

# NAG Library Function Document

## nag\_transport (h03abc)

### 1 Purpose

nag\_transport (h03abc) solves the classical transportation ('Hitchcock') problem.

### 2 Specification

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

void nag_transport (const double cost[], Integer tdcost,
                   const double avail[], Integer navail, const double req[], Integer nreq,
                   Integer maxit, Integer *numit, double optq[], Integer source[],
                   Integer dest[], double *optcost, double unitcost[], NagError *fail)
```

### 3 Description

nag\_transport (h03abc) solves the transportation problem by minimizing

$$z = \sum_i^{m_a} \sum_j^{m_b} c_{ij} x_{ij}.$$

subject to the constraints

$$\sum_j^{m_b} x_{ij} = A_i \quad (\text{availabilities})$$

$$\sum_i^{m_a} x_{ij} = B_j \quad (\text{requirements})$$

where the  $x_{ij}$  can be interpreted as quantities of goods sent from source  $i$  to destination  $j$ , for  $i = 1, 2, \dots, m_a$  and  $j = 1, 2, \dots, m_b$ , at a cost of  $c_{ij}$  per unit, and it is assumed that  $\sum_i^{m_a} A_i = \sum_j^{m_b} B_j$  and  $x_{ij} \geq 0$ .

nag\_transport (h03abc) uses the 'stepping stone' method, modified to accept degenerate cases.

### 4 References

Hadley G (1962) *Linear Programming* Addison–Wesley

### 5 Arguments

- 1: **cost**[navail × tdcost] – const double *Input*  
*On entry:* **cost**[( $i - 1$ ) × tdcost +  $j - 1$ ] contains the coefficients  $c_{ij}$ , for  $i = 1, 2, \dots, m_a$  and  $j = 1, 2, \dots, m_b$ .
- 2: **tdcost** – Integer *Input*  
*On entry:* the stride separating matrix column elements in the array **cost**.  
*Constraint:* **tdcost** ≥ **nreq**.
- 3: **avail**[navail] – const double *Input*  
*On entry:* **avail**[ $i - 1$ ] must be set to the availabilities  $A_i$ , for  $i = 1, 2, \dots, m_a$ ;

- 4: **navail** – Integer *Input*  
*On entry:* the number of sources,  $m_a$ .  
*Constraint:* **navail**  $\geq 1$ .
- 5: **req[nreq]** – const double *Input*  
*On entry:* **req**[ $j - 1$ ] must be set to the requirements  $B_j$ , for  $j = 1, 2, \dots, m_b$ .
- 6: **nreq** – Integer *Input*  
*On entry:* the number of destinations,  $m_b$ .  
*Constraint:* **nreq**  $\geq 1$ .
- 7: **maxit** – Integer *Input*  
*On entry:* the maximum number of iterations allowed.  
*Constraint:* **maxit**  $\geq 1$ .
- 8: **numit** – Integer \* *Output*  
*On exit:* the number of iterations performed.
- 9: **optq[navail + nreq]** – double *Output*  
*On exit:* **optq**[ $k - 1$ ], for  $k = 1, 2, \dots, m_a + m_b - 1$ , contains the optimal quantities  $x_{ij}$  which, when sent from source  $i = \mathbf{source}[k - 1]$  to destination  $j = \mathbf{dest}[k - 1]$ , minimize  $z$ .
- 10: **source[navail + nreq]** – Integer *Output*  
*On exit:* **source**[ $k - 1$ ], for  $k = 1, 2, \dots, m_a + m_b - 1$ , contains the source indices of the optimal solution (see **optq** above).
- 11: **dest[navail + nreq]** – Integer *Output*  
*On exit:* **dest**[ $k - 1$ ], for  $k = 1, 2, \dots, m_a + m_b - 1$ , contains the destination indices of the optimal solution (see **optq** above).
- 12: **optcost** – double \* *Output*  
*On exit:* the value of the minimized total cost.
- 13: **unitcost[navail + nreq]** – double *Output*  
*On exit:* **unitcost**[ $k - 1$ ], for  $k = 1, 2, \dots, m_a + m_b - 1$ , contains the unit cost  $c_{ij}$  associated with the route from source  $i = \mathbf{source}[k - 1]$  to destination  $j = \mathbf{dest}[k - 1]$ .
- 14: **fail** – NagError \* *Input/Output*  
The NAG error argument (see Section 3.6 in the Essential Introduction).

## 6 Error Indicators and Warnings

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

On entry, **tdcost** =  $\langle value \rangle$  while **nreq** =  $\langle value \rangle$ . These arguments must satisfy **tdcost**  $\geq$  **nreq**.

### NE\_ALLOC\_FAIL

Dynamic memory allocation failed.

**NE\_INT\_ARG\_LT**

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

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

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

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

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

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

**NE\_REQ\_AVAIL**

The relative difference between the sum of availabilities and the sum of requirements is greater than *machine precision*. relative difference =  $\langle value \rangle$ , *machine precision* =  $\langle value \rangle$ .

**NE\_TOO\_MANY**

Too many iterations ( $\langle value \rangle$ )

**7 Accuracy**

The computations are stable.

**8 Parallelism and Performance**

Not applicable.

**9 Further Comments**

An a priori estimate of the run time for a particular problem is difficult to obtain.

**10 Example**

A company has three warehouses and three stores. The warehouses have a surplus of 12 units of a given commodity divided between them as follows:

Warehouse	Surplus
1	1
2	5
3	6

The stores altogether need 12 units of commodity, with the following requirements:

Store	Requirement
1	4
2	4
3	4

Costs of shipping one unit of the commodity from warehouse  $i$  to store  $j$  are displayed in the following matrix:

		Store		
		1	2	3
Warehouse	1	8	8	11
	2	5	8	14
	3	4	3	10

It is required to find the units of commodity to be moved from the warehouses to the stores, such that the transportation costs are minimized. The maximum number of iterations allowed is 200.

## 10.1 Program Text

