

Rvmmmin - an R implementation of the Fletcher(1970) variable metric method with bounds and masks

John Nash

2017-07-18

Rvmmmin description, examples and tests

Rvmmmin is an all-R version of the Fletcher-Nash variable metric nonlinear parameter optimization code of Fletcher (1970) as modified by Nash (1979).

This vignette is intended to show various features of the package, so it is rather detailed and “busy”. However, it is also hopefully helpful in showing how to use the method for more difficult problems.

Algorithm implementation

Fletcher’s variable metric method attempts to mimic Newton’s iteration for function minimization approximately.

Newton’s method starts with an original set of parameters x_0 . At a given iteration, which could be the first, we want to solve

$$x_{k+1} = x_k - H^{-1}g$$

where H is the Hessian and g is the gradient at x_k .

Newton’s method is unattractive in general function minimization situations because

- evaluating the Hessian is generally time consuming and error prone;
- solving the equation

$$H\delta = -g$$

(which is much less computational effort than inverting H), is still a lot of work which needs to be carried out every iteration.

While the base Newton algorithm is as given, generally we carry out some sort of line search along the search direction δ from the current iterate x_k . Indeed, many otherwise highly educated workers try to implement it without paying attention to safeguarding the iterations and ensuring appropriate progress towards a minimum.

Termination nuances

Termination variation with control tolerances

Let us use the Chebyquad test problem in $n=4$ parameters with different controls **eps** and **acctol** and tabulate the results to explore how our results change with different values of these program control inputs.

```
cyq.f <- function (x) {  
  rv<-cyq.res(x)  
  f<-sum(rv*rv)  
}
```

```

cyq.res <- function (x) {
# Fletcher's chebyquad function m = n -- residuals
  n<-length(x)
  res<-rep(0,n) # initialize
  for (i in 1:n) { #loop over resids
    rr<-0.0
    for (k in 1:n) {
z7<-1.0
z2<-2.0*x[k]-1.0
      z8<-z2
      j<-1
      while (j<i) {
        z6<-z7
        z7<-z8
        z8<-2*z2*z7-z6 # recurrence to compute Chebyshev polynomial
        j<-j+1
      } # end recurrence loop
      rr<-rr+z8
    } # end loop on k
    rr<-rr/n
    if (2*trunc(i/2) == i) { rr <- rr + 1.0/(i*i - 1) }
    res[i]<-rr
  } # end loop on i
  res
}

```

```

cyq.jac<- function (x) {
# Chebyquad Jacobian matrix
  n<-length(x)
  cj<-matrix(0.0, n, n)
  for (i in 1:n) { # loop over rows
    for (k in 1:n) { # loop over columns (parameters)
      z5<-0.0
      cj[i,k]<-2.0
      z8<-2.0*x[k]-1.0
      z2<-z8
      z7<-1.0
      j<- 1
      while (j<i) { # recurrence loop
        z4<-z5
        z5<-cj[i,k]
        cj[i,k]<-4.0*z8+2.0*z2*z5-z4
        z6<-z7
        z7<-z8
        z8<-2.0*z2*z7-z6
        j<- j+1
      } # end recurrence loop
      cj[i,k]<-cj[i,k]/n
    } # end loop on k
  } # end loop on i
  cj
}

```

```

cyq.g <- function (x) {
  cj<-cyq.jac(x)
  rv<-cyq.res(x)
  gg<- as.vector(2.0* rv %*% cj)
}

require(Rvmmmin)

## Loading required package: Rvmmmin

nn <- 4
xx0 <- 1:nn
xx0 <- xx0 / (nn+1.0) # Initial value suggested by Fletcher

# cat("aed\n")
# aed <- Rvmmminu(xx0, cyq.f, cyq.g, control=list(trace=2, checkgrad=FALSE))
# print(aed)
#=====
# Now build a table of results for different values of eps and acc
veps <- c(1e-3, 1e-5, 1e-7, 1e-9, 1e-11)
vacc <- c(.1, .01, .001, .0001, .00001, .000001)
resdf <- data.frame(eps=NA, acctol=NA, nf=NA, ng=NA, fval=NA, gnorm=NA)
for (eps in veps) {
  for (acctol in vacc) {
    ans <- Rvmmminu(xx0, cyq.f, cyq.g,
      control=list(eps=eps, acctol=acctol, trace=0))
    gn <- as.numeric(crossprod(cyq.g(ans$par)))
    resdf <- rbind(resdf,
      c(eps, acctol, ans$counts[1], ans$counts[2], ans$value, gn))
  }
}
resdf <- resdf[-1,]
# Display the function value found for different tolerances
xtabs(formula = fval ~ acctol + eps, data=resdf)

