Package ‘NAGFWrappers’

November 16, 2011

Version 23.0
Date 2011-11-20
Title Interfaces to routines from the NAG Fortran library
Author NAG
Maintainer NAG <support@nag.co.uk>
Depends R (>= 2.13.2)
SystemRequirements NAG Fortran library, Mark 23
Description Provides interfaces to a subset of routines from the NAG Fortran library
License Artistic-2.0
URL http://www.nag.co.uk
LazyLoad yes
LazyData yes

R topics documented:

a00ad .......................................................... 2
e04ab .......................................................... 5
e04bb .......................................................... 7
e04cb .......................................................... 9
e04dg ........................................................ 11
e04fc ........................................................ 13
e04fy ........................................................ 17
e04gd ........................................................ 18
e04gy ....................................................... 22
e04gz ....................................................... 24
e04hc ....................................................... 26
e04hd ....................................................... 28
e04he ....................................................... 30
e04hy ....................................................... 34
e04jc ....................................................... 37
e04jy ....................................................... 40
e04kd ....................................................... 42
a00ad: Library identification, details of implementation, major and minor marks

Description

a00ad prints information about the version of the NAG Library in use.

Usage

a00ad ()
Details

R interface to the NAG Fortran routine A00ADF.

Value

- **IMPL**  
  string  
  The implementation title which usually lists the target platform, operating system and compiler.

- **PREC**  
  string  
  The working or basic precision of the implementation. Some functions may perform operations in reduced precision or additional precision, but the great majority will perform all operations in basic precision. See the introduction to the Fortran library for definitions of these precisions.

- **PCODE**  
  string  
  The product code for the NAG Library implementation that is being used. The code has a discernible structure, but it is not necessary to know the details of this structure. The product code can be used to differentiate between individual product licence codes.

- **MKMAJ**  
  integer  
  The major mark of the NAG Library implementation that is being used.

- **MKMIN**  
  integer  
  The minor mark of the NAG Library implementation that is being used.

- **HDWARE**  
  string  
  The target hardware for the NAG Library implementation that is being used.

- **OPSYS**  
  string  
  The target operating system for the NAG Library implementation that is being used.

- **FCOMP**  
  string  
  The compiler used to build the NAG Library implementation that is being used.

- **VEND**  
  string  
  The subsidiary library, if any, that must be linked with the NAG Library implementation that is being used. If the implementation does not require a subsidiary library then the string '(self-contained)' will be returned in vend.

- **LICVAL**  
  boolean  
  Specifies whether or not a valid licence has been found for the NAG Library implementation that is being used.

Author(s)

NAG

References

http://www.nag.co.uk/numeric/FL/nagdoc_fl23/pdf/A00/a00adf.pdf
Examples

ans<-a00ad()
if (1){
  writeLines(toString(cat(sprintf(" *** Start of NAG Toolbox for MATLAB implementation details ***

","n")))
}
impl<-ans$IMPL
writeLines(sprintf(" Implementation title: %s\n",impl,"\n"))
prec<-ans$PREC
writeLines(toString(cat(sprintf(" Precision: %s\n",prec,"\n"))))
pcode<-ans$PCODE
writeLines(toString(cat(sprintf(" Product Code: %s\n",pcode,"\n"))))
mkmaj<-ans$MKMAJ
mkmin<-ans$MKMIN
writeLines(toString(cat(sprintf(" Mark: %d.%d\n",mkmaj,mkmin,"\n"))))
vend<-ans$VEND
if (match(vend,"(self-contained)")==1){
  writeLines(toString(cat(sprintf(" Vendor Library: None\n","n"))))
} else {
  writeLines(toString(cat(sprintf(" Vendor Library: %s\n",vend,"\n"))))
}
writeLines(toString(cat(sprintf(" Applicable to:\n","n"))))
hdware<-ans$HDWARE
writeLines(toString(cat(sprintf(" hardware - %s\n",hdware,"\n"))))
opsys<-ans$OPSYS
writeLines(toString(cat(sprintf(" op. sys. - %s\n",opsys,"\n"))))
fcomp<-ans$FCOMP
writeLines(toString(cat(sprintf(" compiler - %s\n",fcomp,"\n"))))


e04ab: Minimum, function of one variable using function values only

Description

e04ab searches for a minimum, in a given finite interval, of a continuous function of a single variable, using function values only. The method (based on quadratic interpolation) is intended for functions which have a continuous first derivative (although it will usually work if the derivative has occasional discontinuities).

Usage

e04ab(funct, e1, e2, a, b, maxcal)

Arguments

funct function
You must supply this function to calculate the value of the function $F(x)$ at any point $x$ in $[a,b]$. It should be tested separately before being used in conjunction with e04ab.

$F(xc) = funct(xc)$

e1 double
The relative accuracy to which the position of a minimum is required. (Note that, since e1 is a relative tolerance, the scaling of $x$ is automatically taken into account.)

e2 double
The absolute accuracy to which the position of a minimum is required. e2 should be no smaller than $2\varepsilon$. 


The lower bound \( a \) of the interval containing a minimum.

The upper bound \( b \) of the interval containing a minimum.

The maximum number of calls of \( F(x) \) to be allowed.

Details

R interface to the NAG Fortran routine E04ABF.

Value

\( E1 \) double
If you set \( e1 \) to 0.0 (or to any value less than \( \epsilon \)), \( e1 \) will be reset to the default value \( \sqrt{\epsilon} \) before starting the minimization process.

\( E2 \) double
If you set \( e2 \) to 0.0 (or to any value less than \( \epsilon \)), \( e2 \) will be reset to the default value \( \sqrt{\epsilon} \).

\( A \) double
An improved lower bound on the position of the minimum.

\( B \) double
An improved upper bound on the position of the minimum.

\( MAXCAL \) integer
The total number of times that funct was actually called.

\( X \) double
The estimated position of the minimum.

\( F \) double
The function value at the final point given in x.

\( IFAIL \) integer
\( ifail = 0 \) unless the function detects an error or a warning has been flagged (see the Errors section in Fortran library documentation).

Author(s)

NAG

References

http://www.nag.co.uk/numeric/FL/nagdoc_fl23/pdf/E04/e04abf.pdf

Examples

```r
ifail <- 0
funct = function(xc) {
   fc <- sin(xc)/xc
   list(FC = fc)
}
```
e04bb

```r
e1 <- 0
e2 <- 0
a <- 3.5
b <- 5
maxcal <- 30
e04ab(funct, e1, e2, a, b, maxcal)
```

e04bb: Minimum, function of one variable, using first derivative

Description
e04bb searches for a minimum, in a given finite interval, of a continuous function of a single vari-
able, using function and first derivative values. The method (based on cubic interpolation) is in-
tended for functions which have a continuous first derivative (although it will usually work if the
derivative has occasional discontinuities).

Usage
e04bb(funct, e1, e2, a, b, maxcal)

Arguments

- **funct** function
  You must supply this function to calculate the values of $F(x)$ and $\frac{dF}{dx}$ at any
  point $x$ in $[ab]$.
  $(FC, GC) = \text{funct}(xc)$

- **e1** double
  The relative accuracy to which the position of a minimum is required. (Note that, since $e1$ is a relative tolerance, the scaling of $x$ is automatically taken into
  account.)

- **e2** double
  The absolute accuracy to which the position of a minimum is required. $e2$ should
  be no smaller than $2\epsilon$.

- **a** double
  The lower bound $a$ of the interval containing a minimum.

- **b** double
  The upper bound $b$ of the interval containing a minimum.

- **maxcal** integer
  The maximum number of calls of funct to be allowed.

Details

R interface to the NAG Fortran routine E04BBF.
Value

E1  double
If you set e1 to 0.0 (or to any value less than \( \epsilon \)), e1 will be reset to the default value \( \sqrt{\epsilon} \) before starting the minimization process.

E2  double
If you set e2 to 0.0 (or to any value less than \( \epsilon \)), e2 will be reset to the default value \( \sqrt{\epsilon} \).

A  double
An improved lower bound on the position of the minimum.

B  double
An improved upper bound on the position of the minimum.

MAXCAL  integer
The total number of times that funct was actually called.

X  double
The estimated position of the minimum.

F  double
The function value at the final point given in x.

G  double
The value of the first derivative at the final point in x.

IFAIL  integer
ifail = 0 unless the function detects an error or a warning has been flagged (see the Errors section in Fortran library documentation).

Author(s)
NAG

References
http://www.nag.co.uk/numeric/FL/nagdoc_fl23/pdf/E04/e04bbf.pdf

Examples

```r
ifail <- 0
defunct = function(xc) {
  fc <- sin(xc)/xc
  gc <- (cos(xc) - fc)/xc
  list(FC = fc, GC = gc)
}
e1 <- 0
e2 <- 0
a <- 3.5
b <- 5
```
maxcal <- 30
e04bb(funct, e1, e2, a, b, maxcal)

---

**Description**

*e04cb* minimizes a general function \( F(x) \) of \( n \) independent variables \( x = (x_1x_2\ldots x_n)^T \) by the Nelder and Mead simplex method (see [Nelder J A Mead R (1965)]). Derivatives of the function need not be supplied.

**Usage**

```r
e04cb(x, tolf, tolx, funct, monit, maxcal,
    n = nrow(x))
```

**Arguments**

- **x** double array
  A guess at the position of the minimum. Note that the problem should be scaled so that the values of the \( x[i] \) are of order unity.

- **tolf** double
  The error tolerable in the function values, in the following sense. If \( f_i \) for \( i = 1\ldots n + 1 \), are the individual function values at the vertices of the current simplex, and if \( f_m \) is the mean of these values, then you can request that *e04cb* should terminate if

  \[
  \sqrt{\frac{1}{n+1} \sum_{i=1}^{n+1} (f_i - f_m)^2} < tolf.
  \]

- **tolx** double
  The error tolerable in the spatial values, in the following sense. If \( LV \) denotes the ‘linearized’ volume of the current simplex, and if \( LV_{init} \) denotes the ‘linearized’ volume of the initial simplex, then you can request that *e04cb* should terminate if

  \[
  \frac{LV}{LV_{init}} < tolx.
  \]

- **funct** function
  funct must evaluate the function \( F \) at a specified point. It should be tested separately before being used in conjunction with *e04cb*.

  \( (FC) = funct(n, xc) \)

- **monit** function
  monit may be used to monitor the optimization process. It is invoked once every iteration.

  \( () = monit(fmin, fmax, sim, n, ncall, serror, vratio) \)
maxcal integer
The maximum number of function evaluations to be allowed.

n integer: default = nrow(x)
n, the number of variables.

Details
R interface to the NAG Fortran routine E04CBF.

Value
X double array
The value of x corresponding to the function value in f.
F double
The lowest function value found.
IFAIL integer
ifail = 0 unless the function detects an error or a warning has been flagged (see the Errors section in Fortran library documentation).

Author(s)
NAG

References
http://www.nag.co.uk/numeric/FL/nagdoc_fl23/pdf/E04/e04cbf.pdf

Examples

ifail <- 0
funct = function(n, xc) {
   list(FC = fc)
}
monit = function(fmin, fmax, sim, n, ncall, serror, vratio) {

   if (user(1) != 0) {
      writeLines(toString(cat(sprintf("\nThere have been %d function calls\n", ncall, "\n"))))
      writeLines(toString(cat(sprintf("The smallest function value is %10.4f\n", fmin, "\n"))))
      writeLines(toString(cat(sprintf("The simplex is\n", "\n"))))
   }
writeLines(toString(cat(sprintf(sim, "\n")))))

writeLines(toString(cat(sprintf("The standard deviation in function values at the vertices of the simplex is %10.4f
", serror, "\n")))))

writeLines(toString(cat(sprintf("The linearized volume ratio of the current simplex to the starting one is %10.4f
", vratio, "\n")))))

}  
list()
}

x <- matrix(c(-1, 1), nrow = 2, ncol = 1, byrow = TRUE)

tolf <- sqrt(x02aj()$"result")
tolx <- sqrt(tolf)
maxcal <- 100
user <- function(switch_integer) {
  switch(switch_integer, 0)
}
e04cb(x, tolf, tolx, funct, monit, maxcal)

e04dg

**e04dg**: Unconstrained minimum, preconditioned conjugate gradient algorithm, function of several variables using first derivatives (comprehensive)

**Description**

*e04dg* minimizes an unconstrained nonlinear function of several variables using a pre-conditioned, limited memory quasi-Newton conjugate gradient method. First derivatives (or an ‘acceptable’ finite difference approximation to them) are required. It is intended for use on large scale problems.

**Usage**

e04dg(objfun, x, optlist,
      n = nrow(x))

**Arguments**

- **objfun**: function
  
  objfun must calculate the objective function \( F(x) \) and possibly its gradient as well for a specified \( n \) element vector \( x \).

(MODE,OBJF,OBJGRD) = objfun(mode,n,x,nstate)
x double array
An initial estimate of the solution.

optlist options list
Optional parameters may be listed, as shown in the following table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Estimated Optimal Function Value</td>
<td>double</td>
<td>Estimated Optimal Function Value = ( \epsilon^{0.9} )</td>
</tr>
<tr>
<td>Function Precision</td>
<td>double</td>
<td>Default = ( \epsilon^{0.9} )</td>
</tr>
<tr>
<td>Iteration Limit</td>
<td>integer</td>
<td>Default = max(50, 5n)</td>
</tr>
<tr>
<td>Iters</td>
<td>integer</td>
<td>Default = max(50, 5n)</td>
</tr>
<tr>
<td>Linesearch Tolerance</td>
<td>double</td>
<td>Default for e04dg = list</td>
</tr>
<tr>
<td>List</td>
<td>integer</td>
<td>Default for e04dg = nolist</td>
</tr>
<tr>
<td>Nolist</td>
<td>integer</td>
<td>Default = 0</td>
</tr>
<tr>
<td>Maximum Step Length</td>
<td>double</td>
<td>Default = ( 10^{20} )</td>
</tr>
<tr>
<td>Optimality Tolerance</td>
<td>double</td>
<td>Default = ( \epsilon^{0.8} )</td>
</tr>
<tr>
<td>Print Level</td>
<td>integer</td>
<td>Default = 0</td>
</tr>
<tr>
<td>Start Objective Check at Variable</td>
<td>integer</td>
<td>Default = 1</td>
</tr>
<tr>
<td>Stop Objective Check at Variable</td>
<td>integer</td>
<td>Default = n</td>
</tr>
<tr>
<td>Verify Level</td>
<td>integer</td>
<td>Default = 0</td>
</tr>
<tr>
<td>Verify</td>
<td>integer</td>
<td>Default = 0</td>
</tr>
<tr>
<td>Verify Gradients</td>
<td>integer</td>
<td>Default = 0</td>
</tr>
<tr>
<td>Verify Objective Gradients</td>
<td>integer</td>
<td>Default = 0</td>
</tr>
</tbody>
</table>

\( n \) integer: default = nrow(x)
\( n \), the number of variables.

Details

R interface to the NAG Fortran routine E04DGF.

Value

ITER integer
The total number of iterations performed.

OBJF double
The value of the objective function at the final iterate.

OBJGRD double array
The gradient of the objective function at the final iterate (or its finite difference approximation).

X double array
The final estimate of the solution.

IFAIL integer
ifail = 0 unless the function detects an error or a warning has been flagged (see the Errors section in Fortran library documentation).

Author(s)

NAG
e04fc

References

http://www.nag.co.uk/numeric/FL/nagdoc_fl23/pdf/E04/e04dgf.pdf

Examples

```r
optlist <- list()
ifail <- 0
objfun = function(mode, n, x, nstate) {
  objgrd <- as.matrix(mat.or.vec(2, 1))
expx1 <- exp(x[1])
  if (mode == 2) {
  } else {
    objgrd <- as.matrix(mat.or.vec(2, 1))
  }
  list(MODE = as.integer(mode), OBJF = objf, OBJGRD = as.matrix(objgrd))
}
x <- matrix(c(-1, 1), nrow = 2, ncol = 1, byrow = TRUE)
e04dg(objfun, x, optlist)
```

---

e04fc: Unconstrained minimum of a sum of squares, combined Gauss-Newton and modified Newton algorithm using function values only (comprehensive)

Description

e04fc is a comprehensive algorithm for finding an unconstrained minimum of a sum of squares of m nonlinear functions in n variables (m ≥ n). No derivatives are required.

The function is intended for functions which have continuous first and second derivatives (although it will usually work even if the derivatives have occasional discontinuities).
Usage

e04fc(m, lsqfun, lsqmon, maxcal, x,
    n = nrow(x),
    iprint = 1,
    eta = if (n==1) 0.0 else 0.5,
    xtol = 0.0,
    stepmx = 100000.0)

Arguments

m  integer
    lsqfun function
    lsqfun must calculate the vector of values \( f_i(x) \) at any point \( x \). (However, if
    you do not wish to calculate the residuals at a particular \( x \), there is the option of
    setting a argument to cause e04fc to terminate immediately.)
    (IFLAG,FVEC) = lsqfun(iflag,m,n,xc)

lsqmon function
    If \( iprint \geq 0 \), you must supply lsqmon which is suitable for monitoring the
    minimization process. lsqmon must not change the values of any of its argu-
    ments.
    () = lsqmon(m,n,xc,fvec,fjac,ldfjac,s,igrade,niter,nf)

maxcal integer
    The limit you set on the number of times that lsqfun may be called by e04fc.
    There will be an error exit (see the Errors section in Fortran library documenta-
    tion) after maxcal calls of lsqfun.

x  double array
    \( x[j] \) must be set to a guess at the \( j \)th component of the position of the minimum
    for \( j = 1 \ldots n \).

n  integer: default = nrow(x)
    The number \( m \) of residuals, \( f_i(x) \), and the number \( n \) of variables, \( x_j \).

iprint integer: default = 1
    The frequency with which lsqmon is to be called.

eta  double: default = if (n==1) 0.0 else 0.5
    Specifies how accurately the linear minimizations are to be performed. The min-
    imum with respect to \( \alpha^{(k)} \) will be located more accurately for small values of
    eta (say, 0.01) than for large values (say, 0.9). Although accurate linear min-
    imizations will generally reduce the number of iterations performed by e04fc,
    they will increase the number of calls of lsqfun made each iteration. On balance
    it is usually more efficient to perform a low accuracy minimization.

xtol  double: default = 0.0
    The accuracy in \( x \) to which the solution is required.

stepmx  double: default = 100000.0
    An estimate of the Euclidean distance between the solution and the starting point
    supplied by you. (For maximum efficiency, a slight overestimate is preferable.)
    e04fc will ensure that, for each iteration,
    \[
    \sum_{j=1}^{n} \left( x_j^{(k)} - x_j^{(k-1)} \right)^2 \leq \left( \text{stepmx} \right)^2,
    \]
where \( k \) is the iteration number. Thus, if the problem has more than one solution, e04fc is most likely to find the one nearest to the starting point. On difficult problems, a realistic choice can prevent the sequence \( x^{(k)} \) entering a region where the problem is ill-behaved and can help avoid overflow in the evaluation of \( F(x) \). However, an underestimate of stepmx can lead to inefficiency.

**Details**

R interface to the NAG Fortran routine E04FCF.

**Value**

- **X**
  - double array
  - The final point \( x^{(k)} \). Thus, if ifail = 0 on exit, \( x[j] \) is the \( j \)th component of the estimated position of the minimum.

- **FSUMSQ**
  - double
  - The value of \( F(x) \), the sum of squares of the residuals \( f_i(x) \), at the final point given in \( x \).

- **FVEC**
  - double array
  - The value of the residual \( f_i(x) \) at the final point given in \( x \) for \( i = 1 \ldots m \).

- **FJAC**
  - double array
  - The estimate of the first derivative \( \frac{\partial f_i}{\partial x_j} \) at the final point given in \( x \) for \( j = 1 \ldots n \) for \( i = 1 \ldots m \).

- **S**
  - double array
  - The singular values of the estimated Jacobian matrix at the final point. Thus \( s \) may be useful as information about the structure of your problem.

- **V**
  - double array
  - The matrix \( V \) associated with the singular value decomposition

\[
J = USV^T
\]

of the estimated Jacobian matrix at the final point, stored by columns. This matrix may be useful for statistical purposes, since it is the matrix of orthonormalized eigenvectors of \( J^TJ \).

- **NITER**
  - integer
  - The number of iterations which have been performed in e04fc.

- **NF**
  - integer
  - The number of times that the residuals have been evaluated (i.e., number of calls of lsqfun).

- **IFAIL**
  - integer
  - ifail = 0 unless the function detects an error or a warning has been flagged (see the Errors section in Fortran library documentation).

**Author(s)**

NAG

**References**

http://www.nag.co.uk/numeric/FL/nagdoc_fl23/pdf/E04/e04fcf.pdf
Examples

```r
ifail <- 0
lsqfun = function(iflag, m, n, xc) {
    fvec <- as.matrix(mat.or.vec(m, 1))
    for (i in c(1:m)) {
             t[i, 3]) - y[i]
    }
    list(IFLAG = as.integer(iflag), FVEC = as.matrix(fvec))
}
lsqmon = function(m, n, xc, fvec, fjacc, ljc, s, igrade,
                   niter, nf) {
    if (niter == 0) {
        writeLines(toString(cat(sprintf(" Itn F evals SUMSQ \n",
                                         "\n"))))
    }
    fsumsq <- crossprod(fvec, fvec)
    writeLines(toString(cat(sprintf(" %3d %3d %12.8f
",
                                    niter, nf, fsumsq, "\n"))))
    list()
}

m <- 15
n <- 3
maxcal <- 1200
x <- matrix(c(0.5, 1, 1.5), nrow = 3, ncol = 1, byrow = TRUE)
iw <- as.matrix(mat.or.vec(1, 1))
w <- as.matrix(mat.or.vec(6 %*% n + m %*% n + 2 %*%
                         m + n %*% ((n - 1)/2), 1))
y <- matrix(c(0.14, 0.18, 0.22, 0.25, 0.29, 0.32,
             0.35, 0.39, 0.37, 0.58, 0.73, 0.96, 1.34, 2.1, 4.39), nrow = 1,
            ncol = 15, byrow = TRUE)

t <- matrix(c(1, 15, 1, 2, 14, 2, 3, 13, 3, 4, 12,
              4, 5, 11, 5, 6, 10, 6, 7, 9, 7, 8, 8, 8, 9, 7, 7, 10, 6,
              6, 11, 5, 5, 12, 4, 4, 13, 3, 3, 14, 2, 2, 15, 1, 1), nrow = 15,
             ncol = 3, byrow = TRUE)
```

Description

e04fy is an easy-to-use algorithm for finding an unconstrained minimum of a sum of squares of \( m \) nonlinear functions in \( n \) variables \((m \geq n)\). No derivatives are required. It is intended for functions which are continuous and which have continuous first and second derivatives (although it will usually work even if the derivatives have occasional discontinuities).

Usage

e04fy(m, lsfun1, x, 
\text{n = nrow(x)})

Arguments

\begin{itemize}
  \item \text{m} \quad \text{integer}
  \item \text{lsfun1} \quad \text{function}
  \begin{itemize}
    \item You must supply this function to calculate the vector of values \( f_i(x) \) at any point \( x \). It should be tested separately before being used in conjunction with e04fy (see the E04 chapter introduction in the Fortran Library documentation).
  \end{itemize}
  \item \text{x} \quad \text{double array}
  \begin{itemize}
    \item \( x[j] \) must be set to a guess at the \( j \)th component of the position of the minimum for \( j = 1 \ldots n \).
  \end{itemize}
  \item \text{n} \quad \text{integer: default = nrow(x)}
  \begin{itemize}
    \item The number \( m \) of residuals, \( f_i(x) \), and the number \( n \) of variables, \( x_j \).
  \end{itemize}
\end{itemize}

Details

R interface to the NAG Fortran routine E04FYF.

Value

\begin{itemize}
  \item \text{X} \quad \text{double array}
  \begin{itemize}
    \item The lowest point found during the calculations. Thus, if ifail = 0 on exit, \( x[j] \) is the \( j \)th component of the position of the minimum.
  \end{itemize}
  \item \text{FSUMSQ} \quad \text{double}
  \begin{itemize}
    \item The value of the sum of squares, \( F(x) \), corresponding to the final point stored in \( x \).
  \end{itemize}
  \item \text{IFAIL} \quad \text{integer}
  \begin{itemize}
    \item ifail = 0 unless the function detects an error or a warning has been flagged (see the Errors section in Fortran library documentation).
  \end{itemize}
\end{itemize}
ifail <- 0
lssf11 = function(m, n, xc) {
  fvec <- as.matrix(mat.or.vec(m, 1))
  for (i in c(1:m)) {
  }
  list(FVEC = as.matrix(fvec))
}
m <- 15
x <- matrix(c(0.5, 1, 1.5), nrow = 3, ncol = 1, byrow = TRUE)
y <- matrix(c(0.14, 0.18, 0.22, 0.25, 0.29, 0.32,
              0.35, 0.39, 0.37, 0.58, 0.73, 0.96, 1.34, 2.1, 4.39), nrow = 1,
              ncol = 15, byrow = TRUE)
t <- matrix(c(1, 15, 1, 2, 14, 2, 3, 13, 3, 4, 12,
              4, 5, 11, 5, 6, 10, 6, 7, 9, 7, 8, 8, 9, 7, 7, 10, 6,
              6, 11, 5, 5, 12, 4, 4, 13, 3, 3, 14, 2, 2, 15, 1, 1), nrow = 15,
              ncol = 3, byrow = TRUE)
user <- function(switch_integer) {
  switch(switch_integer, y, t, 3)
}
e04fy(m, lsfun1, x)
**Description**

e04gd is a comprehensive modified Gauss-Newton algorithm for finding an unconstrained minimum of a sum of squares of $m$ nonlinear functions in $n$ variables ($m \geq n$). First derivatives are required. The function is intended for functions which have continuous first and second derivatives (although it will usually work even if the derivatives have occasional discontinuities).

**Usage**

e04gd(m, lsqfun, lsqmon, maxcal, xtol, x,
     n = nrow(x),
     iprint = 1,
     eta = if (n==1) 0.0 else 0.5,
     stepmx = 100000.0)

**Arguments**

m  integer
lsqfun  function
lsqfun must calculate the vector of values $f_i(x)$ and Jacobian matrix of first derivatives $\frac{\partial f_i}{\partial x_j}$ at any point $x$. (However, if you do not wish to calculate the residuals or first derivatives at a particular $x$, there is the option of setting a argument to cause e04gd to terminate immediately.)

(IFLAG,FVEC,FJAC) = lsqfun(iflag,m,n,xc,ldfjac)

lsqmon  function
If $iprint \geq 0$, you must supply lsqmon which is suitable for monitoring the minimization process. Lsqmon must not change the values of any of its arguments.

() = lsqmon(m,n,xc,fvec,fjac,ldfjac,s,igrade,niter,nf)

maxcal  integer
Enables you to limit the number of times that lsqfun is called by e04gd. There will be an error exit (see the Errors section in Fortran library documentation) after maxcal evaluations of the residuals (i.e., calls of lsqfun with iflag set to 2). It should be borne in mind that, in addition to the calls of lsqfun which are limited directly by maxcal, there will be calls of lsqfun (with iflag set to 1) to evaluate only first derivatives.

xtol  double
The accuracy in $x$ to which the solution is required.

x  double array
$x[j]$ must be set to a guess at the $j$th component of the position of the minimum for $j = 1 \ldots n$.

n  integer: default = nrow(x)
The number $m$ of residuals, $f_i(x)$, and the number $n$ of variables, $x_j$.

iprint  integer: default = 1
The frequency with which lsqmon is to be called.

-iprint > 0: lsqmon is called once every iprint iterations and just before exit from e04gd.
-iprint = 0: lsqmon is just called at the final point.
-iprint < 0: lsqmon is not called at all.
Every iteration of e04gd involves a linear minimization, i.e., minimization of
\( F(\mathbf{x}^{(k)} + \alpha^{(k)} p^{(k)}) \) with respect to \( \alpha^{(k)} \). eta specifies how accurately these
linear minimizations are to be performed. The minimum with respect to \( \alpha^{(k)} \)
will be located more accurately for small values of eta (say, 0.01) than for large
values (say, 0.9).

An estimate of the Euclidean distance between the solution and the starting point
supplied by you. (For maximum efficiency, a slight overestimate is preferable.)
e04gd will ensure that, for each iteration,
\[
\sum_{j=1}^{n} \left( x_j^{(k)} - x_j^{(k-1)} \right)^2 \leq (\text{stepmx})^2
\]
where \( k \) is the iteration number. Thus, if the problem has more than one solution,
e04gd is most likely to find the one nearest to the starting point. On difficult
problems, a realistic choice can prevent the sequence of \( x^{(k)} \) entering a region
where the problem is ill-behaved and can help avoid overflow in the evaluation
of \( F(\mathbf{x}) \). However, an underestimate of stepmx can lead to inefficiency.

Details

R interface to the NAG Fortran routine E04GDF.

Value

X
double array
The final point \( x^{(k)} \). Thus, if ifail = 0 on exit, \( x[j] \) is the \( j \)th component of the
estimated position of the minimum.

FSUMSQ
double
The value of \( F(\mathbf{x}) \), the sum of squares of the residuals \( f_i(\mathbf{x}) \), at the final point
given in \( \mathbf{x} \).

FVEC
double array
The value of the residual \( f_i(\mathbf{x}) \) at the final point given in \( \mathbf{x} \) for \( i = 1 \ldots m \).

FJAC
double array
The value of the first derivative \( \frac{\partial f_i}{\partial x_j} \) evaluated at the final point given in \( \mathbf{x} \) for
\( j = 1 \ldots n \) for \( i = 1 \ldots m \).

S
double array
The singular values of the Jacobian matrix at the final point. Thus \( s \) may be
useful as information about the structure of your problem.

V
double array
The matrix \( V \) associated with the singular value decomposition
\[ J = USV^T \]
of the Jacobian matrix at the final point, stored by columns. This matrix may be
useful for statistical purposes, since it is the matrix of orthonormalized eigen-
ectors of \( J^TJ \).

NITER
integer
The number of iterations which have been performed in e04gd.
The number of times that the residuals have been evaluated (i.e., number of calls of lsqfun with iflag set to 2).

ifail = 0 unless the function detects an error or a warning has been flagged (see the Errors section in Fortran library documentation).

Author(s)
NAG

References
http://www.nag.co.uk/numeric/FL/nagdoc_fl23/pdf/E04/e04gdf.pdf

Examples

```r
ifail <- 0
lsqfun = function(iflag, m, n, xc, ljc) {
  fvec <- as.matrix(mat.or.vec(m, 1))
  fjacc <- as.matrix(mat.or.vec(ljc, n))
  for (i in c(1:m)) {
    if (iflag != 1) {
      fvec[i] <- xc[1] + t[i, 1]/denom - y[i]
    }
    if (iflag != 0) {
      fjacc[i, 1] <- 1
      dummy <- -1/(denom %*% denom)
      fjacc[i, 2] <- t[i, 1] %*% t[i, 2] %*% dummy
      fjacc[i, 3] <- t[i, 1] %*% t[i, 3] %*% dummy
    }
  }
  list(IFLAG = as.integer(iflag), FVEC = as.matrix(fvec), FJAC = as.matrix(fjacc))
}
lsqmon = function(m, n, xc, fvec, fjacc, ljc, s, igrade, niter, nf) {
  list()
}
m <- 15
maxcal <- 150
xtol <- 1.05418557512311e-07
```
\[ x <- \text{matrix}(c(0.5, 1, 1.5), \text{nrow} = 3, \text{ncol} = 1, \text{byrow} = \text{TRUE}) \]

\[ \text{iw} <- \text{matrix}(c(0), \text{nrow} = 1, \text{ncol} = 1, \text{byrow} = \text{TRUE}) \]

\[ w <- \text{as.matrix(mat.or.vec(105, 1))} \]

\[ y <- \text{matrix}(c(0.14, 0.18, 0.22, 0.25, 0.29, 0.32,
                   0.35, 0.39, 0.37, 0.58, 0.73, 0.96, 1.34, 2.1, 4.39), \text{nrow} = 1,
                   \text{ncol} = 15, \text{byrow} = \text{TRUE}) \]

\[ t <- \text{matrix}(c(1, 15, 1, 2, 14, 2, 3, 13, 3, 4, 12,
                   4, 5, 11, 5, 6, 10, 6, 7, 9, 7, 8, 8, 9, 7, 10, 6,
                   6, 11, 5, 12, 4, 4, 13, 3, 3, 14, 2, 2, 15, 1, 1), \text{nrow} = 15,
                   \text{ncol} = 3, \text{byrow} = \text{TRUE}) \]

e04gd(m, lsqfun, lsqmon, maxcal, xtol, x)

e04gy

\[ e04gy: \text{Unconstrained minimum of a sum of squares, combined Gauss-}
\text{Newton and quasi-Newton algorithm, using first derivatives (easy-to-}
\text{use)} \]

**Description**

e04gy is an easy-to-use quasi-Newton algorithm for finding an unconstrained minimum of a sum of squares of \( m \) nonlinear functions in \( n \) variables (\( m \geq n \)). First derivatives are required.

It is intended for functions which are continuous and which have continuous first and second derivatives (although it will usually work even if the derivatives have occasional discontinuities).

**Usage**

e04gy(m, lsfun2, x,
      n = nrow(x))

**Arguments**

- **m** integer
- **lsfun2** function

You must supply this function to calculate the vector of values \( f_i(x) \) and the Jacobian matrix of first derivatives \( \frac{\partial f_i}{\partial x_j} \) at any point \( x \). It should be tested separately before being used in conjunction with e04gy (see the E04 chapter introduction in the Fortran Library documentation).

\( (\text{FVEC}, \text{FJAC}) = \text{lsfun2}(m,n,xc,ldfjac) \)
double array

\( x[j] \) must be set to a guess at the \( j \)th component of the position of the minimum for \( j = 1 \ldots n \). The function checks the first derivatives calculated by \texttt{lsfun2} at the starting point and so is more likely to detect an error in your function if the initial \( x[j] \) are nonzero and mutually distinct.

integer: \texttt{default = nrow(x)}

The number \( m \) of residuals, \( f_i(x) \), and the number \( n \) of variables, \( x_j \).

