18  c = 0.6543  21 / 13
## gradient projection =  -0.7631  g-delta-angle= 91.16
## Stepsize= 1
## <<lamda: 0.002684  SS= 2.008  at  a = 2.608  b = 70.69  c = 0.6949  22 / 14
## gradient projection =  -0.03482  g-delta-angle= 92.15
## Stepsize= 1
## <<lamda: 0.001074  SS= 1.971  at  a = 2.667  b = 71.6  c = 0.7059  23 / 15
## gradient projection =  -0.0002533  g-delta-angle= 91.72
## Stepsize= 1
## <<lamda: 0.0004295  SS= 1.971  at  a = 2.672  b = 71.68  c = 0.7069  24 / 16
## gradient projection =  -9.099e-08  g-delta-angle= 91.2
## Stepsize= 1
## <<lamda: 0.0001718  SS= 1.971  at  a = 2.672  b = 71.68  c = 0.7068  25 / 17
## gradient projection =  -1.552e-11  g-delta-angle= 91.37
## Stepsize= 1
## <<lamda: 6.872e-05  SS= 1.971  at  a = 2.672  b = 71.68  c = 0.7068  26 / 18
## gradient projection =  -5.459e-14  g-delta-angle= 90.93
## Stepsize= 1
## <<lamda: 2.749e-05  SS= 1.971  at  a = 2.672  b = 71.68  c = 0.7068  27 / 19
## gradient projection =  -2.425e-16  g-delta-angle= 90.97
## Stepsize= 1
## <<lamda: 1.1e-05  SS= 1.971  at  a = 2.672  b = 71.68  c = 0.7068  28 / 20
## gradient projection =  -1.146e-18  g-delta-angle= 90.99
## Stepsize= 1
## lamda: 0.00011  SS= 1.971  at  a = 2.672  b = 71.68  c = 0.7068  29 / 21
## gradient projection =  -1.14e-18  g-delta-angle= 90.99
## Stepsize= 1
## lamda: 0.0011  SS= 1.971  at  a = 2.672  b = 71.68  c = 0.7068  30 / 21
## gradient projection =  -1.089e-18  g-delta-angle= 91
## Stepsize= 1
## lamda: 0.011  SS= 1.971  at  a = 2.672  b = 71.68  c = 0.7068  31 / 21
## gradient projection =  -7.612e-19  g-delta-angle= 91.06
## Stepsize= 1
## lamda: 0.11  SS= 1.971  at  a = 2.672  b = 71.68  c = 0.7068  32 / 21
## gradient projection =  -2.294e-19  g-delta-angle= 91.63
## Stepsize= 1
## lamda: 1.1  SS= 1.971  at  a = 2.672  b = 71.68  c = 0.7068  33 / 21
## gradient projection =  -4.484e-20  g-delta-angle= 94.81
## Stepsize= 1
## lamda: 11  SS= 1.971  at  a = 2.672  b = 71.68  c = 0.7068  34 / 21
## gradient projection =  -6.539e-21  g-delta-angle= 121.8
## Stepsize= 1
## <<lamda: 4.398  SS= 1.971  at  a = 2.672  b = 71.68  c = 0.7068  35 / 21
## gradient projection =  -1.29e-20  g-delta-angle= 99.74
## Stepsize= 1
## lamda: 43.98  SS= 1.971  at  a = 2.672  b = 71.68  c = 0.7068  36 / 22
## gradient projection =  -1.482e-21  g-delta-angle= 115.5
## Stepsize= 1
## <<lamda: 17.59  SS= 1.971  at  a = 2.672  b = 71.68  c = 0.7068  37 / 22
## gradient projection =  -3.472e-21  g-delta-angle= 106.9
## Stepsize= 1
## lamda: 175.9  SS= 1.971  at  a = 2.672  b = 71.68  c = 0.7068  38 / 23
## gradient projection =  -3.616e-22  g-delta-angle= 114.3
## Stepsize= 1
## lamda: 1759  SS= 1.971  at  a = 2.672  b = 71.68  c = 0.7068  39 / 23
## gradient projection =  -3.631e-23  g-delta-angle= 115.4
## Stepsize= 1
## No parameter change

print(fit1)

## $resid
## [1]  0.04433 -0.40881  1.11369 -0.74921
##
## $jacobian
##      a        b       c
## [1,] 1 0.493196 -35.352
## [2,] 1 0.119966 -25.798
## [3,] 1 0.029181 -10.458
## [4,] 1 0.007098  -3.562
##
## $feval
## [1] 39
##
## $jeval
## [1] 23
##
## $coeffs
## [1]  2.6720 71.6800  0.7068
##
## $ssquares
## [1] 1.971
##


mexprn <- "y ~ (a+b*exp(-c*x))"
pvec <- c(a = 0, b = 1, c = 1)
bnew <- c(a = 10, b = 3, c = 4)

k.r <- model2resfun(mexprn, pvec)
k.j <- model2jacfun(mexprn, pvec)
k.f <- model2ssfun(mexprn, pvec)
```

```
k.g <- model2grfun(mexprn, pvec)


cat("At pvec:")

## At pvec:

print(pvec)

## a b c
## 0 1 1

rp <- k.r(pvec, x = x, y = y)
cat(" rp=")

##  rp=

print(rp)

## [1] -37.612 -11.630  -3.643  -3.929

rf <- k.f(pvec, x = x, y = y)
cat(" rf=")

##  rf=

print(rf)

## [1] 1579

rj <- k.j(pvec, x = x, y = y)
cat(" rj=")

##  rj=

print(rj)

##      a         b         c
## [1,] 1 0.3678794 -0.367879
## [2,] 1 0.0497871 -0.149361
## [3,] 1 0.0067379 -0.033690
## [4,] 1 0.0009119 -0.006383

rg <- k.g(pvec, x = x, y = y)
cat(" rg=")

##  rg=

print(rg)

## [1] -113.63  -28.89   31.44

cat("modss at pvec gives ")

## modss at pvec gives

print(modss(pvec, k.r, x = x, y = y))

##      [,1]
## [1,] 1579

cat("modgr at pvec gives ")

## modgr at pvec gives

print(modgr(pvec, k.r, k.j, x = x, y = y))

## [1] -113.63  -28.89   31.44

cat("\n\n")
cat("At bnew:")

## At bnew:

print(bnew)

##  a  b  c
## 10  3  4

rb <- k.r(bnew, x = x, y = y)
cat(" rb=")

##  rb=

print(rb)

## [1] -27.93  -1.68   6.35   6.07

rf <- k.f(bnew, x = x, y = y)
cat(" rf=")

##  rf=

print(rf)

## [1] 859.8

rj <- k.j(bnew, x = x, y = y)
cat(" rj=")

##  rj=

print(rj)

##      a         b          c
## [1,] 1 1.832e-02 -5.495e-02
## [2,] 1 6.144e-06 -5.530e-05
## [3,] 1 2.061e-09 -3.092e-08
## [4,] 1 6.914e-13 -1.452e-11

rg <- k.g(bnew, x = x, y = y)
cat(" rg=")

##  rg=

print(rg)

## [1] -34.370  -1.023   3.069

cat("modss at bnew gives ")

## modss at bnew gives

print(modss(bnew, k.r, x = x, y = y))

##       [,1]
## [1,] 859.8

cat("modgr at bnew gives ")

## modgr at bnew gives

print(modgr(bnew, k.r, k.j, x = x, y = y))

## [1] -34.370  -1.023   3.069

cat("\n\n")
```

5.4 nlmrt TODOS

weightings (data or function call?? – try to match nls)
print method(s)
issue of character vs expression
return a class??
guessed starting values

6 minpack.lm

Package `minpack.lm` (?) provides for the minimization of nonlinear sums of squares expressed in residual function form. It is an interfacing of R to the Fortran software called **minpack** (?).

6.1 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`.

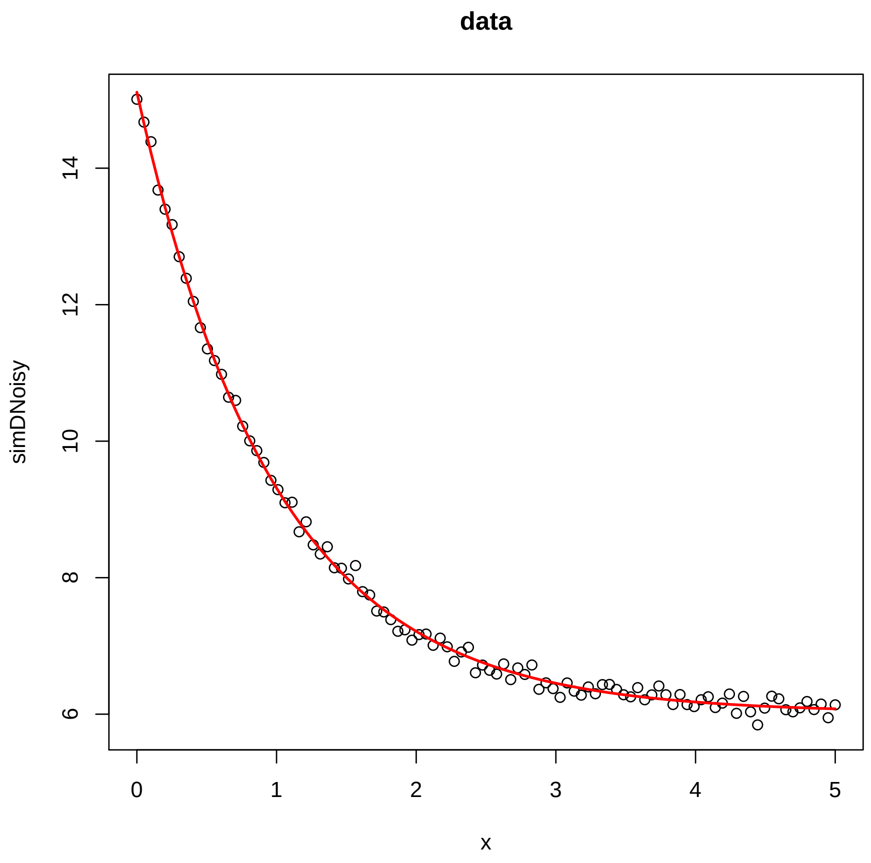
```
require(minpack.lm)
anlslm <- nls.lm(ones, lower = rep(-1000, 4), upper = rep(1000, 4), jres, jjac,
    yield = pastured$yield, time = pastured$time)
cat("anlslm from ones\n")

