Econometric Institute Report EI-9956/A

A Bilinear Programming Solution to the Quadratic Assignment Problem

Johan F. Kaashoek[†] **and Jean H.P. Paelinck**[‡]

[†]Econometric Institute, Erasmus University, P.O.Box 1738, NL - 3000 DR - Rotterdam, The Netherlands; e-mail: kaashoek@few.eur.nl
[‡]Emeritus professor, Erasmus University Rotterdam; Oranjelaan 36, NL - 3062 BT - Rotterdam; e-mail: j.paelinck@poboxes.com

The authors like to thank Rajendra Kulkarni, George Mason University, The institute of Public Policy, Center for Regional Analysis, who performed useful programming in the beginning of the research on the present topic.

Contents

1 Introduction

The quadratic assignment problem (QAP) or maximum acyclical graph problem is well documented (see e.g. Pardanos and Wolkowicz (1994)).

One of the authors has published some material, in which it was tried, by structuring the problem additionally, to bring it as closely as possible in the neighbourhood of a binary solution (see Paelinck (1983), pp. 251-256 and 273-277); good but not optimal solutions could so be obtained (see Paelinck (1985), pp. 247-254).

The problem is taken up again here, in the same spirit but at the same time in a different vein.

2 Specification

The most compact specification of the QAP is:

$$\max_{x} \varphi \overset{\triangle}{=} \mathbf{x}'\mathbf{H}\mathbf{x} \tag{1}$$

s.t.

$$\mathbf{J}\mathbf{x} = \mathbf{i} \tag{2}$$

$$\hat{\mathbf{x}}\mathbf{x} = \mathbf{x} \tag{3}$$

in which (1) is a quadratic form to be maximised over a binary vector $\mathbf{x}$ [conditions (3)] under a set of assignment constraints [conditions (2)]. Viewed as a matrix permutation problem, it amounts to maximising the sum of its elements above the main diagonal by switching simultaneously rows and columns of a given square matrix $\mathbf{A}$; the elements of $\mathbf{x}$ are x_{ij}s, assigning row and column i to place j, conditions (2) and (3) together expressing the fact that each row and column has to occupy one and only one place. If $\mathbf{A}$ is of order n, $\mathbf{H}$ is obviously of order n^2; $\mathbf{J}$ is of order $(2n-1) \times (2n)$, one of the conditions being derivable from the others, and $\mathbf{i}$ is a unit column vector.

3 Solution

The basic idea is to substitute (2) into (1), a method that has been successfully applied in simplifying the solution to mixed integer-continuous linear programs

(Paelinck and Kulkarni (1998)). The result is a reduced system in $n^2 - 2n + 1$ variables, which results in an objective function:

$$\varphi^* = \mathbf{x_2'H^*x_2} + \mathbf{h^{*\prime}x_2} + c^* \tag{4}$$

$\mathbf{H}^*$ and $\mathbf{h}^*$ being obtained as follows. From (2) express $2n-1$ variables $\mathbf{x_1}$ in terms of the remaining $\mathbf{x_2}$-variables (this is possible, as $\mathbf{J}$ is of rank $2n-1$) and substitute the result in (1), conformably partitioned into $\mathbf{H_{11}}, \mathbf{H_{12}}, \mathbf{H_{21}}, \mathbf{H_{22}}$. There comes:

$$\mathbf{H}^* = \mathbf{J_2}'\mathbf{J_1^{-1}}'\mathbf{H_{11}J_1^{-1}J_2} - \mathbf{J_2}'\mathbf{J_1^{-1}}'\mathbf{H_{12}} - \mathbf{H_{21}J_1^{-1}J_2} + \mathbf{H_{22}} \tag{5}$$

$$\mathbf{h}^{*\prime} = -2\mathbf{i}'\mathbf{J_1^{-1}}'\mathbf{H_{11}J_1^{-1}J_2} + \mathbf{i}'\mathbf{J_1^{-1}}'\mathbf{H_{12}} + \mathbf{i}'\mathbf{J_1^{-1}}'\mathbf{H_{21}'} \tag{6}$$

where $\mathbf{J_1}$ and $\mathbf{J_2}$ result from partitioning (2) as:

$$\mathbf{J_1x_1} + \mathbf{J_2x_2} = \mathbf{i}. \tag{7}$$

In practice $\mathbf{x_1}$ has been chosen as $[x_{11}, x_{21}, \cdots, x_{n-1,1}; x_{n1}, x_{n2}, \cdots, x_{nn}]'$, which allows of writing:

$$\mathbf{J_1^{-1}} = \mathbf{I} - [\mathbf{0}'; \mathbf{i}'] \tag{8}$$

where $\mathbf{i}'$ is an $n-1$ unit row vector; the second matrix in the right hand member of (8) is a full zero matrix except that in the central row after the diagonal (zero) element, the row vector $\mathbf{i}'$ is inserted.

