

# NAG Library Function Document

## nag\_opt\_lp (e04mfc)

### 1 Purpose

nag\_opt\_lp (e04mfc) 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, const double a[], Integer tda,
                 const double bl[], const double bu[], const double cvec[], double x[],
                 double *objf, Nag_E04_Opt *options, Nag_Comm *comm, NagError *fail)
```

### 3 Description

nag\_opt\_lp (e04mfc) is designed to solve linear programming (LP) problems of the form

$$\underset{x \in R^n}{\text{minimize}} \quad c^T x \quad \text{subject to} \quad l \leq \begin{Bmatrix} x \\ Ax \end{Bmatrix} \leq u,$$

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

The function 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 **options.inf\_bound** in Section 12.2).

You must supply an initial estimate of the solution.

Details about the algorithm are described in Section 11, but it is not necessary to read this more advanced section before using nag\_opt\_lp (e04mfc).

### 4 References

Gill P E, Hammarling S, Murray W, Saunders M A and Wright M H (1986) Users' guide for LSSOL (Version 1.0) *Report SOL 86-1* Department of Operations Research, Stanford University

Gill P E and Murray W (1978) Numerically stable methods for quadratic programming *Math. 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. Software* **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 *Math. Programming* **45** 437–474

Gill P E, Murray W, Saunders M A and Wright M H (1991) Inertia-controlling methods for general quadratic programming *SIAM Rev.* **33** 1–36

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

## 5 Arguments

- 1: **n** – Integer *Input*  
*On entry:*  $n$ , the number of variables.  
*Constraint:*  $n > 0$ .
  
- 2: **nclin** – Integer *Input*  
*On entry:*  $m_{lin}$ , the number of general linear constraints.  
*Constraint:*  $nclin \geq 0$ .
  
- 3: **a[nclin × tda]** – const double *Input*  
**Note:** the  $(i, j)$ th element of the matrix  $A$  is stored in  $a[(i - 1) \times tda + j - 1]$ .  
*On entry:* the  $i$ th row of **a** must contain the coefficients of the  $i$ th general linear constraint (the  $i$ th row of  $A$ ), for  $i = 1, 2, \dots, m_{lin}$ .  
 If **nclin** = 0 then the array **a** is not referenced.
  
- 4: **tda** – Integer *Input*  
*On entry:* the stride separating matrix column elements in the array **a**.  
*Constraint:* if **nclin** > 0,  $tda \geq n$
  
- 5: **bl[n + nclin]** – const double *Input*
- 6: **bu[n + nclin]** – const double *Input*  
*On entry:* **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 nonexistent lower bound (i.e.,  $l_j = -\infty$ ), set  $bl[j - 1] \leq -options.inf\_bound$ , and to specify a nonexistent upper bound (i.e.,  $u_j = +\infty$ ), set  $bu[j - 1] \geq options.inf\_bound$ ; here **options.inf\_bound** is the value of the optional parameter **options.inf\_bound**, whose default value is  $10^{20}$  (see Section 12.2). To specify the  $j$ th constraint as an *equality*, set  $bl[j - 1] = bu[j - 1] = \beta$ , say, where  $|\beta| < options.inf\_bound$ .  
*Constraint:*  $bl[j - 1] \leq bu[j - 1]$ , for  $j = 1, 2, \dots, n + nclin - 1$ .
  
- 7: **cvec[n]** – const double *Input*  
*On entry:* the coefficients of the objective function when the problem is of type **options.prob** = Nag\_LP. The problem type is specified by the optional parameter **options.prob** (see Section 12.2) and the values **options.prob** = Nag\_LP or Nag\_FP represent linear programming problem and feasible point problem respectively. **options.prob** = Nag\_LP is the default problem type for nag\_opt\_lp (e04mfc).  
 If the problem type **options.prob** = Nag\_FP is specified then **cvec** is not referenced and a NULL pointer may be given.
  
- 8: **x[n]** – double *Input/Output*  
*On entry:* an initial estimate of the solution.  
*On exit:* the point at which nag\_opt\_lp (e04mfc) terminated. If **fail.code** = NE\_NOERROR, NW\_SOLN\_NOT\_UNIQUE or NW\_NOT\_FEASIBLE, **x** contains an estimate of the solution.
  
- 9: **objf** – double \* *Output*  
*On exit:* 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 **options.prob** = Nag\_FP and  $x$  is feasible, **objf** is set to zero.

10: **options** – Nag\_E04\_Opt \*

Input/Output

*On entry/exit:* a pointer to a structure of type Nag\_E04\_Opt whose members are optional parameters for nag\_opt\_lp (e04mfc). These structure members offer the means of adjusting some of the argument 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 12.2. Some of the results returned in **options** can be used by nag\_opt\_lp (e04mfc) to perform a ‘warm start’ if it is re-entered (see the member **options.start** in Section 12.2).

If any of these optional parameters are required, then the structure **options** should be declared and initialized by a call to nag\_opt\_init (e04xxc) immediately before being supplied as an argument to nag\_opt\_lp (e04mfc).

11: **comm** – Nag\_Comm \*

Input/Output

**Note:** **comm** is a NAG defined type (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation).

*On entry/exit:* structure containing pointers for user communication with an optional user-defined printing function. See Section 12.3.1 for details. If you do 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 (e04mfc).

12: **fail** – NagError \*

Input/Output

The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

## 5.1 Description of Printed Output

Intermediate and final results are printed out by default. The level of printed output can be controlled with the structure member **options.print\_level** (see Section 12.2). The default, **options.print\_level** = 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 (e04mfc).

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 <b>options.prob</b> = 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., $\text{Nrz} = n - (\text{Bnd} + \text{Lin} + \text{Nart})$ .
