Package 'nlmrt'

October 13, 2022

Type Package

Title Functions for Nonlinear Least Squares Solutions

Version 2016.3.2

Date 2016-3-2

Author John C. Nash [aut, cre]

Maintainer John C. Nash <nashjc@uottawa.ca>

Description Replacement for nls() tools for working with nonlinear least squares problems. The calling structure is similar to, but much simpler than, that of the nls() function. Moreover, where nls() specifically does NOT deal with small or zero residual problems, nlmrt is quite happy to solve them. It also attempts to be more robust in finding solutions, thereby avoiding 'singular gradient' messages that arise in the Gauss-Newton method within nls(). The Marquardt-Nash approach in nlmrt generally works more reliably to get a solution, though this may be one of a set of possibilities, and may also be statistically unsatisfactory. Added print and summary as of August 28, 2012.

License GPL-2

Depends R (>= 2.15.0)

Suggests minpack.lm, optimx, Rvmmin, Rcgmin, numDeriv

NeedsCompilation no

Repository CRAN

Date/Publication 2016-03-04 23:57:39

R topics documented:

`nlmrt-package` *Tools for solving nonlinear least squares problems. UNDER DEVELOPMENT.*

Description

The package provides some tools related to using the Nash variant of Marquardt's algorithm for nonlinear least squares.

Details

Package:	nlmrt
Type:	Package
Version:	1.0
Date:	2012-03-05
License:	GPL-2

This package includes methods for solving nonlinear least squares problems specified by a modeling expression and given a starting vector of named paramters. Note: You must provide an expression of the form lhs ~ rhsexpression so that the residual expression rhsexpression - lhs can be computed. The expression can be enclosed in quotes, and this seems to give fewer difficulties with R. Data variables must already be defined, either within the parent environment or else in the dot-arguments. Other symbolic elements in the modeling expression must be standard functions or else parameters that are named in the start vector.

The main functions in `nlmrt` are:

```
   nlfb - Nash variant of the Marquardt procedure for nonlinear least squares,
with bounds constraints, using a residual and optionally Jacobian
described as \code{R} functions.
    20120803: Todo: Make masks more consistent between nlfb and nlxb.

   nlxb - Nash variant of the Marquardt procedure for nonlinear least squares,
with bounds constraints, using an expression to describe the residual via
        an \code{R} modeling expression. The Jacobian is computed via symbolic
differentiation.

   wrapnls - Uses nlxb to solve nonlinear least squares then calls nls() to
          create an object of type nls.
```

```
model2grfun.R - Generate a gradient vector function from a nonlinear
     model expression and a vector of named parameters.

model2jacfun.R - Generate a Jacobian matrix function from a nonlinear
     model expression and a vector of named parameters.

model2resfun.R - Generate a residual vector function from a nonlinear
     model expression and a vector of named parameters.

model2ssfun.R - Generate a sum of squares objective function from a
     nonlinear model expression and a vector of named parameters.

modgr.R - compute gradient of the sum of squares function using the
     Jacobian and residuals for a nonlinear least squares problem

modss.R - computer the sum of squares function from the residuals of
     a nonlinear least squares problem

myfn.R, mygr.R, myjac.R, myres.R, myss.R - dummy functions that seem to
    be needed so there is an available handle for output of functions that
    generate various functions from expressions.
```

For testing purposes, there are also some experimental codes using different internal computations for the linear algebraic sub-problems in the inst/dev-codes/ sub-folder.

Author(s)

John C Nash

Maintainer: <nashjc@uottawa.ca>

References

Nash, J. C. (1979, 1990) _Compact Numerical Methods for Computers. Linear Algebra and Function Minimisation._ Adam Hilger./Institute of Physics Publications

others!!??