Another structuring element can now be introduced; indeed, if problem (1) through (3) reaches its maximum, it has a corresponding minimum, which results obviously from the complete permutation of the optimal (maximising) row-cum-column numbers : all the maximising elements will clearly then show up below the main diagonal of $\mathbf{A}$, leaving above it the minimising elements.

Applied to $\mathbf{x}$, this permutation can be obtained through a block-diagonal matrix $\mathbf{P}$, order n^2, each block being composed of a reverse identity matrix (running from the south-west to the north-east corners); $\mathbf{P}$ is obviously symmetric, and can also be shown to be auto-reverse, but this property is irrelevant here.

Now, if φ is maximum, so is:

$$\varphi^\sharp \overset{\triangle}{=} \mathbf{x}'\mathbf{Hx} - \mathbf{x}'\mathbf{PHPx} = \mathbf{x}'(\mathbf{H} - \mathbf{PHP})\mathbf{x} \tag{9}$$

and to this objective function transformation (4) through (8) is then applied, which gives:

$$\varphi^{**} = \mathbf{x_2'H^{**}x_2} + \mathbf{h^{**\prime}x_2} + c^{**}. \tag{10}$$

$\mathbf{H}^{**}$ has the following properties (see also for an example Maple output (23)):

(a) it is of order $(n-1)^2$; its relevant terms are x_{ij}, $i = 1, \cdots, n-1$, $j = 2, \cdots, n$;

(b) it is upper block-triangular;

(c) its diagonal blocks of order $(n-1)$ are zero; indeed (10) nets out the quadratic terms, which in fact is due to operation (9);

(d) each non-diagonal block of order $(n-1)$ is anti-symmetric, in the sense that all of the elements above the main diagonal have the reverse sign but the same absolute value as their counterparts below that diagonal; this property is again taken over from $\mathbf{H} - \mathbf{PHP}$;

(e) there are no non-admissible bilinear elements [conditions (2)]; one more this is due to operation (9).

What results from those properties is that a linear-bilinear function has to be optimised, the bilinear terms moreover being taken care of by a block -triangular matrix with zero diagonal blocks; it is known that the linear part can be optimized by linear programming (Murty (1976)), and the bilinear part is sequential; an efficient algorithm to solve non-linear problems should lead to binary solutions under the following conditions:

$$\mathbf{J_2 x_2} \leq \mathbf{i} \tag{11}$$

$$n - 2 \leq \mathbf{i' x_2} \tag{12}$$

the relaxed conditions:

$$\mathbf{x_2} \leq \mathbf{i} \tag{13}$$

being obviously redundant for non-negative $\mathbf{x_2}$. Be it noted that $\mathbf{J_2}$ should be given its full $2(n-1)$-order.

Preliminary grooming of large $\mathbf{A}$-matrices, first by ranking $\mathbf{A}$ according to its row-sums, and then checking for the presence of perversely dominating off-diagonal terms (Varii Auctores (1966), pp. 17-22), might speed up the solution.

4 Examples

$n = 5$, $n = 6$, $n = 7$ and $n = 10$ examples will be treated next; the solution algorithm used was the Lasdon-Waren Generalized Reduced Gradient ($GRG2$) non-linear optimization code (Lasdon, Waren, Jain and Ratner (1978)).

4.1 Random 5×5 matrix

Matrix $\mathbf{A}$:

$$\mathbf{A} = \begin{pmatrix} 0 & 20 & 22 & 13 & 1 \\ 13 & 0 & 18 & 21 & 9 \\ 5 & 13 & 0 & 10 & 24 \\ 10 & 16 & 11 & 0 & 24 \\ 11 & 23 & 16 & 9 & 0 \end{pmatrix} \tag{14}$$

was constructed by drawing at random between 0 and 25 20 integers using Maple's procedure *rand(25)()* while putting the diagonal elements to zero (Maple (1998)); see also the Maple program in Appendix section (A.1).

