



národní  
úložiště  
šedé  
literatury

## **An Algorithm for Computing All Solutions of an Absolute Value Equation**

Rohn, Jiří  
2010

Dostupný z <http://www.nusl.cz/ntk/nusl-41908>

Dílo je chráněno podle autorského zákona č. 121/2000 Sb.

Tento dokument byl stažen z Národního úložiště šedé literatury (NUŠL).

Datum stažení: 29.08.2024

Další dokumenty můžete najít prostřednictvím vyhledávacího rozhraní [nusl.cz](http://nusl.cz) .



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

## **An Algorithm for Computing All Solutions of an Absolute Value Equation**

Jiří Rohn

Technical report No. V-1091

30.11.2010



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

## **An Algorithm for Computing All Solutions of an Absolute Value Equation**

Jiří Rohn<sup>1</sup>

Technical report No. V-1091

30.11.2010

Abstract:

Presented is an algorithm which in a finite (but exponential) number of steps computes all solutions of an absolute value equation  $Ax + B|x| = b$  ( $A, B$  square), or fails. Failure has never been observed for randomly generated data. The algorithm can also be used for computation of all solutions of a linear complementarity problem.

Keywords:

Absolute value equation, algorithm, all solutions, linear complementarity problem.

---

<sup>1</sup>This work was supported by the Czech Republic Grant Agency under grants 201/09/1957 and 201/08/J020, and by the Institutional Research Plan AV0Z10300504.

# 1 Introduction

We consider here the equation

$$Ax + B|x| = b \tag{1.1}$$

(where  $A, B \in \mathbb{R}^{n \times n}$ ,  $b \in \mathbb{R}^n$ ), called an absolute value equation. This equation was first introduced in [7] and has been since studied by Mangasarian [2], [3], [4], Mangasarian and Meyer [5], Prokopyev [6], and Rohn [8], [9]. In all these papers, the authors are interested in finding *some* solution of (1.1); the problem of finding *all* solutions of (1.1) has been left aside so far apparently because of its expectedly high computational complexity.

In this paper we describe in MATLAB-like style an algorithm named **absvaleqnull** (ABSolute VALue EQUatioN, ALL solutions) called by

$$[X, all] = \text{absvaleqnull}(A, B, b)$$

which in a finite (but exponential) number of steps produces a matrix  $X$  whose columns are solutions of (1.1), and a  $\pm 1$ -number *all* with the following property: if *all* = 1, then  $X$  contains all solutions of (1.1); if *all* = -1, then the columns of  $X$  are still solutions of (1.1), but it is not guaranteed that all of them have been included. Among several hundred examples computed, we have never faced the case of *all* = -1 for randomly generated data. After formulating the algorithm and proving its properties just mentioned in Section 3, we present in Section 5 a randomly generated  $7 \times 7$  example having 10 solutions and a pseudorandomly generated  $10 \times 10$  example having  $2^{10} = 1024$  solutions.

# 2 Notations

We use the following notations.  $A_{k\bullet}$  and  $A_{\bullet k}$  denote the  $k$ th row and the  $k$ th column of  $A$ , respectively. Matrix inequalities, as  $A \leq B$  or  $A < B$ , are understood componentwise. The absolute value of a matrix  $A = (a_{ij})$  is defined by  $|A| = (|a_{ij}|)$ . The same notations also apply to vectors that are considered one-column matrices.  $I$  is the identity matrix and  $e = (1, \dots, 1)^T$  is the vector of all ones. For each  $z \in \mathbb{R}^n$  we denote

$$T_z = \text{diag}(z_1, \dots, z_n) = \begin{pmatrix} z_1 & 0 & \dots & 0 \\ 0 & z_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & z_n \end{pmatrix}.$$

# 3 The algorithm

The algorithm is described in a MATLAB-style code in Fig. 3.1. Following we prove its main property.

**Theorem 1.** *The algorithm (Fig. 3.1) in a finite number of steps produces a matrix  $X$  whose columns are solutions of the equation (1.1). If *all* = 1, then  $X$  contains all solutions of (1.1).*

```

(01) function [X, all] = absvaleqnull(A, B, b)
(02) X = []; all = 1;
(03) n = length(b);
(04) y = 0 ∈ ℝn; z = e ∈ ℝn;
(05) if A + BTz is nonsingular
(06)     x = (A + BTz)-1b;
(07)     C = -(A + BTz)-1B;
(08)     if Tzx ≥ 0, X = [X x]; end
(09) else
(10)     all = -1; return
(11) end
(12) while y ≠ e
(13)     k = min{j | yj = 0};
(14)     for j = 1 : k - 1, yj = 0; end
(15)     yk = 1; zk = -zk;
(16)     if 1 - 2zkCkk ≠ 0
(17)         α = 2zk/(1 - 2zkCkk);
(18)         x = x + αxkC•k;
(19)         C = C + αC•kCk•;
(20)         if Tzx ≥ 0, X = [X x]; end
(21)     else
(22)         all = -1; return
(23)     end
(24) end

```

Figure 3.1: An algorithm for computing all solutions of  $Ax + B|x| = b$ .

