NAG Library Routine Document

H02DAF

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details.

1 Purpose

H02DAF solves general nonlinear programming problems with integer constraints on some of the variables.

2 Specification

```
SUBROUTINE H02DAF (N, NCLIN, NCNLN, A, LDA, D, AX, BL, BU, VARCON, X,      &
                   CONFUN, C, CJAC, OBJFUN, OBJGRD, MAXIT, ACC, OBJMIP,    &
                   IOPTS, OPTS, IUSER, RUSER, IFAIL)
INTEGER            N, NCLIN, NCNLN, LDA, VARCON(N+NCLIN+NCNLN), MAXIT,     &
                   IOPTS(*), IUSER(*), IFAIL
REAL (KIND=nag_wp) A(LDA,*), D(NCLIN), AX(NCLIN), BL(N), BU(N), X(N),      &
                   C(NCNLN), CJAC(NCNLN,N), OBJGRD(N), ACC, OBJMIP,        &
                   OPTS(*), RUSER(*)
EXTERNAL           CONFUN, OBJFUN
```

Before calling H02DAF, H02ZKF **must** be called with OPTSTR set to '`Initialize = h02daf`'. Optional parameters may also be specified by calling H02ZKF before the call to H02DAF.

3 Description

H02DAF solves mixed integer nonlinear programming problems using a modified sequential quadratic programming method. The problem is assumed to be stated in the following general form:

$$\begin{array}{ll} \underset{x\in\{R^{n_c},Z^{n_i}\}}{\text{minimize}} & f(x) \\ \text{subject to} & c_j(x) = 0, \quad j = 1,2,\ldots,m_e \\ & c_j(x) \geq 0, \quad j = m_e+1, m_e+2,\ldots,m \\ & l \leq x_i \leq u, \quad i = 1,2,\ldots,n \end{array}$$

with n_c continuous variables and n_i binary and integer variables in a total of n variables; m_e equality constraints in a total of m constraint functions.

Partial derivatives of $f(x)$ and $c(x)$ are not required for the n_i integer variables. Gradients with respect to integer variables are approximated by difference formulae.

No assumptions are made regarding $f(x)$ except that it is twice continuously differentiable with respect to continuous elements of x. It is not assumed that integer variables are relaxable. In other words, problem functions are evaluated only at integer points.

The method seeks to minimize the exact penalty function:

$$P_\sigma(x) = f(x) + \sigma\|g(x)\|_\infty$$

where σ is adapted by the algorithm and $g(x)$ is given by:

$$\begin{array}{rcll} g(x) & = & c_j(x), & j = 1,2,\ldots,m_e \\ & = & \min\big(c_j(x),0\big), & j = m_e+1, m_e+2,\ldots,m. \end{array}$$

Successive quadratic approximations are applied under the assumption that integer variables have a smooth influence on the model functions, that is function values do not change drastically when incrementing or decrementing an integer value. In practice this requires integer variables to be ordinal not categorical. The algorithm is stabilised by a trust region method including Yuan's second order corrections, see Yuan and Sun (2006). The Hessian of the Lagrangian function is approximated by BFGS (see Section 11.4 in E04UCF/E04UCA) updates subject to the continuous and integer variables.

The mixed-integer quadratic programming subproblems of the SQP-trust region method are solved by a branch and cut method with continuous subproblem solutions obtained by the primal-dual method of Goldfarb and Idnani, see Powell (1983). Different strategies are available for selecting a branching variable:

Maximal fractional branching. Select an integer variable from the relaxed subproblem solution with largest distance from next integer value

Minimal fractional branching. Select an integer variable from the relaxed subproblem solution with smallest distance from next integer value

and a node from where branching, that is the generation of two new subproblems, begins:

Best of two. The optimal objective function values of the two child nodes are compared and the node with a lower value is chosen

Best of all. Select an integer variable from the relaxed subproblem solution with the smallest distance from the next integer value

Depth first. Select a child node whenever possible.

This implementation is based on the algorithm MISQP as described in Exler *et al.* (2013).

Linear constraints may optionally be supplied by a matrix A and vector d rather than the constraint functions $c(x)$ such that

$$Ax = d \quad \text{or} \quad Ax \geq d.$$

Partial derivatives with respect to x of these constraint functions are not requested by H02DAF.

