nag_opt_lp (e04mfc)

1. Purpose

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

2. Specification

```
#include <nag.h>
#include <nage04.h>

void nag_opt_lp(Integer n, Integer nclin, double a[], Integer tda, double bl[],
                double bu[], double cvec[], double x[], double *objf,
                Nag_E04_Opt *options, Nag_Comm *comm, NagError *fail)
```

3. Description

nag_opt_lp is designed to solve linear programming (LP) problems of the form

$$\begin{array}{ll} \text{minimize} & c^T x \\ x \in R^n & \\ \text{subject to} & l \le \left\{ \begin{array}{c} x \\ Ax \end{array} \right\} \le u, \end{array}$$

where c is an n element vector and A is an m_{lin} by n matrix.

The routine allows the linear objective function to be omitted in which case a feasible point (FP) for the set of constraints is sought.

The constraints involving A are called the *general* constraints. Note that upper and lower bounds are specified for all the variables and for all the general constraints. An *equality* constraint can be specified by setting $l_i = u_i$. If certain bounds are not present, the associated elements of l or u can be set to special values that will be treated as $-\infty$ or $+\infty$. (See the description of the optional parameter **inf_bound** in Section 8.2).

The user must supply an initial estimate of the solution.

Details about the algorithm are described in Section 7, but it is not necessary to read this more advanced section before using nag_opt_lp.

4. Parameters

n

Input: n, the number of variables.
Constraint: $\mathbf{n} > 0$.

nclin

Input: m_{lin}, the number of general linear constraints.
Constraint: $\mathbf{nclin} \ge 0$.

a[nclin][tda]

Input: the ith row of **a** must contain the coefficients of the ith general linear constraint (the ith row of A), for $i = 1, 2, \ldots, m_{lin}$.
If $\mathbf{nclin} = 0$ then the array **a** is not referenced.

tda

Input: the second dimension of the array **a** as declared in the function from which nag_opt_lp is called.
Constraint: $\mathbf{tda} \ge \mathbf{n}$ if $\mathbf{nclin} > 0$.

bl[n+nclin]
bu[n+nclin]

Input: **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_{lin} elements the bounds for the general linear constraints (if any). To specify a non-existent lower bound (i.e., $l_j = -\infty$), set $\mathbf{bl}[j] \leq -\mathbf{inf\_bound}$, and to specify a non-existent upper bound (i.e., $u_j = +\infty$), set $\mathbf{bu}[j] \geq \mathbf{inf\_bound}$; here **inf_bound** is the value of the optional parameter **options.inf_bound**, whose default value is 10^{20} (see Section 8.2). To specify the jth constraint as an *equality*, set $\mathbf{bl}[j] = \mathbf{bu}[j] = \beta$, say, where $|\beta| <$ **inf_bound**.
Constraint: $\mathbf{bl}[j] \leq \mathbf{bu}[j]$, for $j = 0, 1, \ldots, \mathbf{n} + \mathbf{nclin} - 1$.

cvec[n]

Input: the coefficients of the objective function when the problem is of type **Nag_LP**. The problem type is specified by the optional parameter **prob** (see Section 8) and the values **Nag_LP** and **Nag_FP** represent linear programming problem and feasible point problem respectively. **Nag_LP** is the default problem type for nag_opt_lp.

If the problem type **Nag_FP** is specified then **cvec** is not referenced and a NULL pointer may be given.

x[n]

Input: an initial estimate of the solution.
Output: the point at which nag_opt_lp terminated. If **fail.code** = **NE_NOERROR**, **NW_SOLN_NOT_UNIQUE** or **NW_NOT_FEASIBLE**, **x** contains an estimate of the solution.

objf

Output: 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 **Nag_FP** and x is feasible, **objf** is set to zero.

options

Input/Output: a pointer to a structure of type Nag_E04_Opt whose members are optional parameters for nag_opt_lp. These structure members offer the means of adjusting some of the parameter values of the algorithm and on output will supply further details of the results. A description of the members of **options** is given below in Section 8. Some of the results returned in **options** can be used by nag_opt_lp to perform a 'warm start' if it is re-entered (see the member **start** in Section 8.2).

If any of these optional parameters are required then the structure **options** should be declared and initialised by a call to nag_opt_init (e04xxc) and supplied as an argument to nag_opt_lp. However, if the optional parameters are not required the NAG defined null pointer, `E04_DEFAULT`, can be used in the function call.

comm

Input/Output: structure containing pointers for user communication with an optional user defined printing function. See Section 8.3.1 for details. If the user does not need to make use of this communication feature then the null pointer `NAGCOMM_NULL` may be used in the call to nag_opt_lp.

fail

The NAG error parameter, see the Essential Introduction to the NAG C Library.
Users are recommended to declare and initialise **fail** and set **fail.print** = **TRUE** for this function. nag_opt_lp returns with **fail.code = NE_NOERROR** if x is a strong local minimizer, i.e., the reduced gradient is negligible and the Lagrange multipliers are optimal.

4.1. Description of Printed Output

Intermediate and final results are printed out by default. The level of printed output can be controlled by the user with the structure member **options.print_level** (see Section 8.2).The default print level of **Nag_Soln_Iter** provides a single line of output at each iteration and the final result. This section describes the default printout produced by nag_opt_lp.

The convention for numbering the constraints in the iteration results is that indices 1 to n refer to the bounds on the variables, and indices $n + 1$ to $n + m_{lin}$ refer to the general constraints. When the status of a constraint changes, the index of the constraint is printed, along with the designation `L` (lower bound), `U` (upper bound), `E` (equality), `F` (temporarily fixed variable) or `A` (artificial constraint).

The single line of intermediate results output on completion of each iteration gives:

`Itn` the iteration count.

`Jdel` the index of the constraint deleted from the working set. If `Jdel` is zero, no constraint was deleted.

`Jadd` the index of the constraint added to the working set. If `Jadd` is zero, no constraint was added.

`Step` the step taken along the computed search direction. If a constraint is added during the current iteration (i.e., `Jadd` is positive), `Step` will be the step to the nearest constraint. When the problem is of type **Nag_LP** the step can be greater than 1.0 during the optimality phase,

`Ninf` the number of violated constraints (infeasibilities). This will be zero during the optimality phase.

`Sinf/Obj` the value of the current objective function. If x is not feasible, `Sinf` gives a weighted sum of the magnitudes of constraint violations. If x is feasible, `Obj` is the value of the objective function. The output line for the final iteration of the feasibility phase (i.e., the first iteration for which `Ninf` is zero) will give the value of the true objective at the first feasible point.

During the optimality phase, the value of the objective function will be non-increasing. During the feasibility phase, the number of constraint infeasibilities will not increase until either a feasible point is found, or the optimality of the multipliers implies that no feasible point exists. Once optimal multipliers are obtained, the number of infeasibilities can increase, but the sum of infeasibilities will either remain constant or be reduced until the minimum sum of infeasibilities is found.

`Bnd` the number of simple bound constraints in the current working set.

`Lin` the number of general linear constraints in the current working set.

`Nart` the number of artificial constraints in the working set.

`Nrz` the dimension of the subspace in which the objective function is currently being minimized. The value of `Nrz` is the number of variables minus the number of constraints in the working set; i.e., $\texttt{Nrz} = n - (\texttt{Bnd} + \texttt{Lin} + \texttt{Nart})$.

`Norm Gz` the Euclidean norm of the reduced gradient. During the optimality phase, this norm will be approximately zero after a unit step.

The printout of the final result consists of:

`Varbl` the name (`V`) and index j, for $j = 1, 2, \ldots, n$ of the variable.

`State` the state of the variable (`FR` if neither bound is in the working set, `EQ` if a fixed variable, `LL` if on its lower bound, `UL` if on its upper bound, `TF` if temporarily fixed at its current value). If `Value` lies outside the upper or lower bounds by more than the feasibility tolerance, `State` will be `++` or `--` respectively.

`Value` the value of the variable at the final iteration.

`Lower bound` the lower bound specified for the variable.
(`None` indicates that **bl**$[j-1] \leq$ **-inf_bound**.)

`Upper bound` the upper bound specified for the variable.
(`None` indicates that **bu**$[j-1] \geq$ **inf_bound**.)

`Lagr mult` the value of the Lagrange multiplier for the associated bound constraint. This will be zero if `State` is `FR`. If x is optimal, the multiplier should be non-negative if `State` is `LL`, and non-positive if `State` is `UL`.

`Residual` the difference between the variable `Value` and the nearer of its bounds $\mathbf{bl}[j-1]$ and $\mathbf{bu}[j-1]$.

The meaning of the printout for general constraints is the same as that given above for variables, with 'variable' replaced by 'constraint', and with the following change in the heading:

`LCon` the name (`L`) and index j, for $j = 1, 2, \ldots, m_{lin}$ of the constraint.

5. Comments

A list of possible error exits and warnings from nag_opt_lp is given in Section 9. Scaling and accuracy are considered in Section 10.