**Details**

R interface to the NAG Fortran routine \texttt{E04GYF}.

**Value**

\texttt{X}

double array

The lowest point found during the calculations. Thus, if \texttt{ifail = 0} on exit, \( x[j] \) is the \( j \)th component of the position of the minimum.

\texttt{FSUMSQ}

double

The value of the sum of squares, \( F(x) \), corresponding to the final point stored in \( x \).

\texttt{IFAIL}

integer

\( \texttt{ifail = 0} \) unless the function detects an error or a warning has been flagged (see the Errors section in Fortran library documentation).

**Author(s)**

NAG

**References**

[http://www.nag.co.uk/numeric/FL/nagdoc_fl23/pdf/E04/e04gyf.pdf](http://www.nag.co.uk/numeric/FL/nagdoc_fl23/pdf/E04/e04gyf.pdf)

**Examples**

```r
ifail <- 0
lsfun2 = function(m, n, xc, ljc) {
  fvec <- as.matrix(mat.or.vec(m, 1))
  fjacc <- as.matrix(mat.or.vec(ljc, n))
  for (i in c(1:m)) {
    fvec[i] <- xc[1] + user(2)[i, 1]/denom - user(1)[i]
    fjacc[i, 1] <- 1
dummy <- -1/(denom %*% denom)
    fjacc[i, 2] <- user(2)[i, 1] %*% user(2)[i, 2] %*% dummy
    fjacc[i, 3] <- user(2)[i, 1] %*% user(2)[i, 3] %*% dummy
  }
}
```
```r
list(FVEC = as.matrix(fvec), FJAC = as.matrix(fjacc))
}

m <- 15
x <- matrix(c(0.5, 1, 1.5), nrow = 3, ncol = 1, byrow = TRUE)

y <- matrix(c(0.14, 0.18, 0.22, 0.25, 0.29, 0.32,
              0.35, 0.39, 0.37, 0.58, 0.73, 0.96, 1.34, 2.1, 4.39), nrow = 1,
              ncol = 15, byrow = TRUE)

t <- matrix(c(1, 15, 1, 2, 14, 2, 3, 13, 3, 4, 12,
              4, 5, 11, 5, 6, 10, 6, 7, 9, 7, 8, 8, 9, 7, 10, 6,
              6, 11, 5, 5, 12, 4, 4, 13, 3, 3, 14, 2, 2, 15, 1, 1), nrow = 15,
              ncol = 3, byrow = TRUE)

user <- function(switch_integer) {
  switch(switch_integer, y, t, 3)
}

e04gy(m, lsfun2, x)
```

### Description

`e04gz` is an easy-to-use modified Gauss-Newton algorithm for finding an unconstrained minimum of a sum of squares of `m` nonlinear functions in `n` variables (`m ≥ n`). First derivatives are required. It is intended for functions which are continuous and which have continuous first and second derivatives (although it will usually work even if the derivatives have occasional discontinuities).

### Usage

```r
e04gz(m, lsfun2, x,
     n = nrow(x))
```

### Arguments

- `m` integer
- `lsfun2` function

You must supply this function to calculate the vector of values $f_i(x)$ and the Jacobian matrix of first derivatives $\frac{\partial f_i}{\partial x_j}$ at any point $x$. It should be tested separately before being used in conjunction with e04gz.

$(FVEC,FJAC) = lsfun2(m,n,xc,ldfjac)$
x double array

\[ x[j] \] must be set to a guess at the \( j \)th component of the position of the minimum for \( j = 1 \ldots n \). The function checks the first derivatives calculated by \text{lsfun2} \ at \ the \ starting \ point \ and \ so \ is \ more \ likely \ to \ detect \ any \ error \ in \ your \ functions \ if \ the \ initial \ \( x[j] \) are nonzero and mutually distinct.

\text{n} integer: \textbf{default} = \text{nrow}(x)

The number \( m \) of residuals, \( f_i(x) \), and the number \( n \) of variables, \( x_j \).

\textbf{Details}

R interface to the NAG Fortran routine E04GZF.

\textbf{Value}

\( X \) double array

The lowest point found during the calculations. Thus, if \text{ifail} = 0 on exit, \( x[j] \) is the \( j \)th component of the position of the minimum.

\( \text{FSUMSQ} \) double

The value of the sum of squares, \( F(x) \), corresponding to the final point stored in \( x \).

\( \text{IFAIL} \) integer

\( \text{ifail} = 0 \) unless the function detects an error or a warning has been flagged (see the Errors section in Fortran library documentation).

\textbf{Author(s)}

NAG

\textbf{References}

http://www.nag.co.uk/numeric/FL/nagdoc_fl23/pdf/E04/e04gzf.pdf

\textbf{Examples}

```r
ifail <- 0
lsfun2 = function(m, n, xc, ljc) {
    fvec <- as.matrix(mat.or.vec(m, 1))
    fjacc <- as.matrix(mat.or.vec(ljc, n))
    for (i in c(1:m)) {
        fvec[i] <- xc[1] + user(2)[i, 1]/denom - user(1)[i]
        fjacc[i, 1] <- 1
        dummy <- -1/(denom %*% denom)
        fjacc[i, 2] <- user(2)[i, 1] %*% user(2)[i, 2] %*% dummy
        fjacc[i, 3] <- user(2)[i, 1] %*% user(2)[i, 3] %*% dummy
    }
}
```
list(FVEC = as.matrix(fvec), FJAC = as.matrix(fjacc))
}

m <- 15
x <- matrix(c(0.5, 1, 1.5), nrow = 3, ncol = 1, byrow = TRUE)

y <- matrix(c(0.14, 0.18, 0.22, 0.25, 0.29, 0.32,
              0.35, 0.39, 0.37, 0.58, 0.73, 0.96, 1.34, 2.1, 4.39),
              nrow = 1, ncol = 15, byrow = TRUE)

t <- matrix(c(1, 15, 1, 2, 14, 2, 3, 13, 3, 4, 12,
              4, 5, 11, 5, 6, 10, 6, 7, 9, 7, 8, 8, 9, 7, 10, 6,
              6, 11, 5, 5, 12, 4, 4, 13, 3, 14, 2, 2, 15, 1, 1),
              nrow = 15, ncol = 3, byrow = TRUE)

user <- function(switch_integer) {
  switch(switch_integer, y, t, 3)
}

e04gz(m, lsfun2, x)

e04hc

**e04hc: Check user’s function for calculating first derivatives of function**

**Description**

e04hc checks that a function for evaluating an objective function and its first derivatives produces derivative values which are consistent with the function values calculated.

**Usage**
e04hc(funct, x,
      n = nrow(x))

**Arguments**  

- **funct**  
  funct must evaluate the function and its first derivatives at a given point. (The minimization functions mentioned in the Description in Fortran library documentation gives you the option of resetting arguments of funct to cause the minimization process to terminate immediately. e04hc will also terminate immediately, without finishing the checking process, if the argument in question is reset.)
  
  (IFLAG, FC, GC) = funct(iflag, n, xc)
double array

\textit{x} for \( j = 1 \ldots n \), must be set to the coordinates of a suitable point at which to check the derivatives calculated by \textit{funct}. ‘Obvious’ settings, such as \( 0.0 \) or \( 1.0 \), should not be used since, at such particular points, incorrect terms may take correct values (particularly zero), so that errors could go undetected. Similarly, it is preferable that no two elements of \textit{x} should be the same.

\textbf{n}

type: integer; \textbf{default} = \text{nrow(x)}

The number \( n \) of independent variables in the objective function.

\textbf{Details}

\textbf{R} interface to the NAG Fortran routine \textbf{E04HCF}.

\textbf{Value}

\textbf{F}

double

Unless you set \textit{iflag} negative in the first call of \textit{funct}, \textit{f} contains the value of the objective function \( F(x) \) at the point given by you in \textit{x}.

\textbf{G}

double array

Unless you set \textit{iflag} negative in the first call of \textit{funct}, \textit{g[j]} contains the value of the derivative \( \frac{\partial F}{\partial x_j} \) at the point given in \textit{x}, as calculated by \textit{funct} for \( j = 1 \ldots n \).

\textbf{IFAIL}

integer

\( \text{ifail} = 0 \) unless the function detects an error or a warning has been flagged (see the \textbf{Errors} section in Fortran library documentation).

\textbf{Author(s)}

\textbf{NAG}

\textbf{References}

\url{http://www.nag.co.uk/numeric/FL/nagdoc_fl23/pdf/E04/e04hcf.pdf}

\textbf{Examples}

\begin{verbatim}
ifail <- 0
funct = function(iflag, n, xc) {
  gc <- as.matrix(mat.or.vec(n, 1))
  fc <- 0
  if (iflag != 1) {
  }
  if (iflag != 0) {
  }
}
\end{verbatim}
\begin{verbatim}


}  #
list(IFLAG = as.integer(iflag), FC = fc, GC = as.matrix(gc))

x <- matrix(c(1.46, -0.82, 0.57, 1.21), nrow = 4,
ncol = 1, byrow = TRUE)

e04hc(funct, x)
\end{verbatim}

**Description**

e04hd checks that a function for calculating second derivatives of an objective function is consistent with a function for calculating the corresponding first derivatives.

**Usage**

e04hd(funct, h, x, lh,
n = nrow(x))

**Arguments**

- **funct**  
  function  
  funct must evaluate the function and its first derivatives at a given point. (e04lb gives you the option of resetting arguments of funct to cause the minimization process to terminate immediately. e04hd will also terminate immediately, without finishing the checking process, if the argument in question is reset.)  
  (IFLAG, FC, GC) = funct(iflag, n, xc)

- **h**  
  function  
  h must evaluate the second derivatives of the function at a given point. (As with funct, a argument can be set to cause immediate termination.)  
  (IFLAG, FHESL, FHESD) = h(iflag, n, xc, lh, fhesd)

- **x**  
  double array  
  x[j] for j = 1...n must contain the coordinates of a suitable point at which to check the derivatives calculated by funct. 'Obvious' settings, such as 0.0 or 1.0,
should not be used since, at such particular points, incorrect terms may take
correct values (particularly zero), so that errors could go undetected. Similarly,
it is advisable that no two elements of \( x \) should be the same.

\begin{verbatim}
integer
\end{verbatim}

\( n \) integer: \textbf{default} = \texttt{nrow(x)}

The number \( n \) of independent variables in the objective function.

\textbf{Details}

R interface to the NAG Fortran routine \texttt{E04HDF}.

\textbf{Value}

\( G \) double array

Unless you set \texttt{iflag} negative in the first call of \texttt{funct}, \( g[j] \) contains the value
of the first derivative \( \frac{\partial F}{\partial x_j} \) at the point given in \( x \), as calculated by \texttt{funct} for
\( j = 1 \ldots n \).

\( \text{HESL} \) double array

Unless you set \texttt{iflag} negative in \texttt{h}, \texttt{hesl} contains the strict lower triangle of the
second derivative matrix of \( F \), as evaluated by \texttt{h} at the point given in \( x \), stored
by rows.

\( \text{HESD} \) double array

Unless you set \texttt{iflag} negative in \texttt{h}, \texttt{hesd} contains the diagonal elements of the
second derivative matrix of \( F \), as evaluated by \texttt{h} at the point given in \( x \).

\( \text{IFAIL} \) integer

\( \text{ifail} = 0 \) unless the function detects an error or a warning has been flagged (see
the \texttt{Errors} section in Fortran library documentation).

\textbf{Author(s)}

NAG

\textbf{References}

http://www.nag.co.uk/numeric/FL/nagdoc_fl23/pdf/E04/e04hdf.pdf

\textbf{Examples}

\begin{verbatim}
ifail <- 0
funct = function(iflag, n, xc) {
    gc <- as.matrix(mat.or.vec(n, 1))
    list(IFLAG = as.integer(iflag), FC = fc, GC = as.matrix(gc))
}
\end{verbatim}
hess = function(iflag, n, xc, lh, fhesd) {

  fhesl <- as.matrix(mat.or.vec(lh, 1))
  fhesd <- as.matrix(mat.or.vec(n, 1))
  fhesl[1] <- 20
  fhesl[2] <- 0
  fhesl[5] <- 0
  fhesl[6] <- -10
  list(IFLAG = as.integer(iflag), FHESL = as.matrix(fhesl),
       FHESD = as.matrix(fhesd))
}

x <- matrix(c(1.46, -0.82, 0.57, 1.21), nrow = 4, ncol = 1, byrow = TRUE)

lh <- 6
iw <- matrix(c(0), nrow = 1, ncol = 1, byrow = TRUE)

w <- as.matrix(mat.or.vec(20, 1))
e04hd(funct, hess, x, lh)

---

**Description**

e04he is a comprehensive modified Gauss-Newton algorithm for finding an unconstrained minimum of a sum of squares of \( m \) nonlinear functions in \( n \) variables \((m \geq n)\). First and second derivatives are required.

The function is intended for functions which have continuous first and second derivatives (although it will usually work even if the derivatives have occasional discontinuities).

**Usage**

e04he(m, lsqfun, lsqhes, lsqmon, maxcal, xtol, x, n = nrow(x),
      iprint = 1,
      eta = if (n==1) 0.0 else 0.5,
      stepmx = 100000.0)
Arguments

m
- integer
lsqfun
- function
lsqfun must calculate the vector of values \( f_i(x) \) and Jacobian matrix of first derivatives \( \frac{\partial f_i}{\partial x_j} \) at any point \( x \). (However, if you do not wish to calculate the residuals or first derivatives at a particular \( x \), there is the option of setting a argument to cause e04he to terminate immediately.)

(IFLAG,FVEC,FJAC) = lsqfun(iflag,m,n,xc,ldfjac)

lsqhes
- function
lsqhes must calculate the elements of the symmetric matrix

\[
B(x) = \sum_{i=1}^{m} f_i(x) G_i(x),
\]

at any point \( x \), where \( G_i(x) \) is the Hessian matrix of \( f_i(x) \). (As with lsqfun, there is the option of causing e04he to terminate immediately.)

(IFLAG,B) = lsqhes(iflag,m,n,fvec,xc,lb)

lsqmon
- function
If \( iprint \geq 0 \), you must supply lsqmon which is suitable for monitoring the minimization process. lsqmon must not change the values of any of its arguments.

() = lsqmon(m,n,xc,fvec,fjac,ldfjac,s,igrade,niter,nf)

maxcal
- integer
This argument is present so as to enable you to limit the number of times that lsqfun is called by e04he. There will be an error exit (see the Errors section in Fortran library documentation) after maxcal calls of lsqfun.

xtol
- double
The accuracy in \( x \) to which the solution is required.

x
- double array
\( x[j] \) must be set to a guess at the \( j \)th component of the position of the minimum for \( j = 1 \ldots n \).

n
- integer: default = nrow(x)
The number \( m \) of residuals, \( f_i(x) \), and the number \( n \) of variables, \( x_j \).

iprint
- integer: default = 1
Specifies the frequency with which lsqmon is to be called.

\( iprint > 0 \): lsqmon is called once every \( iprint \) iterations and just before exit from e04he.

\( iprint = 0 \): lsqmon is just called at the final point.

\( iprint < 0 \): lsqmon is not called at all.

eta
- double: default = if (n==1) 0.0 else 0.5
Every iteration of e04he involves a linear minimization (i.e., minimization of \( F(x^{(k)} + \alpha^{(k)} p^{(k)}) \) with respect to \( \alpha^{(k)} \)). eta must lie in the range \( 0.0 \leq \eta < 1.0 \), and specifies how accurately these linear minimizations are to be performed. The minimum with respect to \( \alpha^{(k)} \) will be located more accurately for small values of \( \eta \) (say, 0.01) than for large values (say, 0.9).

stepmx
- double: default = 100000.0
An estimate of the Euclidean distance between the solution and the starting point supplied by you. (For maximum efficiency, a slight overestimate is preferable.)
Details

R interface to the NAG Fortran routine E04HEF.

Value

- **X**: double array
  The final point $x^{(k)}$. Thus, if ifail = 0 on exit, $x[j]$ is the $j$th component of the estimated position of the minimum.

- **FSUMSQ**: double
  The value of $F(x)$, the sum of squares of the residuals $f_i(x)$, at the final point given in $x$.

- **FVEC**: double array
  The value of the residual $f_i(x)$ at the final point given in $x$ for $i = 1 \ldots m$.

- **FJAC**: double array
  The value of the first derivative $\frac{\partial f_i}{\partial x_j}$ evaluated at the final point given in $x$ for $j = 1 \ldots n$ for $i = 1 \ldots m$.

- **S**: double array
  The singular values of the Jacobian matrix at the final point. Thus $s$ may be useful as information about the structure of your problem.

- **V**: double array
  The matrix $V$ associated with the singular value decomposition $J = USV^T$ of the Jacobian matrix at the final point, stored by columns. This matrix may be useful for statistical purposes, since it is the matrix of orthonormalized eigenvectors of $J^TJ$.

- **NITER**: integer
  The number of iterations which have been performed in e04he.

- **NF**: integer
  The number of times that the residuals and Jacobian matrix have been evaluated (i.e., number of calls of Isqfun).

- **IFAIL**: integer
  ifail = 0 unless the function detects an error or a warning has been flagged (see the Errors section in Fortran library documentation).

Author(s)

NAG

References

http://www.nag.co.uk/numeric/FL/nagdoc_fl23/pdf/E04/e04hef.pdf
Examples

```r
e04he

ifail <- 0
lsqfun = function(iflag, m, n, xc, ljc) {
  fvec <- as.matrix(mat.or.vec(m, 1))
  fjacc <- as.matrix(mat.or.vec(ljc, n))
  for (i in c(1:m)) {
    fvec[i] <- xc[1] + t[i, 1]/denom - y[i]
    if (iflag != 0) {
      fjacc[i, 1] <- 1
      dummy <- -1/(denom %*% denom)
      fjacc[i, 2] <- t[i, 1] %*% t[i, 2] %*% dummy
      fjacc[i, 3] <- t[i, 1] %*% t[i, 3] %*% dummy
    }
  }
  list(IFLAG = as.integer(iflag), FVEC = as.matrix(fvec), FJAC = as.matrix(fjacc))
}

lsqhes = function(iflag, m, n, fvec, xc, lb) {
  b <- as.matrix(mat.or.vec(lb, 1))
  b[1] <- 0
  b[2] <- 0
  sum22 <- 0
  sum32 <- 0
  sum33 <- 0
  for (i in c(1:m)) {
    dummy <- 2 %*% t[i, 1]/(xc[2] %*% t[i, 2] + xc[3] %*% t[i, 3])^3
    sum22 <- sum22 + fvec[i] %*% dummy %*% t[i, 2]^2
    sum32 <- sum32 + fvec[i] %*% dummy %*% t[i, 2] %*% t[i, 3]
    sum33 <- sum33 + fvec[i] %*% dummy %*% t[i, 3]^2
  }
  b[3] <- sum22
  b[4] <- 0
  b[5] <- sum32
  b[6] <- sum33
  list(IFLAG = as.integer(iflag), B = as.matrix(b))
}

lsqmon = function(m, n, xc, fvec, fjacc, ljc, s, igrade, niter, nf) {
  list()
}
```
m <- 15
define maxcal <- 150
xtol <- 1.05418557512311e-07
x <- matrix(c(0.5, 1, 1.5), nrow = 3, ncol = 1, byrow = TRUE)
iw <- matrix(c(0), nrow = 1, ncol = 1, byrow = TRUE)
w <- as.matrix(mat.or.vec(105, 1))
y <- matrix(c(0.14, 0.18, 0.22, 0.25, 0.29, 0.32,
            0.35, 0.39, 0.37, 0.73, 0.96, 1.34, 2.1, 4.39), nrow = 1,
            ncol = 15, byrow = TRUE)
t <- matrix(c(1, 15, 1, 2, 14, 2, 3, 13, 3, 4, 12,
            4, 5, 11, 5, 6, 10, 6, 7, 9, 7, 8, 8, 9, 7, 7, 10, 6,
            6, 11, 5, 12, 4, 4, 13, 3, 3, 14, 2, 2, 15, 1, 1), nrow = 15,
            ncol = 3, byrow = TRUE)
e04he(m, lsqfun, lsqhes, lsqmon, maxcal, xtol, x)

==e04hy==
e04hy: Unconstrained minimum of a sum of squares, combined Gauss-
Newton and modified Newton algorithm, using second derivatives
(easy-to-use)

Description
e04hy is an easy-to-use modified Gauss-Newton algorithm for finding an unconstrained minimum
of a sum of squares of \( m \) nonlinear functions in \( n \) variables \( (m \geq n) \). First and second deriva-
tives are required.

It is intended for functions which are continuous and which have continuous first and second deriva-
tives (although it will usually work even if the derivatives have occasional discontinuities).

Usage
e04hy(m, lsfun2, lsqhes, x,
    n = nrow(x))
**Arguments**

m  integer

lsfun2  function

You must supply this function to calculate the vector of values \( f_i(x) \) and the Jacobian matrix of first derivatives \( \frac{\partial f_i}{\partial x_j} \) at any point \( x \). It should be tested separately before being used in conjunction with e04hy (see the E04 chapter introduction in the Fortran Library documentation).

\[(FVEC, FJAC) = \text{lsfun2}(m, n, xc, ldfjac)\]

lshes2  function

You must supply this function to calculate the elements of the symmetric matrix

\[ B(x) = \sum_{i=1}^{m} f_i(x) G_i(x), \]

at any point \( x \), where \( G_i(x) \) is the Hessian matrix of \( f_i(x) \). It should be tested separately before being used in conjunction with e04hy (see the E04 chapter introduction in the Fortran Library documentation).

\[(B) = \text{lshes2}(m, n, fvec, xc, lb)\]

x  double array

\( x[j] \) must be set to a guess at the \( j \)th component of the position of the minimum for \( j = 1 \ldots n \). The function checks lsfun2 and lshes2 at the starting point and so is more likely to detect any error in your functions if the initial \( x[j] \) are nonzero and mutually distinct.

n  integer: default = nrow(x)

The number \( m \) of residuals, \( f_i(x) \), and the number \( n \) of variables, \( x_j \).

**Details**

R interface to the NAG Fortran routine E04HYF.

**Value**

\( X \)  double array

The lowest point found during the calculations. Thus, if ifail = 0 on exit, \( x[j] \) is the \( j \)th component of the position of the minimum.

FSUMSQ  double

The value of the sum of squares, \( F(x) \), corresponding to the final point stored in \( x \).

IFAIL  integer

ifail = 0 unless the function detects an error or a warning has been flagged (see the Errors section in Fortran library documentation).

**Author(s)**

NAG

**References**

http://www.nag.co.uk/numeric/FL/nagdoc_fl23/pdf/E04/e04hyf.pdf
ifail <- 0
lsfun2 = function(m, n, xc, ljc) {
    fvec <- as.matrix(mat.or.vec(m, 1))
    fjacc <- as.matrix(mat.or.vec(ljc, n))
    for (i in c(1:m)) {
        fvec[i] <- xc[1] + user(2)[i, 1]/denom - user(1)[i]
        fjacc[i, 1] <- 1
        dummy <- -1/(denom * denom)
        fjacc[i, 2] <- user(2)[i, 1] %*% user(2)[i, 2] %*% dummy
        fjacc[i, 3] <- user(2)[i, 1] %*% user(2)[i, 3] %*% dummy
    }
    list(FVEC = as.matrix(fvec), FJAC = as.matrix(fjacc))
}

lshes2 = function(m, n, fvec, xc, lb) {
    b <- as.matrix(mat.or.vec(lb, 1))
    sum22 <- 0
    sum32 <- 0
    sum33 <- 0
    for (i in c(1:m)) {
        dummy <- 2 %*% user(2)[i, 1]/(xc[2] %*% user(2)[i, 2] + xc[3] %*% user(2)[i, 3])^3
        sum22 <- sum22 + fvec[i] %*% dummy %*% user(2)[i, 2]^2
        sum32 <- sum32 + fvec[i] %*% dummy %*% user(2)[i, 2] %*% user(2)[i, 3]
        sum33 <- sum33 + fvec[i] %*% dummy %*% user(2)[i, 3]^2
    }
    b[3] <- sum22
    b[5] <- sum32
    b[6] <- sum33
    list(B = as.matrix(b))
}

m <- 15
x <- matrix(c(0.5, 1, 1.5), nrow = 3, ncol = 1, byrow = TRUE)

x <- matrix(c(0.5, 1, 1.5), nrow = 3, ncol = 1, byrow = TRUE)
e04jc

\[ y \leftarrow \begin{bmatrix} 0.14, 0.18, 0.22, 0.25, 0.29, 0.32, \\ 0.35, 0.39, 0.37, 0.58, 0.73, 0.96, 1.34, 2.1, 4.39 \end{bmatrix}^T \]

\[ t \leftarrow \begin{bmatrix} 1, 15, 1, 2, 14, 2, 3, 13, 3, 4, 12, \\ 4, 5, 11, 5, 6, 10, 6, 7, 9, 7, 8, 8, 9, 7, 7, 10, 6, \\ 6, 11, 5, 5, 12, 4, 4, 13, 3, 3, 14, 2, 2, 15, 1, 1 \end{bmatrix} \]

user <- function(switch_integer) {
  switch(switch_integer, y, t, 3)
}

e04hy(m, lsfun2, lshes2, x)

e04jc

e04jc: Minimum by quadratic approximation, function of several variables, simple bounds, using function values only

Description

e04jc is an easy-to-use algorithm that uses methods of quadratic approximation to find a minimum of an objective function \( F \) over \( x \in \mathbb{R}^n \), subject to fixed lower and upper bounds on the independent variables \( x_1, x_2, \ldots, x_n \). Derivatives of \( F \) are not required.

The function is intended for functions that are continuous and that have continuous first and second derivatives (although it will usually work even if the derivatives have occasional discontinuities). Efficiency is maintained for large \( n \).

Usage

e04jc(objfun, npt, x, bl, bu, rhobeg, rhoend, monfun, maxcal, 
\( n = nrow(x) \))

Arguments

objfun

function

objfun must evaluate the objective function \( F \) at a specified vector \( x \).

\( (F, \text{INFORM}) = \text{objfun}(n, x) \)

npt

integer

\( m \), the number of interpolation conditions imposed on the quadratic approximation at each iteration.

x

double array

An estimate of the position of the minimum. If any component is out-of-bounds it is replaced internally by the bound it violates.

bl

double array
bu double array
The fixed vectors of bounds: the lower bounds $\ell$ and the upper bounds $u$, respectively. To signify that a variable is unbounded you should choose a large scalar $r$ appropriate to your problem, then set the lower bound on that variable to $-r$ and the upper bound to $r$. For well-scaled problems $r = r_{\text{max}}^{\frac{1}{4}}$ may be suitable, where $r_{\text{max}}$ denotes the largest positive model number (see x02al).

rhobeg double
An initial lower bound on the value of the trust-region radius.

rhoend double
A final lower bound on the value of the trust-region radius.

monfun function
monfun may be used to monitor the optimization process. It is invoked every time a new trust-region radius is chosen.

maxcal integer
The maximum permitted number of calls to objfun.

n integer: default = nrow(x)
n, the number of independent variables.

Details
R interface to the NAG Fortran routine E04JCF.

Value

X double array
The lowest point found during the calculations. Thus, if ifail = 0 on exit, x is the position of the minimum.

F double
The function value at the lowest point found (x).

NF integer
Unless ifail = 1, ifail = -999 on exit, the total number of calls made to objfun.

IFAIL integer
ifail = 0 unless the function detects an error or a warning has been flagged (see the Errors section in Fortran library documentation).

Author(s)
NAG

References
http://www.nag.co.uk/numeric/FL/nagdoc_fl23/pdf/E04/e04jcf.pdf
Examples

ifail <- 0
maxcal <- 500
rhobeg <- 0.1
rhoend <- 1e-06
n <- 4
npt <- 2 * n + 1
infbnd <- x02al()["result"]^0.25

bl <- matrix(c(1, -2, -infbnd, 1), nrow = 4, ncol = 1,
byrow = TRUE)

bu <- matrix(c(3, 0, infbnd, 3), nrow = 4, ncol = 1,
byrow = TRUE)

x <- matrix(c(3, -1, 0, 1), nrow = 4, ncol = 1, byrow = TRUE)
e04jc_objfun = function(n, x) {
inform <- 0
f <- (x[1] + 10 %*% x[2])^2 + 5 %*% (x[3] - x[4])^2 + (x[2] -
2 %*% x[3])^4 + 10 %*% (x[1] - x[4])^4
list(F = f, INFORM = as.integer(inform))
}
e04jc_monfun = function(n, nf, x, f, rho) {
inform <- 0
writeLines(sprintf("\nNew rho = %13.5e, number of function evaluations = %d\n",
rho, nf))
writeLines(sprintf("Current function value = %13.5en",
f))
writeLines(sprintf("The corresponding X is:",
"\n"))
writeLines(sprintf(" %13.5e", x, "\n"))
writeLines(sprintf("\n", "\n"))
list(INFORM = as.integer(inform))
}
ans <- e04jc(e04jc_objfun, npt, x, bl, bu, rhobeg, 
rhoend, e04jc_monfun, maxcal)

print(ans$X)
print(ans$F)
print(ans$NF)
print(ans$IFAIL)

e04jy

e04jy: Minimum, function of several variables, quasi-Newton algorithm, simple bounds, using function values only (easy-to-use)

Description

e04jy is an easy-to-use quasi-Newton algorithm for finding a minimum of a function \( F(x_1, x_2, \ldots, x_n) \), subject to fixed upper and lower bounds of the independent variables \( x_1, x_2, \ldots, x_n \), using function values only.

It is intended for functions which are continuous and which have continuous first and second derivatives (although it will usually work even if the derivatives have occasional discontinuities).

Usage

e04jy(ibound, funct1, bl, bu, x,  
n=nrow(bl),  
liw=n+2,  
lw=max(n*(n-1)/2+12*n,13)  
)

Arguments

ibound
integer
Indicates whether the facility for dealing with bounds of special forms is to be used.

funct1
void function
You must supply funct1 to calculate the value of the function \( F(x) \) at any point \( x \). It should be tested separately before being used with e04jy (see the E04 chapter introduction in the Fortran Library documentation).

bl
double array
The lower bounds \( l_j \).

bu
double array
The upper bounds \( u_j \).

x
double array
\( x(j) \) must be set to an estimate of the \( j \)th component of the position of the minimum for \( j = 1 \ldots n \).
\textbf{n} \quad \text{integer: default = nrow(bl)}

The number \( n \) of independent variables.

\textbf{liw} \quad \text{integer: default = n+2}

\textbf{lw} \quad \text{integer: default = max(n*(n-1)/2+12*n,13)}

\textbf{Details}

R interface to the NAG Fortran routine E04JYF.

\textbf{Value}

\textbf{bl} \quad \text{double array}

The lower bounds actually used by e04jy.

\textbf{bu} \quad \text{double array}

The upper bounds actually used by e04jy.

\textbf{x} \quad \text{double array}

The lowest point found during the calculations. Thus, if ifail = 0 on exit, \( x(j) \) is the \( j \)th component of the position of the minimum.

\textbf{f} \quad \text{double}

The value of \( F(x) \) corresponding to the final point stored in \( x \).

\textbf{iw} \quad \text{integer array}

If ifail = 0, ifail = 3, ifail = 5, the first \( n \) elements of \( iw \) contain information about which variables are currently on their bounds and which are free. Specifically, if \( x_i \) is:

- fixed on its upper bound, \( iw(i) \) is \(-1\);
- fixed on its lower bound, \( iw(i) \) is \(-2\);
- effectively a constant (i.e., \( l_j = u_j \)), \( iw(i) \) is \(-3\);
- free, \( iw(i) \) gives its position in the sequence of free variables.

\textbf{w} \quad \text{double array}

If ifail = 0, ifail = 3, ifail = 5, \( w(i) \) contains a finite difference approximation to the \( i \)th element of the projected gradient vector \( g_z \) for \( i = 1 \ldots n \). In addition, \( w(n+1) \) contains an estimate of the condition number of the projected Hessian matrix (i.e., \( k \)). The rest of the array is used as workspace.

\textbf{Author(s)}

NAG

\textbf{References}

http://www.nag.co.uk/numeric/FL/nagdoc_fl23/pdf/E04/e04jyf.pdf

\textbf{Examples}

```r
e04jy_funct1 = function(n, xc, fc) {
    list(FC = fc)
}
```
ibound <- 0
bl <- matrix(c(1, -2, -1e+06, 1), nrow = 4, ncol = 1, byrow = TRUE)

bu <- matrix(c(3, 0, 1e+06, 3), nrow = 4, ncol = 1, byrow = TRUE)

x <- matrix(c(3, -1, 0, 1), nrow = 4, ncol = 1, byrow = TRUE)

e04jy(ibound, e04jy_funct1, bl, bu, x)

e04kd: Minimum, function of several variables, modified Newton algorithm, simple bounds, using first derivatives (comprehensive)

Description

e04kd is a comprehensive modified Newton algorithm for finding:
- an unconstrained minimum of a function of several variables;
- a minimum of a function of several variables subject to fixed upper and/or lower bounds on the variables.

First derivatives are required. The function is intended for functions which have continuous first and second derivatives (although it will usually work even if the derivatives have occasional discontinuities).

Usage

e04kd(funct, monit, eta, ibound, bl, bu, x, lh, iw, w, n=nrow(bl), iprint=1, maxcal=50, xtol=0.0, delta=0.0, stepmx=100000.0)

Arguments

funct

void function

d must evaluate the function \( F(x) \) and its first derivatives \( \frac{\partial F}{\partial x_j} \) at a specified point. (However, if you do not wish to calculate \( F \) or its first derivatives at a particular \( x \), there is the option of setting a argument to cause e04kd to terminate immediately.)

monit

void function

If \( iprint \geq 0 \), you must supply monit which is suitable for monitoring the minimization process. monit must not change the values of any of its arguments.
Every iteration of e04kd involves a linear minimization (i.e., minimization of $F(x + \alpha p)$ with respect to $\alpha$). $\eta$ specifies how accurately these linear minimizations are to be performed. The minimum with respect to $\alpha$ will be located more accurately for small values of $\eta$ (say, 0.01) than large values (say, 0.9).