4 References

Exler O, Lehmann T and Schittkowski K (2013) A comparative study of SQP-type algorithms for nonlinear and nonconvex mixed-integer optimization *Mathematical Programming Computation* **4** 383–412

Mann A (1986) GAMS/MINOS: Three examples *Department of Operations Research Technical Report* Stanford University

Powell M J D (1983) On the quadratic programming algorithm of Goldfarb and Idnani *Report DAMTP 1983/Na 19* University of Cambridge, Cambridge

Yuan Y-x and Sun W (2006) *Optimization Theory and Methods* Springer–Verlag

5 Parameters

1: N – INTEGER *Input*

On entry: n, the total number of variables, $n_c + n_i$.

Constraint: N > 0.

2: NCLIN – INTEGER *Input*

On entry: n_l, the number of general linear constraints defined by A and d.

Constraint: NCLIN ≥ 0.

3: NCNLN – INTEGER *Input*

On entry: n_N, the number of constraints supplied by $c(x)$.

Constraint: NCNLN ≥ 0.

4: A(LDA, $*$) – REAL (KIND=nag_wp) array *Input*

Note: the second dimension of the array A must be at least N if NCLIN > 0.

On entry: the ith row of A must contain the coefficients of the ith general linear constraint, for $i = 1, 2, \ldots, n_l$. Any equality constraints must be specified first.

If NCLIN $= 0$, the array A is not referenced.

5: LDA – INTEGER *Input*

On entry: the first dimension of the array A as declared in the (sub)program from which H02DAF is called.

Constraint: LDA $\geq \max(1, \text{NCLIN})$.

6: D(NCLIN) – REAL (KIND=nag_wp) array *Input*

On entry: d_i, the constant for the ith linear constraint.

If NCLIN $= 0$, the array D is not referenced.

7: AX(NCLIN) – REAL (KIND=nag_wp) array *Output*

On exit: the final values of the linear constraints Ax.

If NCLIN $= 0$, AX is not referenced.

8: BL(N) – REAL (KIND=nag_wp) array *Input*

9: BU(N) – REAL (KIND=nag_wp) array *Input*

On entry: BL must contain the lower bounds, l_i, and BU the upper bounds, u_i, for the variables; bounds on integer variables are rounded, bounds on binary variables need not be supplied.

Constraint: BL$(i) \leq$ BU(i), for $i = 1, 2, \ldots, \text{N}$.

10: VARCON(N + NCLIN + NCNLN) – INTEGER array *Input*

On entry: VARCON indicates the nature of each variable and constraint in the problem. The first n elements of the array must describe the nature of the variables, the next n_L elements the nature of the general linear constraints (if any) and the next n_N elements the general constraints (if any).

VARCON$(j) = 0$
: A continuous variable.

VARCON$(j) = 1$
: A binary variable.

VARCON$(j) = 2$
: An integer variable.

VARCON$(j) = 3$
: An equality constraint.

VARCON$(j) = 4$
: An inequality constraint.

Constraints:

VARCON$(j) = 0$, 1 or 2, for $j = 1, 2, \ldots, \text{N}$;
VARCON$(j) = 3$ or 4, for $j = \text{N} + 1, \ldots, \text{N} + \text{NCLIN} + \text{NCNLN}$;
At least one variable must be either binary or integer;
Any equality constraints must precede any inequality constraints.

11: X(N) – REAL (KIND=nag_wp) array *Input/Output*

On entry: an initial estimate of the solution, which need not be feasible. Values corresponding to integer variables are rounded; if an initial value less than half is supplied for a binary variable the value zero is used, otherwise the value one is used.

On exit: the final estimate of the solution.

12: CONFUN – SUBROUTINE, supplied by the NAG Library or the user. *External Procedure*

CONFUN must calculate the constraint functions supplied by $c(x)$ and their Jacobian at x. If all constraints are supplied by A and d (i.e., NCNLN $= 0$), CONFUN will never be called by H02DAF and CONFUN may be the dummy routine H02DDM. (H02DDM is included in the NAG Library.) If NCNLN > 0, the first call to CONFUN will occur after the first call to OBJFUN.

The specification of CONFUN is:

```
SUBROUTINE CONFUN (MODE, NCNLN, N, VARCON, X, C, CJAC, NSTATE,      &
                   IUSER, RUSER)

INTEGER            MODE, NCNLN, N, VARCON(*), NSTATE, IUSER(*)
REAL (KIND=nag_wp) X(N), C(NCNLN), CJAC(NCNLN,N), RUSER(*)
```

1: MODE – INTEGER *Input/Output*

On entry: indicates which values must be assigned during each call of OBJFUN. Only the following values need be assigned:

MODE $= 0$
Elements of C containing continuous variables.