See Also

`nls`

Examples

```
rm(list=ls())
# library(nlmrt)

# traceval set TRUE to debug or give full history
traceval  <-  FALSE
```

```
## Problem in 1 parameter to ensure methods work in trivial case

cat("Problem in 1 parameter to ensure methods work in trivial case\n")
nobs<-8
tt <- seq(1,nobs)
dd <- 1.23*tt + 4*runif(nobs)

df <- data.frame(tt, dd)

a1par<-nlxb(dd ~ a*tt, start=c(a=1), data=df)
a1par


# Data for Hobbs problem
cat("Hobbs weed problem -- unscaled\n")
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

eunsc  <-   y ~ b1/(1+b2*exp(-b3*tt))

cat("Hobbs unscaled with data in data frames\n")

weeddata1  <-  data.frame(y=ydat, tt=tdat)
# scale the data
weeddata2  <-  data.frame(y=1.5*ydat, tt=tdat)
start1  <-  c(b1=1, b2=1, b3=1)
anlxb1  <-  try(nlxb(eunsc, start=start1, trace=traceval, data=weeddata1))
print(anlxb1)

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

# Problem 2 - Gabor Grothendieck 2016-3-2

cat("Gabor G problem with zero residuals\n")

DF <- data.frame(x = c(5, 4, 3, 2, 1), y = c(1, 2, 3, 4, 5))
library(nlmrt)
nlxb1 <- nlxb(y ~ A * x + B, data = DF, start = c(A = 1, B = 6), trace=TRUE)
print(nlxb1)

# tmp <- readline("continue with start at the minimum -- failed on 2014 version. ")

nlxb0 <- nlxb(y ~ A * x + B, data = DF, start = c(A = -1, B = 6), trace=TRUE)
print(nlxb0)

## Not run:
# WARNING -- using T can get confusion with TRUE
tt  <-  tdat
# A simple starting vector -- must have named parameters for nlxb, nls, wrapnls.
```

```
cat("GLOBAL DATA\n")

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

cat("GLOBAL DATA AND EXPRESSION -- SHOULD FAIL\n")
anlxb1g  <-  try(nlxb(eunsc, start=start1, trace=traceval))
print(anlxb1g)


## End(Not run) # end dontrun

rm(y)
rm(tt)


startf1  <-  c(b1=1, b2=1, b3=.1)


## Not run:

## 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)


## End(Not run) # end dontrun


# Check nls too
## Not run:
cat("check nls result\n")
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)

# tmp  <-  readline("next")


## End(Not run) # end dontrun

## Not run:

anlxb2  <-  try(nlxb(eunsc, start=start1, lower=c(b1=0, b2=0, b3=0),
        upper=c(b1=500, b2=100, b3=.25), trace=traceval, data=weeddata1))
print(anlxb2)


anlsb2  <-  try(nls(eunsc, start=start1, lower=c(b1=0, b2=0, b3=0),
```

```
                upper=c(b1=500, b2=100, b3=.25), trace=traceval,
                data=weeddata1, algorithm='port'))
print(anlsb2)

# tmp  <-  readline("next")

## End(Not run) # end dontrun


## Not run:
cat("UNCONSTRAINED\n")
an1q  <-  try(nlxb(eunsc, start=start1, trace=traceval, data=weeddata1))
print(an1q)
# tmp  <-  readline("next")

## End(Not run) # end dontrun


## Not run:
cat("TEST MASKS\n")

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)

## End(Not run) # end dontrun


## Not run:

cat("MASKED\n")

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

# 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)
# tmp  <-  readline("next")


## End(Not run) # end dontrun


cat("BOUNDS test problem for Hobbs")
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, .3)))
print(an1qb1)
```

```
## tmp  <-  readline("next")


cat("BOUNDS and MASK")

## Not run:

an1qbm2  <-  try(nlxb(eunsc, start=start2, trace=traceval, data=weeddata1,
                      lower=c(0,0,0), upper=c(200, 60, .3), masked=c("b2")))
print(an1qbm2)

# tmp  <-  readline("next")


## End(Not run) # end dontrun


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


cat("EASY start -- unscaled")
anls01  <-  try(nls(eunsc, start=suneasy, trace=traceval, data=weeddata1))
print(anls01)
anlmrt01  <-  try(nlxb(eunsc, start=sseasy, trace=traceval, data=weeddata1))
print(anlmrt01)

## Not run:

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

cat("ones start -- scaled")
anls03  <-  try(nls(escal, start=start1, trace=traceval, data=weeddata1))
print(anls03)
anlmrt03  <-  nlxb(escal, start=start1, trace=traceval, data=weeddata1)
print(anlmrt03)

cat("HARD start -- scaled")
anls04  <-  try(nls(escal, start=st1scal, trace=traceval, data=weeddata1))
print(anls04)
anlmrt04  <-  nlxb(escal, start=st1scal, trace=traceval, data=weeddata1)
print(anlmrt04)
```

```
cat("EASY start -- scaled")
anls05  <-  try(nls(escal, start=ssceasy, trace=traceval, data=weeddata1))
print(anls05)
anlmrt05  <-  nlxb(escal, start=ssceasy, trace=traceval, data=weeddata1)
print(anlmrt03)


## End(Not run) # end dontrun


## Not run:

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  <-  100.0*x[1]/(1+x[2]*10.*exp(-0.1*x[3]*tt)) - y
}

shobbs.jac  <-  function(x) { # scaled Hobbs weeds problem -- Jacobian
    jj  <-  matrix(0.0, 12, 3)
    tt  <-  1:12
    yy  <-  exp(-0.1*x[3]*tt)
    zz  <-  100.0/(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")
st  <-  c(b1=1, b2=1, b3=1)
low  <-  -Inf
up <- Inf

ans1 <- nlfb(st, shobbs.res, shobbs.jac, trace=traceval)
ans1
cat("No jacobian function -- use internal approximation\n")
ans1n <- nlfb(st, shobbs.res, trace=TRUE, control=list(watch=TRUE)) # NO jacfn
ans1n

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



## End(Not run) # end dontrun
```

```
## Not run:

cat("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)

# tmp <- readline("next")

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

cat("BUT ... nls() seems to do better from the TRACE information\n")
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")
print(anlsb)
cat("BUT...crossprod(resid(anlsb))=",crossprod(resid(anlsb)),"\n")


## End(Not run) # end dontrun


# tmp <- readline("next")

cat("Try wrapnls\n")
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)


## Not run:

cat("BOUNDED wrapnls\n")

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

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

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

cat("TRY MASKS ONLY\n")

an1xm3 <- try(nlxb(escal, start1, trace=traceval, data=weeddata1,
                   masked=c("b3")))
printsum(an1xm3)
an1fm3 <- try(nlfb(start1, shobbs.res, shobbs.jac, trace=traceval,
                   data=weeddata1, maskidx=c(3)))
printsum(an1fm3)

an1xm1 <- try(nlxb(escal, start1, trace=traceval, data=weeddata1,
                   masked=c("b1")))
printsum(an1xm1)
an1fm1 <- try(nlfb(start1, shobbs.res, shobbs.jac, trace=traceval,
                   data=weeddata1, maskidx=c(1)))
printsum(an1fm1)


## End(Not run) # end dontrun

# Need to check when all parameters masked.??

## Not run:


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

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")

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

cat("\n\n")

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

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:")
print(pvec)
rp <- k.r(pvec, x=x, y=y)
cat(" rp=")
print(rp)
rf <- k.f(pvec, x=x, y=y)
cat(" rf=")
print(rf)
rj <- k.j(pvec, x=x, y=y)
cat(" rj=")
print(rj)
rg <- k.g(pvec, x=x, y=y)
cat(" rg=")
print(rg)
cat("modss at pvec gives ")
print(modss(pvec, k.r, x=x, y=y))
cat("modgr at pvec gives ")
print(modgr(pvec, k.r, k.j, x=x, y=y))
cat("\n\n")

cat("At bnew:")
print(bnew)
rb <- k.r(bnew, x=x, y=y)
cat(" rb=")
print(rb)
rf <- k.f(bnew, x=x, y=y)
cat(" rf=")
print(rf)
rj <- k.j(bnew, x=x, y=y)
cat(" rj=")
print(rj)
rg <- k.g(bnew, x=x, y=y)
cat(" rg=")
print(rg)
cat("modss at bnew gives ")
print(modss(bnew, k.r, x=x, y=y))
cat("modgr at bnew gives ")
print(modgr(bnew, k.r, k.j, x=x, y=y))
cat("\n\n")


## End(Not run)  ## end of dontrun
```