Indicates whether the problem is unconstrained or bounded. If there are bounds on the variables, ibound can be used to indicate whether the facility for dealing with bounds of special forms is to be used. It must be set to one of the following values:

- $\text{ibound} = 0$: If the variables are bounded and you are supplying all the $l_j$ and $u_j$ individually.
- $\text{ibound} = 1$: If the problem is unconstrained.
- $\text{ibound} = 2$: If the variables are bounded, but all the bounds are of the form $0 \leq x_j$.
- $\text{ibound} = 3$: If all the variables are bounded, and $l_1 = l_2 = \cdots = l_n$ and $u_1 = u_2 = \cdots = u_n$.
- $\text{ibound} = 4$: If the problem is unconstrained. (The $\text{ibound} = 4$ option is provided for consistency with other functions. In e04kd it produces the same effect as $\text{ibound} = 1$.)

The fixed lower bounds $l_j$.

The fixed upper bounds $u_j$.

$x(j)$ must be set to a guess at the $j$th component of the position of the minimum for $j = 1 \ldots n$.

The number $n$ of independent variables.

The frequency with which monit is to be called.

- $\text{iprint} > 0$: monit is called once every $\text{iprint}$ iterations and just before exit from e04kd.
- $\text{iprint} = 0$: monit is just called at the final point.
- $\text{iprint} < 0$: monit is not called at all.

The maximum permitted number of evaluations of $F(x)$, i.e., the maximum permitted number of calls of funct with iflag set to 2. It should be borne in mind that, in addition to the calls of funct which are limited directly by maxcal, there will be calls of funct (with iflag set to 1) to evaluate only first derivatives.

The accuracy in $x$ to which the solution is required.
delta  double: default = 0.0
The differencing interval to be used for approximating the second derivatives of \( F(x) \). Thus, for the finite difference approximations, the first derivatives of \( F(x) \) are evaluated at points which are delta apart. If \( \epsilon \) is the machine precision, then \( \sqrt{\epsilon} \) will usually be a suitable setting for delta. If you set delta to 0.0 (or to any positive value less than \( \epsilon \)), e04kd will automatically use \( \sqrt{\epsilon} \) as the differencing interval.

stepmx  double: default = 100000.0
An estimate of the Euclidean distance between the solution and the starting point supplied by you. (For maximum efficiency a slight overestimate is preferable.)

Details
R interface to the NAG Fortran routine E04KDF.

Value
bl  double array
The lower bounds actually used by e04kd, e.g., if ibound = 2, \( bl(1) = bl(2) = \cdots = bl(n) = 0.0 \).

bu  double array
The upper bounds actually used by e04kd, e.g., if ibound = 2, \( bu(1) = bu(2) = \cdots = bu(n) = 10^6 \).

x  double array
The final point \( x^{(k)} \). Thus, if ifail = 0 on exit, \( x(j) \) is the \( j \)th component of the estimated position of the minimum.

hesl  double array
During the determination of a direction \( p_z \) (see the Description in Fortran library documentation), \( H + E \) is decomposed into the product \( LDL^T \), where \( L \) is a unit lower triangular matrix and \( D \) is a diagonal matrix. (The matrices \( H, E, L \) and \( D \) are all of dimension \( n_z \), where \( n_z \) is the number of variables free from their bounds. \( H \) consists of those rows and columns of the full estimated second derivative matrix which relate to free variables. \( E \) is chosen so that \( H + E \) is positive definite.)

hesd  double array
During the determination of a direction \( p_z \) (see the Description in Fortran library documentation), \( H + E \) is decomposed into the product \( LDL^T \), where \( L \) is a unit lower triangular matrix and \( D \) is a diagonal matrix. (The matrices \( H, E, L \) and \( D \) are all of dimension \( n_z \), where \( n_z \) is the number of variables free from their bounds. \( H \) consists of those rows and columns of the full estimated second derivative matrix which relate to free variables. \( E \) is chosen so that \( H + E \) is positive definite.)

istate  integer array
Information about which variables are currently on their bounds and which are free. If istate\((j)\) is:
- equal to \(-1\), \( x_j \) is fixed on its upper bound;
- equal to \(-2\), \( x_j \) is fixed on its lower bound;
- equal to \(-3\), \( x_j \) is effectively a constant (i.e., \( l_j = u_j \));
- positive, istate\((j)\) gives the position of \( x_j \) in the sequence of free variables.
The function value at the final point given in $x$.

The first derivative vector corresponding to the final point given in $x$. The components of $g$ corresponding to free variables should normally be close to zero.

Author(s)
NAG

References
http://www.nag.co.uk/numeric/FL/nagdoc_fl23/pdf/E04/e04kdf.pdf

Examples

```r
e04kd_funct = function(iflag, n, xc, fc, gc) {
  gc <- as.matrix(mat.or.vec(n, 1))
  fc <- 0
  if (iflag != 1) {
  }
  list(IFLAG = iflag, FC = fc, GC = as.matrix(gc))
}

e04kd_monit = function(n, xc, fc, gc, istate, gpjnrm, cond, posdef, niter, nf) {
  sprintf("\n Itn Fn evals Fn value Norm of proj gradient\n", "\n")
  sprintf(" %d %5d %20.4f %20.4f\n", niter, nf, fc, gpjnrm,
    "\n")
  sprintf(" J XJ GJ Status\n", "\n")
  for (j in 1:n) {
    isj <- istate[j]
    if (isj > 0) {
      sprintf("%2d %16.4f%20.4f %s\n", j, xc, j, gc, j,
        " Free", "\n")
    }
    else if (isj == -1) {
    }
    else if (isj == -2) {
  }
```
else if (isj == -3) {
}

if (cond != 0) {
    if (cond > 1e+06) {
        sprintf("Estimated condition number of projected Hessian is more than 1.0e+6\n\n")
    } else {
        sprintf("Estimated condition number of projected Hessian = %10.2f\n", cond, "\n")
    }
    if (!posdef) {
        sprintf("Projected Hessian matrix is not positive definite\n\n")
    }
    list()
}

eta <- 0.5
ibound <- 0
bl <- matrix(c(1, -2, -1e+06, 1), nrow = 4, ncol = 1, byrow = TRUE)
bu <- matrix(c(3, 0, 1e+06, 3), nrow = 4, ncol = 1, byrow = TRUE)
x <- matrix(c(3, -1, 0, 1), nrow = 4, ncol = 1, byrow = TRUE)
lh <- 6
iw <- matrix(c(0, 0), nrow = 2, ncol = 1, byrow = TRUE)
w <- as.matrix(mat.or.vec(34, 1))
e04kd(e04kd_funct, e04kd_monit, eta, ibound, bl, bu, x, lh, iw, w)

---

e04ky: Minimum, function of several variables, quasi-Newton algorithm, simple bounds, using first derivatives (easy-to-use)
**Description**

e04ky is an easy-to-use quasi-Newton algorithm for finding a minimum of a function \( F(x_1 x_2 \ldots x_n) \), subject to fixed upper and lower bounds on the independent variables \( x_1, x_2, \ldots, x_n \), when first derivatives of \( F \) are available.

It is intended for functions which are continuous and which have continuous first and second derivatives (although it will usually work even if the derivatives have occasional discontinuities).

**Usage**

e04ky(ibound, funct2, bl, bu, x,
   n = nrow(bl),
   liw = (n+2),
   lw = (max((10*n+n*(n-1)/2),11)))

**Arguments**

- **ibound** integer
  Indicates whether the facility for dealing with bounds of special forms is to be used. It must be set to one of the following values:
  - \( ibound = 0 \): If you are supplying all the \( l_j \) and \( u_j \) individually.
  - \( ibound = 1 \): If there are no bounds on any \( x_j \).
  - \( ibound = 2 \): If all the bounds are of the form \( 0 \leq x_j \).
  - \( ibound = 3 \): If \( l_1 = l_2 = \cdots = l_n \) and \( u_1 = u_2 = \cdots = u_n \).

- **funct2** function
  You must supply funct2 to calculate the values of the function \( F(x) \) and its first derivative \( \frac{\partial F}{\partial x_j} \) at any point \( x \). It should be tested separately before being used in conjunction with e04ky (see the E04 chapter introduction in the Fortran Library documentation).

  \[(FC, GC) = \text{funct2}(n, xc)\]

- **bl** double array
  The lower bounds \( l_j \).

- **bu** double array
  The upper bounds \( u_j \).

- **x** double array
  \( x[j] \) must be set to a guess at the \( j \)th component of the position of the minimum for \( j = 1 \ldots n \). The function checks the gradient at the starting point, and is more likely to detect any error in your programming if the initial \( x[j] \) are nonzero and mutually distinct.

- **n** integer: default = nrow(bl)
  The number \( n \) of independent variables.

- **liw** integer: default = (n+2)
  \( liw \)

- **lw** integer: default = (max((10*n+n*(n-1)/2),11))
  \( lw \)

**Details**

R interface to the NAG Fortran routine E04KYF.
Value

BL  double array
   The lower bounds actually used by e04ky.

BU  double array
   The upper bounds actually used by e04ky.

X   double array
   The lowest point found during the calculations. Thus, if ifail = 0 on exit, \( x[j] \)
   is the \( j \)th component of the position of the minimum.

F   double
   The value of \( F(x) \) corresponding to the final point stored in x.

G   double array
   The value of \( \frac{\partial F}{\partial x_j} \) corresponding to the final point stored in x for \( j = 1 \ldots n \); the
   value of \( g[j] \) for variables not on a bound should normally be close to zero.

IW  integer array
   If ifail = 0, ifail = 3, ifail = 5, the first \( n \) elements of iw contain information
   about which variables are currently on their bounds and which are free. Specifically, if \( x_i \) is:
   - fixed on its upper bound, \( iw[i] \) is \(-1\);
   - fixed on its lower bound, \( iw[i] \) is \(-2\);
   - effectively a constant (i.e., \( l_j = u_j \)), \( iw[i] \) is \(-3\);
   - free, \( iw[i] \) gives its position in the sequence of free variables.

W   double array
   If ifail = 0, ifail = 3, ifail = 5, \( w[i] \) contains the \( i \)th element of the projected
   gradient vector \( g_z \) for \( i = 1 \ldots n \). In addition, \( w[n+1] \) contains an estimate of
   the condition number of the projected Hessian matrix (i.e., \( k \)). The rest of the
   array is used as workspace.

IFAIL integer
   ifail = 0 unless the function detects an error or a warning has been flagged (see
   the Errors section in Fortran library documentation).

Author(s)

NAG

References

http://www.nag.co.uk/numeric/FL/nagdoc_fl23/pdf/E04/e04kyf.pdf

Examples

ifail<-0
funct2=function(n,xc){

gc<-as.matrix(mat.or.vec(n,1))
fc<-\((xc[1]+10*x[2])^2+5*(xc[3]-xc[4])^2+(xc[2]-2*x[3])^4+10*(xc[1]-xc[4])^4

if (ifail==0)
cf<-funct2(n,xc)
gc<-grad(funct2,n,xc)

e04kz

```r
list(FC=fc,GC=as.matrix(gc))
```

```c
e04kz
```

```c
e04kz: Minimum, function of several variables, modified Newton algorithm, simple bounds, using first derivatives (easy-to-use)
```

**Description**

e04kz is an easy-to-use modified Newton algorithm for finding a minimum of a function $F(x_1, x_2, \ldots, x_n)$, subject to fixed upper and lower bounds on the independent variables $x_1, x_2, \ldots, x_n$, when first derivatives of $F$ are available. It is intended for functions which are continuous and which have continuous first and second derivatives (although it will usually work even if the derivatives have occasional discontinuities).

**Usage**

```c
e04kz(ibound, funct2, bl, bu, x, n=nrow(bl))
```

**Arguments**

- **ibound** integer
  Indicates whether the facility for dealing with bounds of special forms is to be used. It must be set to one of the following values:
  - $ibound = 0$: If you are supplying all the $l_j$ and $u_j$ individually.
  - $ibound = 1$: If there are no bounds on any $x_j$.
  - $ibound = 2$: If all the bounds are of the form $0 \leq x_j$.
  - $ibound = 3$: If $l_1 = l_2 = \cdots = l_n$ and $u_1 = u_2 = \cdots = u_n$.

- **funct2** void function
  You must supply this function to calculate the values of the function $F(x)$ and its first derivatives $\frac{dF}{dx_j}$ at any point $x$. It should be tested separately before being used in conjunction with e04kz (see the E04 chapter).
b l  double array
The lower bounds \( l_j \).

b u  double array
The upper bounds \( u_j \).

x  double array
\( x(j) \) must be set to a guess at the \( j \)th component of the position of the minimum for \( j = 1 \ldots n \). The function checks the gradient at the starting point, and is more likely to detect any error in your programming if the initial \( x(j) \) are nonzero and mutually distinct.

n  integer: default = nrow(bl)
The number \( n \) of independent variables.

Details
R interface to the NAG Fortran routine E04KZF.

Value

b l  double array
The lower bounds actually used by e04kz.

b u  double array
The upper bounds actually used by e04kz.

x  double array
The lowest point found during the calculations of the position of the minimum.

f  double
The value of \( F(x) \) corresponding to the final point stored in \( x \).

g  double array
The value of \( \frac{\partial F}{\partial x_j} \) corresponding to the final point stored in \( x \) for \( j = 1 \ldots n \); the value of \( g(j) \) for variables not on a bound should normally be close to zero.

Author(s)
NAG

References
http://www.nag.co.uk/numeric/FL/nagdoc_fl23/pdf/E04/e04kzf.pdf

Examples

e04kz_funct2 = function(n, xc, fc, gc) {
  gc <- as.matrix(mat.or.vec(n, 1))
  list(FC = fc, GC = as.matrix(gc))
}
ibound <- 0
bl <- matrix(c(1, -2, -1e+06, 1), nrow = 4, ncol = 1,
byrow = TRUE)

bu <- matrix(c(3, 0, 1e+06, 3), nrow = 4, ncol = 1,
byrow = TRUE)

x <- matrix(c(3, -1, 0, 1), nrow = 4, ncol = 1, byrow = TRUE)
e04kz(ibound, e04kz_funct2, bl, bu, x)

e04lb

**Description**
e04lb is a comprehensive modified Newton algorithm for finding:
an unconstrained minimum of a function of several variables
a minimum of a function of several variables subject to fixed upper and/or lower bounds on the variables.

First and second derivatives are required. The function is intended for functions which have continuous first and second derivatives (although it will usually work even if the derivatives have occasional discontinuities).

**Usage**
e04lb(funct, hess, monit, ibound, bl, bu, x, lh, iw, w,
n=nrow(bl),
iprint=1,
maxcal=50,
eta=if(n ==1) 0.0 else 0.9,
xtol=0.0,
stepmx=100000.0 )

**Arguments**
funct

void function
funct must evaluate the function \( F(x) \) and its first derivatives \( \frac{\partial F}{\partial x_j} \) at any point \( x \). (However, if you do not wish to calculate \( F(x) \) or its first derivatives at a particular \( x \), there is the option of setting an argument to cause e04lb to terminate immediately.)
void function
h must calculate the second derivatives of $F$ at any point $x$. (As with funct, there is the option of causing e04lb to terminate immediately.)

void function
If $iprint \geq 0$, you must supply monit which is suitable for monitoring the minimization process. monit must not change the values of any of its arguments.

integer
Specifies whether the problem is unconstrained or bounded. If there are bounds on the variables, ibound can be used to indicate whether the facility for dealing with bounds of special forms is to be used. It must be set to one of the following values:

- $ibound = 0$: If the variables are bounded and you are supplying all the $l_j$ and $u_j$ individually.
- $ibound = 1$: If the problem is unconstrained.
- $ibound = 2$: If the variables are bounded, but all the bounds are of the form $0 \leq x_j$.
- $ibound = 3$: If all the variables are bounded, and $l_1 = l_2 = \cdots = l_n$ and $u_1 = u_2 = \cdots = u_n$.
- $ibound = 4$: If the problem is unconstrained. (The $ibound = 4$ option is provided purely for consistency with other functions. In e04lb it produces the same effect as $ibound = 1$.)

double array
The fixed lower bounds $l_j$.

double array
The fixed upper bounds $u_j$.

double array
$x(j)$ must be set to a guess at the $j$th component of the position of minimum for $j = 1 \ldots n$.

integer
The number $n$ of independent variables.

integer: default = 1
The frequency with which monit is to be called.

- $i\text{print} > 0$: monit is called once every $i\text{print}$ iterations and just before exit from e04lb.
- $i\text{print} = 0$: monit is just called at the final point.
- $i\text{print} < 0$: monit is not called at all.

integer: default = 50
The maximum permitted number of evaluations of $F(x)$, i.e., the maximum permitted number of calls of funct.

double: default = if(n==1) 0.0 else 0.9
Every iteration of e04lb involves a linear minimization (i.e., minimization of $F(x + \alpha p)$ with respect to $\alpha$). $\eta$ specifies how accurately these linear minimizations are to be performed. The minimum with respect to $\alpha$ will be located more accurately for small values of $\eta$ (say, 0.01) than for large values (say, 0.9).
xtol double: default = 0.0
The accuracy in x to which the solution is required.

stepmx double: default = 100000.0
An estimate of the Euclidean distance between the solution and the starting point supplied by you. (For maximum efficiency a slight overestimate is preferable.)

Details
R interface to the NAG Fortran routine E04LBF.

Value

bl double array
The lower bounds actually used by e04lb, e.g., if ibound = 2, bl(1) = bl(2) = \cdots = bl(n) = 0.0.

bu double array
The upper bounds actually used by e04lb, e.g., if ibound = 2, bu(1) = bu(2) = \cdots = bu(n) = 10^6.

x double array
The final point \( x^{(k)} \). Thus, if ifail = 0 on exit, \( x(j) \) is the \( j \)th component of the estimated position of the minimum.

hesl double array
During the determination of a direction \( p_z \) (see the Description in Fortran library documentation), \( H + E \) is decomposed into the product \( LDL^T \), where \( L \) is a unit lower triangular matrix and \( D \) is a diagonal matrix. (The matrices \( H, E, L \) and \( D \) are all of dimension \( n_z \), where \( n_z \) is the number of variables free from their bounds. \( H \) consists of those rows and columns of the full estimated second derivative matrix which relate to free variables. \( E \) is chosen so that \( H + E \) is positive definite.)

hesd double array
During the determination of a direction \( p_z \) (see the Description in Fortran library documentation), \( H + E \) is decomposed into the product \( LDL^T \), where \( L \) is a unit lower triangular matrix and \( D \) is a diagonal matrix. (The matrices \( H, E, L \) and \( D \) are all of dimension \( n_z \), where \( n_z \) is the number of variables free from their bounds. \( H \) consists of those rows and columns of the full second derivative matrix which relate to free variables. \( E \) is chosen so that \( H + E \) is positive definite.)

istate integer array
Information about which variables are currently on their bounds and which are free. If istrat(j) is:
- equal to \(-1\), \( x_j \) is fixed on its upper bound;
- equal to \(-2\), \( x_j \) is fixed on its lower bound;
- equal to \(-3\), \( x_j \) is effectively a constant (i.e., \( l_j = u_j \));
- positive, istrat(j) gives the position of \( x_j \) in the sequence of free variables.

f double
The function value at the final point given in x.

g double array
The first derivative vector corresponding to the final point given in x. The components of g corresponding to free variables should normally be close to zero.
Author(s)
NAG

References
http://www.nag.co.uk/numeric/FL/nagdoc_fl23/pdf/E04/e04lb.pdf

Examples

e04lb_funct = function(iflag, n, xc, fc, gc) {
  gc <- as.matrix(mat.or.vec(n, 1))
  list(IFLAG = iflag, FC = fc, GC = as.matrix(gc))
}
e04lb_hess = function(iflag, n, xc, fhesl, lh, fhesd) {
  fhesl <- as.matrix(mat.or.vec(lh, 1))
  fhesl[1] <- 20
  fhesl[2] <- 0
  fhesl[5] <- 0
  fhesl[6] <- -10
  list(IFLAG = iflag, FHESL = as.matrix(fhesl), FHESD = as.matrix(fhesd))
}
e04lb_monit = function(n, xc, fc, gc, istate, gpjnrm, cond, posdef, niter, nf) {
  sprintf("\n Itn Fn evals Fn value Norm of proj gradient\n", "\n")
  sprintf(" %d %5d %20.4f %20.4f\n", niter, nf, fc, gpjnrm, "\n")
  sprintf(" J XJ GJ Status\n", "\n")
  for (j in c(1:n)) {
    isj <- istate[j]
    if (isj > 0) {
      sprintf("%d %16.4f%20.4f %s\n", j, xc, j, gc, j, " Free", "\n")
    } else if (isj == -1) {
      }
else if (isj == -2) {
}
else if (isj == -3) {
}
}

if (cond != 0) {
if (cond > 1e+06) {
    sprintf("\nEstimated condition number of projected Hessian is more than 1.0e+\n\n")
} else {
    sprintf("\nEstimated condition number of projected Hessian = %10.2f\n", cond, "\n")
}
if (!posdef) {
    sprintf("\nProjected Hessian matrix is not positive definite\n", "\n")
}
}
list()

ibound <- 0
bl <- matrix(c(1, -2, -1e+06, 1), nrow = 4, ncol = 1, byrow = TRUE)
bu <- matrix(c(3, 0, 1e+06, 3), nrow = 4, ncol = 1, byrow = TRUE)
x <- matrix(c(3, -1, 0, 1), nrow = 4, ncol = 1, byrow = TRUE)
lh <- 6
iw <- matrix(c(0, 0), nrow = 2, ncol = 1, byrow = TRUE)
w <- as.matrix(mat.or.vec(34, 1))
e04lb(e04lb_funct, e04lb_hess, e04lb_monit, ibound, bl, bu, x, lh, iw, w)
The function e04ly is an easy-to-use modified-Newton algorithm for finding a minimum of a function, $F(x_1, x_2, \ldots, x_n)$ subject to fixed upper and lower bounds on the independent variables, $x_1, x_2, \ldots, x_n$ when first and second derivatives of $F$ are available. It is intended for functions which are continuous and which have continuous first and second derivatives (although it will usually work even if the derivatives have occasional discontinuities).

Usage

```c
void e04ly(int ibound, double (*funct2)(void, const double *, double *),
            double (*hess2)(void, const double *, double *, double *),
            const double *bl, const double *bu, double *x,
            int n);
```

Arguments

- `ibound`: integer
  Indicates whether the facility for dealing with bounds of special forms is to be used. It must be set to one of the following values:
  - `ibound = 0`: If you are supplying all the $l_j$ and $u_j$ individually.
  - `ibound = 1`: If there are no bounds on any $x_j$.
  - `ibound = 2`: If all the bounds are of the form $0 \leq x_j$.
  - `ibound = 3`: If $l_1 = l_2 = \cdots = l_n$ and $u_1 = u_2 = \cdots = u_n$.

- `funct2`: void function
  You must supply this function to calculate the values of the function $F(x)$ and its first derivatives $\frac{\partial F}{\partial x_j}$ at any point $x$. It should be tested separately before being used in conjunction with e04ly (see the E04 chapter introduction in the Fortran Library documentation).

- `hess2`: void function
  You must supply this function to evaluate the elements $H_{ij} = \frac{\partial^2 F}{\partial x_i \partial x_j}$ of the matrix of second derivatives of $F(x)$ at any point $x$. It should be tested separately before being used in conjunction with e04ly (see the E04 chapter introduction in the Fortran Library documentation).

- `bl`: double array
  The lower bounds $l_j$.

- `bu`: double array
  The upper bounds $u_j$.

- `x`: double array
  $x(j)$ must be set to a guess at the $j$th component of the position of the minimum for $j = 1 \ldots n$. The function checks the gradient and the Hessian matrix at the starting point, and is more likely to detect any error in your programming if the initial $x(j)$ are nonzero and mutually distinct.

- `n`: integer
  The number $n$ of independent variables.
Details

R interface to the NAG Fortran routine E04LYF.

Value

\[ bl \]

double array

The lower bounds actually used by e04ly.

\[ bu \]

double array

The upper bounds actually used by e04ly.

\[ x \]

double array

The lowest point found during the calculations. Thus, if ifail = 0 on exit, \( x(j) \) is the \( j \)th component of the position of the minimum.

\[ f \]

double

The value of \( F(x) \) corresponding to the final point stored in \( x \).

\[ g \]

double array

The value of \( \frac{\partial F}{\partial x_j} \) corresponding to the final point stored in \( x \) for \( j = 1 \ldots n \); the value of \( g(j) \) for variables not on a bound should normally be close to zero.

Author(s)

NAG

References

http://www.nag.co.uk/numeric/FL/nagdoc_fl23/pdf/E04/e04lyf.pdf

Examples

e04ly_funct2 = function(n, xc, fc, gc) {
  gc <- as.matrix(mat.or.vec(n, 1))
    2 * xc[3])^4 + 10 * (xc[1] - xc[4])^4
  list(FC = fc, GC = as.matrix(gc))
}
e04ly_hess2 = function(n, xc, heslc, lh, hesdc) {
  heslc <- as.matrix(mat.or.vec(lh, 1))
  hesdc <- as.matrix(mat.or.vec(n, 1))
  heslc[1] <- 20
  heslc[2] <- 0
  heslc[5] <- 0
  heslc[6] <- -10
```r
list(HESLC = as.matrix(heslc), HESDC = as.matrix(hesdc))

ibound <- 0
bl <- matrix(c(1, -2, -1e+06, 1), nrow = 4, ncol = 1,
  byrow = TRUE)

bu <- matrix(c(3, 0, 1e+06, 3), nrow = 4, ncol = 1,
  byrow = TRUE)

x <- matrix(c(3, -1, 0, 1), nrow = 4, ncol = 1, byrow = TRUE)

e04ly(ibound, e04ly_funct2, e04ly_hess2, bl, bu, x)
```

---

**e04mf**

**e04mf: LP problem (dense)**

**Description**

e04mf solves general linear programming problems. It is not intended for large sparse problems.

**Usage**

e04mf(a, bl, bu, cvec, istate, x, optlist,
  n = nrow(x),
  nclin = nrow(a))

**Arguments**

- `a` double array  
  The \( i \)th row of `a` must contain the coefficients of the \( i \)th general linear constraint for \( i = 1 \ldots m_L \).
- `bl` double array
- `bu` double array  
  Must contain the lower bounds and `bu` the upper bounds, for all the constraints in the following order. The first \( n \) elements of each array must contain the bounds on the variables, and the next \( m_L \) elements the bounds for the general linear constraints (if any). To specify a nonexistent lower bound (i.e., \( l_j = -\infty \)), set `bl[j] <= -bigbnd`, and to specify a nonexistent upper bound (i.e., \( u_j = +\infty \)), set `bu[j] >= bigbnd`; the default value of `bigbnd` is \( 10^{20} \), but this may be changed by the optional argument `infiniteboundsize`. To specify the \( j \)th constraint as an equality, set `bl[j] = bu[j] = \beta`, say, where \( \text{abs} (\beta) < \text{bigbnd} \).
- `cvec` double array  
  The coefficients of the objective function when the problem is of type LP.
- `istate` integer array  
  Need not be set if the (default) optional argument `coldstart` is used.
An initial estimate of the solution.

Optional parameters may be listed, as shown in the following table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check Frequency</td>
<td>integer</td>
<td>Default = 50</td>
</tr>
<tr>
<td>Cold Start</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Warm Start</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Crash Tolerance</td>
<td>double</td>
<td>Default = 0.01</td>
</tr>
<tr>
<td>Defaults</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Expand Frequency</td>
<td>integer</td>
<td>Default = 5</td>
</tr>
<tr>
<td>Feasibility Tolerance</td>
<td>double</td>
<td>Default = $\sqrt{\epsilon}$</td>
</tr>
<tr>
<td>Infinite Bound Size</td>
<td>double</td>
<td>Default = $10^{26}$</td>
</tr>
<tr>
<td>Infinite Step Size</td>
<td>double</td>
<td>Default = $\max(bign,10^{20})$</td>
</tr>
<tr>
<td>Iteration Limit</td>
<td>integer</td>
<td>Default = $\max(50,5(n+m))$</td>
</tr>
<tr>
<td>Iters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>List</td>
<td></td>
<td>Default for e04mf = list</td>
</tr>
<tr>
<td>Nolist</td>
<td></td>
<td>Default for e04mf = nolist</td>
</tr>
<tr>
<td>Minimum Sum of Infeasibilities</td>
<td>no</td>
<td>Default = NO</td>
</tr>
<tr>
<td>Monitoring File</td>
<td>integer</td>
<td>Default = -1</td>
</tr>
<tr>
<td>Optimality Tolerance</td>
<td>double</td>
<td>Default = $10^{-8}$</td>
</tr>
<tr>
<td>Print Level</td>
<td>integer</td>
<td>Default = 0</td>
</tr>
<tr>
<td>Problem Type</td>
<td>string</td>
<td>Default = LP</td>
</tr>
</tbody>
</table>

**Details**

R interface to the NAG Fortran routine E04MFF.

**Value**

**ISTATE**
integer array
The status of the constraints in the working set at the point returned in x. The significance of each possible value of istate[j] is as follows:

**X**
double array
The point at which e04mf terminated. If ifail = 0, ifail = 1, ifail = 4, x contains an estimate of the solution.

**ITER**
integer
The total number of iterations performed.

**OBJ**
double
The value of the objective function at x if x is feasible, or the sum of infeasibilities at x otherwise. If the problem is of type FP and x is feasible, obj is set to zero.

**n**
integer: default = nrow(x)
n, the number of variables.

**nclin**
integer: default = nrow(a)
mL, the number of general linear constraints.
AX double array
The final values of the linear constraints $Ax$.

CLAMDA double array
The values of the Lagrange multipliers for each constraint with respect to the current working set. The first $n$ elements contain the multipliers for the bound constraints on the variables, and the next $m_L$ elements contain the multipliers for the general linear constraints (if any). If $istate[j] = 0$ (i.e., constraint $j$ is not in the working set), $clamda[j]$ is zero. If $x$ is optimal, $clamda[j]$ should be non-negative if $istate[j] = 1$, non-positive if $istate[j] = 2$ and zero if $istate[j] = 4$.

IFAIL integer
ifail = 0 unless the function detects an error or a warning has been flagged (see the Errors section in Fortran library documentation).

Author(s)
NAG

References
http://www.nag.co.uk/numeric/FL/nagdoc_fl23/pdf/E04/e04mff.pdf

Examples

```r
optlist<-list()
ifail<-0
a<-matrix(c(1,1,1,1,1,1,0.15,0.04,0.02,0.04,0.02,0.01,0.03,0.03,0.05,0.08,0.02,0.06,0.01,0,0.02,0.04,0.01,0.02,0.02,0,0,0.02,0.03,0,0,0.01,0,0,0.7,0.75,0.8,0.75,0.8,0.97,0,0.02,0.06,0.08,0.12,0.02,0.01,0.97),nrow=7,ncol=7,byrow=TRUE)
bl<-matrix(c(-0.01,-0.1,-0.01,-0.04,-0.1,-0.01,-0.01,-0.13,-9.999999999999999e+24,-9.999999999999999e+24,-9.999999999999999e+24,-9.999999999999999e+24,-0.0992,-0.003),nrow=14,ncol=1,byrow=TRUE)
bu<-matrix(c(0.01,0.15,0.03,0.02,0.05,9.999999999999999e+24,9.999999999999999e+24,-0.13,-0.004900000000000001,-0.0064,-0.0037,-0.0012,9.999999999999999e+24,0.002),nrow=14,ncol=1,byrow=TRUE)
cvec<-matrix(c(-0.02,-0.2,-0.2,-0.2,-0.2,0.04,0.04),nrow=7,ncol=1,byrow=TRUE)
istate<-as.matrix(mat.or.vec(14,1))
x<-matrix(c(-0.01,-0.03,0,-0.01,-0.1,0.02,0.01),nrow=7,ncol=1,byrow=TRUE)
e04mf(a,bl,bu,cvec,istate,x,optlist)
```
Description

e04nc solves linearly constrained linear least squares problems and convex quadratic programming problems. It is not intended for large sparse problems.

Usage

```plaintext
e04nc(c, bl, bu, cvec, istate, kx, x, a, b, optlist,
    m = nrow(a),
    n = nrow(kx),
    nclin = nrow(c))
```

Arguments

c        double array
        The $i$th row of $c$ must contain the coefficients of the $i$th general constraint for $i = 1 \ldots \text{nclin}$.

bl        double array
       bu        double array
       Bl must contain the lower bounds and $bu$ the upper bounds, for all the constraints, in the following order. The first $n$ elements of each array must contain the bounds on the variables, and the next $n_L$ elements must contain the bounds for the general linear constraints (if any). To specify a nonexistent lower bound (i.e., $l_j = -\infty$), set $bl[j] \leq -\text{bigbnd}$, and to specify a nonexistent upper bound (i.e., $u_j = +\infty$), set $bu[j] \geq \text{bigbnd}$; the default value of $\text{bigbnd}$ is $10^{20}$, but this may be changed by the optional argument infiniteboundsize. To specify the $j$th constraint as an equality, set $bu[j] = bl[j] = \beta$, say, where $\text{abs}(\beta) < \text{bigbnd}$.

cvec        double array
        The coefficients of the explicit linear term of the objective function.

istate        integer array
        Need not be set if the (default) optional argument coldstart is used.

kx        integer array
        Need not be initialized for problems of type FP, LP, QP1, QP2, LS1 (the default) or LS2.

x        double array
        An initial estimate of the solution.

a        double array
        The array $a$ must contain the matrix $A$ as specified in table 1 (see the Description in Fortran library documentation).

b        double array
        The $m$ elements of the vector of observations.

optlist        options list
        Optional parameters may be listed, as shown in the following table:
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cold Start</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Warm Start</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Crash Tolerance</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Defaults</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Feasibility Phase Iteration Limit</td>
<td>integer</td>
<td>Default = max(50, 5(n + nL))</td>
</tr>
<tr>
<td>Optimality Phase Iteration Limit</td>
<td>integer</td>
<td>Default = max(50, 5(n + nL))</td>
</tr>
<tr>
<td>Feasibility Tolerance</td>
<td>double</td>
<td>Default = 0.01</td>
</tr>
<tr>
<td>Hessian</td>
<td>no</td>
<td>Default = NO</td>
</tr>
<tr>
<td>Infinite Bound Size</td>
<td>double</td>
<td>Default = 10^{20}</td>
</tr>
<tr>
<td>Infinite Step Size</td>
<td>double</td>
<td>Default = max(bigbad, 10^{20})</td>
</tr>
<tr>
<td>Iteration Limit</td>
<td>integer</td>
<td>Default = max(50, 5(n + nL))</td>
</tr>
<tr>
<td>Iters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nolist</td>
<td></td>
<td>Default for e04nc = nolist</td>
</tr>
<tr>
<td>Monitoring File</td>
<td>integer</td>
<td>Default = -1</td>
</tr>
<tr>
<td>Print Level</td>
<td>integer</td>
<td>= 0</td>
</tr>
<tr>
<td>Problem Type</td>
<td>string</td>
<td>Default = LS1</td>
</tr>
<tr>
<td>Rank Tolerance</td>
<td>double</td>
<td>Default = 100 \epsilon or 10\sqrt{\epsilon} (see below)</td>
</tr>
</tbody>
</table>

- $m$: integer; default = nrow(a)
  
  $m$, the number of rows in the matrix $A$. If the problem is specified as type FP or LP, $m$ is not referenced and is assumed to be zero.

