NAG Library Routine Document

E04RHF

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details.

1 Purpose

E04RHF is a part of the NAG optimization modelling suite and defines bounds on the variables of the problem.

2 Specification

```
SUBROUTINE E04RHF (HANDLE, NVAR, BL, BU, IFAIL)

INTEGER              NVAR, IFAIL
REAL (KIND=nag_wp)   BL(NVAR), BU(NVAR)
TYPE (C_PTR)         HANDLE
```

3 Description

After the initialization routine E04RAF has been called, E04RHF may be used to define the variable bounds $l_x \leq x \leq u_x$ of the problem unless the bounds have already been defined. This will typically be used for problems, such as quadratic programming (QP)

$$\begin{array}{llr} \underset{x \in \mathbb{R}^n}{\text{minimize}} & \frac{1}{2}x^{\mathrm{T}}Hx + c^{\mathrm{T}}x & \text{(a)} \\ \text{subject to} & l_B \leq Bx \leq u_B & \text{(b)} \\ & l_x \leq x \leq u_x & \text{(c)} \end{array} \quad (1)$$

nonlinear programming (NLP)

$$\begin{array}{llr} \underset{x \in \mathbb{R}^n}{\text{minimize}} & f(x) & \text{(a)} \\ \text{subject to} & l_g \leq g(x) \leq u_g & \text{(b)} \\ & l_B \leq Bx \leq u_B & \text{(c)} \\ & l_x \leq x \leq u_x & \text{(d)} \end{array} \quad (2)$$

linear semidefinite programming (SDP)

$$\begin{array}{llr} \underset{x \in \mathbb{R}^n}{\text{minimize}} & c^{\mathrm{T}}x & \text{(a)} \\ \text{subject to} & \sum_{i=1}^{n} x_i A_i^k - A_0^k \succeq 0, \quad k = 1, \ldots, m_A & \text{(b)} \\ & l_B \leq Bx \leq u_B & \text{(c)} \\ & l_x \leq x \leq u_x & \text{(d)} \end{array} \quad (3)$$

or semidefinite programming with bilinear matrix inequalities (BMI-SDP)

$$\begin{array}{llr} \underset{x \in \mathbb{R}^n}{\text{minimize}} & \frac{1}{2}x^{\mathrm{T}}Hx + c^{\mathrm{T}}x & \text{(a)} \\ \text{subject to} & \sum_{i,j=1}^{n} x_i x_j Q_{ij}^k + \sum_{i=1}^{n} x_i A_i^k - A_0^k \succeq 0, \quad k = 1, \ldots, m_A & \text{(b)} \\ & l_B \leq Bx \leq u_B & \text{(c)} \\ & l_x \leq x \leq u_x & \text{(d)} \end{array} \quad (4)$$

where l_x and u_x are n-dimensional vectors. Note that upper and lower bounds are specified for all the variables. This form allows full generality in specifying various types of constraint. If certain bounds are not present, the associated elements of l_x or u_x may be set to special values that are treated as $-\infty$ or $+\infty$. See the description of the optional parameter **Infinite Bound Size** of the solvers in the suite, E04STF and E04SVF. Its value is denoted as $bigbnd$ further in this text. Note that the bounds are interpreted based on its value at the time of calling this routine and any later alterations to **Infinite Bound Size** will not affect these constraints.

See E04RAF for more details.

4 References

Candes E and Recht B (2009) Exact matrix completion via convex optimization *Foundations of Computation Mathematics (Volume 9)* 717–772

5 Arguments

1: HANDLE – TYPE (C_PTR) *Input*

On entry: the handle to the problem. It needs to be initialized by E04RAF and **must not** be changed.

2: NVAR – INTEGER *Input*

On entry: n, the number of decision variables x in the problem. It must be unchanged from the value set during the initialization of the handle by E04RAF.

3: BL(NVAR) – REAL (KIND=nag_wp) array *Input*

4: BU(NVAR) – REAL (KIND=nag_wp) array *Input*

On entry: l_x, BL and u_x, BU define lower and upper bounds on the variables, respectively. To specify a nonexistent lower bound (i.e., $l_j = -\infty$), set $\mathrm{BL}(j) \leq -bigbnd$; to specify a nonexistent upper bound (i.e., $u_j = \infty$), set $\mathrm{BU}(j) \geq bigbnd$. Fixing of the variables is not allowed in this release, however, this limitation will be removed in a future release.

