Understanding nlsModel in the base R nls() function

Arkajyoti Bhattacharjee, Indian Institute of Technology, Kanpur
John C. Nash, University of Ottawa, Canada

05/06/2021

Objective

The R function `nls()` for nonlinear least squares modeling uses a structure where a "model" object named `m` is built from an expression in the `nls` call. This object contains a great deal of functionality in the form of data and functions that different parts of the `nls` infrastructure can use. In particular, `m` contains functions to determine if the estimation of the model is "converged" (actually if the process is "terminated") as well as computations of residuals and the search direction for the next set of parameters.

This article is an attempt to document and test the features of the nlsModel object `m`.

Location of the nlsModel code

The code is found in the file `./R-devel/src/library/stats/R/nls.R`

Note that there is a version `nlsModel.plinear` for use with partially linear models (?? a ref would be helpful here). The default case and the use of bounds constraints using the `algorithm` setting "port" are handled by `nlsModel`.

Here we will at time of writing (June 2021) only deal with the default case.

Generating an 'm'' object

Let us use the Croucher problem and/or the Hobbs weed infestation problem (Nash (1979), page 120) again.

```
# Croucher-example1.R -- https://walkingrandomly.com/?p=5254
# construct the data vectors using c()
xdata = c(-2,-1.64,-1.33,-0.7,0,0.45,1.2,1.64,2.32,2.9)
ydata = c(0.699369,0.700462,0.695354,1.03905,1.97389,2.41143,1.91091,0.919576,-0.730975,-1.42001)

# some starting values
p1 = 1
p2 = 0.2

Cdata<-data.frame(xdata, ydata)

# do the fit
## Try numericDeriv() function
Cform <- ydata ~ p1*cos(p2*xdata) + p2*sin(p1*xdata)
Ctheta<-c("p1","p2")
```

```
## nmodc1<-nlspkg::nlsModel(form=Cform, data=Cdata, start=c(p1,p2), wts=NULL, upper=NULL, scaleOffset =
## print(str(nmodc1))
```

```
# 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

# A simple starting vector -- must have named parameters for nlxb, nls, wrapnlsr.
start1  <-  c(b1=1, b2=1, b3=1)
eunsc  <-   y ~ b1/(1+b2*exp(-b3*tt))
str(eunsc)
```

```
## Class 'formula'  language y ~ b1/(1 + b2 * exp(-b3 * tt))
##   ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
```

```
# Can we convert a string form of this "model" to a formula
ceunsc <- " y ~ b1/(1+b2*exp(-b3*tt))"
str(ceunsc)
```

```
##  chr " y ~ b1/(1+b2*exp(-b3*tt))"
```

```
weeddata1  <-  data.frame(y=ydat, tt=tdat)

## Now ready to try things out.
library(nlsalt) # ?? needed because base R does not export nlsModel()
```

```
## Registered S3 methods overwritten by 'nlsalt':
##   method             from
##   anova.nls          stats
##   coef.nls           stats
##   confint.nls        stats
##   deviance.nls       stats
##   df.residual.nls    stats
##   fitted.nls         stats
##   formula.nls        stats
##   logLik.nls         stats
##   nobs.nls           stats
##   plot.profile.nls   stats
##   predict.nls        stats
##   print.nls          stats
##   print.summary.nls  stats
##   profile.nls        stats
##   residuals.nls      stats
##   summary.nls        stats
##   vcov.nls           stats
##   weights.nls        stats
```

```
##
## Attaching package: 'nlsalt'
```

```
## The following objects are masked from 'package:stats':
##
##     asOneSidedFormula, getInitial, nlminb, nls, nls.control,
##     NLSstAsymptotic, NLSstClosestX, NLSstLfAsymptote, NLSstRtAsymptote,
##     numericDeriv, selfStart, setNames, sortedXyData
```

```
nmod1<-nlsModel(form=eunsc, data=weeddata1, start=start1, wts=NULL, upper=NULL, scaleOffset = 0, nDcent
```

```
