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`.

ISSUES

- Why is there only `upper` in the arguments of nlsModel(), i.e., no `lower`? For the default algorithm, upper makes no sense either.
- Why do we have Varying and noVarying versions of some functions? And with so many attributes?

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")
Cstart<-c(p1=p1, p2=p2)


nmodc1<-nlspkg::nlsModel(form=Cform, data=Cdata, start=Cstart, wts=NULL, upper=NULL, scaleOffset = 0, nl
```

```
## Registered S3 methods overwritten by 'nlspkg':
##   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
```

```
print(str(nmodc1))
```

```
## List of 16
##  $ resid     :function ()
##  $ fitted    :function ()
##  $ formula   :function ()
##  $ deviance  :function ()
##  $ lhs       :function ()
##  $ gradient  :function ()
##  $ conv      :function ()
##  $ incr      :function ()
##  $ setVarying:function (vary = rep_len(TRUE, np))
##  $ setPars   :function (newPars)
##  $ getPars   :function ()
##  $ getAllPars:function ()
##  $ getEnv    :function ()
##  $ trace     :function ()
##  $ Rmat      :function ()
##  $ predict   :function (newdata = list(), qr = FALSE)
##  - attr(*, "class")= chr "nlsModel"
## NULL
```

```
# nmodc1a<-nlsalt::nlsModel(form=Cform, data=Cdata, start=Cstart, wts=NULL, upper=NULL, scaleOffset = 0
# print(str(nmodc1a))
```

```
# 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         nlspkg
##   coef.nls          nlspkg
##   confint.nls       nlspkg
##   deviance.nls      nlspkg
##   df.residual.nls   nlspkg
##   fitted.nls        nlspkg
##   formula.nls       nlspkg
##   logLik.nls        nlspkg
##   nobs.nls          nlspkg
##   plot.profile.nls  nlspkg
##   predict.nls       nlspkg
##   print.nls         nlspkg
##   print.summary.nls nlspkg
##   profile.nls       nlspkg
##   residuals.nls     nlspkg
##   summary.nls       nlspkg
##   vcov.nls          nlspkg
##   weights.nls       nlspkg
```

```
##
## 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
```

```
nmodh1<-nlsModel(form=eunsc, data=weeddata1, start=start1, wts=NULL, upper=NULL, scaleOffset = 0, nDcen
```

```
## Running nlsModel -- just set getPars
```

```
## function ()
##  - attr(*, "srcref")= 'srcref' int [1:8] 38 26 38 65 26 65 1579 1579
##   ..- attr(*, "srcfile")=Classes 'srcfilealias', 'srcfile' <environment: 0x55d194fd5dc0>
## NULL
## 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(nmodh1)
```

```
## List of 16
##  $ resid     :function ()
##   ..- attr(*, "srcref")= 'srcref' int [1:8] 129 15 129 30 22 37 1670 1670
##   .. ..- attr(*, "srcfile")=Classes 'srcfilealias', 'srcfile' <environment: 0x55d194fd5dc0>
##  $ fitted    :function ()
##   ..- attr(*, "srcref")= 'srcref' int [1:8] 130 16 130 29 23 36 1671 1671
##   .. ..- attr(*, "srcfile")=Classes 'srcfilealias', 'srcfile' <environment: 0x55d194fd5dc0>
##  $ formula   :function ()
##   ..- attr(*, "srcref")= 'srcref' int [1:8] 131 17 131 31 24 38 1672 1672
##   .. ..- attr(*, "srcfile")=Classes 'srcfilealias', 'srcfile' <environment: 0x55d194fd5dc0>
##  $ deviance  :function ()
##   ..- attr(*, "srcref")= 'srcref' int [1:8] 132 18 132 31 25 38 1673 1673
##   .. ..- attr(*, "srcfile")=Classes 'srcfilealias', 'srcfile' <environment: 0x55d194fd5dc0>
##  $ lhs       :function ()
##   ..- attr(*, "srcref")= 'srcref' int [1:8] 133 13 133 26 20 33 1674 1674
##   .. ..- attr(*, "srcfile")=Classes 'srcfilealias', 'srcfile' <environment: 0x55d194fd5dc0>
##  $ gradient  :function ()
##   ..- attr(*, "srcref")= 'srcref' int [1:8] 134 18 134 57 25 64 1675 1675
##   .. ..- attr(*, "srcfile")=Classes 'srcfilealias', 'srcfile' <environment: 0x55d194fd5dc0>
##  $ conv      :function ()
##   ..- attr(*, "srcref")= 'srcref' int [1:8] 135 14 135 34 21 41 1676 1676
##   .. ..- attr(*, "srcfile")=Classes 'srcfilealias', 'srcfile' <environment: 0x55d194fd5dc0>
##  $ incr      :function ()
##   ..- attr(*, "srcref")= 'srcref' int [1:8] 136 14 136 42 21 49 1677 1677
##   .. ..- attr(*, "srcfile")=Classes 'srcfilealias', 'srcfile' <environment: 0x55d194fd5dc0>
##  $ setVarying:function (vary = rep_len(TRUE, np))
##   ..- attr(*, "srcref")= 'srcref' int [1:8] 137 20 160 7 27 14 1678 1701
##   .. ..- attr(*, "srcfile")=Classes 'srcfilealias', 'srcfile' <environment: 0x55d194fd5dc0>
##  $ setPars   :function (newPars)
##   ..- attr(*, "srcref")= 'srcref' int [1:8] 161 17 168 7 24 14 1702 1709
##   .. ..- attr(*, "srcfile")=Classes 'srcfilealias', 'srcfile' <environment: 0x55d194fd5dc0>
```