- $n$: integer; default = nrow(kx)
  
  $n$, the number of variables.

- $nclin$: integer; default = nrow(c)
  
  $nL$, the number of general linear constraints.

Details

- R interface to the NAG Fortran routine E04NCF.

Value

- **ISTATE**: integer array
  
  The status of the constraints in the working set at the point returned in x. The significance of each possible value of $istate[j]$ is as follows:

- **KX**: integer array
  
  Defines the order of the columns of a with respect to the ordering of x, as described above.

- **X**: double array
  
  The point at which e04nc terminated. If ifail = 0, ifail = 1, ifail = 4, x contains an estimate of the solution.

- **A**: double array
  
  If $hessian = NO$ and the problem is of type LS or QP, a contains the upper triangular Cholesky factor $R$ of eqn8 (see the Fortran library documentation), with columns ordered as indicated by kx. If $hessian = YES$ and the problem is of type LS or QP, a contains the upper triangular Cholesky factor $R$ of the
Hessian matrix $H$, with columns ordered as indicated by $kx$. In either case $R$ may be used to obtain the variance-covariance matrix or to recover the upper triangular factor of the original least squares matrix.

$B$

double array

The transformed residual vector of equation eqn10 (see the Fortran library documentation).

$ITER$

integer

The total number of iterations performed.

$OBJ$

double

The value of the objective function at $x$ if $x$ is feasible, or the sum of infeasibilities at $x$ otherwise. If the problem is of type FP and $x$ is feasible, obj is set to zero.

$CLAMDA$

double array

The values of the Lagrange multipliers for each constraint with respect to the current working set. The first $n$ elements contain the multipliers for the bound constraints on the variables, and the next $nL$ elements contain the multipliers for the general linear constraints (if any). If $istate[j] = 0$ (i.e., constraint $j$ is not in the working set), clamda[$j$] is zero. If $x$ is optimal, clamda[$j$] should be non-negative if $istate[j] = 1$, non-positive if $istate[j] = 2$ and zero if $istate[j] = 4$.

$IFAIL$

integer

$ifail = 0$ unless the function detects an error or a warning has been flagged (see the Errors section in Fortran library documentation).

Author(s)

NAG

References

http://www.nag.co.uk/numeric/FL/nagdoc_fl23/pdf/E04/e04ncf.pdf

Examples

```r
optlist<-list()
ifail<-0
c<-matrix(c(1,1,1,1,1,1,4,1,2,3,4,-2,1,1,1,1,-1,1,1,1,1,1,-1,1,1,1,1,1),nrow=3,ncol=9,byrow=TRUE)
bl<-matrix(c(0,0,-9.999999999999999e+24,0,0,0,0,0,2,-9.999999999999999e+24,1),nrow=12,ncol=1,byrow=TRUE)
bu<-matrix(c(2,2,2,2,2,2,2,2,2,9.999999999999999e+24,2,4),nrow=12,ncol=1,byrow=TRUE)
cvec<-matrix(c(0),nrow=1,ncol=1,byrow=TRUE)
```
istate<-as.matrix(mat.or.vec(12,1))
kx<-as.matrix(mat.or.vec(9,1))
x<-matrix(c(1,0.5,0.3333,0.25,0.2,0.1667,0.1428,0.125,0.1111),nrow=9,ncol=1,byrow=TRUE)
a<-matrix(c(1,1,1,1,1,1,1,1,1,1,1,2,1,1,1,1,2,0,0,1,1,3,1,1,1,-1,-1,-3,1,1,4,1,1,1,1,1,1,1,1,3,1,1,1,1,1,1,1,2,1,1,0,0,0,-1,1,1,1,1,0,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,0,1,1,1,2,2,3,1,0,1,1,1,1,0,2,2),nrow=10,ncol=9,byrow=TRUE)
b<-matrix(c(1,1,1,1,1,1,1,1,1,1),nrow=10,ncol=1,byrow=TRUE)
e04nc(c,bl,bu,cvec,istate,kx,x,a,b,optlist)

e04nf

Description

e04nf solves general quadratic programming problems. It is not intended for large sparse problems.

Usage

e04nf(a, bl, bu, cvec, h, qphess, istate, x, optlist,
   n = nrow(x),
   nclin = nrow(a))

Arguments

a

double array
The $i$th row of $a$ must contain the coefficients of the $i$th general linear constraint for $i = 1 \ldots m_L$.
If $n\text{lin} = 0$, $a$ is not referenced.

bl

double array

bu

double array
Bl must contain the lower bounds and bu the upper bounds, for all the constraints in the following order. The first $n$ elements of each array must contain the bounds on the variables, and the next $m_L$ elements the bounds for the general linear constraints (if any). To specify a nonexistent lower bound (i.e., $l_j = -\infty$), set $bl[j] \leq -\text{bigbnd}$, and to specify a nonexistent upper bound (i.e., $u_j = +\infty$), set $bu[j] \geq \text{bigbnd}$; the default value of $\text{bigbnd}$ is $10^{20}$, but this may be changed by the optional argument infiniteboundsize. To specify the $j$th constraint as an equality, set $bl[j] = bu[j] = \beta$, say, where $\text{abs}(\beta) < \text{bigbnd}$.
cvec double array
The coefficients of the explicit linear term of the objective function when the problem is of type LP, QP2 (the default) and QP4.
If the problem is of type FP, QP1, or QP3, cvec is not referenced.

h double array
May be used to store the quadratic term $H$ of the QP objective function if desired. In some cases, you need not use h to store $H$ explicitly (see the specification of function qphess). The elements of h are referenced only by function qphess. The number of rows of $H$ is denoted by $m$, whose default value is $n$. (The optional argument hessianrows may be used to specify a value of $m < n$.)

double array
May be used to store the quadratic term $H$ of the QP objective function if desired. In some cases, you need not use h to store $H$ explicitly (see the specification of function qphess). The elements of h are referenced only by function qphess. The number of rows of $H$ is denoted by $m$, whose default value is $n$. (The optional argument hessianrows may be used to specify a value of $m < n$.)

qphess function
In general, you need not provide a version of qphess, because a ‘default’ function with name e04nfu is included in the Library. However, the algorithm of e04nf requires only the product of $H$ or $H^T H$ and a vector $x$; and in some cases you may obtain increased efficiency by providing a version of qphess that avoids the need to define the elements of the matrices $H$ or $H^T H$ explicitly.

$$(HX, IWSAV) = qphess(n, jthcol, h, x, iwsav)$$

istate integer array
Need not be set if the (default) optional argument coldstart is used. If the optional argument warmstart has been chosen, istate specifies the desired status of the constraints at the start of the feasibility phase. More precisely, the first $n$ elements of istate refer to the upper and lower bounds on the variables, and the next $m L$ elements refer to the general linear constraints (if any). Possible values for istate$[j]$ are as follows:

x double array
An initial estimate of the solution.

optlist options list
Optional parameters may be listed, as shown in the following table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check Frequency</td>
<td>double</td>
<td>Default  = 50</td>
</tr>
<tr>
<td>Cold Start</td>
<td></td>
<td>Default</td>
</tr>
<tr>
<td>Warm Start</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Crash Tolerance</td>
<td>double</td>
<td>Default  = 0.01</td>
</tr>
<tr>
<td>Defaults</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Expand Frequency</td>
<td>integer</td>
<td>Default  = 5</td>
</tr>
<tr>
<td>Feasibility Phase Iteration Limit</td>
<td>integer</td>
<td>$\text{Default} = \max(50, 5 (n + m_L))$</td>
</tr>
<tr>
<td>Optimality Phase Iteration Limit</td>
<td>integer</td>
<td>$\text{Default} = \max(50, 5 (n + m_L))$</td>
</tr>
<tr>
<td>Feasibility Tolerance</td>
<td>double</td>
<td>$\text{Default} = \sqrt{\epsilon}$</td>
</tr>
<tr>
<td>Hessian Rows</td>
<td>integer</td>
<td>Default  = $n$</td>
</tr>
<tr>
<td>Infinite Bound Size</td>
<td>double</td>
<td>Default  = $10^{20}$</td>
</tr>
<tr>
<td>Infinite Step Size</td>
<td>double</td>
<td>$\text{Default} = \max(bigbnd, 10^{20})$</td>
</tr>
<tr>
<td>Iteration Limit</td>
<td>integer</td>
<td>$\text{Default} = \max(50, 5 (n + m_L))$</td>
</tr>
</tbody>
</table>
### Iters
- **Default for e04nf = list**

### Ittns
- **Default for e04nf = nolist**

### List
- **Default for e04nf = list**

### Nolist
- **Default for e04nf = nolist**

### Maximum Degrees of Freedom
- **Integer**
- **Default = n**

### Minimum Sum of Infeasibilities
- **String**
- **Default = NO**

### Monitoring File
- **Integer**
- **Default = -1**

### Optimality Tolerance
- **Double**
- **Default = $\epsilon^{0.5}$**

### Print Level
- **Integer**
- **Default = 0**

### Problem Type
- **String**
- **Default = QP2**

### Rank Tolerance
- **Double**
- **Default = 100\(\epsilon\)**

---

### n
- **Integer**: default = nrow(x)
- \(n\), the number of variables.

### nclin
- **Integer**: default = nrow(a)
- \(m_L\), the number of general linear constraints.

**Details**

R interface to the NAG Fortran routine E04NFF.

#### Value

- **ISTATE**
  - Integer array
  - The status of the constraints in the working set at the point returned in x. The significance of each possible value of istate\([j]\) is as follows:

- **X**
  - Double array
  - The point at which e04nf terminated. If ifail = 0, ifail = 1, ifail = 4, x contains an estimate of the solution.

- **ITER**
  - Integer
  - The total number of iterations performed.

- **OBJ**
  - Double
  - The value of the objective function at \(x\) if \(x\) is feasible, or the sum of infeasibilities at \(x\) otherwise. If the problem is of type FP and \(x\) is feasible, obj is set to zero.

- **AX**
  - Double array
  - The final values of the linear constraints \(Ax\).
  - If nclin = 0, ax is not referenced.

- **CLAMDA**
  - Double array
  - The values of the Lagrange multipliers for each constraint with respect to the current working set. The first \(n\) elements contain the multipliers for the bound constraints on the variables, and the next \(m_L\) elements contain the multipliers for the general linear constraints (if any). If istate\([j]\) = 0 (i.e., constraint \(j\) is not in the working set), clamda\([j]\) is zero. If \(x\) is optimal, clamda\([j]\) should be non-negative if istate\([j]\) = 1, non-positive if istate\([j]\) = 2 and zero if istate\([j]\) = 4.

- **IFAIL**
  - Integer
  - Ifail = 0 unless the function detects an error or a warning has been flagged (see the Errors section in Fortran library documentation).
Examples

```r
optlist <- list()
ifail <- 0
qphess = function(n, jthcol, h, x, iwsav) {
  ldh <- nrow(h)

  if (iwsav[365] == 3 || iwsav[365] == 4) {
    hx <- h %*% x
  } else if (iwsav[365] == 5 || iwsav[365] == 6) {
    hx <- t(h) %*% h %*% x
  } else {
    hx <- as.matrix(mat.or.vec(n, 1))
  }
  list(HX = as.matrix(hx), IWSAV = as.matrix(iwsav))
}
a <- matrix(c(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0.15, 0.04, 0.02, 0.04, 0.02, 0.01, 0.03, 0.03, 0.05, 0.08, 0.02, 0.06, 0.01, 0, 0.02, 0.04, 0.01, 0.02, 0.02, 0, 0, 0.02, 0.03, 0, 0, 0.01, 0, 0, 0.7, 0.75, 0.8, 0.75, 0.8, 0.97, 0, 0.02, 0.06, 0.08, 0.12, 0.02, 0.01, 0.07), nrow = 7, ncol = 7, byrow = TRUE)

bl <- matrix(c(-0.01, -0.1, -0.01, -0.04, -0.1, -0.01, -0.01, -0.13, -1e+25, -1e+25, -1e+25, -1e+25, -0.0992, -0.003), nrow = 14, ncol = 1, byrow = TRUE)

bu <- matrix(c(0.01, 0.15, 0.03, 0.02, 0.05, 1e+25, 1e+25, -0.13, -0.0049, -0.0064, -0.0037, -0.0012, 1e+25, 0.002), nrow = 14, ncol = 1, byrow = TRUE)

cvec <- matrix(c(-0.02, -0.2, -0.2, -0.2, -0.2, 0.04, 0.04), nrow = 7, ncol = 1, byrow = TRUE)
```
e04nk

Description

e04nk solves sparse linear programming or quadratic programming problems.

Usage

e04nk(n, m, iobj, ncolh, qphx, a, ha, ka, bl, bu, start, names, crname, ns, xs, 
  nnz = nrow(a), 
  nname = nrow(crname))

Arguments

n  integer
  n, the number of variables (excluding slacks). This is the number of columns in the linear constraint matrix A.

m  integer
  m, the number of general linear constraints (or slacks). This is the number of rows in A, including the free row (if any; see iobj).

iobj  integer
  If iobj > 0, row iobj of A is a free row containing the nonzero elements of the vector c appearing in the linear objective term c^T x.

ncolh  integer
  nH, the number of leading nonzero columns of the Hessian matrix H. For FP and LP problems, ncolh must be set to zero.

qphx  function
  For QP problems, you must supply a version of qphx to compute the matrix product H x. If H has zero rows and columns, it is most efficient to order the variables x = (y z)^T so that

  \[ H x = \begin{pmatrix} H_1 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} y \\ z \end{pmatrix} = \begin{pmatrix} H_1 y \\ 0 \end{pmatrix}, \]
where the nonlinear variables \( y \) appear first as shown. For FP and LP problems, qphx will never be called by e04nk and hence qphx may be the dummy function e04nku.

\[
(HX) = \text{qphx}(\text{nstate}, \text{ncolh}, x)
\]

\( a \)

double array

The nonzero elements of \( A \), ordered by increasing column index. Note that elements with the same row and column indices are not allowed.

\( ha \)

integer array

\( ha[i] \) must contain the row index of the nonzero element stored in \( a[i] \) for \( i = 1 \ldots \text{nnz} \). Note that the row indices for a column may be supplied in any order.

\( ka \)

integer array

\( ka[j] \) must contain the index in \( a \) of the start of the \( j \)th column for \( j = 1 \ldots n \). To specify the \( j \)th column as empty, set \( ka[j] = ka[j + 1] \). Note that the first and last elements of \( ka \) must be such that \( ka[1] = 1 \) and \( ka[n + 1] = \text{nnz} + 1 \).

\( bl \)

double array

\( l \), the lower bounds for all the variables and general constraints, in the following order. The first \( n \) elements of \( bl \) must contain the bounds on the variables \( x \), and the next \( m \) elements the bounds for the general linear constraints \( Ax \) (or slacks \( s \)) and the free row (if any). To specify a nonexistent lower bound (i.e., \( l_j = -\infty \)), set \( bl[j] \leq -\text{bigbnd} \), where \( \text{bigbnd} \) is the value of the optional argument infiniteboundsize. To specify the \( j \)th constraint as an equality, set \( bl[j] = bu[j] = \beta \), say, where \( \text{abs}(\beta) < \text{bigbnd} \). Note that the lower bound corresponding to the free row must be set to \(-\infty \) and stored in \( bl[n + iobj] \).

\( bu \)

double array

\( u \), the upper bounds for all the variables and general constraints, in the following order. The first \( n \) elements of \( bu \) must contain the bounds on the variables \( x \), and the next \( m \) elements the bounds for the general linear constraints \( Ax \) (or slacks \( s \)) and the free row (if any). To specify a nonexistent upper bound (i.e., \( u_j = +\infty \)), set \( bu[j] \geq \text{bigbnd} \). Note that the upper bound corresponding to the free row must be set to \(+\infty \) and stored in \( bu[n + iobj] \).

\( \text{start} \)

string

Indicates how a starting basis is to be obtained.

\( \text{start} = 'C' \): An internal Crash procedure will be used to choose an initial basis matrix \( B \).

\( \text{start} = 'W' \): A basis is already defined in istate (probably from a previous call).

\( \text{names} \)

string array

A set of names associated with the so-called MPSX form of the problem, as follows:

\( \text{names}[1] \): Must contain the name for the problem (or be blank).

\( \text{names}[2] \): Must contain the name for the free row (or be blank).

\( \text{names}[3] \): Must contain the name for the constraint right-hand side (or be blank).

\( \text{names}[4] \): Must contain the name for the ranges (or be blank).

\( \text{names}[5] \): Must contain the name for the bounds (or be blank).

\( \text{crname} \)

string array

The optional column and row names, respectively.
ns
integer

\( n_S \), the number of superbasics. For QP problems, \( n_S \) need not be specified if \( \text{start} = 'C' \), but must retain its value from a previous call when \( \text{start} = 'W' \). For FP and LP problems, \( n_S \) need not be initialized.

xs
double array

The initial values of the variables and slacks \( (xs) \). (See the description for istate.)

istate
integer array

If \( \text{start} = 'C' \), the first \( n \) elements of istate and xs must specify the initial states and values, respectively, of the variables \( x \). (The slacks \( s \) need not be initialized.) An internal Crash procedure is then used to select an initial basis matrix \( B \). The initial basis matrix will be triangular (neglecting certain small elements in each column). It is chosen from various rows and columns of \( (A - I) \). Possible values for \( \text{istate}[j] \) are as follows:

leniz
integer

lenz
integer

optlist
options list

Optional parameters may be listed, as shown in the following table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check Frequency</td>
<td>integer</td>
<td>Default = 60</td>
</tr>
<tr>
<td>Crash Option</td>
<td>integer</td>
<td>Default = 2</td>
</tr>
<tr>
<td>Crash Tolerance</td>
<td>double</td>
<td>Default = 0.1</td>
</tr>
<tr>
<td>Defaults</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Expand Frequency</td>
<td>integer</td>
<td>Default = 10000</td>
</tr>
<tr>
<td>Factorization Frequency</td>
<td>integer</td>
<td>Default = 100</td>
</tr>
<tr>
<td>Feasibility Tolerance</td>
<td>double</td>
<td>Default = ( \max(10^{-6}, \sqrt{\varepsilon}) )</td>
</tr>
<tr>
<td>Infinite Bound Size</td>
<td>double</td>
<td>Default = ( 10^{20} )</td>
</tr>
<tr>
<td>Infinite Step Size</td>
<td>double</td>
<td>Default = ( \max(bigbnd, 10^{20}) )</td>
</tr>
<tr>
<td>Iteration Limit</td>
<td>integer</td>
<td>Default = ( \max(50, 5(n + m)) )</td>
</tr>
<tr>
<td>Iters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>List</td>
<td></td>
<td>Default for e04nk = list</td>
</tr>
<tr>
<td>Nolist</td>
<td></td>
<td>Default for e04nk = nolist</td>
</tr>
<tr>
<td>LU Factor Tolerance</td>
<td>double</td>
<td>Default = 100.0</td>
</tr>
<tr>
<td>LU Update Tolerance</td>
<td>double</td>
<td>Default = 10.0</td>
</tr>
<tr>
<td>LU Singuarity Tolerance</td>
<td>double</td>
<td>Default = ( e^{0.67} )</td>
</tr>
<tr>
<td>Minimize</td>
<td></td>
<td>Default</td>
</tr>
<tr>
<td>Maximize</td>
<td></td>
<td>Default</td>
</tr>
<tr>
<td>Monitoring File</td>
<td>integer</td>
<td>Default = -1</td>
</tr>
<tr>
<td>Optimality Tolerance</td>
<td>double</td>
<td>Default = ( \max(10^{-6}, \sqrt{\varepsilon}) )</td>
</tr>
<tr>
<td>Partial Price</td>
<td>integer</td>
<td>Default = 10</td>
</tr>
<tr>
<td>Pivot Tolerance</td>
<td>double</td>
<td>Default = ( e^{0.67} )</td>
</tr>
<tr>
<td>Print Level</td>
<td>integer</td>
<td>( = 0 )</td>
</tr>
<tr>
<td>Rank Tolerance</td>
<td>double</td>
<td>Default = ( 100\varepsilon )</td>
</tr>
<tr>
<td>Scale Option</td>
<td>integer</td>
<td>Default = 2</td>
</tr>
<tr>
<td>Scale Tolerance</td>
<td>double</td>
<td>Default = 0.9</td>
</tr>
<tr>
<td>Superbasics Limit</td>
<td>integer</td>
<td>Default = ( \min(n_H + 1, n) )</td>
</tr>
</tbody>
</table>
nnz  integer: default = nrow(a)
The number of nonzero elements in A.

nname  integer: default = nrow(cname)
The number of column (i.e., variable) and row names supplied incname.
nname = 1: There are no names. Default names will be used in the printed output.
nname = n + m: All names must be supplied.

Details
R interface to the NAG Fortran routine E04NKF.

Value

NS  integer
The final number of superbasics. This will be zero for FP and LP problems.

XS  double array
The final values of the variables and slacks (xs).

ISTATE  integer array
The final states of the variables and slacks (xs). The significance of each possible value of istate[j] is as follows:

MINIZ  integer
The minimum value of leniz required to start solving the problem. If ifail = 12, e04nk may be called again with leniz suitably larger than miniz. (The bigger the better, since it is not certain how much workspace the basis factors need.)

MINZ  integer
The minimum value of lenz required to start solving the problem. If ifail = 13, e04nk may be called again with lenz suitably larger than minz. (The bigger the better, since it is not certain how much workspace the basis factors need.)

NINF  integer
The number of infeasibilities. This will be zero if ifail = 0, ifail = 1.

SINF  double
The sum of infeasibilities. This will be zero if ninf = 0. (Note that e04nk does not attempt to compute the minimum value of sinf if ifail = 3.)

OBJ  double
The value of the objective function.

CLAMDA  double array
A set of Lagrange multipliers for the bounds on the variables and the general constraints. More precisely, the first n elements contain the multipliers (reduced costs) for the bounds on the variables, and the next m elements contain the multipliers (shadow prices) for the general linear constraints.

IFAIL  integer
ifail = 0 unless the function detects an error or a warning has been flagged (see the Errors section in Fortran library documentation).

Author(s)
NAG
References

http://www.nag.co.uk/numeric/FL/nagdoc_fl23/pdf/E04/e04nkf.pdf

Examples

```r
optlist <- list()
ifail <- 0
qphx = function(nstate, ncolh, x) {
  hx <- as.matrix(mat.or.vec(ncolh, 1))
  list(HX = as.matrix(hx))
}

n <- 7
m <- 8
iobj <- 8
ncolh <- 7

a <- matrix(c(0.02, 0.02, 0.03, 1, 0.7, 0.02, 0.15,
              -200, 0.06, 0.75, 0.03, 0.04, 0.05, 0.04, 1, -2000, 0.02,
              1, 0.01, 0.08, 0.08, 0.8, -2000, 1, 0.12, 0.02, 0.02, 0.75,
              0.04, -2000, 0.01, 0.8, 0.02, 1, 0.02, 0.06, 0.02, -2000,
              1, 0.01, 0.01, 0.97, 0.01, 400, 0.97, 0.03, 1, 400), nrow = 48,
              ncol = 1, byrow = TRUE)

ha <- matrix(c(7, 5, 3, 1, 6, 4, 2, 8, 7, 6, 5, 4,
               3, 2, 1, 8, 2, 1, 4, 3, 7, 6, 8, 1, 7, 3, 4, 6, 2, 8, 5,
               6, 7, 1, 2, 3, 4, 8, 1, 2, 3, 6, 7, 8, 7, 2, 1, 8), nrow = 48,
               ncol = 1, byrow = TRUE)

ka <- matrix(c(1, 9, 17, 24, 31, 39, 45, 49), nrow = 8,
             ncol = 1, byrow = TRUE)

bl <- matrix(c(0, 0, 400, 100, 0, 0, 0, 2000, -1e+25,
              -1e+25, -1e+25, -1e+25, 1500, 250, -1e+25), nrow = 15,
             ncol = 1, byrow = TRUE)
```
bu <- matrix(c(200, 2500, 800, 700, 1500, 1e+25, 1e+25, 
              2000, 60, 100, 40, 30, 1e+25, 300, 1e+25), nrow = 15, ncol = 1, 
              byrow = TRUE)

start <- "C"

names <- matrix(c(" ", " ", " ", " ", " ", " ", " ", " "), nrow = 5, byrow = TRUE)

curname <- matrix(c("...X1...", "...X2...", "...X3...", 
                   "...X4...", "...X5...", "...X6...", "...X7...", "..ROW1..", 
                   "..ROW2..", "..ROW3..", "..ROW4..", "..ROW5..", "..ROW6..", 
                   "..ROW7..", "..COST.."), nrow = 15, byrow = TRUE)

ns <- -1232765364

xs <- matrix(c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0), nrow = 15, ncol = 1, byrow = TRUE)

istate <- as.matrix(mat.or.vec(15, 1))

leniz <- 10000

lenz <- 10000

ans <- e04nk(n, m, iobj, ncolh, qphx, a, ha, ka, bl, 
bu, start, names, curname, ns, xs, istate, leniz, lenz, optlist)
ans

---

**e04nq**

**e04nq: LP or QP problem (suitable for sparse problems)**

**Description**

e04nq solves sparse linear programming or convex quadratic programming problems. The initialization function e04np must have been called before calling e04nq.

**Usage**

e04nq(start, qphx, m, n, lenc, ncolh, iobj, objadd, prob, acol, inda, loca, bl, 
     bu, start, names, curname, ns, xs, istate, leniz, lenz, optlist)

     ne = nrow(acol),
     nname = nrow(names))
Arguments

**start**
string
Indicates how a starting basis (and certain other items) will be obtained.

- **start = 'C'**: Requests that an internal Crash procedure be used to choose an initial basis, unless a Basis file is provided via optional arguments oldbasisfile, insertfile or loadfile.
- **start = 'B'**: Is the same as **start = 'C'** but is more meaningful when a Basis file is given.
- **start = 'W'**: Means that a basis is already defined in hs and a start point is already defined in x (probably from an earlier call).

**qphx**
function
For QP problems, you must supply a version of qphx to compute the matrix product $Hx$ for a given vector $x$. If $H$ has rows and columns of zeros, it is most efficient to order $x$ so that the nonlinear variables appear first. For example, if $x = (yz)^T$ and only $y$ enters the objective quadratically then

$$Hx = \begin{pmatrix} H_1 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} y \\ z \end{pmatrix} = \begin{pmatrix} H_1y \\ 0 \end{pmatrix}.$$ 

In this case, ncolh should be the dimension of $y$, and qphx should compute $H_1y$.

For FP and LP problems, qphx will never be called by e04nq and hence qphx may be the dummy function e04nsh.

$$\text{(HX)} = \text{qphx}(\text{ncolh}, x, \text{nstate})$$

**m**
integer
$m$, the number of general linear constraints (or slacks). This is the number of rows in the linear constraint matrix $A$, including the free row (if any; see iobj).

Note that $A$ must have at least one row. If your problem has no constraints, or only upper or lower bounds on the variables, then you must include a dummy row with sufficiently wide upper and lower bounds (see also acol, inda and loca).

**n**
integer
$n$, the number of variables (excluding slacks). This is the number of columns in the linear constraint matrix $A$.

**lenc**
integer
The number of elements in the constant objective vector $c$.

**ncolh**
integer
$nl$, the number of leading nonzero columns of the Hessian matrix $H$. For FP and LP problems, ncolh must be set to zero.

**iobj**
integer
If $iobj > 0$, row iobj of $A$ is a free row containing the nonzero elements of the vector $c$ appearing in the linear objective term $c^T x$.

**objadd**
double
The constant $q$, to be added to the objective for printing purposes. Typically $\text{objadd} = 0.0E0$.

**prob**
string
The name for the problem. It is used in the printed solution and in some functions that output Basis files. A blank name may be used.

**acol**
double array
The nonzero elements of $A$, ordered by increasing column index. Note that all elements must be assigned a value in the calling program.
inda

inda[i] must contain the row index of the nonzero element stored in acol[i] for 
i = 1 \ldots ne. Thus a pair of values (acol[i],inda[i]) contains a matrix element 
and its corresponding row index.

loca

loca[j] must contain the index in acol and inda of the start of the 
\( j \)th column for 
\( j = 1 \ldots n \). Thus for \( j = 1 : n \), the entries of column \( j \) are held in acol[k : l] 
and their corresponding row indices are in inda[k : l], where \( k = loca[j] \) and 
\( l = loca[j+1]−1 \). To specify the \( j \)th column as empty, set loca[j] = loca[j+1].

Note that the first and last elements of loca must be 
loca[1] = 1 and loca[n+1] = ne + 1. If your problem has no constraints, or just bounds on the variables, 
you may include a dummy 'free' row with a single (zero) element by setting 
ne = 1, acol[1] = 0.0, inda[1] = 1, loca[1] = 1, and loca[j] = 2, for \( j = 2 : n + 1 \). This row is made 'free' by setting its bounds to be 
\( bl[n+1] = -\text{bigbnd} \) and \( bu[n+1] = \text{bigbnd} \), where \text{bigbnd} is the value of the optional argument 
infiniteboundsize.

bl

dl, the lower bounds for all the variables and general constraints, in the following 
order. The first \( n \) elements of bl must contain the bounds on the variables \( x \), and 
the next m elements the bounds for the general linear constraints \( Ax \) (which, 
equivalently, are the bounds for the slacks, \( s \)) and the free row (if any). To 
fix the \( j \)th variable, set \( bl[j] = bu[j] = \beta \), say, where \( \abs(\beta) < \text{bigbnd} \). To 

specify a nonexistent lower bound (i.e., \( l_j = -\infty \)), set \( bl[j] \leq -\text{bigbnd} \). Here, 
\text{bigbnd} is the value of the optional argument infiniteboundsize. To specify the 
\( j \)th constraint as an equality, set \( bl[n+j] = bu[n+j] = \beta \), say, where \( \abs(\beta) < \text{bigbnd} \). Note that the lower bound corresponding to the free row must be set to 
\( -\infty \) and stored in \( bl[n + iobj] \).

bu

dl, the upper bounds for all the variables and general constraints, in the following 
order. The first \( n \) elements of bu must contain the bounds on the variables \( x \), and 
the next m elements the bounds for the general linear constraints \( Ax \) (which, 
equivalently, are the bounds for the slacks, \( s \)) and the free row (if any). To 

specify a nonexistent upper bound (i.e., \( u_j = +\infty \)), set \( bu[j] \geq \text{bigbnd} \). Note that the upper bound corresponding to the free row must be set to \( +\infty \) and stored in \( bu[n + iobj] \).

c

double array

Contains the explicit objective vector \( c \) (if any). If the problem is of type FP, or 
if \( lenc = 0 \), then \( c \) is not referenced. (In that case, \( c \) may be dimensioned eqn1, 
or it could be any convenient array.)

double array

Contains the explicit objective vector \( c \) (if any). If the problem is of type FP, or 
if \( lenc = 0 \), then \( c \) is not referenced. (In that case, \( c \) may be dimensioned eqn1, 
or it could be any convenient array.)

names

The optional column and row names, respectively.

helast

integer array

Defines which variables are to be treated as being elastic in elastic mode. The 
allowed values of helast are: helast need not be assigned if optional argument 
elasticmode = 0.
hs  integer array
If \( \text{start} = \text{C}' \), \( \text{B}' \), and a Basis file of some sort is to be input (see the description of the optional arguments oldbasisfile, insertfile or loadfile), then \( hs \) and \( x \) need not be set at all.

\( x \)  double array
The initial values of the variables \( x \), and, if \( \text{start} = \text{W}' \), the slacks \( s \), i.e., \( (xs) \).
(See the description for argument \( hs \).)

