Bug in Modified Cholesky Decomposition Used in nlm()

Marie Böhnstedt (boehnstedt@demogr.mpg.de)
Max Planck Institute for Demographic Research
Konrad-Zuse-Straße 1, 18057 Rostock, Germany
March 29, 2017

Summary: To determine the minimum of a (nonlinear) function analytic derivatives – gradient and Hessian matrix – can be provided to the function nlm(). The following erroneous behavior was observed in several test settings: If only the analytic gradient is provided (but the Hessian is approximated numerically) the algorithm converges, however, if both the gradient and the Hessian are provided analytically the algorithm fails to converge (in a reasonable number of iterations). This counter-intuitive behavior results from an erroneous implementation of a modified Cholesky decomposition of the analytic Hessian matrix.

1 Reproducing the Bug

Function nlm() performs a Newton-type algorithm to minimize an objective function. The gradient and Hessian of the objective function are either approximated numerically or the analytic expression of only the gradient or both the gradient and the Hessian can be specified by the user. In theory, providing more information should improve the performance of the algorithm. In practice, using common test functions, the nlm()-algorithm converges if only the analytic gradient is provided, but it does not converge if additionally the analytic Hessian is provided.

1.1 Example 1

This bug is already present in the demo of the nlm()-function, when minimizing the Rosenbrock banana valley function

\[ f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2, \quad x = (x_1, x_2)^T \in \mathbb{R}^2, \]

which has a minimum of 0 at (1,1).

```r
> fg <- function(x){ # analytic gradient only
+     gr <- function(x1, x2) c(-400*x1*(x2 - x1*x1)-2*(1-x1), 200*(x2 - x1*x1))
+     x1 <- x[1]; x2 <- x[2]
+     res <- 100*(x2 - x1*x1)^2 + (1-x1)^2
+     attr(res, "gradient") <- gr(x1, x2)
+     return(res)
+ }
> nlm.fg <- nlm(fg, c(-1.2, 1))

> fgh <- function(x){ # analytic gradient and Hessian
+     gr <- function(x1, x2) c(-400*x1*(x2 - x1*x1) - 2*(1-x1), 200*(x2 - x1*x1))
+     h <- function(x1, x2){
+         a11 <- 2 - 400*x2 + 1200*x1*x1
+         a21 <- -400*x1
+         matrix(c(a11, a21, a21, 200), 2, 2)
+     }
+ ```
+ x1 <- x[1]; x2 <- x[2]; res <- 100*(x2 - x1*x1)^2 + (1-x1)^2
+ attr(res, "gradient") <- gr(x1, x2)
+ attr(res, "hessian") <- h(x1, x2)
+ return(res)
+
> nlm.fgh <- nlm(fgh, c(-1.2,1))

Thus, using only the analytic gradient of the Rosenbrock function, the algorithm terminates after 24 iterations, because the relative gradient is close to zero. Using additionally the analytic Hessian, the algorithm does not converge, but terminates after the maximum of 100 steps at a point far from the optimum.

1.2 Example 2

The same behavior is found when using `nlm()` for minimization of the Wood function (see for example Dennis and Schnabel, 1996, p. 363)

\[
f(x) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2 + 90(x_3^2 - x_4)^2 + (1 - x_3)^2 \\
+ 10.1[(1 - x_2)^2 + (1 - x_4)^2] + 19.8(1 - x_2)(1 - x_4), \quad x = (x_1, x_2, x_3, x_4)^T \in \mathbb{R}^4,
\]

which has a minimum of 0 at (1, 1, 1, 1).