## par:1  1  1
##  [1] 0.7310586 0.8807971 0.9525741 0.9820138 0.9933071 0.9975274 0.9990889
##  [8] 0.9996646 0.9998766 0.9999546 0.9999833 0.9999939
## attr(,"gradient")
##              [,1]          [,2]         [,3]
##  [1,] 0.7310586 -1.966119e-01 1.966119e-01
##  [2,] 0.8807971 -1.049936e-01 2.099872e-01
##  [3,] 0.9525741 -4.517667e-02 1.355300e-01
##  [4,] 0.9820138 -1.766271e-02 7.065082e-02
##  [5,] 0.9933072 -6.648056e-03 3.324027e-02
##  [6,] 0.9975274 -2.466522e-03 1.479905e-02
##  [7,] 0.9990890 -9.102300e-04 6.371543e-03
##  [8,] 0.9996646 -3.352463e-04 2.681896e-03
##  [9,] 0.9998766 -1.233816e-04 1.110412e-03
## [10,] 0.9999546 -4.540384e-05 4.539490e-04
## [11,] 0.9999833 -1.670420e-05 1.837090e-04
## [12,] 0.9999939 -6.139278e-06 7.373095e-05
```

```
str(nmod1)
```

```
## List of 16
##  $ resid     :function ()
##   ..- attr(*, "srcref")= 'srcref' int [1:8] 127 15 127 30 22 37 1668 1668
##   .. ..- attr(*, "srcfile")=Classes 'srcfilealias', 'srcfile' <environment: 0x55c201e41920>
##  $ fitted    :function ()
##   ..- attr(*, "srcref")= 'srcref' int [1:8] 128 16 128 29 23 36 1669 1669
##   .. ..- attr(*, "srcfile")=Classes 'srcfilealias', 'srcfile' <environment: 0x55c201e41920>
##  $ formula   :function ()
##   ..- attr(*, "srcref")= 'srcref' int [1:8] 129 17 129 31 24 38 1670 1670
##   .. ..- attr(*, "srcfile")=Classes 'srcfilealias', 'srcfile' <environment: 0x55c201e41920>
##  $ deviance  :function ()
##   ..- attr(*, "srcref")= 'srcref' int [1:8] 130 18 130 31 25 38 1671 1671
##   .. ..- attr(*, "srcfile")=Classes 'srcfilealias', 'srcfile' <environment: 0x55c201e41920>
##  $ lhs       :function ()
##   ..- attr(*, "srcref")= 'srcref' int [1:8] 131 13 131 26 20 33 1672 1672
##   .. ..- attr(*, "srcfile")=Classes 'srcfilealias', 'srcfile' <environment: 0x55c201e41920>
##  $ gradient  :function ()
##   ..- attr(*, "srcref")= 'srcref' int [1:8] 132 18 132 57 25 64 1673 1673
##   .. ..- attr(*, "srcfile")=Classes 'srcfilealias', 'srcfile' <environment: 0x55c201e41920>
##  $ conv      :function ()
##   ..- attr(*, "srcref")= 'srcref' int [1:8] 133 14 133 34 21 41 1674 1674
##   .. ..- attr(*, "srcfile")=Classes 'srcfilealias', 'srcfile' <environment: 0x55c201e41920>
##  $ incr      :function ()
##   ..- attr(*, "srcref")= 'srcref' int [1:8] 134 14 134 42 21 49 1675 1675
##   .. ..- attr(*, "srcfile")=Classes 'srcfilealias', 'srcfile' <environment: 0x55c201e41920>
##  $ setVarying:function (vary = rep_len(TRUE, np))
##   ..- attr(*, "srcref")= 'srcref' int [1:8] 135 20 158 7 27 14 1676 1699
##   .. ..- attr(*, "srcfile")=Classes 'srcfilealias', 'srcfile' <environment: 0x55c201e41920>
##  $ setPars   :function (newPars)
##   ..- attr(*, "srcref")= 'srcref' int [1:8] 159 17 166 7 24 14 1700 1707
##   .. ..- attr(*, "srcfile")=Classes 'srcfilealias', 'srcfile' <environment: 0x55c201e41920>
##  $ getPars   :function ()
##   ..- attr(*, "srcref")= 'srcref' int [1:8] 167 17 167 36 24 43 1708 1708
##   .. ..- attr(*, "srcfile")=Classes 'srcfilealias', 'srcfile' <environment: 0x55c201e41920>
##  $ getAllPars:function ()
##   ..- attr(*, "srcref")= 'srcref' int [1:8] 168 20 168 39 27 46 1709 1709
##   .. ..- attr(*, "srcfile")=Classes 'srcfilealias', 'srcfile' <environment: 0x55c201e41920>
##  $ getEnv    :function ()
##   ..- attr(*, "srcref")= 'srcref' int [1:8] 169 16 169 29 23 36 1710 1710
##   .. ..- attr(*, "srcfile")=Classes 'srcfilealias', 'srcfile' <environment: 0x55c201e41920>
##  $ trace     :function ()
##   ..- attr(*, "srcref")= 'srcref' int [1:8] 170 15 176 7 22 14 1711 1717
##   .. ..- attr(*, "srcfile")=Classes 'srcfilealias', 'srcfile' <environment: 0x55c201e41920>
##  $ Rmat      :function ()
##   ..- attr(*, "srcref")= 'srcref' int [1:8] 177 14 177 32 21 39 1718 1718
##   .. ..- attr(*, "srcfile")=Classes 'srcfilealias', 'srcfile' <environment: 0x55c201e41920>
##  $ predict   :function (newdata = list(), qr = FALSE)
##   ..- attr(*, "srcref")= 'srcref' int [1:8] 178 17 179 56 24 56 1719 1720
##   .. ..- attr(*, "srcfile")=Classes 'srcfilealias', 'srcfile' <environment: 0x55c201e41920>
##  - attr(*, "class")= chr "nlsModel"
```

```
ls.str(nmod1)
```

```