```
##  $ getPars   :function ()
##   ..- attr(*, "srcref")= 'srcref' int [1:8] 169 17 169 36 24 43 1710 1710
##   .. ..- attr(*, "srcfile")=Classes 'srcfilealias', 'srcfile' <environment: 0x55d194fd5dc0>
##  $ getAllPars:function ()
##   ..- attr(*, "srcref")= 'srcref' int [1:8] 170 20 170 39 27 46 1711 1711
##   .. ..- attr(*, "srcfile")=Classes 'srcfilealias', 'srcfile' <environment: 0x55d194fd5dc0>
##  $ getEnv    :function ()
##   ..- attr(*, "srcref")= 'srcref' int [1:8] 171 16 171 29 23 36 1712 1712
##   .. ..- attr(*, "srcfile")=Classes 'srcfilealias', 'srcfile' <environment: 0x55d194fd5dc0>
##  $ trace     :function ()
##   ..- attr(*, "srcref")= 'srcref' int [1:8] 172 15 178 7 22 14 1713 1719
##   .. ..- attr(*, "srcfile")=Classes 'srcfilealias', 'srcfile' <environment: 0x55d194fd5dc0>
##  $ Rmat      :function ()
##   ..- attr(*, "srcref")= 'srcref' int [1:8] 179 14 179 32 21 39 1720 1720
##   .. ..- attr(*, "srcfile")=Classes 'srcfilealias', 'srcfile' <environment: 0x55d194fd5dc0>
##  $ predict   :function (newdata = list(), qr = FALSE)
##   ..- attr(*, "srcref")= 'srcref' int [1:8] 180 17 181 56 24 56 1721 1722
##   .. ..- attr(*, "srcfile")=Classes 'srcfilealias', 'srcfile' <environment: 0x55d194fd5dc0>
##  - attr(*, "class")= chr "nlsModel"
```

```
ls.str(nmodh1)
```

```
## 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(nmodh1)
```

```
## $resid
## function() resid
## <bytecode: 0x55d195072ac8>
## <environment: 0x55d1950d5c28>
##
## $fitted
## function() rhs
## <bytecode: 0x55d195079df8>
## <environment: 0x55d1950d5c28>
##
## $formula
## function() form
## <bytecode: 0x55d195078d20>
## <environment: 0x55d1950d5c28>
```

```
##
## $deviance
## function() dev
## <bytecode: 0x55d19507b880>
## <environment: 0x55d1950d5c28>
##
## $lhs
## function() lhs
## <bytecode: 0x55d19507e488>
## <environment: 0x55d1950d5c28>
##
## $gradient
## function() .swts * attr(rhs, "gradient")
## <bytecode: 0x55d19507d3b0>
## <environment: 0x55d1950d5c28>
##
## $conv
## function() convCrit()
## <bytecode: 0x55d1950805a0>
## <environment: 0x55d1950d5c28>
##
## $incr
## function() qr.coef(QR, resid)
## <bytecode: 0x55d19507f9d0>
## <environment: 0x55d1950d5c28>
##
## $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: 0x55d195081f80>
## <environment: 0x55d1950d5c28>
##
```

