

# NAG Library Function Document

## nag\_opt\_bnd\_lin\_lsq (e04pcc)

### 1 Purpose

nag\_opt\_bnd\_lin\_lsq (e04pcc) solves a linear least squares problem subject to fixed lower and upper bounds on the variables.

### 2 Specification

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

void nag_opt_bnd_lin_lsq (Nag-RegularizedType itype, Integer m, Integer n,
    double a[], Integer pda, double b[], const double bl[],
    const double bu[], double tol, double x[], double *rnorm,
    Integer *nfree, double w[], Integer indx[], NagError *fail)
```

### 3 Description

Given an  $m$  by  $n$  matrix  $A$ , an  $n$ -vector  $l$  of lower bounds, an  $n$ -vector  $u$  of upper bounds, and an  $m$ -vector  $b$ , nag\_opt\_bnd\_lin\_lsq (e04pcc) computes an  $n$ -vector  $x$  that solves the least squares problem  $Ax = b$  subject to  $x_i$  satisfying  $l_i \leq x_i \leq u_i$ .

A facility is provided to return a ‘regularized’ solution, which will closely approximate a minimal length solution whenever  $A$  is not of full rank. A minimal length solution is the solution to the problem which has the smallest Euclidean norm.

The algorithm works by applying orthogonal transformations to the matrix and to the right hand side to obtain within the matrix an upper triangular matrix  $R$ . In general the elements of  $x$  corresponding to the columns of  $R$  will be the candidate nonzero solutions. If a diagonal element of  $R$  is small compared to the other members of  $R$  then this is undesirable.  $R$  will be nearly singular and the equations for  $x$  thus ill-conditioned. You may specify the tolerance used to determine the relative linear dependence of a column vector for a variable moved from its initial value.

### 4 References

Lawson C L and Hanson R J (1974) *Solving Least Squares Problems* Prentice–Hall

### 5 Arguments

1: **itype** – Nag-RegularizedType *Input*

*On entry:* provides the choice of returning a regularized solution if the matrix is not of full rank.

**itype** = Nag-Regularized

Specifies that a regularized solution is to be computed.

**itype** = Nag\_NotRegularized

Specifies that no regularization is to take place.

*Suggested value:* unless there is a definite need for a minimal length solution we recommend that

**itype** = Nag\_NotRegularized is used.

*Constraint:* **itype** = Nag-Regularized or Nag\_NotRegularized.

- 2: **m** – Integer *Input*  
*On entry:*  $m$ , the number of linear equations.  
*Constraint:*  $m \geq 0$ .
- 3: **n** – Integer *Input*  
*On entry:*  $n$ , the number of variables.  
*Constraint:*  $n \geq 0$ .
- 4: **a**[**pda** × **n**] – double *Input/Output*  
**Note:** the  $(i, j)$ th element of the matrix  $A$  is stored in  $\mathbf{a}[(j - 1) \times \mathbf{pda} + i - 1]$ .  
*On entry:* the  $m$  by  $n$  matrix  $A$ .  
*On exit:* if **itype** = Nag\_NotRegularized, **a** contains the product matrix  $QA$ , where  $Q$  is an  $m$  by  $m$  orthogonal matrix generated by nag\_opt\_bnd\_lin\_lsqr (e04pcc); otherwise **a** is unchanged.
- 5: **pda** – Integer *Input*  
*On entry:* the stride separating matrix row elements in the array **a**.  
*Constraint:*  $\mathbf{pda} \geq m$ .
- 6: **b**[**m**] – double *Input/Output*  
*On entry:* the right-hand side vector  $b$ .  
*On exit:* if **itype** = Nag\_NotRegularized, the product of  $Q$  times the original vector  $b$ , where  $Q$  is as described in argument **a**; otherwise **b** is unchanged.
- 7: **bl**[**n**] – const double *Input*  
8: **bu**[**n**] – const double *Input*  
*On entry:* **bl**[ $i - 1$ ] and **bu**[ $i - 1$ ] must specify the lower and upper bounds,  $l_i$  and  $u_i$  respectively, to be imposed on the solution vector  $x_i$ .  
*Constraint:*  $\mathbf{bl}[i - 1] \leq \mathbf{bu}[i - 1]$ , for  $i = 1, 2, \dots, n$ .
- 9: **tol** – double *Input*  
*On entry:* **tol** specifies a parameter used to determine the relative linear dependence of a column vector for a variable moved from its initial value. It determines the computational rank of the matrix. Increasing its value from  $\sqrt{\text{machine precision}}$  will increase the likelihood of additional elements of  $x$  being set to zero. It may be worth experimenting with increasing values of **tol** to determine whether the nature of the solution,  $x$ , changes significantly. In practice a value of  $\sqrt{\text{machine precision}}$  is recommended (see nag\_machine\_precision (X02AJC)).  
If on entry  $\mathbf{tol} < \sqrt{\text{machine precision}}$ , then  $\sqrt{\text{machine precision}}$  is used.  
*Suggested value:* **tol** = 0.0
- 10: **x**[**n**] – double *Output*  
*On exit:* the solution vector  $x$ .
