Institute of Computer Science
Academy of Sciences of the Czech Republic

An Algorithm for Solving Basic Interval Linear Problems

Jiří Rohn

Technical report No. V-1105

26.03.2011

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 Basic Interval Linear Problems

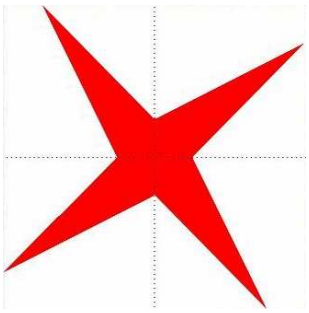
Jiří Rohn[1]

Technical report No. V-1105

26.03.2011

Abstract:

We present an exponential algorithm in MATLAB which, despite consisting of only 23 lines of the source code, solves several basic interval linear problems: solving interval linear systems, inverting interval matrices, checking regularity/singularity and positive definiteness, and finding a singular matrix in an interval matrix (if applicable).



Keywords:
Algorithm, interval linear system, inverse interval matrix, regularity, singularity, positive definiteness, singular matrix.[2]

[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]), see Example 1 here).

1 Introduction

We describe a MATLAB file `basintlinprobs.m` for solving `BAS`ic `INT`erval `LIN`ear `PROB`lem`S`: solving interval linear systems, inverting interval matrices, checking regularity/singularity and positive definiteness, and finding a singular matrix in an interval matrix (if applicable). It can be downloaded from the address
http://uivtx.cs.cas.cz/~rohn/matlab/others/basintlinprobs/basintlinprobs.m .

2 Description

Given a square interval linear system `[Al,Au]x=[bl,bu]` (`l` stands for the lower bound, `u` for the upper bound),
`[xl,xu,Bl,Bu,As,res]=basintlinprobs(Al,Au,bl,bu)`
computes:
in case of regularity of `[Al,Au]`:
the interval hull `[xl,xu]` and
the interval inverse `[Bl,Bu]` of `[Al,Au]`,
in case of singularity:
a singular matrix `As` in `[Al,Au]`;
`res` gives description of the output:
`res=0`: `[Al,Au]` is singular,
`res=1`: `[Al,Au]` is regular,
`res=2`: `[Al,Au]` is positive definite (and regular).
Variables not computed (as `xl`, `xu` in case of singularity) are outputted as empty vectors/matrices, see Example 2 below. The right-hand side `[bl,bu]` must always be present. If you are interested in properties of `[Al,Au]` only, use the dummy variables
`bl=ones(size(Al,1),1)`, `bu=bl` .
The file is written in pure MATLAB, not in INTLAB. Interval arithmetic is not used, so that the results are not verified and may be afflicted with roundoff errors (see the end of Example 1 below).

Because of exponentiality of the algorithm, it is recommended for `size(Al,1)<=10` only. It has been designed as a simple "universal" algorithm mainly for teaching or demo purposes. Use verification software VERSOFT available at http://uivtx.cs.cas.cz/~rohn/matlab/ if you are interested in large-size examples or in verified results.

The algorithm is based on formulae using matrices
`Ayz=(Al+Au)/2-diag(y)*(Au-Al)*diag(z)/2`
and vectors
`by=(bl+bu)/2+diag(y)*(bu-bl)/2`
for all plus/minus-one vectors `y`, `z`, see
http://uivtx.cs.cas.cz/~rohn/publist/!handbook.pdf, pp. 20, 31, 54, 58,
http://uivtx.cs.cas.cz/~rohn/publist/47.pdf, pp. 45, 47.

The algorithm *could be improved in various ways* (e.g., using interval arithmetic, or avoiding unnecessary computations in particular cases [as evaluation of `xl`, `xu` if you are interested in the inverse only]), but the author preferred presentation in the most simple and perhaps also most understandable form consisting of only 23 lines of the source code. Once the basic design of the algorithm is understood, potential users may adapt it to their particular purposes.