6. Example 1

This example minimizes the function

$$-0.02x_1 - 0.2x_2 - 0.2x_3 - 0.2x_4 - 0.2x_5 + 0.04x_6 + 0.04x_7$$

subject to the bounds

$$\begin{aligned}
-0.01 &\le x_1 \le 0.01 \\
-0.10 &\le x_2 \le 0.15 \\
-0.01 &\le x_3 \le 0.03 \\
-0.04 &\le x_4 \le 0.02 \\
-0.10 &\le x_5 \le 0.05 \\
-0.01 &\le x_6 \\
-0.01 &\le x_7
\end{aligned}$$

and the general constraints

$$\begin{array}{rl}
& x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 = -0.13 \\
& 0.15x_1 + 0.04x_2 + 0.02x_3 + 0.04x_4 + 0.02x_5 + 0.01x_6 + 0.03x_7 \le -0.0049 \\
& 0.03x_1 + 0.05x_2 + 0.08x_3 + 0.02x_4 + 0.06x_5 + 0.01x_6 \le -0.0064 \\
& 0.02x_1 + 0.04x_2 + 0.01x_3 + 0.02x_4 + 0.02x_5 \le -0.0037 \\
& 0.02x_1 + 0.03x_2 + 0.01x_5 \le -0.0012 \\
-0.0992 \le & 0.70x_1 + 0.75x_2 + 0.80x_3 + 0.75x_4 + 0.80x_5 + 0.97x_6 \\
-0.003 \le & 0.02x_1 + 0.06x_2 + 0.08x_3 + 0.12x_4 + 0.02x_5 + 0.01x_6 + 0.97x_7 \le 0.002
\end{array}$$

The initial point, which is infeasible, is

$$x_0 = (-0.01,\ -0.03,\ 0.0,\ -0.01,\ -0.1,\ 0.02,\ 0.01)^T.$$

The computed solution (to five figures) is

$$x^* = (-0.01,\ -0.1,\ 0.03,\ 0.02,\ -0.067485,\ -0.0022801,\ -0.00023453)^T.$$

Four bound constraints and three general constraints are active at the solution.

This first example shows the simple use of nag_opt_lp where default values are used for all optional parameters. A second example showing the use of optional parameters is given in Section 13. There is one example program file, the main program of which calls both examples. The main program and example 1 are given below.

6.1. Program Text

```
/* nag_opt_lp (e04mfc) Example Program
 *
 * Copyright 1991 Numerical Algorithms Group.
 *
 * Mark 2, 1991.
 * Mark 6 revised, 2000.
 */

#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <nag_string.h>
#include <nage04.h>

static void ex1(void);
static void ex2(void);

#define MAXN 10
#define MAXLIN 7
#define MAXBND MAXN+MAXLIN

main(void)
{
  /* Two examples are called, ex1() uses the
   * default settings to solve a problem while
   * ex2() solves another problem with some
   * of the optional parameters set by the user.
   */

  Vprintf("e04mfc Example Program Results.\n");
  ex1();
  ex2();
  exit(EXIT_SUCCESS);
}

static void ex1()
{
  double x[MAXN], cvec[MAXN];
  double a[MAXLIN][MAXN];
  double bl[MAXBND], bu[MAXBND];
  double objf;
  Integer i, j, n, nbnd, tda, nclin;
  static NagError fail;

  Vprintf("\nExample 1: default options used.\n");
  Vscanf(" %*[^\n]"); /* Skip headings in data file */
  Vscanf(" %*[^\n]");

  fail.print = TRUE;

  /* Set the actual problem dimensions.
   * n = the number of variables.
   * nclin = the number of general linear constraints (may be 0).
   * nbnd = the number of variables + linear constraints
   */
  tda = MAXN;
  n = 7;
  nclin =  7;
  nbnd = n + nclin;

  /* cvec   = the objective function coefficients.
   * a      = the linear constraint matrix.
   * bl     = the lower bounds on  x  and  A*x.
   * bu     = the upper bounds on  x  and  A*x.
   * x      = the initial estimate of the solution.
   */
```

```
  /* Read the objective function coefficients */
  Vscanf(" %*[^\n]"); /* Skip heading in data file */
  for (i = 0; i < n; ++i)
    Vscanf("%lf",&cvec[i]);

  /* Read the linear constraint matrix A. */
  Vscanf(" %*[^\n]"); /* Skip heading in data file */
  for (i = 0; i < nclin; ++i)
    for (j = 0; j < n; ++j)
      Vscanf("%lf",&a[i][j]);

  /* Read the bounds. */
  nbnd = n + nclin;
  Vscanf(" %*[^\n]"); /* Skip heading in data file */
  for (i = 0; i < nbnd; ++i)
    Vscanf("%lf", &bl[i]);
  Vscanf(" %*[^\n]"); /* Skip heading in data file */
  for (i = 0; i < nbnd; ++i)
    Vscanf("%lf", &bu[i]);

  /* Read the initial estimate of x. */
  Vscanf(" %*[^\n]"); /* Skip heading in data file */
  for (i = 0; i < n; ++i)
    Vscanf("%lf",&x[i]);

  /* Solve the problem. */
  e04mfc(n, nclin, (double *)a, tda, bl, bu, cvec,
         x, &objf, E04_DEFAULT, NAGCOMM_NULL, &fail);

  if (fail.code != NE_NOERROR) exit(EXIT_FAILURE);
}    /* ex1 */
```

6.2. Program Data

```
e04mfc Example Program Data

Data for example 1.

Objective function coefficients
-0.02 -0.2 -0.2 -0.2 -0.2 0.04 0.04

Linear constraint matrix, A.
1.0  1.0  1.0  1.0  1.0  1.0  1.0
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
0.02 0.04 0.01 0.02 0.02 0.0  0.0
0.02 0.03 0.0  0.0  0.01 0.0  0.0
0.70 0.75 0.80 0.75 0.80 0.97 0.0
0.02 0.06 0.08 0.12 0.02 0.01 0.97

Lower bounds
-0.01 -0.1 -0.01 -0.04 -0.1 -0.01 -0.01
-0.13 -1.0e21 -1.0e21 -1.0e21 -1.0e21 -0.0992 -0.003

Upper bounds
 0.01  0.15 0.03  0.02  0.05 1.0e21 1.0e21
-0.13 -0.0049 -0.0064 -0.0037 -0.0012  1.0e21  0.002

Initial estimate of x
-0.01 -0.03 0.0 -0.01 -0.1 0.02 0.01
```

6.3. Program Results

```
e04mfc Example Program Results.

Example 1: default options used.

Parameters to e04mfc
--------------------

Linear constraints............    7    Number of variables...........    7

prob....................     Nag_LP    start...................  Nag_Cold
ftol....................  1.05e-08     reset_ftol..............         5
fcheck..................        50     crash_tol...............  1.00e-02
inf_bound...............  1.00e+20     inf_step................  1.00e+20
max_iter................        70     machine precision.......  1.11e-16
optim_tol...............  1.72e-13     min_infeas..............     FALSE
print_level......... Nag_Soln_Iter
outfile.................    stdout

Memory allocation:
state...................       Nag
ax......................       Nag     lambda..................       Nag

Results from e04mfc:
-------------------

  Itn Jdel  Jadd   Step    Ninf  Sinf/Obj    Bnd  Lin  Nart  Nrz  Norm Gz

   0   0     0    0.0e+00   3   1.0380e-01    3    4    0     0   0.00e+00
   1   9 U  13 L  4.1e-02   1   3.0000e-02    3    4    0     0   0.00e+00
   2  12 U   4 L  4.2e-02   0   3.5000e-02    4    3    0     0   0.00e+00
   3   3 L  14 L  1.9e-01   0   3.0902e-02    3    4    0     0   0.00e+00
   4  11 U  10 U  1.5e-01   0   2.9897e-02    3    4    0     0   0.00e+00
   5   4 L   3 U  3.7e-01   0   2.7257e-02    3    4    0     0   0.00e+00
   6  10 U   4 U  6.5e-01   0   2.4038e-02    4    3    0     0   0.00e+00
   7   5 L   2 L  4.6e+00   0   2.3596e-02    4    3    0     0   0.00e+00

Final solution:

  Varbl State    Value      Lower Bound  Upper Bound    Lagr Mult    Residual

 V   1    LL  -1.00000e-02  -1.0000e-02   1.0000e-02    3.301e-01   0.000e+00
 V   2    LL  -1.00000e-01  -1.0000e-01   1.5000e-01    1.438e-02   0.000e+00
 V   3    UL   3.00000e-02  -1.0000e-02   3.0000e-02   -9.100e-02   0.000e+00
 V   4    UL   2.00000e-02  -4.0000e-02   2.0000e-02   -7.661e-02   0.000e+00
 V   5    FR  -6.74853e-02  -1.0000e-01   5.0000e-02    0.000e+00   3.251e-02
 V   6    FR  -2.28013e-03  -1.0000e-02   None       0.000e+00   7.720e-03
 V   7    FR  -2.34528e-04  -1.0000e-02   None       0.000e+00   9.765e-03

   LCon  State    Value      Lower Bound  Upper Bound    Lagr Mult    Residual

 L   1    EQ  -1.30000e-01  -1.3000e-01  -1.3000e-01   -1.431e+00   0.000e+00
 L   2    FR  -5.47954e-03     None      -4.9000e-03    0.000e+00   5.795e-04
 L   3    FR  -6.57192e-03     None      -6.4000e-03    0.000e+00   1.719e-04
 L   4    FR  -4.84971e-03     None      -3.7000e-03    0.000e+00   1.150e-03
 L   5    FR  -3.87485e-03     None      -1.2000e-03    0.000e+00   2.675e-03
 L   6    LL  -9.92000e-02  -9.9200e-02   None       1.501e+00   0.000e+00
 L   7    LL  -3.00000e-03  -3.0000e-03   2.0000e-03    1.517e+00   6.939e-18

Exit after 7 iterations.

Optimal LP solution found.

Final LP objective value =   2.3596482e-02
```

7. Further Description

This section gives a detailed description of the algorithm used in nag_opt_lp. This, and possibly the next section, Section 8, may be omitted if the more sophisticated features of the algorithm and software are not currently of interest.

