**An Algorithm for Solving the P-Matrix Problem**

Rohn, Jiří
2012

**Institute of Computer Science**
**Academy of Sciences of the Czech Republic**

# An Algorithm for Solving the $P$-Matrix Problem

Jiří Rohn

Technical report No. V-1150

06.01.2012

Pod Vodárenskou věží 2, 182 07 Prague 8, phone: +420 266 051 111, fax: +420 286 585 789,
e-mail:rohn@cs.cas.cz

**Institute of Computer Science**
**Academy of Sciences of the Czech Republic**

# An Algorithm for Solving the $P$-Matrix Problem

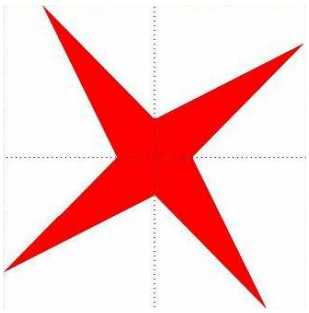
Jiří Rohn[1]

Technical report No. V-1150

06.01.2012

Abstract:

Described is a not-a-priori-exponential algorithm which in a finite number of steps checks or disproves $P$-property of a square matrix $A$, and in the latter case also finds a nonpositive principal minor of $A$.[2]



Keywords:
$P$-matrix, interval matrix, regularity, algorithm.

[1]This work was supported by the Institutional Research Plan AV0Z10300504.

[2]Above: logo of interval computations and related areas (depiction of the solution set of the system $[2,4]x_1 + [-2,1]x_2 = [-2,2]$, $[-1,2]x_1 + [2,4]x_2 = [-2,2]$ (Barth and Nuding [1])).

# 1 Introduction

A square matrix is said to be a $P$-matrix if all its principal minors are positive; the resulting problem of checking whether a given matrix is a $P$-matrix is called the *$P$-matrix problem*. As proved by Coxson [3], this problem is co-NP-complete. Recently, S. M. Rump [12] established a relationship between the $P$-property and regularity of interval matrices and used the algorithm by Jansson and Rohn [6] for checking the latter property. In this report we propose another algorithm for checking regularity, based on a recently published algorithm for computing the hull of the solution set of interval linear equations [11]. We also add a procedure which enables us to find a nonpositive principal minor in case the $P$-matrix property was disproved by the previous part of the algorithm. The algorithm **pmat**, together with its three subalgorithms, is described in the last Section 5. Its finite termination is proved in Theorem 3; Theorems 1 and 2 are auxiliary.

# 2 Auxiliary results

In this section we present two auxiliary results. The first one, due to Rump [12], describes the basic connection between $P$-property of point matrices and regularity of interval matrices; $I$ is the identity matrix.

**Theorem 1.** *Let both $A-I$ and $A+I$ be nonsingular. Then $A$ is a $P$-matrix if and only if the interval matrix*

$$[(A-I)^{-1}(A+I)-I,(A-I)^{-1}(A+I)+I] \tag{2.1}$$

*is regular.*

The second auxiliary result, which comes from [10], establishes a relationship to be used later for finding a nonpositive principal minor in case the matrix $A$ is found not to be a $P$-matrix.

**Theorem 2.** *Let $A-I$ be nonsingular. Then for each $y \in Y_n$ we have*

$$\det((A-I)^{-1}(A+I)-\operatorname{diag}(y)) = \frac{2^n \det(A[J(y)])}{\det(A-I)}, \tag{2.2}$$

*where*

$$J(y) = \{\, j \mid y_j = -1 \,\}.$$

# 3 Description

In order that the algorithm, whose description stretches over several pages, could be presented as a whole and not intertwined with the text, it is given in the last Section 5.

