Adding pracmanm to optimr() in package optimx

John C. Nash (profjcnash *at* gmail.com)

2023-09-05

The function `optimr()` from package `optimx` is the main engine that calls different function minimization solvers for the package. It is a very long piece of R code, and may seem extremely complicated. It is, however, essentially linear code. The sequence of actions is

- check the arguments in the function call and use them to set up necessary working functions for parameter scaled and "no dot-args" objective function, gradient and hessian functions. Where appropriate, numerical approximations are supplied. Inputs are checked for feasibility and methods are checked for applicability. This takes approximately 250 lines of code.
- via a long sequence of "if ... else" blocks, call the selected solver. This is now almost 1300 lines of code, but each block is more or less independent.
- gather the results and pack them in a return list `ans`.

Adding a new solver – pracmanm

Setup in ctrldefault.R

We need to inform package `optimx` that `pracmanm()` is available.

In `ctrldefault.R` this requires adding

- "pracmanm" to the vectors `allmeth` and `nogrmeth` (since it does not use gradients).
- in the same vector position as in `allmeth`, vector `truename` has element "nelder_mead", which is the name of the routine within package "pracma"
- in vector `allpkg` in the same vector position, we put element "pracma", since that is the package where we will find function `nelder_mead()`

Setup in NAMESPACE

We add the following line to the NAMESPACE file.

```
importFrom("pracma", "nelder_mead")
```

New block in optimr()

The following block – to which extra comments have been added here, is inserted into the file `optimr.R`. It can go between other "if ... else" blocks, but generally new methods are added just before the comment line

```
# ---  UNDEFINED METHOD ---
```

Here is the new code block:

```
## ----------------------------------------
  else if (method == "pracmanm") {# Use nelder_mead from pracma, Gao-Han adaptive NelderMead
    if (control$trace > 1) cat("pracmanm\n")
    ans <- list() # to define the answer object
```

```
    errmsg <- NA
    class(ans)[1] <- "undefined" # initial setting
    if (inherits(ans, "undefined")){
      if (control$have.bounds) {
        if (control$trace > 0) cat("pracmanm cannot handle bounds\n")
        errmsg <- "pracmanm cannot handle bounds\n"
        stop(errmsg)
        ans <- list()
        class(ans)[1] <- "try-error"
      } else {
        pnmtol <- 1.0e-08 # default in pracma
        if (! is.null(mcontrol$pracmanmtol)) pnmtol <- mcontrol$pracmanmtol
        tans <- try(pracma::nelder_mead(fn=efn, x0=spar, tol=pnmtol, maxfeval=control$maxfeval))
        # above line is the call to the nelder_mead routine in package pracma
      }
    }
    if (control$trace > 3) { # output interim answer for diagnostic purposes
        cat("interim answer:")
        str(tans)
    }
    if (! inherits(tans, "try-error")) { ## Need to check these carefully!!?
      ans$par <- tans$xmin*pscale # rescale parameters
      ans$value <- tans$fmin # and function
      ans$counts[1] <- tans$count # only function evaluation cound
      ans$counts[2] <- NA # no gradients evaluated
      ans$convergence<-0 # report successful exit
      attr(ans$convergence, "restarts") <- tans$info$restarts # extra info about restarts saved
      ans$hessian <- NULL # ensure hessian is empty
      ans$message <- tans$errmess # save any error message
      if (tans$count >= control$maxfeval) { ans$convergence <- 1 }
      tans <- NULL # cleanup
    } else {
      if (control$trace>0) cat("pracmanm failed for current problem \n")
      ans<-list() # ans not yet defined, so set as list
      ans$value <- control$badval
      ans$par <- rep(NA,npar)
      ans$convergence <- 9999 # failed in run
      ans$counts[1] <- NA
      ans$counts[2] <- NA # was [1] until 20211122
      ans$hessian <- NULL
      if (! is.na(errmsg)) ans$message <- errmsg
    }
  }  ## end if using pracmanm
```

Testing the result

We assume the resulting `optimr()` has been incorporated into the installed package `optimx`.

```
library(optimx)
fnR <- function (x, gs=100.0)
{
  n <- length(x)
  x1 <- x[2:n]
  x2 <- x[1:(n - 1)]
```

```
  sum(gs * (x1 - x2^2)^2 + (1 - x2)^2)
}
x0 <- rep(pi, 4)

cat("Extended Rosenbrock:\n")
```

```
## Extended Rosenbrock:
```

```
apnm0<-optimr(x0, fnR, method="pracmanm")
proptimr(apnm0)
```

```
## Result  apnm0 ( pracmanm  ->  (no_name) ) calc. min. = 2.480291e-06  at
## 0.9999931     1.000134     1.000308     1.00063     NA NA
## After  523  fn evals, and  0  gr evals and  0  hessian evals
## Termination code is  0 :
##
## ------------------------------------------------
```

```
apnm1<-optimr(x0, fnR, method="pracmanm", control=list(pracmanmtol=1e-13))
```

```
## Warning in optimr(x0, fnR, method = "pracmanm", control = list(pracmanmtol =
## 1e-13)): Special controls present for optimr with method pracmanm
```

```
proptimr(apnm1)
```

```
## Result  apnm1 ( pracmanm  ->  (no_name) ) calc. min. = 2.733038e-14  at
## 1     1     1     0.9999999     NA NA
## After  705  fn evals, and  0  gr evals and  0  hessian evals
## Termination code is  0 :
##
## ------------------------------------------------
```

```
maxfn<-function(x) {# fn to be MAXIMIZED
  # max = 10 at 1:n
  n<-length(x)
  ss<-seq(1,n)
  f<-10-sum((x-ss)^2)
  f
}
cat("maxfn:\n")
```

```
## maxfn:
```

```
x0<-rep(0.1,4)
runmax1<-opm(x0, maxfn, method=c("pracmanm", "Nelder-Mead"), control=list(maximize=TRUE,trace=0))
```

```
## Warning in optimr(par, fn, gr, hess = hess, method = meth, lower = lower, :
## optimr: use maximize control rather than fnscale
```

```
## Warning in optimr(par, fn, gr, hess = hess, method = meth, lower = lower, :
## optimr: use maximize control rather than fnscale
```

```
summary(runmax1)
```

```
##                  p1 s1       p2 s2       p3 s3       p4 s4     value fevals
## pracmanm    1.000052    1.999936    2.999681    3.999736    10.000000    192
## Nelder-Mead 1.000552    2.000731    2.999502    3.999518     9.999999    421
##             gevals hevals conv kkt1 kkt2 xtime
## pracmanm         0      0    0 TRUE TRUE 0.005
```

```
## Nelder-Mead      0      0    0 TRUE TRUE 0.003
```

Conclusion

`optimr()` is a long and sometimes messy R code. The example here shows how it is relatively straightforward to incorporate another solver into the function and hence into the package. Note that I have found it important to NULL objects that are not needed, as otherwise they may be used when such usage is not intended.