> #gradient:
> woodfunc.g <- function(x){
+ g1 <- 400*x[1]^3-400*x[1]*x[2]+2*x[1]-2
+ g2 <- -200*x[1]^2+220.2*x[2]+19.8*x[4]-40
+ g3 <- 360*x[3]^3-360*x[3]*x[4]+2*x[3]-2
+ g4 <- -180*x[3]^2+200.2*x[4]+19.8*x[2]-40
+ return(c(g1,g2,g3,g4))
+ }

2
+ attr(res,"gradient") <- wfg(x)
+ return(res)
+ }
> nlm.wfg <- nlm(p=c(-3,-1,-3,-1),woodfunc.g)
> #hessian:
> wfh <- function(x){
+ h11 <- 1200*x[1]^2-400*x[2]+2; h12 <- -400*x[1]; h13 <- h14 <- 0
+ h22 <- 220.2; h23 <- 0; h24 <- 19.8
+ h33 <- 1080*x[3]^2-360*x[4]+2; h34 <- -360*x[3]
+ h44 <- 200.2
+ H <- matrix(c(h11,h12,h13,h14,h22,h23,h24,
+ h13,h23,h33,h34,h14,h24,h34,h44),ncol=4)
+ return(H)
+ }
> woodfunc.gh <- function(x){
+ res <- 100*(x[1]^2-x[2])^2+(1-x[1])^2+90*(x[3]^2-x[4])^2+(1-x[3])^2+
+ 10.1*((1-x[2])^2+(1-x[4])^2)+19.8*(1-x[2])*(1-x[4])
+ attr(res,"gradient") <- wfg(x)
+ attr(res,"hessian") <- wfh(x)
+ return(res)
+ }
> nlm.wfgh <- nlm(p=c(-3,-1,-3,-1),woodfunc.gh)

> nlm.wfg
$minimum
[1] 1.550316e-17

$estimate
[1] 1 1 1 1

$gradient
[1] 6.046878e-08
[2] -3.166417e-08
[3] -6.902938e-08
[4] 2.708807e-08

$code
[1] 1

$iterations
[1] 33

> nlm.wfgh
$minimum
[1] 7.876867

$estimate
[1] -0.9545583 0.9214171
[3] -0.9827179 0.9769463

$gradient
[1] -0.0009631411
[2] 0.0032627860
[3] 0.0010866021
[4] -0.0034870511

$code
[1] 4

$iterations
[1] 100

In section 2 we will explain that this results from a bug in the C-function choldc() in R\src\appl\uncmin.c, which is used at each iteration to perform a modified Cholesky decomposition of the user-specified analytic Hessian.
1.3 Session Info

All computations were performed using R version 3.3.1.

```
> sessionInfo()
```

R version 3.3.1 (2016-06-21)
Platform: x86_64-w64-mingw32/x64 (64-bit)
Running under: Windows 7 x64 (build 7601) Service Pack 1

locale:
[1] LC_COLLATE=English_United States.1252
[2] LC_CTYPE=English_United States.1252
[3] LC_MONETARY=English_United States.1252
[4] LC_NUMERIC=C
[5] LC_TIME=English_United States.1252

attached base packages:
[1] stats graphics
[3] grDevices utils
[5] datasets methods
[7] base

loaded via a namespace (and not attached):
[1] tools_3.3.1

2 Computational Methods of nlm()

Information on the nlm()-routine can be gathered from the nlm()-reference Dennis and Schnabel (1996) as well as by directly investigating the nlm()-source code, primarily the routines in uncmin.c.

2.1 Algorithm

Basically, nlm() is based on (a modified) Newton’s method: starting from an initial estimate \( x_0 \), the minimum \( x^* \) of the objective function \( f \) is found as a zero of the gradient \( \nabla f(x) \) by iterating

\[
x_{k+1} = x_k + s_N^k \quad \text{with} \quad H(x_k)s_N^k = -\nabla f(x_k).
\]