7.1. Overview

nag_opt_lp is based on an inertia-controlling method due to Gill and Murray (1978) and is described in detail by Gill *et al* (1991a). Here the main features of the method are summarized. Where possible, explicit reference is made to the names of variables that are parameters of nag_opt_lp or appear in the printed output. nag_opt_lp has two phases: finding an initial feasible point by minimizing the sum of infeasibilities (the *feasibility phase*), and minimizing the linear objective function within the feasible region (the *optimality phase*). The computations in both phases are performed by the same functions. The two-phase nature of the algorithm is reflected by changing the function being minimized from the sum of infeasibilities to the linear objective function. The feasibility phase does *not* perform the standard simplex method (i.e., it does not necessarily find a vertex), except in the LP case when $m_{lin} \leq n$. Once any iterate is feasible, all subsequent iterates remain feasible.

In general, an iterative process is required to solve a linear program. (For simplicity, we shall always consider a typical iteration and avoid reference to the index of the iteration.) Each new iterate $\bar{x}$ is defined by

$$\bar{x} = x + \alpha p, \tag{1}$$

where the *steplength* α is a non-negative scalar, and p is called the *search direction*.

At each point x, a *working set* of constraints is defined to be a linearly independent subset of the constraints that are satisfied 'exactly' (to within the tolerance defined by the optional parameter **ftol**; see Section 8.2). The working set is the current prediction of the constraints that hold with equality at a solution of an LP problem. The search direction is constructed so that the constraints in the working set remain *unaltered* for any value of the step length. For a bound constraint in the working set, this property is achieved by setting the corresponding component of the search direction to zero. Thus, the associated variable is *fixed* and the specification of the working set induces a partition of x into *fixed* and *free* variables. During a given iteration, the fixed variables are effectively removed from the problem; since the relevant components of the search direction are zero, the columns of A corresponding to fixed variables may be ignored.

Let m_w denote the number of general constraints in the working set and let n_{fx} denote the number of variables fixed at one of their bounds (m_w and n_{fx} are the quantities `Lin` and `Bnd` in the printed output from nag_opt_lp). Similarly, let n_{fr} ($n_{fr} = n - n_{fx}$) denote the number of free variables. At every iteration, *the variables are re-ordered so that the last* n_{fx} *variables are fixed*, with all other relevant vectors and matrices ordered accordingly.

7.2. Definition of the Search Direction

Let A_{fr} denote the m_w by n_{fr} sub-matrix of general constraints in the working set corresponding to the free variables, and let p_{fr} denote the search direction with respect to the free variables only. The general constraints in the working set will be unaltered by any move along p if

$$A_{fr} p_{fr} = 0. \tag{2}$$

In order to compute p_{fr}, the TQ factorization of A_{fr} is used:

$$A_{fr} Q_{fr} = (0 \; T), \tag{3}$$

where T is a non-singular m_w by m_w upper triangular matrix (i.e., $t_{ij} = 0$ if $i > j$), and the non-singular n_{fr} by n_{fr} matrix Q_{fr} is the product of orthogonal transformations (see Gill *et al* (1984)). If the columns of Q_{fr} are partitioned so that

$$Q_{fr} = (Z \; Y),$$

where Y is $n_{fr} \times m_w$, then the n_z ($n_z = n_{fr} - m_w$) columns of Z form a basis for the null space of A_{fr}. Let n_r be an integer such that $0 \le n_r \le n_z$, and let Z_r denote a matrix whose n_r columns are a subset of the columns of Z. (The integer n_r is the quantity `Nrz` in the printed output from nag_opt_lp. In many cases, Z_r will include *all* the columns of Z.) The direction p_{fr} will satisfy (2) if

$$p_{fr} = Z_r p_r, \tag{4}$$

where p_r is any n_r-vector.

7.3. The Main Iteration

Let Q denote the n by n matrix

$$Q = \begin{pmatrix} Q_{fr} & \\ & I_{fx} \end{pmatrix},$$

where I_{fx} is the identity matrix of order n_{fx}. Let g_q denote the transformed gradient

$$g_q = Q^T c$$

and let the vector of first n_r elements of g_q be denoted by g_r. The quantity g_r is known as the *reduced gradient* of $c^T x$. If the reduced gradient is zero, x is a constrained stationary point in the subspace defined by Z. During the feasibility phase, the reduced gradient will usually be zero only at a vertex (although it may be zero at non-vertices in the presence of constraint dependencies). During the optimality phase, a zero reduced gradient implies that x minimizes the linear objective when the constraints in the working set are treated as equalities. At a constrained stationary point, Lagrange multipliers λ_c and λ_b for the general and bound constraints are defined from the equations

$$A_{fr}^T \lambda_c = g_{fr} \quad \text{and} \quad \lambda_b = g_{fx} - A_{fx}^T \lambda_c. \tag{5}$$

Given a positive constant δ of the order of the ***machine precision***, a Lagrange multiplier λ_j corresponding to an inequality constraint in the working set is said to be *optimal* if $\lambda_j \le \delta$ when the associated constraint is at its *upper bound*, or if $\lambda_j \ge -\delta$ when the associated constraint is at its *lower bound*. If a multiplier is non-optimal, the objective function (either the true objective or the sum of infeasibilities) can be reduced by deleting the corresponding constraint (with index `Jdel`; see Section 8.3) from the working set.

If optimal multipliers occur during the feasibility phase and the sum of infeasibilities is non-zero, there is no feasible point, and nag_opt_lp will continue until the minimum value of the sum of infeasibilities has been found. At this point, the Lagrange multiplier λ_j corresponding to an inequality constraint in the working set will be such that $-(1+\delta) \le \lambda_j \le \delta$ when the associated constraint is at its *upper bound*, and $-\delta \le \lambda_j \le (1+\delta)$ when the associated constraint is at its *lower bound*. Lagrange multipliers for equality constraints will satisfy $|\lambda_j| \le 1 + \delta$.

If the reduced gradient is not zero, Lagrange multipliers need not be computed and the non-zero elements of the search direction p are given by $Z_r p_r$. The choice of step length is influenced by the need to maintain feasibility with respect to the satisfied constraints.

Each change in the working set leads to a simple change to A_{fr}: if the status of a general constraint changes, a *row* of A_{fr} is altered; if a bound constraint enters or leaves the working set, a *column* of A_{fr} changes. Explicit representations are recurred of the matrices T and Q_{fr} and of vectors $Q^T g$, and $Q^T c$.

One of the most important features of nag_opt_lp is its control of the conditioning of the working set, whose nearness to linear dependence is estimated by the ratio of the largest to smallest diagonal elements of the TQ factor T (the printed value `Cond T`; see Section 8.3). In constructing the initial working set, constraints are excluded that would result in a large value of `Cond T`.

nag_opt_lp includes a rigorous procedure that prevents the possibility of cycling at a point where the active constraints are nearly linearly dependent (see Gill *et al* (1989)). The main feature of the anti-cycling procedure is that the feasibility tolerance is increased slightly at the start of every iteration. This not only allows a positive step to be taken at every iteration, but also provides, whenever possible, a *choice* of constraints to be added to the working set. Let α_m denote the maximum step at which $x + \alpha_m p$ does not violate any constraint by more than its feasibility tolerance. All constraints at a distance α ($\alpha \le \alpha_m$) along p from the current point are then viewed as acceptable candidates for inclusion in the working set. The constraint whose normal makes the largest angle with the search direction is added to the working set.

7.4. Choosing the Initial Working Set

Let Z be partitioned as $Z = (Z_r \; Z_a)$. A working set for which Z_r defines the null space can be obtained by including *the rows* of Z_a^T as 'artificial constraints'. Minimization of the objective function then proceeds within the subspace defined by Z_r, as described in Section 7.2.

The artificially augmented working set is given by

$$\bar{A}_{fr} = \begin{pmatrix} {Z_a}^T \\ A_{fr} \end{pmatrix}, \tag{6}$$

so that p_{fr} will satisfy $A_{fr}p_{fr} = 0$ *and* ${Z_a}^T p_{fr} = 0$. By definition of the TQ factorization, $\bar{A}_{fr}$ *automatically* satisfies the following:

$$\bar{A}_{fr}Q_{fr} = \begin{pmatrix} {Z_a}^T \\ A_{fr} \end{pmatrix} Q_{fr} = \begin{pmatrix} {Z_a}^T \\ A_{fr} \end{pmatrix} (Z_r \; Z_a \; Y) = (0 \; \bar{T}),$$

where

$$\bar{T} = \begin{pmatrix} I & 0 \\ 0 & T \end{pmatrix},$$

and hence the TQ factorization of (6) is available trivially from T and Q_{fr} without additional expense.