```

```

##          eps
## acctol    1e-11    1e-09    1e-07    1e-05    0.001
## 1e-06 3.964816e-29 3.964816e-29 3.964816e-29 7.049696e-24 7.486504e-15
## 1e-05 3.964816e-29 3.964816e-29 3.964816e-29 7.049696e-24 7.486504e-15
## 1e-04 3.964816e-29 3.964816e-29 3.964816e-29 7.049696e-24 7.486504e-15
## 0.001 3.964816e-29 3.964816e-29 3.964816e-29 7.049696e-24 7.486504e-15
## 0.01  3.964816e-29 3.964816e-29 3.964816e-29 7.049696e-24 7.486504e-15
## 0.1   3.964816e-29 3.964816e-29 3.964816e-29 7.049696e-24 7.486504e-15

```

```

# Display the gradient norm found for different tolerances
xtabs(formula = gnorm ~ acctol + eps, data=resdf)

```

```

##          eps
## acctol    1e-11    1e-09    1e-07    1e-05    0.001
## 1e-06 7.809261e-30 7.809261e-30 7.809261e-30 3.645064e-22 1.089927e-13
## 1e-05 7.809261e-30 7.809261e-30 7.809261e-30 3.645064e-22 1.089927e-13
## 1e-04 7.809261e-30 7.809261e-30 7.809261e-30 3.645064e-22 1.089927e-13
## 0.001 7.809261e-30 7.809261e-30 7.809261e-30 3.645064e-22 1.089927e-13
## 0.01  7.809261e-30 7.809261e-30 7.809261e-30 3.645064e-22 1.089927e-13
## 0.1   7.809261e-30 7.809261e-30 7.809261e-30 3.645064e-22 1.089927e-13

```

```
# Display the number of function evaluations used for different tolerances
xtabs(formula = nf ~ acctol + eps, data=resdf)
```

```
##          eps
## acctol 1e-11 1e-09 1e-07 1e-05 0.001
## 1e-06   22   22   22   17   12
## 1e-05   22   22   22   17   12
## 1e-04   22   22   22   17   12
## 0.001   22   22   22   17   12
## 0.01    22   22   22   17   12
## 0.1     22   22   22   17   12
```

```
# Display the number of gradient evaluations used for different tolerances
xtabs(formula = ng ~ acctol + eps, data=resdf)
```

```
##          eps
## acctol 1e-11 1e-09 1e-07 1e-05 0.001
## 1e-06   15   15   15   12   9
## 1e-05   15   15   15   12   9
## 1e-04   15   15   15   12   9
## 0.001   15   15   15   12   9
## 0.01    15   15   15   12   9
## 0.1     15   15   15   12   9
```

Here – and we caution that this is but a single instance of a single test problem – the differences in results and level of effort to obtain them are regulated by the values of `eps` only. This control is used to judge the size of the gradient norm and the gradient projection on the search vector.

Problems of function scale

One of the more difficult aspects of termination decisions is that we need to decide when we have a “nearly” zero gradient. However, this “zero gradient” is relative to the overall scale of the function. Let us see what happens when we consider solving a problem where the function scale is adjustable. Note that we multiply the constant sequence `yy` by `pi/4` to avoid integer values which may give results that are fortuitously better than may be normally found.

```
require(Rvmmmin)
sq<-function(x, exfs=1){
  nn<-length(x)
  yy<-(1:nn)*pi/4
  f<-(10^exfs)*sum((yy-x)^2)
  f
}
sq.g <- function(x, exfs=1){
  nn<-length(x)
  yy<-(1:nn)*pi/4
  gg<- 2*(x - yy)*(10^exfs)
}
require(Rvmmmin)
nn <- 4
xx0 <- rep(pi, nn) # crude start

# Now build a table of results for different values of eps and acc
veps <- c(1e-3, 1e-5, 1e-7, 1e-9, 1e-11)
exfsi <- 1:6
```

```

resdf <- data.frame(eps=NA, exfs=NA, nf=NA, ng=NA, fval=NA, gnorm=NA)
for (eps in veps) {
  for (exfs in exfsi) {
    ans <- Rvminu(xx0, sq, sq.g,
                 control=list(eps=eps, trace=0), exfs=exfs)
    gn <- as.numeric(crossprod(sq.g(ans$par)))
    resdf <- rbind(resdf,
                  c(eps, exfs, ans$counts[1], ans$counts[2], ans$value, gn))
  }
}
resdf <- resdf[-1,]
# Display the function value found for different tolerances
xtabs(formula = fval ~ exfs + eps, data=resdf)