MODE $= 1$
Elements of CJAC containing continuous variables.

On exit: may be set to a negative value if you wish to terminate the solution to the current problem, and in this case H02DAF will terminate with IFAIL set to MODE.

2: NCNLN – INTEGER *Input*

On entry: the dimension of the array C and the first dimension of the array CJAC as declared in the (sub)program from which H02DAF is called. The number of constraints supplied by $c(x)$, n_N.

3: N – INTEGER *Input*

On entry: the second dimension of the array CJAC as declared in the (sub)program from which H02DAF is called. n, the total number of variables, $n_c + n_i$.

4: VARCON($*$) – INTEGER array *Input*

On entry: the array VARCON as supplied to H02DAF.

5: X(N) – REAL (KIND=nag_wp) array *Input*

On entry: the vector of variables at which the objective function and/or all continuous elements of its gradient are to be evaluated.

6: C(NCNLN) – REAL (KIND=nag_wp) array *Output*

On exit: must contain NCNLN constraint values, with the value of the jth constraint $c_j(x)$ in C(j).

7: CJAC(NCNLN, N) – REAL (KIND=nag_wp) array *Input/Output*

Note: the derivative of the ith constraint with respect to the jth variable, $\dfrac{\partial c_i}{\partial x_j}$, is stored in CJAC$(i, j)$.

On entry: continuous elements of CJAC are set to the value of NaN.

On exit: the ith row of CJAC must contain elements of $\dfrac{\partial c_i}{\partial x_j}$ for each continuous variable x_j.

8: NSTATE – INTEGER *Input*

On entry: if NSTATE $= 1$, H02DAF is calling CONFUN for the first time. This parameter setting allows you to save computation time if certain data must be read or calculated only once.

9: IUSER$(*)$ – INTEGER array *User Workspace*

10: RUSER$(*)$ – REAL (KIND=nag_wp) array *User Workspace*

CONFUN must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which H02DAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

13: C(NCNLN) – REAL (KIND=nag_wp) array *Output*

On exit: if NCNLN > 0, C(j) contains the value of the jth constraint function $c_j(x)$ at the final iterate, for $j = 1, 2, \ldots,$ NCNLN.

If NCNLN $= 0$, the array C is not referenced.

14: CJAC(NCNLN, N) – REAL (KIND=nag_wp) array *Output*

Note: the derivative of the ith constraint with respect to the jth variable, $\dfrac{\partial c_i}{\partial x_j}$, is stored in CJAC$(i, j)$.

On exit: if NCNLN > 0, CJAC contains the Jacobian matrix of the 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 $i = 1, 2, \ldots,$ NCNLN and $j = 1, 2, \ldots,$ N. (See the discussion of parameter CJAC under CONFUN.)

If NCNLN $= 0$, the array CJAC is not referenced.

15: OBJFUN – SUBROUTINE, supplied by the user. *External Procedure*

OBJFUN must calculate the objective function $f(x)$ and its gradient for a specified n-element vector x.

The specification of OBJFUN is:

```
SUBROUTINE OBJFUN (MODE, N, VARCON, X, OBJMIP, OBJGRD, NSTATE,      &
                   IUSER, RUSER)

INTEGER            MODE, N, VARCON(*), NSTATE, IUSER(*)
REAL (KIND=nag_wp) X(N), OBJMIP, OBJGRD(N), RUSER(*)
```

1: MODE – INTEGER *Input/Output*

On entry: indicates which values must be assigned during each call of OBJFUN. Only the following values need be assigned:

MODE $= 0$

The objective function value, OBJMIP.

MODE = 1
The continuous elements of OBJGRD.

On exit: may be set to a negative value if you wish to terminate the solution to the current problem, and in this case H02DAF will terminate with IFAIL set to MODE.

2: N – INTEGER *Input*

On entry: n, the total number of variables, $n_c + n_i$.

3: VARCON(∗) – INTEGER array *Input*

On entry: the array VARCON as supplied to H02DAF.

4: X(N) – REAL (KIND=nag_wp) array *Input*

On entry: the vector of variables at which the objective function and/or all continuous elements of its gradient are to be evaluated.

5: OBJMIP – REAL (KIND=nag_wp) *Output*

On exit: must be set to the objective function value, f, if MODE = 0; otherwise OBJMIP is not referenced.

6: OBJGRD(N) – REAL (KIND=nag_wp) array *Input/Output*

On entry: continuous elements of OBJGRD are set to the value of NaN.