From an initialising vector $\mathbf{0}$, as will be the case further down unless otherwise specified, the maximising vector $[x_{41}, x_{52}, x_{13}, x_{24}, x_{35}]'$ with $\phi^{**} = 53$ was obtained under conditions (12) and (13) from the following data:

$$\begin{aligned} \mathbf{h}^{**\prime} &= [-46, -26, -6, 14; -30, -2, 26, 54; 40, 24, 8, -8; 38, 8, -22, -52] \\ \mathbf{u}'\mathbf{H}^{**} &= [3, 35, 28; 27, 34; 6] \end{aligned} \tag{15}$$

where $\mathbf{u}'\mathbf{H}^{**}$ denotes the row-vector of values appearing in the upper triangular part of the off-diagonal blocks of $\mathbf{H}^{**}$ in the row order of those blocks. Computing time: 1s on a desktop using a standard Microsoft Excel routine as will be again the case further down.

4.2 Random 6×6 matrix

The following matrix was explored:

$$\mathbf{A} = \begin{pmatrix} 0 & 20 & 22 & 13 & 1 & 13 \\ 10 & 0 & 21 & 9 & 5 & 13 \\ 7 & 10 & 0 & 10 & 16 & 11 \\ 7 & 24 & 11 & 0 & 16 & 9 \\ 23 & 6 & 24 & 16 & 0 & 17 \\ 16 & 18 & 0 & 11 & 13 & 0 \end{pmatrix} \tag{16}$$

with optimal ranking $[x_{51}, x_{12}, x_{43}, x_{24}, x_{35}, x_{66}]$ and $\phi^{**} = 94$, which optimal ranking was found by sharpening the left-hand side of constraint (12) to $\mathbf{i}'\mathbf{x_2} = 4$, and from the following data:

$$\begin{aligned} \mathbf{h}^{**\prime} &= [-23, -17, -11, -5, 1; -5, 5, 15, 25, 35; 68, 41, 19, -3, -25; \\ &\quad -21, -17, -13, -9, -5; -24, -32, -40, -48, -56] \\ \mathbf{u}'\mathbf{H}^{**} &= [8, 29, 7, -15; 27, -12, 8; , -14, -15; 6] \end{aligned} \tag{17}$$

Computing time: 2s, to be compared to 69s CPU for complete (see Appendix (A.1)).

It should be noted that before constraint (12) was sharpened, a "good" solution with $\phi^{**} = 67$ was obtained; from this experience it could be advised to sharpen the constraint to check for non-optimality from inequality (12).

4.3 Random 7×7 matrix

The generated matrix was the following one:

$$\mathbf{A} = \begin{pmatrix} 0 & 61 & 7 & 49 & 86 & 98 & 66 \\ 9 & 0 & 81 & 74 & 66 & 73 & 42 \\ 91 & 93 & 0 & 11 & 38 & 13 & 20 \\ 44 & 65 & 91 & 0 & 74 & 9 & 60 \\ 82 & 92 & 13 & 77 & 0 & 35 & 61 \\ 48 & 3 & 23 & 95 & 73 & 0 & 37 \\ 57 & 99 & 94 & 28 & 15 & 55 & 0 \end{pmatrix} \tag{18}$$

From the following data:

$$\begin{aligned} \mathbf{h}^{**\prime} &= [71, 53, 35, 17, -1, -19; -155, -41, 73, 187, 301, 415; \\ &\quad -265, -117, 31, 179, 327, 475; 213, 149, 85, 21 - 43; \\ &\quad 284, 192, 100, 8, -43, -176; -24, 12, 48, 84, 120, 156] \\ \mathbf{u}'\mathbf{H}^{**} &= [-14, -167, 28, 41, 23; -29, 98, 77, 109; \\ &\quad 26, 145, 46; 11, -136; -102] \end{aligned} \tag{19}$$

and with sharpened left-hand constraint (12), the optimal *minimising* ranking was correctly computed as

$$[x_{61}, x_{22}, x_{13}, x_{34}, x_{75}, x_{46}, x_{57}]$$

with $\phi^{**} = -418$ in 5s computing time; the maximising attempt produced a fractional solution. The complete enumeration required 1981s CPU time.