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 $j = 1, 2, \dots, n$ 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 $\mathbf{bl}[j-1] \leq -\text{options.inf\_bound.}$ )
Upper bound	the upper bound specified for the variable. (None indicates that $\mathbf{bu}[j-1] \geq \text{options.inf\_bound.}$ )
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 $j = 1, 2, \dots, m_{lin}$ of the constraint.
------	--

## 6 Error Indicators and Warnings

If one of NE\_INT\_ARG\_LT, NE\_2\_INT\_ARG\_LT, NE\_OPT\_NOT\_INIT, NE\_BAD\_PARAM, NE\_INVALID\_INT\_RANGE\_1, NE\_INVALID\_INT\_RANGE\_2, NE\_INVALID\_REAL\_RANGE\_FF, NE\_INVALID\_REAL\_RANGE\_F, NE\_CVEC\_NULL, NE\_WARM\_START, NE\_BOUND, NE\_-BOUND\_LCON, NE\_STATE\_VAL and NE\_ALLOC\_FAIL occurs, no values will have been assigned to **objf**, or to **options.ax** and **options.lambda**. **x** and **options.state** will be unchanged.

### NE\_2\_INT\_ARG\_LT

On entry, **tda** =  $\langle \text{value} \rangle$  while **n** =  $\langle \text{value} \rangle$ . These arguments must satisfy  $\mathbf{tda} \geq \mathbf{n}$ .

### NE\_ALLOC\_FAIL

Dynamic memory allocation failed.

### NE\_BAD\_PARAM

On entry, argument **options.print\_level** had an illegal value.

On entry, argument **options.prob** had an illegal value.

On entry, argument **options.start** had an illegal value.

#### NE\_BOUND

The lower bound for variable  $\langle value \rangle$  (array element **bl**[ $\langle value \rangle$ ]) is greater than the upper bound.

#### NE\_BOUND\_LCON

The lower bound for linear constraint  $\langle value \rangle$  (array element **bl**[ $\langle value \rangle$ ]) is greater than the upper bound.

#### NE\_CVEC\_NULL

**options.prob** =  $\langle value \rangle$  but argument **cvec** = **NULL**.

#### NE\_INT\_ARG\_LT

On entry, **n** =  $\langle value \rangle$ .

Constraint: **n**  $\geq$  1.

On entry, **nclin** =  $\langle value \rangle$ .

Constraint: **nclin**  $\geq$  0.

#### NE\_INVALID\_INT\_RANGE\_1

Value  $\langle value \rangle$  given to **options.fcheck** not valid. Correct range is **options.fcheck**  $\geq$  1.

Value  $\langle value \rangle$  given to **options.max\_iter** not valid. Correct range is **options.max\_iter**  $\geq$  0.

#### NE\_INVALID\_INT\_RANGE\_2

Value  $\langle value \rangle$  given to **options.reset\_ftol** not valid. Correct range is  $0 < \text{options.reset\_ftol} < 10000000$ .

#### NE\_INVALID\_REAL\_RANGE\_F

Value  $\langle value \rangle$  given to **options.ftol** not valid. Correct range is **options.ftol**  $>$  0.0.

Value  $\langle value \rangle$  given to **options.inf\_bound** not valid. Correct range is **options.inf\_bound**  $>$  0.0.

Value  $\langle value \rangle$  given to **options.inf\_step** not valid. Correct range is **options.inf\_step**  $>$  0.0.

#### NE\_INVALID\_REAL\_RANGE\_FF

Value  $\langle value \rangle$  given to **options.crash\_tol** not valid. Correct range is  $0.0 \leq \text{options.crash\_tol} \leq 1.0$ .

#### NE\_NOT\_APPEND\_FILE

Cannot open file  $\langle string \rangle$  for appending.

#### NE\_NOT\_CLOSE\_FILE

Cannot close file  $\langle string \rangle$ .

#### NE\_OPT\_NOT\_INIT

**options** structure not initialized.

#### NE\_STATE\_VAL

**options.state**[ $\langle value \rangle$ ] is out of range. **options.state**[ $\langle value \rangle$ ] =  $\langle value \rangle$ .

**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 **options.prob** = Nag\_LP and at least one variable has no upper or lower bound.

**NE\_WARM\_START**

**options.start** = Nag\_Warm but pointer **options.state** = NULL.

**NE\_WRITE\_ERROR**

Error occurred when writing to file  $\langle string \rangle$ .

**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. You should check that there are no constraint redundancies. If the data for the constraints are accurate only to the absolute precision  $\sigma$ , you should ensure that the value of the optional parameter **options.ftol** is greater than  $\sigma$ . For example, if all elements of  $A$  are of order unity and are accurate only to three decimal places, the optional parameter **options.ftol** should be at least  $10^{-3}$ .

**NW\_OVERFLOW\_WARN**

Serious ill-conditioning in the working set after adding constraint  $\langle value \rangle$ . 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 **options.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.

**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.

**NW\_TOO\_MANY\_ITER**

The maximum number of iterations,  $\langle value \rangle$ , have been performed.

The value of the optional parameter **options.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 (e04mfc) (possibly using the **options.start** = Nag\_Warm facility to specify the initial working set).

## 7 Accuracy

nag\_opt\_lp (e04mfc) implements a numerically stable active set strategy and returns solutions that are as accurate as the condition of the problem warrants on the machine.

## 8 Parallelism and Performance

nag\_opt\_lp (e04mfc) is not threaded in any implementation.

## 9 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 e04 Chapter Introduction and Gill *et al.* (1986) for further information and advice.

## 10 Example

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

$$-5x_1 - 2x_3$$

subject to the bounds

$$\begin{aligned} x_1 &\geq -75 \\ x_2 &\geq -1000 \\ x_3 &\geq -25 \end{aligned}$$

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.

The **options** structure is declared and initialized by `nag_opt_init` (e04xxc), a value is assigned directly to option **options.inf\_bound** and `nag_opt_lp` (e04mfc) 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. You must **not** use the standard C function `free()` for this purpose.

### 10.1 Program Text