## anlslm from ones

print(strwrap(anlslm))

##  [1] "c(NaN, NaN, NaN, NaN)"
##  [2] "c(NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN, NaN,"
##  [3] "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 of the"
##  [7] "Jacobian is at most `gtol' in absolute value."
##  [8] "list(t1 = 3, t2 = 2.3723939879224e-11, t3 = 5.8039519205899e-10,"
##  [9] "t4 = 1.27525858056086e-09)"
## [10] "3"
## [11] "c(17533.3402000004, 16864.5616372991, NaN, 1.112549661455e-308)"
## [12] "NaN"

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

## anlslmh from huetstart

print(strwrap(anlslmh))
```

```
##  [1] "c(69.9551973916736, 61.6814877170941, -9.20891880263443,"
##  [2] "2.37781455978467)"
##  [3] "c(9, -4.54037977686007, 105.318033221555, 403.043210394647,"
##  [4] "-4.54037977686007, 3.51002837648689, -39.5314537948583,"
##  [5] "-137.559566823766, 105.318033221555, -39.5314537948583,"
##  [6] "1668.11894086464, 6495.67702199832, 403.043210394647,"
##  [7] "-137.559566823766, 6495.67702199832, 25481.4530263827)"
##  [8] "c(0.480682793156298, 0.669303022602289, -2.28431914156848,"
##  [9] "0.84375480165378, 0.734587578832198, 0.0665510313004845,"
## [10] "-0.985814877917491, -0.0250630130722556, 0.500317790294616)"
## [11] "1"
## [12] "Relative error in the sum of squares is at most `ftol'."
## [13] "list(t1 = 3, t2 = 2.35105755434962, t3 = 231.250186433367, t4 ="
## [14] "834.778914353853)"
## [15] "42"
## [16] "c(13386.9099465603, 13365.3097414383, 13351.1970260154,"
## [17] "13321.6478455192, 13260.1135652244, 13133.6391318145,"
## [18] "12877.8542053848, 12373.5432344283, 11428.8257706578,"
## [19] "9832.87890178625, 7138.12187613238, 3904.51162830831,"
## [20] "2286.64875980737, 1978.18149980306, 1620.89081508973,"
## [21] "1140.58638304326, 775.173148616759, 635.256627921485,"
## [22] "383.73614705125, 309.34124999335, 219.735856060243,"
## [23] "177.39873817915, 156.718991828473, 135.513594568191,"
## [24] "93.4016394568244, 72.8219383036213, 66.331560983492,"
## [25] "56.2809616213412, 54.9453021619837, 53.6227655715772,"
## [26] "51.9760950696957, 50.1418078879664, 48.130702164752,"
## [27] "44.7097757109316, 42.8838792615125, 32.3474231559281,"
## [28] "26.5253835687528, 15.3528215541113, 14.7215507012991,"
## [29] "8.37980617628204, 8.37589765770224, 8.37588365348112,"
## [30] "8.37588355972579)"
## [31] "8.37588355972579"
```

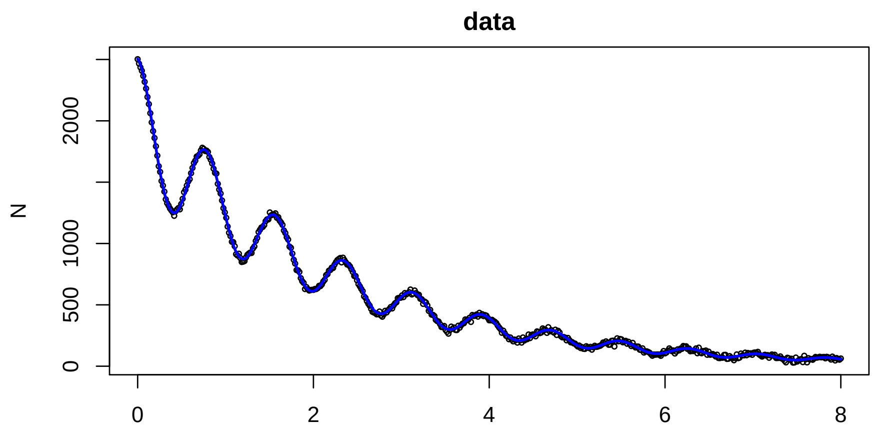
?? include minpack.lm failure example and explain why things go wrong

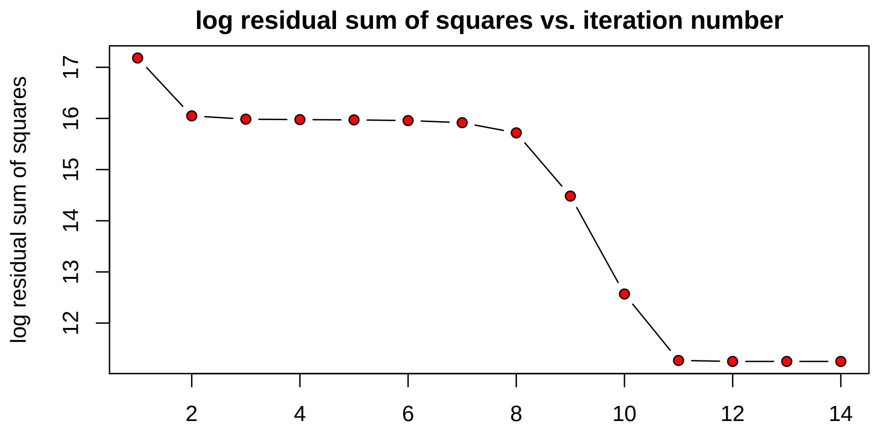
6.2 Examples `nls.lm`

```
###### example 1

## values over which to simulate data
x <- seq(0, 5, length = 100)

## model based on a list of parameters
getPred <- function(parS, xx) parS$a * exp(xx * parS$b) + parS$c

## parameter values used to simulate data
pp <- list(a = 9, b = -1, c = 6)

## simulated data, with noise
simDNoisy <- getPred(pp, x) + rnorm(length(x), sd = 0.1)

## plot data
plot(x, simDNoisy, main = "data")

## residual function
residFun <- function(p, observed, xx) observed - getPred(p, xx)

## starting values for parameters
parStart <- list(a = 3, b = -0.001, c = 1)

## perform fit
nls.out <- nls.lm(par = parStart, fn = residFun, observed = simDNoisy, xx = x,
    control = nls.lm.control(nprint = 1))

## It.    0, RSS =    1991.58, Par. =          3     -0.001          1
## It.    1, RSS =    336.515, Par. =    5.25336  -0.148765    3.23844
## It.    2, RSS =    107.539, Par. =    6.69971  -0.307776    4.28561
## It.    3, RSS =    55.0841, Par. =    7.49893  -0.426417      4.717
## It.    4, RSS =    33.3158, Par. =     7.6875  -0.719363    6.13311
## It.    5, RSS =    3.16022, Par. =    8.78829    -1.0265    6.22142
## It.    6, RSS =   0.999588, Par. =    9.09096   -1.01553    6.02157
## It.    7, RSS =   0.999502, Par. =    9.09134   -1.01604    6.02157
## It.    8, RSS =   0.999502, Par. =    9.09134   -1.01603    6.02157


## plot model evaluated at final parameter estimates
lines(x, getPred(as.list(coef(nls.out)), x), col = 2, lwd = 2)
```



```
## summary information on parameter estimates
summary(nls.out)

##
## Parameters:
##   Estimate Std. Error t value Pr(>|t|)
## a   9.0913     0.0439   206.9   <2e-16 ***
## b  -1.0160     0.0106   -95.4   <2e-16 ***
## c   6.0216     0.0194   310.3   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.102 on 97 degrees of freedom
## Number of iterations to termination: 8
## Reason for termination: Relative error in the sum of squares is at most `ftol'.