```
## $setPars
## function(newPars) {
##       setPars(newPars)
##       resid <<- .swts * (lhs - (rhs <<- getRHS())) # envir = thisEnv {2 x}
##       dev   <<- sum(resid^2) # envir = thisEnv
##       if(length(gr <- attr(rhs, "gradient")) == 1L) gr <- c(gr)
##       QR <<- qr(.swts * gr) # envir = thisEnv
##       (QR$rank < min(dim(QR$qr))) # to catch the singular gradient matrix
##       }
## <bytecode: 0x55d1950b99c0>
## <environment: 0x55d1950d5c28>
##
## $getPars
## function() getPars()
## <bytecode: 0x55d1950c8018>
## <environment: 0x55d1950d5c28>
##
## $getAllPars
## function() getPars()
## <bytecode: 0x55d1950cb3c8>
## <environment: 0x55d1950d5c28>
##
## $getEnv
## function() env
## <bytecode: 0x55d1950ca948>
## <environment: 0x55d1950d5c28>
##
## $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: 0x55d1950cd710>
## <environment: 0x55d1950d5c28>
##
## $Rmat
## function() qr.R(QR)
## <bytecode: 0x55d1950d2638>
## <environment: 0x55d1950d5c28>
##
## $predict
## function(newdata = list(), qr = FALSE)
##                  eval(form[[3L]], as.list(newdata), env)
## <bytecode: 0x55d1950d5320>
## <environment: 0x55d1950d5c28>
##
## attr(,"class")
## [1] "nlsModel"
```

Examining the nlsModel output

The output of `nlsModel` is an object of class attribute "nlsModel" with 16 elements. (??Is this always the case, or can the call cause elements to be added or not produced?)

resid()

This function computes the residuals in the current environment. (?? Is this .GlobalEnv? Can that environment be changed i.e., is it an argument to resid()?)

```
cat("Display Croucher residuals from nmodc1 as currently set:\n")
```

```
## Display Croucher residuals from nmodc1 as currently set:
```

```
print(nmodc1$resid())
```

```
##  [1] -0.03983251 -0.04670526 -0.07524643  0.17767754  0.97389000  1.32848416
##  [7]  0.75316421 -0.22663381 -1.77169083 -2.30432252
```

```
cat("Display Hobbs residuals from nmodc1 as currently set:\n")
```

```
## Display Hobbs residuals from nmodc1 as currently set:
```

```
print(nmodh1$resid())
```

```
##  [1]  4.576941  6.359203  8.685426 11.883986 16.075693 22.194473 30.443911
##  [8] 37.558335 49.156123 61.948045 74.995017 90.972006
```

fitted()

This function computes the fitted values of the nonlinear model in the current environment. The output includes an attribute, "gradient," which is actually the Jacobian matrix of which the k, j element is the

```
cat("Display Croucher fitted model values from nmodc1 as currently set:\n")
```

```
## Display Croucher fitted model values from nmodc1 as currently set:
```

```
print(nmodc1$fitted())
```

```
##  [1] 0.7392015 0.7471673 0.7706004 0.8613725 1.0000000 1.0829458 1.1577458
##  [8] 1.1462098 1.0407158 0.8843125
## attr(,"gradient")
##            [,1]       [,2]
##  [1,] 1.0875197 -1.6881341
##  [2,] 0.9693692 -1.5259328
##  [3,] 0.9013955 -1.3207711
##  [4,] 0.8831381 -0.7418979
##  [5,] 1.0000000  0.0000000
##  [6,] 1.0769930  0.3945202
##  [7,] 1.0583038  0.6467959
##  [8,] 0.9240078  0.4692800
##  [9,] 0.5782596 -0.3060353
## [10,] 0.2733069 -1.3500201
```

```
cat("Display Hobbs fitted model values from nmodc1 as currently set:\n")
```

```
## Display Hobbs fitted model values from nmodc1 as currently set:
```

```
print(nmodh1$fitted())
```

```
##  [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
```

formula()