\( ns \)  integer
\( n_S \), the number of superbasics. For QP problems, \( ns \) need not be specified if \( \text{start} = \text{C}' \), but must retain its value from a previous call when \( \text{start} = \text{W}' \). For FP and LP problems, \( ns \) need not be initialized.

optlist  options list
Optional parameters may be listed, as shown in the following table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check Frequency</td>
<td>integer</td>
<td>Default = 60</td>
</tr>
<tr>
<td>Crash Option</td>
<td>integer</td>
<td>Default = 3</td>
</tr>
<tr>
<td>Crash Tolerance</td>
<td>double</td>
<td>Default = 0.1</td>
</tr>
<tr>
<td>Defaults</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dump File</td>
<td>integer</td>
<td>Default = 0</td>
</tr>
<tr>
<td>Load File</td>
<td>integer</td>
<td>Default = 0</td>
</tr>
<tr>
<td>Elastic Mode</td>
<td>integer</td>
<td>Default = 1</td>
</tr>
<tr>
<td>Elastic Objective</td>
<td>integer</td>
<td>Default = 1</td>
</tr>
<tr>
<td>Elastic Weight</td>
<td>double</td>
<td>Default = 1.0</td>
</tr>
<tr>
<td>Expand Frequency</td>
<td>integer</td>
<td>Default = 10000</td>
</tr>
<tr>
<td>Factorization Frequency</td>
<td>integer</td>
<td>Default = 100 (( LP )) or 50 (( QP ))</td>
</tr>
<tr>
<td>Feasibility Tolerance</td>
<td>double</td>
<td>Default = ( \max{10^{-6} \sqrt{\epsilon}} )</td>
</tr>
<tr>
<td>Infinite Bound Size</td>
<td>double</td>
<td>Default = ( 10^{20} )</td>
</tr>
<tr>
<td>Iterations Limit</td>
<td>integer</td>
<td>Default = ( \max{100000 \max{nn}} )</td>
</tr>
<tr>
<td>LU Density Tolerance</td>
<td>double</td>
<td>Default = 0.6</td>
</tr>
<tr>
<td>LU Singularity Tolerance</td>
<td>double</td>
<td>Default = ( \epsilon^{2} )</td>
</tr>
<tr>
<td>LU Factor Tolerance</td>
<td>double</td>
<td>Default = 100.0</td>
</tr>
<tr>
<td>LU Update Tolerance</td>
<td>double</td>
<td>Default = 10.0</td>
</tr>
<tr>
<td>LU Partial Pivoting</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LU Rook Pivoting</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minimize</td>
<td></td>
<td>Default</td>
</tr>
<tr>
<td>Maximize</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Feasible Point</td>
<td></td>
<td></td>
</tr>
<tr>
<td>New Basis File</td>
<td>integer</td>
<td>Default = 0</td>
</tr>
<tr>
<td>Backup Basis File</td>
<td>integer</td>
<td>Default = 0</td>
</tr>
<tr>
<td>Save Frequency</td>
<td>integer</td>
<td>Default = 100</td>
</tr>
<tr>
<td>Nolist</td>
<td></td>
<td>Default</td>
</tr>
<tr>
<td>List</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Old Basis File</td>
<td>integer</td>
<td>Default = 0</td>
</tr>
<tr>
<td>Optimality Tolerance</td>
<td>double</td>
<td>Default = ( \max{10^{-6} \sqrt{\epsilon}} )</td>
</tr>
<tr>
<td>Partial Price</td>
<td>integer</td>
<td>Default = 10 (( LP )) or 1 (( QP ))</td>
</tr>
<tr>
<td>Pivot Tolerance</td>
<td>double</td>
<td>Default = ( \epsilon^{2} )</td>
</tr>
<tr>
<td>Print File</td>
<td>integer</td>
<td>Default = 0</td>
</tr>
<tr>
<td>Print Frequency</td>
<td>integer</td>
<td>Default = 100</td>
</tr>
<tr>
<td>Print Level</td>
<td>integer</td>
<td>Default = 1</td>
</tr>
</tbody>
</table>
Punch File  
integer  Default = 0

Insert File  
integer  Default = 0

QPSolver Cholesky
Default

QPSolver CG

QPSolver QN

Reduced Hessian Dimension  
integer  Default = 1 (LP) or min (2000\(n_H + 1\)) (QP)

Scale Option  
integer  Default = 2

Scale Tolerance  
double  Default = 0.9

Scale Print

Solution File  
integer  Default = 0

Summary File  
integer  Default = 0

Summary Frequency  
integer  Default = 100

Superbasics Limit  
integer  Default = 1 (LP) or min \(n_H + 1\) (QP)

Suppress Parameters

System Information No
Default

System Information Yes

Timing Level  
integer  Default = 0

Unbounded Step Size  
double  Default = inf\(bnd\)

e
integer: default = nrow(acol)
The number of nonzero elements in A.

nname  
integer: default = nrow(names)
The number of column (i.e., variable) and row names supplied in the array names.

\(nname = 1\): There are no names. Default names will be used in the printed output.

\(nname = n + m\): All names must be supplied.

Details

R interface to the NAG Fortran routine E04NQF.

Value

HS
integer array
The final states of the variables and slacks \((xs)\). The significance of each possible value of \(hs[j]\) is as follows:

X
double array
The final values of the variables and slacks \((xs)\).

PI
double array
Contains the dual variables \(\pi\) (a set of Lagrange multipliers (shadow prices) for the general constraints).

RC
double array
Contains the reduced costs, \(g = (A - I)^T \pi\). The vector \(g\) is the gradient of the objective if \(x\) is feasible, otherwise it is the gradient of the Phase 1 objective. In the former case, \(g (i) = 0\), for \(i = n + 1 : m\), hence \(rc (n + 1 : m) = \pi\).

NS
integer
The final number of superbasics. This will be zero for FP and LP problems.
NINF integer
The number of infeasibilities.

SINF double
The sum of the scaled infeasibilities. This will be zero if \( ninf = 0 \), and is most meaningful when \( scaleopt = 0 \).

OBJ double
The value of the objective function.

IFAIL integer
\( ifail = 0 \) unless the function detects an error or a warning has been flagged (see the Errors section in Fortran library documentation).

Author(s)
NAG

References
http://www.nag.co.uk/numeric/FL/nagdoc_fl23/pdf/E04/e04nqf.pdf

Examples

```r
optlist<-list()
ifail<-0
qphx=function(ncolh,x,nstate){

hx<-as.matrix(mat.or.vec(ncolh,1))
list(HX=as.matrix(hx))
}

start<-'C'
m<-8
n<-7
lenc<-0
ncolh<-7
iobj<-8
objadd<-0
prob<-'','
e04uc

e04uc: Minimum, function of several variables, sequential QP method, nonlinear constraints, using function values and optionally first derivatives (comprehensive)

Description

e04uc is designed to minimize an arbitrary smooth function subject to constraints (which may include simple bounds on the variables, linear constraints and smooth nonlinear constraints) using
a sequential quadratic programming (SQP) method. As many first derivatives as possible should be supplied by you; any unspecified derivatives are approximated by finite differences. It is not intended for large sparse problems.

e04uc may also be used for unconstrained, bound-constrained and linearly constrained optimization.

e04uc uses forward communication for evaluating the objective function, the nonlinear constraint functions, and any of their derivatives.

Usage

e04uc(a, bl, bu, confun, objfun, istate, cjac, clamda, r, x, optlist,
  n = nrow(x),
  nclin = nrow(a),
  ncnln = nrow(cjac))

Arguments

da

    double array
    The i-th row of a contains the i-th row of the matrix \( A_L \) of general linear constraints in eqn1. That is, the i-th row contains the coefficients of the i-th general linear constraint for \( i = 1 \ldots nlin \).

bl

    double array
    bu

    double array
    Bl must contain the lower bounds and bu the upper bounds for all the constraints in the following order. The first \( n \) elements of each array must contain the bounds on the variables, the next \( n_L \) elements the bounds for the general linear constraints (if any) and the next \( n_N \) elements the bounds for the general nonlinear constraints (if any). To specify a nonexistent lower bound (i.e., \( l_j = -\infty \)), set \( bl[j] \leq -\text{bigbnd} \), and to specify a nonexistent upper bound (i.e., \( u_j = +\infty \)), set \( bu[j] \geq \text{bigbnd} \); the default value of \( \text{bigbnd} \) is 1.0e20, but this may be changed by the optional argument infiniteboundsize. To specify the j-th constraint as an equality, set \( bl[j] = bu[j] = \beta \), say, where abs(\( \beta \)) < \( \text{bigbnd} \).

cFUN

    function
    confun must calculate the vector \( c(x) \) of nonlinear constraint functions and (optionally) its Jacobian (\( \frac{\partial c}{\partial x} \)) for a specified \( n \) element vector \( x \). If there are no nonlinear constraints (i.e., \( ncnln = 0 \)), confun will never be called by e04uc and confun may be the dummy function e04udm. (e04udm is included in the NAG Library.) If there are nonlinear constraints, the first call to confun will occur before the first call to objfun.

\[(\text{MODE,C,CJAC}) = \text{confun}(\text{mode},n,\text{ncnln},\text{n,needc,x,cjac,nstate})\]

objfun

    function
    objfun must calculate the objective function \( F(x) \) and (optionally) its gradient \( g(x) = \frac{\partial F}{\partial x} \) for a specified \( n \)-vector \( x \). If there are no nonlinear constraints (i.e., \( ncnln = 0 \)), objfun will never be called by e04uc and confun may be the dummy function e04udm. (e04udm is included in the NAG Library.) If there are nonlinear constraints, the first call to confun will occur before the first call to objfun.

\[(\text{MODE,OBJF,OBJGRD}) = \text{objfun}(\text{mode},n,x,\text{objgrd,nstate})\]

istate

    integer array
    Need not be set if the (default) optional argument coldstart is used.

cjac

    double array
    In general, cjac need not be initialized before the call to e04uc. However, if \( \text{derivativelevel} = 2,3 \), you may optionally set the constant elements of cjac
(see argument nstate in the description of confun). Such constant elements need not be re-assigned on subsequent calls to confun.

**clamda**

double array

Need not be set if the (default) optional argument coldstart is used.

**r**

double array

Need not be initialized if the (default) optional argument coldstart is used.

**x**

double array

An initial estimate of the solution.

**optlist**

options list

Optional parameters may be listed, as shown in the following table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Central Difference Interval</td>
<td>double</td>
<td>Default values are computed</td>
</tr>
<tr>
<td>Cold Start</td>
<td></td>
<td>Default</td>
</tr>
<tr>
<td>Warm Start</td>
<td></td>
<td>Default = 0.01</td>
</tr>
<tr>
<td>Crash Tolerance</td>
<td>double</td>
<td>Default = 0.01</td>
</tr>
<tr>
<td>Defaults</td>
<td></td>
<td>Default = 3</td>
</tr>
<tr>
<td>Derivative Level</td>
<td>integer</td>
<td>Default values are computed</td>
</tr>
<tr>
<td>Difference Interval</td>
<td>double</td>
<td>Default = $\sqrt{\epsilon}$</td>
</tr>
<tr>
<td>Feasibility Tolerance</td>
<td>double</td>
<td>Default = $\epsilon^{0.9}$</td>
</tr>
<tr>
<td>Function Precision</td>
<td></td>
<td>Default = $\epsilon^{0.8}$</td>
</tr>
<tr>
<td>Hessian</td>
<td>no</td>
<td>Default = NO</td>
</tr>
<tr>
<td>Infinite Bound Size</td>
<td>double</td>
<td>Default = $10^{20}$</td>
</tr>
<tr>
<td>Infinite Step Size</td>
<td>double</td>
<td>Default = max(bigbnd, $10^{20}$)</td>
</tr>
<tr>
<td>Line Search Tolerance</td>
<td>double</td>
<td>Default = 0.9</td>
</tr>
<tr>
<td>Linear Feasibility Tolerance</td>
<td>double</td>
<td>Default = $\sqrt{\epsilon}$</td>
</tr>
<tr>
<td>Nonlinear Feasibility Tolerance</td>
<td>double</td>
<td>Default = $\epsilon^{0.33}$ or $\sqrt{\epsilon}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Major Iteration Limit</td>
<td>integer</td>
<td>Default = max(50, 3(n + n_L) + 10n_N)</td>
</tr>
<tr>
<td>Iteration Limit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Itns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Major Print Level</td>
<td>integer</td>
<td>Default for e04uc = 10</td>
</tr>
<tr>
<td>Print Level</td>
<td></td>
<td>Default for e04uc = 0</td>
</tr>
<tr>
<td>Minor Iteration Limit</td>
<td>integer</td>
<td>Default = max(50, 3(n + n_L + n_N))</td>
</tr>
<tr>
<td>Minor Print Level</td>
<td>integer</td>
<td>Default = 0</td>
</tr>
<tr>
<td>Monitoring File</td>
<td>integer</td>
<td>Default = -1</td>
</tr>
<tr>
<td>Optimality Tolerance</td>
<td>double</td>
<td>Default = $\epsilon^{0.8}$</td>
</tr>
<tr>
<td>Start Objective Check At Variable</td>
<td>integer</td>
<td>Default = 1</td>
</tr>
<tr>
<td>Stop Objective Check At Variable</td>
<td>integer</td>
<td>Default = n</td>
</tr>
<tr>
<td>Start Constraint Check At Variable</td>
<td>integer</td>
<td>Default = 1</td>
</tr>
<tr>
<td>Stop Constraint Check At Variable</td>
<td>integer</td>
<td>Default = n</td>
</tr>
<tr>
<td>Step Limit</td>
<td>double</td>
<td>Default = 2.0</td>
</tr>
<tr>
<td>Verify Level</td>
<td>integer</td>
<td>Default = 0</td>
</tr>
<tr>
<td>Verify</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Verify Constraint Gradients</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>Verify Gradients</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>Verify Objective Gradients</td>
<td>integer</td>
<td></td>
</tr>
</tbody>
</table>
Details

R interface to the NAG Fortran routine E04UCF.

Value

ITER  
integer  
The number of major iterations performed.

ISTATE  
integer array  
The status of the constraints in the QP working set at the point returned in x.  
The significance of each possible value of istate[j] is as follows:

C  
double array  
If ncnln > 0, c[i] contains the value of the ith nonlinear constraint function c_i at the final iterate for i = 1 \ldots n.

CJAC  
double array  
If ncnln > 0, cjac contains the Jacobian matrix of the nonlinear constraint functions at the final iterate, i.e., cjac[i, j] contains the partial derivative of the ith constraint function with respect to the jth variable for j = 1 \ldots n for i = 1 \ldots ncnln. (See the discussion of argument cjac under confun.)

CLAMDA  
double array  
The values of the QP multipliers from the last QP subproblem. clama[j] should be non-negative if istate[j] = 1 and non-positive if istate[j] = 2.

OBJF  
double  
The value of the objective function at the final iterate.

OBJGRD  
double array  
The gradient of the objective function at the final iterate (or its finite difference approximation).

R  
double array  
If hessian = NO, r contains the upper triangular Cholesky factor R of Q^T \tilde{H}Q, an estimate of the transformed and reordered Hessian of the Lagrangian at x (see eqn6 in the optional parameter description in the Fortran Library documentation). If hessian = YES, r contains the upper triangular Cholesky factor R of H, the approximate (untransformed) Hessian of the Lagrangian, with the variables in the natural order.

X  
double array  
The final estimate of the solution.

IFAIL  
integer  
ifail = 0 unless the function detects an error or a warning has been flagged (see the Errors section in Fortran library documentation).
optlist <- list()
ifail <- 0
confun = function(mode, ncnln, n, needc, x, cjac, nstate) {
  ldcj <- nrow(cjac)
  c <- as.matrix(mat.or.vec(ncnln, 1))
  if (nstate == 1) {
    cjac <- as.matrix(mat.or.vec(ldcj, n))
  }
  if (needc[1] > 0) {
    if (mode == 0 || mode == 2) {
    }
    if (mode == 1 || mode == 2) {
      cjac[1, 1] <- 2 * x[1]
      cjac[1, 2] <- 2 * x[2]
      cjac[1, 3] <- 2 * x[3]
      cjac[1, 4] <- 2 * x[4]
    }
  }
  if (needc[2] > 0) {
    if (mode == 0 || mode == 2) {
    }
    if (mode == 1 || mode == 2) {
}
list(MODE = as.integer(mode), C = as.matrix(c), CJAC = as.matrix(cjac))

objfun = function(mode, n, x, objgrd, nstate) {
  if (mode == 0 || mode == 2) {
  }
  else {
    objf <- 0
  }
  if (mode == 1 || mode == 2) {
  }
  list(MODE = as.integer(mode), OBJF = objf, OBJGRD = as.matrix(objgrd))
}
a <- matrix(c(1, 1, 1, 1), nrow = 1, ncol = 4, byrow = TRUE)
bl <- matrix(c(1, 1, 1, 1, -1e+25, -1e+25, 25), nrow = 7,
ncol = 1, byrow = TRUE)

bu <- matrix(c(5, 5, 5, 5, 20, 40, 1e+25), nrow = 7,
ncol = 1, byrow = TRUE)

istate <- as.matrix(mat.or.vec(7, 1))
cjac <- as.matrix(mat.or.vec(2, 4))
clamda <- as.matrix(mat.or.vec(7, 1))
r <- as.matrix(mat.or.vec(4, 4))
x <- matrix(c(1, 5, 5, 1), nrow = 4, ncol = 1, byrow = TRUE)
e04uc(a, bl, bu, confun, objfun, istate, cjac, clamda, r, x, optlist)

**Description**

e04uf is designed to minimize an arbitrary smooth function subject to constraints (which may include simple bounds on the variables, linear constraints and smooth nonlinear constraints) using a sequential quadratic programming (SQP) method. As many first derivatives as possible should be supplied by you; any unspecified derivatives are approximated by finite differences. It is not intended for large sparse problems.

e04uf may also be used for unconstrained, bound-constrained and linearly constrained optimization.

e04uf uses reverse communication for evaluating the objective function, the nonlinear constraint functions and any of their derivatives.

**Usage**

e04uf(irevcm, nclin, a, bl, bu, iter, istate, c, cjac, clamda, objf, objgrd, r, n = nrow(objgrd),
    ncnln = nrow(c))

**Arguments**

- **irevcm** integer
  Must be set to 0.

- **nclin** integer
  \( n_L \), the number of general linear constraints.

- **a** double array
  The \( i \)th row of the array \( a \) must contain the \( i \)th row of the matrix \( A_L \) of general linear constraints in eqn1. That is, the \( i \)th row contains the coefficients of the \( i \)th general linear constraint for \( i = 1 \ldots nclin \).

- **bl** double array
bu double array

Bl must contain the lower bounds and bu the upper bounds, for all the constraints in the following order. The first \( n \) elements of each array must contain the bounds on the variables, the next \( n_L \) elements the bounds for the general linear constraints (if any) and the next \( n_N \) elements the bounds for the general nonlinear constraints (if any). To specify a nonexistent lower bound (i.e., \( l_j = -\infty \)), set \( bl[j] = -\text{bigbnd} \), and to specify a nonexistent upper bound (i.e., \( u_j = +\infty \)), set \( bu[j] = \text{bigbnd} \); the default value of \( \text{bigbnd} \) is \( 10^{20} \), but this may be changed by the optional argument infiniteboundsize. To specify the \( j \)th constraint as an equality, set \( bl[j] = bu[j] = \beta \), say, where \( \text{abs}(\beta) < \text{bigbnd} \).

iter integer

Must remain unchanged from a previous call to e04uf.

istate integer array

Need not be set if the (default) optional argument coldstart is used.

c double array

Need not be set. If \( irevcm = 4, 6 \) and \( needc[i] > 0 \), \( c[i] \) must contain the value of the \( i \)th constraint at \( x \). The remaining elements of \( c \), corresponding to the non-positive elements of \( \text{needc} \), are ignored.

cjac double array

In general, \( \text{cjac} \) need not be initialized before the call to e04uf. However, if the optional argument \( \text{derivativelevel} = 2, 3 \), you may optionally set the constant elements of \( \text{cjac} \). Such constant elements need not be re-assigned on subsequent intermediate exits.

If \( irevcm = 5, 6 \) and \( needc[i] > 0 \), the \( i \)th row of \( \text{cjac} \) must contain the available elements of the vector \( \nabla c_i \) given by

\[
\nabla c_i = \begin{pmatrix}
\frac{\partial c_i}{\partial x_1}, & \frac{\partial c_i}{\partial x_2}, & \ldots, & \frac{\partial c_i}{\partial x_n}
\end{pmatrix}^T,
\]

where \( \frac{\partial c_i}{\partial x_j} \) is the partial derivative of the \( i \)th constraint with respect to the \( j \)th variable, evaluated at the point \( x \). The remaining rows of \( \text{cjac} \), corresponding to non-positive elements of \( \text{needc} \), are ignored.

clamda double array

Need not be set if the (default) optional argument coldstart is used.

objf double

Need not be set. If \( irevcm = 1, 3 \), \( \text{objf} \) must be set to the value of the objective function at \( x \).

objgrd double array

Need not be set. If \( irevcm = 2, 3 \), \( \text{objgrd} \) must contain the available elements of the gradient evaluated at \( x \).

r double array

Need not be initialized if the (default) optional argument coldstart is used.

x double array

An initial estimate of the solution.

iwork integer array
The arrays lwsav, iwsav, rwsav and cwsav must not be altered between calls to any of the functions e04wb, e04uf, e04ud e04ue.

Optional parameters may be listed, as shown in the following table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Central Difference Interval</td>
<td>double</td>
<td>Default values are computed</td>
</tr>
<tr>
<td>Cold Start</td>
<td></td>
<td>Default</td>
</tr>
<tr>
<td>Warm Start</td>
<td></td>
<td>Default = 0.01</td>
</tr>
<tr>
<td>Crash Tolerance</td>
<td>double</td>
<td>Default = 0.01</td>
</tr>
<tr>
<td>Derivative Level</td>
<td>integer</td>
<td>Default = 3</td>
</tr>
<tr>
<td>Difference Interval</td>
<td>double</td>
<td>Default values are computed</td>
</tr>
<tr>
<td>Feasibility Tolerance</td>
<td>double</td>
<td>Default = $\sqrt{\epsilon}$</td>
</tr>
<tr>
<td>Function Precision</td>
<td>double</td>
<td>Default = $\epsilon^{0.9}$</td>
</tr>
<tr>
<td>Hesslan</td>
<td></td>
<td>Default = NO</td>
</tr>
<tr>
<td>Infinite Bound Size</td>
<td>double</td>
<td>Default = $10^{20}$</td>
</tr>
<tr>
<td>Infinite Step Size</td>
<td>double</td>
<td>Default = $\max(bigbnd, 10^{20})$</td>
</tr>
<tr>
<td>Line Search Tolerance</td>
<td>double</td>
<td>Default = 0.9</td>
</tr>
<tr>
<td>Linear Feasibility Tolerance</td>
<td>double</td>
<td>Default = $\sqrt{\epsilon}$</td>
</tr>
<tr>
<td>Nonlinear Feasibility Tolerance List</td>
<td>double</td>
<td>Default = $\epsilon^{0.33}$ or $\sqrt{\epsilon}$</td>
</tr>
<tr>
<td>Major Iteration Limit</td>
<td>integer</td>
<td>Default = $\max(50, 3(n + n_L) + 10n_N)$</td>
</tr>
<tr>
<td>Iterns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Major Print Level</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>Print Level</td>
<td>integer</td>
<td>$= 0$</td>
</tr>
<tr>
<td>Minor Iteration Limit</td>
<td>integer</td>
<td>Default = $\max(50, 3(n + n_L + n_N))$</td>
</tr>
<tr>
<td>Minor Print Level</td>
<td>integer</td>
<td>Default = 0</td>
</tr>
<tr>
<td>Monitoring File</td>
<td>integer</td>
<td>Default = $-1$</td>
</tr>
<tr>
<td>Optimality Tolerance</td>
<td>double</td>
<td>Default = $\epsilon^{0.8}$</td>
</tr>
<tr>
<td>Start Objective Check At Variable</td>
<td>integer</td>
<td>Default = 1</td>
</tr>
<tr>
<td>Stop Objective Check At Variable</td>
<td>integer</td>
<td>Default = $n$</td>
</tr>
<tr>
<td>Start Constraint Check At Variable</td>
<td>integer</td>
<td>Default = 1</td>
</tr>
<tr>
<td>Stop Constraint Check At Variable</td>
<td>integer</td>
<td>Default = $n$</td>
</tr>
<tr>
<td>Step Limit</td>
<td>double</td>
<td>Default = 2.0</td>
</tr>
<tr>
<td>Verify</td>
<td>integer</td>
<td>Default = 0</td>
</tr>
<tr>
<td>Verify Constraint Gradients</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Verify Gradients</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Verify Objective Gradients</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Details

R interface to the NAG Fortran routine E04UFF.

Value

IREVCM

integer

Specifies what values the calling program must assign to arguments of e04uf before re-entering the function.

irevcm = 1: Set objf to the value of the objective function \( F(x) \).

irevcm = 2: Set objgrd[<j] to the value \( \frac{\partial F}{\partial x_j} \) if available for \( j = 1 \ldots n \).

irevcm = 3: Set objf and objgrd[<j] as for irevcm = 1 and irevcm = 2.

irevcm = 4: Set \( c[i] \) to the value of the constraint function \( c_i(x) \), for each \( i \) such that \( \text{needc}[i] > 0 \).

irevcm = 5: Set cjac[i,j] to the value \( \frac{\partial c_i}{\partial x_j} \) if available, for each \( i \) such that \( \text{needc}[i] > 0 \) and \( j = 1, 2, \ldots, n \).

irevcm = 6: Set \( c[i] \) and cjac[i,j] as for irevcm = 4 and irevcm = 5.

irevcm = 0.

ITER

integer

The number of major iterations performed.

ISTATE

integer array

The status of the constraints in the QP working set at the point returned in x. The significance of each possible value of \( \text{istate}[j] \) is as follows:

C

double array

If \( \text{ncnln} > 0 \), \( c[i] \) contains the value of the \( i \)th nonlinear constraint function \( c_i \) at the final iterate for \( i = 1 \ldots \text{ncnln} \).

CJAC

double array

If \( \text{ncnln} > 0 \), cjac contains the Jacobian matrix of the nonlinear constraint functions at the final iterate, i.e., cjac[i,j] contains the partial derivative of the \( i \)th constraint function with respect to the \( j \)th variable for \( j = 1 \ldots n \) for \( i = 1 \ldots \text{ncnln} \).

CLAMDA

double array

The values of the QP multipliers from the last QP subproblem. \( \text{clamda}[j] \) should be non-negative if \( \text{istate}[j] = 1 \) and non-positive if \( \text{istate}[j] = 2 \).

OBJF

double

The value of the objective function at the final iterate.

OBJGRD

double array

The gradient of the objective function at the final iterate (or its finite difference approximation).

R

double array

If \( \text{hessian} = \text{NO} \), \( r \) contains the upper triangular Cholesky factor \( R \) of \( Q^T \bar{H}Q \), an estimate of the transformed and reordered Hessian of the Lagrangian at \( x \) (see eqn6 in the optional parameter description in the Fortran Library documentation).
X  
double array
The point $x$ at which the objective function, constraint functions or their derivatives are to be evaluated.
The final estimate of the solution.

NEEDC  
integer array
If $irevcm \geq 4$, needc specifies the indices of the elements of c and/or cjac that must be assigned. If $needc[i] > 0$, then the $i$th element of c and/or the available elements of the $i$th row of cjac must be evaluated at $x$.

IWORK  
integer array
WORK  
double array
The amounts of workspace provided and required may be (by default for e04uf) output on the current advisory message unit (as defined by x04ab). As an alternative to computing liwork and lwork from the formulae given above, you may prefer to obtain appropriate values from the output of a preliminary run with liwork and lwork set to 1. (e04uf will then terminate with ifail = 9.)

CWSAV  
string array
The arrays lwsav, iwsav, rwsav and cwsav must not be altered between calls to any of the functions e04wb, e04uf, e04ud e04ue.

string array
The arrays lwsav, iwsav, rwsav and cwsav must not be altered between calls to any of the functions e04wb, e04uf, e04ud e04ue.

LNSAV  
boolean array
The arrays lwsav, iwsav, rwsav and cwsav must not be altered between calls to any of the functions e04wb, e04uf, e04ud e04ue.

IWSAV  
integer array
The arrays lwsav, iwsav, rwsav and cwsav must not be altered between calls to any of the functions e04wb, e04uf, e04ud e04ue.

RWSAV  
double array
The arrays lwsav, iwsav, rwsav and cwsav must not be altered between calls to any of the functions e04wb, e04uf, e04ud e04ue.

IFAIL  
integer
ifail = 0 unless the function detects an error or a warning has been flagged (see the Errors section in Fortran library documentation).

Author(s)
NAG

References
http://www.nag.co.uk/numeric/FL/nagdoc_fl23/pdf/E04/e04uff.pdf

Examples

