nag_transport (h03abc)

1. Purpose

nag_transport solves the classical transportation ('Hitchcock') problem.

2. Specification

```
#include <nag.h>
#include <nagh03.h>

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

3. Description

nag_transport solves the transportation problem by minimizing

$$z = \sum_{i}^{m_a} \sum_{j}^{m_b} c_{ij} x_{ij}.$$

subject to the constraints

$$\sum_{j}^{m_b} x_{ij} = A_i \qquad \text{(availabilities)}$$

$$\sum_{i}^{m_a} x_{ij} = B_j \qquad \text{(requirements)}$$

where the x_{ij} can be interpreted as quantities of goods sent from source i to destination j, for $i = 1, 2, \ldots, m_a$; $j = 1, 2, \ldots, m_b$, 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 uses the 'stepping stone' method, modified to accept degenerate cases.

4. Parameters

cost[nreq][tdcost]
Input: **cost**$[i-1][j-1]$ contains the coefficients c_{ij}, for $i = 1, 2, \ldots, m_a$; $j = 1, 2, \ldots, m_b$.

tdcost
Input: the second dimension of the array **cost** as declared in the function from which nag_transport is called.
Constraint: **tdcost** $\geq$ **nreq**.

avail[navail]
Input: **avail**$[i-1]$ must be set to the availabilities A_i, for $i = 1, 2, \ldots, m_a$;

navail
Input: the number of sources, m_a.
Constraint: **navail** ≥ 1.

req[nreq]
Input: **req**$[j-1]$ must be set to the requirements B_j, for $j = 1, 2, \ldots, m_b$.

nreq
Input: the number of destinations, m_b.
Constraint: **nreq** ≥ 1.

maxit
Input: the maximum number of iterations allowed.
Constraint: **maxit** ≥ 1.

numit
Output: the number of iterations performed.

optq[navail+nreq]
Output: **optq**$[k-1]$, for $k = 1, 2, \ldots, m_a + m_b - 1$, contains the optimal quantities x_{ij} which, when sent from source $i =$ **source**$[k-1]$ to destination $j =$ **dest**$[k-1]$, minimize z.

source[navail+nreq]
Output: **source**$[k-1]$, for $k = 1, 2, \ldots, m_a + m_b - 1$, contains the source indices of the optimal solution (see **optq** above).

dest[navail+nreq]
Output: **dest**$[k-1]$, for $k = 1, 2, \ldots, m_a + m_b - 1$, contains the destination indices of the optimal solution (see **optq** above).

optcost
Output: the value of the minimized total cost.

unitcost[navail+nreq]
Output: **unitcost**$[k-1]$, for $k = 1, 2, \ldots, m_a + m_b - 1$, contains the unit cost c_{ij} associated with the route from source $i =$ **source**$[k-1]$ to destination $j =$ **dest**$[k-1]$.

fail
The NAG error parameter, see the Essential Introduction to the NAG C Library.

5. Error Indications and Warnings

NE_INT_ARG_LT
On entry, **navail** must not be less than 1: **navail** = ⟨*value*⟩.
On entry, **nreq** must not be less than 1: **nreq** = ⟨*value*⟩.
On entry, **maxit** must not be less than 1: **maxit** = ⟨*value*⟩.

NE_2_INT_ARG_LT
On entry **tdcost** = ⟨*value*⟩while **nreq** = ⟨*value*⟩. These parameters must satisfy **tdcost** $\geq$ **nreq**.

NE_REQ_AVAIL
The relative difference between the sum of availabilities and the sum of requirements is greater than ***machine precision***.
relative difference = ⟨*value*⟩, ***machine precision*** = ⟨*value*⟩

NE_TOO_MANY
Too many iterations (⟨*value*⟩)

NE_ALLOC_FAIL
Memory allocation failed.

6. Further Comments

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

6.1. Accuracy

The computations are stable.

6.2. References

Hadley, G. (1962) *Linear Programming* Addison-Wesley, New York.

7. See Also

None.

8. 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.

8.1. Program Text

```
/* nag_transport(h03abc) Example Program.
 *
 * Copyright 1992 Numerical Algorithms Group.
 *
 * Mark 3, 1992.
 *
 */

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

#define NAVAIL 3
#define NREQ   3
#define M      NAVAIL+NREQ
#define TDCOST 5

main()
{

  double cost[NAVAIL][TDCOST];
  double avail[NAVAIL], req[NREQ], optq[M];
  Integer source[M], dest[M];
  double unitcost[M];
  Integer tdcost, navail, nreq, m;
  Integer maxit, numit;
  double optcost;
  Integer i;
  static NagError fail;

  Vprintf("h03abc Example Program Results\n");
  tdcost = TDCOST;
  navail = NAVAIL;
  nreq = NREQ;
  m = M;

  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;

  h03abc((double *)cost, tdcost, avail, navail, req, nreq, maxit, &numit,
         optq, source, dest, &optcost, unitcost, &fail);

  Vprintf("\nGoods From     To      Number        Cost per Unit\n");
  for (i=0; i < m-1; i++)
    Vprintf("   %ld            %ld    %8.3f         %8.3f\n",
            source[i], dest[i], optq[i], unitcost[i]);
  Vprintf("\nTotal Cost %8.4f\n", optcost);
  exit(EXIT_SUCCESS);
}
```

8.2. Program Data

None.

8.3. Program Results

```
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
```