Here, \( H(x_k) = \nabla^2 f(x_k) + \mu_k I \) is a modification of the Hessian matrix \( \nabla^2 f(x_k) \) to ensure its positive definiteness. This is achieved via a perturbed Cholesky decomposition of \( \nabla^2 f(x_k) \), which results in \( \nabla^2 f(x_k) + D = LL^T \) with a diagonal matrix \( D \). If \( \nabla^2 f(x_k) \) is safely positive definite, then \( D = 0 \) and \( \mu = 0 \). Otherwise a minimum \( \mu \) is calculated (based on the Gershgorin circle theorem and the maximum element of \( D \)), for which \( H(x_k) = \nabla^2 f(x_k) + \mu I \) is safely positive definite and a (normal) Cholesky decomposition can be found to solve \( H(x_k)s_N^k = -\nabla f(x_k) \). Recall that, for a symmetric and positive definite \( n \times n \) matrix \( H \), we can find the Cholesky decomposition \( H = LL^T \), with a lower triangular matrix \( L \) and its transpose \( L^T \), by filling \( L \) column-wise (Cholesky-Crout algorithm) with

\[
l_{ii} = \sqrt{h_{ii} - \sum_{k<i} l_{ik}^2} \quad \text{and} \quad l_{ij} = \frac{1}{l_{jj}} \left( h_{ji} - \sum_{k<j} l_{ik}l_{jk} \right) \quad \text{for} \quad i = 1, \ldots, n, \quad j < i.
\]
2.2 Bug in the Implementation of the Cholesky Decomposition

We translated most of the C-code called by \texttt{nlm()} into \texttt{R}-functions to further examine the computational methods. In particular, we focused on the functions \texttt{chlhsn()} and \texttt{choldc()} for obtaining the modified Hessian matrix $H(x_k)$. We ran the \texttt{R}-code on the Rosenbrock function using both the analytic gradient and Hessian and obtained the same results as with a call to \texttt{nlm()}, suggesting that we reproduced the code correctly.

Function \texttt{chlhsn()} creates a model Hessian based on the current (estimate of) the Hessian $H$. \texttt{epsm} is the machine’s double precision and \texttt{sx} is a scaling vector for the argument, defaulting to a vector of ones. After a check for (almost) negative diagonal elements (where a positive result would already lead to a modification of the Hessian matrix), function \texttt{choldc()} is called, which performs a perturbed Cholesky decomposition of $H$. \texttt{choldc()} attempts a normal Cholesky decomposition, but will augment the diagonal elements of $H$, if a diagonal element of $L$ falls below some threshold. The elements of $L$ are written into the corresponding elements of $H$. The maximum added diagonal element is returned as \texttt{addmax}. (In the code given here, some lines are skipped, indicated by \texttt{#...})

```r
> chlhsn <- function(H,epsm,sx){
+ n <- nrow(H); H <- H/(sx%*%t(sx)); tol <- sqrt(epsm)
+ diagmx <- max(diag(H)); diagmn <- min(diag(H)); posmax <- max(0,diagmx)
+ if(diagmn<posmax*tol) #... FALSE for pd matrices
+ udiag <- diag(H)
+ c1 <- choldc(H,diagmx,tol) #first attempt of Cholesky decomposition
+ H <- c1$H; addmax <- c1$addmax
+ if(addmax>0){ #if H has to be modified to be safely pd
+ diag(H) <- udiag; H[lower.tri(H)] <- H[upper.tri(H)] #restore H
+ evmin <- 0; evmax <- H[1,1]
+ for(i in 1:n){ #Gershgorin circle theorem
+ offrow <- sum(abs(H[i,-i]))
+ tmp <- H[i,i]-offrow; evmin <- min(evmin,tmp)
+ tmp <- H[i,i]+offrow; evmax <- max(evmax,tmp)
+ }
+ sdd <- tol*(evmax-evmin)-evmin
+ amu <- min(sdd,addmax) #minimal mu to be added to Hessian
+ diag(H) <- diag(H)+amu; udiag <- diag(H)
+ c2 <- choldc(H,0,tol) #Cholesky decomposition of modified Hessian
+ H <- c2$H; addmax <- c2$addmax
+ }
+ #...unscale Hessian matrix here...
+ return(list(H=H,udiag=udiag))
+ }
```

```r
> choldc <- function(H,diagmx,tol){
+ addmax <- 0; aminl <- sqrt(diagmx*tol); amnlsq <- aminl^2; n <- nrow(H)
+ for(i in 1:n){ #for each row i...
+ sum <- 0
+ if(i>1){ for(k in 1:(i-1)){ sum <- sum+H[i,k]^2 } }
+ tmp1 <- H[i,i]-sum
+ if(tmp1>amnlsq){ #normal Cholesky
+ H[i,i] <- sqrt(tmp1)
+ }else{ #augment diagonal of H
+ }
```
In function \texttt{choldc()}, the elements of the Hessian matrix \( H \) are overwritten row-wise by the elements of the Cholesky factor \( L \). Calculating \( L \) row by row is possible (Cholesky-Banachiewicz algorithm), however, in that case, the rows have to be filled left to right, i.e., the off-diagonal elements \( l_{ij}, i > j \), of \( L \) have to be calculated before the diagonal element \( l_{ii} \).