- 11: **rnorm** – double \* *Output*  
*On exit:* the Euclidean norm of the residual vector  $b - Ax$ .
- 12: **nfree** – Integer \* *Output*  
*On exit:* indicates the number of components of the solution vector that are not at one of the constraints.

13: **w[n]** – double

Output

*On exit:* contains the dual solution vector. The magnitude of  $\mathbf{w}[i-1]$  gives a measure of the improvement in the objective value if the corresponding bound were to be relaxed so that  $x_i$  could take different values.

A value of  $\mathbf{w}[i-1]$  equal to the special value  $-999.0$  is indicative of the matrix  $A$  not having full rank. It is only likely to occur when **itype** = Nag\_NotRegularized. However a matrix may have less than full rank without  $\mathbf{w}[i-1]$  being set to  $-999.0$ . If **itype** = Nag\_NotRegularized then the values contained in **w** (other than those set to  $-999.0$ ) may be unreliable; the corresponding values in **indx** may likewise be unreliable. If you have any doubts set **itype** = Nag\_Regularized. Otherwise the values of  $\mathbf{w}[i-1]$  have the following meaning:

- $\mathbf{w}[i-1] = 0$   
if  $x_i$  is unconstrained.
- $\mathbf{w}[i-1] < 0$   
if  $x_i$  is constrained by its lower bound.
- $\mathbf{w}[i-1] > 0$   
if  $x_i$  is constrained by its upper bound.
- $\mathbf{w}[i-1]$   
may be any value if  $l_i = u_i$ .

14: **indx[n]** – Integer

Output

*On exit:* the contents of this array describe the components of the solution vector as follows:

- indx**[ $i-1$ ], for  $i = 1, 2, \dots, \mathbf{nfree}$   
These elements of the solution have not hit a constraint; i.e.,  $\mathbf{w}[i-1] = 0$ .
- indx**[ $i-1$ ], for  $i = \mathbf{nfree} + 1, \dots, k$   
These elements of the solution have been constrained by either the lower or upper bound.
- indx**[ $i-1$ ], for  $i = k + 1, \dots, \mathbf{n}$   
These elements of the solution are fixed by the bounds; i.e.,  $\mathbf{bl}[i-1] = \mathbf{bu}[i-1]$ .

Here  $k$  is determined from **nfree** and the number of fixed components. (Often the latter will be 0, so  $k$  will be  $\mathbf{n} - \mathbf{nfree}$ .)

15: **fail** – NagError \*

Input/Output

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

## 6 Error Indicators and Warnings

### NE\_ALLOC\_FAIL

Dynamic memory allocation failed.  
See Section 2.3.1.2 in How to Use the NAG Library and its Documentation for further information.

### NE\_BAD\_PARAM

On entry, argument  $\langle value \rangle$  had an illegal value.

### NE\_CONVERGENCE

The function failed to converge in  $3 \times n$  iterations. This is not expected. Please contact NAG.

### NE\_INT

On entry,  $\mathbf{m} = \langle value \rangle$ .  
Constraint:  $\mathbf{m} \geq 0$ .

On entry,  $\mathbf{n} = \langle \text{value} \rangle$ .  
 Constraint:  $\mathbf{n} \geq 0$ .

## NE\_INT\_2

On entry,  $\mathbf{m} = \langle \text{value} \rangle$  and  $\mathbf{pda} = \langle \text{value} \rangle$ .  
 Constraint:  $\mathbf{pda} \geq \mathbf{m}$ .

## NE\_INTERNAL\_ERROR

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.

An unexpected error has been triggered by this function. Please contact NAG.  
 See Section 2.7.6 in How to Use the NAG Library and its Documentation for further information.

## NE\_NO\_LICENCE

Your licence key may have expired or may not have been installed correctly.  
 See Section 2.7.5 in How to Use the NAG Library and its Documentation for further information.

## NE\_REAL\_2

On entry, when  $i = \langle \text{value} \rangle$ ,  $\mathbf{bl}[i - 1] = \langle \text{value} \rangle$  and  $\mathbf{bu}[i - 1] = \langle \text{value} \rangle$ .  
 Constraint:  $\mathbf{bl}[i - 1] \leq \mathbf{bu}[i - 1]$ .

## 7 Accuracy

Orthogonal rotations are used.

## 8 Parallelism and Performance

`nag_opt_bnd_lin_lsq (e04pcc)` makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

Please consult the x06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also consult the Users' Note for your implementation for any additional implementation-specific information.

## 9 Further Comments

If either  $\mathbf{m}$  or  $\mathbf{n}$  is zero on entry then `nag_opt_bnd_lin_lsq (e04pcc)` sets `fail.code` = NE\_NOERROR and simply returns without setting any other output arguments.

## 10 Example

The example minimizes  $\|Ax - b\|_2$  where

$$A = \begin{pmatrix} 0.05 & 0.05 & 0.25 & -0.25 \\ 0.25 & 0.25 & 0.05 & -0.05 \\ 0.35 & 0.35 & 1.75 & -1.75 \\ 1.75 & 1.75 & 0.35 & -0.35 \\ 0.30 & -0.30 & 0.30 & 0.30 \\ 0.40 & -0.40 & 0.40 & 0.40 \end{pmatrix}$$

and

$$b = (1.0 \ 2.0 \ 3.0 \ 4.0 \ 5.0 \ 6.0)^T$$

subject to  $1 \leq x \leq 5$ .

## 10.1 Program Text