```
cat("Display Croucher formula from nmodc1 as currently set:\n")
```

```
## Display Croucher formula from nmodc1 as currently set:
```

```
print(nmodc1$formula())
```

```
## ydata ~ p1 * cos(p2 * xdata) + p2 * sin(p1 * xdata)
```

```
cat("Display Hobbs formula from nmodc1 as currently set:\n")
```

```
## Display Hobbs formula from nmodc1 as currently set:
```

```
print(nmodh1$formula())
```

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

deviance()

```
cat("Display Croucher deviance from nmodc1 as currently set:\n")
```

```
## Display Croucher deviance from nmodc1 as currently set:
```

```
print(nmodc1$deviance())
```

```
## [1] 11.82174
```

```
cat("Display Hobbs deviance from nmodc1 as currently set:\n")
```

```
## Display Hobbs deviance from nmodc1 as currently set:
```

```
print(nmodh1$deviance())
```

```
## [1] 23520.58
```

lhs()

```
cat("Display Croucher lhs from nmodc1 as currently set:\n")
```

```
## Display Croucher lhs from nmodc1 as currently set:
```

```
print(nmodc1$lhs())
```

```
##  [1]  0.699369  0.700462  0.695354  1.039050  1.973890  2.411430  1.910910
##  [8]  0.919576 -0.730975 -1.420010
```

```
cat("Display Hobbs lhs from nmodc1 as currently set:\n")
```

```
## Display Hobbs lhs from nmodc1 as currently set:
```

```
print(nmodh1$lhs())
```

```
##  [1]  5.308  7.240  9.638 12.866 17.069 23.192 31.443 38.558 50.156 62.948
## [11] 75.995 91.972
```

gradient()

```
cat("Display Croucher gradient from nmodc1 as currently set:\n")
```

```
## Display Croucher gradient from nmodc1 as currently set:
```

```
print(nmodc1$gradient())
```

```
##            [,1]       [,2]
##  [1,] 1.0875197 -1.6881341
##  [2,] 0.9693692 -1.5259328
##  [3,] 0.9013955 -1.3207711
##  [4,] 0.8831381 -0.7418979
##  [5,] 1.0000000  0.0000000
##  [6,] 1.0769930  0.3945202
##  [7,] 1.0583038  0.6467959
##  [8,] 0.9240078  0.4692800
##  [9,] 0.5782596 -0.3060353
## [10,] 0.2733069 -1.3500201
```

```
cat("Display Hobbs gradient from nmodc1 as currently set:\n")
```

```
## Display Hobbs gradient from nmodc1 as currently set:
```

```
print(nmodh1$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
```

conv()

```
cat("Display Croucher conv from nmodc1 as currently set:\n")
```

```
## Display Croucher conv from nmodc1 as currently set:
```

```
print(nmodc1$conv())
```

```
## [1] 0.680674
```

```
cat("Display Hobbs conv from nmodc1 as currently set:\n")
```

```
## Display Hobbs conv from nmodc1 as currently set:
```

```
print(nmodh1$conv())
```

```
## [1] 2.076871
```

incr()

?? This function appears to return the current search direction from the Gauss-Newton equations. We need to be very CAREFUL as we augment the `nls()` procedure to allow for Levenberg-Marquardt stabilizations and some different approaches to the solution iteration.

```
cat("Display Croucher incr from nmodc1 as currently set:\n")
```

```
## Display Croucher incr from nmodc1 as currently set:
```

```
print(nmodc1$incr())
```

```
## [1] 0.4951056 0.6604326
```

```
cat("Display Hobbs incr from nmodc1 as currently set:\n")
```

```
## Display Hobbs incr from nmodc1 as currently set:
```

```
print(nmodh1$incr())
```

```
## [1]   49.97713 -248.39739 -379.03452
```

setVarying(vary = rep_len(TRUE, np))

?? This seems to return the number of free (i.e., unconstrained) parameters in the current nonlinear model.

```
cat("Display Croucher setVarying from nmodc1 as currently set:\n")
```

```
## Display Croucher setVarying from nmodc1 as currently set:
```

```
print(nmodc1$setVarying())
```

```
## [1] 2
```

```
cat("Display Hobbs setVarying from nmodc1 as currently set:\n")
```

```
## Display Hobbs setVarying from nmodc1 as currently set:
```

```
print(nmodh1$setVarying())
```

```
## [1] 3
```

setPars(newPars)

?? This function does not seem to execute. Listing it shows some quite involved operations on the weghts, the model, the deviance and the QR decomposition of the Jacobian.