```

/* nag_transport (h03abc) Example Program.
 *
 * Copyright 1992 Numerical Algorithms Group.
 *
 * Mark 3, 1992.
 *
 * Mark 5 revised, 1998.
 * Mark 8 revised, 2004.
 *
 */

#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <nagh03.h>

#define COST(I, J) cost[(I) *tdcost + J]
int main(void)
{
    Integer  *dest = 0, exit_status = 0, i, m, maxit, navail, nreq, numit;
    Integer  *source = 0;
    Integer  tdcost;
    NagError fail;
    double   *avail = 0, *cost = 0, optcost, *optq = 0, *req = 0, *unitcost = 0;

    INIT_FAIL(fail);

    printf("nag_transport (h03abc) Example Program Results\n");
    navail = 3;
    nreq = 3;
    m = navail + nreq;

    if (!(cost = NAG_ALLOC(navail*nreq, double)) ||
        !(avail = NAG_ALLOC(navail, double)) ||
        !(req = NAG_ALLOC(nreq, double)) ||
        !(optq = NAG_ALLOC(navail+nreq, double)) ||
        !(unitcost = NAG_ALLOC(navail+nreq, double)) ||
        !(source = NAG_ALLOC(navail+nreq, Integer)) ||
        !(dest = NAG_ALLOC(navail+nreq, Integer)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
    tdcost = nreq;

    COST(0, 0) = 8.0;
    COST(0, 1) = 8.0;
    COST(0, 2) = 11.0;
    COST(1, 0) = 5.0;
    COST(1, 1) = 8.0;
    COST(1, 2) = 14.0;
    COST(2, 0) = 4.0;
    COST(2, 1) = 3.0;
    COST(2, 2) = 10.0;

    avail[0] = 1.0;
    avail[1] = 5.0;
    avail[2] = 6.0;

    req[0] = 4.0;
    req[1] = 4.0;
    req[2] = 4.0;

    maxit = 200;

```

```

/* nag_transport (h03abc).
 * Classical transportation algorithm
 */
nag_transport(cost, tdcost, avail, navail, req, nreq, maxit, &numit,
              optq, source, dest, &optcost, unitcost, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_transport (h03abc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

printf("\nGoods From      To      Number      Cost per Unit\n");
for (i = 0; i < m-1; i++)
    printf("    %ld      %ld      %8.3f      %8.3f\n",
          source[i], dest[i], optq[i], unitcost[i]);
printf("\nTotal Cost %8.4f\n", optcost);
END:
NAG_FREE(cost);
NAG_FREE(avail);
NAG_FREE(req);
NAG_FREE(optq);
NAG_FREE(unitcost);
NAG_FREE(source);
NAG_FREE(dest);
return exit_status;
}

```

## 10.2 Program Data

None.

## 10.3 Program Results

nag\_transport (h03abc) Example Program Results

Goods From	To	Number	Cost per Unit
3	2	4.000	3.000
3	3	2.000	10.000
2	3	1.000	14.000
1	3	1.000	11.000
2	1	4.000	5.000

Total Cost 77.0000

---