```r
optlist <- list()
ifail <- 0
iwork <- as.matrix(mat.or.vec(0, 0))
```
work <- as.matrix(mat.or.vec(0, 0))
cwsav <- as.matrix(mat.or.vec(0, 0))
lwsav <- as.matrix(mat.or.vec(0, 0))
iwsav <- as.matrix(mat.or.vec(0, 0))
rwsav <- as.matrix(mat.or.vec(0, 0))
irevcm <- 0
nclin <- 1
a <- matrix(c(1, 1, 1, 1), nrow = 1, ncol = 4, byrow = TRUE)
bl <- matrix(c(1, 1, 1, 1, -1e+25, -1e+25, 25), nrow = 7,
ncol = 1, byrow = TRUE)
bu <- matrix(c(5, 5, 5, 5, 20, 40, 1e+25), nrow = 7,
ncol = 1, byrow = TRUE)
iter <- 0
istate <- as.matrix(mat.or.vec(7, 1))
c <- matrix(c(0, 0), nrow = 2, ncol = 1, byrow = TRUE)
cjac <- matrix(c(0, 0, 0, 0, 0, 0, 0, 0), nrow = 2,
ncol = 4, byrow = TRUE)
clamda <- as.matrix(mat.or.vec(7, 1))
objf <- 0
objgrd <- as.matrix(mat.or.vec(4, 1))
r <- as.matrix(mat.or.vec(4, 4))
x <- matrix(c(1, 5, 5, 1), nrow = 4, ncol = 1, byrow = TRUE)
iwork <- as.matrix(mat.or.vec(17, 1))
work <- as.matrix(mat.or.vec(192, 1))
if (ifail == 0) {
  ans <- e04uf(irevcm, nclin, a, bl, bu, iter, istate, c, cjac,
    clamda, objf, objgrd, r, x, iwork, work, cwsav, lwsav,
irevcm <- ans$IREVCM
iter <- ans$ITER
istate <- ans$ISTATE
c <- ans$C
cjac <- ans$CJAC
clamda <- ans$CLAMDA
objf <- ans$OBJF
objgrd <- ans$OBJGRD
r <- ans$R
x <- ans$X
needc <- ans$NEEDC
iwork <- ans$IWORK
work <- ans$WORK
cwsav <- ans$CWSAV
lwsav <- ans$LWSAV
iwsav <- ans$IWSAV
rwsav <- ans$RWSAV
ifail <- ans$IFAIL
while (irevcm > 0) {
  if (irevcm == 1 || irevcm == 3) {
  }
  if (irevcm == 2 || irevcm == 3) {
  }
  if (irevcm == 4 || irevcm == 6) {
    if (needc[1] > 0) {
    }
    if (needc[2] > 0) {
    }
  }
  if (irevcm == 5 || irevcm == 6) {
    if (needc[1] > 0) {
      cjac[1, 1] <- 2 %*% x[1]
      cjac[1, 2] <- 2 %*% x[2]
cjac[1, 3] <- 2 * x[3]
cjac[1, 4] <- 2 * x[4]

}  # end for
if (needc[2] > 0) {
}

ans <- e04uf(irevcm, nclin, a, bl, bu, iter, istate, c, cjac, clamda, objf, objgrd, r, x, iwork, work, cwsav, lwsav, iwsav, rwsav, optlist)
irevcm <- ans$IREVCM
iter <- ans$ITER
istate <- ans$ISTATE
c <- ans$C
cjac <- ans$CJAC
clamda <- ans$CLAMDA
objf <- ans$OBJF
objgrd <- ans$OBJGRD
r <- ans$R
x <- ans$X
needc <- ans$NEEDC
iwork <- ans$IWORK
work <- ans$WORK
cwsav <- ans$CWSAVE
lwsav <- ans$LWSAVE
iwsav <- ans$IWSAVE
rwsav <- ans$RWSAVE
ifail <- ans$IFAIL
if (ifail == 0) {
  writeLines(toString(cat(sprintf("\n Varbl Istate Value Lagr Mult\n", "\n"))))
  for (i in c(1:4)) {
    istate <- ans$ISTATE
    x <- ans$X
    clamda <- ans$CLAMDA
    writeLines(toString(cat(sprintf("% V %d %d %14.4f %12.4f \n", i, istate[i], x[i], clamda[i], "\n"))))
  }
}


ax <- a %*% x
writeLines(toString(cat(sprintf("\n L Con Istate Value Lagr Mult\n", "\n"))))

for (i in c(5:(4 + nclin))) {
  j <- i - 4
  istate <- ans$ISTATE
  clamda <- ans$CLAMDA
  writeLines(toString(cat(sprintf(" L %3d %3d %14.4f %12.4f\n", j, istate[i], ax[j], clamda[i], "\n"))))
}
writeLines(toString(cat(sprintf("\n L Con Istate Value Lagr Mult\n", "\n"))))

for (i in c((5 + nclin):((6 + nclin)))) {
  j <- i - 4 - nclin
  istate <- ans$ISTATE
  c <- ans$C
  clamda <- ans$CLAMDA
  writeLines(toString(cat(sprintf(" N %3d %3d %14.4f%12.4f\n", j, istate[i], c[j], clamda[i], "\n"))))
}
objf <- ans$OBJF
writeLines(toString(cat(sprintf("\n Final objective value = %15.7f\n", objf, "\n"))))

---
e04ug: NLP problem (sparse)

Description
e04ug solves sparse nonlinear programming problems.

Usage
e04ug(confun, objfun, n, m, ncnl, nonln, njnl, iobj, a, ha, ka, bl, bu, start,
\begin{verbatim}
nnz = nrow(a),
nname = nrow(names),
leniz = (1000),
lenz = (1000))

Arguments

confun function
confun must calculate the vector \( F(x) \) of nonlinear constraint functions and (optionally) its Jacobian \( \left( = \frac{\partial F}{\partial x} \right) \) for a specified \( n' \) \(( \leq n)\) element vector \( x \). If there are no nonlinear constraints (i.e., \( ncnln = 0 \)), confun will never be called by e04ug and confun may be the dummy function e04ugm. (e04ugm is included in the NAG Library.) If there are nonlinear constraints, the first call to confun will occur before the first call to objfun.

\((MODE,F,FJAC) = \text{confun}(mode,ncnln,njnln,nnzjac,x,fjac,nstate)\)

objfun function
objfun must calculate the nonlinear part of the objective function \( f(x) \) and (optionally) its gradient \( \left( = \frac{\partial f}{\partial x} \right) \) for a specified \( n' \) \(( \leq n)\) element vector \( x \). If there are no nonlinear objective variables (i.e., \( nonln = 0 \)), objfun will never be called by e04ug and objfun may be the dummy function e04ugn. (e04ugn is included in the NAG Library.)

\((MODE,OBJF,OBJGRD) = \text{objfun}(mode,nonln,x,objgrd,nstate)\)

n integer
\( n \), the number of variables (excluding slacks). This is the number of columns in the full Jacobian matrix \( A \).

m integer
\( m \), the number of general constraints (or slacks). This is the number of rows in \( A \), including the free row (if any; see iobj). Note that \( A \) must contain at least one row. If your problem has no constraints, or only upper and lower bounds on the variables, then you must include a dummy ‘free’ row consisting of a single (zero) element subject to ‘infinite’ upper and lower bounds. Further details can be found under the descriptions for iobj, nnz, a, ha, ka, bl and bu.

ncnln integer
\( n_N \), the number of nonlinear constraints.

nonln integer
\( n'_1 \), the number of nonlinear objective variables. If the objective function is nonlinear, the leading \( n'_1 \) columns of \( A \) belong to the nonlinear objective variables. (See also the description for njnln.)

njnln integer
\( n''_1 \), the number of nonlinear Jacobian variables. If there are any nonlinear constraints, the leading \( n''_1 \) columns of \( A \) belong to the nonlinear Jacobian variables. If \( n'_1 > 0 \) and \( n''_1 > 0 \), the nonlinear objective and Jacobian variables overlap. The total number of nonlinear variables is given by \( n = \max(n'_1,n''_1) \).

iobj integer
If \( iobj > ncnln \), row \( iobj \) of \( A \) is a free row containing the nonzero elements of the linear part of the objective function.
\( iobj = 0 \): There is no free row.
\( iobj = -1 \): There is a dummy ‘free’ row.
\end{verbatim}
double array

The nonzero elements of the Jacobian matrix \( A \), ordered by increasing column index. Since the constraint Jacobian matrix \( J(x') \) must always appear in the top left-hand corner of \( A \), those elements in a column associated with any nonlinear constraints must come before any elements belonging to the linear constraint matrix \( G \) and the free row (if any; see \( iobj \)).

integer array

\( ha[i] \) must contain the row index of the nonzero element stored in \( a[i] \) for \( i = 1 \ldots \text{nnz} \). The row indices for a column may be supplied in any order subject to the condition that those elements in a column associated with any nonlinear constraints must appear before those elements associated with any linear constraints (including the free row, if any). Note that \( \text{con} \) must define the Jacobian elements in the same order. If \( iobj = -1 \), set \( ha[1] = 1 \).

integer array

\( ka[j] \) must contain the index in \( a \) of the start of the \( j \)th column for \( j = 1 \ldots n \). To specify the \( j \)th column as empty, set \( ka[j] = ka[j+1] \). Note that the first and last elements of \( ka \) must be such that \( ka[1] = 1 \) and \( ka[n+1] = \text{nnz} + 1 \). If \( iobj = -1 \), set \( ka[j] = 2 \) for \( j = 2 \ldots n \).

double array

\( l \), the lower bounds for all the variables and general constraints, in the following order. The first \( n \) elements of \( bl \) must contain the bounds on the variables \( x \), the next \( n \text{ncnln} \) elements the bounds for the nonlinear constraints \( F(x) \) (if any) and the next \((m - n \text{ncln}) \) elements the bounds for the linear constraints \( Gx \) and the free row (if any). To specify a nonexistent lower bound (i.e., \( l_j = -\infty \)), set \( bl[j] \leq -\text{bigbnd} \). To specify the \( j \)th constraint as an equality, set \( bl[j] = bu[j] = \beta \), say, where \( \text{abs}(\beta) < \text{bigbnd} \). If \( iobj = -1 \), set \( bl[n + \text{abs}(iobj)] \leq -\text{bigbnd} \).

double array

\( u \), the upper bounds for all the variables and general constraints, in the following order. The first \( n \) elements of \( bu \) must contain the bounds on the variables \( x \), the next \( n \text{ncnln} \) elements the bounds for the nonlinear constraints \( F(x) \) (if any) and the next \((m - n \text{ncln}) \) elements the bounds for the linear constraints \( Gx \) and the free row (if any). To specify a nonexistent upper bound (i.e., \( u_j = +\infty \)), set \( bu[j] \geq \text{bigbnd} \). To specify the \( j \)th constraint as an equality, set \( bu[j] = bl[j] = \beta \), say, where \( \text{abs}(\beta) < \text{bigbnd} \). If \( iobj = -1 \), set \( bu[n + \text{abs}(iobj)] \geq \text{bigbnd} \).

string

Indicates how a starting basis is to be obtained.

\( start = 'C' \): An internal Crash procedure will be used to choose an initial basis.

\( start = 'W' \): A basis is already defined in istate and ns (probably from a previous call).

string array

Specifies the column and row names to be used in the printed output.

integer

\( nS \), the number of superbasics. It need not be specified if \( start = 'C' \), but must retain its value from a previous call when \( start = 'W' \).

double array

The initial values of the variables and slacks \( (xs) \). (See the description for istate.)
istate
integer array
If \( \text{start} = 'C' \), the first \( n \) elements of istate and xs must specify the initial states and values, respectively, of the variables \( x \). (The slacks \( s \) need not be initialized.) An internal Crash procedure is then used to select an initial basis matrix \( B \). The initial basis matrix will be triangular (neglecting certain small elements in each column). It is chosen from various rows and columns of \( (A - I) \). Possible values for istate[\( j \)] are as follows:

clamda
double array
If \( \text{ncnln} > 0 \), clamda[\( j \)] must contain a Lagrange multiplier estimate for the \( j \)th nonlinear constraint \( F_j(x) \) for \( j = n + 1 \ldots n + \text{ncnln} \). If nothing special is known about the problem, or there is no wish to provide special information, you may set clamda[\( j \)] = 0.0. The remaining elements need not be set.

optlist
options list
Optional parameters may be listed, as shown in the following table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Central Difference Interval</td>
<td>double</td>
<td>Default ( = \sqrt{\text{functionprecision}} )</td>
</tr>
<tr>
<td>Check Frequency</td>
<td>integer</td>
<td>Default = 60</td>
</tr>
<tr>
<td>Crash Option</td>
<td>integer</td>
<td>Default = 0 or 3</td>
</tr>
<tr>
<td>Crash Tolerance</td>
<td>double</td>
<td>Default = 0.1</td>
</tr>
<tr>
<td>Defaults</td>
<td>integer</td>
<td>Default = 3</td>
</tr>
<tr>
<td>Derivative Level</td>
<td>integer</td>
<td>Default</td>
</tr>
<tr>
<td>Derivative Linesearch</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nonderivative Linesearch</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Elastic Weight</td>
<td>double</td>
<td>Default = 1.0 or 100.0</td>
</tr>
<tr>
<td>Expand Frequency</td>
<td>integer</td>
<td>Default = 10000</td>
</tr>
<tr>
<td>Factorization Frequency</td>
<td>integer</td>
<td>Default = 50 or 100</td>
</tr>
<tr>
<td>Infeasible Exit</td>
<td></td>
<td>Default</td>
</tr>
<tr>
<td>Feasible Exit</td>
<td></td>
<td>Default</td>
</tr>
<tr>
<td>Minimize</td>
<td></td>
<td>Default</td>
</tr>
<tr>
<td>Maximize</td>
<td></td>
<td>Default</td>
</tr>
<tr>
<td>Forward Difference Interval</td>
<td>double</td>
<td>Default ( = \sqrt{\text{functionprecision}} )</td>
</tr>
<tr>
<td>Function Precision</td>
<td>double</td>
<td>Default ( = \epsilon^{0.8} )</td>
</tr>
<tr>
<td>Hessian Frequency</td>
<td>integer</td>
<td>Default = 999999999</td>
</tr>
<tr>
<td>Hessian Full Memory</td>
<td></td>
<td>Default when ( \bar{n} &lt; 75 )</td>
</tr>
<tr>
<td>Hessian Limited Memory</td>
<td></td>
<td>Default when ( \bar{n} \geq 75 )</td>
</tr>
<tr>
<td>Hessian Updates</td>
<td>integer</td>
<td>Default = 20 or ( 999999999 )</td>
</tr>
<tr>
<td>Infinite Bound Size</td>
<td>double</td>
<td>Default = 10^{20}</td>
</tr>
<tr>
<td>Iteration Limit</td>
<td>integer</td>
<td>Default = 10000</td>
</tr>
<tr>
<td>Linesearch Tolerance</td>
<td>double</td>
<td>Default = 0.9</td>
</tr>
<tr>
<td>List</td>
<td></td>
<td>Default for ( e04ug = \text{list} )</td>
</tr>
<tr>
<td>Nolist</td>
<td></td>
<td>Default for ( e04ug = \text{nolist} )</td>
</tr>
<tr>
<td>LU Density Tolerance</td>
<td>double</td>
<td>Default = 0.6</td>
</tr>
<tr>
<td>LU Singularity Tolerance</td>
<td>double</td>
<td>Default ( = \epsilon^{0.67} )</td>
</tr>
<tr>
<td>LU Factor Tolerance</td>
<td>double</td>
<td>Default = 5.0 or 100.0</td>
</tr>
<tr>
<td>LU Update Tolerance</td>
<td>double</td>
<td>Default = 5.0 or 10.0</td>
</tr>
<tr>
<td>Major Feasibility Tolerance</td>
<td>double</td>
<td>Default ( = \sqrt{\epsilon} )</td>
</tr>
<tr>
<td>Major Iteration Limit</td>
<td>integer</td>
<td>Default = 1000</td>
</tr>
<tr>
<td>Major Optimality Tolerance</td>
<td>double</td>
<td>Default = ( \sqrt{\epsilon} )</td>
</tr>
<tr>
<td>Optimality Tolerance</td>
<td>double</td>
<td>= 0</td>
</tr>
<tr>
<td>Major Print Level</td>
<td>integer</td>
<td>= 0</td>
</tr>
</tbody>
</table>
Print Level

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Major Step Limit</td>
<td>double</td>
<td>Default = 2.0</td>
</tr>
<tr>
<td>Minor Feasibility Tolerance</td>
<td>double</td>
<td>Default = $\sqrt{\varepsilon}$</td>
</tr>
<tr>
<td>Feasibility Tolerance</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>Minor Iteration Limit</td>
<td>integer</td>
<td>Default = 500</td>
</tr>
<tr>
<td>Minor Optimality Tolerance</td>
<td>double</td>
<td>Default = $\sqrt{\varepsilon}$</td>
</tr>
<tr>
<td>Minor Print Level</td>
<td>integer</td>
<td>Default = 0</td>
</tr>
<tr>
<td>Monitoring File</td>
<td>integer</td>
<td>Default = −1</td>
</tr>
<tr>
<td>Partial Price</td>
<td>integer</td>
<td>Default = 1 or 10</td>
</tr>
<tr>
<td>Pivot Tolerance</td>
<td>double</td>
<td>Default = $\varepsilon^{0.67}$</td>
</tr>
<tr>
<td>Scale Option</td>
<td>integer</td>
<td>Default = 1 or 2</td>
</tr>
<tr>
<td>Scale Tolerance</td>
<td>double</td>
<td>Default = 0.9</td>
</tr>
<tr>
<td>Start Objective Check At Column</td>
<td>integer</td>
<td>Default = 1</td>
</tr>
<tr>
<td>Stop Objective Check At Column</td>
<td>integer</td>
<td>Default = $n_1'$</td>
</tr>
<tr>
<td>Start Constraint Check At Column</td>
<td>integer</td>
<td>Default = 1</td>
</tr>
<tr>
<td>Stop Constraint Check At Column</td>
<td>integer</td>
<td>Default = $n_1''$</td>
</tr>
<tr>
<td>Superbasics Limit</td>
<td>integer</td>
<td>Default = $\min(500, n + 1)$</td>
</tr>
<tr>
<td>Unbounded Objective</td>
<td>double</td>
<td>Default = $10^{15}$</td>
</tr>
<tr>
<td>Unbounded Step Size</td>
<td>double</td>
<td>Default = $\max(\text{bigbnd}, 10^{20})$</td>
</tr>
<tr>
<td>Verify Level</td>
<td>integer</td>
<td>Default = 0</td>
</tr>
<tr>
<td>Violation Limit</td>
<td>double</td>
<td>Default = 10.0</td>
</tr>
</tbody>
</table>

nnz: integer: default = nrow(a)

The number of nonzero elements in $A$ (including the Jacobian for any nonlinear constraints). If $iobj = -1$, set nnz = 1.

nname: integer: default = nrow(names)

The number of column (i.e., variable) and row (i.e., constraint) names supplied in names.

nname = 1: There are no names. Default names will be used in the printed output.

nname = $n + m$: All names must be supplied.

leniz: integer: default = (max(500,(n+m)))

integer: default = (max(500,(n+m)))

lenz: integer: default = (500)

integer: default = (500)

Details

R interface to the NAG Fortran routine E04UGF.

Value

A: double array

Elements in the nonlinear part corresponding to nonlinear Jacobian variables are overwritten.

NS: integer

The final number of superbasics.

XS: double array

The final values of the variables and slacks ($xs$).
ISTATE integer array
The final states of the variables and slacks \((xs)\). The significance of each possible value of \(istate[j]\) is as follows:

CLAMDA double array
A set of Lagrange multipliers for the bounds on the variables (reduced costs) and the general constraints (shadow costs). More precisely, the first \(n\) elements contain the multipliers for the bounds on the variables, the next \(ncln\) elements contain the multipliers for the nonlinear constraints \(F(x)\) (if any) and the next \((m - ncnln)\) elements contain the multipliers for the linear constraints \(Gx\) and the free row (if any).

MINIZ integer
The minimum value of \(leniz\) required to start solving the problem. If \(ifail = 12\), \e04ug may be called again with \(leniz\) suitably larger than \(miniz\). (The bigger the better, since it is not certain how much workspace the basis factors need.)

MINZ integer
The minimum value of \(lenz\) required to start solving the problem. If \(ifail = 13\), \e04ug may be called again with \(lenz\) suitably larger than \(minz\). (The bigger the better, since it is not certain how much workspace the basis factors need.)

NINF integer
The number of constraints that lie outside their bounds by more than the value of the optional argument minorfeasibilitytolerance.

SINF double
The sum of the infeasibilities of constraints that lie outside their bounds by more than the value of the optional argument minorfeasibilitytolerance.

OBJ double
The value of the objective function.

IFAIL integer
\(ifail = 0\) unless the function detects an error or a warning has been flagged (see the Errors section in Fortran library documentation).

Author(s)
NAG

References
http://www.nag.co.uk/numeric/FL/nagdoc_fl23/pdf/E04/e04ugf.pdf

Examples

```r
optlist <- list()
ifail <- 0
confun = function(mode, ncnln, njnln, nnzjac, x, fjac, nstate) {
    f <- as.matrix(mat.or.vec(ncnln, 1))
    if (mode == 0 || mode == 2) {
```
\[ f[1] \leftarrow 1000 \times \sin(-x[1] - 0.25) + 1000 \times \sin(-x[2] - 0.25) \]
\[ f[2] \leftarrow 1000 \times \sin(x[1] - 0.25) + 1000 \times \sin(x[1] - x[2] - 0.25) \]
\[ f[3] \leftarrow 1000 \times \sin(x[2] - x[1] - 0.25) + 1000 \times \sin(x[2] - 0.25) \]

\[
\text{if (mode == 1 || mode == 2) }
\]
\[
fjac[1] \leftarrow -1000 \times \cos(-x[1] - 0.25) \\
fjac[2] \leftarrow 1000 \times \cos(x[1] - 0.25) + 1000 \times \cos(x[1] - x[2] - 0.25) \\
fjac[3] \leftarrow -1000 \times \cos(x[2] - x[1] - 0.25) \\
fjac[4] \leftarrow -1000 \times \cos(-x[2] - 0.25) \\
fjac[5] \leftarrow -1000 \times \cos(x[1] - x[2] - 0.25) \\
\]

\[
\text{list(MODE = as.integer(mode), F = as.matrix(f), FJAC = as.matrix(fjac))}
\]

\[
\text{objfun = function(mode, nonln, x, objgrd, nstate) }
\]

\[
\text{if (mode == 0 || mode == 2) }
\]
\[
\text{objf} \leftarrow 1e-06 \times x[3]^3 + 2e-06 \times x[4]^3/3 \\
\]
\[
\text{if (mode == 1 || mode == 2) }
\]
\[
\text{objgrd[1] \leftarrow 0 } \\
\text{objgrd[2] \leftarrow 0} \\
\text{objgrd[3] \leftarrow 3e-06 \times x[3]^2} \\
\text{objgrd[4] \leftarrow 2e-06 \times x[4]^2} \\
\]

\[
\text{list(MODE = as.integer(mode), OBJF = objf, OBJGRD = as.matrix(objgrd))}
\]

n <- 4
m <- 6
ncnl <- 3
nonl <- 4
njnl <- 2
iobj <- 6

a <- matrix(c(1e+25, 1e+25, 1e+25, 1, -1, 1e+25, 1e+25,
              1e+25, -1, 1, 3, -1, -1, 2), nrow = 14, ncol = 1, byrow = TRUE)

ha <- matrix(c(1, 2, 3, 5, 4, 1, 2, 3, 5, 4, 6, 1,
               2, 6), nrow = 14, ncol = 1, byrow = TRUE)

ka <- matrix(c(1, 6, 11, 13, 15), nrow = 5, ncol = 1,
              byrow = TRUE)

bl <- matrix(c(-0.55, -0.55, 0, 0, -894.8, -894.8,-1294.8, -0.55, -0.55, -1e+25), nrow = 10, ncol = 1, byrow = TRUE)

bu <- matrix(c(0.55, 0.55, 1200, 1200, -894.8, -894.8,-1294.8, 1e+25, 1e+25, 1e+25), nrow = 10, ncol = 1, byrow = TRUE)

start <- "C"

names <- matrix(c("Varble 1", "Varble 2", "Varble 3",
"Varble 4", "NlnCon 1", "NlnCon 2", "NlnCon 3", "LinCon 1",
"LinCon 2", "Free Row"), nrow = 10, byrow = TRUE)

ns <- 0
xs <- matrix(c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0), nrow = 10,
ncol = 1, byrow = TRUE)

istate <- as.matrix(mat.or.vec(10, 1))

clamda <- matrix(c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
                  nrow = 10, ncol = 1, byrow = TRUE)

leniz <- 1000
e04us

\textit{e04us} is designed to minimize an arbitrary smooth sum of squares function subject to constraints (which may include simple bounds on the variables, linear constraints and smooth nonlinear constraints) using a sequential quadratic programming (SQP) method. As many first derivatives as possible should be supplied by you; any unspecified derivatives are approximated by finite differences. See the description of the optional argument derivativelevel, in the Fortran library documentation. It is not intended for large sparse problems.

e04us may also be used for unconstrained, bound-constrained and linearly constrained optimization.

\textbf{Usage}

e04us(a, bl, bu, y, confun, objfun, istate, cjac, fjac, clamda, r, x, optlist,
    m = nrow(y),
    n = nrow(x),
    nclin = nrow(a),
    ncnln = nrow(cjac))

\textbf{Arguments}

\textbf{a}

double array

The \textit{i}th row of \textit{a} contains the \textit{i}th row of the matrix $A_L$ of general linear constraints in eqn\ref{eqn1}. That is, the \textit{i}th row contains the coefficients of the \textit{i}th general linear constraint for $i = 1 \ldots nlin$.

\textbf{bl}

double array

Must contain the lower bounds and \textit{bu} the upper bounds, for all the constraints in the following order. The first \textit{n} elements of each array must contain the bounds on the variables, the next \textit{n}_L elements the bounds for the general linear constraints (if any) and the next \textit{n}_N elements the bounds for the general nonlinear constraints (if any). To specify a nonexistent lower bound (i.e., $l_j = -\infty$), set $bl[j] = -\text{bigbnd}$, and to specify a nonexistent upper bound (i.e., $u_j = +\infty$), set $bu[j] = \text{bigbnd}$; the default value of \text{bigbnd} is $10^{20}$, but this may be changed by the optional argument infiniteboundsize. To specify the \textit{j}th constraint as an equality, set $bl[j] = bu[j] = \beta$, say, where $\text{abs} (\beta) < \text{bigbnd}$.

\textbf{y}

double array

The coefficients of the constant vector $y$ of the objective function.
confun function
confun must calculate the vector \( c(x) \) of nonlinear constraint functions and (optionally) its Jacobian \( \frac{\partial c}{\partial x} \) for a specified \( n \) element vector \( x \). If there are no nonlinear constraints (i.e., \( ncnln = 0 \)), confun will never be called by e04us and confun may be the dummy function e04udm. (e04udm is included in the NAG Library.) If there are nonlinear constraints, the first call to confun will occur before the first call to objfun.

\[(MODE, C, CJAC) = \text{confun}(mode, ncnln, n, needc, x, cjac, nstate)\]

objfun function
objfun must calculate either the \( i \)th element of the vector \( f(x) = (f_1(x) \, f_2(x) \ldots f_m(x))^T \) or all \( m \) elements of \( f(x) \) and (optionally) its Jacobian \( \frac{\partial f}{\partial x} \) for a specified \( n \) element vector \( x \).

\[(MODE, F, FJAC) = \text{objfun}(mode, m, n, needfi, x, fjac, nstate)\]

istate integer array
Need not be set if the (default) optional argument coldstart is used.

cjac double array
In general, cjac need not be initialized before the call to e04us. However, if \( \text{derivativelevel} = 3 \), you may optionally set the constant elements of cjac (see argument nstate in the description of confun). Such constant elements need not be re-assigned on subsequent calls to confun.

fjac double array
In general, fjac need not be initialized before the call to e04us. However, if \( \text{derivativelevel} = 3 \), you may optionally set the constant elements of fjac (see argument nstate in the description of objfun). Such constant elements need not be re-assigned on subsequent calls to objfun.

clamda double array
Need not be set if the (default) optional argument coldstart is used.

r double array
Need not be initialized if the (default) optional argument coldstart is used.

x double array
An initial estimate of the solution.

optlist options list
Optional parameters may be listed, as shown in the following table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Central Difference Interval</td>
<td>double</td>
<td>Default values are computed</td>
</tr>
<tr>
<td>Cold Start</td>
<td></td>
<td>Default</td>
</tr>
<tr>
<td>Warm Start</td>
<td></td>
<td>Default</td>
</tr>
<tr>
<td>Crash Tolerance</td>
<td>double</td>
<td>Default ( = 0.01 )</td>
</tr>
<tr>
<td>Derivative Level</td>
<td>integer</td>
<td>Default ( = 3 )</td>
</tr>
<tr>
<td>Difference Interval</td>
<td>double</td>
<td>Default values are computed</td>
</tr>
<tr>
<td>Feasibility Tolerance</td>
<td>double</td>
<td>Default ( = \sqrt{\epsilon} )</td>
</tr>
<tr>
<td>Function Precision</td>
<td>double</td>
<td>Default ( = \epsilon^{0.9} )</td>
</tr>
<tr>
<td>Hessian</td>
<td>no</td>
<td>Default ( = NO )</td>
</tr>
<tr>
<td>Infinite Bound Size</td>
<td>double</td>
<td>Default ( = 10^{20} )</td>
</tr>
<tr>
<td>Infinite Step Size</td>
<td>double</td>
<td>Default ( = \max(\text{bigbnd}, 10^{20}) )</td>
</tr>
</tbody>
</table>
JTJ Initial Hessian
Unit Initial Hessian
Line Search Tolerance
Linear Feasibility Tolerance
Nonlinear Feasibility Tolerance
List
Nolist
Major Iteration Limit
Iteration Limit
Iters
Itns
Major Print Level
Print Level
Minor Iteration Limit
Minor Print Level
Monitoring File
Optimality Tolerance
Reset Frequency
Start Objective Check At Variable
Stop Objective Check At Variable
Start Constraint Check At Variable
Stop Constraint Check At Variable
Step Limit
Verify Level
Verify
Verify Constraint Gradients
Verify Gradients
Verify Objective Gradients

\[ m \] integer: \texttt{default} = \texttt{nrow(y)}
\[ n \] integer: \texttt{default} = \texttt{nrow(x)}
\[ n_{\text{cl}} \] integer: \texttt{default} = \texttt{nrow(a)}
\[ n_{\text{cn}} \] integer: \texttt{default} = \texttt{nrow(cjac)}

Details
R interface to the NAG Fortran routine E04USF.

Value

\texttt{ITER} integer
The number of major iterations performed.

\texttt{ISTATE} integer array
The status of the constraints in the QP working set at the point returned in \( x \).
The significance of each possible value of \( \text{istate}[j] \) is as follows:
C
do double array
If ncln > 0, \( c[i] \) contains the value of the \( i \)th nonlinear constraint function \( c_i \) at the final iterate for \( i = 1 \ldots ncln \).

CJAC
do double array
If ncln > 0, cjac contains the Jacobian matrix of the nonlinear constraint functions at the final iterate, i.e., \( cjac[i,j] \) contains the partial derivative of the \( i \)th constraint function with respect to the \( j \)th variable for \( j = 1 \ldots n \) for \( i = 1 \ldots ncln \). (See the discussion of argument cjac under confun.)

F
do double array
\( f[i] \) contains the value of the \( i \)th function \( f_i \) at the final iterate for \( i = 1 \ldots m \).

FJAC
do double array
The Jacobian matrix of the functions \( f_1, f_2, \ldots, f_m \) at the final iterate, i.e., \( fjac[i,j] \) contains the partial derivative of the \( i \)th function with respect to the \( j \)th variable for \( j = 1 \ldots n \) for \( i = 1 \ldots m \). (See also the discussion of argument fjac under objfun.)

CLAMDA
do double array
The values of the QP multipliers from the last QP subproblem. \( clamda[j] \) should be non-negative if \( istate[j] = 1 \) and non-positive if \( istate[j] = 2 \).

OBJF
do double
The value of the objective function at the final iterate.

R
do double array
If hessian = NO, \( r \) contains the upper triangular Cholesky factor \( R \) of \( Q^T \tilde{H} Q \), an estimate of the transformed and reordered Hessian of the Lagrangian at \( x \) (see eqn6). If hessian = YES, \( r \) contains the upper triangular Cholesky factor \( R \) of \( H \), the approximate (untransformed) Hessian of the Lagrangian, with the variables in the natural order.

X
do double array
The final estimate of the solution.

IFAIL
integer
ifail = 0 unless the function detects an error or a warning has been flagged (see the Errors section in Fortran library documentation).

Author(s)
NAG

References
http://www.nag.co.uk/numeric/FL/nagdoc_fl23/pdf/E04/e04usf.pdf

Examples

optlist <- list()
ifail <- 0
confun <- function(mode, ncnl, n, needc, x, cjac,
nstate) {
lcrj <- nrow(cjac)
c <- as.matrix(mat.or.vec(ncnln, 1))

if (nstate == 1) {
    cjac <- as.matrix(mat.or.vec(ncnln, n))
}

if (needc[1] > 0) {
    if (mode == 0 || mode == 2) {
    }
    if (mode == 1 || mode == 2) {
        cjac[1, 1] <- -x[2]
        cjac[1, 2] <- -x[1] + 0.49
    }
}
list(MODE = as.integer(mode), C = as.matrix(c), CJAC = as.matrix(cjac))

objfun = function(mode, m, n, needfi, x, fjac, nstate) {
    ldfj <- nrow(fjac)
    f <- as.matrix(mat.or.vec(m, 1))
    a <- matrix(c(8, 8, 10, 10, 10, 10, 12, 12, 12, 12, 14, 14,
                  14, 16, 16, 18, 18, 20, 20, 20, 22, 22, 22, 24, 24,
                  24, 26, 26, 28, 28, 30, 30, 30, 32, 32, 34, 34, 36, 36,
                  38, 38, 40, 42), nrow = 1, ncol = 44, byrow = TRUE)
    for (i in c(1:m)) {
        temp <- exp(-x[2] * (a[i] - 8))
        if (mode == 0 || mode == 2) {
            f[i] <- x[1] + (0.49 - x[1]) * temp
        }
        if (mode == 1 || mode == 2) {
            fjac[i, 1] <- 1 - temp
            fjac[i, 2] <- -(0.49 - x[1]) * (a[i] - 8) * temp
        }
    }
    list(MODE = as.integer(mode), F = as.matrix(f), FJAC = as.matrix(fjac))
}

a <- matrix(c(1, 1), nrow = 1, ncol = 2, byrow = TRUE)

bl <- matrix(c(0.4, -4, 1, 0), nrow = 4, ncol = 1,
byrow = TRUE)

bu <- matrix(c(1e+25, 1e+25, 1e+25, 1e+25), nrow = 4,
ncol = 1, byrow = TRUE)

y <- matrix(c(0.49, 0.49, 0.48, 0.47, 0.48, 0.47,
0.46, 0.46, 0.45, 0.43, 0.45, 0.43, 0.43, 0.44, 0.43, 0.43,
0.46, 0.45, 0.42, 0.43, 0.43, 0.41, 0.41, 0.41, 0.4, 0.41,
0.4, 0.41, 0.38, 0.4, 0.38, 0.39, 0.39), nrow = 44, ncol = 1,
byrow = TRUE)

istate <- as.matrix(mat.or.vec(4, 1))
cjac <- matrix(c(0, 0), nrow = 1, ncol = 2, byrow = TRUE)

fjac <- matrix(c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0), nrow = 44, ncol = 2, byrow = TRUE)

clamda <- as.matrix(mat.or.vec(4, 1))
r <- matrix(c(0, 0, 0, 0), nrow = 2, ncol = 2, byrow = TRUE)

x <- matrix(c(0.4, 0), nrow = 2, ncol = 1, byrow = TRUE)
e04us(a, bl, bu, y, confun, objfun, istate, cjac,
fjac, clamda, r, x, optlist)

e04vj

Description

e04vj may be used before e04vh to determine the sparsity pattern for the Jacobian.
Usage

e04vj(nf, usrfun, lena, leng, x, xlow, xupp,
    n = nrow(x))

Arguments

nf  integer
    nf, the number of problem functions in $F(x)$, including the objective function (if any) and the linear and nonlinear constraints. Simple upper and lower bounds on $x$ can be defined using the arguments xlow and xupp and should not be included in $F$.

usrfun  function
    usrfun must define the problem functions $F(x)$. This function is passed to e04vj as the external argument usrfun.

    $(STATUS,F,G) = usrfun(status,n,x,needf,nf,f,needg,leng,g)$

lena  integer
    Lena should be an overestimate of the number of elements in the linear part of the Jacobian.

leng  integer
    Leng should be an overestimate of the number of elements in the nonlinear part of the Jacobian.

x  double array
    An initial estimate of the variables $x$. The contents of $x$ will be used by e04vj in the call of usrfun, and so each element of $x$ should be within the bounds given by xlow xupp.

xlow  double array

xupp  double array
    Contain the lower and upper bounds $l_x$ and $u_x$ on the variables $x$.

n  integer: default = nrow(x)
    n, the number of variables.

Details

R interface to the NAG Fortran routine E04VJF.

Value

IAFUN  integer array

JAVAR  integer array

NEA  integer
    Is the number of nonzero entries in $A$ such that $F(x) = f(x) + Ax$.

A  double array
    Define the coordinates $(ij)$ and values $A_{ij}$ of the nonzero elements of the linear part $A$ of the function $F(x) = f(x) + Ax$.

IGFUN  integer array
JGVAR  integer array
Define the coordinates \((ij)\) of the nonzero elements of \(G\), the nonlinear part of the derivatives \(J(x) = G(x) + A\) of the function \(F(x) = f(x) + Ax\).

NEG  integer
The number of nonzero entries in \(G\).

IFAIL  integer
ifail = 0 unless the function detects an error or a warning has been flagged (see the Errors section in Fortran library documentation).

Author(s)
NAG

References
http://www.nag.co.uk/numeric/FL/nagdoc_fl23/pdf/E04/e04vjf.pdf

Examples