```
/* nag_opt_lp (e04mfc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

/* 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.
*/

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

#define A(I, J) a[(I) *tda + J]

int main(void)
{
    const char *optionsfile = "e04mfce.opt";
    Integer exit_status = 0;
    Nag_Boolean print;
    Integer n, nbnd, nclin, tda;
    Nag_E04_Opt options;
    double *a = 0, bigbnd, *bl = 0, *bu = 0, *cvec = 0, objf, *x = 0;
    Nag_Comm comm;
    NagError fail;

    INIT_FAIL(fail);

    printf("nag_opt_lp (e04mfc) Example Program Results\n");
    /* Set the actual problem dimensions.
     * n      = the number of variables.
     * nclin  = the number of general linear constraints (may be 0).
     */
    n = 3;
    nclin = 5;
    nbnd = n + nclin;
    if (n >= 1 && nclin >= 0) {
        if (!(x = NAG_ALLOC(n, double)) ||
            !(cvec = NAG_ALLOC(n, double)) ||
            !(a = NAG_ALLOC(nclin * n, double)) ||
            !(bl = NAG_ALLOC(nbnd, double)) || !(bu = NAG_ALLOC(nbnd, double)))
        {
            printf("Allocation failure\n");
            exit_status = -1;
            goto END;
        }
        tda = n;
    }
    else {
        printf("Invalid n or nclin.\n");
        exit_status = 1;
        return exit_status;
    }

    /* 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 >= 0.0 */
bl[1] = -1000.0;
bu[1] = bigbnd;

/* For Trusty:   x[2] + 25 >= 0.0 */
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.,
 *  $18*(75+x[0]) + 3*(1000+x[1]) + 102*(25+x[2]) \geq 6300$ , or
 *  $18*x[0] + 3*x[1] + 102*x[2] \geq -600$ .
 */
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  $15*x[0] - 0.5*x[1] - 25*x[2] \geq 0$ .
 */
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  $-5*x[0] + 1.5*x[1] - 25*x[2] \geq -500$ .
 */
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  $-5*x[0] - 0.5*x[1] + 75*x[2] \geq -1000$ .
 */
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;

```