**Theorem 3.** *For each square matrix $A$ the algorithm* **pmat** *(Fig. 5.1) terminates in a finite (but not a priori exponential) number of steps with one of the outputs described in lines (03)-(05).*

*Proof.* Finite and not-a-priori-exponential termination of the algorithm follows from the analogous properties of **intervalhull** established in [11]. If some of the matrices $A-I$, $A+I$ is singular, then Theorem 1 does not apply and the algorithm terminates in line (10) with output described in line (05). Next we have three returns (in lines (08), (13) and (24)) with default output from line (06) described in line (03). The first case is obvious since a symmetric positive definite matrix is a $P$-matrix [4]. In the remaining two cases the $P$-property of $A$ follows from regularity of the interval matrix $\mathbf{B} = [(A-I)^{-1}(A+I)-I, (A-I)^{-1}(A+I)+I]$ (Theorem 1): in line (13) regularity is established from Beeck's sufficient regularity condition [2], and in line (24) regularity follows from the sole fact that the interval hull $\mathbf{x}$ of the solution set of the system of interval linear equations $\mathbf{B}x = [b, b]$ has been computed since this implies boundedness of the solution set which precludes singularity of $\mathbf{B}$ (Jansson [5]). Hence we are left with proving that if the algorithm is brought to conclusion (i.e., it terminates in line (35)), then $\det(A[J]) \leq 0$ which shows that $A$ is not a $P$-matrix (output description in line (04)).

To this end, define a function

$$f(t) = \det(A - I)\det(C - \mathrm{diag}(t)), \qquad t \in \mathbb{R}^n.$$

We shall later essentially use the fact that $f$ is *linear* in each $t_i$ (because the variable $t_i$ appears in the matrix $C - \mathrm{diag}(t)$ only once, namely in the $ii$th position). First we show that the vector $y$ computed in lines (26)-(28) satisfies

$$f(y) = 0.$$

Indeed, since the algorithm did not return in line (24), it must have been $\mathbf{x} = [\,]$, hence the algorithm **intervalhull** in line (23) must have constructed a singular matrix $S \in \mathbf{B}$ [11], so that a nonzero null vector $x$ of $S$ (line (25)) can be found. Since $S \in \mathbf{B} = [C-I, C+I]$ and $Sx = 0$, we have

$$|Cx| = |(C - S)x| \leq |C - S||x| \leq I|x| = |x|.$$

In particular, for each $i$, $x_i = 0$ implies $(Cx)_i = 0$. Thus the vector $y$ constructed in lines (26)-(28) satisfies $|y_i| \leq 1$ and $(Cx)_i = y_i x_i$ for each $i$, hence $Cx = \mathrm{diag}(y)x$, which gives that $(C - \mathrm{diag}(y))x = 0$, where $x \neq 0$, so that $\det(C - \mathrm{diag}(y)) = 0$, implying $f(y) = 0$.

Finally we prove by induction on $i = 0, 1, \ldots, n$ that the vector $y$ obtained after completing line (33) satisfies

$$y_j = \pm 1 \qquad (j = 1, \ldots, i) \tag{3.1}$$

and

$$f(y) \leq 0. \tag{3.2}$$

This is obviously so for $i = 0$. Thus assume that the induction hypothesis holds for some $i - 1 \geq 0$. At that moment,

$$f(y_1, \ldots, y_{i-1}, y_i, \ldots, y_n) \leq 0$$

for some $y_i \in [-1, 1]$. If $y_i = -1$ or $y_i = 1$, then we are done (line (30)). Thus assume that $y_i \in (-1, 1)$. If

$$f(y_1, \ldots, y_{i-1}, 1, \ldots, y_n) \leq 0,$$

then $y_i$ is set to 1 and (3.1), (3.2) are satisfied. If

$$f(y_1, \ldots, y_{i-1}, 1, \ldots, y_n) > 0,$$

then the function of one variable $t_i$

$$f(y_1, \ldots, y_{i-1}, t_i, \ldots, y_n)$$

is linear (as established above), is positive at $t_i = 1$ and nonpositive at $t_i = y_i \in (-1, 1)$, hence it is increasing in $[-1, 1]$, which means that it is negative at $-1$. In this case $y_i$ is set to $-1$ (line (32)) and the induction hypothesis (3.1), (3.2) is proved.