###### example 2

## function to simulate data
f <- function(TT, tau, N0, a, f0) {
    expr <- expression(N0 * exp(-TT/tau) * (1 + a * cos(f0 * TT)))
    eval(expr)
}

## helper function for an analytical gradient
j <- function(TT, tau, N0, a, f0) {
    expr <- expression(N0 * exp(-TT/tau) * (1 + a * cos(f0 * TT)))
    c(eval(D(expr, "tau")), eval(D(expr, "N0")), eval(D(expr, "a")), eval(D(expr,
        "f0")))
}

## values over which to simulate data
TT <- seq(0, 8, length = 501)

## parameter values underlying simulated data
p <- c(tau = 2.2, N0 = 1000, a = 0.25, f0 = 8)

## get data
Ndet <- do.call("f", c(list(TT = TT), as.list(p)))
## with noise
N <- Ndet + rnorm(length(Ndet), mean = Ndet, sd = 0.01 * max(Ndet))

## plot the data to fit
par(mfrow = c(2, 1), mar = c(3, 5, 2, 1))
plot(TT, N, bg = "black", cex = 0.5, main = "data")

## define a residual function
fcn <- function(p, TT, N, fcall, jcall) (N - do.call("fcall", c(list(TT = TT),
    as.list(p))))

## define analytical expression for the gradient
fcn.jac <- function(p, TT, N, fcall, jcall) -do.call("jcall", c(list(TT = TT),
    as.list(p)))

## starting values
guess <- c(tau = 2.2, N0 = 1500, a = 0.25, f0 = 10)

## to use an analytical expression for the gradient found in fcn.jac
## uncomment jac = fcn.jac
out <- nls.lm(par = guess, fn = fcn, jac = fcn.jac, fcall = f, jcall = j, TT = TT,
    N = N, control = nls.lm.control(nprint = 1))

## It.    0, RSS = 2.89615e+07, Par. =        2.2       1500       0.25         10
## It.    1, RSS = 9.34478e+06, Par. =    2.09435    2043.23 -0.0259052    9.92006
## It.    2, RSS = 8.75741e+06, Par. =    2.16282    2023.82  0.0496954    10.6429
## It.    3, RSS = 8.68497e+06, Par. =    2.15203    2030.42  0.0300175     10.511
## It.    4, RSS = 8.63728e+06, Par. =    2.15396    2029.45  0.0322733    10.2565
## It.    5, RSS = 8.52189e+06, Par. =     2.1539     2029.3  0.0374148    9.91887
## It.    6, RSS = 8.1816e+06, Par. =    2.15328    2029.16  0.0480098    9.45435
## It.    7, RSS = 6.70488e+06, Par. =    2.15249    2028.59  0.0720382    8.81314
## It.    8, RSS = 1.94908e+06, Par. =    2.15419    2024.79   0.141003    7.87235
## It.    9, RSS =     287438, Par. =    2.18792    2004.54   0.245261     8.1077
## It.   10, RSS =      78460, Par. =    2.18873    2004.42   0.247582    8.00414
## It.   11, RSS =    76947.1, Par. =    2.19234    2002.69   0.250656    8.00584
## It.   12, RSS =      76947, Par. =    2.19236    2002.68   0.250659    8.00579
## It.   13, RSS =      76947, Par. =    2.19236    2002.68   0.250659    8.00579
## get the fitted values
N1 <- do.call("f", c(list(TT = TT), out$par))

## add a blue line representing the fitting values to the plot of data
lines(TT, N1, col = "blue", lwd = 2)

## add a plot of the log residual sum of squares as it is made to decrease
## each iteration; note that the RSS at the starting parameter values is
## also stored
plot(1:(out$niter + 1), log(out$rsstrace), type = "b", main = "log residual sum of squares vs. iteration number",
    xlab = "iteration", ylab = "log residual sum of squares", pch = 21, bg = 2)
```





```
## get information regarding standard errors
summary(out)

##
## Parameters:
##     Estimate Std. Error t value Pr(>|t|)
## tau 2.19e+00   3.25e-03     674   <2e-16 ***
## N0  2.00e+03   2.09e+00     958   <2e-16 ***
## a   2.51e-01   1.08e-03     233   <2e-16 ***
## f0  8.01e+00   2.77e-03    2895   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 12.4 on 497 degrees of freedom
## Number of iterations to termination: 13
## Reason for termination: Relative error in the sum of squares is at most `ftol'.
```

6.3 Examples for `nlsLM`

```
### Examples from 'nls' doc ###
DNase1 <- subset(DNase, Run == 1)
## using a selfStart model
fm1DNase1 <- nlsLM(density ~ SSlogis(log(conc), Asym, xmid, scal), DNase1)
## using logistic formula
fm2DNase1 <- nlsLM(density ~ Asym/(1 + exp((xmid - log(conc))/scal)), data = DNase1,
    start = list(Asym = 3, xmid = 0, scal = 1))

## all generics are applicable
coef(fm1DNase1)

##  Asym  xmid  scal
## 2.345 1.483 1.041

confint(fm1DNase1)

## Waiting for profiling to be done...

##        2.5% 97.5%
## Asym 2.1935 2.539
## xmid 1.3215 1.679
## scal 0.9743 1.115

deviance(fm1DNase1)

## [1] 0.00479

df.residual(fm1DNase1)

## [1] 13

fitted(fm1DNase1)

##  [1] 0.03068 0.03068 0.11205 0.11205 0.20858 0.20858 0.37433 0.37433
##  [9] 0.63278 0.63278 0.98086 0.98086 1.36751 1.36751 1.71499 1.71499
## attr(,"label")
## [1] "Fitted values"

formula(fm1DNase1)

## density ~ SSlogis(log(conc), Asym, xmid, scal)
## <environment: 0x9dd747c>

logLik(fm1DNase1)

## 'log Lik.' 42.21 (df=4)

predict(fm1DNase1)

##  [1] 0.03068 0.03068 0.11205 0.11205 0.20858 0.20858 0.37433 0.37433
##  [9] 0.63278 0.63278 0.98086 0.98086 1.36751 1.36751 1.71499 1.71499
```

```
print(fm1DNase1)

## Nonlinear regression model
##   model:  density ~ SSlogis(log(conc), Asym, xmid, scal)
##    data:  DNase1
## Asym xmid scal
## 2.35 1.48 1.04
##  residual sum-of-squares: 0.00479
##
## Number of iterations to convergence: 1
## Achieved convergence tolerance: 1.49e-08

profile(fm1DNase1)

## $Asym
##        tau par.vals.Asym par.vals.xmid par.vals.scal
## 1  -3.1915        2.1320        1.2581        0.9551
## 2  -2.5508        2.1695        1.2985        0.9713
## 3  -1.9084        2.2095        1.3412        0.9882
## 4  -1.2643        2.2523        1.3865        1.0056
## 5  -0.6192        2.2981        1.4344        1.0236
## 6   0.0000        2.3452        1.4831        1.0415
## 7   0.5790        2.3923        1.5312        1.0587
## 8   1.1426        2.4413        1.5805        1.0759
## 9   1.7053        2.4936        1.6326        1.0936
## 10  2.2665        2.5497        1.6874        1.1118
## 11  2.8263        2.6099        1.7453        1.1305
## 12  3.3845        2.6747        1.8065        1.1497
##
## $xmid
##        tau par.vals.Asym par.vals.xmid par.vals.scal
## 1  -3.1513        2.1382        1.2562        0.9566
## 2  -2.5187        2.1745        1.2973        0.9724
## 3  -1.8849        2.2132        1.3405        0.9888
## 4  -1.2500        2.2547        1.3860        1.0059
## 5  -0.6145        2.2991        1.4341        1.0237
## 6   0.0000        2.3452        1.4831        1.0415
## 7   0.5830        2.3920        1.5321        1.0589
## 8   1.1547        2.4412        1.5828        1.0766
## 9   1.7258        2.4940        1.6361        1.0949
## 10  2.2960        2.5507        1.6924        1.1137
## 11  2.8654        2.6117        1.7519        1.1331
## 12  3.4339        2.6776        1.8149        1.1531
##
## $scal
##        tau par.vals.Asym par.vals.xmid par.vals.scal
## 1  -3.0759        2.1593        1.2850        0.9475
## 2  -2.4589        2.1920        1.3203        0.9655
## 3  -1.8416        2.2268        1.3577        0.9839
## 4  -1.2240        2.2639        1.3973        1.0027
## 5  -0.6063        2.3036        1.4393        1.0220
## 6   0.0000        2.3452        1.4831        1.0415
## 7   0.5913        2.3886        1.5284        1.0609
## 8   1.1789        2.4349        1.5761        1.0807
## 9   1.7663        2.4845        1.6267        1.1010
## 10  2.3534        2.5380        1.6805        1.1218
## 11  2.9402        2.5956        1.7377        1.1432
## 12  3.5268        2.6581        1.7987        1.1652
##
## attr(,"original.fit")
## Nonlinear regression model
##   model:  density ~ SSlogis(log(conc), Asym, xmid, scal)
##    data:  DNase1
##  Asym  xmid  scal
## 2.345 1.483 1.041
##  residual sum-of-squares: 0.00479
##
```

```
## Number of iterations to convergence: 1
## Achieved convergence tolerance: 1.49e-08
## attr(,"summary")
##
## Formula: density ~ SSlogis(log(conc), Asym, xmid, scal)
##
## Parameters:
##      Estimate Std. Error t value Pr(>|t|)
## Asym  2.34518    0.07815   30.01 2.17e-13 ***
## xmid  1.48309    0.08135   18.23 1.22e-10 ***
## scal  1.04145    0.03227   32.27 8.51e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.01919 on 13 degrees of freedom
##
## Number of iterations to convergence: 1
## Achieved convergence tolerance: 1.49e-08
##
## attr(,"class")
## [1] "profile.nls" "profile"
```

```
residuals(fm1DNase1)
```

```
##  [1] -0.0136806 -0.0126806  0.0089488  0.0119488 -0.0025804  0.0064196
##  [7]  0.0026723 -0.0003277 -0.0187778 -0.0237778  0.0381370  0.0201370
## [13] -0.0335131 -0.0035131  0.0150122 -0.0049878
## attr(,"label")
## [1] "Residuals"
```

