Specifying and Using Fixed Parameters in Nonlinear Estimation

John C. Nash

2023-05-09

Motivation

In finding optimal parameters in nonlinear optimization and nonlinear least squares problems, we frequently wish to fix one or more parameters while allowing the rest to be adjusted to explore or optimize an objective function.

This vignette discusses some ideas about specifying the fixed parameters. A lot of the material is drawn from Nash J C (2014) **Nonlinear parameter optimization using R tools** Chichester UK: Wiley, in particular chapters 11 and 12. There is, however, additional material concerning ways to manage extensible models, as well as some update to the package `nlsr`. The algorithm has been marginally altered to allow for different sub-variants to be used, and mechanisms for specifying parameter constraints and providing Jacobian approximations have been changed.

Background

Here are some of the ways fixed parameters may be specified in R packages.

Function `nlxb()` in package `nlsr` has argument `masked`:

> Character vector of quoted parameter names. These parameters will NOT be altered by the algorithm.

This approach has a simplicity that is attractive, but introduces an extra argument to calling sequences. (This approach was previously in defunct package `nlmrt`.)

Simlarly, function `nlfb()` in `nlsr` has argument `maskidx`:

> Vector of indices of the parameters to be masked. These parameters will NOT be altered by the algorithm. Note that the mechanism here is different from that in nlxb which uses the names of the parameters.

From `Rvmmin` and `Rcgmin` in package `optimx` the argument `bdmsk`:

> An indicator vector, having 1 for each parameter that is "free" or unconstrained, and 0 for any parameter that is fixed or MASKED for the duration of the optimization.

Note that the function `bmchk()` in package `optimx` contains a much more extensive examination of the bounds on parameters. In particular, it considers the issues of inadmissible bounds (lower > upper), when to convert a pair of bounds where upper["parameter"] - lower["parameter"] < tol to a fixed or masked parameter (`maskadded`) and whether parameters outside of bounds should be moved to the nearest bound (`parchanged`). It may be useful to use **inadmissible** to refer to situations where a lower bound is higher than an upper bound and **infeasible** where a parameter value, especially in a given starting vector, is outside the bounds.

Further in package `optimx`, the function `optimr()` can call many different "optimizers" (actually function minimization methods that may include bounds and possibly masks). These may be specified by setting the lower and upper bounds equal for the parameters to be fixed. This seems a simple method for specifying

masks, but does pose some issues. For example, what happens when the upper bound is only very slightly greater than the lower bound? Also should we stop or declare an error if starting values are NOT on the fixed value?

Of these methods, my preference is now to use the last one – setting lower and upper bounds equal, and furthermore requiring the starting value of such a parameter to this fixed value, otherwise declaring an error. The approach does not add any special argument for masking, and is relatively obvious to novice users. However, such users may be tempted to put in narrow bounds rather than explicit equalities, and this could have deleterious consequences.

In the revision to package `nlsr`, package `nlsr`, I have stopped using `masked` in `nlxb()` and `maskidx` in `nlfb()` (though the latter is a returned value). This is because I feel the use of equal lower and upper bounds is a better approach. Moreover, though it is not documented, it appears to "mostly work" for the base R function `nls()` with the `algorithm="port"` option and with `minpack.lm::nlsLM()`.

Internal structures

`bdmsk` is the internal structure used in `Rcgmin`, `Rvmmin` and `nlfb` to handle bounds constraints as well as masks. There is one element of `bdmsk` for each parameter, and in `Rcgmin` and `Rvmmin`, this is used on input to specify parameter i as fixed or masked by setting `bdmsk[i] <- 0`. Free parameters have their `bdmsk` element `1`, but during optimization in the presence of bounds, we can set other values. The full set is as follows

- 1 for a free or unconstrained parameter
- 0 for a masked or fixed parameter
- -0.5 for a parameter that is out of bounds high (> upper bound)
- -1 for a parameter at its upper bound
- -3 for a parameter at its lower bound
- -3.5 for a parameter that is out of bounds low (< lower bound)

Not all these possibilities will be used by all methods that use `bdmsk`.