`coef.nlmrt` *Output model coefficients for nlmrt object.*

Description

`coef.nlmrt` extracts and displays the coefficients for a model estimated by `nlxb` or `nlfb` in the `nlmrt` structured `object`.

Usage

```
    ## S3 method for class 'nlmrt'
coef(object, ...)
```

Arguments

`object`	An object of class 'nlmrt'
`...`	Any data needed for the function. We do not know of any!

Details

`coef.nlmrt` extracts and displays the coefficients for a model estimated by `nlxb` or `nlfb`.

Value

returns the coefficients from the nlmrt object.

Note

Special notes, if any, will appear here.

Author(s)

John C Nash <nashjc@uottawa.ca>

See Also

Function `nls()`, packages `optim` and `optimx`.

model2grfun *Generate a gradient function from a nonlinear model expression and a vector of named parameters.*

Description

Given a nonlinear model expressed as an expression of the form lhs ~ formula_for_rhs and a start vector where parameters used in the model formula are named, attempts to build the the R function for the gradient of the residual sum of squares. As a side effect, a text file with the program code is generated.

Usage

```
model2grfun(modelformula, pvec, funname="mygr", filename=NULL)
```

Arguments

modelformula	This is a modeling formula of the form (as in nls) lhsvar ~ rhsexpression for example, y ~ b1/(1+b2*exp(-b3*tt)) You may also give this as a string.
pvec	A named parameter vector. For our example, we could use start=c(b1=1, b2=2.345, b3=0.123) WARNING: the parameters in the output function will be used in the order presented in this vector. Names are NOT respected in the output function.
funname	The (optional) name for the function that is generated in the file named in the next argument. The default name is 'mygr'.
filename	The (optional) name of a file that is written containing the (text) program code for the function. If NULL, no file is written.

Details

None.

Value

An R function object that computes the gradient of the sum of squared residuals of a nonlinear model at a set of parameters.

Author(s)

John C Nash <nashjc@uottawa.ca>

References

Nash, J. C. (1979, 1990) _Compact Numerical Methods for Computers. Linear Algebra and Function Minimisation._ Adam Hilger./Institute of Physics Publications

See Also

Function nls(), packages optim and optimx.