```

/* Initialize options structure to null values. */
/* nag_opt_init (e04xxc).
 * Initialization function for option setting
 */
nag_opt_init(&options);
options.inf_bound = bigbnd;

/* Solve the problem. */
/* nag_opt_lp (e04mfc), see above. */
fflush(stdout);
nag_opt_lp(n, nclin, a, tda, bl, bu, cvec,
           x, &objf, &options, &comm, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_opt_lp (e04mfc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Re-solve the problem with some additonal options. */

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

/* Read additional options from a file. */
print = Nag_TRUE;
/* nag_opt_read (e04xyc).
 * Read options from a text file
 */
fflush(stdout);
nag_opt_read("e04mfc", optionsfile, &options, print, "stdout", &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_opt_read (e04xyc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

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

/* Solve the problem again. */
/* nag_opt_lp (e04mfc), see above. */
nag_opt_lp(n, nclin, a, tda, bl, bu, cvec,
           x, &objf, &options, &comm, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_opt_lp (e04mfc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Free memory allocated by nag_opt_lp (e04mfc) to pointers in options. */
/* nag_opt_free (e04xzc).
 * Memory freeing function for use with option setting
 */
nag_opt_free(&options, "all", &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_opt_free (e04xzc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

END:
    NAG_FREE(x);
    NAG_FREE(cvec);
    NAG_FREE(a);

```

```

NAG_FREE(bl);
NAG_FREE(bu);

return exit_status;
}

```

## 10.2 Program Data

nag\_opt\_lp (e04mfc) Example Program Optional Parameters

Following options for e04mfc are read by e04xyc.

begin e04mfc