The -1 and -3 are historical, and arose in the development of BASIC codes for Nash and Walker-Smith (1987) (This is now available for free download from archive.org. (https://archive.org/details/NLPE87plus). In particular, adding 2 to the `bdmsk` element gives 1 for an upper bound and -1 for a lower bound, simplifying the expression to decide if an optimization trial step will move away from a bound.

Proposed algorithmic approaches

Because masks (fixed parameters) reduce the dimensionality of the optimization problem, we can consider modifying the problem to the lower dimension space. This is Duncan Murdoch's suggestion, using

- `fn0(par0)` to be the initial user function of the full dimension parameter vector `par0`
- `fn1(par1)` to be the reduced or internal functin of the reduced dimension vector `par1`
- `par1 <- forward(par0)`
- `par0 <- inverse(par1)`

The major advantage of this approach is explicit dimension reduction. The main disadvantage is the effort of transformation at every step of an optimization.

An alternative is to use the `bdmsk` vector to **mask** the optimization search or adjustment vector, including gradients and (approximate) Hessian or Jacobian matrices. A 0 element of `bdmsk` "multiplies" any adjustment. The principal difficulty is to ensure we do not essentially divide by zero in applying any inverse Hessian. This approach avoids `forward`, `inverse` and `fn1`. However, it may hide the reduction in dimension, and caution is necessary in using the function and its derived gradient, Hessian and derived information.

Examples of use

For optimx

```
require(optimx)
```

```
## Loading required package: optimx
```

```
sq<-function(x){
   nn<-length(x)
   yy<-1:nn
   f<-sum((yy-x)^2)
   f
}
sq.g <- function(x){
   nn<-length(x)
   yy<-1:nn
   gg<- 2*(x - yy)
}
xx <- c(.3, 4)
uncans <- Rvmmin(xx, sq, sq.g)
proptimr(uncans)
```

```
## Result  uncans   proposes optimum function value = 0  at parameters
## [1] 1 2
## After  4  fn evals, and  3  gr evals
## Termination code is  2 : Rvmminu appears to have converged
## ----------------------------------------
```

```
mybm <- c(0,1) # fix parameter 1
cans <- Rvmmin(xx, sq, sq.g, bdmsk=mybm)
```

```
## trace= 0
```

```
proptimr(cans)
```

```
## Result  cans   proposes optimum function value = 0.49  at parameters
## [1] 0.3 2.0
## After  6  fn evals, and  4  gr evals
## Termination code is  2 : Rvmminb appears to have converged
## ----------------------------------------
```

```
require(nlsr)
```

```
## Loading required package: nlsr
```

```
weed <- 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)
ii <- 1:12
wdf <- data.frame(weed, ii)
weedux <- nlxb(weed~b1/(1+b2*exp(-b3*ii)), start=c(b1=200, b2=50, b3=0.3))
weedux
```

```
## residual sumsquares =  2.5873  on  12 observations
##     after  6    Jacobian and  6 function evaluations
##   name            coeff          SE       tstat      pval      gradient    JSingval
## b1               196.186         11.31       17.35  3.167e-08  -5.069e-11        1011
## b2               49.0916         1.688       29.08  3.284e-10  -2.192e-09      0.4605
```

```
## b3               0.31357      0.006863      45.69  5.768e-12    2.61e-07     0.04714
```

```
weedcx <- nlxb(weed~b1/(1+b2*exp(-b3*ii)), start=c(b1=200, b2=50, b3=0.3), masked=c("b1"))
weedcx
```

```
## residual sumsquares =  2.5873  on  12 observations
##     after  6    Jacobian and  6 function evaluations
##   name            coeff          SE       tstat      pval      gradient    JSingval
## b1               196.186         11.31      17.35  3.167e-08  -5.069e-11        1011
## b2               49.0916         1.688      29.08  3.284e-10  -2.192e-09      0.4605
## b3               0.31357      0.006863      45.69  5.768e-12    2.61e-07     0.04714
```

```
rfn <- function(bvec, weed=weed, ii=ii){
  res <- rep(NA, length(ii))
  for (i in ii){
    res[i]<- bvec[1]/(1+bvec[2]*exp(-bvec[3]*i))-weed[i]
  }
  res
}
weeduf <- nlfb(start=c(200, 50, 0.3),resfn=rfn,weed=weed, ii=ii,
               control=list(japprox="jacentral"))
weeduf
```

```
## residual sumsquares =  2.5873  on  12 observations
##     after  6    Jacobian and  6 function evaluations
##   name            coeff          SE       tstat      pval      gradient    JSingval
## p1               196.186         11.31      17.35  3.167e-08   1.378e-11        1011
## p2               49.0916         1.688      29.08  3.284e-10  -2.466e-09      0.4605
## p3               0.31357      0.006863      45.69  5.768e-12   5.067e-07     0.04714
```

```
weedcf <- nlfb(start=c(200, 50, 0.3),resfn=rfn,weed=weed, ii=ii, lower=c(200, 0, 0),
               upper=c(200, 100,100), control=list(japprox="jacentral"))
weedcf
```

```
## residual sumsquares =  2.6182  on  12 observations
##     after  4    Jacobian and  4 function evaluations
##   name            coeff          SE       tstat      pval      gradient    JSingval
## p1                  200U M         NA         NA         NA           0          NA
## p2               49.5108          1.12      44.21  8.421e-13  -2.888e-07        1022
## p3              0.311461      0.002278      136.8  1.073e-17   0.0001634      0.4569
```