## conv : function ()
## deviance : function ()
## fitted : function ()
## formula : function ()
## getAllPars : function ()
## getEnv : function ()
## getPars : function ()
## gradient : function ()
## incr : function ()
## lhs : function ()
## predict : function (newdata = list(), qr = FALSE)
## resid : function ()
## Rmat : function ()
## setPars : function (newPars)
## setVarying : function (vary = rep_len(TRUE, np))
## trace : function ()
```

```
print(nmod1)
```

```
## $resid
## function() resid
## <bytecode: 0x55c201ed5fd0>
## <environment: 0x55c201f0b7f8>
##
## $fitted
## function() rhs
## <bytecode: 0x55c201ed4f68>
## <environment: 0x55c201f0b7f8>
##
## $formula
## function() form
## <bytecode: 0x55c201ed7da0>
## <environment: 0x55c201f0b7f8>
##
```

```
## $deviance
## function() dev
## <bytecode: 0x55c201ed6d70>
## <environment: 0x55c201f0b7f8>
##
## $lhs
## function() lhs
## <bytecode: 0x55c201ed9ba8>
## <environment: 0x55c201f0b7f8>
##
## $gradient
## function() .swts * attr(rhs, "gradient")
## <bytecode: 0x55c201ed8bb0>
## <environment: 0x55c201f0b7f8>
##
## $conv
## function() convCrit()
## <bytecode: 0x55c201ed8018>
## <environment: 0x55c201f0b7f8>
##
## $incr
## function() qr.coef(QR, resid)
## <bytecode: 0x55c201edb390>
## <environment: 0x55c201f0b7f8>
##
## $setVarying
## function(vary = rep_len(TRUE, np)) {
##                  np <- length(useParams)
##       useParams <<- useP <-
##                      if(is.character(vary)) {
##                          temp <- logical(np)
##                          temp[unlist(ind[vary])] <- TRUE
##                          temp
##                      } else if(is.logical(vary) && length(vary) != np)
##                          stop("setVarying : 'vary' length must match length of parameters")
##                      else
##                          vary # envir = thisEnv
##       gradCall[[length(gradCall) - 1L]] <<- useP
##       if(all(useP)) {
##           setPars <<- setPars.noVarying
##           getPars <<- getPars.noVarying
##           getRHS  <<-  getRHS.noVarying
##           npar    <<- length(useP)
##       } else {
##           setPars <<- setPars.varying
##           getPars <<- getPars.varying
##           getRHS  <<-  getRHS.varying
##           npar    <<- sum(useP)
##       }
##       }
## <bytecode: 0x55c201eddb00>
## <environment: 0x55c201f0b7f8>
##
## $setPars
## function(newPars) {
##       setPars(newPars)
##       resid <<- .swts * (lhs - (rhs <<- getRHS())) # envir = thisEnv {2 x}
##       dev   <<- sum(resid^2) # envir = thisEnv
##       if(length(gr <- attr(rhs, "gradient")) == 1L) gr <- c(gr)
##       QR <<- qr(.swts * gr) # envir = thisEnv
##       (QR$rank < min(dim(QR$qr))) # to catch the singular gradient matrix
##       }
## <bytecode: 0x55c201ef0718>
## <environment: 0x55c201f0b7f8>
##
## $getPars
## function() getPars()
## <bytecode: 0x55c201eff2a8>
## <environment: 0x55c201f0b7f8>
##
## $getAllPars
## function() getPars()
## <bytecode: 0x55c201efe828>
## <environment: 0x55c201f0b7f8>
##
## $getEnv
## function() env
## <bytecode: 0x55c201efdda8>
## <environment: 0x55c201f0b7f8>
##
## $trace
## function() {
##       d <- getOption("digits")
##       cat(sprintf("%-*s (%.2e): par = (%s)\n", d+4L+2L*(scaleOffset > 0),
##               formatC(dev, digits=d, flag="#"),
##               convCrit(),
##               paste(vapply(getPars(), format, ""), collapse=" ")))
##       }
## <bytecode: 0x55c201f00780>
## <environment: 0x55c201f0b7f8>
##
## $Rmat
## function() qr.R(QR)
## <bytecode: 0x55c201f08c50>
## <environment: 0x55c201f0b7f8>
##
## $predict
## function(newdata = list(), qr = FALSE)
##                 eval(form[[3L]], as.list(newdata), env)
## <bytecode: 0x55c201f07ad0>
## <environment: 0x55c201f0b7f8>
##
## attr(,"class")
## [1] "nlsModel"
```

Nash, John C. 1979. *Compact Numerical Methods for Computers : Linear Algebra and Function Minimisation.* Book. Hilger: Bristol.