On exit: must contain the gradient vector of the objective function if MODE = 1, with OBJGRD(j) containing the partial derivative of f with respect to continuous variable x_j; otherwise OBJGRD is not referenced.

7: NSTATE – INTEGER *Input*

On entry: if NSTATE = 1, H02DAF is calling OBJFUN for the first time. This parameter setting allows you to save computation time if certain data must be read or calculated only once.

8: IUSER(∗) – INTEGER array *User Workspace*
9: RUSER(∗) – REAL (KIND=nag_wp) array *User Workspace*

OBJFUN is called with the parameters IUSER and RUSER as supplied to H02DAF. You are free to use the arrays IUSER and RUSER to supply information to OBJFUN as an alternative to using COMMON global variables.

OBJFUN must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which H02DAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

16: OBJGRD(N) – REAL (KIND=nag_wp) array *Output*

On exit: the objective function gradient at the solution.

17: MAXIT – INTEGER *Input*

On entry: the maximum number of iterations within which to find a solution. If MAXIT is less than or equal to zero, the suggested value below is used.

Suggested value: MAXIT = 500.

18: ACC – REAL (KIND=nag_wp) *Input*

On entry: the requested accuracy for determining feasible points during iterations and for halting the method when the predicted improvement in objective function is less than ACC. If ACC is

less than or equal to ϵ (ϵ being the ***machine precision*** as given by X02AJF), the below suggested value is used.

Suggested value: ACC $= 0.0001$.

19: OBJMIP – REAL (KIND=nag_wp) *Output*

On exit: with IFAIL $= 0$, OBJMIP contains the value of the objective function for the MINLP solution.

20: IOPTS$(*)$ – INTEGER array *Communication Array*

Note: the dimension of this array is dictated by the requirements of associated functions that must have been previously called. This array **must** be the same array passed as argument IOPTS in the previous call to H02ZKF.

21: OPTS$(*)$ – REAL (KIND=nag_wp) array *Communication Array*

Note: the dimension of this array is dictated by the requirements of associated functions that must have been previously called. This array **must** be the same array passed as argument OPTS in the previous call to H02ZKF.

On entry: the real option array as returned by H02ZKF.

22: IUSER$(*)$ – INTEGER array *User Workspace*

23: RUSER$(*)$ – REAL (KIND=nag_wp) array *User Workspace*

IUSER and RUSER are not used by H02DAF, but are passed directly to CONFUN and OBJFUN and may be used to pass information to these routines as an alternative to using COMMON global variables.

24: IFAIL – INTEGER *Input/Output*

On entry: IFAIL must be set to 0, -1 or 1. If you are unfamiliar with this parameter you should refer to Section 3.3 in the Essential Introduction for details.

For environments where it might be inappropriate to halt program execution when an error is detected, the value -1 or 1 is recommended. If the output of error messages is undesirable, then the value 1 is recommended. Otherwise, if you are not familiar with this parameter, the recommended value is 0. **When the value -1 or 1 is used it is essential to test the value of IFAIL on exit.**

On exit: IFAIL $= 0$ unless the routine detects an error or a warning has been flagged (see Section 6).

6 Error Indicators and Warnings

If on entry IFAIL $= 0$ or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

Errors or warnings detected by the routine:

IFAIL $= 1$

On entry, N $= \langle value \rangle$.
Constraint: N > 0.

IFAIL $= 2$

On entry, NCLIN $= \langle value \rangle$.
Constraint: NCLIN ≥ 0.

IFAIL $= 3$

On entry, NCNLN $= \langle value \rangle$.
Constraint: NCNLN ≥ 0.

IFAIL $= 4$

On entry, LDA $= \langle value \rangle$ and NCLIN $= \langle value \rangle$.
Constraint: LDA $\geq$ NCLIN.

IFAIL $= 5$

On entry, BL($\langle value \rangle$) $>$ BU($\langle value \rangle$).
Constraint: BL(i) $\leq$ BU(i), for $i = 1, 2, \ldots,$ N.

IFAIL $= 6$

On entry, VARCON($\langle value \rangle$) $= \langle value \rangle$.
Constraint: VARCON(i) $= 0$, 1 or 2, for $i = 1, 2, \ldots,$ N.

IFAIL $= 7$

On entry, VARCON($\langle value \rangle$) $= \langle value \rangle$.
Constraint: VARCON(i) $= 3$ or 4, for $i =$ N $+ 1, \ldots,$ N $+$ NCLIN $+$ NCNLN.