An extensible bell-curve model

Package `nlraa` has a selfStart model `SSbell` (Archontoulis and Miguez (2013)) of which the formula is

$$y \approx ymax * exp(a * (x - xc)^2 + b * (x - xc)^3)$$

This is essentially the Gaussian bell curve with an additional cubic element in the exponential function. If we **fix** $b = 0$, then we have the usual Gaussian, and we can use the standard deviation $sigma$ of the variable x with xc equal to its mean and our parameters will be given approximately by

$$ymax = max(y)$$
$$a = -0.5/sigma^2$$
$$xc = mean(y)$$

We illustrate this in the following example.

```r
# BellX.R
library(nlraa)
require(ggplot2)
```

```
## Loading required package: ggplot2
```

```r
set.seed(1234)
x <- 1:20
y <- bell(x, 8, -0.0314, 0.000317, 13) + rnorm(length(x), 0, 0.5)
dat <- data.frame(x = x, y = y)
fit <- nls(y ~ SSbell(x, ymax, a, b, xc), data = dat)
fit
```

```
## Nonlinear regression model
##   model: y ~ SSbell(x, ymax, a, b, xc)
##    data: dat
##      ymax         a         b        xc
##  7.760143 -0.031131  0.000509 13.087298
##  residual sum-of-squares: 4.52
##
## Number of iterations to convergence: 6
## Achieved convergence tolerance: 5.66e-06
```

```r
gfrm<- y ~ ymax * exp(a *(x - xc)^2)
gfrmx<- y ~ ymax * exp(a *(x - xc)^2 + b*(x - xc)^3)
stgauss<-c(ymax=max(y), a=-0.5/(sd(x)^2), xc=mean(x))
cat("stgauss:"); print(stgauss)
```

```
## stgauss:
```

```
##      ymax         a        xc
##  7.787390 -0.014286 10.500000
```

```r
st2<-c(ymax=8, a= 0.03, xc= 13)
st3<-c(ymax=8, a= 0.03, b=0, xc= 13)
fit2 <- nls(gfrm, start=st2, data=dat, trace=TRUE)
```

```
## 4.9372e+05 (4.02e+01): par = (8 0.03 13)
## 6154.6     (4.38e+00): par = (0.98813 0.030012 12.821)
## 1234.0     (1.98e+00): par = (0.99099 0.028622 11.633)
## 311.09     (1.50e+00): par = (1.7487 0.012083 11.581)
## 101.19     (3.03e+00): par = (4.4218 -0.013307 10.761)
## 66.601     (3.15e+00): par = (6.3851 -0.022334 15.968)
## 20.365     (1.83e+00): par = (6.3175 -0.021534 12.632)
## 5.5580     (4.18e-01): par = (7.6068 -0.030079 13.503)
## 4.7165     (2.95e-02): par = (7.7656 -0.031319 13.252)
## 4.7125     (8.56e-04): par = (7.7891 -0.031571 13.258)
## 4.7125     (2.95e-05): par = (7.7891 -0.031572 13.257)
## 4.7125     (1.13e-06): par = (7.7891 -0.031572 13.257)
```

```r
summary(fit2)
```

```
##
## Formula: y ~ ymax * exp(a * (x - xc)^2)
##
## Parameters:
##      Estimate Std. Error t value Pr(>|t|)
```

```