```

/* nag_opt_bnd_lin_lsqr (e04pcc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

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

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

int main(void)
{
    Integer exit_status = 0;
    double tol = 0.0;
    Nag_RegularizedType itype = Nag_NotRegularized;
    double rnorm;
    Integer i, j, m, n, nfree, pda;
    double *a = 0, *b = 0, *bl = 0, *bu = 0, *w = 0, *x = 0;
    Integer *indx = 0;
    NagError fail;

    INIT_FAIL(fail);

    printf("nag_opt_bnd_lin_lsqr (e04pcc) Example Program Results\n\n");

#ifdef _WIN32
    scanf_s("%*[^\\n] "); /* Skip heading in data file */
#else
    scanf("%*[^\\n] "); /* Skip heading in data file */
#endif

#ifdef _WIN32
    scanf_s("%" NAG_IFMT "%" NAG_IFMT "%*[^\\n]", &m, &n);
#else
    scanf("%" NAG_IFMT "%" NAG_IFMT "%*[^\\n]", &m, &n);
#endif

    if (m < 0 || n < 0) {
        printf("Invalid m or n.\n");
        exit_status = 1;
        goto END;
    }

    pda = m;

    if (!(a = NAG_ALLOC(pda * n, double)) ||
        !(b = NAG_ALLOC(m, double)) ||
        !(w = NAG_ALLOC(n, double)) ||
        !(bl = NAG_ALLOC(n, double)) ||
        !(bu = NAG_ALLOC(n, double)) ||
        !(x = NAG_ALLOC(n, double)) || !(indx = NAG_ALLOC(n, Integer)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }

    /* Read the matrix A */
    for (i = 0; i < m; i++)
        for (j = 0; j < n; j++)
#ifdef _WIN32
            scanf_s("%lf", &A(i, j));
#else

```

```

        scanf("%lf", &A(i, j));
    #endif

    #ifdef _WIN32
        scanf_s("%*[^\\n] "); /* Remove remainder of line */
    #else
        scanf("%*[^\\n] "); /* Remove remainder of line */
    #endif

    /* Read the right-hand side vector b */
    for (j = 0; j < m; j++)
    #ifdef _WIN32
        scanf_s("%lf", &b[j]);
    #else
        scanf("%lf", &b[j]);
    #endif

    #ifdef _WIN32
        scanf_s("%*[^\\n] ");
    #else
        scanf("%*[^\\n] ");
    #endif

    /* Read the lower bounds vector bl */
    for (i = 0; i < n; i++)
    #ifdef _WIN32
        scanf_s("%lf", &bl[i]);
    #else
        scanf("%lf", &bl[i]);
    #endif

    #ifdef _WIN32
        scanf_s("%*[^\\n] ");
    #else
        scanf("%*[^\\n] ");
    #endif

    /* Read the upper bounds vector bu */
    for (i = 0; i < n; i++)
    #ifdef _WIN32
        scanf_s("%lf", &bu[i]);
    #else
        scanf("%lf", &bu[i]);
    #endif

    #ifdef _WIN32
        scanf_s("%*[^\\n] ");
    #else
        scanf("%*[^\\n] ");
    #endif

    /* nag_opt_bnd_lin_lsqr (e04pcc). Computes the least squares solution
       to a set of linear equations subject to fixed upper and lower
       bounds on the variables */
    nag_opt_bnd_lin_lsqr(itype, m, n, a, pda, b, bl, bu, tol, x, &rnrm, &nfree,
                        w, indx, &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_opt_bnd_lin_lsqr (e04pcc).\\n%s\\n", fail.message);
        exit_status = 2;
        goto END;
    }

    printf("Solution vector\\n");
    for (i = 0; i < n; i++)
        printf("%9.4f", x[i]);
    printf("\\n\\n");

    printf("Dual Solution\\n");
    for (i = 0; i < n; i++)
        printf("%9.4f", w[i]);
    printf("\\n\\n");

```

```

    printf("Residual %9.4f\n", rnorm);

END:
    NAG_FREE(a);
    NAG_FREE(b);
    NAG_FREE(bl);
    NAG_FREE(bu);
    NAG_FREE(w);
    NAG_FREE(x);
    NAG_FREE(indx);

    return exit_status;
}

```

## 10.2 Program Data

```

nag_opt_bnd_lin_lsq (e04pcc) Example Program Data
6 4                                     : m, n
0.05 0.05 0.25 -0.25
0.25 0.25 0.05 -0.05
0.35 0.35 1.75 -1.75
1.75 1.75 0.35 -0.35
0.30 -0.30 0.30 0.30
0.40 -0.40 0.40 0.40 : matrix A
1.0 2.0 3.0 4.0 5.0 6.0 : vector b
1.0 1.0 1.0 1.0 : Lower bounds
5.0 5.0 5.0 5.0 : Upper bounds

```

## 10.3 Program Results

```

nag_opt_bnd_lin_lsq (e04pcc) Example Program Results

Solution vector
1.8133 1.0000 5.0000 4.3467

Dual Solution
0.0000 -2.7200 2.7200 0.0000

Residual 3.4246

```

---