In contrast, the implementation in \texttt{choldc()} calculates the diagonal element \( l_{ii} \) (here \texttt{tmp1}) first and hence uses the original elements \( h_{ik}, k < i \), which at that moment have not yet been updated to \( l_{ik} \). This leads to wrong and unnecessary modifications of the diagonal.

We illustrate the effect of this error by applying the functions \texttt{chlhsn()} and \texttt{choldc()} to the initial Hessian matrix in the Rosenbrock-example. Note that the Hessian matrix evaluated at the initial argument is symmetric, positive definite and has an acceptable condition number. So no modification is needed to obtain the Cholesky decomposition.

We first determine the Cholesky factor of \( H \) by the R-function \texttt{chol()}.

> \( H \leftarrow h(c(-1.2,1)); H; \text{eigen}(H)$values; kappa(H) \)

\[
\begin{array}{cc}
[,1] & [,2] \\
[1,] & 1330 & 480 \\
[2,] & 480 & 200 \\
[1] & 1506.36698 & 23.63302 \\
[1] & 75.81504 \\
\end{array}
\]

> \text{t(chol}(H))

\[
\begin{array}{cc}
[,1] & [,2] \\
[1,] & 36.46917 & 0.000000 \\
[2,] & 13.16180 & 5.173675 \\
\end{array}
\]

If we now apply the functions \texttt{chlhsn()} and \texttt{choldc()} the following results are obtained: The first checks of \( H \) in \texttt{chlhsn()} do not lead to modifications of \( H \), which is correct, so we can directly call function \texttt{choldc()}. 

6
The upper right element of the resulting matrix is the original element of $H$ that is not overwritten by the function call. However, while the first column of the \texttt{nlm()}-implemented Cholesky factor coincides with the first column of the \texttt{cho1()}-based Cholesky factor, the lower diagonal element clearly does not. Besides, \texttt{choldc()} attempts to add a large value of 230,680 to the diagonal of $H$, although $H$ is safely positive definite. (In fact, even in the second call to \texttt{choldc()} from \texttt{chlhsn()} we get a positive \texttt{addmax}, although $H$ then has already been modified.)