## ymax  7.78914    0.24694    31.5  < 2e-16 ***
## a    -0.03157    0.00247   -12.8  3.8e-10 ***
## xc   13.25700    0.14687    90.3  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.527 on 17 degrees of freedom
##
## Number of iterations to convergence: 11
## Achieved convergence tolerance: 1.13e-06
```

```
# Use the selfStart model
xfrm <- y ~ SSbell(x, ymax, a, b, xc)
library(nlsr)
fx2mx <- nlxb(xfrm, start=st3, data=dat, lower=c(0, -1e5, 0, 0),
            upper=c(1e5, 1e5, 0, 1e5), trace=TRUE, control=list(japprox="jacentral"))
```

```
## The following parameters are masked:[1] "b"
## Using approximation  jacentral
##       is.character(ctrl$japprox)= TRUE
## No backtrack
## The following parameters are masked:[1] "b"
## 00lamda: 1e-04  SS= 493716 ( NA ) at  ymax = 8  a = 0.03  b = 0  xc = 13 f/j  1 / 0
## <<lamda: 4e-05  SS= 6154.7 ( 0.99956 ) at  ymax = 0.98814  a = 0.030012  b = 0  xc = 12.821 f/j  2 /
## <<lamda: 1.6e-05  SS= 1234 ( 0.96446 ) at  ymax = 0.99099  a = 0.028622  b = 0  xc = 11.633 f/j  3 /
## <<lamda: 6.4e-06  SS= 311.09 ( 0.81427 ) at  ymax = 1.7487  a = 0.012083  b = 0  xc = 11.581 f/j  4 /
## <<lamda: 2.56e-06  SS= 101.19 ( 0.80911 ) at  ymax = 4.4218  a = -0.013306  b = 0  xc = 10.761 f/j  5
## <<lamda: 1.024e-06  SS= 66.601 ( 0.79802 ) at  ymax = 6.3851  a = -0.022335  b = 0  xc = 15.968 f/j
## <<lamda: 4.096e-07  SS= 20.365 ( 0.76055 ) at  ymax = 6.3175  a = -0.021534  b = 0  xc = 12.632 f/j
## <<lamda: 1.6384e-07  SS= 5.558 ( 0.55992 ) at  ymax = 7.6068  a = -0.030079  b = 0  xc = 13.503 f/j
## <<lamda: 6.5536e-08  SS= 4.7165 ( 0.33555 ) at  ymax = 7.7656  a = -0.031319  b = 0  xc = 13.252 f/j
## <<lamda: 2.6214e-08  SS= 4.7125 ( 0.023492 ) at  ymax = 7.7891  a = -0.031571  b = 0  xc = 13.258 f/
## <<lamda: 1.0486e-08  SS= 4.7125 ( 0.00077672 ) at  ymax = 7.7891  a = -0.031572  b = 0  xc = 13.257
## <<lamda: 4.1943e-09  SS= 4.7125 ( 2.6324e-05 ) at  ymax = 7.7891  a = -0.031572  b = 0  xc = 13.257
## <<lamda: 1.6777e-09  SS= 4.7125 ( 9.1042e-07 ) at  ymax = 7.7891  a = -0.031572  b = 0  xc = 13.257
```

```
fx2mx
```

```
## residual sumsquares =  4.7125  on  20 observations
##     after  13    Jacobian and  13 function evaluations
##   name            coeff          SE       tstat      pval      gradient    JSingval
## ymax             7.78914        0.2469       31.54   1.58e-16  -4.097e-10       265.5
## a             -0.0315725      0.002471      -12.78   3.84e-10    1.84e-05       3.589
## b                     OU M          NA          NA         NA           0          NA
## xc                13.257        0.1469       90.27  3.076e-24   1.543e-07       2.131
```

```
# Or the formula for the same model with Jacobian approximation
fx2mgc <- nlxb(gfrmx, start=st3, data=dat, lower=c(0, -1e5, 0, 0),