Examples

```
cat("See also examples in nlmrt-package.Rd\n")
require(numDeriv)
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)  # for testing
tt <- seq_along(y)  # for testing
f <- y ~ b1/(1 + b2 * exp(-1 * b3 * tt))
p <- c(b1 = 1, b2 = 1, b3 = 1)
mygr <- model2grfun(f, p)
myss <- model2ssfun(f, p) # for check
cat("mygr:\n")
print(mygr)

ans <- mygr(p, tt = tt, y = y)
print(ans)
gnum <- grad(myss, p, tt = tt, y = y)
cat("Max(abs(ans-gnum)) = ",max(abs(ans-gnum)),"\n")

bnew <- c(b1 = 200, b2 = 50, b3 = 0.3)
ans <- mygr(prm = bnew, tt = tt, y = y)
print(ans)
gnum <- grad(myss, bnew, tt = tt, y = y)
cat("Max(abs(ans-gnum)) = ",max(abs(ans-gnum)),"\n")

cat("Test with un-named vector\n")  # At 20120424 should fail
bthree <- c(100, 40, 0.1)
ans <- mygr(prm = bthree, tt = tt, y = y)
print(ans)
gnum <- grad(myss, bthree, tt = tt, y = y)
cat("Max(abs(ans-gnum)) = ",max(abs(ans-gnum)),"\n")
```

`model2jacfun` *Generate a Jacobian matrix function from a nonlinear model expression and a vector of named parameters.*

Description

Given a nonlinear model expressed as an expression of the form lhs ~ formula_for_rhs and a start vector where parameters used in the model formula are named, attempts to build the the R function for the Jacobian of the residuals. As a side effect, a text file with the program code is generated.

Usage

```
model2jacfun(modelformula, pvec, funname="myjac", filename=NULL)
```

Arguments

`modelformula`	This is a modeling formula of the form (as in `nls`) lhsvar ~ rhsexpression for example, y ~ b1/(1+b2*exp(-b3*tt)) You may also give this as a string.

pvec	A named parameter vector. For our example, we could use start=c(b1=1, b2=2.345, b3=0.123) WARNING: the parameters in the output function will be used in the order presented in this vector. Names are NOT respected in the output function.
funname	The (optional) name for the function that is generated in the file named in the next argument. The default name is 'myjac'.
filename	The (optional) name of a file that is written containing the (text) program code for the function. If NULL, no file is written.

Details

None.

Value

An R function object that computes the Jacobian of the nonlinear model at a set of parameters.

Author(s)

John C Nash <nashjc@uottawa.ca>

References

Nash, J. C. (1979, 1990) _Compact Numerical Methods for Computers. Linear Algebra and Function Minimisation._ Adam Hilger./Institute of Physics Publications

See Also

Function nls(), packages optim and optimx.

Examples

```
cat("See also examples in nlmrt-package.Rd\n")
require(numDeriv)
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)  # for testing
tt <- seq_along(y)  # for testing
f <- y ~ b1/(1 + b2 * exp(-1 * b3 * tt))
p <- c(b1 = 1, b2 = 1, b3 = 1)
myfn <- model2jacfun(f, p)
myres <- model2resfun(f, p)
cat("myfn:\n")
print(myfn)

ans <- myfn(p, tt = tt, y = y)
print(ans)
Jnum<-jacobian(myres, p,  tt = tt, y = y)
cat("max(abs(ans-Jnum)) = ",max(abs(ans-Jnum)),"\n")

bnew <- c(b1 = 200, b2 = 50, b3 = 0.3)
ans <- myfn(prm = bnew, tt = tt, y = y)
print(ans)
```

```
Jnum<-jacobian(myres, bnew,  tt = tt, y = y)
cat("max(abs(ans-Jnum)) = ",max(abs(ans-Jnum)),"\n")

cat("Test with un-named vector\n")  # At 20120424 should fail
bthree <- c(100, 40, 0.1)
ans <- try(myfn(prm = bthree, tt = tt, y = y))
print(ans)
Jnum<-jacobian(myres, bthree,  tt = tt, y = y)
cat("max(abs(ans-Jnum)) = ",max(abs(ans-Jnum)),"\n")
```

`model2resfun` *Generate a residual function from a nonlinear model expression and a vector of named parameters.*

Description

Given a nonlinear model expressed as an expression of the form lhs ~ formula_for_rhs and a start vector where parameters used in the model formula are named, attempts to build the the R function for the residuals of the model. As a side effect, a text file with the program code is generated.

Usage

```
model2resfun(modelformula, pvec, funname="myres", filename=NULL)
```

Arguments

`modelformula`	This is a modeling formula of the form (as in `nls`) lhsvar ~ rhsexpression for example, y ~ b1/(1+b2*exp(-b3*tt)) You may also give this as a string.
`pvec`	A named parameter vector. For our example, we could use start=c(b1=1, b2=2.345, b3=0.123) WARNING: the parameters in the output function will be used in the order presented in this vector. Names are NOT respected in the output function.
`funname`	The (optional) name for the function that is generated in the file named in the next argument. The default name is 'myres'.
`filename`	The (optional) name of a file that is written containing the (text) program code for the function. If NULL, no file is written.

Details

None.

Value

An `R` function object that computes the gradient of the sum of squared residuals of a nonlinear model at a set of parameters.

Author(s)

John C Nash <nashjc@uottawa.ca>

References

Nash, J. C. (1979, 1990) _Compact Numerical Methods for Computers. Linear Algebra and Function Minimisation._ Adam Hilger./Institute of Physics Publications

See Also

Function `nls()`, packages `optim` and `optimx`.