The matrix Z_a is not kept fixed, since its role is purely to define an appropriate null space; the TQ factorization can therefore be updated in the normal fashion as the iterations proceed. No work is required to 'delete' the artificial constraints associated with Z_a when ${Z_r}^T g_{fr} = 0$, since this simply involves repartitioning Q_{fr}. The 'artificial' multiplier vector associated with the rows of ${Z_a}^T$ is equal to ${Z_a}^T g_{fr}$, and the multipliers corresponding to the rows of the 'true' working set are the multipliers that would be obtained if the artificial constraints were not present. If an artificial constraint is 'deleted' from the working set, an `A` appears alongside the entry in the `Jdel` column of the printed output (see Section 8.3).

The number of columns in Z_a and Z_r and the Euclidean norm of ${Z_r}^T g_{fr}$, appear in the printed output as `Nart`, `Nrz` and `Norm Gz` (see Section 8.3).

Under some circumstances, a different type of artificial constraint is used when solving a linear program. Although the algorithm of nag_opt_lp does not usually perform simplex steps (in the traditional sense), there is one exception: a linear program with fewer general constraints than variables (i.e., $m_{lin} \le n$). (Use of the simplex method in this situation leads to savings in storage.) At the starting point, the 'natural' working set (the set of constraints exactly or nearly satisfied at the starting point) is augmented with a suitable number of 'temporary' bounds, each of which has the effect of temporarily fixing a variable at its current value. In subsequent iterations, a temporary bound is treated as a standard constraint until it is deleted from the working set, in which case it is never added again. If a temporary bound is 'deleted' from the working set, an `F` (for 'Fixed') appears alongside the entry in the `Jdel` column of the printed output (see Section 8.3).

8. Optional Parameters

A number of optional input and output parameters to nag_opt_lp are available through the structure argument **options**, type Nag_E04_Opt. A parameter may be selected by assigning an appropriate value to the relevant structure member; those parameters not selected will be assigned default values. If no use is to be made of any of the optional parameters the user should use the NAG defined null pointer, `E04_DEFAULT`, in place of **options** when calling nag_opt_lp; the default settings will then be used for all parameters.

Before assigning values to **options** directly the structure **must** be initialised by a call to the function nag_opt_init (e04xxc). Values may then be assigned to the structure members in the normal C manner.

Option settings may also be read from a file using the function nag_opt_read (e04xyc) in which case initialisation of the **options** structure will be performed automatically if not already done. Any subsequent direct assignment to the **options** structure must **not** be preceded by initialisation.

If assignment of functions and memory to pointers in the **options** structure is required, this must be done directly in the calling program; they cannot be assigned using nag_opt_read (e04xyc).

8.1. Optional Parameter Checklist and Default Values

For easy reference, the following list shows the members of **options** which are valid for nag_opt_lp together with their default values where relevant. The number ϵ is a generic notation for ***machine precision*** (see nag_machine_precision (X02AJC)).

`Nag_ProblemType prob`	**Nag_LP**
`Nag_Start start`	**Nag_Cold**
`Boolean list`	**TRUE**
`Nag_PrintType print_level`	**Nag_Soln_Iter**
`char outfile[80]`	`stdout`
`void (*print_fun)()`	NULL
`Integer max_iter`	max(50, 5(**n**+**nclin**))
`double crash_tol`	0.01
`double ftol`	$\sqrt{\epsilon}$
`double optim_tol`	$\epsilon^{0.8}$
`Integer reset_ftol`	10000
`Integer fcheck`	50
`double inf_bound`	10^{20}
`double inf_step`	max(**inf_bound**,10^{20})
`Integer *state`	size **n**+**nclin**
`double *ax`	size **nclin**
`double *lambda`	size **n**+**nclin**
`Integer iter`	

8.2. Description of Optional Parameters

prob – Nag_ProblemType Default = **Nag_LP**

Input: specifies the problem type. The following are the two possible values of **prob** and the size of the array **cvec** that is required to define the objective function:

Nag_FP **cvec** not accessed;

Nag_LP **cvec**[**n**] required;

Nag_FP denotes a feasible point problem and **Nag_LP** a linear programming problem.
Constraint: **options.prob** = **Nag_FP** or **Nag_LP**.

start – Nag_Start Default = **Nag_Cold**

Input: specifies how the initial working set is chosen. With **options.start** = **Nag_Cold**, nag_opt_lp chooses the initial working set based on the values of the variables and constraints at the initial point. Broadly speaking, the initial working set will include equality constraints and bounds or inequality constraints that violate or 'nearly' satisfy their bounds (to within **crash_tol**; see below).

With **options.start** = **Nag_Warm**, the user must provide a valid definition of every element of the array pointer **options.state** (see below for the definition of this member of **options**). nag_opt_lp will override the users' specification of **state** if necessary, so that a poor choice of the working set will not cause a fatal error. **Nag_Warm** will be advantageous if a good estimate of the initial working set is available – for example, when nag_opt_lp is called repeatedly to solve related problems.
Constraint: **options.start** = **Nag_Cold** or **Nag_Warm**.

list – Boolean Default = **TRUE**

Input: if **options.list** = **TRUE** the parameter settings in the call to nag_opt_lp will be printed.

print_level – **Nag_PrintType** Default = **Nag_Soln_Iter**

Input: the level of results printout produced by nag_opt_lp. The following values are available.

Nag_NoPrint	No output.
Nag_Soln	The final solution.
Nag_Iter	One line of output for each iteration.
Nag_Iter_Long	A longer line of output for each iteration with more information (line exceeds 80 characters).
Nag_Soln_Iter	The final solution and one line of output for each iteration.
Nag_Soln_Iter_Long	The final solution and one long line of output for each iteration (line exceeds 80 characters).
Nag_Soln_Iter_Const	As **Nag_Soln_Iter_Long** with the Lagrange multipliers, the variables x, the constraint values Ax and the constraint status also printed at each iteration.
Nag_Soln_Iter_Full	As **Nag_Soln_Iter_Const** with the diagonal elements of the upper triangular matrix T associated with the TQ factorization (3) of the working set.

Details of each level of results printout are described in Section 8.3.
Constraint: **options.print_level** = **Nag_NoPrint** or **Nag_Soln** or **Nag_Iter** or **Nag_Soln_Iter** or **Nag_Iter_Long** or **Nag_Soln_Iter_Long** or **Nag_Soln_Iter_Const** or **Nag_Soln_Iter_Full**.

outfile – char[80] Default = `stdout`

Input: the name of the file to which results should be printed. If **options.outfile**[0] = '\0' then the `stdout` stream is used.

print_fun – pointer to function Default = NULL

Input: printing function defined by the user; the prototype of **print_fun** is

```
void (*print_fun)(const Nag_Search_State *st, Nag_Comm *comm);
```

See Section 8.3.1.below for further details.

max_iter – Integer Default = max (50,5(**n** + **nclin**))

Input: **max_iter** specifies the maximum number of iterations to be performed by nag_opt_lp.

If the user wishes to check that a call to nag_opt_lp is correct before attempting to solve the problem in full then **max_iter** may be set to 0. No iterations will then be performed but the initialisation stages prior to the first iteration will be processed and a listing of parameter settings output if **options.list** = **TRUE** (the default setting).
Constraint: **options.max_iter** ≥ 0.

crash_tol – double Default = 0.01

Input: **crash_tol** is used in conjunction with the optional parameter **start**. When **options.start** has the default setting, i.e., **options.start** = **Nag_Cold**, nag_opt_lp selects an initial working set. The initial working set will include bounds or general inequality constraints that lie within **crash_tol** of their bounds. In particular, a constraint of the form ${a_j}^T x \geq l$ will be included in the initial working set if $|{a_j}^T x - l| \leq$ **crash_tol** $\times$ $(1 + |l|)$.
Constraint: $0.0 \leq$ **options.crash_tol** ≤ 1.0.

ftol – double Default = $\sqrt{\epsilon}$

Input: **ftol** defines the maximum acceptable *absolute* violation in each constraint at a 'feasible' point. For example, if the variables and the coefficients in the general constraints are of order unity, and the latter are correct to about 6 decimal digits, it would be appropriate to specify **ftol** as 10^{-6}.

nag_opt_lp attempts to find a feasible solution before optimizing the objective function. If the sum of infeasibilities cannot be reduced to zero, nag_opt_lp finds the minimum value of the sum. Let `Sinf` be the corresponding sum of infeasibilities. If `Sinf` is quite small, it may be appropriate to raise **ftol** by a factor of 10 or 100. Otherwise, some error in the data should be suspected.

Note that a 'feasible solution' is a solution that satisfies the current constraints to within the tolerance **ftol**.
Constraint: **options.ftol** > 0.0.

optim_tol – double Default $= \epsilon^{0.8}$

Input: options.optim_tol defines the tolerance used to determine whether the bounds and generated constraints have the correct sign for the solution to be judged optimal.

reset_ftol – Integer Default = 10000

Input: this option is part of an anti-cycling procedure designed to guarantee progress even on highly degenerate problems.