```
summary(fm1DNase1)
```

```
##
## Formula: density ~ SSlogis(log(conc), Asym, xmid, scal)
##
## Parameters:
##      Estimate Std. Error t value Pr(>|t|)
## Asym   2.3452     0.0782    30.0  2.2e-13 ***
## xmid   1.4831     0.0814    18.2  1.2e-10 ***
## scal   1.0415     0.0323    32.3  8.5e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0192 on 13 degrees of freedom
##
## Number of iterations to convergence: 1
## Achieved convergence tolerance: 1.49e-08
##
```

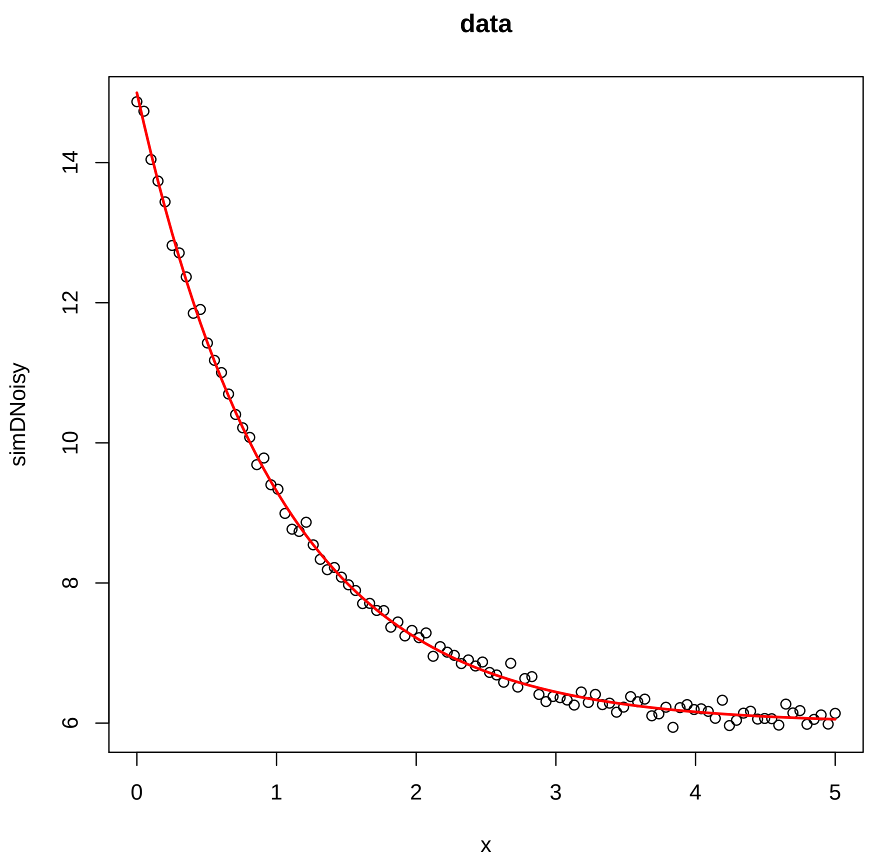
```
update(fm1DNase1)
```

```
## Nonlinear regression model
##   model:  density ~ SSlogis(log(conc), Asym, xmid, scal)
##    data:  DNase1
## Asym xmid scal
## 2.35 1.48 1.04
##  residual sum-of-squares: 0.00479
##
## Number of iterations to convergence: 1
## Achieved convergence tolerance: 1.49e-08
```

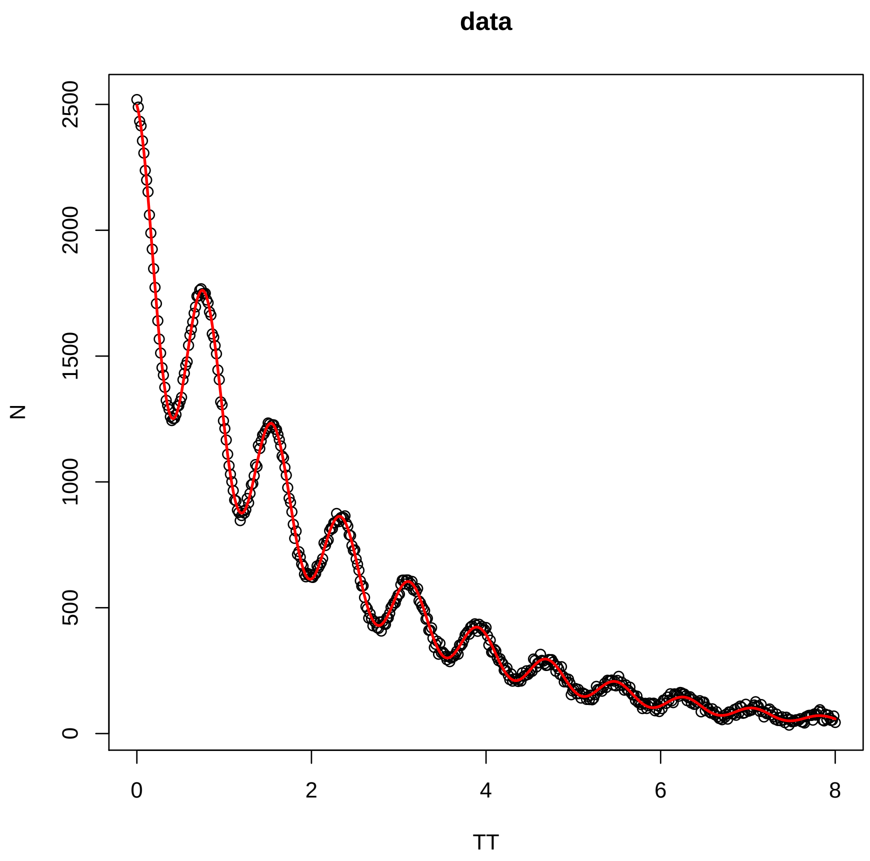
```
vcov(fm1DNase1)
```

```
##          Asym     xmid     scal
## Asym 0.006108 0.006274 0.002272
## xmid 0.006274 0.006618 0.002379
## scal 0.002272 0.002379 0.001041
```

```r
weights(fm1DNase1)

## NULL


## weighted nonlinear regression using inverse squared variance of the
## response gives same results as original 'nls' function
Treated <- Puromycin[Puromycin$state == "treated", ]
var.Treated <- tapply(Treated$rate, Treated$conc, var)
var.Treated <- rep(var.Treated, each = 2)
Pur.wt1 <- nls(rate ~ (Vm * conc)/(K + conc), data = Treated, start = list(Vm = 200,
    K = 0.1), weights = 1/var.Treated^2)
Pur.wt2 <- nlsLM(rate ~ (Vm * conc)/(K + conc), data = Treated, start = list(Vm = 200,
    K = 0.1), weights = 1/var.Treated^2)
all.equal(coef(Pur.wt1), coef(Pur.wt2))

## [1] TRUE


## 'nlsLM' can fit zero-noise data in contrast to 'nls'
x <- 1:10
y <- 2 * x + 3

try(nls(y ~ a + b * x, start = list(a = 0.12345, b = 0.54321)))

nlsLM(y ~ a + b * x, start = list(a = 0.12345, b = 0.54321))

## Nonlinear regression model
##   model:  y ~ a + b * x
##    data:  parent.frame()
## a b
## 3 2
##  residual sum-of-squares: 5.68e-29
##
## Number of iterations to convergence: 3
## Achieved convergence tolerance: 1.49e-08


### Examples from 'nls.lm' doc values over which to simulate data
x <- seq(0, 5, length = 100)
## model based on a list of parameters
getPred <- function(parS, xx) parS$a * exp(xx * parS$b) + parS$c
## parameter values used to simulate data
pp <- list(a = 9, b = -1, c = 6)
## simulated data with noise
simDNoisy <- getPred(pp, x) + rnorm(length(x), sd = 0.1)
## make model
mod <- nlsLM(simDNoisy ~ a * exp(b * x) + c, start = c(a = 3, b = -0.001, c = 1),
    trace = TRUE)

## It.    0, RSS =    1972.85, Par. =          3     -0.001          1
## It.    1, RSS =     329.29, Par. =    5.24963  -0.149375    3.23471
## It.    2, RSS =    103.581, Par. =    6.68637  -0.307052    4.27207
## It.    3, RSS =    52.2127, Par. =    7.48285  -0.424632    4.69863
## It.    4, RSS =    31.8902, Par. =    7.65413   -0.71461    6.11157
## It.    5, RSS =    2.77404, Par. =    8.71946   -1.00748    6.17887
## It.    6, RSS =   0.938304, Par. =    9.00272  -0.998119     5.9942
## It.    7, RSS =   0.938252, Par. =    9.00292  -0.998469    5.99412
## It.    8, RSS =   0.938252, Par. =    9.00292  -0.998469    5.99412

## plot data
plot(x, simDNoisy, main = "data")
## plot fitted values
lines(x, fitted(mod), col = 2, lwd = 2)
```



```
## create declining cosine with noise
TT <- seq(0, 8, length = 501)
tau <- 2.2
N0 <- 1000
a <- 0.25
f0 <- 8
Ndet <- N0 * exp(-TT/tau) * (1 + a * cos(f0 * TT))
N <- Ndet + rnorm(length(Ndet), mean = Ndet, sd = 0.01 * max(Ndet))
## make model
mod <- nlsLM(N ~ N0 * exp(-TT/tau) * (1 + a * cos(f0 * TT)), start = c(tau = 2.2,
    N0 = 1500, a = 0.25, f0 = 10), trace = TRUE)

