VERSOFT: Examples

Rohn, Jiří
2011

Institute of Computer Science
Academy of Sciences of the Czech Republic

VERSOFT: Examples

Jiří Rohn

Technical report No. V-1119

27.07.2011

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

VERSOFT: Examples

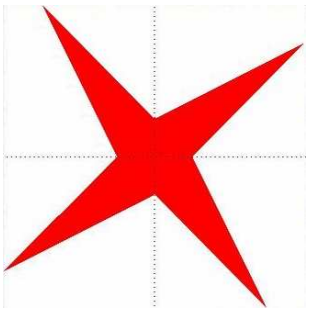
Jiří Rohn[1]

Technical report No. V-1119

27.07.2011

Abstract:

Some examples of the use of VERSOFT files are presented.



Keywords:
Verification, basis, rank, RREF, pseudoinverse, SVD, QR, determinant, spectral decomposition, Cholesky decomposition, positive definiteness, matrix function, linear equations, linear programming.[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])).

1 Introduction

Some examples of the use of VERSOFT [2] files are presented here. For space reasons, we use consistently the `format short` option for the output. To view the results in full accuracy, type `format long` and run the respective commands on your computer.

2 Basis, rank and nullspace

Let us start with the matrix

```
>> A=[1 2 3 4; 5 6 7 8; 9 10 11 12]
A =
     1     2     3     4
     5     6     7     8
     9    10    11    12
```

A verified basis of the range space of A, chosen from among the columns of A, can be computed by

```
>> B=verbasis(A)
B =
     1     2
     5     6
     9    10
```

(notice the non-interval output), and a verified *orthonormal* basis by

```
>> Q=verorth(A)
intval Q =
[    0.1378,    0.1379] [   -0.9025,   -0.9024]
[    0.4972,    0.4973] [   -0.2935,   -0.2934]
[    0.8566,    0.8567] [    0.3154,    0.3155]
```

This means that the interval matrix Q is verified to contain a real matrix Qo whose columns are orthonormal and form a basis of the range space of A (see `help verorth`). Hence, Q'*Q must contain the unit matrix. This can be checked by

```
>> Q'*Q
intval ans =
[    0.9999,    1.0001] [   -0.0001,    0.0001]
[   -0.0001,    0.0001] [    0.9999,    1.0001]
```

Since size(Q,2)=2, it follows that the rank of A is equal to 2. This can also be established independently by using

```
>> [r1,r2]=verrank(A)
r1 =
     2
r2 =
     2
```

Here (see `help verrank`), r1, r2 are integers such that the rank of A is verified to satisfy r1 $\leq$ rank(A) $\leq$ r2. Since r1=r2, we conclude that rank(A)=2.

The null space of A can be described by

```
>> N=vernull(A)
intval N =
[  0.2999,    0.3001] [ -0.4001,   -0.3999] [ -0.1001,   -0.0999] [  0.1999,    0.2001]
[ -0.4001,   -0.3999] [  0.6999,    0.7001] [ -0.2001,   -0.1999] [ -0.1001,   -0.0999]
[ -0.1001,   -0.0999] [ -0.2001,   -0.1999] [  0.6999,    0.7001] [ -0.4001,   -0.3999]
[  0.1999,    0.2001] [ -0.1001,   -0.0999] [ -0.4001,   -0.3999] [  0.2999,    0.3001]
```

Here (see `help vernull`), N is an interval matrix verified to contain a real matrix No such that the null space Null(A) of A is described (in exact arithmetic) by Null(A) = { No*y | y $\in R^4$ }.

3 RREF and pseudoinverse

The RREF (reduced row-echelon form) of A can be computed by

```
>> AR=verrref(A)
intval AR =
[  1.0000,    1.0000] [  0.0000,    0.0000] [ -1.0001,   -0.9999] [ -2.0001,   -1.9999]
[  0.0000,    0.0000] [  1.0000,    1.0000] [  1.9999,    2.0001] [  2.9999,    3.0001]
[  0.0000,    0.0000] [  0.0000,    0.0000] [  0.0000,    0.0000] [  0.0000,    0.0000]
```

The underlying ones and zeros are always exact. We can see again that the rank of A equals 2.