Constraints:

$\mathrm{BL}(j) < \mathrm{BU}(j)$, for $j = 1, 2, \dots, \mathrm{NVAR}$;
$\mathrm{BL}(j) < bigbnd$, for $j = 1, 2, \dots, \mathrm{NVAR}$;
$\mathrm{BU}(j) > -bigbnd$, for $j = 1, 2, \dots, \mathrm{NVAR}$.

5: IFAIL – INTEGER *Input/Output*

On entry: IFAIL must be set to 0, -1 or 1. If you are unfamiliar with this argument you should refer to Section 3.4 in How to Use the NAG Library and its Documentation for details.

For environments where it might be inappropriate to halt program execution when an error is detected, the value -1 or 1 is recommended. If the output of error messages is undesirable, then the value 1 is recommended. Otherwise, the recommended value is -1. **When the value -1 or 1 is used it is essential to test the value of IFAIL on exit.**

On exit: $\mathrm{IFAIL} = 0$ unless the routine detects an error or a warning has been flagged (see Section 6).

6 Error Indicators and Warnings

If on entry $\mathrm{IFAIL} = 0$ or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

Errors or warnings detected by the routine:

$\mathrm{IFAIL} = 1$

The supplied HANDLE does not define a valid handle to the data structure for the NAG optimization modelling suite. It has not been initialized by E04RAF or it has been corrupted.

IFAIL $= 2$

The problem cannot be modified in this phase any more, the solver has already been called.

IFAIL $= 3$

Variable bounds have already been defined.

IFAIL $= 4$

On entry, NVAR $= \langle value \rangle$, expected value $= \langle value \rangle$.
Constraint: NVAR must match the value given during initialization of HANDLE.

IFAIL $= 10$

On entry, $j = \langle value \rangle$, BL$(j) = \langle value \rangle$, $bigbnd = \langle value \rangle$.
Constraint: BL$(j) < bigbnd$.

On entry, $j = \langle value \rangle$, BL$(j) = \langle value \rangle$ and BU$(j) = \langle value \rangle$.
Constraint: BL$(j) <$ BU(j).

On entry, $j = \langle value \rangle$, BU$(j) = \langle value \rangle$, $bigbnd = \langle value \rangle$.
Constraint: BU$(j) > -bigbnd$.

IFAIL $= -99$

An unexpected error has been triggered by this routine. Please contact NAG.

See Section 3.9 in How to Use the NAG Library and its Documentation for further information.

IFAIL $= -399$

Your licence key may have expired or may not have been installed correctly.

See Section 3.8 in How to Use the NAG Library and its Documentation for further information.

IFAIL $= -999$

Dynamic memory allocation failed.

See Section 3.7 in How to Use the NAG Library and its Documentation for further information.

7 Accuracy

Not applicable.

8 Parallelism and Performance

E04RHF is not threaded in any implementation.

9 Further Comments

None.

10 Example

There is a vast number of problems which can be reformulated as SDP. This example follows Candes and Recht (2009) to show how a rank minimization problem can be approximated by SDP. In addition, it demonstrates how to work with the monitor mode of E04SVF.

The problem can be stated as follows: Let's have m respondents answering k questions where they express their preferences as a number between 0 and 1 or the question can be left unanswered. The task is to fill in the missing entries, i.e., to guess the unexpressed preferences. This problem falls into the category of matrix completion. The idea is to choose the missing entries to minimize the rank of the matrix as it is commonly believed that only a few factors contribute to an individual's tastes or preferences.

Rank minimization is in general NP-hard but it can be approximated by a heuristic, minimizing the nuclear norm of the matrix. The nuclear norm of a matrix is the sum of its singular values. A rank deficient matrix must have (several) zero singular values. Given the fact that the singular values are always non-negative, a minimization of the nuclear norm has the same effect as ℓ_1 norm in compress sensing, i.e., it encourages many singular values to be zero and thus it can be considered as a heuristic for the original rank minimization problem.