```

```

##      eps
## exfs  1e-11  1e-09  1e-07  1e-05  0.001
## 1 8.246062e-29 8.246062e-29 8.246062e-29 2.001735e-28 2.001735e-28
## 2 0.000000e+00 0.000000e+00 0.000000e+00 1.772768e-25 1.772768e-25
## 3 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 1.210198e-22
## 4 1.232595e-28 1.232595e-28 1.232595e-28 1.232595e-28 5.241000e-20
## 5 4.930381e-27 4.930381e-27 4.930381e-27 4.930381e-27 4.930381e-27
## 6 1.232595e-26 1.232595e-26 1.232595e-26 1.232595e-26 1.232595e-26

```

```

# Display the gradient norm found for different tolerances
xtabs(formula = gnorm ~ exfs + eps, data=resdf)

```

```

##      eps
## exfs  1e-11  1e-09  1e-07  1e-05  0.001
## 1 1.602374e-27 1.602374e-27 1.602374e-27 8.006938e-27 8.006938e-27
## 2 0.000000e+00 0.000000e+00 0.000000e+00 7.091071e-25 7.091071e-25
## 3 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 4.840792e-23
## 4 0.000000e+00 0.000000e+00 0.000000e+00 4.930381e-30 2.096400e-21
## 5 0.000000e+00 0.000000e+00 0.000000e+00 1.972152e-29 1.972152e-29
## 6 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 4.930381e-30

```

```

# Display the number of function evaluations used for different tolerances
xtabs(formula = nf ~ exfs + eps, data=resdf)

```

```

##      eps
## exfs 1e-11 1e-09 1e-07 1e-05 0.001
## 1 11 11 11 5 5
## 2 16 16 16 6 6
## 3 22 22 22 22 8
## 4 30 30 30 26 9
## 5 36 36 36 30 30
## 6 42 42 42 42 35

```

```

# Display the number of gradient evaluations used for different tolerances
xtabs(formula = ng ~ exfs + eps, data=resdf)

```

```

##      eps
## exfs 1e-11 1e-09 1e-07 1e-05 0.001
## 1 4 4 4 3 3
## 2 7 7 7 3 3
## 3 7 7 7 7 3
## 4 7 7 7 7 3

```

```
##    5    7    7    7    7    7
##    6    7    7    7    7    7
```

The general tendency here is for the amount of work in terms of function evaluations to rise with the function scale and with tighter (smaller) test tolerances, while the quality of the solution is poorer with larger scale and also larger (looser) tolerances. However, some exceptions can be seen, though the overall quality of solutions (function and gradient norm) is very good. Moreover, the number of gradient evaluations does not climb notably with the scale or inverse tolerance.