```

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

```

end

## 10.3 Program Results

nag\_opt\_lp (e04mfc) Example Program Results

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.....	Nag_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: e04mfce.opt

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.....	Nag_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

## 11 Further Description

This section gives a detailed description of the algorithm used in `nag_opt_lp` (e04mfc). This, and possibly the next section, Section 12, may be omitted if the more sophisticated features of the algorithm and software are not currently of interest.

### 11.1 Overview

`nag_opt_lp` (e04mfc) is based on an inertia-controlling method due to Gill and Murray (1978) and is described in detail by Gill *et al.* (1991). Here the main features of the method are summarised. Where possible, explicit reference is made to the names of variables that are arguments of `nag_opt_lp` (e04mfc) or appear in the printed output. `nag_opt_lp` (e04mfc) 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

$$\bar{x} = x + \alpha p, \quad (1)$$

where the *steplength*  $\alpha$  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 **options.ftol**; see Section 12.2). 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` (e04mfc)). 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.

## 11.2 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

$$A_{fr}p_{fr} = 0. \quad (2)$$

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

$$A_{fr}Q_{fr} = \begin{pmatrix} 0 & T \end{pmatrix}, \quad (3)$$

where  $T$  is a nonsingular  $m_w$  by  $m_w$  upper triangular matrix (i.e.,  $t_{ij} = 0$  if  $i > j$ ), and the nonsingular  $n_{fr}$  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

$$Q_{fr} = \begin{pmatrix} Z & Y \end{pmatrix},$$

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  $0 \leq n_r \leq n_z$ , 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` (e04mfc)). In many cases,  $Z_r$  will include *all* the columns of  $Z$ .) The direction  $p_{fr}$  will satisfy (2) if

$$p_{fr} = Z_r p_r, \quad (4)$$

where  $p_r$  is any  $n_r$ -vector.

### 11.3 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  $\lambda_c$  and  $\lambda_b$  for the general and bound constraints are defined from the equations

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

Given a positive constant  $\delta$  of the order of the **machine precision**, a Lagrange multiplier  $\lambda_j$  corresponding to an inequality constraint in the working set is said to be *optimal* if  $\lambda_j \leq \delta$  when the associated constraint is at its *upper bound*, or if  $\lambda_j \geq -\delta$  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 12.3) from the working set.

If optimal multipliers occur during the feasibility phase and the sum of infeasibilities is nonzero, there is no feasible point, and `nag_opt_lp` (e04mfc) will continue until the minimum value of the sum of infeasibilities has been found. At this point, the Lagrange multiplier  $\lambda_j$  corresponding to an inequality constraint in the working set will be such that  $-(1 + \delta) \leq \lambda_j \leq \delta$  when the associated constraint is at its *upper bound*, and  $-\delta \leq \lambda_j \leq (1 + \delta)$  when the associated constraint is at its *lower bound*. Lagrange multipliers for equality constraints will satisfy  $|\lambda_j| \leq 1 + \delta$ .

If the reduced gradient is not zero, Lagrange multipliers need not be computed and the nonzero 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` (e04mfc) 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 12.3). In constructing the initial working set, constraints are excluded that would result in a large value of `Cond T`.

`nag_opt_lp` (e04mfc) 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  $\alpha_m$  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  $\alpha$  ( $\alpha \leq \alpha_m$ ) 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.

### 11.4 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 11.2.

The artificially augmented working set is given by

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

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:

$$\bar{A}_{FR} Q_{FR} = \begin{pmatrix} Z_A^T \\ A_{FR} \end{pmatrix} Q_{FR} = \begin{pmatrix} Z_A^T \\ A_{FR} \end{pmatrix} (Z_R \ Z_A \ Y) = \begin{pmatrix} 0 & \bar{T} \end{pmatrix},$$

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 12.3).

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 12.3).

Under some circumstances, a different type of artificial constraint is used when solving a linear program. Although the algorithm of nag\_opt\_lp (e04mfc) 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.,  $m_{lin} \leq n$ ). (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 12.3).

## 12 Optional Parameters

A number of optional input and output arguments to nag\_opt\_lp (e04mfc) are available through the structure argument **options**, type Nag\_E04\_Opt. An argument may be selected by assigning an appropriate value to the relevant structure member; those arguments not selected will be assigned default values. If no use is to be made of any of the optional parameters you should use the NAG defined null pointer, E04\_DEFAULT, in place of **options** when calling nag\_opt\_lp (e04mfc); the default settings will then be used for all arguments.

Before assigning values to **options** directly the structure **must** be initialized 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 initialization 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 initialization.

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).