## It.    0, RSS = 2.87899e+07, Par. =        2.2       1500       0.25         10
## It.    1, RSS = 9.1663e+06, Par. =     2.1107    2038.77  -0.022985    9.91972
## It.    2, RSS = 8.6723e+06, Par. =    2.17422    2019.67  0.0515522    10.7117
## It.    3, RSS = 8.56898e+06, Par. =    2.16338    2026.25  0.0283512    10.5471
## It.    4, RSS = 8.48735e+06, Par. =    2.16587    2024.99  0.0319723     10.197
## It.    5, RSS = 8.31215e+06, Par. =    2.16609    2024.59  0.0407957    9.79623
## It.    6, RSS = 7.84445e+06, Par. =    2.16464    2024.68  0.0550713    9.32309
## It.    7, RSS = 5.85794e+06, Par. =    2.16332    2024.18  0.0833864    8.66845
## It.    8, RSS = 1.61027e+06, Par. =     2.1691    2018.07   0.163444    7.79673
## It.    9, RSS =     237539, Par. =    2.19334    2003.66   0.237168     8.0845
## It.   10, RSS =    80678.5, Par. =    2.19969    2000.61   0.246596    7.99601
## It.   11, RSS =    79774.1, Par. =    2.20232    1999.37   0.248728    7.99915
## It.   12, RSS =    79774.1, Par. =    2.20232    1999.37   0.248731    7.99911
## It.   13, RSS =    79774.1, Par. =    2.20232    1999.37   0.248731    7.99911


## plot data
plot(TT, N, main = "data")
## plot fitted values
lines(TT, fitted(mod), col = 2, lwd = 2)
```



7 nls2 - INRIA

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.

August 15, 2012 – was not able to figure out the INSTALL script. ?? Should suggest a debian and/or Ubuntu package if this is possible, or else some improvement of INSTALL for mere mortals.

8 nlstools

9 Self-starting models

R provides for so-called ”self-starting models”. ?? starting values.

10 ALL THE OTHER STUFF NOT MOVED UP!!

Let us try an example initially presented by (**?**) and developed by (**?**). 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 (**?**). 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)

## $resid
## [1]  0.48070  0.66931 -2.28433  0.84374  0.73458  0.06655 -0.98581 -0.02506
## [9]  0.50032
##
## $jacobian
##       t1      t2     t3     t4
##  [1,]  1 -0.9816  1.126  2.475
##  [2,]  1 -0.9482  3.111  8.211
##  [3,]  1 -0.8698  7.485 22.787
##  [4,]  1 -0.7584 12.935 43.102
##  [5,]  1 -0.4843 21.659 80.956
##  [6,]  1 -0.2234 20.652 83.498
##  [7,]  1 -0.1493 17.515 72.569
##  [8,]  1 -0.0869 13.095 55.634
##  [9,]  1 -0.0385  7.735 33.798
##
## $feval
## [1] 76
##
## $jeval
## [1] 50
##
## $coeffs
## [1] 69.955 61.681 -9.209  2.378
##
## $ssquares
## [1] 8.376
##
```

```
anmrtx <- try(nlxb(regmod, start = huetstart, trace = FALSE, data = pastured))
print(strwrap(anmrtx))

##  [1] "c(0.480699476110992, 0.669309701586503, -2.28432650017661,"
##  [2] "0.843738460841614, 0.734575256138093, 0.0665546618861583,"
##  [3] "-0.985808933151056, -0.0250584603521418, 0.500316337120296)"
##  [4] "c(1, 1, 1, 1, 1, 1, 1, 1, 1, -0.981567160420883,"
##  [5] "-0.948192289406167, -0.869783557170751, -0.758436212560273,"
##  [6] "-0.484272123696113, -0.223383622127412, -0.149331587423979,"
##  [7] "-0.0869019449646661, -0.0385020596618461, 1.12642043233262,"
##  [8] "3.11132895498809, 7.48468988716119, 12.9349083313689,"
##  [9] "21.6594224095687, 20.652293670436, 17.51548586967,"
## [10] "13.0949252904654, 7.73503096811733, 2.47499865833493,"
## [11] "8.2109754835055, 22.7873063008638, 43.1017598804902,"
## [12] "80.9557650898109, 83.4982821079476, 72.56901775625,"
## [13] "55.6337277915341, 33.7978144524062)"
## [14] "61"
## [15] "39"
## [16] "c(69.9551789601637, 61.6814436396711, -9.20893535565824,"
## [17] "2.37781880027694)"
## [18] "8.37588355893792"
```

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 gradient"
## [2] "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 = FALSE, data ="
## [2] "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 (**?**). ?? Do we want a page ref?

11 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)
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: 8.33e-08

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

## Note that the above is just the nls() summary result.
```

12 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`** (**?**). The latter is built on the Minpack Fortran codes of (**?**) 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 * 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, 12, 3)
    tt <- 1:12
    yy <- exp(-0.1 * x[3] * tt)  # We don't need data for the Jacobian
    zz <- 100/(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)

## $resid
##  [1]  0.01190 -0.03276  0.09203  0.20878  0.39263 -0.05759 -1.10573
##  [8]  0.71579 -0.10765 -0.34840  0.65259 -0.28757
##
## $jacobian
##         [,1]   [,2]    [,3]
##  [1,]  2.712 -1.054  0.5176
##  [2,]  3.674 -1.414  1.3885
##  [3,]  4.960 -1.884  2.7742
##  [4,]  6.664 -2.486  4.8814
##  [5,]  8.901 -3.240  7.9537
##  [6,] 11.792 -4.157 12.2438
##  [7,] 15.464 -5.224 17.9522
##  [8,] 20.019 -6.399 25.1294
##  [9,] 25.511 -7.594 33.5526
## [10,] 31.908 -8.683 42.6252
## [11,] 39.069 -9.513 51.3725
## [12,] 46.733 -9.948 58.6047
##
## $feval
## [1] 24
##
## $jeval
## [1] 15
##
## $coeffs
## [1] 1.962 4.909 3.136
##
## $ssquares
## [1] 2.587
##
```

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)

## $resid
##  [1]  0.01190 -0.03276  0.09203  0.20878  0.39263 -0.05759 -1.10573
##  [8]  0.71579 -0.10765 -0.34840  0.65259 -0.28757
##
## $jacobian
##         [,1]   [,2]    [,3]
##  [1,]  2.712 -1.054  0.5176
##  [2,]  3.674 -1.414  1.3885
##  [3,]  4.960 -1.884  2.7742
##  [4,]  6.664 -2.486  4.8814
##  [5,]  8.901 -3.240  7.9537
##  [6,] 11.792 -4.157 12.2438
##  [7,] 15.464 -5.224 17.9522
##  [8,] 20.019 -6.399 25.1294
##  [9,] 25.511 -7.594 33.5526
## [10,] 31.908 -8.683 42.6252
## [11,] 39.069 -9.513 51.3725
## [12,] 46.733 -9.948 58.6047
##
## $feval
## [1] 29
##
## $jeval
## [1] 15
##
## $coeffs
## [1] 1.962 4.909 3.136
##
## $ssquares
## [1] 2.587
##
```

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 calculated
    JJ <- shobbs.jac(x)
    2 * as.vector(crossprod(JJ, res))
}
require(optimx)
aopx <- optimx(st, shobbs.f, shobbs.g, control = list(all.methods = TRUE))
```

```
## end topstuff in optimxCRAN

optansout(aopx, NULL)  # no file output

##                      par   par      method  fns grs itns conv  KKT1 KKT2
## 2  1.912, 4.825, 3.159 2.668          CG  427 101 NULL    1 FALSE TRUE
## 3  1.964, 4.912, 3.134 2.588 Nelder-Mead  196  NA NULL    0 FALSE TRUE
## 7  1.962, 4.909, 3.136 2.587         spg  188  NA  150    0  TRUE TRUE
## 5  1.962, 4.909, 3.136 2.587         nlm   NA  NA   50    0  TRUE TRUE
## 1  1.962, 4.909, 3.136 2.587        BFGS  119  36 NULL    0  TRUE TRUE
## 12 1.962, 4.909, 3.136 2.587      bobyqa  705  NA NULL    0  TRUE TRUE
## 11 1.962, 4.909, 3.136 2.587      newuoa 1957  NA NULL    0  TRUE TRUE
## 4  1.962, 4.909, 3.136 2.587    L-BFGS-B   41  41 NULL    0  TRUE TRUE
## 10 1.962, 4.909, 3.136 2.587      Rvmmin   83  47 NULL    0  TRUE TRUE
## 6  1.962, 4.909, 3.136 2.587      nlminb   31  29   28    0  TRUE TRUE
## 9  1.962, 4.909, 3.136 2.587      Rcgmin  138  50 NULL    0  TRUE TRUE
## 8  1.962, 4.909, 3.136 2.587      ucminf   46  46 NULL    0  TRUE TRUE
##    xtimes
## 2   0.016
## 3   0.004
## 7   0.036
## 5   0.008
## 1   0.004
## 12   0.02
## 11   0.06
## 4   0.004
## 10  0.012
## 6   0.004
## 9   0.012
## 8   0.004

## [1] TRUE

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))

## end topstuff in optimxCRAN

## Warning:  A NULL gradient function is being replaced numDeriv 'grad()'for
## Rcgmin

## function(x) {
##     res <- shobbs.res(x)
##     as.numeric(crossprod(res))
## }
## <environment: 0x8fde188>

## Warning:  A gradient calculation (analytic or numerical) MUST be provided
## for Rvmmin

## Error:  missing value where TRUE/FALSE needed

optansout(aopxn, NULL)  # no file output

## Error:  object 'aopxn' not found
```

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.