To compute a verified pseudoinverse (or, Moore-Penrose inverse) of A, use

```
>> X=verpinv(A)
intval X =
[   -0.3751,   -0.3749] [   -0.1001,   -0.0999] [    0.1749,    0.1751]
[   -0.1459,   -0.1458] [   -0.0334,   -0.0333] [    0.0791,    0.0792]
[    0.0833,    0.0834] [    0.0333,    0.0334] [   -0.0167,   -0.0166]
[    0.3124,    0.3126] [    0.0999,    0.1001] [   -0.1126,   -0.1124]
```

As before, this means that the interval matrix X is verified to contain the exact pseudoinverse Xo of A. Hence, A*Xo*A-A=0, so that A*X*A-A must contain the zero matrix. This can be checked by

```
>> A*X*A-A
intval ans =
  1.0e-011 *
[ -0.0160,    0.0159] [ -0.0195,    0.0194] [ -0.0230,    0.0229] [ -0.0264,    0.0264]
[ -0.0441,    0.0441] [ -0.0538,    0.0539] [ -0.0635,    0.0636] [ -0.0732,    0.0734]
[ -0.0727,    0.0729] [ -0.0887,    0.0890] [ -0.1047,    0.1052] [ -0.1205,    0.1212]
```

4 SVD and QR

The function verthinsvd computes a verified *thin* SVD (singular value decomposition) of A (see `help verthinsvd`):

```
>> [U,S,V]=verthinsvd(A)
intval U =
[   -0.2068,   -0.2067] [   -0.8892,   -0.8891] [   -0.4083,   -0.4082]
[   -0.5183,   -0.5182] [   -0.2544,   -0.2543] [    0.8164,    0.8165]
[   -0.8299,   -0.8298] [    0.3803,    0.3804] [   -0.4083,   -0.4082]
intval S =
[   25.4368,   25.4369] [    0.0000,    0.0000] [    0.0000,    0.0000]
[    0.0000,    0.0000] [    1.7226,    1.7227] [    0.0000,    0.0000]
[    0.0000,    0.0000] [    0.0000,    0.0000] [    0.0000,    0.0001]
intval V =
[   -0.4037,   -0.4036] [    0.7328,    0.7329] [    0.4775,    0.4776]
[   -0.4648,   -0.4647] [    0.2898,    0.2899] [   -0.8367,   -0.8366]
[   -0.5259,   -0.5258] [   -0.1532,   -0.1531] [    0.2407,    0.2408]
[   -0.5870,   -0.5869] [   -0.5962,   -0.5961] [    0.1183,    0.1184]
```

The interval matrices U, S, V are verified to contain real matrices Uo, So and Vo such that A=Uo*So*Vo' is a singular value decomposition of A (in exact arithmetic). Indeed, we have

```
>> A-U*S*V'
intval ans =
  1.0e-013 *
[ -0.1200,    0.1222] [ -0.1111,    0.1022] [ -0.0933,    0.1244] [ -0.1244,    0.1555]
[ -0.1866,    0.2132] [ -0.1777,    0.2399] [ -0.1954,    0.2576] [ -0.2310,    0.2754]
[ -0.2487,    0.3020] [ -0.2665,    0.2843] [ -0.2665,    0.3908] [ -0.3020,    0.3908]
```

Notice that the third singular value is verified to belong to the interval S(3,3)= [0.0000, 0.0001], so that it cannot be decided whether it is zero or not. This shows that, contrary to unverified computations, SVD is not a proper way to compute verified rank; use `verrank` instead.