Examples

```
cat("See also examples in nlmrt-package.Rd\n")
  # a test
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)  # for testing
tt <- seq_along(y)  # for testing
# NOTE: use of t gives confusion with t() in R CMD check,
# but not in direct use with source() 120429
f <- y ~ b1/(1 + b2 * exp(-1 * b3 * tt))
p <- c(b1 = 1, b2 = 1, b3 = 1)
myres <- model2resfun(f, p)
cat("myres:\n")
print(myres)
ans <- myres(p, tt = tt, y = y)
cat("ans:")
print(ans)
cat("ss ( =? 23520.58):", as.numeric(crossprod(ans)),"\n")
bnew <- c(b1 = 200, b2 = 50, b3 = 0.3)
# anew<-myres(prm=bnew, t=t, y=y) # Note issue with t vs
# t()
anew <- eval(myres(prm = bnew, tt = tt, y = y))
cat("anew:")
print(anew)
cat("ss ( =? 158.2324):", as.numeric(crossprod(anew)),"\n")
cat("Test with vector of un-named parameters\n")
bthree <- c(100, 40, 0.1)
athree <- try(myres(prm = bthree, tt = tt, y = y))
print(athree)
cat("ss ( =? 19536.65):", as.numeric(crossprod(athree)),"\n")
```

`model2ssfun` *Generate a sum of squares objective function from a nonlinear model expression and a vector of named parameters.*

Description

Given a nonlinear model expressed as an expression of the form lhs ~ formula_for_rhs and a start vector where parameters used in the model formula are named, attempts to build the the R function for the residual sum of squares. As a side effect, a text file with the program code is generated.

Usage

```
model2ssfun(modelformula, pvec, funname="myss", filename=NULL)
```

Arguments

modelformula	This is a modeling formula of the form (as in nls) lhsvar ~ rhsexpression for example, y ~ b1/(1+b2*exp(-b3*tt)) You may also give this as a string.
pvec	A named parameter vector. For our example, we could use start=c(b1=1, b2=2.345, b3=0.123) WARNING: the parameters in the output function will be used in the order presented in this vector. Names are NOT respected in the output function.
funname	The (optional) name for the function that is generated in the file named in the next argument. The default name is 'myss'.
filename	The (optional) name of a file that is written containing the (text) program code for the function. If NULL, no file is written.

Details

None.

Value

An R function object that computes the sum of squares of the residuals of the nonlinear model at a set of parameters.

Author(s)

John C Nash <nashjc@uottawa.ca>

References

Nash, J. C. (1979, 1990) _Compact Numerical Methods for Computers. Linear Algebra and Function Minimisation._ Adam Hilger./Institute of Physics Publications

See Also

Function nls(), packages optim and optimx.

Examples

```
# a test
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)  # for testing
tt <- seq_along(y)  # for testing
f <- y ~ b1/(1 + b2 * exp(-1 * b3 * tt))
p <- c(b1 = 1, b2 = 1, b3 = 1)
myfn <- model2ssfun(f, p)
cat("myfn: \n")
print(myfn) # list the function
cat("Compute the function at several points\n")
```

```
ans <- myfn(p, tt = tt, y = y)
print(ans) # should be  23520.58
bnew <- c(b1 = 200, b2 = 50, b3 = 0.3)
anew <- myfn(prm = bnew, tt = tt, y = y)
print(anew)# should be  158.2324

cat("Test with vector of un-named parameters \n")
bthree <- c(100, 40, 0.1)
athree <- try(myfn(prm = bthree, tt = tt, y = y))
print(athree) # should be 19536.65
```

---

modgr *Compute gradient from residuals and Jacobian.*

Description

For a nonlinear model originally expressed as an expression of the form lhs ~ formula_for_rhs assume we have a resfn and jacfn that compute the residuals and the Jacobian at a set of parameters. This routine computes the gradient, that is, t(Jacobian) . residuals.

Usage

```
modgr(prm, resfn, jacfn, ...)
```

Arguments

prm	A parameter vector. For our example, we could use start=c(b1=1, b2=2.345, b3=0.123) However, the names are NOT used, only positions in the vector.
resfn	A function to compute the residuals of our model at a parameter vector.
jacfn	A function to compute the Jacobian of the residuals at a paramter vector.
...	Any data needed for computation of the residual vector from the expression rhsexpression - lhsvar. Note that this is the negative of the usual residual, but the sum of squares is the same.

Details

modgr calls resfn to compute residuals and jacfn to compute the Jacobian at the parameters prm using external data in the dot arguments. It then computes the gradient using t(Jacobian) . residuals.

Note that it appears awkward to use this function in calls to optimization routines. The author would like to learn why.

Value

The numeric vector with the gradient of the sum of squares at the paramters.

Note

Special notes, if any, will appear here.

Author(s)

John C Nash <nashjc@uottawa.ca>

References

Nash, J. C. (1979, 1990) _Compact Numerical Methods for Computers. Linear Algebra and Function Minimisation._ Adam Hilger./Institute of Physics Publications

See Also

Function `nls()`, packages `optim` and `optimx`.