4.4 Random 10×10 matrix

The matrix was generated as follows:

$$\mathbf{A} = \begin{pmatrix} 0 & 44 & 65 & 91 & 95 & 74 & 9 & 60 & 82 & 92 \\ 13 & 0 & 49 & 35 & 61 & 48 & 3 & 23 & 95 & 73 \\ 89 & 37 & 0 & 99 & 94 & 28 & 15 & 55 & 7 & 51 \\ 62 & 97 & 88 & 0 & 97 & 98 & 27 & 27 & 74 & 25 \\ 7 & 82 & 29 & 52 & 0 & 85 & 45 & 98 & 38 & 76 \\ 75 & 74 & 23 & 0 & 19 & 0 & 49 & 47 & 13 & 65 \\ 44 & 11 & 36 & 59 & 41 & 56 & 0 & 23 & 24 & 93 \\ 19 & 26 & 50 & 6 & 70 & 35 & 5 & 0 & 36 & 66 \\ 62 & 87 & 11 & 4 & 75 & 56 & 47 & 85 & 0 & 20 \\ 22 & 80 & 38 & 95 & 84 & 99 & 10 & 79 & 44 & 0 \end{pmatrix} \tag{20}$$

and the derived vector were:

$$\begin{aligned} \mathbf{h}^{**\prime} = [&331, 191, 51, -89, -229, -369, -509, -649, -789; 72, 86, 100, \\ &114, 128, 142, 156, 170, 184; 8, -18, -44, -70, -96, -122, \\ &-148, -174, -200; -724, -584, -444, -304, -164, -24, 116, \\ &256, 396; 50, 66, 82, 98, 114, 130, 146, 162, 178; -68, 0, 68, \\ &136, 204, 272, 340, 408, 476; 477, 311, 145, -21, -187, -353, \\ &-519, -685, -851; 70, 96, 122, 148, 174, 200, 226, 252, 278; \\ &-236, -188, -140, -92, -44, 4, 52, 100, 148] \\ \mathbf{u}'\mathbf{H}^{**} = [&-46, -81, -111, 10, -105, -22, -42, -74; 32, -125, -22, \\ &-53, 82, -9, -9; -72, 44, -42, 49, -21, -41; 107, 134, 121, \\ &78, 116; 40, 95, 23, -53; 110, 33, -33; -78, -130; -60] \end{aligned} \tag{21}$$

For matrix (20) as ranked there $\phi^{**} = 291$; the matrix was then groomed according to section 3 *in fine*, producing $\phi^{**} = 903$ and that ranking was taken as starting point for the computations. With a sharpened constraint (12), two extra constraints blocking the places of rows and columns 3 (ranking first after the grooming) and 10 (from the problem specification), and the constraint $\phi^{**} \geq 903$, the value $\phi^{**} = 1109$ was obtained; then without the two extra constraints, but constraining $\phi^{**} \geq 1109$, the value $\phi^{**} = 1291$ was generated. No further improvement could be obtained.

As the overall optimum was not known at that moment, a control via the minimising ranking was effectuated, starting from the reverse ranking of the maximising one, and perturbating it by switching at two rows and columns; the program returned to the value -1291.
The resulting - presumingly - maximising ranking was

$$[x_{71}, x_{32}, x_{13}, x_{10,4}, x_{45}, x_{96}, x_{57}, x_{68}, x_{89}, x_{2,10}],$$

this was confirmed by complete enumeration which took 35m CPU computing time using a compiled pascal version of the Maple code in Appendix (A.1).

Subsequently some other specifications have been tried out. For matrix (20), with sharpened constraint (12), and fixing x_{21}, the correct minimum was obtained in 22s CPU time. The idea was generated by the first experiences, showing that a few additional constraints were instrumental in finding the absolute optimum; possibly putting alternatively each row-column pair of the $\mathbf{A}$-matrix in the first position, i.e. running n successive programs, could be the lesson drawn from this, in fact a polynomial extension of the above exercises.

For the groomed version of (20), the absolute minimum was obtained with fixed x_{21} and $x_{7,10}$ in 15s CPU time, corroborating the hypothesis presented at the end of section 3; the obvious extension would be to run n^2 successive programs, still a polynomial case.