So far all the computations terminated with verified results. This, however, is no more true with the QR decomposition:

```
>> [Q,R]=verqr(A)
intval Q =
[       NaN,       NaN] [       NaN,       NaN] [       NaN,       NaN]
[       NaN,       NaN] [       NaN,       NaN] [       NaN,       NaN]
[       NaN,       NaN] [       NaN,       NaN] [       NaN,       NaN]
intval R =
[     NaN,       NaN] [     NaN,       NaN] [     NaN,       NaN] [     NaN,       NaN]
[  0.0000,    0.0000] [     NaN,       NaN] [     NaN,       NaN] [     NaN,       NaN]
[  0.0000,    0.0000] [  0.0000,    0.0000] [     NaN,       NaN] [     NaN,       NaN]
```

A NaN output is a VERSOFT way to say "no verified output" (R is preset to be always upper triangular, therefore the zeros). To see the reason for this outcome, use an additional output variable E:

```
>> [Q,R,E]=verqr(A)
intval Q =
[       NaN,       NaN] [       NaN,       NaN] [       NaN,       NaN]
[       NaN,       NaN] [       NaN,       NaN] [       NaN,       NaN]
[       NaN,       NaN] [       NaN,       NaN] [       NaN,       NaN]
intval R =
[     NaN,       NaN] [     NaN,       NaN] [     NaN,       NaN] [     NaN,       NaN]
[  0.0000,    0.0000] [     NaN,       NaN] [     NaN,       NaN] [     NaN,       NaN]
[  0.0000,    0.0000] [  0.0000,    0.0000] [     NaN,       NaN] [     NaN,       NaN]
E =
    error: 'house: normalizing not possible'
    where: 'j = 3'
    value: 'nx = [0, 2.5802e-013]'
```

Here, the fields of the structure E show that the breakdown occurred in the subroutine `house` (Householder transformation) for the index value j=3 in a **for ... end** loop, where a division by a normalizing factor nx was to be performed, but it could not be realized because nx contained zero. Attention: an error variable is not present in all files; if yes, it is always denoted by E and is placed as the last output variable (see the respective `helps`).

5 Determinant, spectral decomposition, Cholesky decomposition and positive definiteness

So far we have been using a 3-by-4 matrix A. For the purpose of files handling square matrices, let us now switch to a 3-by-3 symmetric positive definite matrix A3:

```
>> A3=[1 2 3; 4 5 6; 7 8 10]
A3 =
     1     2     3
     4     5     6
     7     8    10
>> A3=A3'*A3
A3 =
    66    78    97
    78    93   116
    97   116   145
```

To compute the determinant, use

```
>> dt=verdet(A3)
intval dt =
[    9.0000,    9.0000]
```

The five-decimal-digit output suggests that both the bounds might be equal. To verify this, evaluate the width of dt:

```
>> dt.sup-dt.inf
ans =
     0
```

The zero width shows that the determinant has been indeed computed exactly (which, however, is an exception occurring only for integer matrices of moderate sizes, usually up to 7).

Verified spectral decomposition of a symmetric matrix can be computed by

```
>> [L,X]=verspectdec(A3)
intval L =
[  303.1953,  303.1954] [    0.0000,    0.0000] [    0.0000,    0.0000]
[    0.0000,    0.0000] [    0.7659,    0.7660] [    0.0000,    0.0000]
[    0.0000,    0.0000] [    0.0000,    0.0000] [    0.0387,    0.0388]
intval X =
[    0.4646,    0.4647] [   -0.8333,   -0.8332] [    0.2995,    0.2996]
[    0.5537,    0.5538] [    0.0094,    0.0095] [   -0.8327,   -0.8326]
[    0.6909,    0.6910] [    0.5527,    0.5528] [    0.4658,    0.4659]
```

Here, L and X are interval matrices verified to contain real matrices Lo, Xo such that Lo is a diagonal matrix whose diagonal is formed by all eigenvalues of A3, Xo is orthogonal, and A3=Xo*Lo*Xo' holds in exact arithmetic (a spectral decomposition of A3). Since all three eigenvalues of A3 are verified to be positive, this also verifies that A3 is positive definite.