Problems of parameter scale

There are similar issues of parameter scaling. Let us look at very simple sum of squares function where we scale the parameters in a nasty way.

```
ssq.f<-function(x){
  nn<-length(x)
  yy <- 1:nn
  f<-sum((yy-x/10^yy)^2)
  f
}
ssq.g <- function(x){
  nn<-length(x)
  yy<-1:nn
  gg<- 2*(x/10^yy - yy)*(1/10^yy)
}

xy <- c(1, 1/10, 1/100, 1/1000)
# note: gradient was checked using numDeriv
veps <- c(1e-3, 1e-5, 1e-7, 1e-9, 1e-11)
vacc <- c(.1, .01, .001, .0001, .00001, .000001)
resdf <- data.frame(eps=NA, acctol=NA, nf=NA, ng=NA, fval=NA, gnorm=NA)
for (eps in veps) {
  for (acctol in vacc) {
    ans <- Rvmminu(xy, ssq.f, ssq.g,
      control=list(eps=eps, acctol=acctol, trace=0))
    gn <- as.numeric(crossprod(ssq.g(ans$par)))
    resdf <- rbind(resdf,
      c(eps, acctol, ans$counts[1], ans$counts[2], ans$value, gn))
  }
}
resdf <- resdf[-1,]
# Display the function value found for different tolerances
xtabs(formula = fval ~ acctol + eps, data=resdf)
```

```
##          eps
## acctol    1e-11    1e-09    1e-07    1e-05    0.001
## 1e-06 0.000000e+00 0.000000e+00 1.475416e-29 5.767419e-19 8.977439e-11
## 1e-05 0.000000e+00 0.000000e+00 1.475416e-29 5.767419e-19 8.977439e-11
## 1e-04 0.000000e+00 0.000000e+00 1.475416e-29 5.767419e-19 8.977439e-11
## 0.001 0.000000e+00 0.000000e+00 1.475416e-29 5.767419e-19 8.977439e-11
## 0.01  0.000000e+00 0.000000e+00 1.475416e-29 5.767419e-19 8.977439e-11
## 0.1   0.000000e+00 0.000000e+00 1.475416e-29 5.767419e-19 8.977439e-11
```

```
# Display the gradient norm found for different tolerances
xtabs(formula = gnorm ~ acctol + eps, data=resdf)
```

```
##          eps
## acctol   1e-11   1e-09   1e-07   1e-05   0.001
## 1e-06 0.000000e+00 0.000000e+00 7.783028e-33 3.430257e-23 3.473135e-14
## 1e-05 0.000000e+00 0.000000e+00 7.783028e-33 3.430257e-23 3.473135e-14
## 1e-04 0.000000e+00 0.000000e+00 7.783028e-33 3.430257e-23 3.473135e-14
## 0.001 0.000000e+00 0.000000e+00 7.783028e-33 3.430257e-23 3.473135e-14
## 0.01 0.000000e+00 0.000000e+00 7.783028e-33 3.430257e-23 3.473135e-14
## 0.1 0.000000e+00 0.000000e+00 7.783028e-33 3.430257e-23 3.473135e-14
```

```
# Display the number of function evaluations used for different tolerances
xtabs(formula = nf ~ acctol + eps, data=resdf)
```

```
##          eps
## acctol 1e-11 1e-09 1e-07 1e-05 0.001
## 1e-06   56   56   55   53   51
## 1e-05   56   56   55   53   51
## 1e-04   56   56   55   53   51
## 0.001   56   56   55   53   51
## 0.01    56   56   55   53   51
## 0.1     56   56   55   53   51
```

```
# Display the number of gradient evaluations used for different tolerances
xtabs(formula = ng ~ acctol + eps, data=resdf)
```

```
##          eps
## acctol 1e-11 1e-09 1e-07 1e-05 0.001
## 1e-06   56   56   55   53   51
## 1e-05   56   56   55   53   51
## 1e-04   56   56   55   53   51
## 0.001   56   56   55   53   51
## 0.01    56   56   55   53   51
## 0.1     56   56   55   53   51
```

The results above suggest that parameter scaling is not much of a problem. Actually, these are the very best results I have found with any method for this problem, which is actually rather nasty. I suggest trying this problem on your favourite optimizer. Alternatively, use the package `optimr` and run the function `opm()` with `method="ALL"`.

Weeds problem with random starts

This notorious problem (see Nash (1979), page 120, Nash (2014), page 205, for details under the heading **Hobbs Weeds problem**) is small but generally difficult due to the possibility of bad scaling of both function and parameters and a near-singular Hessian in the original parameterization.

The Fletcher variable metric method can solve this problem quite well, though default termination settings should be overridden. It is important to ensure there are enough iterations to allow the method to “grind” at the problem. If one uses default settings for `maxit` in `optim:BFGS`, then the success rate drops to less than 2/3 of cases tried below.