IFAIL $= 8$

The supplied OBJFUN returned a NaN value.

IFAIL $= 9$

The supplied CONFUN returned a NaN value.

IFAIL $= 10$

On entry, the optional parameter arrays IOPTS and OPTS have either not been initialized or been corrupted.

IFAIL $= 11$

On entry, there are no binary or integer variables.

IFAIL $= 12$

On entry, linear equality constraints do not precede linear inequality constraints.

IFAIL $= 13$

On entry, nonlinear equality constraints do not precede nonlinear inequality constraints.

IFAIL $= 81$

One or more objective gradients appear to be incorrect.

IFAIL $= 91$

One or more constraint gradients appear to be incorrect.

IFAIL $= 1001$

On entry, MAXIT $= \langle value \rangle$. Exceeded the maximum number of iterations.

IFAIL $= 1002$

More than the maximum number of feasible steps without improvement in the objective function. If the maximum number of feasible steps is small, say less than 5, try increasing it. Optional parameter **Feasible Steps** $= \langle value \rangle$.

IFAIL $= 1003$

Penalty parameter tends to infinity in an underlying mixed-integer quadratic program; the problem may be infeasible. If σ is relatively low value, try a higher one, for example 10^{20}.
Optional parameter **Penalty** $= \langle value \rangle$.

IFAIL $= 1004$

Termination at an infeasible iterate; if the problem is feasible, try a different starting value.

IFAIL $= 1005$

Termination with zero integer trust region for integer variables; try a different starting value.
Optional parameter **Integer Trust Radius** $= \langle value \rangle$.

IFAIL $= 1008$

The optimization failed due to numerical difficulties. Set optional parameter **Print Level** $= 3$ for more information.

IFAIL < 0

The optimization halted because you set MODE negative in OBJFUN or MODE negative in CONFUN, to $\langle value \rangle$.

IFAIL $= -99$

An unexpected error has been triggered by this routine. Please contact NAG.

See Section 3.8 in the Essential Introduction for further information.

IFAIL $= -399$

Your licence key may have expired or may not have been installed correctly.

See Section 3.7 in the Essential Introduction for further information.

IFAIL $= -999$

Dynamic memory allocation failed.

See Section 3.6 in the Essential Introduction for further information.

7 Accuracy

Not applicable.

8 Parallelism and Performance

H02DAF is not threaded by NAG in any implementation.

H02DAF makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

Please consult the X06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this routine. Please also consult the Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

None.

10 Example

Select a portfolio of at most p assets from n available with expected return ρ, is fully invested and that minimizes

$$\begin{array}{lll} & x^{\mathrm{T}}\Sigma x & \\ \text{subject to} \quad r^{\mathrm{T}}x & = & \rho \\ \sum_{i=1}^{n} x_i & = & 1 \\ x_i & \leq & y_i \\ \sum_{i=1}^{n} y_i & \leq & p \\ x_i & \geq & 0 \\ y_i & = & 0 \quad \text{or} \quad 1 \end{array}$$

where

x is a vector of proportions of selected assets

y is an indicator variable that describes if an asset is in or out

r is a vector of mean returns

Σ is the covariance matrix of returns.

This example is taken from Mann (1986) with

$$\begin{array}{lll} r & = & \begin{pmatrix} 8 & 9 & 12 & 7 \end{pmatrix} \\ \Sigma & = & \begin{pmatrix} 4 & 3 & -1 & 0 \\ 3 & 6 & 1 & 0 \\ -1 & 1 & 10 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \\ p & = & 3 \\ \rho & = & 10. \end{array}$$

Linear constraints are supplied through both A and d, and CONFUN.