## 12.1 Optional Parameter Checklist and Default Values

For easy reference, the following list shows the members of **options** which are valid for nag\_opt\_lp (e04mfc) together with their default values where relevant. The number  $\epsilon$  is a generic notation for *machine precision* (see nag\_machine\_precision (X02AJC)).

Nag_ProblemType prob	Nag_LP
Nag_Start start	Nag_Cold
Boolean list	Nag_TRUE
Nag_PrintType print_level	<b>Nag_Soln_Iter</b>
char outfile[80]	stdout
void (*print_fun)()	<b>NULL</b>
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(options.inf_bound, $10^{20}$ )
Integer *state	size n + nclin
double *ax	size nclin
double *lambda	size n + nclin
Integer iter	

## 12.2 Description of the Optional Parameters

**prob** – Nag\_ProblemType Default = Nag\_LP

*On entry:* specifies the problem type. The following are the two possible values of **options.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

*On entry:* specifies how the initial working set is chosen. With **options.start** = Nag\_Cold, nag\_opt\_lp (e04mfc) 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 **options.crash\_tol**; see below).

With **options.start** = Nag\_Warm, you 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 (e04mfc) will override your specification of **options.state** if necessary, so that a poor choice of the working set will not cause a fatal error. **options.start** = Nag\_Warm will be advantageous if a good estimate of the initial working set is available – for example, when nag\_opt\_lp (e04mfc) is called repeatedly to solve related problems.

*Constraint:* **options.start** = Nag\_Cold or Nag\_Warm.

**list** – Nag\_Boolean Default = Nag\_TRUE

*On entry:* if **options.list** = Nag\_TRUE the argument settings in the call to nag\_opt\_lp (e04mfc) will be printed.



**print\_level** – Nag\_PrintType Default = Nag\_Soln\_Iter

*On entry:* the level of results printout produced by nag\_opt\_lp (e04mfc). 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 12.3.

*Constraint:* **options.print\_level** = Nag\_NoPrint, Nag\_Soln, Nag\_Iter, Nag\_Soln\_Iter, Nag\_Iter\_Long, Nag\_Soln\_Iter\_Long, Nag\_Soln\_Iter\_Const or Nag\_Soln\_Iter\_Full.

**outfile** – const char[80] Default = stdout

*On entry:* 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

*On entry:* printing function defined by you; the prototype of **options.print\_fun** is

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

See Section 12.3.1 below for further details.

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

*On entry:* **options.max\_iter** specifies the maximum number of iterations to be performed by nag\_opt\_lp (e04mfc).

If you wish to check that a call to nag\_opt\_lp (e04mfc) is correct before attempting to solve the problem in full then **options.max\_iter** may be set to 0. No iterations will then be performed but the initialization stages prior to the first iteration will be processed and a listing of argument settings output if **options.list** = Nag\_TRUE (the default setting).

*Constraint:* **options.max\_iter**  $\geq 0$ .

**crash\_tol** – double Default = 0.01

*On entry:* **options.crash\_tol** is used in conjunction with the optional parameter **options.start**. When **options.start** has the default setting, i.e., **options.start** = Nag\_Cold, nag\_opt\_lp (e04mfc) selects an initial working set. The initial working set will include bounds or general inequality constraints that lie within **options.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  $|a_j^T x - l| \leq \text{options.crash\_tol} \times (1 + |l|)$ .

*Constraint:*  $0.0 \leq \text{options.crash\_tol} \leq 1.0$ .

**ftol** – doubleDefault =  $\sqrt{\epsilon}$ 

*On entry:* **options.ftol** defines the maximum acceptable 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 **options.ftol** as  $10^{-6}$ .

**nag\_opt\_lp** (e04mfc) attempts to find a feasible solution before optimizing the objective function. If the sum of infeasibilities cannot be reduced to zero, **nag\_opt\_lp** (e04mfc) finds the minimum value of the sum. Let  $S_{inf}$  be the corresponding sum of infeasibilities. If  $S_{inf}$  is quite small, it may be appropriate to raise **options.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 **options.ftol**.

*Constraint:* **options.ftol** > 0.0.

**optim\_tol** – double $\epsilon^{0.8}$ 

*On entry:* **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.

*Constraint:* **options.optim\_tol**  $\neq \epsilon$ .

**reset\_ftol** – Integer

Default = 10000

*On entry:* 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 **options.ftol** is  $\delta$ . Over a period of **options.reset\_ftol** iterations, the feasibility tolerance actually used by **nag\_opt\_lp** (e04mfc) increases from  $0.5\delta$  to  $\delta$  (in steps of  $0.5\delta/\mathbf{options.reset\_ftol}$ ).

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  $0.5\delta$ .

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

The resetting procedure is also invoked when **nag\_opt\_lp** (e04mfc) 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:*  $0 < \mathbf{options.reset\_ftol} < 10000000$ .

**fcheck** – Integer

Default = 50

*On entry:* every **options.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**  $\geq 1$ .

**inf\_bound** – doubleDefault =  $10^{20}$ 

*On entry:* **options.inf\_bound** defines the ‘infinite’ bound in the definition of the problem constraints. Any upper bound greater than or equal to **options.inf\_bound** will be regarded as  $+\infty$  (and similarly for a lower bound less than or equal to  $-\mathbf{options.inf\_bound}$ ).

*Constraint:* **options.inf\_bound** > 0.0.

**inf\_step** – double Default =  $\max(\text{options.inf\_bound}, 10^{20})$

*On entry:* **options.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 **options.prob** = Nag\_LP). If the change in  $x$  during an iteration would exceed the value of **options.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 =  $n + \text{nclin}$