5 Conclusions

Once more it has been shown that using the full structural information on a complex problem leads to extremely simplifying its solution (other examples in Paelinck (1996), Paelinck (1998) and in Paelinck and Paelinck (1998)).

The algorithm proposed can easily be programmed; the appendix in section (A.2) presenting the program for the transformations (9) and (10). implements it.

References

Lasdon, L.S., Ware, A.D., Jain, A. and Ratner, M., 1978, Design and testing of a generalized reduced gradient code for nonlinear programming, *ACM Transactions on Mathematical Software*, Vol.4, No1, pp/34-49.

Maple V Release 5, 1998, Waterloo Maple Software, Waterloo, Ontario.

Murty, K.G., 1976, *Linear and Combinatorial Programming*, Wiley, N.Y.et al. loc.

Paelinck, J.H.P. (with the assistance of J.-P. Ancot and J.H. Kuiper), 1983, *Formal Spatial Economic Analysis*, Gower, Aldershot.

Paelinck, J.H.P.,(avec l'assistance de J.-P. Ancot, H. Gravesteijn, J.H. Kuiper et Th. ten Raa), 1985,*Eléments d'Analyse Economique Spatiale*, Edition Régionales Européennes, Diffusion Anthropos, Paris.

Paelinck, J.H.P., 1996, On Solving the Maximal Flow Capturing Problem by Linear Programming, in *Four Studies in Theoretical Spatial Economics*, Section 3, University of Munich, Center for Economic Studies, Working Papers Series, No 100.

Paelinck, J.H.P., 1998, Controlling Complexity in Spatial Modelling, submitted for publication.

Paelinck, J.H.P. and Kulkarni, R.E.,1998, Location-Allocation Aspects of Tinbergen-Bos Systems, accepted for publication in *The Annals of Regional Science.*

Paelinck, H.C. and Paelinck, J.H.P., 1998, Queuing Problems and Optimal Design of Conntainer Ports, *Tijdschrift vervoerswetenschap, 98/3*, pp. 307-316.

Pardanos, P.M. and Wolkowicz, H. (eds), 1994, *Quadratic Assignment and Related Problems*, American Mathematical Society, DIMACS Series in Discrete Mathematics and Theoretical Computer Science.

Varii Auctores, 1966, *Etude comparée des tableaux d'entrés et de sorties des Communnautés Européennes*, Centre de Recherches Economiques et Sociales, Département d'économétrie, Namur.

A Computer program

The computer programs are written in Maple (1998). Apart from the dimension of the problem, given by the Maple variable nx, and the specific problem matrix A, the programs are written in general terms and variables. Output is suppressed as much as possible.

A.1 Complete enumeration of all row-column permutations

In this section a Maple program is given for calculating the optimal value of sum of upper diagonal minus lower diagonal elements over all row-column permutations of given matrix **A** by just enumerating.

```
> restart:
> with(combinat, permute):
> with(linalg):
```

Procedure *Perm_matrix*(*vects*) outputs for given permutation vector *vects*, a row-column permutation matrix.

```
> perm_matrix := proc (vects) local k, n, m;
> n := vectdim(vects); m := matrix(n,n,0);
> for k to n do m[vects[k],k] := 1 od;
> evalm(m);
> end:
```

The dimension of the problem is defined by *nx*:

```
> nx := 5:
```

Defining specific problem matrix **A**:

```
> AA0:=matrix(nx,nx,0):
```

An example of a random matrix (not used here):

```
> # random matrix
> #AA0:=matrix(nx,nx,(i,j)->rand(100)()):
> #for k from 1 to nx do AA0[k,k]:=0:od:evalm(AA0);
```

Here we use the following 5×5 matrix:

```
> AA5:=matrix([[0, 20, 22, 13, 1], [13, 0, 18,21, 9],
> [5, 13, 0, 10, 24], [10, 16, 11, 0, 24], [11, 23, 16, 9,0]]);
```

$$AA5 := \begin{bmatrix} 0 & 20 & 22 & 13 & 1 \\ 13 & 0 & 18 & 21 & 9 \\ 5 & 13 & 0 & 10 & 24 \\ 10 & 16 & 11 & 0 & 24 \\ 11 & 23 & 16 & 9 & 0 \end{bmatrix}$$

```
> AA0:=evalm(AA5):
```