10.1 Program Text

```
!   H02DAF Example Program Text
!   Mark 25 Release. NAG Copyright 2014.

    Module h02dafe_mod

!     .. Implicit None Statement ..
      Implicit None
!     .. Accessibility Statements ..
      Private
      Public                                 :: confun, objfun
    Contains
      Subroutine objfun(mode,n,varcon,x,objmip,objgrd,nstate,iuser,ruser)

!       .. Use Statements ..
        Use nag_library, Only: nag_wp
!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (Out)     :: objmip
        Integer, Intent (Inout)              :: mode
        Integer, Intent (In)                 :: n, nstate
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Inout)   :: objgrd(n), ruser(*)
        Real (Kind=nag_wp), Intent (In)      :: x(n)
        Integer, Intent (Inout)              :: iuser(*)
```

```
        Integer, Intent (In)                :: varcon(*)
!       .. Executable Statements ..
        Continue

        If (mode==0) Then
!         Objective value
          objmip = x(1)*(4.0_nag_wp*x(1)+3.0_nag_wp*x(2)-x(3)) + &
            x(2)*(3.0_nag_wp*x(1)+6.0_nag_wp*x(2)+x(3)) + &
            x(3)*(x(2)-x(1)+10.0_nag_wp*x(3))
        Else
!         Objective gradients for continuous variables
          objgrd(1) = 8.0_nag_wp*x(1) + 6.0_nag_wp*x(2) - 2.0_nag_wp*x(3)
          objgrd(2) = 6.0_nag_wp*x(1) + 12.0_nag_wp*x(2) + 2.0_nag_wp*x(3)
          objgrd(3) = 2.0_nag_wp*(x(2)-x(1)) + 20.0_nag_wp*x(3)
          objgrd(4) = 0.0_nag_wp
        End If
        Return
      End Subroutine objfun

      Subroutine confun(mode,ncnln,n,varcon,x,c,cjac,nstate,iuser,ruser)

!       .. Use Statements ..
        Use nag_library, Only: nag_wp
!       .. Parameters ..
        Real (Kind=nag_wp), Parameter       :: eight = 8.0_nag_wp
        Real (Kind=nag_wp), Parameter       :: nine = 9.0_nag_wp
        Real (Kind=nag_wp), Parameter       :: seven = 7.0_nag_wp
        Real (Kind=nag_wp), Parameter       :: twelve = 12.0_nag_wp
        Real (Kind=nag_wp), Parameter       :: zero = 0.0_nag_wp
!       .. Scalar Arguments ..
        Integer, Intent (Inout)             :: mode
        Integer, Intent (In)                :: n, ncnln, nstate
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out)    :: c(ncnln)
        Real (Kind=nag_wp), Intent (Inout)  :: cjac(ncnln,n), ruser(*)
        Real (Kind=nag_wp), Intent (In)     :: x(n)
        Integer, Intent (Inout)             :: iuser(*)
        Integer, Intent (In)                :: varcon(*)
!       .. Local Scalars ..
        Real (Kind=nag_wp)                  :: rho
        Integer                             :: p
!       .. Intrinsic Procedures ..
        Intrinsic                           :: real
!       .. Executable Statements ..
        Continue

        If (mode==0) Then
!         Constraints
          p = iuser(1)
          rho = ruser(1)

!         Mean return rho:
          c(1) = eight*x(1) + nine*x(2) + twelve*x(3) + seven*x(4) - rho
!         Maximum of p assets in portfolio:
          c(2) = real(p,kind=nag_wp) - x(5) - x(6) - x(7) - x(8)
        Else
!         Jacobian
          cjac(1,1:4) = (/eight,nine,twelve,seven/)
!          c(2) does not include continuous variables which requires
!          that their derivatives are zero
          cjac(2,1:4) = zero
        End If

        Return
      End Subroutine confun
    End Module h02dafe_mod

    Program h02dafe

!     .. Use Statements ..
      Use nag_library, Only: h02daf, h02zkf, h02zlf, nag_wp, x04caf
```

```
      Use h02dafe_mod, Only: confun, objfun
!     .. Implicit None Statement ..
      Implicit None
!     .. Parameters ..
      Real (Kind=nag_wp), Parameter        :: bigish = 1.0E3_nag_wp
      Real (Kind=nag_wp), Parameter        :: one = 1.0_nag_wp
      Real (Kind=nag_wp), Parameter        :: zero = 0.0_nag_wp
      Integer, Parameter                   :: nout = 6
!     .. Local Scalars ..
      Real (Kind=nag_wp)                   :: acc, accqp, objmip
      Integer                              :: ifail, ivalue, lda, liopts,      &
                                              lopts, maxit, n, nclin, ncnln,   &
                                              optype
      Character (40)                       :: cvalue
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable      :: a(:,:), ax(:), bl(:), bu(:),     &
                                              c(:), cjac(:,:), d(:),           &
                                              objgrd(:), x(:)
      Real (Kind=nag_wp)                   :: opts(100), ruser(1)
      Integer                              :: iopts(200), iuser(1)
      Integer, Allocatable                 :: varcon(:)
!     .. Intrinsic Procedures ..
      Intrinsic                            :: size
!     .. Executable Statements ..
      Write (nout,*) 'H02DAF Example Program Results'
      Write (nout,*)

      n = 8
      nclin = 5
      ncnln = 2

      lda = nclin
      Allocate (a(lda,n),d(nclin),ax(nclin),bl(n),bu(n),varcon(n+nclin+ncnln), &
        x(n),c(ncnln),cjac(ncnln,n),objgrd(n))

!     Set variable types: continuous then binary
      varcon(1:4) = 0
      varcon(5:8) = 1

!     Set continuous variable bounds
      bl(1:4) = zero
      bu(1:4) = bigish

!     Bounds for binary variables need not be provided
      bl(5:8) = zero
      bu(5:8) = one

!     Set linear constraint, equality first
      varcon(n+1) = 3
      varcon(n+2:n+nclin) = 4

!     Set Ax=d then Ax>=d
      a(1:nclin,1:n) = zero
      a(1,1:4) = one
      a(2,(/1,5/)) = (/-one,one/)
      a(3,(/2,6/)) = (/-one,one/)
      a(4,(/3,7/)) = (/-one,one/)
      a(5,(/4,8/)) = (/-one,one/)
      d(1) = one
      d(2:5) = zero

!     Set constraints supplied by CONFUN, equality first
      varcon(n+nclin+1) = 3
      varcon(n+nclin+2) = 4

      liopts = size(iopts)
      lopts = size(opts)

!     Initialise communication arrays
      ifail = 0
      Call h02zkf('Initialize = H02DAF',iopts,liopts,opts,lopts,ifail)
```

```
!     Optimisation parameters
      maxit = 500
      acc = 1.0E-6_nag_wp

!     Initial estimate (binary variables need not be given)
      x(1:4) = one
      x(5:8) = zero

!     Portfolio parameters p and rho
      iuser(1) = 3
      ruser(1) = 10.0_nag_wp

      ifail = 0
      Call h02daf(n,nclin,ncnln,a,lda,d,ax,bl,bu,varcon,x,confun,c,cjac, &
        objfun,objgrd,maxit,acc,objmip,iopts,opts,iuser,ruser,ifail)

!     Results
      If (ifail==0) Then
        Call x04caf('G','N',n,1,x,n,'Final estimate:',ifail)

!       Query the accuracy of the mixed integer QP solver
        ifail = -1
        Call h02zlf('QP Accuracy',ivalue,accqp,cvalue,optype,iopts,opts,ifail)
        If (ifail==0) Then

          Write (nout,'(/1x,a,1x,g12.4)') &
            'Requested accuracy of QP subproblems', accqp
        End If
        Write (nout,'(1x,a,1x,g12.4)') 'Optimised value:', objmip
      Else
        Write (nout,'(/1x,a,i4/)') 'h02daf returns ifail = ', ifail
      End If
    End Program h02dafe
```