             upper=c(1e5, 1e5, 0, 1e5), trace=TRUE, control=list(japprox="jacentral"))
```

```
## The following parameters are masked:[1] "b"
## Using approximation  jacentral
##       is.character(ctrl$japprox)= TRUE
## No backtrack
## The following parameters are masked:[1] "b"
## 00lamda: 1e-04  SS= 493716 ( NA ) at  ymax = 8  a = 0.03  b = 0  xc = 13 f/j  1 / 0
## <<lamda: 4e-05  SS= 6154.7 ( 0.99956 ) at  ymax = 0.98814  a = 0.030012  b = 0  xc = 12.821 f/j  2 /
```

```
## <<lamda: 1.6e-05  SS= 1234 ( 0.96446 ) at  ymax = 0.99099  a = 0.028622  b = 0  xc = 11.633 f/j  3 /
## <<lamda: 6.4e-06  SS= 311.09 ( 0.81427 ) at  ymax = 1.7487  a = 0.012083  b = 0  xc = 11.581 f/j  4 /
## <<lamda: 2.56e-06  SS= 101.19 ( 0.80911 ) at  ymax = 4.4218  a = -0.013306  b = 0  xc = 10.761 f/j  5
## <<lamda: 1.024e-06  SS= 66.601 ( 0.79802 ) at  ymax = 6.3851  a = -0.022335  b = 0  xc = 15.968 f/j
## <<lamda: 4.096e-07  SS= 20.365 ( 0.76055 ) at  ymax = 6.3175  a = -0.021534  b = 0  xc = 12.632 f/j
## <<lamda: 1.6384e-07  SS= 5.558 ( 0.55992 ) at  ymax = 7.6068  a = -0.030079  b = 0  xc = 13.503 f/j
## <<lamda: 6.5536e-08  SS= 4.7165 ( 0.33555 ) at  ymax = 7.7656  a = -0.031319  b = 0  xc = 13.252 f/j
## <<lamda: 2.6214e-08  SS= 4.7125 ( 0.023492 ) at  ymax = 7.7891  a = -0.031571  b = 0  xc = 13.258 f/j
## <<lamda: 1.0486e-08  SS= 4.7125 ( 0.00077672 ) at  ymax = 7.7891  a = -0.031572  b = 0  xc = 13.257 i
## <<lamda: 4.1943e-09  SS= 4.7125 ( 2.6324e-05 ) at  ymax = 7.7891  a = -0.031572  b = 0  xc = 13.257 i
## <<lamda: 1.6777e-09  SS= 4.7125 ( 9.1042e-07 ) at  ymax = 7.7891  a = -0.031572  b = 0  xc = 13.257 i
```

```
fx2mgc
```

```
## residual sumsquares =  4.7125  on  20 observations
##     after  13    Jacobian and  13 function evaluations
##   name            coeff          SE       tstat      pval      gradient    JSingval
## ymax             7.78914        0.2469      31.54   1.58e-16  -4.097e-10       265.5
## a             -0.0315725      0.002471     -12.78   3.84e-10    1.84e-05       3.589
## b                    0U M          NA         NA         NA           0          NA
## xc                13.257        0.1469      90.27  3.076e-24   1.543e-07       2.131
```

```
# Or the formula and analytic derivatives
fx2mg <- nlxb(gfrmx, start=st3, data=dat, lower=c(0, -1e5, 0, 0),
              upper=c(1e5, 1e5, 0, 1e5), trace=TRUE)
```

```