13 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: 0x9cb3014>

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 -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: 0xa038650>

# 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 -2.372e-11 5.804e-10 1.275e-09
##  [2,]  1 -2.968e-17 1.130e-15 2.981e-15
##  [3,]  1 -1.617e-25 9.232e-24 2.811e-23
##  [4,]  1 -8.811e-34 6.706e-32 2.235e-31
##  [5,]  1 -2.615e-50 2.986e-48 1.116e-47
##  [6,]  1 -5.123e-68 7.938e-66 3.209e-65
##  [7,]  1 -4.230e-75 7.243e-73 3.001e-72
##  [8,]  1 -2.304e-83 4.385e-81 1.863e-80
##  [9,]  1 -5.467e-94 1.174e-91 5.130e-91

# Now compute the numerical approximation
Jn <- jacobian(jres, ones, , yield = pastured$yield, time = pastured$time)
cat("maxabsdiff=", max(abs(Jn - valjjac)), "\n")

## maxabsdiff= 3.774e-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: 0xa0b02ac>

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: 0x9ff9584>

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

## valgr:

print(valgr)

## [1] -6.811e+02  3.763e-10 -9.205e-09 -2.023e-08

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

## maxabsdiff= 7.477e-08
```

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] 61.06 59.20 51.41 47.67 30.65 13.89  8.27  5.38  2.92

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 -1.234e-04 6.664e-02 1.464e-01
##  [2,]  1 -8.315e-07 6.985e-04 1.843e-03
##  [3,]  1 -7.583e-10 9.554e-07 2.909e-06
##  [4,]  1 -6.914e-13 1.162e-09 3.871e-09
##  [5,]  1 -5.750e-19 1.449e-15 5.415e-15
##  [6,]  1 -1.759e-25 6.015e-22 2.432e-21
##  [7,]  1 -4.360e-28 1.648e-24 6.828e-24
##  [8,]  1 -3.975e-31 1.670e-27 7.094e-27
##  [9,]  1 -4.906e-35 2.325e-31 1.016e-30

Jn <- jacobian(jres, huetstart, , yield = pastured$yield, time = pastured$time)
cat("maxabsdiff=", max(abs(Jn - valjjac)), "\n")

## maxabsdiff= 5.395e-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= 5.953e-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(0.480699475409779, 0.669309701325741, -2.28432649983562,"
##  [2] "0.843738461541676, 0.734575256578069, 0.0665546616416748,"
##  [3] "-0.985808933450038, -0.0250584605193325, 0.500316337308163)"
##  [4] "c(1, 1, 1, 1, 1, 1, 1, 1, 1, -0.981567160415026,"
##  [5] "-0.948192289394349, -0.869783557151951, -0.758436212539591,"
##  [6] "-0.484272123689345, -0.22338362214097, -0.14933158744104,"
##  [7] "-0.086901944981799, -0.0385020596749348, 1.12642043272705,"
##  [8] "3.1113289557883, 7.48468988842378, 12.9349083327494,"
##  [9] "21.6594224104496, 20.6522936715837, 17.5154858712384,"
## [10] "13.0949252924535, 7.73503097021314, 2.47499865920158,"
## [11] "8.21097548561731, 22.7873063047078, 43.1017598850905,"
## [12] "80.9557650931036, 83.498282112588, 72.569017762748,"
## [13] "55.6337277999807, 33.7978144615637)"
## [14] "74"
## [15] "48"
## [16] "c(69.9551789612429, 61.6814436418531, -9.20893535490747,"
## [17] "2.37781880008123)"
## [18] "8.37588355893788"

cat("Huet start\n")

## Huet start

anlfbh <- nlfb(huetstart, jres, jjac, trace = FALSE, yield = pastured$yield,
    time = pastured$time)
print(strwrap(anlfbh))

##  [1] "c(0.480699465869456, 0.669309697775223, -2.28432649519877,"
##  [2] "0.84373847107085, 0.734575262591456, 0.0665546583437617,"
##  [3] "-0.985808937499776, -0.0250584627932966, 0.500316339841277)"
##  [4] "c(1, 1, 1, 1, 1, 1, 1, 1, 1, -0.981567160335378,"
##  [5] "-0.94819228923362, -0.869783556896137, -0.75843621225793,"
##  [6] "-0.484272123596337, -0.223383622324199, -0.149331587672017,"
##  [7] "-0.0869019452139657, -0.0385020598524092, 1.12642043808933,"
##  [8] "3.11132896666899, 7.48468990559557, 12.9349083515304,"
##  [9] "21.6594224224275, 20.652293687139, 17.5154858924942,"
## [10] "13.0949253194057, 7.73503099863509, 2.47499867098372,"
## [11] "8.21097551433206, 22.7873063569877, 43.1017599476725,"
## [12] "80.9557651378729, 83.498282175479, 72.5690178508139,"
## [13] "55.6337279144867, 33.7978145857519)"
## [14] "60"
## [15] "37"
## [16] "c(69.9551789758633, 61.6814436714725, -9.20893534470294,"
## [17] "2.37781879742191)"
## [18] "8.37588355893793"
```

14 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)

## $resid
##  [1]  0.01190 -0.03276  0.09203  0.20878  0.39263 -0.05759 -1.10573
##  [8]  0.71579 -0.10765 -0.34840  0.65259 -0.28757
##
## $jacobian
##            b1      b2      b3
##  [1,] 0.02712 -0.1054   5.176
##  [2,] 0.03674 -0.1414  13.885
##  [3,] 0.04960 -0.1884  27.742
##  [4,] 0.06664 -0.2486  48.814
##  [5,] 0.08901 -0.3240  79.537
##  [6,] 0.11792 -0.4157 122.438
##  [7,] 0.15464 -0.5224 179.522
##  [8,] 0.20019 -0.6399 251.294
##  [9,] 0.25511 -0.7594 335.526
## [10,] 0.31908 -0.8683 426.252
## [11,] 0.39069 -0.9513 513.725
## [12,] 0.46733 -0.9948 586.047
##
## $feval
## [1] 36
##
## $jeval
## [1] 22
##
## $coeffs
## [1] 196.1863  49.0916   0.3136
##
## $ssquares
## [1] 2.587
##
```

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 = 0.1)  # 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)

## $resid
##  [1]  0.01190 -0.03276  0.09203  0.20878  0.39263 -0.05759 -1.10573
##  [8]  0.71579 -0.10765 -0.34840  0.65259 -0.28757
##
## $jacobian
##            b1      b2      b3
##  [1,] 0.02712 -0.1054   5.176
##  [2,] 0.03674 -0.1414  13.885
##  [3,] 0.04960 -0.1884  27.742
##  [4,] 0.06664 -0.2486  48.814
##  [5,] 0.08901 -0.3240  79.537
##  [6,] 0.11792 -0.4157 122.438
##  [7,] 0.15464 -0.5224 179.522
##  [8,] 0.20019 -0.6399 251.294
##  [9,] 0.25511 -0.7594 335.526
## [10,] 0.31908 -0.8683 426.252
## [11,] 0.39069 -0.9513 513.725
## [12,] 0.46733 -0.9948 586.047
##
## $feval
## [1] 29
##
## $jeval
## [1] 17
##
## $coeffs
## [1] 196.1863  49.0916   0.3136
##
## $ssquares
## [1] 2.587
##
```

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 = 0.25), 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), da
anlsb2i <- try(nls(eunsc, start = start1, lower = c(b1 = 0, b2 = 0, b3 = 0),
    upper = c(b1 = 500, b2 = 100, b3 = 0.25), 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 c
## 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), da
```

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 = 0.25), data = weeddata1))

## Warning:  NaNs produced

print(anlxb2f)

## $resid
##  [1]  1.8873  1.9614  2.1153  2.1255  2.0179  1.0532 -0.7345  0.1965
##  [9] -1.4661 -2.1116 -0.4888  0.9925
##
## $jacobian
##       b1       b2 b3
##  [1,]  0 -0.08064  0
##  [2,]  0 -0.10270  0
##  [3,]  0 -0.13051  0
##  [4,]  0 -0.16536  0
##  [5,]  0 -0.20875  0
##  [6,]  0 -0.26233  0
##  [7,]  0 -0.32774  0
##  [8,]  0 -0.40652  0
##  [9,]  0 -0.49974  0
## [10,]  0 -0.60761  0
## [11,]  0 -0.72893  0
## [12,]  0 -0.86056  0
##
## $feval
## [1] 32
##
## $jeval
## [1] 16
##
## $coeffs
## [1] 500.00  87.94   0.25
##
## $ssquares
## [1] 29.99
##

anlsb2f <- try(nls(eunsc, start = startf1, lower = c(b1 = 0, b2 = 0, b3 = 0),
    upper = c(b1 = 500, b2 = 100, b3 = 0.25), 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

## $resid
##  [1]  22.387  22.901  22.856  21.850  19.709  15.468   8.911   3.299
##  [9]  -6.981 -18.628 -30.690 -45.827
##
## $jacobian
##           b1 b2    b3
##  [1,] 0.5495  0 12.48
##  [2,] 0.5980  0 24.23
##  [3,] 0.6447  0 34.64
##  [4,] 0.6888  0 43.22
##  [5,] 0.7297  0 49.71
##  [6,] 0.7670  0 54.04
##  [7,] 0.8006  0 56.31
##  [8,] 0.8305  0 56.77
##  [9,] 0.8566  0 55.71
## [10,] 0.8793  0 53.48
## [11,] 0.8989  0 50.40
## [12,] 0.9156  0 46.76
##
## $feval
## [1] 57
##
## $jeval
## [1] 33
##
## $coeffs
## [1] 50.4018  1.0000  0.1986
##
## $ssquares
## [1] 6181
##

an1qm3 <- try(nlxb(eunsc, start = start1, data = weeddata1, masked = c("b3")))
print(an1qm3)  # b3 masked

## $resid
##  [1]  -5.2150  -6.9877  -8.9560 -11.0394 -12.2945 -11.4407  -6.0304
##  [8]   5.8440  11.0794   8.2119  -0.3233 -14.4932
##
## $jacobian
##             b1         b2 b3
##  [1,] 0.001184 -4.049e-05  0
##  [2,] 0.003211 -1.096e-04  0
##  [3,] 0.008680 -2.947e-04  0
##  [4,] 0.023248 -7.778e-04  0
##  [5,] 0.060766 -1.955e-03  0
##  [6,] 0.149563 -4.357e-03  0
##  [7,] 0.323435 -7.495e-03  0
##  [8,] 0.565121 -8.418e-03  0
##  [9,] 0.779365 -5.890e-03  0
## [10,] 0.905678 -2.926e-03  0
## [11,] 0.963101 -1.217e-03  0
## [12,] 0.986101 -4.694e-04  0
```

```
##
## $feval
## [1] 48
##
## $jeval
## [1] 31
##
## $coeffs
## [1]   78.57 2293.95    1.00
##
## $ssquares
## [1] 1031
##

# 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, 0.3), masked = c("b2")))

## Warning:  NaNs produced

print(an1qbm2)

## $resid
##  [1]  22.387  22.901  22.856  21.850  19.709  15.468   8.911   3.299
##  [9]  -6.981 -18.628 -30.690 -45.827
##
## $jacobian
##           b1 b2    b3
##  [1,] 0.5495  0 12.48
##  [2,] 0.5980  0 24.23
##  [3,] 0.6447  0 34.64
##  [4,] 0.6888  0 43.22
##  [5,] 0.7297  0 49.71
##  [6,] 0.7670  0 54.04
##  [7,] 0.8006  0 56.31
##  [8,] 0.8305  0 56.77
##  [9,] 0.8566  0 55.71
## [10,] 0.8793  0 53.48
## [11,] 0.8989  0 50.40
## [12,] 0.9156  0 46.76
##
## $feval
## [1] 49
##
## $jeval
## [1] 27
##
## $coeffs
## [1] 50.4018  1.0000  0.1986
##
## $ssquares
## [1] 6181
##
```

```
an1qbm2x <- try(nlxb(eunsc, start = startf1, data = weeddata1, lower = c(0,
    0, 0), upper = c(48, 60, 0.3), masked = c("b2")))

## Warning:  NaNs produced

## Warning:  NaNs produced

## Warning:  NaNs produced

## Warning:  NaNs produced

## Warning:  NaNs produced

print(an1qbm2x)

## $resid
##  [1]  21.274  21.864  21.876  20.901  18.761  14.494   7.885   2.200
##  [9]  -8.167 -19.913 -32.077 -47.317
##
## $jacobian
##       b1 b2    b3
##  [1,]  0  0 11.86
##  [2,]  0  0 22.91
##  [3,]  0  0 32.47
##  [4,]  0  0 40.05
##  [5,]  0  0 45.42
##  [6,]  0  0 48.59
##  [7,]  0  0 49.74
##  [8,]  0  0 49.19
##  [9,]  0  0 47.33
## [10,]  0  0 44.51
## [11,]  0  0 41.09
## [12,]  0  0 37.34
##
## $feval
## [1] 37
##
## $jeval
## [1] 19
##
## $coeffs
## [1] 48.000  1.000  0.216
##
## $ssquares
## [1] 6206
##
```