```
newPars <- c(p1=0.95, p2=0.15)
cat("Display Croucher setPars(newPars) from nmodc1 as currently set:\n")
print(nmodc1$setPars(newPars)())
cat("Display Hobbs setPars(newPars) from nmodc1 as currently set:\n")
newPars<- c(b1=1.1, b2=1.4, b3=0.1)
print(nmodh1$setPars(newPars)())
```

getPars()

This function and getAllPars() are defined identically in nlsModel() as

```
        getPars = function() getPars(),
        getAllPars = function() getPars(),
```

within the assignment of the list elements in creating the "m" object at the end of `nlsModel()`.

Prior to this, we have

```
cat("Display Croucher getPars from nmodc1 as currently set:\n")
```

```
## Display Croucher getPars from nmodc1 as currently set:
```

```
print(nmodc1$fitted())
```

```
##  [1] 0.7392015 0.7471673 0.7706004 0.8613725 1.0000000 1.0829458 1.1577458
##  [8] 1.1462098 1.0407158 0.8843125
## attr(,"gradient")
##            [,1]       [,2]
##  [1,] 1.0875197 -1.6881341
##  [2,] 0.9693692 -1.5259328
##  [3,] 0.9013955 -1.3207711
##  [4,] 0.8831381 -0.7418979
##  [5,] 1.0000000  0.0000000
##  [6,] 1.0769930  0.3945202
##  [7,] 1.0583038  0.6467959
##  [8,] 0.9240078  0.4692800
##  [9,] 0.5782596 -0.3060353
## [10,] 0.2733069 -1.3500201
```

```
cat("Display Hobbs getPars from nmodc1 as currently set:\n")
```

```
## Display Hobbs getPars from nmodc1 as currently set:
```

```
print(nmodh1$getPars())
```

```
## b1 b2 b3
##  1  1  1
```

getAllPars()

This function returns all parameters for the current nonlinear model as vector of named quantities. It is NOT a list as shown in the example.

```
cat("Display Croucher getAllPars from nmodc1 as currently set:\n")
```

```
## Display Croucher getAllPars from nmodc1 as currently set:
print(str(nmodc1$getAllPars()))
```

```
##  Named num [1:2] 1 0.2
##  - attr(*, "names")= chr [1:2] "p1" "p2"
## NULL
```

```
print(nmodc1$getAllPars())
```

```
##  p1  p2
## 1.0 0.2
```

```
is.list(nmodc1$getAllPars())
```

```
## [1] FALSE
```

```
is.numeric(nmodc1$getAllPars())
```

```
## [1] TRUE
```

```
cat("Display Hobbs getAllPars from nmodc1 as currently set:\n")
```

```
## Display Hobbs getAllPars from nmodc1 as currently set:
```

```
print(nmodh1$getAllPars())
```

```
## b1 b2 b3
##  1  1  1
```

getEnv()

The `getEnv()` function returns the identifier of the current environment for the nonlinear model in focus. By using the `ls()` function we can display the names of the objects available in this environment.

```
cat("Display Croucher fitted model values from nmodc1 as currently set:\n")
```

```
## Display Croucher fitted model values from nmodc1 as currently set:
```

```
print(nmodc1$getEnv())
```

```
## <environment: 0x55d194174788>
```

```
cat("ls(nmodc1$getEnv():")
```

```
## ls(nmodc1$getEnv():
```

```
print(ls(nmodc1$getEnv()))
```

```
## [1] "p1"    "p2"    "xdata" "ydata"
```

```
cat("Display Hobbs fitted model values from nmodc1 as currently set:\n")
```

```
## Display Hobbs fitted model values from nmodc1 as currently set:
```

```
print(nmodh1$getEnv())
```

```
## <environment: 0x55d1950d5870>
```

```
print(ls(nmodh1$getEnv()))
```

```
## [1] "b1" "b2" "b3" "tt" "y"
```

trace()

The "trace" function seems to display information about the current state of the model estimation.

ISSUE: because there is a "trace" argument to nls() (and to functions in package `nlsr` and others), would it not be better to use `tracefn()`??

Output from the `trace()` appears to be the current evaluated `deviance` (essentially the residual sum of squares), then in brackets the value of the `convCrit()` function – the value that is compared to a convergence tolerance, and then the current parameters. In printing the evaluated function, NULL is appended. Is this because no return value is provided??