In this way, we obtain that the vector $y$ constructed after completing the **for**-loop in lines (29)-(34) is a $\pm 1$-vector satisfying

$$f(y) = \det(A - I) \det(C - \mathrm{diag}(y)) \le 0.$$

Now from Theorem 2 we have

$$\det(A[J]) = \frac{1}{2^n} \det(A - I) \det(C - \mathrm{diag}(y)) \le 0, \tag{3.3}$$

where

$$J = \{\, j \mid y_j = -1 \,\}$$

(see line (35)), which shows that the principal minor $\det(A[J])$ is nonpositive and $A$ is not a $P$-matrix (output description in line (04)).

This completes the proof. □

As the reader may have noticed, we did not mention at all the lines (16) to (22) in the proof. These statements coming from [6] form only a heuristic aimed at diminishing the number of orthants intersected by the solution set of $\mathbf{B}x = [b, b]$ and may be deleted without any effect on the algorithm, except possibly its speed.

## 4 Examples

We give three examples here. In the first one the $P$-property of a $6 \times 6$ matrix is checked.

```
A =
    9.1385   -1.6859   -2.9462   -0.3107   -0.8035   -2.2418
   -1.1156    9.7549   -1.0233   -0.7262   -0.5831   -0.4063
   -3.2339   -2.6475    8.5355    0.2031   -2.1402   -0.6397
    1.5433    1.3389    1.7474   11.2676    0.8410    1.8611
    1.2911    1.4137    1.4649   -0.2478   11.4692    1.5882
    2.5236    2.7236    2.2318    1.2000    1.6646   11.8223
>> tic, [pm,J]=pmat(A), toc
pm =
     1
J =
     []
Elapsed time is 0.015724 seconds.
```

In the second example the $P$-property is disproved and a negative principal minor is found.

```
A =
    6.8514    8.2212    6.6785    0.6503    7.7268    1.9566
    5.1015    3.1775    0.1363    4.7659    1.0618    7.8714
    7.1396    5.8770    5.6158    9.8371    0.0107    6.1856
    5.1521    1.3020    4.5456    9.2235    5.4176    0.1552
    6.0587    2.5435    9.0495    5.6120    0.0686    8.9085
    9.6670    8.0303    2.8216    6.5232    4.5134    7.6170
>> tic, [pm,J]=pmat(A), minor=det(A(J,J)), toc
pm =
     0
J =
     1     3
minor =
   -9.2055
Elapsed time is 0.140764 seconds.
```

For the purpose of the third example we first describe a MATLAB function for generating random $P$-matrices, coming from [9].

```
function A=randpmat(n)
% Generates a random nxn matrix A.
C=2*rand(n,n)-1; Ci=inv(C);
D=rand(n,n);
alpha=0.95/max(abs(eig(abs(Ci)*D)));
A=inv(C-alpha*D)*(C+alpha*D);
```

Finally we use this function for creating a random $500 \times 500$ $P$-matrix. The computation was performed on a slow netbook.

```
>> tic,n=500;rand('state',1),A=randpmat(n);[pm,J]=pmat(A),toc
pm =
     1
J =
     []
Elapsed time is 23.749405 seconds.
```