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, 12, 3)
    tt <- 1:12
    yy <- exp(-x[3] * tt)
    zz <- 1/(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

## $resid
##  [1]  0.01190 -0.03276  0.09203  0.20878  0.39263 -0.05759 -1.10573
##  [8]  0.71579 -0.10765 -0.34840  0.65259 -0.28757
##
## $jacobian
##          [,1]    [,2]    [,3]
##  [1,] 0.02712 -0.1054   5.176
##  [2,] 0.03674 -0.1414  13.885
##  [3,] 0.04960 -0.1884  27.742
##  [4,] 0.06664 -0.2486  48.814
##  [5,] 0.08901 -0.3240  79.537
##  [6,] 0.11792 -0.4157 122.438
##  [7,] 0.15464 -0.5224 179.522
##  [8,] 0.20019 -0.6399 251.294
##  [9,] 0.25511 -0.7594 335.526
## [10,] 0.31908 -0.8683 426.252
## [11,] 0.39069 -0.9513 513.725
## [12,] 0.46733 -0.9948 586.047
##
## $feval
## [1] 37
##
## $jeval
## [1] 24
##
## $coeffs
## [1] 196.1863  49.0916   0.3136
##
## $ssquares
## [1] 2.587
##

## No jacobian - use internal approximation
ans1n <- nlfb(start1, hobbs.res)
ans1n

## $resid
##  [1]  0.01190 -0.03276  0.09203  0.20878  0.39263 -0.05759 -1.10573
##  [8]  0.71579 -0.10765 -0.34840  0.65259 -0.28757
##
## $jacobian
##          [,1]    [,2]    [,3]
##  [1,] 0.02712 -0.1054   5.176
##  [2,] 0.03674 -0.1414  13.885
##  [3,] 0.04960 -0.1884  27.742
##  [4,] 0.06664 -0.2486  48.814
##  [5,] 0.08901 -0.3240  79.537
##  [6,] 0.11792 -0.4157 122.438
##  [7,] 0.15464 -0.5224 179.522
##  [8,] 0.20019 -0.6399 251.294
##  [9,] 0.25511 -0.7594 335.526
## [10,] 0.31908 -0.8683 426.252
## [11,] 0.39069 -0.9513 513.725
## [12,] 0.46733 -0.9948 586.047
##
## $feval
## [1] 40
##
```

```
## $jeval
## [1] 22
##
## $coeffs
## [1] 196.1863  49.0916   0.3136
##
## $ssquares
## [1] 2.587
##

# 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 = 0.25)))
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

# 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 = 0.25))

## Warning:  NaNs produced

## Warning:  NaNs produced

## Warning:  NaNs produced

## Warning:  NaNs produced

ans2f

## $resid
##  [1]  1.8873  1.9614  2.1153  2.1255  2.0179  1.0532 -0.7345  0.1965
##  [9] -1.4661 -2.1116 -0.4888  0.9925
##
## $jacobian
##       [,1]     [,2] [,3]
##  [1,]    0 -0.08064    0
##  [2,]    0 -0.10270    0
##  [3,]    0 -0.13051    0
##  [4,]    0 -0.16536    0
##  [5,]    0 -0.20875    0
##  [6,]    0 -0.26233    0
##  [7,]    0 -0.32774    0
##  [8,]    0 -0.40652    0
##  [9,]    0 -0.49974    0
## [10,]    0 -0.60761    0
## [11,]    0 -0.72893    0
## [12,]    0 -0.86056    0
##
## $feval
## [1] 31
##
## $jeval
## [1] 16
##
## $coeffs
## [1] 500.00  87.94   0.25
##
## $ssquares
## [1] 29.99
##
```

```
# Mask b2
ansm2 <- nlfb(start1, hobbs.res, hobbs.jac, maskidx = c(2))
ansm2

## $resid
##  [1]  22.387  22.901  22.856  21.850  19.709  15.468   8.911   3.299
##  [9]  -6.981 -18.628 -30.690 -45.827
##
## $jacobian
##         [,1] [,2]  [,3]
##  [1,] 0.5495    0 12.48
##  [2,] 0.5980    0 24.23
##  [3,] 0.6447    0 34.64
##  [4,] 0.6888    0 43.22
##  [5,] 0.7297    0 49.71
##  [6,] 0.7670    0 54.04
##  [7,] 0.8006    0 56.31
##  [8,] 0.8305    0 56.77
##  [9,] 0.8566    0 55.71
## [10,] 0.8793    0 53.48
## [11,] 0.8989    0 50.40
## [12,] 0.9156    0 46.76
##
## $feval
## [1] 56
##
## $jeval
## [1] 32
##
## $coeffs
## [1] 50.4018  1.0000  0.1986
##
## $ssquares
## [1] 6181
##

# Mask b3
ansm3 <- nlfb(start1, hobbs.res, hobbs.jac, maskidx = c(3))
ansm3

## $resid
##  [1]  -5.2150  -6.9877  -8.9560 -11.0394 -12.2945 -11.4407  -6.0304
##  [8]   5.8440  11.0794   8.2119  -0.3233 -14.4932
##
## $jacobian
##           [,1]       [,2] [,3]
##  [1,] 0.001184 -4.049e-05    0
##  [2,] 0.003211 -1.096e-04    0
##  [3,] 0.008680 -2.947e-04    0
##  [4,] 0.023248 -7.778e-04    0
##  [5,] 0.060766 -1.955e-03    0
##  [6,] 0.149563 -4.357e-03    0
##  [7,] 0.323435 -7.495e-03    0
##  [8,] 0.565121 -8.418e-03    0
##  [9,] 0.779365 -5.890e-03    0
## [10,] 0.905678 -2.926e-03    0
## [11,] 0.963101 -1.217e-03    0
## [12,] 0.986101 -4.694e-04    0
##
## $feval
## [1] 48
##
## $jeval
## [1] 31
##
## $coeffs
## [1]   78.57 2293.95    1.00
```

```
##
## $ssquares
## [1] 1031
##

# 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, 0.3))

## Warning:  NaNs produced

ansmbm2

## $resid
##  [1]  22.387  22.901  22.856  21.850  19.709  15.468   8.911   3.299
##  [9]  -6.981 -18.628 -30.690 -45.827
##
## $jacobian
##         [,1] [,2]  [,3]
##  [1,] 0.5495    0 12.48
##  [2,] 0.5980    0 24.23
##  [3,] 0.6447    0 34.64
##  [4,] 0.6888    0 43.22
##  [5,] 0.7297    0 49.71
##  [6,] 0.7670    0 54.04
##  [7,] 0.8006    0 56.31
##  [8,] 0.8305    0 56.77
##  [9,] 0.8566    0 55.71
## [10,] 0.8793    0 53.48
## [11,] 0.8989    0 50.40
## [12,] 0.9156    0 46.76
##
## $feval
## [1] 50
##
## $jeval
## [1] 28
##
## $coeffs
## [1] 50.4018  1.0000  0.1986
##
## $ssquares
## [1] 6181
##