Examples

```
cat("See examples in nlmrt-package.Rd\n")
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)  # for testing
tt <- seq_along(y)  # for testing
f <- y ~ b1/(1 + b2 * exp(-1 * b3 * tt))
p <- c(b1 = 1, b2 = 1, b3 = 1)
myres <- model2resfun(f, p)
myjac <- model2jacfun(f, p)
mygr  <- model2grfun(f, p)
gr <- mygr(p, tt = tt, y = y)
grm <- modgr(p, myres, myjac, tt = tt, y = y)
cat("max(abs(grm - gr)) =",max(abs(grm-gr)),"\n")
```

`modss` *Compute gradient from residuals and Jacobian.*

Description

For a nonlinear model originally expressed as an expression of the form lhs ~ formula_for_rhs assume we have a resfn and jacfn that compute the residuals and the Jacobian at a set of parameters. This routine computes the gradient, that is, t(Jacobian)

Usage

```
modss(prm, resfn, ...)
```

Arguments

`prm`	A parameter vector. For our example, we could use start=c(b1=1, b2=2.345, b3=0.123) However, the names are NOT used, only positions in the vector.
`resfn`	A function to compute the residuals of our model at a parameter vector.
`...`	Any data needed for computation of the residual vector from the expression rhsexpression - lhsvar. Note that this is the negative of the usual residual, but the sum of squares is the same.

Details

modss calls resfn to compute residuals and then uses crossprod to compute the sum of squares.

At 2012-4-26 there is no checking for errors.

Note that it appears awkward to use this function in calls to optimization routines. The author would like to learn why.

Value

The numeric value of the sum of squares at the paramters.

Note

Special notes, if any, will appear here.

Author(s)

John C Nash <nashjc@uottawa.ca>

References

Nash, J. C. (1979, 1990) _Compact Numerical Methods for Computers. Linear Algebra and Function Minimisation._ Adam Hilger./Institute of Physics Publications

others!!

See Also

Function nls(), packages optim and optimx.

Examples

```
cat("See examples in nlmrt-package.Rd\n")
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)  # for testing
tt <- seq_along(y)  # for testing
f <- y ~ b1/(1 + b2 * exp(-1 * b3 * tt))
p <- c(b1 = 1, b2 = 1, b3 = 1)
myres <- model2resfun(f, p)
myssval <- modss(p, myres, tt = tt, y = y)
cat("ss at (1,1,1) (should be 23520.58) = ",myssval,"\n")
```

nlfb *Nash variant of Marquardt nonlinear least squares solution via qr linear solver.*

Description

Given a nonlinear model expressed as an expression of the form lhs ~ formula_for_rhs and a start vector where parameters used in the model formula are named, attempts to find the minimum of the residual sum of squares using the Nash variant (Nash, 1979) of the Marquardt algorithm, where the linear sub-problem is solved by a qr method.

Usage

```
nlfb(start, resfn, jacfn=NULL, trace=FALSE, lower=-Inf, upper=Inf,
      maskidx=NULL, control, ...)
```

Arguments

resfn	A function that evaluates the residual vector for computing the elements of the sum of squares function at the set of parameters start.
jacfn	A function that evaluates the Jacobian of the sum of squares function, that is, the matrix of partial derivatives of the residuals with respect to each of the parameters. If NULL (default), uses an approximation. ?? put in character form as in optimx??
start	A named parameter vector. For our example, we could use start=c(b1=1, b2=2.345, b3=0.123) nls() takes a list, and that is permitted here also.
trace	Logical TRUE if we want intermediate progress to be reported. Default is FALSE.
lower	Lower bounds on the parameters. If a single number, this will be applied to all parameters. Default -Inf.
upper	Upper bounds on the parameters. If a single number, this will be applied to all parameters. Default Inf.
maskidx	Vector if 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.
control	A list of controls for the algorithm. These are: watch Monitor progress if TRUE. Default is FALSE. phi Default is phi=1, which adds phi*Identity to Jacobian inner product. lamda Initial Marquardt adjustment (Default 0.0001). Odd spelling is deliberate. offset Shift to test for floating-point equality. Default is 100. laminc Factor to use to increase lamda. Default is 10. lamdec Factor to use to decrease lamda is lamdec/laminc. Default lamdec=4.

`femax` Maximum function (sum of squares) evaluations. Default is 10000, which is extremely aggressive.

`jemax` Maximum number of Jacobian evaluations. Default is 5000.

`ndstep` Stepsize to use to computer numerical Jacobian approximatin. Default is 1e-7.

`rofftest` Default is TRUE. Use a termination test of the relative offset orthogonality type. Useful for nonlinear regression problems.

`smallsstest` Default is TRUE. Exit the function if the sum of squares falls below (100 * .Machine$double.eps)^4 times the initial sumsquares. This is a test for a "small" sum of squares, but there are problems which are very extreme for which this control needs to be set FALSE.

`...` Any data needed for computation of the residual vector from the expression rhsexpression - lhsvar. Note that this is the negative of the usual residual, but the sum of squares is the same. It is not clear how the dot variables should be used, since data should be in 'data'.