### 2.3 A Corrected Implementation of the Cholesky Decomposition

In Dennis and Schnabel (1996), slightly different algorithms for obtaining the modified Hessian (Algorithms A5.5.1 and A5.5.2 in their Appendix A) are presented, which we also implemented in \texttt{R} (with small modifications to the names for comparability). In \texttt{chlhsn()}() we first check for (almost) negative diagonal elements, large off-diagonal elements and zero-matrices, which all would lead to modifications of $H$ before the Cholesky decomposition routine \texttt{choldc()} is called. Function \texttt{choldc()} again attempts a normal Cholesky decomposition, but will augment the diagonal elements of $H$ if a diagonal element of $L$ falls below a given threshold or if an off-diagonal element exceeds another threshold, thereby bounding the condition number of the modified matrix $H + D$.

```r
> chlhsnDS <- function(H, epsm, sx){
+ n <- nrow(H); H <- H/(sx%*%t(sx)); tol <- sqrt(epsm)
+ diagmx <- max(diag(H)); diagmn <- min(diag(H)); posmax <- max(0, diagmx)
+ if(diagmn<tol*posmax) #...
+ H1 <- abs(H); diag(H1) <- NA; offmax <- max(0, max(H1, na.rm=TRUE))
+ if(diagmx<=(offmax*(1+2*tol))) #...
+ if(diagmx==0) #...
+ if(amu>0) diag(H) <- diag(H)+amu
+ udiag <- diag(H)
+ maxoffl <- sqrt(max(diagmx, offmax/n))
+ c1 <- choldcDS(H, maxoffl, tol)
+ H[lower.tri(H, diag=TRUE)] <- c1$H[lower.tri(H, diag=TRUE)]
+ addmax <- c1$addmax
+ if(addmax>0) #...as in chlhsn(), if addmax==0 return
+ return(list(H=H, udiag=udiag))
+ }
> #
> #
> choldcDS <- function(H,maxoffl,tol){
+ L <- matrix(0,ncol=ncol(H),nrow=nrow(H))
+ minl <- maxoffl*sqrt(tol)
+ if(maxoffl==0) maxoffl <- sqrt(max(abs(diag(H))))
+ minl2 <- maxoffl*tol; addmax <- 0; n <- nrow(H)
+ for(j in 1:n){ #for each column j...
+ sum <- 0
+ if(j>1){ for(i in 1:(j-1)){ sum <- sum+L[j,i]^2 } }
+ L[j,j] <- H[j,j]-sum; minljj <- 0
+ if(j<n){
+ for(i in (j+1):n){
+ sum2 <- 0
+ if(j>1) sum2 <- sum(L[i,1:(j-1)]*L[j,1:(j-1)])
+ L[i,j] <- H[j,i]-sum2; minljj <- max(abs(L[i,j]),minljj)
+ }
+ minljj <- max(minljj/maxoffl,minl)
+ if(L[j,j]>minljj^2){ #normal Cholesky
+ L[j,j] <- sqrt(L[j,j])
+ }else{ #augment diagonal
+ minljj <- max(minljj,minl2)
+ addmax <- max(addmax,minljj^2-L[j,j])
+ L[j,j] <- minljj
+ }
+ if(j<n){ for(i in (j+1):n){ L[i,j] <- L[i,j]/L[j,j] } }
+ }
+ H <- L; return(list(H=H,addmax=addmax))
+ }

We now apply this algorithm to the initial Hessian in the Rosenbrock-example. Note that the pre-checks in chlhsnDS() do not lead to any modification of H before the call to choldcDS().

> H1 <- abs(H); diag(H1) <- NA; offmax <- max(0,max(H1,na.rm=TRUE))
> maxoffl <- sqrt(max(diagmx,offmax/nrow(H)))
> choldcDS(H,maxoffl,tol); t(chol(H))

$H

[,1]      [,2]
[1,] 36.46917 0.000000
[2,] 13.16180 5.173675

$addmax

[1] 0

[,1]      [,2]
[1,] 36.46917 0.000000
[2,] 13.16180 5.173675
With this code the Cholesky factor $L$ of $H$ coincides with the one calculated with the R-function \texttt{chol()} and we do not add to the diagonal, suggesting that these functions yield the correct results. Taking a closer look at \texttt{cholcDS()}, we see that here the Cholesky factor is filled column-wise and thus, elements are addressed correctly.

We conclude that the implementation of the perturbed Cholesky decomposition, which is used to obtain the model Hessian in the \texttt{nlm()} function, is erroneous and should be replaced with the above algorithms by Dennis and Schnabel (1996).

In the next section, we present the minimization results of the Rosenbrock function and the Wood function, when the perturbed Cholesky decomposition is performed by the modified algorithms suggested above.

### 3 Examples Revisited

We want to find the minimum of the Rosenbrock function and the Wood function, respectively, using the Newton-type algorithm of \texttt{nlm()} with the different versions of the code for the perturbed Cholesky decomposition. Our R-function \texttt{optdrv()} implements the \texttt{nlm()}-algorithm, neglecting checks of input arguments. The boolean \texttt{perturb} indicates whether the perturbed Cholesky decomposition or a normal one (without check for positive definiteness) shall be used. The boolean \texttt{DS96} determines whether the perturbed Cholesky decomposition is based on the algorithms of Dennis and Schnabel (1996) or on the erroneous \texttt{nlm()}-code. The output for the Rosenbrock function follows.