```
cat("Display Croucher trace from nmodc1 as currently set:\n")
```

```
## Display Croucher trace from nmodc1 as currently set:
```

```
print(nmodc1$trace())
```

```
## 11.82174    (6.81e-01): par = (1 0.2)
## NULL
```

```
cat("Display Hobbs trace from nmodc1 as currently set:\n")
```

```
## Display Hobbs trace from nmodc1 as currently set:
```

```
print(nmodh1$trace())
```

```
## 23520.58    (2.08e+00): par = (1 1 1)
## NULL
```

Rmat()

"Rmat" appears to return the current R matrix of the QR decomposition.

This could lead to some confusion if we build an augmented matrix for the Levenberg-Marquardt and LM + Nash variant of the stabilization. We may need to watch this carefully to see where it is used. A search shows ONLY usage within the packages `nlspkg` and `nlsalt` to compute `XtXinv`, to get the variance-covariance of the parameters in `summary.nls()`.

?? We need to decide how to generate `Rmat` for the purposes of getting this information. In particular, in running the estimation iteration, we augment the matrix, so it is important to think how to get the right information for `summary.nls`.

```
cat("Display Croucher Rmat from nmodc1 as currently set:\n")
```

```
## Display Croucher Rmat from nmodc1 as currently set:
```

```
print(nmodc1$Rmat())
```

```
##           [,1]     [,2]
## [1,] -2.873607 1.448954
## [2,]  0.000000 2.843251
```

```
cat("Display Hobbs Rmat from nmodc1 as currently set:\n")
```

```
## Display Hobbs Rmat from nmodc1 as currently set:
```

```
print(nmodh1$Rmat())
```

```
##           [,1]       [,2]       [,3]
## [1,] -3.340785 0.09192216 -0.1753546
## [2,]  0.000000 0.20888840 -0.2499881
```

```
## [3,]  0.000000 0.00000000 -0.1193344
```

predict(newdata = list(), qr = FALSE)

This function allows predicted values of the model to be calculated for dependent variable data in the list "newdata."

?? What does qr=FALSE do??

```
cat("Display Croucher predict from nmodc1 as currently set:\n")
```

```
## Display Croucher predict from nmodc1 as currently set:
```

```
cat("first with no new parameters -- we get the fitted values\n")
```

```
## first with no new parameters -- we get the fitted values
```

```
print(nmodc1$predict())
```

```
##  [1] 0.7392015 0.7471673 0.7706004 0.8613725 1.0000000 1.0829458 1.1577458
##  [8] 1.1462098 1.0407158 0.8843125
```

```
cat("Now with some new data\n")
```

```
## Now with some new data
```

```
newcdata<-data.frame(xdata=0.1, ydata=1.1)
print(nmodc1$predict(newdata=newcdata))
```

```
## [1] 1.019767
```

```
cat("Display Hobbs predict from nmodc1 as currently set:\n")
```

```
## Display Hobbs predict from nmodc1 as currently set:
```

```
cat("first with no new parameters -- we get the fitted values\n")
```

```
## first with no new parameters -- we get the fitted values
```

```
print(nmodh1$predict())
```

```
##  [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
```

```
cat("Now with some new data\n")
```

```
## Now with some new data
```

```
newhdata<-data.frame(tt=c(13,14), y=c(120,200))
print(nmodh1$predict(newdata=newhdata))
```

```
## [1] 0.9999977 0.9999992
```

```
cat("trying again with qr=TRUE\n")
```

```
## trying again with qr=TRUE
```

```
print(nmodh1$predict(newdata=newhdata, qr=TRUE))
```

```
## [1] 0.9999977 0.9999992
```

Using "qr=TRUE" does not seem to make any difference. Need to delve further??

Discussion

Here we will look at some of the issues of using nlsModel. In particular, we will examine how this function may need to be modified to make it easier to maintain and use.

References

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