Let $\hat{Y}$ denote the partially filled in $m \times k$ matrix with the valid responses on $(i,j) \in \Omega$ positions. We are looking for Y of the same size so that the valid responses are unchanged and the nuclear norm (denoted here as $\|\cdot\|_*$) is minimal.

$$\begin{array}{ll} \underset{Y}{\text{minimize}} & \|Y\|_* \\ \text{subject to} & Y_{ij} = \hat{Y}_{ij} \quad \text{for all} \quad (i,j) \in \Omega. \end{array}$$

This is equivalent to

$$\begin{array}{ll} \underset{W_1,W_2,Y}{\text{minimize}} & \text{trace}(W_1) + \text{trace}(W_2) \\ \text{subject to} & Y_{ij} = \hat{Y}_{ij} \quad \text{for all} \quad (i,j) \in \Omega \\ & \begin{pmatrix} W_1 & Y \\ Y^{\mathrm{T}} & W_2 \end{pmatrix} \succeq 0 \end{array}$$

which is the linear semidefinite problem solved in this example, see Candes and Recht (2009) and the references therein for details.

This example has $m = 15$ respondents and $k = 6$ answers. The obtained answers are

$$\hat{Y} = \begin{pmatrix} * & * & * & * & * & 0.4 \\ 0.6 & 0.4 & 0.8 & * & * & * \\ * & * & 0.8 & * & 0.2 & * \\ 0.8 & 0.2 & * & * & * & * \\ * & 0.4 & * & 0.0 & * & 0.2 \\ 0.4 & * & * & 0.2 & * & 0.2 \\ * & 0.8 & 0.2 & 0.6 & * & * \\ * & * & 0.2 & * & * & * \\ * & 0.4 & * & 0.6 & 0.0 & * \\ * & * & 0.4 & * & * & * \\ * & * & 0.2 & 0.2 & 0.4 & 0.4 \\ * & * & * & * & 1.0 & 0.8 \\ 1.0 & * & 0.2 & * & * & 0.6 \\ * & * & * & * & * & 0.2 \\ 0.6 & * & 0.2 & 0.4 & * & * \end{pmatrix}$$

where $*$ denotes missing entries (-1.0 is used instead in the data file). The obtained matrix has rank 4 and it is shown below printed to 1-digit accuracy:

$$Y = \begin{pmatrix} 0.5 & 0.3 & 0.2 & 0.2 & 0.4 & 0.4 \\ 0.6 & 0.4 & 0.8 & 0.2 & 0.3 & 0.4 \\ 0.4 & 0.3 & 0.8 & 0.0 & 0.2 & 0.2 \\ 0.8 & 0.2 & 0.3 & 0.4 & 0.3 & 0.4 \\ 0.0 & 0.4 & 0.2 & 0.0 & 0.2 & 0.2 \\ 0.4 & 0.1 & 0.2 & 0.2 & 0.1 & 0.2 \\ 0.6 & 0.8 & 0.2 & 0.6 & 0.2 & 0.4 \\ 0.1 & 0.1 & 0.2 & 0.0 & 0.0 & 0.1 \\ 0.6 & 0.4 & 0.1 & 0.6 & 0.0 & 0.3 \\ 0.2 & 0.1 & 0.4 & 0.0 & 0.1 & 0.1 \\ 0.5 & 0.3 & 0.2 & 0.2 & 0.4 & 0.4 \\ 0.7 & 0.4 & 0.3 & 0.0 & 1.0 & 0.8 \\ 1.0 & 0.3 & 0.2 & 0.5 & 0.5 & 0.6 \\ 0.2 & 0.1 & 0.1 & 0.1 & 0.2 & 0.2 \\ 0.6 & 0.3 & 0.2 & 0.4 & 0.2 & 0.3 \end{pmatrix}.$$

The example also turns on monitor mode of E04SVF, there is a time limit introduced for the solver which is being checked at the end of every outer iteration. If the time limit is reached, the routine is stopped by setting INFORM $= 0$ within the monitor step.

See also Section 10 in E04RAF for links to further examples in the suite.