Initialisation:

```
> NumberofPerm:=nx!:
```

All permutations are given by the Maple procedure *permute*:

```
> permvec:=permute(nx):
> StartTime := time():
> maxoverall := -infinity:
```

Start of loop:

```
> for kk from 1 to NumberofPerm do
```

Given permutation *permvec[kk]*, the row-column permutation $\mathbf{P_n}$ is calculated:

```
> P_n:=evalm(perm_matrix(permvec[kk]));
```

Row-column permutation applied to matrix $\mathbf{A_{00}}$:

```
> A:=evalm(transpose(P_n)&*AA0&*P_n);
```

Calculation of sum of upper-diagonal minus lower-diagonal elements:

```
> sumd:=0:
> for k from 1 to nx-1 do for m from (k+1) to
> nx do sumd:=sumd+A[k,m]-A[m,k]:od:od:
```

Check for new maximal value and keep new optimal permutation in *maxperm*:

```
>  if sumd>=maxoverall then maxperm :=
>  permvec[kk]:maxoverall:=sumd:fi
>  od:
```

End of loop.

```
>  EndTime:=time()-StartTime:print(`Time used `= EndTime);
```

$$\mathit{Time\ used}\ = 7.110$$

The optimal permutation is given as *maxperm*:

```
>  print(`Optimal permutation` = evalm(maxperm));
>  print(`Maximal value` = maxoverall);
>  P_n:=evalm(perm_matrix(maxperm)):
```

$$\mathit{Optimal\,permutation} = [4,\ 5,\ 1,\ 2,\ 3]$$

$$\mathit{Maximal\ value} = 53$$

```
>  print(`Optimal permutation of A ` =
>  evalm(transpose(P_n)&*AA0&*P_n));
```

$$\mathit{Optimal\ permutation\ of\ A}\ = \begin{bmatrix} 0 & 24 & 10 & 16 & 11 \\ 9 & 0 & 11 & 23 & 16 \\ 13 & 1 & 0 & 20 & 22 \\ 21 & 9 & 13 & 0 & 18 \\ 10 & 24 & 5 & 13 & 0 \end{bmatrix}$$

A.2 Linear- and Bilinear coefficients

In this program formula (10) is calculated for given matrix **A**.

```
>  restart:
>  with(linalg):
```

The dimension of the problem is defined by *nx*:

```
>  nx := 3:
```

Some non problem specific definitions, depending only on the dimension of the problem:

```
>  nx1:=2*nx-1:nn:=nx*nx:nx2:=nn-nx1:
>  ti0:=vector(nx1,1):
```

The matrix **J**:

```
>  J := matrix(2*nx-1,nn,0):
>  for m from 1 to nx do for k from 1 to nx do
>  J[m,k+nx*(m-1)]:=1: od:od:
>  for m from 1 to nx-1 do for k from 1 to nx do
>  J[m+nx,2+(k-1)*nx+(m-1)]:=1: od: od:
```

Permutation lists (see definition of $\mathbf{x_1}$ and $\mathbf{x_2}$ above equation (5)):

```
>  listAll:=[seq(k,k=1..nx*nx)]:
>  listExcl:=[seq(1+(k-1)*nx,k=1..nx)
>  ,seq(k+(nx-1)*nx,k=2..nx)]:
>  listIncl:=[op({op(listAll)} minus {op(listExcl)})]:
>  listIncl:=sort(listIncl):listExcl:=sort(listExcl):
>  listTot:=[op(listExcl),op(listIncl)]:
```

The matrix $\mathbf{J_1}$ (see eq. (7) and (8)):

```
>  J1:=matrix(nx1,nx1,0):
>  for k from 1 to nx1 do for l from 1 to nx1 do
>  J1[k,l]:=J[k,listExcl[l]]: od: od:
>  J1inv:=evalm(inverse(J1)):
```

And the matrix $\mathbf{J_2}$:

```
>  J2:=matrix(nx1,nx2):
>  for m from 1 to nx1 do for k from 1 to nx2 do
>  J2[m,k]:=J[m,listIncl[k]] od:od:evalm(J2):
```

Make a vector $\mathbf{xx}$ with elements in proper notation according to the text:

```
>  x:=matrix(nx,nx):xx:=matrix(1,nn,(i,j)->x[1+floor((j-1)/nx),
>  1+((j-1) mod nx)]):
```