The strategy is to force a positive step at every iteration, at the expense of violating the constraints by a small amount. Suppose that the value of the optional parameter **ftol** is δ. Over a period of **reset_ftol** iterations, the feasibility tolerance actually used by nag_opt_lp increases from 0.5δ to δ (in steps of $0.5\delta/$**reset_ftol**).

At certain stages the following 'resetting procedure' is used to remove constraint infeasibilities. First, all variables whose upper or lower bounds are in the working set are moved exactly onto their bounds. A count is kept of the number of nontrivial adjustments made. If the count is positive, iterative refinement is used to give variables that satisfy the working set to (essentially) ***machine precision***. Finally, the current feasibility tolerance is reinitialized to 0.5δ.

If a problem requires more than **reset_ftol** iterations, the resetting procedure is invoked and a new cycle of **reset_ftol** iterations is started. (The decision to resume the feasibility phase or optimality phase is based on comparing any constraint infeasibilities with δ.)

The resetting procedure is also invoked when nag_opt_lp reaches an apparently optimal, infeasible or unbounded solution, unless this situation has already occurred twice. If any nontrivial adjustments are made, iterations are continued.
Constraint: $0 <$ **options.reset_ftol** < 10000000.

fcheck – Integer Default = 50

Input: every **fcheck** iterations, a numerical test is made to see if the current solution x satisfies the constraints in the working set. If the largest residual of the constraints in the working set is judged to be too large, the current working set is re-factorized and the variables are recomputed to satisfy the constraints more accurately.
Constraint: **options.fcheck** ≥ 1.

inf_bound – double Default $= 10^{20}$

Input: **inf_bound** defines the 'infinite' bound in the definition of the problem constraints. Any upper bound greater than or equal to **inf_bound** will be regarded as plus infinity (and similarly for a lower bound less than or equal to −**inf_bound**).
Constraint: **options.inf_bound** > 0.0.

inf_step – double Default = max (**inf_bound**,10^{20})

Input: **inf_step** specifies the magnitude of the change in variables that will be considered a step to an unbounded solution. (Note that an unbounded solution can occur only when the problem is of type **Nag_LP**). If the change in x during an iteration would exceed the value of **inf_step**, the objective function is considered to be unbounded below in the feasible region.
Constraint: **options.inf_step** > 0.0.

state – Integer * Default memory = **n+nclin**

Input: **state** need not be set if the default option of **options.start** = **Nag_Cold** is used as **n+nclin** values of memory will be automatically allocated by nag_opt_lp.

If the option **start** = **Nag_Warm** has been chosen, **state** must point to a minimum of **n+nclin** elements of memory. This memory will already be available if the **options** structure has been used in a previous call to nag_opt_lp from the calling program, using the same values of **n** and **nclin** and **start** = **Nag_Cold**. If a previous call has not been made sufficient memory must be allocated to **state** by the user.

When a warm start is chosen **state** should specify the desired status of the constraints at the start of the feasibility phase. More precisely, the first n elements of **state** refer to the upper and lower bounds on the variables, and the next m_{lin} elements refer to the general linear constraints (if any). Possible values for **state**[j] are as follows:

state[j]	**Meaning**
0	The corresponding constraint should *not* be in the initial working set.
1	The constraint should be in the initial working set at its lower bound.
2	The constraint should be in the initial working set at its upper bound.
3	The constraint should be in the initial working set as an equality. This value should only be specified if **bl**[j] = **bu**[j]. The values 1,2 or 3 all have the same effect when **bl**[j] = **bu**[j].

The values $-2, -1$ and 4 are also acceptable but will be reset to zero by the function. In particular, if nag_opt_lp has been called previously with the same values of **n** and **nclin**, **state** already contains satisfactory information. (See also the description of the optional parameter **start**). The function also adjusts (if necessary) the values supplied in **x** to be consistent with the values supplied in **state**.

Output: if nag_opt_lp exits with **fail.code** = **NE_NOERROR**, **NW_SOLN_NOT_UNIQUE** or **NW_NOT_FEASIBLE**, the values in **state** indicate the status of the constraints in the working set at the solution. Otherwise, **state** indicates the composition of the working set at the final iterate. The significance of each possible value of **state**[j] is as follows:

state[j]	**Meaning**
-2	The constraint violates its lower bound by more than the feasibility tolerance.
-1	The constraint violates its upper bound by more than the feasibility tolerance.
0	The constraint is satisfied to within the feasibility tolerance, but is not in the working set.
1	This inequality constraint is included in the working set at its lower bound.
2	This inequality constraint is included in the working set at its upper bound.
3	This constraint is included in the working set as an equality. This value of **state** can occur only when **bl**[j] = **bu**[j].
4	This corresponds to optimality being declared with **x**[j] being temporarily fixed at its current value. This value of **state** can only occur when **fail.code** = **NW_SOLN_NOT_UNIQUE**.

ax – double * Default memory = **nclin**

Input: **nclin** values of memory will be automatically allocated by nag_opt_lp and this is the recommended method of use of **options.ax**. However a user may supply memory from the calling program.
Output: If **nclin** > 0, **ax** points to the final values of the linear constraints Ax.

lambda – double * Default memory = **n+nclin**

Input: **n+nclin** values of memory will be automatically allocated by nag_opt_lp and this is the recommended method of use of **options.lambda**. However a user may supply memory from the calling program.

Output: 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_{lin} elements contain the multipliers for the general linear constraints (if any). If **state**[j] = 0 (i.e., constraint j is not in the working set), **lambda**[j] is zero. If x is optimal, **lambda**[j] should be non-negative if **state**[j] = 1, non-positive if **state**[j] = 2 and zero if **state**[j] = 4.

iter – Integer

Output: the total number of iterations performed in the feasibility phase and (if appropriate) the optimality phase.

8.3. Description of Printed Output

The level of printed output can be controlled by the user with the structure members **options.list** and **options.print_level** (see Section 8.2). If **list** = **TRUE** then the parameter values to nag_opt_lp are listed, whereas the printout of results is governed by the value of **print_level**. The default of **print_level** = **Nag_Soln_Iter** provides a single line of output at each iteration and the final result. This section describes all of the possible levels of results printout available from nag_opt_lp.

The convention for numbering the constraints in the iteration results is that indices 1 to n refer to the bounds on the variables, and indices $n+1$ to $n+m_{lin}$ refer to the general constraints. When the status of a constraint changes, the index of the constraint is printed, along with the designation `L` (lower bound), `U` (upper bound), `E` (equality), `F` (temporarily fixed variable) or `A` (artificial constraint).

When **print_level** = **Nag_Iter** or **Nag_Soln_Iter** the following line of output is produced on completion of each iteration.

`Itn` the iteration count.

`Jdel` the index of the constraint deleted from the working set. If `Jdel` is zero, no constraint was deleted.

`Jadd` the index of the constraint added to the working set. If `Jadd` is zero, no constraint was added.

`Step` the step taken along the computed search direction. If a constraint is added during the current iteration (i.e., `Jadd` is positive), `Step` will be the step to the nearest constraint. During the optimality phase, the step can be greater than one only if the reduced Hessian is not positive-definite.

`Ninf` the number of violated constraints (infeasibilities). This will be zero during the optimality phase.

`Sinf/Obj` the value of the current objective function. If x is not feasible, `Sinf` gives a weighted sum of the magnitudes of constraint violations. If x is feasible, `Obj` is the value of the objective function. The output line for the final iteration of the feasibility phase (i.e., the first iteration for which `Ninf` is zero) will give the value of the true objective at the first feasible point.

During the optimality phase, the value of the objective function will be non-increasing. During the feasibility phase, the number of constraint infeasibilities will not increase until either a feasible point is found, or the optimality of the multipliers implies that no feasible point exists. Once optimal multipliers are obtained, the number of infeasibilities can increase, but the sum of infeasibilities will either remain constant or be reduced until the minimum sum of infeasibilities is found.

`Bnd` the number of simple bound constraints in the current working set.

`Lin` the number of general linear constraints in the current working set.

`Nart` the number of artificial constraints in the working set, i.e., the number of columns of Z_a (see Section 7). At the start of the optimality phase, `Nart` provides an estimate of the number of nonpositive eigenvalues in the reduced Hessian.

`Nrz` is the number of columns of Z_r (see Section 7).`Nrz` is the dimension of the subspace in which the objective function is currently being minimized. The value of `Nrz` is the number of variables minus the number of constraints in the working set; i.e., $\texttt{Nrz} = n - (\texttt{Bnd} + \texttt{Lin} + \texttt{Nart})$.

The value of n_z, the number of columns of Z (see Section 7) can be calculated as $n_z = n - (\texttt{Bnd} + \texttt{Lin})$. A zero value of n_z implies that x lies at a vertex of the feasible region.

`Norm Gz` $\|{Z_r}^T g_{fr}\|$, the Euclidean norm of the reduced gradient with respect to Z_r. During the optimality phase, this norm will be approximately zero after a unit step.