10.2 Program Data

None.

10.3 Program Results

```
 H02DAF Example Program Results

 Final estimate:
          1
 1   0.3750
 2   0.0000
 3   0.5250
 4   0.1000
 5   1.0000
 6   0.0000
 7   1.0000
 8   1.0000

 Requested accuracy of QP subproblems   0.1000E-09
 Optimised value:    2.925
```

11 Optional Parameters

This section can be skipped if you wish to use the default values for all optional parameters, otherwise, the following is a list of the optional parameters available and a full description of each optional parameter is provided in Section 11.1.

Branch Bound Steps

Branching Rule

Check Gradients

Continuous Trust Radius

Descent
Descent Factor
Feasible Steps
Improved Bounds
Integer Trust Radius
Maximum Restarts
Minor Print Level
Modify Hessian
Node Selection
Non Monotone
Objective Scale Bound
Penalty
Penalty Factor
Print Level
QP Accuracy
Scale Continuous Variables
Scale Objective Function
Warm Starts

11.1 Description of the Optional Parameters

For each option, we give a summary line, a description of the optional parameter and details of constraints.

The summary line contains:

the keywords;

a parameter value, where the letters a, i and r denote options that take character, integer and real values respectively.

All options accept the value DEFAULT in order to return single options to their default states.

Keywords and character values are case insensitive, however they must be separated by at least one space.

H02ZKF can be called to supply options, one call being necessary for each optional parameter. For example,

```
Call H02ZKF('Check Gradients = Yes', iopts, liopts, opts, lopts, ifail)
```

H02ZKF should be consulted for a full description of the method of supplying optional parameters.

For H02DAF the maximum length of the parameter CVALUE used by H02ZLF is 12.