```r
optlist <- list()
ifail <- 0
usrfun = function(status, n, x, needf, nf, f, needg,
  leng, g) {
  list(STATUS = as.integer(status), F = as.matrix(f), G = as.matrix(g))
}

nf <- 6
lena <- 300
leng <- 300
x <- matrix(c(0, 0, 0, 0), nrow = 4, ncol = 1, byrow = TRUE)

xlow <- matrix(c(-0.55, -0.55, 0, 0), nrow = 4, ncol = 1,
  byrow = TRUE)

xupp <- matrix(c(0.55, 0.55, 1200, 1200), nrow = 4,
  byrow = TRUE)
```
**Description**

e04wd is designed to minimize an arbitrary smooth function subject to constraints (which may include simple bounds on the variables, linear constraints and smooth nonlinear constraints) using a sequential quadratic programming (SQP) method. As many first derivatives as possible should be supplied by you; any unspecified derivatives are approximated by finite differences. It is not intended for large sparse problems.

e04wd may also be used for unconstrained, bound-constrained and linearly constrained optimization.

e04wd uses forward communication for evaluating the objective function, the nonlinear constraint functions, and any of their derivatives.

The initialization function e04wc must have been called before to calling e04wd.

**Usage**

e04wd(a, bl, bu, confun, objfun, istate, ccon, cjac, clamda, h, x, optlist, n = nrow(x),
nclin = nrow(a),
ncnln = nrow(cjac))

**Arguments**

- `a` double array
  The \( i \)th row of a contains the \( i \)th row of the matrix \( A_L \) of general linear constraints in eqn1. That is, the \( i \)th row contains the coefficients of the \( i \)th general linear constraint for \( i = 1 \ldots nclin \).

- `bl` double array

- `bu` double array

Bl must contain the lower bounds and bu the upper bounds for all the constraints, in the following order. The first \( n \) elements of each array must contain the bounds on the variables, the next \( n_L \) elements the bounds for the general linear constraints (if any) and the next \( n_N \) elements the bounds for the general nonlinear constraints (if any). To specify a nonexistent lower bound (i.e., \( l_j = -\infty \)), set \( bl[j] \leq -\text{bigbnd} \), and to specify a nonexistent upper bound (i.e., \( u_j = +\infty \)), set \( bu[j] \geq \text{bigbnd} \); where \text{bigbnd} is the optional argument infiniteboundsize.

To specify the \( j \)th constraint as an equality, set \( bl[j] = bu[j] = \beta \), say, where \( \text{abs}(\beta) < \text{bigbnd} \).
function
confun must calculate the vector $c(x)$ of nonlinear constraint functions and (optionally) its Jacobian, $\frac{\partial c}{\partial x}$, for a specified $n$-vector $x$. If there are no nonlinear constraints (i.e., $ncnln = 0$), e04wd will never call confun, so it may be the dummy function e04wdp. (e04wdp is included in the NAG Library). If there are nonlinear constraints, the first call to confun will occur before the first call to objfun.

\[(MODE, CCON, CJAC) = \text{confun}(\text{mode}, ncnln, n, needc, x, cjac, nstate)\]

function
objfun must calculate the objective function $F(x)$ and (optionally) its gradient $g(x) = \frac{\partial F}{\partial x}$ for a specified $n$-vector $x$.

\[(MODE, OBJF, GRAD) = \text{objfun}(\text{mode}, n, x, grad, nstate)\]

istate
integer array
Is an integer array that need not be initialized if e04wd is called with the coldstart option (the default).

ccon
double array
Ccon need not be initialized if the (default) optional argument coldstart is used.

cjac
double array
In general, cjac need not be initialized before the call to e04wd. However, if $\text{derivative level} = 2, 3$, any constant elements of cjac may be initialized. Such elements need not be reassigned on subsequent calls to confun.

clamda
double array
Need not be set if the (default) optional argument coldstart is used.

h
double array
$H$ need not be initialized if the (default) optional argument coldstart is used, and will be set to the identity.

x
double array
An initial estimate of the solution.

optlist
options list
Optional parameters may be listed, as shown in the following table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Central Difference Interval</td>
<td>double</td>
<td>Default = $\epsilon \frac{1}{r}$</td>
</tr>
<tr>
<td>Check Frequency</td>
<td>integer</td>
<td>Default = 60</td>
</tr>
<tr>
<td>Cold Start</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Warm Start</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Crash Option</td>
<td>integer</td>
<td>Default = 3</td>
</tr>
<tr>
<td>Crash Tolerance</td>
<td>double</td>
<td>Default = 0.1</td>
</tr>
<tr>
<td>Defaults</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Derivative Level</td>
<td>integer</td>
<td>Default = 3</td>
</tr>
<tr>
<td>Derivative Linesearch</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nonderivative Linesearch</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Difference Interval</td>
<td>double</td>
<td>Default = $\sqrt{\epsilon}$</td>
</tr>
<tr>
<td>Dump File</td>
<td>integer</td>
<td>Default = 0</td>
</tr>
<tr>
<td>Load File</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Elastic Weight</td>
<td>double</td>
<td>Default = $10^4$</td>
</tr>
<tr>
<td>Expand Frequency</td>
<td>integer</td>
<td>Default = 10000</td>
</tr>
</tbody>
</table>
Factorization Frequency
Function Precision
Hessian Full Memory
Hessian Limited Memory
Hessian Frequency
Hessian Updates
Infinite Bound Size
Iterations Limit
Linesearch Tolerance
Nolist
List
LU Density Tolerance
LU Singularity Tolerance
LU Factor Tolerance
LU Update Tolerance
LU Partial Pivoting
LU Complete Pivoting
LU Rook Pivoting
Major Feasibility Tolerance
Major Optimality Tolerance
Major Iterations Limit
Major Print Level
Major Step Limit
Minimize
Maximize
Feasible Point
Minor Feasibility Tolerance
Feasibility Tolerance
Minor Iterations Limit
Minor Print Level
New Basis File
Backup Basis File
Save Frequency
New Superbasics Limit
Old Basis File
Partial Price
Pivot Tolerance
Print File
Print Frequency
Proximal Point Method
Punch File
Insert File
QPSolver Cholesky
QPSolver CG
QPSolver QN
Reduced Hessian Dimension
Scale Option
Scale Tolerance
Scale Print
Solution File
Start Objective Check At Variable
Stop Objective Check At Variable

**Integer Values:**
- Default = 50
- Default = 99999999
- Default = max (1000010max (nnL + nN))
- Default = min (2000n)

**Double Values:**
- Default = e^{0.8}
- Default = max (10^{-6}\sqrt{\varepsilon})
- Default = max (10^{-6}\sqrt{\varepsilon})
- Default = max (10003max (nnL + nN))

**Other:**
- Default = max \{10^{-6}\sqrt{\varepsilon}\}
- Default = max \{10^{-6}\sqrt{\varepsilon}\}
- Default = max (10003max (nnL + nN))
Start Constraint Check At Variable integer Default = 1
Stop Constraint Check At Variable integer Default = n
Summary File integer Default = 0
Summary Frequency integer Default = 100
Superbasics Limit integer Default = n
Suppress Parameters
System Information No
System Information Yes
Timing Level integer Default = 0
Unbounded Objective double Default = 1.0E+15
Unbounded Step Size double Default = bigbnd
Verify Level integer Default = 0
Violation Limit double Default = 1.0E+6

\[ n \] integer: default = nrow(x)
   \( n \), the number of variables.

\[ n_{\text{c}} \] integer: default = nrow(a)
   \( n_{\text{c}} \), the number of general linear constraints.

\[ n_{\text{n}} \] integer: default = nrow(cjac)
   \( n_{\text{n}} \), the number of nonlinear constraints.

Details
R interface to the NAG Fortran routine E04WDF.

Value

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAJITS</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>ISTATE</td>
<td>integer array</td>
<td></td>
</tr>
<tr>
<td>CCON</td>
<td>double array</td>
<td></td>
</tr>
<tr>
<td>CJAC</td>
<td>double array</td>
<td></td>
</tr>
<tr>
<td>CLAMDA</td>
<td>double array</td>
<td></td>
</tr>
<tr>
<td>OBJF</td>
<td>double</td>
<td></td>
</tr>
</tbody>
</table>

MAJITS: The number of major iterations performed.

ISTATE: Describes the status of the constraints \( l \leq r(x) \leq u \). For the \( j \)th lower or upper bound, \( j = 1, 2, \ldots, n + n_{\text{c}} + n_{\text{f}} \), the possible values of istate[\( j \)] are as follows (see the figure in the Fortran library documentation). \( \delta \) is the appropriate feasibility tolerance.

CCON: double array
If \( n_{\text{f}} > 0 \), ccon[\( i \)] contains the value of the \( i \)th nonlinear constraint function \( c_{i} \) at the final iterate for \( i = 1 \ldots n_{\text{f}} \).

CJAC: double array
If \( n_{\text{f}} > 0 \), cjac contains the Jacobian matrix of the nonlinear constraint functions at the final iterate, i.e., cjac[\( i, j \)] contains the partial derivative of the \( i \)th constraint function with respect to the \( j \)th variable for \( j = 1 \ldots n \) for \( i = 1 \ldots n_{\text{f}} \). (See the discussion of argument cjac under confun.)

CLAMDA: double array
The values of the QP multipliers from the last QP subproblem. clamda[\( j \)] should be non-negative if istate[\( j \)] = 1 and non-positive if istate[\( j \)] = 2.

OBJF: double
The value of the objective function at the final iterate.
**Author(s)**

NAG

**References**

http://www.nag.co.uk/numeric/FL/nagdoc_fl23/pdf/E04/e04wdf.pdf

**Examples**

```r
optlist <- list()

ifail <- 0
confun = function(mode, ncnln, n, needc, x, cjac, nstate) {
  ldcj <- nrow(cjac)
  ccon <- as.matrix(mat.or.vec(ncnln, 1))
  if (nstate == 1) {
    cjac <- as.matrix(mat.or.vec(ncnln, n))
  }
  if (needc[1] > 0) {
    if (mode == 0 || mode == 2) {
    }
    if (mode == 1 || mode == 2) {
      cjac[1, 1] <- 2 * x[1]
      cjac[1, 2] <- 2 * x[2]
      cjac[1, 3] <- 2 * x[3]
      cjac[1, 4] <- 2 * x[4]
    }
  }
}
```
if (needc[2] > 0) {
  if (mode == 0 || mode == 2) {
  }
  if (mode == 1 || mode == 2) {
  }
}
list(MODE = as.integer(mode), CCON = as.matrix(ccon), CJAC = as.matrix(cjac))
}

objfun = function(mode, n, x, grad, nstate) {
  if (mode == 0 || mode == 2) {
  }
  if (mode == 1 || mode == 2) {
  }
  list(MODE = as.integer(mode), OBJF = objf, GRAD = as.matrix(grad))
}

a <- matrix(c(1, 1, 1, 1), nrow = 1, ncol = 4, byrow = TRUE)

bl <- matrix(c(1, 1, 1, 1, -1e+25, -1e+25, 25), nrow = 7, ncol = 1, byrow = TRUE)

bu <- matrix(c(5, 5, 5, 5, 20, 40, 1e+25), nrow = 7,
ncol = 1, byrow = TRUE)

istate <- as.matrix(mat.or.vec(7, 1))
ccon <- as.matrix(mat.or.vec(2, 1))
cjac <- as.matrix(mat.or.vec(2, 4))
clamda <- as.matrix(mat.or.vec(7, 1))
h <- as.matrix(mat.or.vec(4, 4))
x <- matrix(c(1, 5, 5, 1), nrow = 4, ncol = 1, byrow = TRUE)
e04wd(a, bl, bu, confun, objfun, istate, ccon, cjac,
       clamda, h, x, optlist)

---

**e04xa**

**e04xa: Estimate (using numerical differentiation) gradient and/or Hessian of a function**

**Description**

*e04xa* computes an approximation to the gradient vector and/or the Hessian matrix for use in conjunction with, or following the use of an optimization function (such as *e04uf*).

**Usage**

```r
e04xa(msglvl, epsrf, x, mode, objfun, hforw, lwsav, iwsav, rwsav, 
n = nrow(x))
```

**Arguments**

- **msglvl**  
  integer  
  Must indicate the amount of intermediate output desired (see the printed output description in the Fortran library documentation for a description of the printed output). All output is written on the current advisory message unit (see *x04ab*).

- **epsrf**  
  double  
  Must define $\epsilon_R$, which is intended to be a measure of the accuracy with which the problem function $F$ can be computed. The value of $\epsilon_R$ should reflect the relative precision of $1 + \text{abs}(F(x))$, i.e., acts as a relative precision when $\text{abs}(F)$ is large, and as an absolute precision when $\text{abs}(F)$ is small. For example, if $F(x)$ is typically of order 1000 and the first six significant digits are known to be correct, an appropriate value for $\epsilon_R$ would be $1.0 \times 10^{-6}$.

- **x**  
  double array  
  The point $x$ at which the derivatives are to be computed.
mode integer
Indicates which derivatives are required.
mode = 0: The gradient and Hessian diagonal values having supplied the objective function via objfun.
mode = 1: The Hessian matrix having supplied both the objective function and gradients via objfun.
mode = 2: The gradient values and Hessian matrix having supplied the objective function via objfun.

objfun function
If mode = 0, 2, objfun must calculate the objective function; otherwise if mode = 1, objfun must calculate the objective function and the gradients.
(MODE,OBJF,OBJGRD) = objfun(mode,n,x,nstate)

hforw double array
The initial trial interval for computing the appropriate partial derivative to the jth variable.
lwsav boolean array
iwsav integer array
rwsav double array
These arguments are no longer required by e04xa.
n integer: default = nrow(x)
The number n of independent variables.

Details
R interface to the NAG Fortran routine E04XAF.

Value

MODE integer
Is changed only if you set mode negative in objfun, i.e., you have requested termination of e04xa.

HFORW double array
hforw[j] is the best interval found for computing a forward-difference approximation to the appropriate partial derivative for the jth variable.

OBJF double
The value of the objective function evaluated at the input vector in x.

OBJGRD double array
If mode = 0, 2, objgrd[j] contains the best estimate of the first partial derivative for the jth variable.

HCNTRL double array
hcntrl[j] is the best interval found for computing a central-difference approximation to the appropriate partial derivative for the jth variable.

H double array
If mode = 0, the estimated Hessian diagonal elements are contained in the first column of this array.

IWARN integer
iwarn = 0 on successful exit.
INFO  
integer array
info[j] represents diagnostic information on variable j. (See the Errors section in Fortran library documentation for more details.)

IFAIL  
integer
ifail = 0 unless the function detects an error or a warning has been flagged (see the Errors section in Fortran library documentation).

Author(s)
NAG

References
http://www.nag.co.uk/numeric/FL/nagdoc_fl23/pdf/E04/e04xaf.pdf

Examples

```r
optlist <- list()
ifail <- 0
objfun <- function(mode, n, x, nstate) {

    objgrd <- as.matrix(mat.or.vec(n, 1))
    a <- x[1] + 10 %*% x[2]
    objf <- a^2 + 5 %*% b^2 + c^4 + 10 %*% d^4

    if (mode == 1) {
        }
        list(MODE = as.integer(mode), OBJF = objf, OBJGRD = as.matrix(objgrd))
    }

msglvl <- 0
epsrf <- -1
```
e04ya

x <- matrix(c(3, -1, 0, 1), nrow = 4, ncol = 1, byrow = TRUE)

mode <- 0
hforw <- matrix(c(-1, -1, -1, -1), nrow = 4, ncol = 1, byrow = TRUE)
lwsav <- as.matrix(mat.or.vec(120, 1))
iwsav <- as.matrix(mat.or.vec(610, 1))
rwsav <- as.matrix(mat.or.vec(475, 1))
e04xa(msglvl, epsrf, x, mode, objfun, hforw, lwsav, iwsav, rwsav)

e04ya: Check user’s function for calculating Jacobian of first derivatives

Description

e04ya checks that a user-supplied function for evaluating a vector of functions and the matrix of their first derivatives produces derivative values which are consistent with the function values calculated.

Usage

e04ya(m, lsqfun, x, 
n = nrow(x))

Arguments

m integer

lsqfun function

lsqfun must calculate the vector of values $f_i(x)$ and their first derivatives $\frac{\partial f_i}{\partial x_j}$ at any point $x$. (The minimization functions mentioned in the Description in Fortran library documentation give you the option of resetting a argument to terminate immediately. e04ya will also terminate immediately, without finishing the checking process, if the argument in question is reset.)

(IFLAG,FVEC,FJAC) = lsqfun(iflag,m,n,xc,ldfjac)

x double array

$x[j]$ for $j = 1 \ldots n$, must be set to the coordinates of a suitable point at which to check the derivatives calculated by lsqfun. ‘Obvious’ settings, such as 0 or 1, should not be used since, at such particular points, incorrect terms may take correct values (particularly zero), so that errors can go undetected. For a similar reason, it is preferable that no two elements of $x$ should have the same value.

n integer: default = nrow(x)

The number $m$ of residuals, $f_i(x)$, and the number $n$ of variables, $x_j$. 
Details

R interface to the NAG Fortran routine E04YAF.

Value

FVEC  
double array
Unless you set iflag negative in the first call of lsqfun, \( fvec[i] \) contains the value of \( f_i \) at the point supplied by you in \( x \) for \( i = 1 \ldots m \).

FJAC  
double array
Unless you set iflag negative in the first call of lsqfun, \( fjac[i,j] \) contains the value of the first derivative \( \frac{df_i}{dx_j} \) at the point given in \( x \), as calculated by lsqfun for \( j = 1 \ldots n \) for \( i = 1 \ldots m \).

IFAIL  
integer
\( ifail = 0 \) unless the function detects an error or a warning has been flagged (see the Errors section in Fortran library documentation).

Author(s)

NAG

References

http://www.nag.co.uk/numeric/FL/nagdoc_fl23/pdf/E04/e04yaf.pdf

Examples

```r
ifail <- 0
lsqfun = function(iflag, m, n, xc, ljc) {

  fvec <- as.matrix(mat.or.vec(m, 1))
  fjacc <- as.matrix(mat.or.vec(ljc, n))
  for (i in c(1:m)) {
    if (iflag != 1) {
      fvec[i] <- xc[1] + t[i, 1]/denom - y[i]
    }
    if (iflag != 0) {
      fjacc[i, 1] <- 1
      dummy <- -1/(denom %*% denom)
      fjacc[i, 2] <- t[i, 1] %*% t[i, 2] %*% dummy
      fjacc[i, 3] <- t[i, 1] %*% t[i, 3] %*% dummy
    }
  }
  list(IFLAG = as.integer(iflag), FVEC = as.matrix(fvec), FJAC = as.matrix(fjacc))
}
```
\( m \leftarrow 15 \)
\( x \leftarrow \text{matrix}(c(0.19, -1.34, 0.88), \text{nrow} = 3, \text{ncol} = 1, \text{byrow} = \text{TRUE}) \)

\( iw \leftarrow \text{as.matrix(mat.or.vec(0, 0))} \)
\( w \leftarrow \text{as.matrix(mat.or.vec(69, 1))} \)
\( y \leftarrow \text{matrix}(c(0.14, 0.18, 0.22, 0.25, 0.29, 0.32, 0.35, 0.39, 0.37, 0.58, 0.73, 0.96, 1.34, 2.1, 4.39), \text{nrow} = 1, \text{ncol} = 15, \text{byrow} = \text{TRUE}) \)

\( t \leftarrow \text{matrix}(c(1, 15, 1, 2, 14, 2, 3, 13, 3, 4, 12, 4, 5, 11, 5, 6, 10, 6, 7, 9, 7, 8, 8, 9, 7, 7, 10, 6, 6, 11, 5, 5, 12, 4, 4, 13, 3, 3, 14, 2, 2, 15, 1, 1), \text{nrow} = 15, \text{ncol} = 3, \text{byrow} = \text{TRUE}) \)

\( \text{e04ya}(m, \text{lsqfun}, x) \)

---

**e04yb**

**e04yb: Check user’s function for calculating Hessian of a sum of squares**

**Description**

\( \text{e04yb} \) checks that a user-supplied function for evaluating the second derivative term of the Hessian matrix of a sum of squares is consistent with a user-supplied function for calculating the corresponding first derivatives.

**Usage**

\( \text{e04yb}(m, \text{lsqfun}, \text{lsqhes}, x, \text{lb}, iw, w, n = \text{nrow}(x)) \)

**Arguments**

- \( m \) integer
- \( \text{lsqfun} \) function
  \( \text{lsqfun} \) must calculate the vector of values \( f_i(x) \) and their first derivatives \( \frac{\partial f_i}{\partial x_j} \) at any point \( x \). (\text{e04he} gives you the option of resetting arguments of \text{lsqfun} to cause the minimization process to terminate immediately. \text{e04yb} will also terminate immediately, without finishing the checking process, if the argument in question is reset.)

\((\text{IFLAG}, FVEC, FJAC) = \text{lsqfun}(\text{iflag}, m, n, xc, ldfjac)\)
function

lsqhes must calculate the elements of the symmetric matrix

\[ B(x) = \sum_{i=1}^{m} f_i(x) G_i(x), \]

at any point \( x \), where \( G_i(x) \) is the Hessian matrix of \( f_i(x) \). (As with lsqfun, an argument can be set to cause immediate termination.)

\((\text{IFLAG}, B) = \text{lsqhes} (\text{iflag}, m, n, \text{fvec}, xc, lb)\)

\( x[j] \) for \( j = 1 \ldots n \), must be set to the coordinates of a suitable point at which to check the \( b_{jk} \) calculated by lsqhes. 'Obvious' settings, such as 0 or 1, should not be used since, at such particular points, incorrect terms may take correct values (particularly zero), so that errors could go undetected. For a similar reason, it is preferable that no two elements of \( x \) should have the same value.

\( \text{lb} \)

integer

\( \text{iw} \)

integer array

This array appears in the argument list purely so that, if e04yb is called by another library function, the library function can pass quantities to functions lsqfun and lsqhes via iw. iw is not examined or changed by e04yb. In general you must provide an array iw, but are advised not to use it. integer array

\( \text{w} \)

double array

The actual length of w as declared in the function from which e04yb is called. double array

The actual length of w as declared in the function from which e04yb is called.

\( \text{n} \)

integer: \textbf{default} = \text{nrow}(x)

The number \( m \) of residuals, \( f_i(x) \), and the number \( n \) of variables, \( x_j \).

Details

R interface to the NAG Fortran routine E04YBF.

Value

\( \text{FVEC} \)

double array

Unless you set iflag negative in the first call of lsqfun, \( fvec[i] \) contains the value of \( f_i \) at the point supplied by you in \( x \) for \( i = 1 \ldots m \).

\( \text{FJAC} \)

double array

Unless you set iflag negative in the first call of lsqfun, \( fjac[i, j] \) contains the value of the first derivative \( \frac{\partial f_i}{\partial x_j} \) at the point given in \( x \), as calculated by lsqfun for \( j = 1 \ldots n \) for \( i = 1 \ldots m \).

\( \text{B} \)

double array

Unless you set iflag negative in lsqhes, \( b[j \times (j - 1) / 2 + k] \) contains the value of \( b_{jk} \) at the point given in \( x \) as calculated by lsqhes for \( k = 1 \ldots j \) for \( j = 1 \ldots n \).
This array appears in the argument list purely so that, if e04yb is called by another library function, the library function can pass quantities to functions lsqfun and lsqhes via iw. iw is not examined or changed by e04yb. In general you must provide an array iw, but are advised not to use it.

Author(s)
NAG

References
http://www.nag.co.uk/numeric/FL/nagdoc_fl23/pdf/E04/e04ybf.pdf

Examples

```r
ifail <- 0
lsqfun = function(iflag, m, n, xc, ljc) {
  fvec <- as.matrix(mat.or.vec(m, 1))
  fjacc <- as.matrix(mat.or.vec(ljc, n))
  for (i in c(1:m)) {
    fvec[i] <- xc[1] + t[i, 1]/denom - y[i]
    if (iflag != 0) {
      fjacc[i, 1] <- 1
      dummy <- -1/(denom %*% denom)
      fjacc[i, 2] <- t[i, 1] %*% t[i, 2] %*% dummy
      fjacc[i, 3] <- t[i, 1] %*% t[i, 3] %*% dummy
    }
  }
  list(IFLAG = as.integer(iflag), FVEC = as.matrix(fvec), FJAC = as.matrix(fjacc))
}
lsqhes = function(iflag, m, n, fvec, xc, lb) {
```
```r
b <- as.matrix(mat.or.vec(lb, 1))
sum22 <- 0
sum32 <- 0
sum33 <- 0
for (i in c(1:m)) {
  sum22 <- sum22 + fvec[i] %*% dummy %*% t[i, 2]^2
  sum32 <- sum32 + fvec[i] %*% dummy %*% t[i, 2] %*% t[i, 3]
  sum33 <- sum33 + fvec[i] %*% dummy %*% t[i, 3]^2
}
b[3] <- sum22
b[5] <- sum32
b[6] <- sum33
list(IFLAG = as.integer(iflag), B = as.matrix(b))
}

m <- 15
x <- matrix(c(0.19, -1.34, 0.88), nrow = 3, ncol = 1, byrow = TRUE)
lb <- 6
iw <- as.matrix(mat.or.vec(1, 1))
w <- as.matrix(mat.or.vec(78, 1))
y <- matrix(c(0.14, 0.18, 0.22, 0.25, 0.29, 0.32, 0.35, 0.39, 0.37, 0.58, 0.73, 0.96, 1.34, 2.1, 4.39), nrow = 1, ncol = 15, byrow = TRUE)
t <- matrix(c(1, 15, 1, 2, 14, 2, 3, 13, 3, 4, 12, 4, 5, 11, 5, 6, 10, 6, 7, 9, 7, 8, 8, 9, 7, 10, 6, 6, 11, 5, 12, 4, 4, 13, 3, 14, 2, 2, 15, 1, 1), nrow = 15, ncol = 3, byrow = TRUE)
e04yb(m, lsqfun, lsqhes, x, lb, iw, w)
```

---

**e04yc**

*e04yc*: Covariance matrix for nonlinear least squares problem (unconstrained)
Description

e04yc returns estimates of elements of the variance-covariance matrix of the estimated regression coefficients for a nonlinear least squares problem. The estimates are derived from the Jacobian of the function $f(x)$ at the solution.

This function may be used following any one of the nonlinear least squares functions e04fc e04fy e04gb e04gy e04gd e04gz e04he e04hy.

Usage

e04yc(job, m, fsumsq, s, v,
   n = nrow(s))

Arguments

job
integer
Which elements of $C$ are returned as follows:

job $= -1$: The $n$ by $n$ symmetric matrix $C$ is returned.

job $= 0$: The diagonal elements of $C$ are returned.

job $> 0$: The elements of column job of $C$ are returned.

m
integer
The number $m$ of observations (residuals $f_i(x)$).

fsumsq
double
The sum of squares of the residuals, $F(\bar{x})$, at the solution $\bar{x}$, as returned by the nonlinear least squares function.

s
double array
The $n$ singular values of the Jacobian as returned by the nonlinear least squares function. See the Description in Fortran library documentation for information on supplying $s$ following one of the easy-to-use functions.

v
double array
The $n$ by $n$ right-hand orthogonal matrix (the right singular vectors) of $J$ as returned by the nonlinear least squares function. See the Description in Fortran library documentation for information on supplying $v$ following one of the easy-to-use functions.

n
integer: default $= \text{nrow}(s)$
The number $n$ of variables ($x_j$).

Details

R interface to the NAG Fortran routine E04YCF.

Value

V
double array
If job $\geq 0$, v is unchanged.

CJ
double array
If job $= 0$, cj returns the $n$ diagonal elements of $C$.

IFAIL
integer
ifail = 0 unless the function detects an error or a warning has been flagged (see the Errors section in Fortran library documentation).
e05jb

Author(s)
NAG

References
http://www.nag.co.uk/numeric/FL/nagdoc_fl23/pdf/E04/e04ycf.pdf

Examples

```r
ifail <- 0
job <- 0
m <- 15
fsumsq <- 0.00821487730657898
s <- matrix(c(4.096503460741, 1.59495793805472, 0.0612584931217495),
           nrow = 3, ncol = 1, byrow = TRUE)

v <- matrix(c(0.935395908691802, 0.352951220949886,
              -0.0214459700788422, -0.259228425671719, 0.643234592093676,
              -0.72045116618536, -0.240489328924174, 0.679466478322564,
              0.693173995119214), nrow = 3, ncol = 3, byrow = TRUE)
e04yc(job, m, fsumsq, s, v)
```

e05jb

e05jb: Global optimization by multi-level coordinate search, simple bounds, using function values only

Description

e05jb is designed to find the global minimum or maximum of an arbitrary function, subject to simple bound-constraints using a multi-level coordinate search method. Derivatives are not required, but convergence is only guaranteed if the objective function is continuous in a neighbourhood of a global optimum. It is not intended for large problems.

The initialization function e05ja must have been called before calling e05jb.

Usage

e05jb(objfun, ibound, iinit, bl, bu, list, numpts, initpt, monit, optlist,
       n = nrow(bl),
       sdlist = ncol(list))
Arguments

**objfun**

Function

objfun must evaluate the objective function \( F(x) \) for a specified \( n \)-vector \( x \).

\[ (F, \text{INFORM}) = \text{objfun}(n, x, \text{nstate}) \]

**ibound**

Integer

Indicates whether the facility for dealing with bounds of special forms is to be used. ibound must be set to one of the following values.

- \( ibound = 0 \): You will supply \( \ell \) and \( u \) individually.
- \( ibound = 1 \): There are no bounds on \( x \).
- \( ibound = 2 \): There are semi-infinite bounds \( 0 \leq x \).
- \( ibound = 3 \): There are constant bounds \( \ell = \ell_1 \) and \( u = u_1 \).

**iinit**

Integer

Selects which initialization method to use.

- \( iinit = 0 \): Simple initialization (boundary and midpoint), with \( \text{numpts}[i] = 3 \), \( \text{initpt}[i] = 2 \) and \( \text{list}[i, j] = (\text{bl}[i] + \text{bu}[i]) / 2 \), for \( i = 1, 2, \ldots, n \) and \( j = 1, 2, 3 \).
- \( iinit = 1 \): Simple initialization (off-boundary and midpoint), with \( \text{numpts}[i] = 3 \), \( \text{initpt}[i] = 2 \) and \( \text{list}[i, j] = ((5\text{bl}[i] + \text{bu}[i]) / 6(\text{bl}[i] + \text{bu}[i]) / 2(\text{bl}[i] + 5\text{bu}[i]) / 6) \), for \( i = 1, 2, \ldots, n \) and \( j = 1, 2, 3 \).
- \( iinit = 2 \): Initialization using linesearches.
- \( iinit = 3 \): You are providing your own initialization list.
- \( iinit = 4 \): Generate a random initialization list.

**bl**

Double array

\( bl \) is \( \ell \), the array of lower bounds. \( bu \) is \( u \), the array of upper bounds.

**bu**

Double array

This argument need not be set on entry if you wish to use one of the preset initialization methods (\( iinit \neq 3 \)).

**list**

Double array

This argument need not be set on entry if you wish to use one of the preset initialization methods (\( iinit \neq 3 \)).

**numpts**

Integer array

This argument need not be set on entry if you wish to use one of the preset initialization methods (\( iinit \neq 3 \)).

**initpt**

Integer array

This argument need not be set on entry if you wish to use one of the preset initialization methods (\( iinit \neq 3 \)).

**monit**

Function

monit may be used to monitor the optimization process. It is invoked upon every successful completion of the procedure in which a sub-box is considered for splitting. It will also be called just before e05jb exits if that splitting procedure was not successful.

\[ \text{INFORM} = \text{monit}(n, \text{ncall}, x\text{best}, \text{icount}, \text{ninit}, \text{list}, \text{numpts}, \text{initpt}, \text{nbaskt}, x\text{baskt}, \text{boxl}, \text{boxu}, \text{nstate}) \]

**optlist**

Options list

Optional parameters may be listed, as shown in the following table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defaults</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Function Evaluations Limit  integer  Default = 100n^2_r
Infinite Bound Size    double  Default = \frac{1}{\max_r}
Local Searches         string  Default = ‘ON’
Local Searches Limit   integer  Default = 50
Local Searches Tolerance  double  Default = 2\epsilon
Minimize                Default
Maximize                Default
Nolist                  Default
List                    
Repeatability          string  Default = ‘OFF’
Splits Limit           integer  Default = \left\lfloor d(n_r + 2)/3 \right\rfloor
Static Limit           integer  Default = 3n_r
Target Objective Error double  Default = \epsilon^\frac{1}{4}
Target Objective Safeguard  double  Default = \epsilon^\frac{1}{2}
Target Objective Value  double

n  integer: default = nrow(bl)
n, the number of variables.

sdlist  integer: default = ncol(list)
sdlist is, at least, the maximum over i of the number of points in coordinate i at which to split according to the initialization list list; that is, sdlist \geq \max_i\numpts[i].

Details
R interface to the NAG Fortran routine E05JBF.

Value

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BL</td>
<td>double array</td>
</tr>
<tr>
<td>BU</td>
<td>double array</td>
</tr>
<tr>
<td>Unless ifail = 1, ifail = 2 on exit, bl and bu are the actual arrays of bounds used by e05jb.</td>
<td></td>
</tr>
<tr>
<td>LIST</td>
<td>double array</td>
</tr>
<tr>
<td>Unless ifail = 1, ifail = 2, ifail = −999 on exit, the actual initialization data used by e05jb. If you wish to monitor the contents of list you are advised to do so solely through monit, not through the output value here.</td>
<td></td>
</tr>
<tr>
<td>NUMPTS</td>
<td>integer array</td>
</tr>
<tr>
<td>Unless ifail = 1, ifail = 2, ifail = −999 on exit, the actual initialization data used by e05jb.</td>
<td></td>
</tr>
<tr>
<td>INITPT</td>
<td>integer array</td>
</tr>
<tr>
<td>Unless ifail = 1, ifail = 2, ifail = −999 on exit, the actual initialization data used by e05jb.</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>double array</td>
</tr>
<tr>
<td>If ifail = 0, contains an estimate of the global optimum (see also the Accuracy section in the Fortran library documentation).</td>
<td></td>
</tr>
<tr>
<td>OBJ</td>
<td>double</td>
</tr>
<tr>
<td>If ifail = 0, contains the function value at x.</td>
<td></td>
</tr>
</tbody>
</table>
IFAIL integer

ifail = 0 unless the function detects an error or a warning has been flagged (see the Errors section in Fortran library documentation).

Author(s)
NAG

References
http://www.nag.co.uk/numeric/FL/nagdoc_fl23/pdf/E05/e05jbf.pdf

Examples

optlist <- list()

ifail <- 0
peaks <- function(x1, x2) {
  f = 3 * (1 - x1)^2 * exp(-(x1^2) - (x2 + 1)^2) + 10 * (x1/5 -
  x1^3 - x2^5) * exp(-x1^2 - x2^2) + 1/3 * exp(-(x1 + 1)^2 -
  x1^2)
}

objective = function(n, x, nstate) {

  print(x[1])
  print(x[2])
  if (n == 2) {
    inform <- 0
  } else {
    inform <- -1
  }
  if (inform >= 0) {
    if (nstate == 1) {
      writeLines(toString(cat(sprintf("\n", "\n"))))
      writeLines(toString(cat(sprintf("OBJFUN was just called for the first time",
  "\n"))))
    }
    f <- peaks(x[1], x[2])
  } else {
    list(F = f, INFORM = as.integer(inform))
  }
}
monitor = function(n, ncall, xbest, icount, ninit, list, numpts, initpt, nbaskt, xbaskt, boxl, boxu, nstate) {
    inform <- 0
    if (nstate == 0 || nstate == 1) {
        writeLines(toString(cat(sprintf("\n", "\n"))))
        writeLines(toString(cat(sprintf("*** Begin monitoring information ***", "\n"))))
        writeLines(toString(cat(sprintf("\n", "\n"))))
    }
    if (nstate <= 0) {
        writeLines(toString(cat(sprintf("Total sub-boxes = %s", toString(icount[1]), "\n"))))
        writeLines(toString(cat(sprintf("Total function evaluations = %s", toString(ncall), "\n"))))
        writeLines(toString(cat(sprintf("Total function evaluations used in local searches = %s", toString(icount[2]), "\n"))))
        writeLines(toString(cat(sprintf("Total points used in local search = %s", toString(icount[3]), "\n"))))
        writeLines(toString(cat(sprintf("Total sweeps through levels = %s", toString(icount[4]), "\n"))))
        writeLines(toString(cat(sprintf("Total splits by init. list = %s", toString(icount[5]), "\n"))))
        writeLines(toString(cat(sprintf("Lowest level with nonsplit boxes = %s", toString(icount[6]), "\n"))))
        writeLines(toString(cat(sprintf("Number of candidate minima in the %s", "shopping basket%s", " = %s", toString(nbaskt), "\n"))))
        writeLines(toString(cat(sprintf("Shopping basket:", "\n"))))
    }
}
print(xbaskt)

writeLines(toString(cat(sprintf("\n", "\n"))))

writeLines(toString(cat(sprintf("*** End monitoring information ***", 
"\n"))))

writeLines(toString(cat(sprintf("\n", "\n"))))

})

list(INFORM = as.integer(inform))

prob <- "peaks"
xres <- 100
yres <- 100
bl <- matrix(c(-3, -3), nrow = 2, ncol = 1, byrow = TRUE)

bu <- -bl
fglob <- -6.55
xglob <- matrix(c(0.23, -1.63), nrow = 2, ncol = 1, byrow = TRUE)

n <- length(bl)
if (ifail == 0) {
  writeLines(toString(cat(sprintf("\n", "\n"))))

  writeLines(toString(cat(sprintf("Solve with no options or init.-list data", 
"\n"))))

  ibound <- 0
  iinit <- 0
  list <- as.matrix(mat.or.vec(n, 3))
  numpts <- as.matrix(mat.or.vec(n, 1))
  initpt <- as.matrix(mat.or.vec(n, 1))
ans <- e05jb(objective, ibound, iinit, bl, bu, list, numpts, 
        initpt, monitor, optlist)
bl <- ans$BL
bu <- ans$BU
list <- ans$LIST
numpts <- ans$NUMPTS
initpt <- ans$INITPT
x <- ans$X
obj <- ans$OBJ
ifail <- ans$IFAIL

ifail <- ans$IFAIL

writeLines(toString(cat(sprintf("e05jbno options exited with ifail = %s", 
                      toString(ifail), "\n"))))

if (ifail == 0) {
    writeLines(toString(cat(sprintf("xbest:\n", "\n"))))

    xbest <- ans$XBEST
    print(xbest)
    writeLines(toString(cat(sprintf("\n", "\n"))))

    obj <- ans$OBJ
    writeLines(toString(cat(sprintf("obj = %s", toString(obj), 
                        "\n")))))
}
writeLines(toString(cat(sprintf("\n", "\n"))))
writeLines(toString(cat(sprintf("Solve with options and init.-list data", 
                        "\n")))))

infbnd <- 1.1579E+077
iinit <- 3
list <- as.matrix(mat.or.vec(n, 3))
list[, 1] <- bl
list[, 3] <- bu
list[, 2] <- matrix(c(-1, 0), nrow = 2, ncol = 1, byrow = TRUE)

numpts <- 3 * matrix(1, n, 1)
initpt <- 2 * matrix(1, n, 1)
ans <- e05jb(objective, ibound, iinit, b1, bu, list, numpts, initpt, monitor, optlist)
ifail <- ans$IFAIL
writeLines(toString(cat(sprintf("e05jboptions exited with ifail = %s", toString(ifail), "\n"))))
if (ifail == 0) {
    writeLines(toString(cat(sprintf("xbest:\n"))))
    xbest <- ans$X
    print(xbest)
    writeLines(toString(cat(sprintf("\n"))))
    obj <- ans$OBJ
    writeLines(toString(cat(sprintf("obj = %s", toString(obj), "\n"))))
    }

---

f08fa

f08fa: Computes all eigenvalues and, optionally, eigenvectors of a real symmetric matrix

Description
f08fa computes all the eigenvalues and, optionally, all the eigenvectors of a real \( n \times n \) symmetric matrix \( A \).

Usage
f08fa(jobz, uplo, a, 
      n = nrow(a))

Arguments

jobz string
If \( \text{jobz} = 'N' \), compute eigenvalues only.

uplo string
If \( \text{uplo} = 'U' \), the upper triangular part of \( A \) is stored.
**f08fa**

**a**

double array

The \( n \) by \( n \) matrix \( A \).

See the Fortran Library documentation for a description of the storage layout for this array.

\( n \)

type: integer; **default** = nrow(a)

\( n \), the order of the matrix \( A \).

**Details**

R interface to the NAG Fortran routine F08FAF.

**Value**

\( A \)

double array

If \( jobz = 'V' \), then if \( IN = 0 \), \( A \) contains the orthonormal eigenvectors of the matrix \( A \).

\( \tilde{W} \)

double array

If \( IN = 0 \), the eigenvalues in ascending order.

\( INFO \)

integer

\( info = 0 \) unless the function detects an error (see the Errors section in Fortran library documentation).

**Author(s)**

NAG

**References**

http://www.nag.co.uk/numeric/FL/nagdoc_fl23/pdf/F08/f08faf.pdf

**Examples**