Below we use 100 “random” starting points for both `Rvmmmin` and the `optim:BFGS` minimizers (which should be, but are not quite, the same).

```
## hobbsstarts.R -- starting points for Hobbs problem
hobbs.f<- function(x){ ## Hobbs weeds problem -- function
  if (abs(12*x[3]) > 500) { # check computability
    fbad<- .Machine$double.xmax
    return(fbad)
  }
}
```

```

}
res<-hobbs.res(x)
f<-sum(res*res)
##   cat("fval =",f,"\n")
##   f
}
hobbs.res<-function(x){ # 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)
  t<-1:12
  if(abs(12*x[3])>50) {
    res<-rep(Inf,12)
  } else {
    res<-x[1]/(1+x[2]*exp(-x[3]*t)) - y
  }
}

hobbs.jac<-function(x){ # Jacobian of Hobbs weeds problem
  jj<-matrix(0.0, 12, 3)
  t<-1:12
  yy<-exp(-x[3]*t)
  zz<-1.0/(1+x[2]*yy)
  jj[t,1] <- zz
  jj[t,2] <- -x[1]*zz*zz*yy
  jj[t,3] <- x[1]*zz*zz*yy*x[2]*t
  return(jj)
}

hobbs.g<-function(x){ # gradient of Hobbs weeds problem
# NOT EFFICIENT TO CALL AGAIN
  jj<-hobbs.jac(x)
  res<-hobbs.res(x)
  gg<-as.vector(2.*t(jj) %*% res)
  return(gg)
}

require(Rvmin)
set.seed(12345)
nrun<-100
sstart<-matrix(runif(3*nrun, 0, 5), nrow=nrun, ncol=3)
ustart<-sstart %*% diag(c(100, 10, 0.1))
nsuccR <- 0
nsucc0 <- 0
vRvm <- rep(NA, nrun)
voptim <- vRvm
fRvm <- vRvm
gRvm <- vRvm
foptim <- vRvm
goptim <- vRvm

for (irun in 1:nrun) {
  us <- ustart[irun,]

```



```

# print(us)
# ans <- Rvminu(us, hobbs.f, hobbs.g, control=list(trace=1))
# ans <- optim(us, hobbs.f, hobbs.g, method="BFGS")
ans <- Rvminu(us, hobbs.f, hobbs.g, control=list(trace=0))
ao <- optim(us, hobbs.f, hobbs.g, method="BFGS",
            control=list(maxit=3000))
# ensure does not max function out

# cat(irun, " Rvminu value =", ans$value, " optim:BFGS value=", ao$value, "\n")
if (ans$value < 2.5879) nsuccR <- nsuccR + 1
if (ao$value < 2.5879) nsucc0 <- nsucc0 + 1
# tmp <- readline()
vRvm[irun] <- ans$value
voptim[irun] <- ao$value
fRvm[irun] <- ans$counts[1]
gRvm[irun] <- ans$counts[2]
foptim[irun] <- ao$counts[1]
goptim[irun] <- ao$counts[2]
}

```

```

## Warning in Rvminu(us, hobbs.f, hobbs.g, control = list(trace = 0)): Too
## many gradient evaluations

```

```

cat("Rvminu: number of successes=", nsuccR, " propn=", nsuccR/nrun, "\n")

```

```

## Rvminu: number of successes= 100 propn= 1

```

```

cat("optim:BFGS no. of successes=", nsucc0, " propn=", nsucc0/nrun, "\n")

```

```

## optim:BFGS no. of successes= 99 propn= 0.99

```

```

fgc <- data.frame(fRvm, foptim, gRvm, goptim)
summary(fgc)

```

```

##      fRvm      foptim      gRvm      goptim
## Min.   : 41.0   Min.   : 58.0   Min.   : 26.00  Min.   : 16.0
## 1st Qu.:105.8  1st Qu.: 140.5  1st Qu.: 39.00  1st Qu.: 53.0
## Median :155.5  Median : 184.0  Median : 53.00  Median : 68.5
## Mean   :205.7  Mean   : 323.5  Mean   : 59.57  Mean   :131.2
## 3rd Qu.:258.0  3rd Qu.: 453.5  3rd Qu.: 66.00  3rd Qu.:178.8
## Max.   :920.0  Max.   :1427.0  Max.   :507.00  Max.   :610.0

```

From this summary, it appears that Rvmin, on average, uses fewer gradient and function evaluations to achieve the desired result.