*Proof.* According to Theorem 2.1 in [1], the subalgorithm consisting solely of lines (04), (12)-(15), and (24) is finite and constructs all the  $\pm 1$ -vectors  $z$  in  $\mathbb{R}^n$ , with each two subsequently constructed vectors differing in exactly one entry (because of the updating in line (15);  $y$  is an auxiliary  $(0, 1)$ -vector used for finding the  $k$  for which  $z_k$  should be changed to  $-z_k$ ). Thus, the **while** loop is finite, which proves finiteness of the whole algorithm.

Next, in part 2.2 of the proof of Theorem 3.1 in [8] it is proved that after updating in lines (18), (19), the quantities  $x$  and  $C$  always satisfy  $x = (A + BT_z)^{-1}b$ ,  $C = -(A + BT_z)^{-1}B$  for the current  $z$  (invertibility of  $A + BT_z$  is guaranteed by fulfillment of the condition in line (16)). This updating is used in order to circumvent the necessity of solving a large number of systems of linear equations.

A new column  $x$  is added to  $X$  either in line (08), or in line (20). In both cases we have  $x = (A + BT_z)^{-1}b$  (as we have shown in the previous paragraph) and  $T_z x \geq 0$ , hence  $T_z x = |x|$  and  $b = (A + BT_z)x = Ax + BT_z x = Ax + B|x|$ , so that  $x$  is a solution of (1.1).

Finally, if  $all = 1$ , then then the algorithm has constructed all the  $\pm 1$ -vectors  $z$ , and all the matrices of the form  $A + BT_z$ ,  $z$  a  $\pm 1$ -vector, have been found nonsingular (lines (05), (16)). Assume  $x$  is a solution of (1.1). Put  $z_i = 1$  if  $x_i \geq 0$  and  $z_i = -1$  otherwise ( $i = 1, \dots, n$ ), then  $z$  is a  $\pm 1$ -vector satisfying  $T_z x \geq 0$ , so that  $(A + BT_z)x = A + B|x| = b$

and  $x = (A + BT_z)^{-1}b$ , and  $T_z x \geq 0$ . Thus, at the moment the algorithm constructs this vector  $z$ , the condition  $T_z x \geq 0$  is satisfied and  $x$  is added into  $X$  (lines (08) or (20)). This proves that in the case of  $all = 1$  all the solutions of the equation (1.1) have been included into  $X$  as its columns.  $\square$

We have this immediate consequence of the algorithm construction and of Theorem 1:

**Proposition 2.** *In the output of the algorithm, we have  $all = 1$  if and only if  $A + BT_z$  is nonsingular for each  $\pm 1$ -vector  $z$ .*

This result explains why it is almost certain that we get all solutions of (1.1) for randomly generated data: it is almost impossible to generate randomly singular matrices.

## 4 Numerical aspects

The algorithm works as shown in infinite precision arithmetic. However, care should be taken in finite precision arithmetic because frequent updates of  $x$  and  $C$  may lead to essential deterioration of their accuracy. As a remedy, we suggest changing line (20) to

(20) **if**  $T_z x \geq 0$ ,  $x = (A + BT_z)^{-1}b$ ;  $C = -(A + BT_z)^{-1}B$ ;  $X = [X \ x]$ ; **end**  
i.e., to restart  $x$  and  $C$  whenever a new column is being added into  $X$ .

## 5 Examples

If we generate the data in MATLAB randomly by

```
>> A=2*rand(n,n)-1; B=2*rand(n,n)-1; b=2*rand(n,1)-1;
```

(i.e., with entries randomly distributed over  $(-1,1)$ ), then, as a rule, about half of the examples have no solution at all and if solutions exist, their number is usually relatively small (typically less than  $n$ ). However, exceptions do exist. The following randomly generated  $7 \times 7$  example has 10 solutions.

```
>> tic, n=7; rand('state',671); A=2*rand(n,n)-1, B=2*rand(n,n)-1,
>> b=2*rand(n,1)-1, [x,all]=absvaleqnall(A,B,b), toc
```