Make permutation $\mathbf{xp}$ of vector $\mathbf{xx}$ according to $\mathbf{x} = [\mathbf{x_1}|\mathbf{x_2}]$:

```
>  xp:=matrix(1,nn):for k from 1 to nn do
>  xp[1,k]:=xx[1,listTot[k]]:od:
```

Make subvectors $\mathbf{x_1}$ and $\mathbf{x_2}$:

```
>  x1:=matrix(1,nx1):for k from 1 to nx1 do
>  x1[1,k]:=xx[1,listExcl[k]]:od:
>  x2 := matrix(1,nx2):for k from 1 to nx2 do
>  x2[1,k]:=xx[1,listIncl[k]]:od:evalm(x2);
```

$$\begin{bmatrix} x_{1,2} & x_{1,3} & x_{2,2} & x_{2,3} \end{bmatrix}$$

Construction of permutation matrix $\mathbf{P}$ to get the matrix $\mathbf{H} - \mathbf{PHP}$ with $\mathbf{P}' = \mathbf{P}$:

```
>  II:=matrix(nx,nx,0):for k from 1 to nx do
>  II[nx-k+1,k]:=1: od: IO:=matrix(nx,nx,0):
>  Dum:=matrix(nx,nx):IK:=array(1..nx):
>  for k from 1 to nx do if (k=1) then IK[k] :=
>  II else IK[k] := IO fi od:
```

```
> for k from 1 to nx do for m from 2 to nx do
> if (k=m) then Dum := II else Dum:=IO fi:
> IK[k]:=augment(IK[k],Dum): od: od:
> P:=IK[1]:for k from 2 to nx do
> P:=stackmatrix(P,IK[k]) od:
```

Defining specific problem matrix **A**:

```
> AA0:=matrix(nx,nx,(i,j)->rand(100)()):for k
> from 1 to nx do AA0[k,k]:=0:od:evalm(AA0);
```

$$\begin{bmatrix} 0 & 70 & 97 \\ 63 & 0 & 38 \\ 85 & 68 & 0 \end{bmatrix}$$

The matrix **A**:

```
> A:=evalm(AA0):
```

Make matrix **H** based on matrix **A**:

```
> H:=matrix(nn,nn,0):
> for t from 1 to nx-1 do for k from 2+(t-1) to
> nx do for l from 1 to nx do for m from 1 to nx do
> H[1+(t-1)+(m-1)*nx,k+(l-1)*nx ]:=A[m,l]:od:od:od:od:
```

Make the matrix $H0 = \mathbf{H} - \mathbf{PHP}$, with **P** permutation matrix defined above:

```
> H0:=evalm(H - P&* H&* P):
```

The matrix $\mathbf{H} - \mathbf{PHP}$ has a special form:

```
> ‘H - PHP‘=evalm(H0);
```

$$H - PHP = \begin{bmatrix} 0 & 0 & 0 & 0 & 70 & 70 & 0 & 97 & 97 \\ 0 & 0 & 0 & -70 & 0 & 70 & -97 & 0 & 97 \\ 0 & 0 & 0 & -70 & -70 & 0 & -97 & -97 & 0 \\ 0 & 63 & 63 & 0 & 0 & 0 & 0 & 38 & 38 \\ -63 & 0 & 63 & 0 & 0 & 0 & -38 & 0 & 38 \\ -63 & -63 & 0 & 0 & 0 & 0 & -38 & -38 & 0 \\ 0 & 85 & 85 & 0 & 68 & 68 & 0 & 0 & 0 \\ -85 & 0 & 85 & -68 & 0 & 68 & 0 & 0 & 0 \\ -85 & -85 & 0 & -68 & -68 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad (22)$$

Make a matrix Hp which is reordening of $H0$ according to $\mathbf{x} = [\mathbf{x_1}|\mathbf{x_2}]$:

```
> Hp:= matrix(nn,nn):
> for k from 1 to nn do for l from 1 to nn do
> Hp[k,l]:=H0[listTot[k],listTot[l]]:od:od:
```