For comparison, we now re-run the example with default settings for maxit in optim:BFGS.

```

nsuccR <- 0
nsucc0 <- 0
for (irun in 1:nrun) {
  us <- ustart[irun,]
  # print(us)
  # ans <- Rvminu(us, hobbs.f, hobbs.g, control=list(trace=1))
  # ans <- optim(us, hobbs.f, hobbs.g, method="BFGS")
  ans <- Rvminu(us, hobbs.f, hobbs.g, control=list(trace=0))
  ao <- optim(us, hobbs.f, hobbs.g, method="BFGS")
}

```

```

# ensure does not max function out

# cat(irun, " Rvmminu value =", ans$value, " optim:BFGS value=", ao$value, "\n")
if (ans$value < 2.5879) nsuccR <- nsuccR + 1
if (ao$value < 2.5879) nsucc0 <- nsucc0 + 1
# tmp <- readline()
vRvm[irun] <- ans$value
voptim[irun] <- ao$value
fRvm[irun] <- ans$counts[1]
gRvm[irun] <- ans$counts[2]
foptim[irun] <- ao$counts[1]
goptim[irun] <- ao$counts[2]
}

```

```

## Warning in Rvmminu(us, hobbs.f, hobbs.g, control = list(trace = 0)): Too
## many gradient evaluations

```

```

cat("Rvmminu: number of successes=", nsuccR, " propn=", nsuccR/nrun, "\n")

```

```

## Rvmminu: number of successes= 100 propn= 1

```

```

cat("optim:BFGS no. of successes=", nsucc0, " propn=", nsucc0/nrun, "\n")

```

```

## optim:BFGS no. of successes= 64 propn= 0.64

```

```

fgc <- data.frame(fRvm, foptim, gRvm, goptim)
summary(fgc)

```

```

##      fRvm      foptim      gRvm      goptim
## Min.   : 41.0   Min.   : 58.0   Min.   : 26.00  Min.   : 16.00
## 1st Qu.:105.8   1st Qu.:140.5   1st Qu.: 39.00  1st Qu.: 53.00
## Median :155.5   Median :184.0   Median : 53.00  Median : 68.50
## Mean   :205.7   Mean   :184.0   Mean   : 59.57  Mean   : 71.73
## 3rd Qu.:258.0   3rd Qu.:236.0   3rd Qu.: 66.00  3rd Qu.:100.00
## Max.   :920.0   Max.   :425.0   Max.   :507.00  Max.   :100.00

```

Bounds and masks

Let us make sure that Rvmminb is doing the right thing with bounds and masks. (This is actually a test in the package.)

Bounds

```

bt.f<-function(x){
  sum(x*x)
}

bt.g<-function(x){
  gg<-2.0*x
}

lower <- c(0, 1, 2, 3, 4)
upper <- c(2, 3, 4, 5, 6)

```

```

bdmsk <- rep(1,5)
xx <- rep(0,5) # out of bounds
ans <- Rvmmmin(xx, bt.f, bt.g, lower=lower, upper=upper, bdmsk=bdmsk)

## Warning in Rvmmmin(xx, bt.f, bt.g, lower = lower, upper = upper, bdmsk =
## bdmsk): Parameter out of bounds has been moved to nearest bound
ans

## $par
## [1] 0 1 2 3 4
##
## $value
## [1] 30
##
## $counts
## function gradient
##      1      1
##
## $convergence
## [1] 0
##
## $message
## [1] "Rvmmminb appears to have converged"
##
## $bdmsk
## [1] 1 -3 -3 -3 -3

```

Masks

Here we fix one or more paramters and minimize over the rest.

```

sq.f<-function(x){
  nn<-length(x)
  yy<-1:nn
  f<-sum((yy-x)^2)
  f
}
sq.g <- function(x){
  nn<-length(x)
  yy<-1:nn
  gg<- 2*(x - yy)
}

xx0 <- rep(pi,3)
bdmsk <- c(1, 0, 1) # Middle parameter fixed at pi
cat("Check final function value (pi-2)^2 = ", (pi-2)^2,"\n")

## Check final function value (pi-2)^2 = 1.303234

require(Rvmmmin)
ans <- Rvmmmin(xx0, sq.f, sq.g, lower=-Inf, upper=Inf, bdmsk=bdmsk,
               control=list(trace=2))

## Bounds: nolower = TRUE  noupper = TRUE  bounds = TRUE
## Gradient test with tolerance = 6.055454e-06