Details

`nlfb` attempts to solve the nonlinear sum of squares problem by using a variant of Marquardt's approach to stabilizing the Gauss-Newton method using the Levenberg-Marquardt adjustment. This is explained in Nash (1979 or 1990) in the sections that discuss Algorithm 23.

In this code, we solve the (adjusted) Marquardt equations by use of the `qr.solve()`. Rather than forming the J'J + lambda*D matrix, we augment the J matrix with extra rows and the y vector with null elements.

Value

A list of the following items

`coefficients` A named vector giving the parameter values at the supposed solution.

`ssquares` The sum of squared residuals at this set of parameters.

`resid` The residual vector at the returned parameters.

`jacobian` The jacobian matrix (partial derivatives of residuals w.r.t. the parameters) at the returned parameters.

`feval` The number of residual evaluations (sum of squares computations) used.

`jeval` The number of Jacobian evaluations used.

Note

Special notes, if any, will appear here.

Author(s)

John C Nash <nashjc@uottawa.ca>

References

Nash, J. C. (1979, 1990) _Compact Numerical Methods for Computers. Linear Algebra and Function Minimisation._ Adam Hilger./Institute of Physics Publications

others!!

See Also

Function `nls()`, packages `optim` and `optimx`.

Examples

```
cat("See examples in nlmrt-package.Rd\n")
```

`nlxb` *Nash variant of Marquardt nonlinear least squares solution via qr linear solver.*

Description

Given a nonlinear model expressed as an expression of the form lhs ~ formula_for_rhs and a start vector where parameters used in the model formula are named, attempts to find the minimum of the residual sum of squares using the Nash variant (Nash, 1979) of the Marquardt algorithm, where the linear sub-problem is solved by a qr method.

Usage

```
nlxb(formula, start, trace=FALSE, data, lower=-Inf, upper=Inf,
     masked=NULL, control, ...)
```

Arguments

`formula`	This is a modeling formula of the form (as in `nls`) lhsvar ~ rhsexpression for example, y ~ b1/(1+b2*exp(-b3*tt)) You may also give this as a string.
`start`	A named parameter vector. For our example, we could use start=c(b1=1, b2=2.345, b3=0.123)
`trace`	Logical TRUE if we want intermediate progress to be reported. Default is FALSE.
`data`	A data frame containing the data of the variables in the formula. This data may, however, be supplied directly in the parent frame.
`lower`	Lower bounds on the parameters. If a single number, this will be applied to all parameters. Default -Inf.
`upper`	Upper bounds on the parameters. If a single number, this will be applied to all parameters. Default Inf.
`masked`	Character vector of quoted parameter names. These parameters will NOT be altered by the algorithm.

`control` A list of controls for the algorithm. These are:

`watch` Monitor progress if TRUE. Default is FALSE.

`phi` Default is phi=1, which adds phi*Identity to Jacobian inner product.

`lamda` Initial Marquardt adjustment (Default 0.0001). Odd spelling is deliberate.

`offset` Shift to test for floating-point equality. Default is 100.

`laminc` Factor to use to increase lamda. Default is 10.

`lamdec` Factor to use to decrease lamda is lamdec/laminc. Default lamdec=4.

`femax` Maximum function (sum of squares) evaluations. Default is 10000, which is extremely aggressive.

`jemax` Maximum number of Jacobian evaluations. Default is 5000.

`rofftest` Default is TRUE. Use a termination test of the relative offset orthogonality type. Useful for nonlinear regression problems.

`smallsstest` Default is TRUE. Exit the function if the sum of squares falls below (100 * .Machine$double.eps)^4 times the initial sumsquares. This is a test for a "small" sum of squares, but there are problems which are very extreme for which this control needs to be set FALSE.

`...` Any data needed for computation of the residual vector from the expression rhsexpression - lhsvar. Note that this is the negative of the usual residual, but the sum of squares is the same.

Details

`nlxb` attempts to solve the nonlinear sum of squares problem by using a variant of Marquardt's approach to stabilizing the Gauss-Newton method using the Levenberg-Marquardt adjustment. This is explained in Nash (1979 or 1990) in the sections that discuss Algorithm 23. (?? do we want a vignette. Yes, because folk don't have access to book easily, but finding time.)

In this code, we solve the (adjusted) Marquardt equations by use of the `qr.solve()`. Rather than forming the J'J + lambda*D matrix, we augment the J matrix with extra rows and the y vector with null elements.

Value

A list of the following items

`coefficients` A named vector giving the parameter values at the supposed solution.

`ssquares` The sum of squared residuals at this set of parameters.

`resid` The residual vector at the returned parameters.

`jacobian` The jacobian matrix (partial derivatives of residuals w.r.t. the parameters) at the returned parameters.

`feval` The number of residual evaluations (sum of squares computations) used.

`jeval` The number of Jacobian evaluations used.

Note

Special notes, if any, will appear here.

Author(s)

John C Nash <nashjc@uottawa.ca>

References

Nash, J. C. (1979, 1990) _Compact Numerical Methods for Computers. Linear Algebra and Function Minimisation._ Adam Hilger./Institute of Physics Publications

others!!

See Also

Function `nls()`, packages `optim` and `optimx`.