```r
jobz<-'Vectors'
uplo<-'Upper'

a<-matrix(c(1,2,3,4,0,2,3,4,0,3,4,0,0,0,0,4),nrow=4,ncol=4,byrow=TRUE)

f08fa(jobz,uplo,a)
```
g02aa

**g02aa**: Computes the nearest correlation matrix to a real square matrix, using the method of Qi and Sun

**Description**

*g02aa* computes the nearest correlation matrix, in the Frobenius norm, to a given square, input matrix.

**Usage**

```r
g02aa(g,
  n = nrow(g),
  errtol = 0.0,
  maxits = 0,
  maxit = 0)
```

**Arguments**

- **g**: double array
  
  $G$, the initial matrix.

- **n**: integer
  
  Default: `nrow(g)`
  
  The size of the matrix $G$.

- **errtol**: double
  
  Default: `0.0`
  
  The termination tolerance for the Newton iteration. If `errtol ≥ 0.0` then $n \times \sqrt{\text{machineprecision}}$ is used.

- **maxits**: integer
  
  Default: `0`
  
  Maxits specifies the maximum number of iterations used for the iterative scheme used to solve the linear algebraic equations at each Newton step.

- **maxit**: integer
  
  Default: `0`
  
  Specifies the maximum number of Newton iterations.

**Details**

R interface to the NAG Fortran routine G02AAF.

**Value**

- **G**: double array
  
  A symmetric matrix $\frac{1}{2} (G + G^T)$ with the diagonal set to 1.

- **X**: double array
  
  Contains the nearest correlation matrix.

- **ITER**: integer
  
  The number of Newton steps taken.

- **FEVAL**: integer
  
  The number of function evaluations of the dual problem.

- **NRMGRD**: double
  
  The norm of the gradient of the last Newton step.
IFAIL

integer

ifail = 0 unless the function detects an error or a warning has been flagged (see
the Errors section in Fortran library documentation).

Author(s)

NAG

References

http://www.nag.co.uk/numeric/FL/nagdoc_fl23/pdf/G02/g02aaf.pdf

Examples

ifail <- 0

g <- matrix(c(2, -1, 0, 0, -1, 2, -1, 0, 0, -1, 2,
             -1, 0, 0, -1, 2), nrow = 4, ncol = 4, byrow = TRUE)

errtol <- 1e-07

maxits <- 200

maxit <- 10

ans <- g02aa(g)

if (ifail == 0) {
    writeLines(sprintf("\n Nearest Correlation Matrix\n",
                   "\n")

x <- ans$X

print(x)

iter <- ans$ITER

writeLines(sprintf("\n Number of Newton steps taken: %d",
               iter))

feval <- ans$FEVAL

writeLines(sprintf(" Number of function evaluations: %d",
               feval))

nrmgrd <- ans$NRMGRD

if (nrmgrd > errtol) {
    writeLines(sprintf(" Norm of gradient of last Newton step: %6.4f",
}
g02ab: Computes the nearest correlation matrix to a real square matrix, augmented g02aa to incorporate weights and bounds

**Description**

g02ab computes the nearest correlation matrix, in the Frobenius norm or weighted Frobenius norm, and optionally with bounds on the eigenvalues, to a given square, input matrix.

**Usage**

```plaintext
g02ab(g, opt, alpha, w,
   n = nrow(w),
   errtol = 0.0,
   maxits = 0,
   maxit = 0)
```

**Arguments**

- **g** double array
  
  \( G \), the initial matrix.

- **opt** string
  
  Indicates the problem to be solved.

  - \( \text{opt} = 'A' \): The lower bound problem is solved.
  - \( \text{opt} = 'W' \): The weighted norm problem is solved.
  - \( \text{opt} = 'B' \): Both problems are solved.

- **alpha** double
  
  The value of \( \alpha \).

- **w** double array
  
  The square roots of the diagonal elements of \( W \), that is the diagonal of \( W^{1/2} \).

- **n** integer: default = nrow(w)
  
  The size of the matrix \( G \).

- **errtol** double: default = 0.0
  
  The termination tolerance for the Newton iteration. If \( \text{errtol} \leq 0.0 \) then \( n \times \sqrt{\text{machineprecision}} \) is used.

- **maxits** integer: default = 0
  
  Specifies the maximum number of iterations to be used by the iterative scheme to solve the linear algebraic equations at each Newton step.

- **maxit** integer: default = 0
  
  Specifies the maximum number of Newton iterations.
Details

R interface to the NAG Fortran routine G02ABF.

Value

- **G**: Double array
  - A symmetric matrix $\frac{1}{2} (G + G^T)$ with the diagonal set to $I$.

- **W**: Double array
  - If `opt = 'W', 'B'`, the array is scaled so $\max (W_i) = 1$ for $i = 1 \ldots n$.

- **X**: Double array
  - Contains the nearest correlation matrix.

- **ITER**: Integer
  - The number of Newton steps taken.

- **FEVAL**: Integer
  - The number of function evaluations of the dual problem.

- **NRMGRD**: Double
  - The norm of the gradient of the last Newton step.

- **IFAIL**: Integer
  - `ifail = 0` unless the function detects an error or a warning has been flagged (see the Errors section in Fortran library documentation).

Author(s)

NAG

References

http://www.nag.co.uk/numeric/FL/nagdoc_fl23/pdf/G02/g02abf.pdf

Examples

```r
ifail <- 0
opt <- "b"
alpha <- 0.02
g <- matrix(c(2, -1, 0, 0, -1, 2, -1, 0, 0, -1, 2, -1, 0, 0, -1, 2), nrow = 4, ncol = 4, byrow = TRUE)
w <- matrix(c(100, 20, 20, 20), nrow = 4, ncol = 1, byrow = TRUE)
errtol <- 1e-07
maxits <- 200
```
maxit <- 10
ans <- g02ab(g, opt, alpha, w)
if (ifail == 0) {
  writeLines(sprintf("\n Nearest Correlation Matrix\n", "\n"))
  x <- ans$X
  print(x)
  iter <- ans$ITER
  writeLines(sprintf("\n Number of Newton steps taken: %d\n", iter))
  feval <- ans$FEVAL
  writeLines(sprintf(" Number of function evaluations: %d\n", feval))
  alpha <- ans$ALPHA
  writeLines(sprintf(" \n\n Alpha: %30.3f\n", alpha))
}

---

**g02ae**

**g02ae**: Computes the nearest correlation matrix with k-factor structure to a real square matrix

**Description**

g02ae computes the factor loading matrix associated with the nearest correlation matrix with k-factor structure, in the Frobenius norm, to a given square, input matrix.

**Usage**

```r
> g02ae(g, k, 
  n = nrow(g),
  errtol = 0.0,
  maxit = 0)
```
Arguments

\( g \quad \text{double array} \)
\( G \), the initial matrix.

\( k \quad \text{integer} \)
\( k \), the number of factors and columns of \( X \).

\( n \quad \text{integer: default = nrow(g)} \)
\( n \), the size of the matrix \( G \).

\( \text{errtol} \quad \text{double: default = 0.0} \)
The termination tolerance for the projected gradient norm. See references for further details. If \( \text{errtol} \leq 0.0 \) then 0.01 is used. This is often a suitable default value.

\( \text{maxit} \quad \text{integer: default = 0} \)
Specifies the maximum number of iterations in the spectral projected gradient method.

Details

R interface to the NAG Fortran routine G02AEF.

Value

\( G \quad \text{double array} \)
A symmetric matrix \( \frac{1}{2} (G + G^T) \) with the diagonal elements set to unity.

\( X \quad \text{double array} \)
Contains the matrix \( X \).

\( \text{ITER} \quad \text{integer} \)
The number of steps taken in the spectral projected gradient method.

\( \text{FEVAL} \quad \text{integer} \)
The number of function evaluations.

\( \text{NRMPGD} \quad \text{double} \)
The norm of the projected gradient at the final iteration.

\( \text{IFAIL} \quad \text{integer} \)
\( \text{ifail = 0} \) unless the function detects an error or a warning has been flagged (see the Errors section in Fortran library documentation).

Author(s)

NAG

References

http://www.nag.co.uk/numeric/FL/nagdoc_fl23/pdf/G02/g02aef.pdf
Examples

```r
ifail <- 0
errtol <- 1e-07
g <- matrix(c(2, -1, 0, 0, -1, 2, -1, 0, 0, -1, 2, -1, 0, 0, -1, 2), nrow = 4, ncol = 4, byrow = TRUE)

k <- 2
maxits <- 200
maxit <- 10
ans <- g02ae(g, k)
if (ifail == 0) {
  writeLines(sprintf("\n Factor Loading Matrix x:\n", "\n"))

  x <- ans$X
  print(x)
  iter <- ans$ITER
  writeLines(sprintf("\n Number of Newton steps taken: %d\n", iter))

  feval <- ans$FEVAL
  writeLines(sprintf(" Number of function evaluations: %d\n", feval))
}
```

---

**NAGFWrappers**

Provides interfaces to NAG Fortran Library

**Description**

Provides interfaces to a selection of routines from the NAG Fortran Library

**Details**

Package: NAGFWrapper
**s17dc**

Type: Package  
Version: 22.0  
Date: 2011-06-01  
License: Artistic-2.0  
LazyLoad: yes

**Author(s)**

NAG  
Maintainer: NAG <support@nag.co.uk>

**References**

www.nag.co.uk

---

**s17dc**  
**s17dc**: Bessel functions $Y_{\nu} + a(z)$, real $a \geq 0$, complex $z$, $\nu = 0$, 1, 2, \ldots.

---

**Description**

s17dc returns a sequence of values for the Bessel functions $Y_{\nu+n}(z)$ for complex $z$, non-negative $\nu$ and $n = 0, 1, \ldots, N-1$, with an option for exponential scaling.

**Usage**

`s17dc(fnu, z, n, scal)`

**Arguments**

- **fnu**  
  double  
  $\nu$, the order of the first member of the sequence of functions.

- **z**  
  complex  
  $z$, the argument of the functions.

- **n**  
  integer  
  $N$, the number of members required in the sequence $Y_{\nu}(z), Y_{\nu+1}(z), \ldots, Y_{\nu+N-1}(z)$.

- **scal**  
  string  
  The scaling option.  
  $scal = 'U'$: The results are returned unscaled.  
  $scal = 'S'$: The results are returned scaled by the factor $e^{-\text{abs}(\text{Im}(z))}$.

**Details**

R interface to the NAG Fortran routine S17DCF.
Value

**CY**
- complex array
  - The \( N \) required function values: \( cy[i] \) contains \( J_{\nu+i-1}(z) \) for \( i = 1 \ldots N \).

**NZ**
- integer
  - The number of components of \( cy \) that are set to zero due to underflow. The positions of such components in the array \( cy \) are arbitrary.

**IFAIL**
- integer
  - ifail = 0 unless the function detects an error or a warning has been flagged (see the Errors section in Fortran library documentation).

Author(s)

NAG

References


Examples

```r
ifail<-0
fnu<-0
z<-complex(1,0.3,0.4)
n<-2
scal<-'U'
s17dc(fnu,z,n,scal)
```

---

**s17de**

**s17de**: Bessel functions \( J_{\nu} + a(z) \), real \( a \geq 0 \), complex \( z \), \( \nu = 0 \), 1, 2, . . .

Description

\( s17de \) returns a sequence of values for the Bessel functions \( J_{\nu+n}(z) \) for complex \( z \), non-negative \( \nu \) and \( n = 0, 1, \ldots, N - 1 \), with an option for exponential scaling.

Usage

```r
s17de(fnu, z, n, scal)
```
Arguments

- **fnu**: double
  
  $\nu$, the order of the first member of the sequence of functions.

- **z**: complex
  
  The argument $z$ of the functions.

- **n**: integer
  
  $N$, the number of members required in the sequence $J_\nu(z), J_{\nu+1}(z), \ldots, J_{\nu+N-1}(z)$.

- **scal**: string
  
  The scaling option.
  - `scal = 'U'`: The results are returned unscaled.
  - `scal = 'S'`: The results are returned scaled by the factor $e^{-\text{abs}(\text{Im}(z))}$.

Details

R interface to the NAG Fortran routine S17DEF.

Value

- **CY**: complex array
  
  The $N$ required function values: $cy[i]$ contains $J_{\nu+i-1}(z)$ for $i = 1 \ldots N$.

- **NZ**: integer
  
  The number of components of $cy$ that are set to zero due to underflow. If $nz > 0$, then elements $cy[n - nz + 1], cy[n - nz + 2], \ldots, cy[n]$ are set to zero.

- **IFAIL**: integer
  
  ifail = 0 unless the function detects an error or a warning has been flagged (see the Errors section in Fortran library documentation).

Author(s)

NAG

References


Examples

```r
ifail<-0
fnu<-0
z<-complex(1.0+0.3i,0.4)
n<-2
scal<-‘U’
s17de(fnu,z,n,scal)
```
Description

s17dg returns the value of the Airy function \( \text{Ai}(z) \) or its derivative \( \text{Ai}'(z) \) for complex \( z \), with an option for exponential scaling.

Usage

\[
s17dg(deriv, z, scal)
\]

Arguments

- deriv: string
  Specifies whether the function or its derivative is required.
  - If \( \text{deriv} = 'F' \), \( \text{Ai}(z) \) is returned.
  - If \( \text{deriv} = 'D' \), \( \text{Ai}'(z) \) is returned.

- z: complex
  The argument \( z \) of the function.

- scal: string
  The scaling option.
  - \( \text{scal} = 'U' \): The result is returned unscaled.
  - \( \text{scal} = 'S' \): The result is returned scaled by the factor \( e^{2z\sqrt{z}/3} \).

Details

R interface to the NAG Fortran routine S17DGF.

Value

- AI: complex
  The required function or derivative value.

- NZ: integer
  Indicates whether or not ai is set to zero due to underflow. This can only occur when \( \text{scal} = 'U' \).
  - \( nz = 0 \): ai is not set to zero.
  - \( nz = 1 \): ai is set to zero.

- IFAIL: integer
  \( \text{ifail} = 0 \) unless the function detects an error or a warning has been flagged (see the Errors section in Fortran library documentation).

Author(s)

NAG

References

http://www.nag.co.uk/numeric/FL/nagdoc_fl123/pdf/S/s17dgf.pdf
Examples

```r
ifail<-0
deriv<-'F'
z<-complex(1,0.3,0.4)
scal<-'U'
s17dg(deriv,z,scal)
```

Description

`s17dh` returns the value of the Airy function $Bi(z)$ or its derivative $Bi'(z)$ for complex $z$, with an option for exponential scaling.

Usage

```r
s17dh(deriv, z, scal)
```

Arguments

- `deriv` string
  - Specifies whether the function or its derivative is required.
  - `deriv = 'F'`: $Bi(z)$ is returned.
  - `deriv = 'D'`: $Bi'(z)$ is returned.

- `z` complex
  - The argument $z$ of the function.

- `scal` string
  - The scaling option.
  - `scal = 'U'`: The result is returned unscaled.
  - `scal = 'S'`: The result is returned scaled by the factor $e^{\text{abs}(\text{Re}(z^2 \sqrt{3}/z))}$.

Details

R interface to the NAG Fortran routine S17DHF.

Value

- `BI` complex
  - The required function or derivative value.

- `IFAIL` integer
  - `ifail = 0` unless the function detects an error or a warning has been flagged (see the Errors section in Fortran library documentation).
Author(s)
NAG

References

Examples

ifail<-0
deriv<-'F'

z<-complex(1,0.3,0.4)
scal<-'U'

s17dh(deriv,z,scal)

s17dl: Hankel functions $H_{\nu+j}(z)$, $j = 1, 2$, real $a \geq 0$, complex $z$, $\nu=0, 1, 2, \ldots$

Description

s17dl returns a sequence of values for the Hankel functions $H^{(1)}_{\nu+j}(z)$ or $H^{(2)}_{\nu+j}(z)$ for complex $z$, non-negative $\nu$ and $n = 0, 1, \ldots, N-1$, with an option for exponential scaling.

Usage

s17dl(m, fnu, z, n, scal)

Arguments

m
integer
The kind of functions required.
$m = 1$: The functions are $H^{(1)}_{\nu+j}(z)$.
$m = 2$: The functions are $H^{(2)}_{\nu+j}(z)$.

fnu
double
$\nu$, the order of the first member of the sequence of functions.

z
complex
The argument $z$ of the functions.

n
integer
$N$, the number of members required in the sequence $H^{(m)}_{\nu+j}(z)$, $H^{(m)}_{\nu+j+1}(z), \ldots, H^{(m)}_{\nu+j+N-1}(z)$.

scal
string
The scaling option.
$scal = 'U'$: The results are returned unscaled.
$scal = 'S'$: The results are returned scaled by the factor $e^{-iz}$ when $m = 1$, or by the factor $e^{iz}$ when $m = 2$. 
Details

R interface to the NAG Fortran routine S17DLF.

Value

<table>
<thead>
<tr>
<th>C Y</th>
<th>complex array</th>
</tr>
</thead>
<tbody>
<tr>
<td>The $N$ required function values: $c[y[i]]$ contains $H_{\nu+i-1}^{(m)}(z)$ for $i = 1 \ldots N$.</td>
<td></td>
</tr>
</tbody>
</table>

| N Z | integer |
| The number of components of $c[y]$ that are set to zero due to underflow. If $nz > 0$, then if $\text{Im}(z) > 0.0$ and $m = 1$, or $\text{Im}(z) < 0.0$ and $m = 2$, elements $c[y][1], c[y][2], \ldots, c[y][nz]$ are set to zero. In the complementary half-planes, nz simply states the number of underflows, and not which elements they are. |

| IFAIL | integer |
| ifail = 0 unless the function detects an error or a warning has been flagged (see the Errors section in Fortran library documentation). |

Author(s)

NAG

References


Examples

```r
ifail<-0
m<-1
fnu<-0
z<-complex(1,0.3,0.4)
n<-2
scal<-'U'

s17dl(m,fnu,z,n,scal)
```

Description

s18dc: Modified Bessel functions $K_{\nu+n}(z)$, real $a \geq 0$, complex $z$, $nu = 0, 1, 2, \ldots$. 

s18dc returns a sequence of values for the modified Bessel functions $K_{\nu+n}(z)$ for complex $z$, non-negative $\nu$ and $n = 0, 1, \ldots, N - 1$, with an option for exponential scaling.
Usage

\texttt{s18dc(fnu, z, n, scal)}

Arguments

\texttt{fnu} \hspace{1cm} \text{double} \\
\quad \nu, \ the \ order \ of \ the \ first \ member \ of \ the \ sequence \ of \ functions.

\texttt{z} \hspace{1cm} \text{complex} \\
\quad \text{The \ argument} \ z \ \text{of \ the \ functions.}

\texttt{n} \hspace{1cm} \text{integer} \\
\quad \text{\textit{N}, \ the \ number \ of \ members \ required \ in \ the \ sequence} \ K_\nu(z), K_{\nu+1}(z), \ldots, K_{\nu+N-1}(z).

\texttt{scal} \hspace{1cm} \text{string} \\
\quad \text{The \ scaling \ option.} \\
\quad \texttt{scal} = 'U': \ The \ results \ are \ returned \ unscaled. \\
\quad \texttt{scal} = 'S': \ The \ results \ are \ returned \ scaled \ by \ the \ factor \ e^z.

Details

R interface to the NAG Fortran routine S18DCF.

Value

\texttt{CY} \hspace{1cm} \text{complex array} \\
\quad \text{The} \ N \ \text{required \ function \ values:} \ cy[i] \ \text{contains} \ K_{\nu+i-1}(z) \ \text{for} \ i = 1 \ldots N.

\texttt{NZ} \hspace{1cm} \text{integer} \\
\quad \text{The \ number \ of \ components \ of \ cy \ that \ are \ set \ to \ zero \ due \ to \ underflow. \ If} \ nz > 0 \ \text{and} \ \text{Re}(z) \geq 0.0, \ \text{elements} \ cy[1], cy[2], \ldots, cy[nz] \ \text{are \ set \ to \ zero. \ If} \ \text{Re}(z) < 0.0, \ nz \ \text{simply \ states \ the \ number \ of \ underflows, \ and \ not \ which \ elements \ they \ are.}

\texttt{IFAIL} \hspace{1cm} \text{integer} \\
\quad \text{ifail} = 0 \ \text{unless \ the \ function \ detects \ an \ error \ or \ a \ warning \ has \ been \ flagged \ (see} \ \text{the \ Errors \ section \ in \ Fortran \ library \ documentation).}

Author(s)

NAG

References

\url{http://www.nag.co.uk/numeric/FL/nagdoc_fl23/pdf/S/s18dcf.pdf}

Examples

\texttt{ifail<-0} \\
\texttt{fnu<-0} \\
\texttt{z<-complex(1,0.3,0.4)} \\
\texttt{n<-2}
s18de

s18de: Modified Bessel functions \( I_{\nu} + a(z) \), real \( a \geq 0 \), complex \( z \), \( \nu = 0, 1, 2, \ldots \)

Description

s18de returns a sequence of values for the modified Bessel functions \( I_{\nu} + n(z) \) for complex \( z \), non-negative \( \nu \) and \( n = 0, 1, \ldots, N - 1 \), with an option for exponential scaling.

Usage

s18de(fnu, z, n, scal)

Arguments

fnu
double 
\( \nu \), the order of the first member of the sequence of functions.

z 
complex 
The argument \( z \) of the functions.

n 
integer 
\( N \), the number of members required in the sequence \( I_{\nu} (z), I_{\nu+1} (z), \ldots, I_{\nu+N-1} (z) \).

scal 
string 
The scaling option. 
\( scal = 'U' \): The results are returned unscaled. 
\( scal = 'S' \): The results are returned scaled by the factor \( e^{-\text{abs(Re}(z))} \).

Details

R interface to the NAG Fortran routine S18DEF.

Value

CY 
complex array 
The \( N \) required function values: \( cy[i] \) contains \( I_{\nu+i-1}(z) \) for \( i = 1 \ldots N \).

NZ 
integer 
The number of components of cy that are set to zero due to underflow.

IFAIL 
integer 
\( ifail = 0 \) unless the function detects an error or a warning has been flagged (see the Errors section in Fortran library documentation).

Author(s)

NAG
References


Examples

```r
ifail<-0
fnu<-0
z<-complex(1,0.3,-0.4)
n<-2
scal<-'U'
s18de(fnu,z,n,scal)
```

**s18gk**

*s18gk: Bessel function of the 1st kind J_alpha +/- n(z)*

**Description**

s18gk returns a sequence of values for the Bessel functions $J_{\alpha+n-1}(z)$ or $J_{\alpha-n+1}(z)$ for complex $z$, non-negative $\alpha < 1$ and $n = 1, 2, \ldots, \text{abs}(N) + 1$.

**Usage**

```r
s18gk(z, a, nl)
```

**Arguments**

- **z**
  - complex
  - The argument $z$ of the function.
- **a**
  - double
  - The order $\alpha$ of the first member in the required sequence of function values.
- **nl**
  - integer
  - The value of $N$.

**Details**

R interface to the NAG Fortran routine S18GKF.

**Value**

- **B**
  - complex array
  - With ifail = 0, ifail = 3, the required sequence of function values: $b[n]$ contains $J_{\alpha+n-1}(z)$ if $nl \geq 0$ and $J_{\alpha-n+1}(z)$ otherwise for $n = 1 \ldots \text{abs}(nl) + 1$.
- **IFAIL**
  - integer
  - ifail = 0 unless the function detects an error or a warning has been flagged (see the Errors section in Fortran library documentation).
Author(s)
NAG

References

Examples

```r
ifail<-0
z<-complex(1,0.6,-0.8)
a<-0
nl<-3
s18gk(z,a,nl)
```

Description

s22aa returns a sequence of values for either the unnormalized or normalized Legendre functions of the first kind $P_n^m(x)$ or $\overline{P}_n^m(x)$ for real $x$ of a given order $m$ and degree $n = 0, 1, \ldots, N$.

Usage

`s22aa(mode, x, m, nl)`

Arguments

- **mode**: integer
  Indicates whether the sequence of function values is to be returned unnormalized or normalized.
  - $mode = 1$: The sequence of function values is returned unnormalized.
  - $mode = 2$: The sequence of function values is returned normalized.

- **x**: double
  The argument $x$ of the function.

- **m**: integer
  The order $m$ of the function.

- **nl**: integer
  The degree $N$ of the last function required in the sequence.

Details

R interface to the NAG Fortran routine S22AAF.
**Value**

P  
   double array  
The required sequence of function values as follows:  
   if \( \text{mode} = 1 \), \( p[n] \) contains \( P_m^n(x) \) for \( n = 0 \ldots N \);  
   if \( \text{mode} = 2 \), \( p[n] \) contains \( T_m^n(x) \) for \( n = 0 \ldots N \).

IFAIL  
   integer  
   ifail = 0 unless the function detects an error or a warning has been flagged (see the Errors section in Fortran library documentation).

**Author(s)**

NAG

**References**


**Examples**

```r
ifail<-0
mode<-1
x<-0.5
m<-2
nl<-3
s22aa(mode,x,m,nl)
```

---

**Description**

\texttt{x02aj} returns \( \epsilon \), the value machine precision.

**Usage**

\texttt{x02aj()}

**Details**

R interface to the NAG Fortran routine X02AJF.

**Value**

\texttt{x02aj} returns \( \epsilon \), the value machine precision.
\textit{x02al: The largest positive model number}

\section*{Description}
\textit{x02al} returns the largest positive floating point number.

\section*{Usage}
\texttt{x02al()}

\section*{Details}
R interface to the NAG Fortran routine X02ALF.

\section*{Value}
\textit{x02al} returns the largest positive floating point number.

\section*{Author(s)}
NAG

\section*{References}
http://www.nag.co.uk/numeric/FL/nagdoc_fl23/pdf/X02/x02ajf.pdf

\section*{Examples}
\texttt{x02aj}()["result"]

\texttt{x02al}()["result"]
Index

*Topic Fortran
  NAGFWrappers, 140

*Topic NAG
  NAGFWrappers, 140

*Topic math
  a00ad, 2
  e04ab, 5
  e04bb, 7
  e04cb, 9
  e04dg, 11
  e04fc, 13
  e04fy, 17
  e04gd, 18
  e04gy, 22
  e04gz, 24
  e04hc, 26
  e04hd, 28
  e04he, 30
  e04hy, 34
  e04jc, 37
  e04jy, 40
  e04kd, 42
  e04ky, 46
  e04kz, 49
  e04lb, 51
  e04ly, 56
  e04mf, 58
  e04nc, 61
  e04nf, 64
  e04nk, 68
  e04nq, 73
  e04uc, 79
  e04uf, 85
  e04ug, 93
  e04us, 101
  e04vj, 106
  e04wd, 109
  e04xa, 115
  e04ya, 118
  e04yb, 120
  e04yc, 123
  e05jb, 125
  f08fa, 132

  g02aa, 134
  g02ab, 136
  g02ae, 138
  s17dc, 141
  s17de, 142
  s17dg, 144
  s17dh, 145
  s17dl, 146
  s18dc, 147
  s18de, 149
  s18gk, 150
  s22aa, 151
  x02aj, 152
  x02al, 153

*Topic optimize
  e04ab, 5
  e04bb, 7
  e04cb, 9
  e04dg, 11
  e04fc, 13
  e04fy, 17
  e04gd, 18
  e04gy, 22
  e04gz, 24
  e04hc, 26
  e04hd, 28
  e04he, 30
  e04hy, 34
  e04jc, 37
  e04jy, 40
  e04kd, 42
  e04ky, 46
  e04kz, 49
  e04lb, 51
  e04ly, 56
  e04mf, 58
  e04nc, 61
  e04nf, 64
  e04nk, 68
  e04nq, 73
  e04uc, 79
  e04uf, 85
  e04ug, 93
INDEX

NAGFWrappers, 140

a00ad, 2

e04ab, 5
e04bb, 7
e04cb, 9
e04dg, 11
e04fc, 13
e04fy, 17
e04gd, 18
e04gy, 22
e04gz, 24
e04hc, 26
e04hd, 28
e04he, 30
e04hy, 34
e04jc, 37
e04jy, 40
e04kd, 42
e04ky, 46
e04kz, 49
e04lb, 51
e04ly, 55
e04mf, 58
e04nc, 61
e04nf, 64
e04nk, 68
e04ng, 73
e04uc, 79
e04uf, 85
e04ug, 93
e04us, 101
e04vj, 106
e04wd, 109
e04xa, 115
e04ya, 118
e04yb, 120
e04yc, 123
e05jb, 125

f08fa, 132

g02aa, 134

g02ab, 136
g02ae, 138

NAGFWrappers, 140

s17dc, 141
s17de, 142
s17dg, 144
s17dh, 145
s17dl, 146
s18dc, 147
s18de, 149
s18gk, 150
s22aa, 151
x02aj, 152
x02al, 153