```

```

## Analytic gradient uses function gr
## function at parameters = 5.909701 with attributes:
## NULL
## Compute analytic gradient
## [1] 4.2831853 2.2831853 0.2831853
## Compute numeric gradient
## [1] 4.2831853 2.2831853 0.2831853
## gradient test tolerance = 6.055454e-06 fval= 5.909701
## compare to max(abs(gn-ga))/(1+abs(fval)) = 3.242827e-12
## admissible = TRUE
## maskadded = FALSE
## parchanged = FALSE
## Bounds: nolower = FALSE noupper = FALSE bounds = TRUE
## Rvmmminb -- J C Nash 2009-2015 - an R implementation of Alg 21
## Problem of size n= 3 Dot arguments:
## list()
## Initial fn= 5.909701
## 1 1 5.909701
## Gradproj = -18.42587
## reset steplength= 1
## *reset steplength= 0.2
## ig= 2 gnorm= 2.575522 3 2 2.961562
## Gradproj = -15.04576
## reset steplength= 1
## *reset steplength= 0.2
## ig= 3 gnorm= 0.23879 5 3 1.317489
## Gradproj = -0.02851034
## reset steplength= 1
## ig= 4 gnorm= 0 Small gradient norm
## Seem to be done Rvmmminb

```

```
ans
```

```

## $par
## [1] 1.000000 3.141593 3.000000
##
## $value
## [1] 1.303234
##
## $counts
## function gradient
##      6      4
##
## $convergence
## [1] 2
##
## $message
## [1] "Rvmmminb appears to have converged"
##
## $bdmsk
## [1] 1 0 1

```

```
ansnog <- Rvmmmin(xx0, sq.f, lower=-Inf, upper=Inf, bdmsk=bdmsk,
                 control=list(trace=2))
```

```
## Bounds: nolower = TRUE noupper = TRUE bounds = TRUE
```

```

## WARNING: forward gradient approximation being used
## admissible = TRUE
## maskadded = FALSE
## parchanged = FALSE
## Bounds: nolower = FALSE  noupper = FALSE  bounds = TRUE
## Rvmmminb -- J C Nash 2009-2015 - an R implementation of Alg 21
## Problem of size n= 3  Dot arguments:
## list()
## WARNING: using gradient approximation ' grfwd '
## Initial fn= 5.909701
## 1 1 5.909701
## Gradproj = -18.42587
## reset steplength= 1
## *reset steplength= 0.2
## ig= 2  gnorm= 2.575522      3  2  2.961562
## Gradproj = -15.04576
## reset steplength= 1
## *reset steplength= 0.2
## ig= 3  gnorm= 0.23879      5  3  1.317489
## Gradproj = -0.02851034
## reset steplength= 1
## ig= 4  gnorm= 2.668644e-08  6  4  1.303234
## Gradproj = -4.446061e-16
## reset steplength= 1
## *reset steplength= 0.2
## *reset steplength= 0.04
## *reset steplength= 0.008
## *reset steplength= 0.0016
## *reset steplength= 0.00032
## *reset steplength= 6.4e-05
## *reset steplength= 1.28e-05
## *reset steplength= 2.56e-06
## *reset steplength= 5.12e-07
## *reset steplength= 1.024e-07
## Unchanged in step redn
## No acceptable point
## Reset to gradient search
## 16 4 1.303234
## Gradproj = -7.121661e-16
## reset steplength= 1
## *reset steplength= 0.2
## *reset steplength= 0.04
## *reset steplength= 0.008
## *reset steplength= 0.0016
## *reset steplength= 0.00032
## *reset steplength= 6.4e-05
## *reset steplength= 1.28e-05
## *reset steplength= 2.56e-06
## *reset steplength= 5.12e-07
## *reset steplength= 1.024e-07
## Unchanged in step redn
## No acceptable point
## Converged
## Seem to be done Rvmmminb

```

```
ansnog
```

```
## $par
## [1] 1.000000 3.141593 3.000000
##
## $value
## [1] 1.303234
##
## $counts
## function gradient
##      26      4
##
## $convergence
## [1] 0
##
## $message
## [1] "Rvminb appears to have converged"
##
## $bdmsk
## [1] 1 0 1
```

References

- Fletcher, R. 1970. "A New Approach to Variable Metric Algorithms." *Computer Journal* 13 (3): 317–22.
- Nash, John C. 1979. *Compact Numerical Methods for Computers: Linear Algebra and Function Minimisation*. Bristol: Adam Hilger.
- . 2014. *Nonlinear Parameter Optimization Using R Tools*. Book. John Wiley & Sons: Chichester. <http://www.wiley.com//legacy/wileychi/nash/>.