10.1 Program Text

```
    Program e04rhfe

!     E04RHF Example Program Text
!     Mark 26 Release. NAG Copyright 2016.

!     Matrix completion problem (rank minimization) solved approximately
!     by SDP via nuclear norm minimization formulated as:
!        min    trace(X1) + trace(X2)
!        s.t.   [ X1, Y; Y', X2 ] >=0
!               0 <= Y_ij <= 1

!     .. Use Statements ..
      Use nag_library, Only: e04raf, e04rff, e04rhf, e04rnf, e04rzf, e04svf,   &
                             e04zmf, f08kbf, nag_wp, x04cbf
      Use, Intrinsic                   :: iso_c_binding, Only: c_null_ptr,     &
                                          c_ptr
!     .. Implicit None Statement ..
      Implicit None
!     .. Parameters ..
      Real (Kind=nag_wp), Parameter    :: stol = 1E-5_nag_wp
      Real (Kind=nag_wp), Parameter    :: time_limit = 120.0_nag_wp
      Integer, Parameter               :: nin = 5, nout = 6
!     .. Local Scalars ..
      Type (c_ptr)                     :: h
      Integer                          :: dima, i, idblk, idx, idxobj, idxx,   &
                                          ifail, info, inform, j, lwork, m, n, &
                                          nblk, nnz, nnzasum, nnzc, nnzh,      &
                                          nnzu, nnzua, nnzuc, nvar, rank
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable  :: a(:), bl(:), bu(:), c(:), s(:),      &
                                          work(:), x(:), y(:,:)
      Real (Kind=nag_wp)               :: rdummy(1), rinfo(32), stats(32)
      Integer, Allocatable             :: blksizea(:), icola(:), idxc(:),      &
                                          irowa(:), nnza(:)
      Integer                          :: idummy(1)
      Character (1)                    :: cdummy(1)
!     .. Intrinsic Procedures ..
      Intrinsic                        :: int, max, min, sum
!     .. Executable Statements ..
      Continue

      Write (nout,*) 'E04RHF Example Program Results'
```

```
      Write (nout,*)
      Flush (nout)

!     Skip heading in data file.
      Read (nin,*)

!     Read in the problem size and allocate space for the input data.
      Read (nin,*) m, n
      Allocate (y(m,n))

!     Read in the matrix Y.
      Read (nin,*)(y(i,1:n),i=1,m)

!     Count the number of specified elements (i.e., nonnegative)
      nnz = 0
      Do i = 1, m
        Do j = 1, n
          If (y(i,j)>=0.0_nag_wp) Then
            nnz = nnz + 1
          End If
        End Do
      End Do

!     Initialize handle.
      h = c_null_ptr

!     There are as many variables as missing entries in the Y matrix
!     plus two full symmetric matrices m x m and n x n.
      nvar = m*(m+1)/2 + n*(n+1)/2 + m*n - nnz
      Allocate (x(nvar),bl(nvar),bu(nvar))

!     Initialize an empty problem handle with NVAR variables.
      ifail = 0
      Call e04raf(h,nvar,ifail)

!     Create bounds for the missing entries in Y matrix to be between 0 and 1
      bl(:) = -1E+20_nag_wp
      bu(:) = 1E+20_nag_wp
      bl(m*(m+1)/2+n*(n+1)/2+1:nvar) = 0.0_nag_wp
      bu(m*(m+1)/2+n*(n+1)/2+1:nvar) = 1.0_nag_wp
      ifail = 0
      Call e04rhf(h,nvar,bl,bu,ifail)

!     Allocate space for the objective - minimize trace of the matrix
!     constraint. There is no quadratic part in the objective.
      nnzc = m + n
      nnzh = 0
      Allocate (idxc(nnzc),c(nnzc))

!     Construct linear matrix inequality to request that
!     [ X1, Y; Y', X2] is positive semidefinite.

!     How many nonzeros do we need? As many as number of variables
!     and the number of specified elements together.
      nnzasum = m*(m+1)/2 + n*(n+1)/2 + m*n

      Allocate (nnza(nvar+1),irowa(nnzasum),icola(nnzasum),a(nnzasum))
      nnza(1) = nnz
      nnza(2:nvar+1) = 1

!     Copy Y to the upper block of A_0 with the different sign
!     (because of the sign at A_0!)
!     (upper triangle)
      idx = 1
      Do i = 1, m
        Do j = 1, n
          If (y(i,j)>=0.0_nag_wp) Then
            irowa(idx) = i
            icola(idx) = m + j
            a(idx) = -y(i,j)
            idx = idx + 1
```

```
            End If
          End Do
        End Do
!       One matrix for each variable, A_i has just one nonzero - it binds
!       x_i with its position in the matrix constraint. Set also the objective.
!       1,1 - block, X1 matrix (mxm)
        idxobj = 1
        idxx = 1
        Do i = 1, m