Partitioning of matrix Hp; see text above equation (5):

```
> H11:=matrix(nx1,nx1):
> for k from 1 to nx1 do for l from 1 to nx1 do
> H11[k,l] := Hp[k,l]: od od:
```

```
> H12:=matrix(nx1,nx2):
> for m from 1 to nx2 do for k from 1 to nx1 do
> H12[k,m]:=Hp[k,nx1+m]: od:od:

> H22:=matrix(nx2,nx2):for m from 1 to nx2 do
> for k from 1 to nx2 do H22[m,k]:= Hp[nx1+m,nx1+k]:od:od:

> H21:=matrix(nx2,nx1): for m from 1 to nx2 do
> for k from 1 to nx1 do H21[m,k]:=Hp[nx1+m,k]:od:od:
```

Writing the problem in a quadratic, linear and constant part; see equations (4) and (10):
First quadratic part:

```
> Hq:=evalm(transpose(J2)&*transpose(J1inv)&*H11&*J1inv&*J2
> -transpose(J2)&*transpose(J1inv)&*H12-H21&*J1inv&*J2+ H22):
```

Already the matrix H_q will give the correct bilinear form $\mathbf{x_2'}\mathbf{H}^{**}\mathbf{x_2}$. However, the matrix referred in the text as $\mathbf{H}^{**}$, see equation (10), is given below as the matrix H_{00}:

```
> H_00:=matrix((nx-1)^2,(nx-1)^2,0):
> for k from 1 to (nx-1)^2 do for l from k to (nx-1)^2 do
> H_00[k,l]:=Hq[k,l]+Hq[l,k]:od:od:
```

So we get the very special form of $\mathbf{H}^{**}$:

$$H^{**} = \begin{bmatrix} 0 & 0 & 0 & -35 \\ 0 & 0 & 35 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{23}$$

The linear part ($\mathbf{h}^{**\prime}$ in equation (10)):

```
> h0:=evalm(ti0&*transpose(J1inv)&*H11&*J1inv&*J2):
> h1:=evalm(ti0&*transpose(J1inv)&*transpose(H11)&*J1inv&*J2):
> h2:=evalm(ti0&*transpose(J1inv)&*H12):
> h3:=evalm(ti0&*transpose(J1inv)&*transpose(H21)):
> Linpart:=evalm(-1*h0-1*h1+h2+h3);
```

$$\mathit{Linpart} := [11,\ -13,\ 25,\ 85]$$

Now writing object function as sum of quadratic, linear and constant terms. First quadratic part (based on H_q one will get precisely the same quadratic form)::

```
> qap:=simplify(evalm(x2&*H00&*transpose(x2))[1,1]);
> lap:=evalm(Linpart&*transpose(x2))[1];
> cap:=evalm(ti0&*transpose(J1inv)&*H11&*J1inv&*ti0);
```

So ϕ^{**} as in (10), is the sum of the next three terms:

$$\begin{aligned}\mathbf{x_2'H^{**}x_2} &= -35\,x_{1,2}\,x_{2,3} + 35\,x_{1,3}\,x_{2,2}\\ \mathbf{h}^{**\prime} &= 11\,x_{1,2} - 13\,x_{1,3} + 25\,x_{2,2} + 85\,x_{2,3}\\ c^{**} &= -36\end{aligned}$$

As a conclusion, the relevant elements are collected in $\mathbf{h}^{**\prime}$ and $\mathbf{u'H^{**}}$:

```
> CC:=matrix(nx-1,nx-1,0):
> for k from 1 to (nx-2) do
> k0:=1+(k-1)*(nx-1):dx:=x2[1,k0]:d1:=coeff(qap,dx):k1:=k0+nx;CC[k+1,1]:
> =dx:for l from 1 to (nx-2) do
> dxx:=x2[1,k1+(l-k)*(nx-1)]:d2:=coeff(d1,dxx):CC[1,l+1]:=dxx;CC[k+1,l+1
> ]:=d2;od:od:
> `Coefficients Matrix Bilinear`=evalm(CC);
```

$$\mathit{Coefficients\ Matrix\ Bilinear} = \begin{bmatrix} 0 & x_{2,3} \\ x_{1,2} & -35 \end{bmatrix}$$

```
> uH00:=[]:for k from 2 to (nx-1) do for l from
> k to (nx-1) do uH00:=[op(uH00),CC[k,l]]:od:od:
> `h**'`=evalm(Linpart);
```

$$h**' = [11,\ -13,\ 25,\ 85]$$

```
> `u'H**` = uH00;
```

$$u'H** = [-35]$$