If **print_level** = **Nag_Iter_Long**, **Nag_Soln_Iter_Long**, **Nag_Soln_Iter_Const** or **Nag_Soln_Iter_Full** the line of printout is extended to give the following information. (Note this longer line extends over more than 80 characters).

`NOpt` is the number of non-optimal Lagrange multipliers at the current point. `NOpt` is not printed if the current x is infeasible or no multipliers have been calculated. At a minimizer, `NOpt` will be zero.

`Min LM` is the value of the Lagrange multiplier associated with the deleted constraint. If `Min LM` is negative, a lower bound constraint has been deleted; if `Min LM` is positive, an upper bound constraint has been deleted. If no multipliers are calculated during a given iteration, `Min LM` will be zero.

`Cond T` is a lower bound on the condition number of the working set.

When **options.print_level** = **Nag_Soln_Iter_Const** or **Nag_Soln_Iter_Full** more detailed results are given at each iteration. For the setting **Nag_Soln_Iter_Const** additional values output are:

`Value of x` the value of x currently held in **x**.

`State` the current value of **options.state** associated with x.

`Value of Ax` the value of Ax currently held in **options.ax**.

`State` the current value of **options.state** associated with Ax.

Also printed are the Lagrange Multipliers for the bound constraints, linear constraints and artificial constraints.

If **print_level** = **Nag_Soln_Iter_Full** then the diagonal of T and Z_r are also output at each iteration.

When **print_level** = **Nag_Soln**, **Nag_Soln_Iter**, **Nag_Soln_Iter_Const** or **Nag_Soln_Iter_Full** the final printout from nag_opt_lp includes a listing of the status of every variable and constraint. The following describes the printout for each variable.

`Varbl` the name (`V`) and index j, for $j = 1, 2, \ldots, n$ of the variable.

`State` the state of the variable (`FR` if neither bound is in the working set, `EQ` if a fixed variable, `LL` if on its lower bound, `UL` if on its upper bound, `TF` if temporarily fixed at its current value). If `Value` lies outside the upper or lower bounds by more than the feasibility tolerance, `State` will be `++` or `--` respectively.

`Value` the value of the variable at the final iteration.

`Lower bound` the lower bound specified for the variable.
(`None` indicates that $\mathbf{bl}[j-1] \leq$ **-inf_bound**.)

`Upper bound` the upper bound specified for the variable.
(`None` indicates that **bu**$[j-1] \geq$ **inf_bound**.)

`Lagr mult` the value of the Lagrange multiplier for the associated bound constraint. This will be zero if `State` is `FR`. If x is optimal, the multiplier should be non-negative if `State` is `LL`, and non-positive if `State` is `UL`.

`Residual` the difference between the variable `Value` and the nearer of its bounds **bl**$[j-1]$ and **bu**$[j-1]$.

The meaning of the printout for general constraints is the same as that given above for variables, with 'variable' replaced by 'constraint', and with the following change in the heading:

`LCon` the name (`L`) and index j, for $j = 1, 2, \ldots, m_{lin}$ of the constraint.

8.3.1. Output of results via a user defined printing function

Users may also specify their own print function for output of iteration results and the final solution by use of the **options.print_fun** function pointer, which has prototype

```
void (*print_fun)(const Nag_Search_State *st, Nag_Comm *comm);
```

The rest of this section can be skipped by a user who only wishes to use the default printing facilities.

When a user defined function is assigned to **options.print_fun** this will be called in preference to the internal print function of nag_opt_lp. Calls to the user defined function are again controlled by means of the **options.print_level** member. Information is provided through **st** and **comm**, the two structure arguments to **print_fun**.

If **comm->it_prt** = **TRUE** then the results from the last iteration of nag_opt_lp are set in the following members of **st**:

first – Boolean
 TRUE on the first call to **print_fun**.

iter – Integer
 the number of iterations performed.

n – Integer
 the number of variables.

nclin – Integer
 the number of linear constraints.

jdel – Integer
 index of constraint deleted.

jadd – Integer
 index of constraint added.

step – double
 the step taken along the current search direction.

ninf – Integer
 the number of infeasibilities.

f – double
 the value of the current objective function.

bnd – Integer
 number of bound constraints in the working set.

lin – Integer
 number of general linear constraints in the working set.

nart – Integer
 number of artificial constraints in the working set.

nrz – Integer
number of columns of Z_r.

norm_gz – double
Euclidean norm of the reduced gradient, $\|{Z_r}^T g_{fr}\|$.

nopt – Integer
number of non-optimal Lagrange multipliers.

min_lm – double
value of the Lagrange multiplier associated with the deleted constraint.

condt – double
a lower bound on the condition number of the working set.

x – double *
x points to the **n** memory locations holding the current point x.

ax – double *
ax points to the **nclin** memory locations holding the current values Ax.

state – Integer *
state points to the **n+nclin** memory locations holding the status of the variables and general linear constraints. See Section 8.2 for a description of the possible status values.

t – double *
the upper triangular matrix T with **st->lin** columns. Matrix element i, j is held in **st->t**[$(i-1)$***st->tdt** $+ j - 1$].

tdt – Integer
the trailing dimension for **st->t**.

If **comm->new_lm = TRUE** then the Lagrange multipliers have been updated and the following members, **kx**, **kactive**, **lambda** and **gq**, are set:

kx – Integer *
Indices of the bound constraints with associated multipliers.
Value of **st->kx**[i] is the index of the constraint with multiplier **st->lambda**[i] for $i = 0, 1, \ldots,$**st->bnd**-1.

kactive – Integer *
Indices of the linear constraints with associated multipliers.
Value of **st->kactive**[i] is the index of the constraint with multiplier **st->lambda**[**st->bnd** $+ i$] for $i = 0, 1, \ldots,$**st->lin**-1.

lambda – double *
the multipliers for the constraints in the working set. **lambda**[i] for $i = 0, 1, \ldots,$**st->bnd**-1 hold the multipliers for the bound constraints while the multipliers for the linear constraints are held at indices $i =$ **st->bnd**,...,**st->bnd** + **st->lin**-1.

gq – double *
st->gq[i] for $i = 0, 1, \ldots,$**st->nart** $- 1$ hold the multipliers for the artificial constraints.

The following members of **st** are also relevant and apply when **comm->it_prt** or **comm->new_lm** is **TRUE**.

refactor – Boolean
TRUE if iterative refinement performed. See Section 7.3 and optional parameter **reset_ftol**.

jmax – Integer
if **st->refactor = TRUE** then **st->jmax** holds the index of the constraint with the maximum violation.

errmax – double
if **st->refactor = TRUE** then **st->errmax** holds the value of the maximum violation.

moved – Boolean
TRUE if some variables moved to their bounds. See the optional parameter **reset_ftol**.

nmoved – Integer
if **st->moved** = **TRUE** then **st->nmoved** holds the number of variables which were moved to their bounds.

rowerr – Boolean
TRUE if some constraints are not satisfied to within **options.ftol**.

feasible – Boolean
TRUE when a feasible point has been found.

If **comm->sol_prt** = **TRUE** then the final result from nag_opt_lp is available and the following members of **st** are set:

iter – Integer
the number of iterations performed.

n – Integer
the number of variables.

nclin – Integer
the number of linear constraints.

x – double *
x points to the **n** memory locations holding the final point x.

f – double
the final objective function value or, if x is not feasible, the sum of infeasibilities. If the problem is of type **Nag_FP** and x is feasible then **f** is set to zero.

ax – double *
ax points to the **nclin** memory locations holding the final values Ax.

state – Integer *
state points to the **n+nclin** memory locations holding the final status of the variables and general linear constraints. See Section 8.2for a description of the possible status values.

lambda – double *
lambda points to the **n** + **nclin** final values of the Lagrange multipliers.

bl – double *
bl points to the **n** + **nclin** lower bound values.

bu – double *
bu points to the **n** + **nclin** upper bound values.

endstate – Nag_EndState
the state of termination of nag_opt_lp. Possible values of **endstate** and their correspondence to the exit value of **fail.code** are:

Value of **endstate**	Value of **fail.code**
Nag_Feasible and **Nag_Optimal**	**NE_NOERROR**
Nag_Weakmin	**NW_SOLN_NOT_UNIQUE**
Nag_Unbounded	**NE_UNBOUNDED**
Nag_Infeasible	**NW_NOT_FEASIBLE**
Nag_Too_Many_Iter	**NW_TOO_MANY_ITER**

The relevant members of the structure **comm** are:

it_prt – Boolean
will be **TRUE** when the print function is called with the result of the current iteration.

sol_prt – Boolean
will be **TRUE** when the print function is called with the final result.

new_lm – Boolean
: will be **TRUE** when the Lagrange multipliers have been updated.

user – double *
iuser – Integer *
p – Pointer
: Pointers for communication of user information. If used they must be allocated memory by the user either before entry to nag_opt_lp or during a call to **print_fun**. The type Pointer is `void *`.

9. Error Indications and Warnings

NE_INT_ARG_LT
On entry, **n** must not be less than 1: **n** = ⟨*value*⟩.
On entry, **nclin** must not be less than 0: **nclin** = ⟨*value*⟩.