Examples

```
cat("See examples in nlmrt-package.Rd\n")
```

`print.nlmrt` *Print method for an object of class nlmrt.*

Description

Print summary output (but involving some serious computations!) of an object of class nlmrt from `nlxb` or `nlfb` from package `nlmrt`.

Usage

```
    ## S3 method for class 'nlmrt'
print(x, ...)
```

Arguments

`x`	An object of class 'nlmrt'
`...`	Any data needed for the function. We do not know of any!

Details

`printsum.nlmrt` performs a print method for an object of class 'nlmrt' that has been created by a routine such as `nlfb` or `nlxb` for nonlinear least squares problems.

Value

Invisibly returns the input object.

Note

Special notes, if any, will appear here.

Author(s)

John C Nash <nashjc@uottawa.ca>

See Also

Function `nls()`, packages `optim` and `optimx`.

`summary.nlmrt` *Summary output for nlmrt object.*

Description

Provide a summary output (but involving some serious computations!) of an object of class nlmrt from `nlxb` or `nlfb` from package `nlmrt`.

Usage

```
    ## S3 method for class 'nlmrt'
summary(object, ...)
```

Arguments

`object`	An object of class 'nlmrt'
`...`	Any data needed for the function. We do not know of any!

Details

`summary.nlmrt` performs a summary method for an object of class 'nlmrt' that has been created by a routine such as `nlfb` or `nlxb` for nonlinear least squares problems.

Issue: When there are bounded parameters, `nls` returns a Standard Error for each of the parameters. However, this summary does NOT have a Jacobian value (it is set to 0) for columns where a parameter is masked or at (or very close to) a bound. See the `R` code for the determination of whether we are at a bound. In this case, users may wish to look in the `inst/dev-codes` directory of this package, where there is a script `seboundsnlmrtx.R` that computes the `nls()` standard errors for comparison on a simple problem.

Issue: The printsum() of this object includes the singular values of the Jacobian. These are displayed, one per coefficient row, with the coefficients. However, the Jacobian singular values do NOT have a direct correspondence to the coefficients on whose display row they appear. It simply happens that there are as many Jacobian singular values as coefficients, and this is a convenient place to display them. The same issue applies to the gradient components.

Value

returns an invisible copy of the nlmrt object.

Note

Special notes, if any, will appear here.

Author(s)

John C Nash <nashjc@uottawa.ca>

See Also

Function `nls()`, packages `optim` and `optimx`.

`wrapnls` *Nash variant of Marquardt nonlinear least squares solution via qr linear solver.*

Description

Given a nonlinear model expressed as an expression of the form lhs ~ formula_for_rhs and a start vector where parameters used in the model formula are named, attempts to find the minimum of the residual sum of squares using the Nash variant (Nash, 1979) of the Marquardt algorithm, where the linear sub-problem is solved by a qr method.

Usage

```
wrapnls(formula, start, trace=FALSE, data, lower=-Inf, upper=Inf,
        control=list(), ...)
```

Arguments

`formula`	This is a modeling formula of the form (as in `nls`) lhsvar ~ rhsexpression for example, y ~ b1/(1+b2*exp(-b3*tt)) You may also give this as a string.
`start`	A named parameter vector. For our example, we could use start=c(b1=1, b2=2.345, b3=0.123)
`trace`	Logical TRUE if we want intermediate progress to be reported. Default is FALSE.
`data`	A data frame containing the data of the variables in the formula. This data may, however, be supplied directly in the parent frame.
`lower`	Lower bounds on the parameters. If a single number, this will be applied to all parameters. Default -Inf.
`upper`	Upper bounds on the parameters. If a single number, this will be applied to all parameters. Default Inf.
`control`	A list of controls for the algorithm. These are as for `nlxb()`.

... Any data needed for computation of the residual vector from the expression rhsexpression - lhsvar. Note that this is the negative of the usual residual, but the sum of squares is the same.

Details

wrapnls first attempts to solve the nonlinear sum of squares problem by using nlsmnq, then takes the parameters from that method to call nls.

Value

An object of type nls.

Note

Special notes, if any, will appear here.

Author(s)

John C Nash <nashjc@uottawa.ca>

See Also

Function nls(), packages optim and optimx.

Examples

```
cat("See examples in nlmrt-package.Rd\n")

## Not run:
cat("kvanderpoel.R test\n")
# require(nlmrt)
x<-c(1,3,5,7)
y<-c(37.98,11.68,3.65,3.93)
pks28<-data.frame(x=x,y=y)
fit0<-try(nls(y~(a+b*exp(1)^(-c*x)), data=pks28, start=c(a=0,b=1,c=1),
          trace=TRUE))
print(fit0)
cat("\n\n")
fit1<-nlxb(y~(a+b*exp(-c*x)), data=pks28, start=c(a=0,b=1,c=1), trace = TRUE)
print(fit1)
cat("\n\nor better\n")
fit2<-wrapnls(y~(a+b*exp(-c*x)), data=pks28, start=c(a=0,b=1,c=1),
              lower=-Inf, upper=Inf, trace = TRUE)


## End(Not run)
```

Index