Another way of verifying positive definiteness consists in computing a verified Cholesky decomposition:

```
>> [ch,L]=verchol(A3)
ch =
     1
intval L =
[    8.1240,    8.1241] [    0.0000,    0.0000] [    0.0000,    0.0000]
[    9.6011,    9.6012] [    0.9045,    0.9046] [    0.0000,    0.0000]
[   11.9398,   11.9399] [    1.5075,    1.5076] [    0.4082,    0.4083]
```

Here, L is verified to contain the exact Cholesky factor Lo of A3, so that A3=Lo*Lo' holds in exact arithmetic. "ch=1" indicates that A3 is verified positive definite; "ch=0" would mean that A3 was verified *not* to be positive definite.

6 Matrix functions

The multi-purpose function VERMATFUN allows computation of verified matrix functions. The respective function (in our case, the square root) should be given as an inline function (between apostrophes) in the first input variable. The third input value "1" should be included if we know in advance that the result is real (otherwise it could be computed as complex).

```
>> F=vermatfun('sqrt(x)',A3,1)
intval F =
[    4.3849,    4.3850] [    4.4244,    4.4245] [    5.2150,    5.2151]
[    4.4244,    4.4245] [    5.4760,    5.4761] [    6.5907,    6.5908]
[    5.2150,    5.2151] [    6.5907,    6.5908] [    8.6235,    8.6236]
```

The interval matrix F is verified to contain a real matrix Fo satisfying Fo*Fo=A3 in exact arithmetic, so that F*F-A3 must contain the zero matrix. This can be checked by

```
>> F*F-A3
intval ans =
  1.0e-010 *
[   -0.1493,    0.1493] [   -0.1258,    0.1255] [   -0.1638,    0.1638]
[   -0.1778,    0.1781] [   -0.1498,    0.1498] [   -0.1954,    0.1952]
[   -0.2186,    0.2188] [   -0.1831,    0.1831] [   -0.2391,    0.2391]
```

7 Linear equations

Now, let us return to our original rectangular matrix

```
>> A
A =
     1     2     3     4
     5     6     7     8
     9    10    11    12
```

and let

```
>> b=A*ones(4,1)
b =
    10
    26
    42
```

To solve the system of linear equations A*x=b, use

```
>> [sol,x,B]=verlinsys(A,b)
sol =
   Inf
intval x =
[   -2.0001,   -1.9999]
[    5.9999,    6.0001]
[    0.0000,    0.0000]
[    0.0000,    0.0000]
intval B =
[  0.2999,    0.3001] [ -0.4001,   -0.3999] [ -0.1001,   -0.0999] [  0.1999,    0.2001]
[ -0.4001,   -0.3999] [  0.6999,    0.7001] [ -0.2001,   -0.1999] [ -0.1001,   -0.0999]
[ -0.1001,   -0.0999] [ -0.2001,   -0.1999] [  0.6999,    0.7001] [ -0.4001,   -0.3999]
[  0.1999,    0.2001] [ -0.1001,   -0.0999] [ -0.4001,   -0.3999] [  0.2999,    0.3001]
```

x, B are verified to contain xo and Bo such that the set of solutions of A*x=b is described as { xo+Bo*y | y $\in R^4$ } in exact arithmetic; thus the system has infinitely many solutions, which is also expressed by "sol=Inf". Observe that xo is a "basic solution": it has size(A,2)-rank(A) zero entries.

Let A be reduced to its first two columns. Then we have

```
>> [sol,x,B]=verlinsys(A(:,1:2),b)
sol =
     1
intval x =
[   -2.0001,   -1.9999]
[    5.9999,    6.0001]
intval B =
[    0.0000,    0.0000] [    0.0000,    0.0000]
[    0.0000,    0.0000] [    0.0000,    0.0000]
```

"sol=1" indicates that the system is verified to have exactly one solution which is verified to be contained in x. In this case B is a matrix of interval zeros by default.

Now take a randomly generated right-hand side:

```
>> b=rand(3,1)
b =
    0.3529
    0.8132
    0.0099
>> [sol,x,B]=verlinsys(A(:,1:2),b)
sol =
     0
intval x =
[       NaN,       NaN]
[       NaN,       NaN]
intval B =
[       NaN,       NaN] [       NaN,       NaN]
[       NaN,       NaN] [       NaN,       NaN]
```

The system is verified to possess no solution ("sol=0"); x and B consist of NaN's.