NE_2_INT_ARG_LT
On entry, **tda** = ⟨*value*⟩ while **n** = ⟨*value*⟩. These parameters must satisfy **tda** ≥ **n**.

NE_OPT_NOT_INIT
options structure not initialised.

NE_BAD_PARAM
On entry parameter **options.print_level** had an illegal value.
On entry parameter **options.prob** had an illegal value.
On entry parameter **options.start** had an illegal value.

NE_INVALID_INT_RANGE_1
Value ⟨*value*⟩ given to **options.max_iter** not valid. Correct range is **max_iter** ≥ 0.
Value ⟨*value*⟩ given to **options.fcheck** not valid. Correct range is **fcheck** ≥ 1.

NE_INVALID_INT_RANGE_2
Value ⟨*value*⟩given to **options.reset_ftol** not valid. Correct range is 0 < **reset_ftol** < 10000000.

NE_INVALID_REAL_RANGE_FF
Value ⟨*value*⟩ given to **options.crash_tol** not valid. Correct range is 0.0 ≤ **crash_tol** ≤ 1.0.

NE_INVALID_REAL_RANGE_F
Value ⟨*value*⟩ given to **options.ftol** not valid. Correct range is **ftol** > 0.0.
Value ⟨*value*⟩ given to **options.inf_bound** not valid. Correct range is **inf_bound** > 0.0.
Value ⟨*value*⟩ given to **options.inf_step** not valid. Correct range is **inf_step** > 0.0.

NE_CVEC_NULL
options.prob = ⟨*value*⟩ but argument **cvec** = NULL.

NE_WARM_START
options.start = **Nag_Warm** but pointer **options.state** = NULL.

NE_BOUND
The lower bound for variable ⟨*value*⟩ (array element **bl**[⟨*value*⟩]) is greater than the upper bound.

NE_BOUND_LCON
The lower bound for linear constraint ⟨*value*⟩ (array element **bl**[⟨*value*⟩]) is greater than the upper bound.

NE_STATE_VAL
options.state[⟨*value*⟩] is out of range. **state**[⟨*value*⟩] = ⟨*value*⟩.

NE_ALLOC_FAIL
Memory allocation failed.

If one of the above exits occurs, no values will have been assigned to **objf**, or to **options.ax** and **options.lambda**. **x** and **options.state** will be unchanged.

NW_SOLN_NOT_UNIQUE

Optimal solution is not unique.

x is a weak local minimum (the projected gradient is negligible, the Lagrange multipliers are optimal but there is a small multiplier). This means that the solution x is not unique.

NE_UNBOUNDED

Solution appears to be unbounded.

This value of **fail.code** implies that a step as large as **options.inf_step** would have to be taken in order to continue the algorithm. This situation can occur only when the problem is of type **Nag_LP** and at least one variable has no upper or lower bound.

NW_NOT_FEASIBLE

No feasible point was found for the linear constraints.

It was not possible to satisfy all the constraints to within the feasibility tolerance. In this case, the constraint violations at the final x will reveal a value of the tolerance for which a feasible point will exist – for example, if the feasibility tolerance for each violated constraint exceeds its `Residual` at the final point. The user should check that there are no constraint redundancies. If the data for the constraints are accurate only to the absolute precision σ, the user should ensure that the value of the optional parameter **ftol** is *greater* than σ. For example, if all elements of A are of order unity and are accurate only to three decimal places, the optional parameter **ftol** should be at least 10^{-3}.

NW_TOO_MANY_ITER

The maximum number of iterations, ⟨*value*⟩, have been performed.

The value of the optional parameter **max_iter** may be too small. If the method appears to be making progress (e.g. the objective function is being satisfactorily reduced), increase the value of **options.max_iter** and rerun nag_opt_lp (possibly using the **options.start** = **Nag_Warm** facility to specify the initial working set).

NW_OVERFLOW_WARN

Serious ill-conditioning in the working set after adding constraint ⟨*value*⟩. Overflow may occur in subsequent iterations.

If overflow occurs preceded by this warning then serious ill-conditioning has probably occurred in the working set when adding a constraint. It may be possible to avoid the difficulty by increasing the magnitude of the optional parameter **ftol** and re-running the program. If the message recurs even after this change, the offending linearly dependent constraint j must be removed from the problem.

NE_NOT_APPEND_FILE

Cannot open file ⟨*string*⟩ for appending.

NE_WRITE_ERROR

Error occurred when writing to file ⟨*string*⟩.

NE_NOT_CLOSE_FILE

Cannot close file ⟨*string*⟩.

10. Further Comments

Sensible scaling of the problem is likely to reduce the number of iterations required and make the problem less sensitive to perturbations in the data, thus improving the condition of the problem. In the absence of better information it is usually sensible to make the Euclidean lengths of each constraint of comparable magnitude. See the Chapter Introduction and Gill *et al*(1986) for further information and advice.

10.1. Accuracy

nag_opt_lp implements a numerically stable active set strategy and returns solutions that are as accurate as the condition of the problem warrants on the machine.

11. References

Gill P E, Hammarling S J, Murray W, Saunders M A and Wright M H (1986) *User's Guide for LSSOL (Version 1.0): A Fortran Package for Constrained Least-squares and Convex Quadratic Programming* Report SOL 86-1, Department of Operations Research, Stanford University.

Gill P E and Murray W (1978) Numerically Stable Methods for Quadratic Programming *Mathematical Programming* **14** 349–372.

Gill P E, Murray W, Saunders M A and Wright M H (1984) Procedures for Optimization Problems with a Mixture of Bounds and General Linear Constraints *ACM Trans. Math. Softw.* **10** 282–298.

Gill P E, Murray W, Saunders M A and Wright M H (1989) A Practical Anti-cycling Procedure for Linearly Constrained Optimization *Mathematical Programming* **45** 437–474.

Gill P E, Murray W, Saunders M A and Wright M H (1991a) Inertia-controlling Methods for General Quadratic Programming *SIAM Review* **33** 1–36.

Gill P E, Murray W and Wright M H (1991b) *Numerical Linear Algebra and Optimization (Volume 1)* Addison Wesley, Redwood City, California.

12. See Also

nag_opt_qp (e04nfc)
nag_opt_init (e04xxc)
nag_opt_read (e04xyc)
nag_opt_free (e04xzc)

13. Example 2

This example is a portfolio investment problem taken from Gill *et al* (1991b). The objective function to be minimized is

$$-5x_1 - 2x_3$$

subject to the bounds

$$x_1 \geq -75$$
$$x_2 \geq -1000$$
$$x_3 \geq -25$$

and the general constraints

$$\begin{aligned} 20x_1 + 2x_2 + 100x_3 &= 0 \\ 18x_1 + 3x_2 + 102x_3 &\geq -600 \\ 15x_1 - \frac{1}{2}x_2 - 25x_3 &\geq 0 \\ -5x_1 + \frac{3}{2}x_2 - 25x_3 &\geq -500 \\ -5x_1 - \frac{1}{2}x_2 + 75x_3 &\geq -1000 \end{aligned}$$

The initial point, which is feasible, is

$$x_0 = (10.0,\ 20.0,\ 100.0)^T.$$

The solution is

$$x^* = (75.0,\ -250.0,\ -10.0)^T.$$

Three general constraints are active at the solution, the bound constraints are all inactive.

This example shows an optional parameter value assigned directly within the program text while others are read from a data file. The **options** structure is declared and initialised by nag_opt_init (e04xxc), a value is assigned directly to option **inf_bound** and nag_opt_lp is then called. On successful return two further options are read from a data file by use of nag_opt_read (e04xyc) and the problem is re-run. The memory freeing function nag_opt_free (e04xzc) is used to free the memory assigned to the pointers in the options structure. Users should **not** use the standard C function `free()` for this purpose.

