Specifying Fixed Parameters

John C. Nash

2021-07-02

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.

Background

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

From `nlxb()` in package `nlsr`. (This approach was previously in defunct package `nlmrt`.)

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.

From `nlfb()` in `nlsr`.

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`

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 packages `optimx` and `optimz` 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 is outside the bounds.

From `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 setting the starting value to this fixed value, and 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.

Internal structures

`bdmsk` is the internal structure used in `Rcgmin` and `Rvmmin` 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 J C and Walker-Smith M (1987) **Nonlinear parameter estimation: and integrated system in BASIC** New York: Dekker. Now available for free download from archive.org. (https://archive.org/details/NLPE87plus). In particular, adding 2 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 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 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

More examples would be useful here.

```
require(optimx)
```

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

```
sq<-function(x){
   nn<-length(x)
   yy<-1:nn
   f<-sum((yy-x)^2)
#   cat("Fu=",f," at ")
#   print(x)
   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)
uncans
```

```
## $par
## [1] 1 2
##
## $value
## [1] 0
##
## $counts
## function gradient
##        4        3
##
## $convergence
## [1] 2
##
## $message
## [1] "Rvmminu appears to have converged"
```

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

```
## $par
## [1] 0.3 2.0
##
## $value
## [1] 0.49
##
## $counts
## function gradient
##        6        4
##
## $convergence
## [1] 2
##
## $message
## [1] "Rvmminb appears to have converged"
##
## $bdmsk
## [1] 0 1
```

```
require(nlsralt)
```

```
## Loading required package: nlsralt
```

```
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
```

```
## nlsr object: x
## residual sumsquares =  2.5873  on  12 observations
##     after  6    Jacobian and  7 function evaluations
##   name            coeff          SE       tstat      pval      gradient    JSingval
## b1               196.186         11.31       17.35  3.167e-08  -1.334e-10        1011
## b2               49.0916         1.688       29.08  3.284e-10  -3.589e-09      0.4605
## b3               0.31357      0.006863       45.69  5.768e-12    4.09e-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
```

```
## nlsr object: x
## residual sumsquares =  2.6182  on  12 observations
##     after  5    Jacobian and  6 function evaluations
##   name            coeff          SE       tstat      pval      gradient    JSingval
## b1                  200  M         NA         NA         NA           0        1022
## b2               49.5108            NA         NA         NA  -3.501e-10      0.4569
## b3              0.311461            NA         NA         NA  -7.631e-08           0
```

```
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)
weeduf
```

```
## nlsr object: x
## residual sumsquares =  2.5873  on  12 observations
##     after  6    Jacobian and  7 function evaluations
##   name            coeff          SE       tstat      pval      gradient    JSingval
## p_1              196.186         11.31       17.35  3.167e-08   5.505e-10        1011
## p_2              49.0916         1.688       29.08  3.284e-10   1.314e-09      0.4605
## p_3              0.31357      0.006863       45.69  5.768e-12  -5.034e-08     0.04714
```

```
weedcf <- nlfb(start=c(200, 50, 0.3),resfn=rfn,weed=weed, ii=ii, maskidx=c(1))
weedcf
```

```
## nlsr object: x
## residual sumsquares =  2.6182  on  12 observations
##     after  5    Jacobian and  6 function evaluations
##   name            coeff          SE       tstat      pval      gradient    JSingval
## p_1                 200  M         NA         NA         NA           0        1022
## p_2              49.5108            NA         NA         NA  -4.231e-13      0.4569
```

```
## p_3          0.311461          NA          NA          NA  -2.383e-10          0
```