A =

-0.1479	-0.5985	-0.2265	-0.2292	-0.2426	-0.4978	0.4772
0.3503	0.7914	-0.8554	0.2560	-0.4149	-0.3221	-0.5674
-0.8144	0.8176	-0.9111	-0.9181	0.1953	-0.9376	0.0201
0.1143	-0.8706	-0.1203	0.5198	-0.6242	-0.7633	-0.1536
0.7850	-0.7964	0.6195	-0.5218	0.9041	0.7736	0.9708
-0.4198	-0.5983	0.9180	-0.5057	-0.6677	0.1967	0.0734
-0.1962	0.6255	-0.3860	0.1035	0.4396	-0.7893	-0.9860

```

B =
  -0.8464  -0.5703  -0.9208  -0.0867   0.2831   0.9318   0.8203
  -0.7984   0.3861  -0.1074  -0.1288   0.8478   0.8475   0.8466
   0.3445   0.4156   0.7606  -0.4585   0.9195   0.0428   0.0485
  -0.1394   0.8962  -0.2990  -0.2622  -0.6214  -0.5709  -0.1978
   0.8221   0.1798  -0.2713   0.9308  -0.9663   0.9149  -0.0731
   0.8508  -0.2720  -0.7906  -0.8783   0.5006  -0.9402   0.6437
   0.7253   0.0865   0.5792  -0.1374  -0.0348   0.4932  -0.2036

b =
  -0.6525
   0.3719
   0.6019
  -0.3199
   0.2327
  -0.3168
   0.5135

x =
   0.2842  -1.9018   0.1484  -0.6615  -4.3204  -1.8897   0.2118
   0.2852  -0.3674   0.4041   0.5318  -0.2405   0.4361   0.3700
  -0.0841  -0.7374   0.7863   0.5816  -1.2114  -0.3516   0.1697
  -0.0106   2.2570   0.1354   0.9473   6.0074   2.5083   0.0233
   0.2235  -1.0900  -0.1987  -0.3510  -2.2160  -0.7775   0.2024
   0.0125   0.4788  -0.3220  -0.2792  -0.1360  -0.0891  -0.0745
   0.0045   0.8552   0.2679   0.6275   2.8807   1.2929   0.0600

   0.1584   0.1048   0.2798
   0.3711   0.3815   0.2885
   0.1642   0.2708  -0.0792
  -0.1676  -0.2813  -0.0201
   0.1049  -0.0257   0.2208
  -0.0478  -0.0703   0.0114
  -0.0899  -0.1583  -0.0032

```

```

all =
  1

```

Elapsed time is 0.118065 seconds.

(Computation has been performed on a not-too-fast netbook.) The following pseudorandomly generated  $10 \times 10$  example (notice premultiplication by 0.1 in  $A$ , taking the inverse of  $B$ , and positivity of  $b$ ) has  $2^{10} = 1024$  solutions. We write down neither the data that can be reconstructed because `rand('state',1)` is used, nor the solution matrix  $x$  which is too large; we output in the variable `sols` the number of columns of  $x$  only.

```

>> tic, n=10; rand('state',1); A=0.1*(2*rand(n,n)-1); B=rand(n,n);
>> B=inv(B); b=rand(n,1); [x,all]=absvaleqnall(A,B,b);
>> sols=size(x,2), all, toc
sols =
      1024
all =
      1
Elapsed time is 0.243606 seconds.

```

## 6 Computation of all solutions of a linear complementarity problem

A linear complementarity problem

$$x^+ = Mx^- + q$$

can be recast as an absolute value equation

$$(I + M)x + (I - M)|x| = 2q$$

and solved as such. In this way, our algorithm can be used for computation of all its solutions.



## Bibliography

- [1] M. Fiedler, J. Nedoma, J. Ramík, J. Rohn, and K. Zimmermann, *Linear Optimization Problems with Inexact Data*, Springer-Verlag, New York, 2006. 3
- [2] O. Mangasarian, *Absolute value equation solution via concave minimization.*, Optimization Letters, 1 (2007), pp. 3–8. 2
- [3] O. Mangasarian, *Absolute value programming*, Computational Optimization and Applications, 36 (2007), pp. 43–53. 2
- [4] O. L. Mangasarian, *A Generalized Newton Method for Absolute Value Equations*, Optimization Letters, 3 (2009), pp. 101–108. 2
- [5] O. L. Mangasarian and R. R. Meyer, *Absolute value equations*, Linear Algebra Appl., 419 (2006), pp. 359–367. 2
- [6] O. Prokopyev, *On Equivalent Reformulations for Absolute Value Equations*, Comput. Optim. Appl., 44 (2009), pp. 363–372. 2
- [7] J. Rohn, *Systems of linear interval equations*, Linear Algebra and Its Applications, 126 (1989), pp. 39–78. 2
- [8] 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](http://www.math.technion.ac.il/iic/ela/ela-articles/articles/vol18_pp589--599.pdf). 2, 3
- [9] 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>. 2