## The following parameters are masked:[1] "b"
## No backtrack
## The following parameters are masked:[1] "b"
## 00lamda: 1e-04  SS= 493716 ( NA ) at  ymax = 8  a = 0.03  b = 0  xc = 13 f/j  1 / 0
## <<lamda: 4e-05  SS= 6154.7 ( 0.99956 ) at  ymax = 0.98814  a = 0.030012  b = 0  xc = 12.821 f/j  2 /
## <<lamda: 1.6e-05  SS= 1234 ( 0.96446 ) at  ymax = 0.99099  a = 0.028622  b = 0  xc = 11.633 f/j  3 /
## <<lamda: 6.4e-06  SS= 311.09 ( 0.81427 ) at  ymax = 1.7487  a = 0.012083  b = 0  xc = 11.581 f/j  4 /
## <<lamda: 2.56e-06  SS= 101.19 ( 0.80911 ) at  ymax = 4.4218  a = -0.013306  b = 0  xc = 10.761 f/j  5
## <<lamda: 1.024e-06  SS= 66.601 ( 0.79802 ) at  ymax = 6.3851  a = -0.022335  b = 0  xc = 15.968 f/j
## <<lamda: 4.096e-07  SS= 20.365 ( 0.76055 ) at  ymax = 6.3175  a = -0.021534  b = 0  xc = 12.632 f/j
## <<lamda: 1.6384e-07  SS= 5.558 ( 0.55992 ) at  ymax = 7.6068  a = -0.030079  b = 0  xc = 13.503 f/j
## <<lamda: 6.5536e-08  SS= 4.7165 ( 0.33555 ) at  ymax = 7.7656  a = -0.031319  b = 0  xc = 13.252 f/j
## <<lamda: 2.6214e-08  SS= 4.7125 ( 0.023492 ) at  ymax = 7.7891  a = -0.031571  b = 0  xc = 13.258 f/j
## <<lamda: 1.0486e-08  SS= 4.7125 ( 0.00077672 ) at  ymax = 7.7891  a = -0.031572  b = 0  xc = 13.257 i
## <<lamda: 4.1943e-09  SS= 4.7125 ( 2.6324e-05 ) at  ymax = 7.7891  a = -0.031572  b = 0  xc = 13.257 i
## <<lamda: 1.6777e-09  SS= 4.7125 ( 9.1041e-07 ) at  ymax = 7.7891  a = -0.031572  b = 0  xc = 13.257 i
```

```
fx2mg
```

```
## residual sumsquares =  4.7125  on  20 observations
##     after  13    Jacobian and  13 function evaluations
##   name            coeff          SE       tstat      pval      gradient    JSingval
## ymax             7.78914        0.2469      31.54   1.58e-16  -3.035e-13       265.5
## a             -0.0315725      0.002471     -12.78   3.84e-10   1.838e-05       3.589
## b                    0U M          NA         NA         NA           0          NA
## xc                13.257        0.1469      90.27  3.076e-24   1.548e-07       2.131
```

```
# Display results together
pshort(fx2mx)
```

```
## fx2mx  -- ss= 4.7125 :  ymax = 7.7891  a = -0.031572  b = 0  xc = 13.257; 13 res/ 13 jac
```

```
pshort(fx2mgc)
```

```
## fx2mgc  -- ss= 4.7125 :  ymax = 7.7891  a = -0.031572  b = 0  xc = 13.257; 13 res/ 13 jac
```

```
pshort(fx2mg)
```

```
## fx2mg  -- ss= 4.7125 :  ymax = 7.7891  a = -0.031572  b = 0  xc = 13.257; 13 res/ 13 jac
```

References

Archontoulis, S. V., and Fernando Miguez. 2013. "Nonlinear Regression Models and Applications in Agricultural Research." *Agronomy Journal* 105 (January): 1. https://doi.org/10.2134/agronj2012.0506.

Nash, John C., and Mary Walker-Smith. 1987. *Nonlinear Parameter Estimation: An Integrated System in Basic*. Book. New York: Marcel Dekker.