!         the next element is diagonal ==> part of the objective as a trace()
          idxc(idxobj) = idxx
          c(idxobj) = 1.0_nag_wp
          idxobj = idxobj + 1
          Do j = i, m
            irowa(idx) = i
            icola(idx) = j
            a(idx) = 1.0_nag_wp
            idx = idx + 1
            idxx = idxx + 1
          End Do
        End Do
!       2,2 - block, X2 matrix (nxn)
        Do i = 1, n
!         the next element is diagonal ==> part of the objective as a trace()
          idxc(idxobj) = idxx
          c(idxobj) = 1.0_nag_wp
          idxobj = idxobj + 1
          Do j = i, n
            irowa(idx) = m + i
            icola(idx) = m + j
            a(idx) = 1.0_nag_wp
            idx = idx + 1
            idxx = idxx + 1
          End Do
        End Do
!       1,2 - block, missing element in Y we are after
        Do i = 1, m
          Do j = 1, n
            If (y(i,j)<0.0_nag_wp) Then
              irowa(idx) = i
              icola(idx) = m + j
              a(idx) = 1.0_nag_wp
              idx = idx + 1
            End If
          End Do
        End Do

!       Add the sparse linear objective to the handle.
        ifail = 0
        Call e04rff(h,nnzc,idxc,c,nnzh,idummy,idummy,rdummy,ifail)

!       Just one matrix inequality of the dimension of the extended matrix.
        nblk = 1
        Allocate (blksizea(nblk))
        dima = m + n
        blksizea(:) = (/dima/)

!       Add the constraint to the problem formulation.
        idblk = 0
        ifail = 0
        Call e04rnf(h,nvar,dima,nnza,nnzasum,irowa,icola,a,nblk,blksizea,idblk,  &
          ifail)

!       Set optional arguments of the solver.
!       Completely turn off printing, allow timing and
!       turn on the monitor mode to stop every iteration.
        ifail = 0
        Call e04zmf(h,'Print File = -1',ifail)
        ifail = 0
        Call e04zmf(h,'Stats Time = Yes',ifail)
        ifail = 0
```

```
      Call e04zmf(h,'Monitor Frequency = 1',ifail)
      ifail = 0
      Call e04zmf(h,'Initial X = Automatic',ifail)
      ifail = 0
      Call e04zmf(h,'Dimacs = Check',ifail)

!     Pass the handle to the solver, we are not interested in
!     Lagrangian multipliers.
      nnzu = 0
      nnzuc = 0
      nnzua = 0
loop: Do
        ifail = -1
        Call e04svf(h,nvar,x,nnzu,rdummy,nnzuc,rdummy,nnzua,rdummy,rinfo,      &
          stats,inform,ifail)

        If (inform==1) Then
!         Monitor stop
          Write (nout,99998) int(stats(1)), rinfo(1),                          &
            sum(rinfo(2:4))/3.0_nag_wp
          Flush (nout)

!         Check time limit and possibly stop the solver.
          If (stats(8)>time_limit) Then
            inform = 0
          End If
        Else
!         Final exit, solver finished.
          Write (nout,99997) int(stats(1)), rinfo(1),                          &
            sum(rinfo(2:4))/3.0_nag_wp
          Flush (nout)
          Exit loop
        End If

      End Do loop

      If (ifail==0 .Or. ifail==50) Then
!       Successful run, fill the missing elements in the matrix Y.
        idx = m*(m+1)/2 + n*(n+1)/2 + 1
        Do i = 1, m
          Do j = 1, n
            If (y(i,j)<0.0_nag_wp) Then
              y(i,j) = x(idx)
              idx = idx + 1
            End If
          End Do
        End Do

!       Print the matrix.
        ifail = 0
        Call x04cbf('General','N',m,n,y,m,'F7.1','Completed Matrix','Integer', &
          cdummy,'Integer',cdummy,80,0,ifail)

!       Compute rank of the matrix via SVD, use the fact that the order
!       of the singular values is descending.
        lwork = 20*max(m,n)
        Allocate (s(min(m,n)),work(lwork))
        Call f08kbf('No','No',m,n,y,m,s,rdummy,1,rdummy,1,work,lwork,info)
        If (info==0) Then
lp_rank:  Do rank = 1, min(m,n)
            If (s(rank)<=stol) Then
              Exit lp_rank
            End If
          End Do lp_rank
          Write (nout,99999) 'Rank is', rank - 1
99999     Format (1X,A,I20)
        End If
      Else If (ifail==20) Then
        Write (nout,*) 'The given time limit was reached, run aborted.'
      End If
```

```
!     Destroy the handle.
      ifail = 0
      Call e04rzf(h,ifail)

99998 Format (1X,'Monitor at iteration  ',I2,': objective ',F7.2,              &
        ', avg.error ',Es9.2e2)
99997 Format (1X,'Finished at iteration ',I2,': objective ',F7.2,              &
        ', avg.error ',Es9.2e2)
    End Program e04rhfe
```