## 5 The algorithm

The main algorithm **pmat** is described here together with three its subalgorithms **intervalhull**, **qzmatrix**, and **absvaleqn**. The structure is as follows:
**pmat** calls **intervalhull**,
**intervalhull** calls **qzmatrix**,
**qzmatrix** calls **absvaleqn**.
The algorithms **intervalhull** and **qzmatrix** are described in [11], **absvaleqn** in [7], [8].

```
(01)  function [pm, J] = pmat (A)
(02)  % Checks (not-)P-property of A.
(03)  % pm = 1: A is a P-matrix, J = [];
(04)  % pm = 0: A is not a P-matrix, det(A[J]) ≤ 0;
(05)  % pm = −1: no result (see lines (09)-(11)), J = [];
(06)  pm = 1; J = [];
(07)  n = size(A, 1); e = (1, ..., 1)^T ∈ R^n; I = diag(e);
(08)  if A is symmetric positive definite, return, end
(09)  if A − I is singular or A + I is singular
(10)    pm = −1; return
(11)  end
(12)  C = (A − I)^{−1}(A + I); R = C^{−1};
(13)  if ϱ(|R|) < 1, return, end
(14)  B = [C − I, C + I];
(15)  b = e;
(16)  γ = min_k |Rb|_k;
(17)  for i = 1 : n
(18)    for j = 1 : n
(19)      b' = b; b'_j = −b'_j;
(20)      if min_k |Rb'|_k > γ, γ = min_k |Rb'|_k; b = b'; end
(21)    end
(22)  end
(23)  [x, S] = intervalhull (B, [b, b]);
(24)  if x ≠ [], return, end
(25)  find x ≠ 0 such that Sx = 0;
(26)  for i = 1 : n
(27)    if x_i ≠ 0, y_i = (Cx)_i/x_i; else y_i = 1; end
(28)  end
(29)  for i = 1 : n
(30)    if y_i ≠ −1 and y_i ≠ 1
(31)      y_i = 1;
(32)      if det(A − I) det(C − diag(y)) > 0, y_i = −1; end
(33)    end
(34)  end
(35)  pm = 0; J = { i | y_i = −1 };
```

Figure 5.1: An algorithm for checking the $P$-property.

```
(01)  function [x, S] = intervalhull (A, b)
(02)  % Computes either the interval hull x
(03)  % of the solution set of Ax = b,
(04)  % or a singular matrix S ∈ A.
(05)   x = [ ]; S = [ ];
(06)   if A_c is singular, S = A_c; return, end
(07)   x_c = A_c^{-1} b_c; z = sgn(x_c); x_ = x_c; x¯ = x_c;
(08)   Z = {z}; D = ∅;
(09)   while Z ≠ ∅
(10)      select z ∈ Z; Z = Z − {z}; D = D ∪ {z};
(11)      [Q_z, S] = qzmatrix (A, z);
(12)      if S ≠ [ ], x = [ ]; return, end
(13)      [Q_{−z}, S] = qzmatrix (A, −z);
(14)      if S ≠ [ ], x = [ ]; return, end
(15)      x¯_z = Q_z b_c + |Q_z|δ;
(16)      x_z = Q_{−z} b_c − |Q_{−z}|δ;
(17)      if x_z ≤ x¯_z
(18)         x_ = min(x_, x_z); x¯ = max(x¯, x¯_z);
(19)         for j = 1 : n
(20)            z' = z; z'_j = −z'_j;
(21)            if ((x_z)_j (x¯_z)_j ≤ 0 and z' ∉ Z ∪ D)
(22)               Z = Z ∪ {z'};
(23)            end
(24)         end
(25)      end
(26)   end
(27)   x = [x_, x¯];
```

Figure 5.2: An algorithm for computing the interval hull.

```
(01)  function [Q_z, S] = qzmatrix (A, z)
(02)  % Computes either a solution Q_z
(03)  % of the equation QA_c − |Q|ΔT_z = I,
(04)  % or a singular matrix S ∈ A.
(05)  for i = 1 : n
(06)     [x, S] = absvaleqn (A_c^T, −T_z Δ^T, e_i);
(07)     if S ≠ [ ], S = S^T; Q_z = [ ]; return
(08)     end
(09)     (Q_z)_{i•} = x^T;
(10)  end
(11)  S = [ ];
```