8 Linear programming

The last set of examples concerns the VERLINPROG file for verified linear programming. In all three examples the data are randomly generated with the random number generator being always initialized anew, so that the data are reproducible:

```
>> rand('state',2); A=2*rand(3,5)-1, b=2*rand(3,1)-1, c=2*rand(5,1)-1,
>> [flag,x,y,h]=verlinprog(A,b,c)
A =
    0.7503    0.3530    0.0582   -0.5126   -0.2864
   -0.3642   -0.8577   -0.6565    0.6858   -0.5352
   -0.4535   -0.6068    0.7399    0.1153    0.2952
b =
    0.9853
   -0.2309
   -0.7043
```

```
c =
    0.9433
   -0.5629
    0.2150
    0.1218
    0.4669
flag =
verified optimum
intval x =
[    1.7025,    1.7026]
[    0.0000,    0.0000]
[    0.0028,    0.0029]
[    0.5701,    0.5702]
[    0.0000,    0.0000]
intval y =
[    4.1681,    4.1682]
[    2.8710,    2.8711]
[    2.5103,    2.5104]
intval h =
[    1.6760,    1.6761]
```

As stated in the variable flag, a verified optimum has been achieved: it is verified that x contains an optimum, y contains a dual optimum, and h contains the optimal value.

```
>> rand('state',1); A=2*rand(3,5)-1, b=2*rand(3,1)-1, c=2*rand(5,1)-1,
>> [flag,x,y,h]=verlinprog(A,b,c)
A =
    0.9056    0.1963    0.6736   -0.2482   -0.6009
    0.4081    0.6815    0.0374    0.7972   -0.3938
    0.9078   -0.1144   -0.9556   -0.1420    0.0766
b =
    0.8205
    0.0506
   -0.3863
c =
   -0.9311
    0.4307
    0.5374
   -0.8810
    0.2542
flag =
verified unbounded
intval x =
[    0.5391,    0.5392]
[    0.0000,    0.0000]
[    0.9582,    0.9583]
[    0.0000,    0.0000]
```

```
[    0.5212,    0.5213]
intval y =
[    3.2172,    3.2173]
[    0.0000,    0.0000]
[    3.3771,    3.3772]
[    2.0431,    2.0432]
[    7.7909,    7.7910]
intval h =
[       NaN,       NaN]
```

The problem is verified unbounded; x is verified to enclose a primal feasible solution xo, and y is verified to enclose a vector yo such that the objective tends to -Inf along the feasible half-line { xo+t*yo | t≥0 } (see `help verlinprog`).

```
>> rand('state',3); A=2*rand(3,5)-1, b=2*rand(3,1)-1, c=2*rand(5,1)-1,
>> [flag,x,y,h]=verlinprog(A,b,c)
A =
    0.0323   -0.5674    0.6431   -0.6163   -0.1337
   -0.5495   -0.1456   -0.2613   -0.5058    0.2221
   -0.6327    0.9412   -0.9409    0.1344   -0.9030
b =
    0.6154
    0.0173
   -0.3695
c =
    0.8260
    0.1815
    0.1503
   -0.9993
   -0.6816
flag =
verified infeasible
intval x =
[       NaN,       NaN]
[       NaN,       NaN]
[       NaN,       NaN]
[       NaN,       NaN]
[       NaN,       NaN]
intval y =
[   -2.5958,   -2.5957]
[   -0.4838,   -0.4837]
[   -1.6398,   -1.6397]
intval h =
[       NaN,       NaN]
```

The problem is verified infeasible; y is verified to enclose a Farkas vector yo satisfying A'*yo≥0, b'*yo<0 in exact arithmetic (whose existence proves primal infeasibility).

Bibliography

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

[2] J. Rohn, *VERSOFT: Verification software in MATLAB/INTLAB*, 2009. http://uivtx.cs.cas.cz/~rohn/matlab. 2