*On entry:* **options.state** need not be set if the default option of **options.start** = Nag\_Cold is used as  $n + \text{nclin}$  values of memory will be automatically allocated by nag\_opt\_lp (e04mfc).

If the option **options.start** = Nag\_Warm has been chosen, **options.state** must point to a minimum of  $n + \text{nclin}$  elements of memory. This memory will already be available if the **options** structure has been used in a previous call to nag\_opt\_lp (e04mfc) from the calling program, using the same values of  $n$  and **nclin** and **options.start** = Nag\_Cold. If a previous call has not been made sufficient memory must be allocated to **options.state** by you.

When a warm start is chosen **options.state** should specify the desired status of the constraints at the start of the feasibility phase. More precisely, the first  $n$  elements of **options.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 **options.state**[ $j - 1$ ] are as follows:

<b>options.state</b> [ $j - 1$ ]	Meaning
0	The corresponding constraint should <i>not</i> 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 <b>bl</b> [ $j - 1$ ] = <b>bu</b> [ $j - 1$ ]. The values 1, 2 or 3 all have the same effect when <b>bl</b> [ $j - 1$ ] = <b>bu</b> [ $j - 1$ ].

The values  $-2$ ,  $-1$  and  $4$  are also acceptable but will be reset to zero by the function. In particular, if nag\_opt\_lp (e04mfc) has been called previously with the same values of  $n$  and **nclin**, **options.state** already contains satisfactory information. (See also the description of the optional parameter **options.start**). The function also adjusts (if necessary) the values supplied in **x** to be consistent with the values supplied in **options.state**.

*On exit:* if nag\_opt\_lp (e04mfc) exits with **fail.code** = NE\_NOERROR, NW\_SOLN\_NOT\_UNIQUE or NW\_NOT\_FEASIBLE, the values in **options.state** indicate the status of the constraints in the working set at the solution. Otherwise, **options.state** indicates the composition of the working set at the final iterate. The significance of each possible value of **options.state**[ $j - 1$ ] is as follows:

<b>options.state</b> [ $j - 1$ ]	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 <b>options.state</b> can occur only when <b>bl</b> [ $j - 1$ ] = <b>bu</b> [ $j - 1$ ].
4	This corresponds to optimality being declared with <b>x</b> [ $j - 1$ ] being temporarily fixed at its current value. This value of <b>options.state</b> can only occur when <b>fail.code</b> = NW_SOLN_NOT_UNIQUE.

**ax** – double \*

Default memory = **nclin**

*On entry:* **nclin** values of memory will be automatically allocated by nag\_opt\_lp (e04mfc) and this is the recommended method of use of **options.ax**. However you may supply memory from the calling program.

*On exit:* if **nclin** > 0, **options.ax** points to the final values of the linear constraints  $Ax$ .

**lambda** – double \*

Default memory = **n + nclin**

*On entry:* **n + nclin** values of memory will be automatically allocated by nag\_opt\_lp (e04mfc) and this is the recommended method of use of **options.lambda**. However you may supply memory from the calling program.

*On exit:* 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 **options.state**[ $j - 1$ ] = 0 (i.e., constraint  $j$  is not in the working set), **options.lambda**[ $j - 1$ ] is zero. If  $x$  is optimal, **options.lambda**[ $j - 1$ ] should be non-negative if **options.state**[ $j - 1$ ] = 1, non-positive if **options.state**[ $j - 1$ ] = 2 and zero if **options.state**[ $j - 1$ ] = 4.

**iter** – Integer

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

### 12.3 Description of Printed Output

The level of printed output can be controlled with the structure members **options.list** and **options.print\_level** (see Section 12.2). If **options.list** = Nag\_TRUE then the argument values to nag\_opt\_lp (e04mfc) are listed, whereas the printout of results is governed by the value of **options.print\_level**. The default of **options.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 (e04mfc).

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 **options.print\_level** = Nag\_Iter or Nag\_Soln\_Iter the following line of output is produced on completion of each iteration.

<b>Itn</b>	the iteration count.
<b>Jdel</b>	the index of the constraint deleted from the working set. If <b>Jdel</b> is zero, no constraint was deleted.
<b>Jadd</b>	the index of the constraint added to the working set. If <b>Jadd</b> is zero, no constraint was added.
<b>Step</b>	the step taken along the computed search direction. If a constraint is added during the current iteration (i.e., <b>Jadd</b> is positive), <b>Step</b> 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.
<b>Ninf</b>	the number of violated constraints (infeasibilities). This will be zero during the optimality phase.
<b>Sinf/Obj</b>	the value of the current objective function. If $x$ is not feasible, <b>Sinf</b> gives a weighted sum of the magnitudes of constraint violations. If $x$ is feasible, <b>Obj</b> 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 <b>Ninf</b> 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 11). At the start of the optimality phase, Nart provides an estimate of the number of non-positive eigenvalues in the reduced Hessian.
Nrz	is the number of columns of $Z_r$ (see Section 11). 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., $\text{Nrz} = n - (\text{Bnd} + \text{Lin} + \text{Nart})$ .  The value of $n_z$ , the number of columns of $Z$ (see Section 11) can be calculated as $n_z = n - (\text{Bnd} + \text{Lin})$ . 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 **options.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 **options.print\_level** = Nag\_Soln\_Iter\_Const additional values output are:

Value of x	the value of $x$ currently held in <b>x</b> .
State	the current value of <b>options.state</b> associated with $x$ .
Value of Ax	the value of $Ax$ currently held in <b>options.ax</b> .
State	the current value of <b>options.state</b> associated with $Ax$ .

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

If **options.print\_level** = Nag\_Soln\_Iter\_Full then the diagonal of  $T$  and  $Z_r$  are also output at each iteration.

When **options.print\_level** = Nag\_Soln, Nag\_Soln\_Iter, Nag\_Soln\_Iter\_Const or Nag\_Soln\_Iter\_Full the final printout from nag\_opt\_lp (e04mfc) 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 $j = 1, 2, \dots, n$ , 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 $\mathbf{bl}[j-1] \leq -\mathbf{options.inf\_bound}$ .)
Upper bound	the upper bound specified for the variable. (None indicates that $\mathbf{bu}[j-1] \geq \mathbf{options.inf\_bound}$ .)
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  $j = 1, 2, \dots, m_{lin}$  of the constraint.

### 12.3.1 Output of results via a user-defined printing function

You may also specify your 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 if you wish 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 (e04mfc). 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 **options.print\_fun**.

If **comm**→**it\_prt** = Nag\_TRUE then the results from the last iteration of nag\_opt\_lp (e04mfc) are set in the following members of **st**:

**first** – Nag\_Boolean

Nag\_TRUE on the first call to **options.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 \*

**options.ax** points to the **nclin** memory locations holding the current values  $Ax$ .

**state** – Integer \*

**options.state** points to the **n + nclin** memory locations holding the status of the variables and general linear constraints. See Section 12.2 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** = Nag\_TRUE then the Lagrange multipliers have been updated and the following members are set:

**kx** – Integer \*

Indices of the bound constraints with associated multipliers. Value of **st**→**kx**[ $i - 1$ ] is the index of the constraint with multiplier **st**→**lambda**[ $i - 1$ ], for  $i = 1, 2, \dots, \mathbf{st} \rightarrow \mathbf{bnd}$ .

**kactive** – Integer \*

Indices of the linear constraints with associated multipliers. Value of **st**→**kactive**[ $i - 1$ ] is the index of the constraint with multiplier **st**→**lambda**[**st**→**bnd** +  $i - 1$ ], for  $i = 1, 2, \dots, \mathbf{st} \rightarrow \mathbf{lin}$ .

**lambda** – double \*

The multipliers for the constraints in the working set. **options.lambda** $[i - 1]$ , for  $i = 1, 2, \dots, \mathbf{st} \rightarrow \mathbf{bnd}$  hold the multipliers for the bound constraints while the multipliers for the linear constraints are held at indices  $i - 1 = \mathbf{st} \rightarrow \mathbf{bnd}, \dots, \mathbf{st} \rightarrow \mathbf{bnd} + \mathbf{st} \rightarrow \mathbf{lin}$ .

**gq** – double \*

$\mathbf{st} \rightarrow \mathbf{gq}[i - 1]$ , for  $i = 1, 2, \dots, \mathbf{st} \rightarrow \mathbf{nart}$  hold the multipliers for the artificial constraints.

The following members of **st** are also relevant and apply when **comm**  $\rightarrow$  **it\_prt** or **comm**  $\rightarrow$  **new\_lm** is Nag\_TRUE.

**refactor** – Nag\_Boolean

Nag\_TRUE if iterative refinement performed. See Section 11.3 and optional parameter **options.reset\_ftol**.

**jmax** – Integer

If **st**  $\rightarrow$  **refactor** = Nag\_TRUE then **st**  $\rightarrow$  **jmax** holds the index of the constraint with the maximum violation.

**errmax** – double

If **st**  $\rightarrow$  **refactor** = Nag\_TRUE then **st**  $\rightarrow$  **errmax** holds the value of the maximum violation.

**moved** – Nag\_Boolean

Nag\_TRUE if some variables moved to their bounds. See the optional parameter **options.reset\_ftol**.

**nmoved** – Integer

If **st**  $\rightarrow$  **moved** = Nag\_TRUE then **st**  $\rightarrow$  **nmoved** holds the number of variables which were moved to their bounds.

**rowerr** – Nag\_Boolean

Nag\_TRUE if some constraints are not satisfied to within **options.ftol**.

**feasible** – Nag\_Boolean

Nag\_TRUE when a feasible point has been found.

If **comm**  $\rightarrow$  **sol\_prt** = Nag\_TRUE then the final result from nag\_opt\_lp (e04mfc) 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 **options.prob** = Nag\_FP and  $x$  is feasible then **st**  $\rightarrow$  **f** is set to zero.

**ax** – double \*

**options.ax** points to the **nclin** memory locations holding the final values  $Ax$ .



**state** – Integer \*

**st**→**state** points to the **n** + **nclin** memory locations holding the final status of the variables and general linear constraints. See Section 12.2 for a description of the possible status values.

**lambda** – double \*

**st**→**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 (e04mfc). Possible values of **st**→**endstate** and their correspondence to the exit value of **fail.code** are:

Value of <b>st</b> → <b>endstate</b>	Value of <b>fail.code</b>
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** – Nag\_Boolean

Will be Nag\_TRUE when the print function is called with the result of the current iteration.

**sol\_prt** – Nag\_Boolean

Will be Nag\_TRUE when the print function is called with the final result.

**new\_lm** – Nag\_Boolean

Will be Nag\_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 either before entry to nag\_opt\_lp (e04mfc) or during a call to **options.print\_fun**. The type Pointer will be void \* with a C compiler that defines void \*.

---