Figure 5.3: An algorithm for computing the matrix $Q_z$.

```
(01)  function [x, S] = absvaleqn (A, B, b)
(02)  % Finds either a solution x to Ax + B|x| = b, or
(03)  % a singular matrix S satisfying |S − A| ≤ |B|.
(04)  x = [ ]; S = [ ]; i = 0; r = 0 ∈ R^n; X = 0 ∈ R^{n×n};
(05)  if A is singular, S = A; return, end
(06)  z = sgn(A^{-1}b);
(07)  if A + BT_z is singular, S = A + BT_z; return, end
(08)  x = (A + BT_z)^{-1}b;
(09)  C = −(A + BT_z)^{-1}B;
(10)  while z_j x_j < 0 for some j
(11)      i = i + 1;
(12)      k = min{j | z_j x_j < 0};
(13)      if 1 + 2z_k C_kk ≤ 0
(14)          S = A + B(T_z + (1/C_kk)e_k e_k^T);
(15)          x = [ ]; return
(16)      end
(17)      if ((k < n and r_k > max_{k<j} r_j) or (k = n and r_n > 0))
(18)          x = x − X_•k;
(19)          for j = 1 : n
(20)              if (|B||x|)_j > 0, y_j = (Ax)_j/(|B||x|)_j; else y_j = 1; end
(21)          end
(22)          z = sgn(x);
(23)          S = A − T_y|B|T_z;
(24)          x = [ ]; return
(25)      end
(26)      r_k = i;
(27)      X_•k = x;
(28)      z_k = −z_k;
(29)      α = 2z_k/(1 − 2z_k C_kk);
(30)      x = x + αx_k C_•k;
(31)      C = C + αC_•k C_k•;
(32)  end
```

Figure 5.4: An algorithm for solving an absolute value equation.

# Bibliography


[1] W. Barth and E. Nuding, *Optimale Lösung von Intervallgleichungssystemen*, Computing, 12 (1974), pp. 117–125. 1

[2] H. Beeck, *Zur Problematik der Hüllenbestimmung von Intervallgleichungssystemen*, in Interval Mathematics, K. Nickel, ed., Lecture Notes in Computer Science 29, Berlin, 1975, Springer-Verlag, pp. 150–159. 3

[3] G. E. Coxson, *The P-matrix problem is co-NP-complete*, Mathematical Programming, 64 (1994), pp. 173–178. 2

[4] M. Fiedler and V. Pták, *On matrices with non-positive off-diagonal elements and positive principal minors*, Czechoslovak Mathematical Journal, 12 (1962), pp. 382–400. 3

[5] C. Jansson, *Calculation of exact bounds for the solution set of linear interval systems*, Linear Algebra and Its Applications, 251 (1997), pp. 321–340. 3

[6] C. Jansson and J. Rohn, *An algorithm for checking regularity of interval matrices*, SIAM Journal on Matrix Analysis and Applications, 20 (1999), pp. 756–776. 2, 4

[7] J. Rohn, *An algorithm for solving the absolute value equation*, Electronic Journal of Linear Algebra, 18 (2009), pp. 589–599. `http://www.math.technion.ac.il/iic/ela/ela-articles/articles/vol18_pp589-599.pdf`. 5

[8] J. Rohn, *An algorithm for solving the absolute value equation: An improvement*, Technical Report 1063, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague, January 2010. `http://uivtx.cs.cas.cz/~rohn/publist/absvaleqnreport.pdf`. 5

[9] J. Rohn, *A note on generating P-matrices*, Technical Report 1090, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague, November 2010. `http://uivtx.cs.cas.cz/~rohn/publist/genpmat.pdf`. 5

[10] J. Rohn, *On Rump's characterization of P-matrices*, Technical Report 1093, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague, November 2010. `http://uivtx.cs.cas.cz/~rohn/publist/rumppmat.pdf`. 2

[11] J. Rohn, *An algorithm for computing the hull of the solution set of interval linear equations*, Linear Algebra and Its Applications, 435 (2011), pp. 193–201. 2, 3, 5

[12] S. M. Rump, *On P-matrices.*, Linear Algebra Appl., 363 (2003), pp. 237–250. 2