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

\[ z = \sum_{i} \sum_{j} c_{ij} x_{ij}. \]

subject to the constraints

\[ \sum_{j} x_{ij} = A_i \quad \text{(availabilities)} \]
\[ \sum_{i} x_{ij} = B_j \quad \text{(requirements)} \]

where the \( x_{ij} \) can be interpreted as quantities of goods sent from source \( i \) to destination \( j \), for \( i = 1, 2, \ldots, m_a \) and \( j = 1, 2, \ldots, m_b \), at a cost of \( c_{ij} \) per unit, and it is assumed that \( \sum_{i} A_i = \sum_{j} B_j \) 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
   On entry: cost[(i − 1) × tdcost + j − 1] contains the coefficients \( c_{ij} \), for \( i = 1, 2, \ldots, m_a \) and \( j = 1, 2, \ldots, m_b \).

2: tdcost – Integer
   On entry: the stride separating matrix column elements in the array cost.
   Constraint: tdcost ≥ nreq.

3: avail[navail] – const double
   On entry: avail[i − 1] must be set to the availabilities \( A_i \), for \( i = 1, 2, \ldots, m_a \);
4: navail – Integer
   On entry: the number of sources, ma.
   Constraint: navail ≥ 1.

5: req[nreq] – const double
   On entry: req[j – 1] must be set to the requirements B_j, for j = 1, 2, …, mb.

6: nreq – Integer
   On entry: the number of destinations, mb.
   Constraint: nreq ≥ 1.

7: maxit – Integer
   On entry: the maximum number of iterations allowed.
   Constraint: maxit ≥ 1.

8: numit – Integer *
   On exit: the number of iterations performed.

9: optq[navail + nreq] – double
   On exit: optq[k – 1], for k = 1, 2, …, ma + mb – 1, contains the optimal quantities x_ij which, when sent from source i = source[k – 1] to destination j = dest[k – 1], minimize $z$.

10: source[navail + nreq] – Integer
    On exit: source[k – 1], for k = 1, 2, …, ma + mb – 1, contains the source indices of the optimal solution (see optq above).

11: dest[navail + nreq] – Integer
    On exit: dest[k – 1], for k = 1, 2, …, ma + mb – 1, contains the destination indices of the optimal solution (see optq above).

12: octcost – double *
    On exit: the value of the minimized total cost.

13: unitcost[navail + nreq] – double
    On exit: unitcost[k – 1], for k = 1, 2, …, ma + mb – 1, contains the unit cost c_ij associated with the route from source i = source[k – 1] to destination j = dest[k – 1].

14: fail – NagError *
    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 = ⟨value⟩ while nreq = ⟨value⟩. These arguments must satisfy tdcost ≥ nreq.

Ne_ALLOC_FAIL
   Dynamic memory allocation failed.
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.

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.

Too many iterations (\langle value\rangle)

The computations are stable.

Not applicable.

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

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:

<table>
<thead>
<tr>
<th>Warehouse</th>
<th>Surplus</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Store</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Store</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>8</td>
<td>11</td>
</tr>
<tr>
<td>Warehouse 2</td>
<td>5</td>
<td>8</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>3</td>
<td>10</td>
</tr>
</tbody>
</table>

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 2014 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 || !navail || !nreq || !numit || !source || !tdcost)
    {
        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;

END:
/* nag_transport (h03abc).
 * Classical transportation algorithm
 */

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

if (fail.code != NE_NOERROR)
{
    printf("Error from nag_transport (h03abc).
%s
", fail.message);
    exit_status = 1;
    goto END;
}

printf("%s
", fail.message);
exit_status = 1;
printf("Goods From To Number Cost per Unit
");
for (i = 0; i < m-1; i++)
    printf("%d  %d  %8.3f  %8.3f
", source[i], dest[i], optq[i], unitcost[i]);
printf("Total Cost %8.4f
", 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

<table>
<thead>
<tr>
<th>Goods From</th>
<th>To</th>
<th>Number</th>
<th>Cost per Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>4.000</td>
<td>3.000</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2.000</td>
<td>10.000</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1.000</td>
<td>14.000</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>1.000</td>
<td>11.000</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4.000</td>
<td>5.000</td>
</tr>
</tbody>
</table>

Total Cost 77.0000