# Active bound
ansmbm2x <- nlfb(startf1, hobbs.res, hobbs.jac, maskidx = c(2), lower = c(0,
    0, 0), upper = c(48, 60, 0.3))

## Warning:  NaNs produced

## Warning:  NaNs produced

## Warning:  NaNs produced

ansmbm2x
```

```
## $resid
##  [1]  21.274  21.864  21.876  20.901  18.761  14.494   7.885   2.200
##  [9]  -8.167 -19.913 -32.077 -47.317
##
## $jacobian
##       [,1] [,2]  [,3]
##  [1,]    0    0 11.86
##  [2,]    0    0 22.91
##  [3,]    0    0 32.47
##  [4,]    0    0 40.05
##  [5,]    0    0 45.42
##  [6,]    0    0 48.59
##  [7,]    0    0 49.74
##  [8,]    0    0 49.19
##  [9,]    0    0 47.33
## [10,]    0    0 44.51
## [11,]    0    0 41.09
## [12,]    0    0 37.34
##
## $feval
## [1] 35
##
## $jeval
## [1] 17
##
## $coeffs
## [1] 48.000  1.000  0.216
##
## $ssquares
## [1] 6206
##
```

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 * as.numeric(crossprod(JJ, res))
}

# Check unconstrained
a1cg <- Rcgmin(start1, hobbs.f, hobbs.g)
a1cg

## $par
##       b1       b2       b3
## 196.1844  49.0909   0.3136
##
## $value
## [1] 2.587
##
## $counts
## [1] 1004  351
##
## $convergence
## [1] 1
##
## $message
## [1] "Too many function evaluations (> 1000) "
##

a1vm <- Rvmmin(start1, hobbs.f, hobbs.g)
a1vm

## $par
##       b1       b2       b3
## 196.1863  49.0916   0.3136
##
## $value
## [1] 2.587
##
## $counts
## [1] 199  52
##
## $convergence
## [1] 0
##
## $message
## [1] "Converged"
##

## No jacobian - use internal approximation
a1cgn <- try(Rcgmin(start1, hobbs.f))

## Warning:  A NULL gradient function is being replaced numDeriv 'grad()'for
## Rcgmin

## function(x) {
##     res <- hobbs.res(x)
##     as.numeric(crossprod(res))
## }
## <environment: 0xa34e070>

a1cgn

## $par
##       b1       b2       b3
## 196.1862  49.0916   0.3136
##
## $value
## [1] 2.587
##
## $counts
## [1] 775 258
##
## $convergence
## [1] 0
##
## $message
## [1] "Rcgmin seems to have converged"
##

a1vmn <- try(Rvmmin(start1, hobbs.f))

## Warning:  A gradient calculation (analytic or numerical) MUST be provided
## for Rvmmin

a1vmn

## [[1]]
## b1 b2 b3
##  1  1  1
##
## [[2]]
## [1] NA
##
## [[3]]
## [1] 0 0
##
## [[4]]
## [1] 9999
##
## [[5]]
## [1] "No gradient function provided for Rvmmin"
##

# But
grfwd <- function(par, userfn, fbase = NULL, eps = 1e-07, ...) {
    # 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

## [1] "Error in do.call(gr, list(par, userfn, ...)) : \n  could not find function \"grfwd\"\n"
## attr(,"class")
## [1] "try-error"
## attr(,"condition")
## <simpleError in do.call(gr, list(par, userfn, ...)): could not find function "grfwd">

# 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 = 0.25))

## Warning:  x[3], set 1 to upper bound = 0.25

a1cg2i

## $par
##     b1     b2     b3
## 500.00  87.94   0.25
##
## $value
## [1] 29.99
##
## $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 = 0.25))

## Warning:  Parameter out of bounds has been moved to nearest bound
## Warning:  Too many function evaluations

a1vm2i  # Fails to get to solution!

## $par
##      b1      b2      b3
## 35.9647  1.1238  0.4096
##
## $value
## [1] 7220
##
## $counts
## [1] 3001    6
##
## $convergence
## [1] 1
##
## $message
## [1] "Too many function evaluations"
##
## $bdmsk
## [1] 1 1 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 = 0.25))
a1cg2f

## $par
##     b1     b2     b3
## 500.00  87.94   0.25
##
## $value
## [1] 29.99
##
## $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 = 0.25))

## Warning:  Too many function evaluations

a1vm2f  # Gets there, but only just!

## $par
##     b1     b2     b3
## 499.96  87.93   0.25
##
## $value
## [1] 29.99
##
## $counts
## [1] 3001  494
##
```

```
## $convergence
## [1] 1
##
## $message
## [1] "Too many function evaluations"
##
## $bdmsk
## [1]  1  1 -1
##

# Mask b2
a1cgm2 <- Rcgmin(start1, hobbs.f, hobbs.g, bdmsk = c(1, 0, 1))
a1cgm2

## $par
##      b1      b2      b3
## 50.4018  1.0000  0.1986
##
## $value
## [1] 6181
##
## $counts
## [1] 112  39
##
## $convergence
## [1] 0
##
## $message
## [1] "Rcgmin seems to have converged"
##
## $bdmsk
## [1] 1 0 1
##

a1vmm2 <- Rvmmin(start1, hobbs.f, hobbs.g, bdmsk = c(1, 0, 1))
a1vmm2

## $par
##      b1      b2      b3
## 50.4018  1.0000  0.1986
##
## $value
## [1] 6181
##
## $counts
## [1] 58 14
##
## $convergence
## [1] 0
##
## $message
## [1] "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
##   78.57 2293.94    1.00
##
## $value
## [1] 1031
##
## $counts
## [1] 181  80
##
## $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
##   78.57 2293.95    1.00
##
## $value
## [1] 1031
##
## $counts
## [1] 102  32
##
## $convergence
## [1] 0
##
## $message
## [1] "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
## [1] 1 1
##
## $convergence
## [1] 0
##
## $message
## [1] "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, 0.3))

## Warning:  NaNs produced

ansmbm2

## $resid
##  [1]  22.387  22.901  22.856  21.850  19.709  15.468   8.911   3.299
##  [9]  -6.981 -18.628 -30.690 -45.827
##
## $jacobian
##          [,1] [,2]  [,3]
##  [1,] 0.5495    0 12.48
##  [2,] 0.5980    0 24.23
##  [3,] 0.6447    0 34.64
##  [4,] 0.6888    0 43.22
##  [5,] 0.7297    0 49.71
##  [6,] 0.7670    0 54.04
##  [7,] 0.8006    0 56.31
##  [8,] 0.8305    0 56.77
##  [9,] 0.8566    0 55.71
## [10,] 0.8793    0 53.48
## [11,] 0.8989    0 50.40
## [12,] 0.9156    0 46.76
##
## $feval
## [1] 50
##
## $jeval
## [1] 28
##
## $coeffs
## [1] 50.4018  1.0000  0.1986
##
## $ssquares
## [1] 6181
##

a1cgbm2 <- Rcgmin(start1, hobbs.f, hobbs.g, bdmsk = c(1, 0, 1), lower = c(0,
    0, 0), upper = c(200, 60, 0.3))

## Warning:  x[3], set 1 to upper bound = 0.3

a1cgbm2
```

```
## $par
##      b1      b2      b3
## 50.4018  1.0000  0.1986
##
## $value
## [1] 6181
##
## $counts
## [1] 76 29
##
## $convergence
## [1] 0
##
## $message
## [1] "Rcgmin seems to have converged"
##
## $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, 0.3))

## Warning:  Parameter out of bounds has been moved to nearest bound

a1vmbm2

## $par
##      b1      b2      b3
## 50.4018  1.0000  0.1986
##
## $value
## [1] 6181
##
## $counts
## [1] 79 24
##
## $convergence
## [1] 0
##
## $message
## [1] "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, 0.3))

## Warning:  x[3], set 1 to upper bound = 0.3

a1cgm2x

## $par
##     b1     b2     b3
## 48.000  1.000  0.216
##
## $value
## [1] 6206
##
## $counts
## [1] 37 14
##
## $convergence
## [1] 0
##
## $message
## [1] "Rcgmin seems to have converged"
##
## $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, 0.3))
```

```
## Warning:  Parameter out of bounds has been moved to nearest bound
```

```
a1vmm2x
```

```
## $par
##     b1     b2     b3
## 48.000  1.000  0.216
##
## $value
## [1] 6206
##
## $counts
## [1] 74 42
##
## $convergence
## [1] 0
##
## $message
## [1] "Converged"
##
## $bdmsk
## [1] 1 0 1
##
```

References

Dennis, J. E. and R. B. Schnabel (1983). *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* Englewood Cliffs, NJ: Prentice-Hall.

Elzhov, T. V., K. M. Mullen, A.-N. Spiess, and B. Bolker (2012). *minpack.lm: R interface to the Levenberg-Marquardt nonlinear least-squares algorithm found in MINPACK, plus support for bounds.* R Project for Statistical Computing. R package version 1.1-6.

Huet, S. S. et al. (1996). *Statistical tools for nonlinear regression: a practical guide with S-PLUS examples.* Springer series in statistics.

Moré, J. J., B. S. Garbow, and K. E. Hillstrom (1980). ANL-80-74, User Guide for MINPACK-1. Technical report.

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

Nash, J. C. (1979b). *Compact Numerical Methods for Computers: Linear Algebra and Function Minimisation.* Bristol: Adam Hilger. Second Edition, 1990, Bristol: Institute of Physics Publications.

Ratkowsky, D. A. (1983). *Nonlinear Regression Modeling: A Unified Practical Approach*. New York and Basel: Marcel Dekker Inc.