EXAMPLE 1: Regular matrix (Barth and Nuding [1], see the logo on the abstract page).

```
Al =
     2    -2
    -1     2
Au =
     4     1
     2     4
bl =
    -2
    -2
bu =
     2
     2
>> [xl,xu,Bl,Bu,As,res]=basintlinprobs(Al,Au,bl,bu)
xl =
   -4.0000
   -4.0000
xu =
    4.0000
    4.0000
Bl =
    0.1667   -0.5000
   -1.0000    0.1667
Bu =
    1.0000    1.0000
    0.5000    1.0000
As =
     []
res =
    1
```

But a closer inspection reveals the influence of roundoff errors:

```
>> format long, xl, xu
xl =
  -4.000000000000004
  -4.000000000000000
xu =
   4.000000000000000
   3.999999999999996
```

This is why this file is destined for teaching or demo purposes only, not for scientific computations.

EXAMPLE 2: Singular matrix ([2], p. 69).

```
Al =
     2     4     1
    -6    -3     3
    -4    -5     2
Au =
     3     5     2
    -5    -2     4
     0    -4     3
bl =
     1
     1
     1
bu =
     1
     1
     1
>> [xl,xu,Bl,Bu,As,res]=basintlinprobs(Al,Au,bl,bu)
xl =
     []
xu =
     []
Bl =
     []
Bu =
     []
As =
    2.0000    5.0000    1.0000
   -5.0000   -3.0000    4.0000
   -3.9355   -4.0161    2.0161
res =
    0
>> det(As)
ans =
  1.4344e-013
```

3 The file

Following we give a screenshot of the MATLAB editor showing the file[3] which itself can be downloaded from
http://uivtx.cs.cas.cz/~rohn/matlab/others/basintlinprobs/basintlinprobs.m .

```
1   function [xl,xu,Bl,Bu,As,res]=basintlinprobs(Al,Au,bl,bu)
2 - Ac=(Al+Au)/2; Delta=(Au-Al)/2; delta=(bu-bl)/2; xl=[]; xu=[]; Bl=[]; Bu=[]; As=[]; res=0; [n,n]=size(Ac); I=eye(n,n)
3 - if rank(Al)<n, As=Al; return, end
4 - D=inv(Al); b=bu; x=Al\bu; Bl=D; Bu=D; xl=x; xu=x; y=ones(1,n); z=ones(1,n); t=zeros(1,2*n);
5 - while any(t~=ones(1,2*n))
6 -     k=find(t==0,1); t(1:(k-1))=zeros(1,k-1); t(k)=1;
7 -     if k<=n
8 -         i=k; p=Ac(i,:)*D-I(i,:);
9 -         if 2*p(i)+1<=0, tau=(-y(i))/(2*p(i)); As=Ac-(diag(y)-2*tau*I(:,i)*I(i,:))*Delta*diag(z);
10 -            xl=[]; xu=[]; Bl=[]; Bu=[]; return
11 -        end
12 -        alpha=2/(2*p(i)+1); x=x-alpha*(y(i)*delta(i)+p*b)*D(:,i);
13 -        D=D-alpha*D(:,i)*p; b(i)=b(i)-2*y(i)*delta(i); y(i)=-y(i);
14 -    else
15 -        j=k-n; p=D*Ac(:,j)-I(:,j);
16 -        if 2*p(j)+1<=0, tau=(-z(j))/(2*p(j)); As=Ac-diag(y)*Delta*(diag(z)-2*tau*I(:,j)*I(j,:));
17 -            xl=[]; xu=[]; Bl=[]; Bu=[]; return
18 -        end
19 -        alpha=2/(2*p(j)+1); x=x-alpha*x(j)*p; D=D-alpha*p*D(j,:); z(j)=-z(j);
20 -    end
21 -    xl=min(xl,x); xu=max(xu,x); Bl=min(Bl,D); Bu=max(Bu,D);
22 - end
23 - [R,pd]=chol(Al); if (isequal(Al',Al)&&isequal(Au',Au)&&(pd==0)), res=2; else res=1; end
```

The first version of the file comes from April 2005. The author had been hesitating for years to go ahead with publication of this exponential algorithm. The final impetus came from Günter Mayer (Rostock, Germany) on March 22, 2011. The author wishes to thank him for his encouragement.

[3] Line 2 should end with a semicolon ";". It had been wrongly cut off while processing the jpg file.

Bibliography

[1] W. Barth and E. Nuding, *Optimale Lösung von Intervallgleichungssystemen*, Computing, 12 (1974), pp. 117–125. 1, 3

[2] J. Rohn, *Systems of linear interval equations*, Linear Algebra and Its Applications, 126 (1989), pp. 39–78. 4