13.1. Program Text

```
static void ex2()
{
  /* This sample linear program (LP) is a portfolio investment problem
   * (see Chapter 7, pp 258--262 of ``Numerical Linear Algebra and
   * Optimization'', by Gill, Murray and Wright, Addison Wesley, 1991).
   * The problem involves the rearrangement of a portfolio of three
   * stocks, Glitter, Risky and Trusty, so that the net worth of the
   * investor is maximized.
   * The problem is characterized by the following data:
   *                           Glitter     Risky        Trusty
   * 1990 Holdings               75        1000           25
   * 1990 Priceshare($)          20           2          100
   * 2099 Priceshare($)          18           3          102
   * 2099 Dividend                5           0            2
   *
   * The variables x[0], x[1] and x[2] represent the change in each of
   * the three stocks.
   */

  double x[MAXN], cvec[MAXN];
  double a[MAXLIN][MAXN];
  double bl[MAXBND], bu[MAXBND];
  double bigbnd, objf;
  Integer n, tda, nclin;
  Boolean print;
  Nag_E04_Opt options;
  static NagError fail, fail2;

  Vprintf("\nExample 2: some optional parameters are set.\n");

  fail.print = TRUE;
  fail2.print = TRUE;

  /* Set the actual problem dimensions.
   * n      = the number of variables.
   * nclin  = the number of general linear constraints (may be 0).
   */
  tda = MAXN;
  n = 3;
  nclin = 5;

  /* Define the value used to denote ``infinite'' bounds. */
  bigbnd = 1e+25;

  /* Objective function:  maximize  5*X[0] + 2*X[2], or equivalently,
   * minimize -5*X[0] - 2*X[2].
   */
  cvec[0] = -5.0;
  cvec[1] = 0.0;
  cvec[2] = -2.0;

  /* a  = the general constraint matrix.
   * bl = the lower bounds on  x  and  A*x.
   * bu = the upper bounds on  x  and  A*x.
   * x  = the initial estimate of the solution.
   *
   * A nonnegative amount of stock must be present after rearrangement.
   * For Glitter:  x[0] + 75 >= 0.
   */
  bl[0] = -75.0;
  bu[0] = bigbnd;

  /* For Risky:    x[1] + 1000 >= 0. */
  bl[1] = -1000.0;
  bu[1] = bigbnd;
```

```
/* For Trusty:   x[2] + 25 >= 0. */
bl[2] = -25.0;
bu[2] = bigbnd;

/* The current value of the portfolio must be the same after
 * rearrangement, i.e.,
 *  20*(75+x[0]) + 2*(1000+x[1]) + 100*(25+x[2]) = 6000, or
 *  20*x[0] + 2*x[1] + 100*x[2] = 0.
 */
a[0][0] = 20.0;
a[0][1] = 2.0;
a[0][2] = 100.0;
bl[n] = 0.0;
bu[n] = 0.0;

/* The value of the portfolio must increase by at least 5 per cent
 * at the end of the year, i.e.,
 * 18*(75+x[0]) + 3*(1000+x[1]) + 102*(25+x[2]) >= 6300, or
 * 18*x[0] + 3*x[1] + 102*x[2] >= -600.
 */
a[1][0] = 18.0;
a[1][1] = 3.0;
a[1][2] = 102.0;
bl[n + 1] = -600.0;
bu[n + 1] = bigbnd;

/* There are three ``balanced portfolio'' constraints.  The value of
 * a stock must constitute at least a quarter of the total final
 * value of the portfolio.  After rearrangement, the value of the
 * portfolio after is  20*(75+x[0]) + 2*(1000+x[1]) + 100*(25+x[2]).
 *
 * If Glitter is to constitute at least a quarter of the final
 * portfolio, then  15*x[0] - 0.5*x[1] - 25*x[2] >= 0.
 */
a[2][0] = 15.0;
a[2][1] = -0.5;
a[2][2] = -25.0;
bl[n + 2] = 0.0;
bu[n + 2] = bigbnd;

/* If Risky is to constitute at least a quarter of the final
 * portfolio, then  -5*x[0] + 1.5*x[1] - 25*x[2] >= -500.
 */
a[3][0] = -5.0;
a[3][1] = 1.5;
a[3][2] = -25.0;
bl[n + 3] = -500.0;
bu[n + 3] = bigbnd;

/* If Trusty is to constitute at least a quarter of the final
 * portfolio, then  -5*x[0] - 0.5*x[1] + 75*x[2] >= -1000.
 */
a[4][0] = -5.0;
a[4][1] = -0.5;
a[4][2] = 75.0;
bl[n + 4] = -1000.0;
bu[n + 4] = bigbnd;

/* Set the initial estimate of the solution.
 * This portfolio is infeasible.
 */
x[0] = 10.0;
x[1] = 20.0;
x[2] = 100.0;

/* Initialise options structure to null values. */
e04xxc(&options);

/* Set one option */
options.inf_bound = bigbnd;
```

```
  /* Solve the problem. */
  e04mfc(n, nclin, (double *)a, tda, bl, bu, cvec,
         x, &objf, &options, NAGCOMM_NULL, &fail);

  if (fail.code == NE_NOERROR)
    {
      /* Re-solve the problem with some additonal options. */

      Vprintf("Re-solve problem with output of iteration results");
      Vprintf(" suppressed and ftol = 1.0e-10.\n");

      /* Read additional options from a file. */
      fail.print = TRUE;
      print = TRUE;
      e04xyc("e04mfc", "stdin", &options, print, "stdout", &fail);

      /* Reset starting point */
      x[0] = 0.0;
      x[1] = 0.0;
      x[2] = 0.0;

      /* Solve the problem again. */
      e04mfc(n, nclin, (double *)a, tda, bl, bu, cvec,
             x, &objf, &options, NAGCOMM_NULL, &fail);
    }
  /* Free memory allocated by e04mfc to pointers in options. */
  e04xzc(&options, "all", &fail2);
  if (fail.code != NE_NOERROR || fail2.code != NE_NOERROR) exit(EXIT_FAILURE);
} /* ex2 */
```

13.2. Program Data

```
Following options for e04mfc are read by e04xyc in example 2.

begin e04mfc

 print_level = Nag_Soln /* Print solution only */
 ftol = 1e-10           /* Set feasiblity tolerance */

end
```

13.3. Program Results

```
Example 2: some optional parameters are set.

Parameters to e04mfc
--------------------

Linear constraints...........    5    Number of variables..........    3

prob...................     Nag_LP    start..................  Nag_Cold
ftol...................   1.05e-08    reset_ftol.............         5
fcheck.................         50    crash_tol..............  1.00e-02
inf_bound..............   1.00e+25    inf_step...............  1.00e+25
max_iter...............         50    machine precision......  1.11e-16
optim_tol..............   1.72e-13    min_infeas.............     FALSE
print_level......... Nag_Soln_Iter
outfile................     stdout

Memory allocation:
state..................        Nag
ax.....................        Nag    lambda.................       Nag

Results from e04mfc:
-------------------

  Itn Jdel  Jadd   Step    Ninf  Sinf/Obj    Bnd  Lin  Nart  Nrz  Norm Gz

   0   0     0    0.0e+00   1   1.9369e+02    0    1    2     0   1.96e+01
   1   2 A   6 L  5.0e-01   0   7.2049e-01    0    2    1     0   4.00e-02
```

```
   2   6 L   8 L  1.1e+01   0  -2.2109e+02    0    2    1     0   4.98e-01
   3   1 A   7 L  5.4e+02   0  -3.5500e+02    0    3    0     0   0.00e+00

Final solution:

  Varbl State    Value      Lower Bound  Upper Bound    Lagr Mult    Residual

 V   1    FR   7.50000e+01  -7.5000e+01   None       0.000e+00   1.500e+02
 V   2    FR  -2.50000e+02  -1.0000e+03   None       0.000e+00   7.500e+02
 V   3    FR  -1.00000e+01  -2.5000e+01   None       0.000e+00   1.500e+01

   LCon  State    Value      Lower Bound  Upper Bound    Lagr Mult    Residual

 L   1    EQ  -3.01303e-13   0.0000e+00   0.0000e+00   -1.300e-01  -3.013e-13
 L   2    FR  -4.20000e+02  -6.0000e+02   None       0.000e+00   1.800e+02
 L   3    FR   1.50000e+03   0.0000e+00   None       0.000e+00   1.500e+03
 L   4    LL  -5.00000e+02  -5.0000e+02   None       2.500e-01   5.684e-14
 L   5    LL  -1.00000e+03  -1.0000e+03   None       2.300e-01   0.000e+00

Exit after 3 iterations.

Optimal LP solution found.

Final LP objective value =  -3.5500000e+02

Re-solve problem with output of iteration results suppressed and ftol = 1.0e-10.

Optional parameter setting for e04mfc.
--------------------------------------

Option file: stdin

print_level set to Nag_Soln
ftol set to 1.00e-10

Parameters to e04mfc
--------------------

Linear constraints............   5    Number of variables...........   3

prob....................    Nag_LP    start...................  Nag_Cold
ftol....................  1.00e-10    reset_ftol..............         5
fcheck..................        50    crash_tol...............  1.00e-02
inf_bound...............  1.00e+25    inf_step................  1.00e+25
max_iter................        50    machine precision.......  1.11e-16
optim_tol...............  1.72e-13    min_infeas..............     FALSE
print_level.........      Nag_Soln
outfile.................    stdout

Memory allocation:
state...................       Nag
ax......................       Nag    lambda..................       Nag

Final solution:

  Varbl State    Value      Lower Bound  Upper Bound    Lagr Mult    Residual

 V   1    FR   7.50000e+01  -7.5000e+01   None       0.000e+00   1.500e+02
 V   2    FR  -2.50000e+02  -1.0000e+03   None       0.000e+00   7.500e+02
 V   3    FR  -1.00000e+01  -2.5000e+01   None       0.000e+00   1.500e+01

   LCon  State    Value      Lower Bound  Upper Bound    Lagr Mult    Residual

 L   1    EQ   4.78019e-13   0.0000e+00   0.0000e+00   -1.300e-01   4.780e-13
 L   2    FR  -4.20000e+02  -6.0000e+02   None       0.000e+00   1.800e+02
 L   3    FR   1.50000e+03   0.0000e+00   None       0.000e+00   1.500e+03
 L   4    LL  -5.00000e+02  -5.0000e+02   None       2.500e-01   0.000e+00
 L   5    LL  -1.00000e+03  -1.0000e+03   None       2.300e-01   3.411e-13

Exit after 2 iterations.
```

```
Optimal LP solution found.

Final LP objective value =  -3.5500000e+02
```