Branch Bound Steps i Default $= 500$

Maximum number of branch-and-bound steps for solving the mixed integer quadratic problems.

Constraint: **Branch Bound Steps** > 1.

Branching Rule a Default $=$ Maximum

Branching rule for branch and bound search.

Branching Rule $=$ Maximum
Maximum fractional branching.

Branching Rule $=$ Minimum
Minimum fractional branching.

Check Gradients a Default = No

Perform an internal check of supplied objective and constraint gradients. It is advisable to set **Check Gradients** = Yes during code development to avoid difficulties associated with incorrect user-supplied gradients.

Continuous Trust Radius r Default = 10.0

Initial continuous trust region radius, Δ_0^c; the initial trial step $d \in R^{n_c}$ for the SQP approximation must satisfy $\|d\|_\infty \leq \Delta_0^c$.

Constraint: **Continuous Trust Radius** > 0.0.

Descent r Default = 0.05

Initial descent parameter, δ, larger values of δ allow penalty parameter σ to increase faster which can lead to faster convergence.

Constraint: $0.0 <$ **Descent** < 1.0.

Descent Factor r Default = 0.1

Factor for decreasing the internal descent parameter, δ, between iterations.

Constraint: $0.0 <$ **Descent Factor** < 1.0.

Feasible Steps i Default = 10

Maximum number of feasible steps without improvements, where feasibility is measured by $\|g(x)\|_\infty \leq \sqrt{\text{ACC}}$.

Constraint: **Feasible Steps** > 1.

Improved Bounds a Default = No

Calculate improved bounds in case of 'Best of all' node selection strategy.

Integer Trust Radius r Default = 10.0

Initial integer trust region radius, Δ_0^i; the initial trial step $e \in R^{n_i}$ for the SQP approximation must satisfy $\|e\|_\infty \leq \Delta_0^i$.

Constraint: **Integer Trust Radius** ≥ 1.0.

Maximum Restarts i Default = 2

Maximum number of restarts that allow the mixed integer SQP algorithm to return to a better solution. Setting a value higher than the default might lead to better results at the expense of function evaluations.

Constraint: $0 <$ **Maximum Restarts** ≤ 15.

Minor Print Level i Default = 0

Print level of the subproblem solver. Active only if **Print Level** $\neq 0$.

Constraint: $0 <$ **Minor Print Level** < 4.

Modify Hessian a Default = Yes

Modify the Hessian approximation in an attempt to get more accurate search directions. Calculation time is increased when the number of integer variables is large.

Node Selection a Default = Depth First

Node selection strategy for branch and bound.

Node Selection = Best of all
Large tree search; this method is the slowest as it solves all subproblem QPs independently.

Node Selection = Best of two
Uses warm starts and less memory.

Node Selection = Depth first
Uses more warm starts. If warm starts are applied, they can speed up the solution of mixed integer subproblems significantly when solving almost identical QPs.

Non Monotone i Default = 10

Maximum number of successive iterations considered for the non-monotone trust region algorithm, allowing the penalty function to increase.

Constraint: $0 <$ **Non Monotone** < 100.

Objective Scale Bound r Default = 1.0

When **Scale Objective Function** > 0 internally scale absolute function values greater than 1.0 or **Objective Scale Bound**.

Constraint: **Objective Scale Bound** > 0.0.

Penalty r Default = 1000.0

Initial penalty parameter, σ.

Constraint: **Penalty** ≥ 0.0.

Penalty Factor r Default = 10.0

Factor for increasing penalty parameter σ when the trust regions cannot be enlarged at a trial step.

Constraint: **Penalty Factor** > 1.0.

Print Level i Default = 0

Specifies the desired output level of printing.

Print Level = 0
No output.

Print Level = 1
Final convergence analysis.

Print Level = 2
One line of intermediate results per iteration.

Print Level = 3
Detailed information printed per iteration.

QP Accuracy r Default = 1.0E−10

Termination tolerance of the relaxed quadratic program subproblems.

Constraint: **QP Accuracy** > 0.0.

Scale Continuous Variables a Default = Yes

Internally scale continuous variables values.

Scale Objective Function i Default = 1

Internally scale objective function values.

Scale Objective Function = 0
No scaling.

Scale Objective Function = 1
Scale absolute values greater than **Objective Scale Bound**.

Warm Starts i Default $= 100$

Maximum number of warm starts within the mixed integer QP solver, see **Node Selection**.

Constraint: **Warm Starts** ≥ 0.