```r
def f(x) = (x[1] - 1)^2 + 100 * (x[2] - x[1]^2)^2


def h(x) = diag(rep(1, 2))

> res1 <- optdrv(x=c(-1.2,1),f,gr,h,perturb=TRUE,DS96=FALSE)
> res2 <- optdrv(x=c(-1.2,1),f,gr,h,perturb=TRUE,DS96=TRUE)
> res3 <- optdrv(x=c(-1.2,1),f,gr,h,perturb=FALSE)

> res1
$minimum
[1] 2.829175

$estimate
[1] -0.6786981 0.4711891

$gradient
[1] -0.4911201 2.1115987

$code
[1] 4

$iterations
[1] 100

> res2
$minimum
[1] 2.829175

$estimate
[1] -0.6786981 0.4711891

$gradient
[1] -0.4911201 2.1115987

$code
[1] 4

$iterations
[1] 100

> res3
$minimum
[1] 2.829175

$estimate
[1] -0.6786981 0.4711891

$gradient
[1] -0.4911201 2.1115987

$code
[1] 4

$iterations
[1] 100
```

In the next section, we present the minimization results of the Rosenbrock function and the Wood function, when the perturbed Cholesky decomposition is performed by the modified algorithms suggested above.
The first result (res1) using the perturbed Cholesky decomposition as implemented in \texttt{nlm()} coincides with the result of a direct call to \texttt{nlm()} (\texttt{nlm.fgh}), hence our implementation is correct. The algorithm does not converge to the minimum in 100 iterations because of the unnecessary, large perturbations of the model Hessian. In contrast, however, the optimum (res2) obtained using a perturbed Cholesky decomposition based on DS96 is found after 24 iterations, when the algorithm terminates because the relative gradient is close to zero. Since in this example the Hessian matrix is always sufficiently positive definite, running the algorithm without any perturbation of the Hessian matrix (res3) yields the same result.

Repeating this comparison for the Wood function, replacing the \texttt{nlm}()-algorithms by the DS96-routine solves the problem of non-convergence.

\begin{verbatim}
> reswf1 <- optdrv(x=c(-3,-1,-3,-1),woodfunc,wfg,wfh,perturb=TRUE,DS96=FALSE)
> reswf2 <- optdrv(x=c(-3,-1,-3,-1),woodfunc,wfg,wfh,perturb=TRUE,DS96=TRUE)
\end{verbatim}
4 Summary and Conclusion

We demonstrate a bug in function \texttt{nlm()}, which occurs when an analytic Hessian is specified by the user. In that case, the algorithm often does not converge in an acceptable number of iterations, although it does so if only an analytic gradient is provided and the Hessian is approximated numerically. We identified an erroneous implementation of the perturbed Cholesky decomposition in function \texttt{choldc()} in \texttt{uncmin.c} as the source of this problem. The problem can be solved by replacing the erroneous part by an implementation of algorithms proposed in Dennis and Schnabel (1996).

References


11