10.2 Program Data

```
E04RHF Example Program Data
15   6     :: m, n - number of respondents and questions
   -1.0   -1.0   -1.0   -1.0   -1.0    0.4
    0.6    0.4    0.8   -1.0   -1.0   -1.0
   -1.0   -1.0    0.8   -1.0    0.2   -1.0
    0.8    0.2   -1.0   -1.0   -1.0   -1.0
   -1.0    0.4   -1.0    0.0   -1.0    0.2
    0.4   -1.0   -1.0    0.2   -1.0    0.2
   -1.0    0.8    0.2    0.6   -1.0   -1.0
   -1.0   -1.0    0.2   -1.0   -1.0   -1.0
   -1.0    0.4   -1.0    0.6    0.0   -1.0
   -1.0   -1.0    0.4   -1.0   -1.0   -1.0
   -1.0   -1.0    0.2    0.2    0.4    0.4
   -1.0   -1.0   -1.0   -1.0    1.0    0.8
    1.0   -1.0    0.2   -1.0   -1.0    0.6
   -1.0   -1.0   -1.0   -1.0   -1.0    0.2
    0.6   -1.0    0.2    0.4   -1.0   -1.0  :: -1.0 for missing entries
```

10.3 Program Results

```
 E04RHF Example Program Results

 Monitor at iteration   0: objective    0.00, avg.error  3.14E+01
 Monitor at iteration   1: objective  154.74, avg.error  4.98E+01
 Monitor at iteration   2: objective   71.71, avg.error  2.15E+01
 Monitor at iteration   3: objective   36.88, avg.error  9.13E+00
 Monitor at iteration   4: objective   22.50, avg.error  3.84E+00
 Monitor at iteration   5: objective   16.47, avg.error  1.61E+00
 Monitor at iteration   6: objective   13.88, avg.error  6.87E-01
 Monitor at iteration   7: objective   12.76, avg.error  2.97E-01
 Monitor at iteration   8: objective   12.27, avg.error  1.29E-01
 Monitor at iteration   9: objective   12.06, avg.error  5.63E-02
 Monitor at iteration  10: objective   11.97, avg.error  2.50E-02
 Monitor at iteration  11: objective   11.93, avg.error  1.17E-02
 Monitor at iteration  12: objective   11.91, avg.error  5.77E-03
 Monitor at iteration  13: objective   11.91, avg.error  3.33E-03
 Monitor at iteration  14: objective   11.90, avg.error  9.11E-04
 Monitor at iteration  15: objective   11.90, avg.error  3.77E-04
 Monitor at iteration  16: objective   11.90, avg.error  1.64E-04
 Monitor at iteration  17: objective   11.90, avg.error  7.07E-05
 Monitor at iteration  18: objective   11.90, avg.error  3.05E-05
 Monitor at iteration  19: objective   11.90, avg.error  1.31E-05
 Monitor at iteration  20: objective   11.90, avg.error  5.60E-06
 Monitor at iteration  21: objective   11.90, avg.error  2.38E-06
 Monitor at iteration  22: objective   11.90, avg.error  1.01E-06
 Finished at iteration 23: objective   11.90, avg.error  4.31E-07
 Completed Matrix
          1      2      3      4      5      6
  1     0.5    0.3    0.2    0.2    0.4    0.4
  2     0.6    0.4    0.8    0.2    0.3    0.4
  3     0.4    0.3    0.8    0.0    0.2    0.2
  4     0.8    0.2    0.3    0.4    0.3    0.4
  5     0.0    0.4    0.2    0.0    0.2    0.2
  6     0.4    0.1    0.2    0.2    0.1    0.2
  7     0.6    0.8    0.2    0.6    0.2    0.4
  8     0.1    0.1    0.2    0.0    0.0    0.1
  9     0.6    0.4    0.1    0.6    0.0    0.3
 10     0.2    0.1    0.4    0.0    0.1    0.1
```

```
 11     0.5    0.3    0.2    0.2    0.4    0.4
 12     0.7    0.4    0.3    0.0    1.0    0.8
 13     1.0    0.3    0.2    0.5    0.5    0.6
 14     0.2    0.1    0.1    0.1    0.2    0.2
 15     0.6    0.3    0.2    0.4    0.2    0.3
 Rank is                  4
```