BERGISCHE UNIVERSITÄT WUPPERTAL
FACHBEREICH C – MATHEMATIK
UND NATURWISSENSCHAFTEN

WISSENSCHAFTLICHES RECHNEN/
SOFTWARETECHNOLOGIE
W. KRÄMER

Solving large banded systems of linear equations with componentwise small error bounds

El Paso, October 2, 2008

Overview

1. Remarks on freely available software
2. The mathematical background
3. Linear vector iteration/wrapping effect
4. Using local coordinate systems to improve interval forward/backward substitutions
5. Numerical examples

References: Cordes 89, Rump 92, 93, K 91, K & Lohner 94, K & Hölbig & Diverio 05, ...

Free software to solve sparse linear (interval) systems $Ax = b$

- Intlab (only spd systems)
- PASCAL-XSC (point systems, banded)
- C-XSC (banded coefficient matrices)

Trivial test case: $A =$ identity matrix, right hand side $b = 1$:

- Intlab: smaller than dimension $n = \quad 47.000$
- XSC: $n = 2.000.000$ and more

MatLab: $n = 10.000.000$ (point system, no verification)

Free software, Intlab code

```
n=47000;
A=speye(n);
full(A(1:5,1:5))
b=ones(n,1);
b(1:3)
tic; x= verifylss(A,b); toc
x(1:3)
ans =
     1     0     0     0     0
     0     1     0     0     0
     0     0     1     0     0
     0     0     0     1     0
     0     0     0     0     1
ans =
     1
     1
     1
??? Matrix is too large to convert to linear index.
Error in ==> verifylss at 98
  if any(isnan(A(:))) | any(isinf(A(:))) | any(isnan(b(:))) | any(isinf(b(:)))
```

Comparing free software

$$A := \begin{pmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{pmatrix}, \; b_i := [0,1], i = 1, 2, \ldots$$

Results for $n = 10.000$:

Intlab: $x_1 = x_n =$ `[-6.4E8, 6.4E8]`
$x_{n/2} =$ `[-6.3E8, 6.4E8]`

XSC: $x_1 = x_n =$ `[-2.1E-6, 5.1E3]`
$x_{n/2} =$ `[-6.6E-3, 1.3E7]`

Comparing free software

$$A = \begin{pmatrix} 5 & -4 & 1 & & & & \\ -4 & 6 & -4 & 1 & & & \\ 1 & -4 & 6 & -4 & 1 & & \\ & & \ddots & \ddots & \ddots & & \\ & & 1 & -4 & 6 & -4 & 1 \\ & & & 1 & -4 & 6 & -4 \\ & & & & 1 & -4 & 5 \end{pmatrix}, \; b_i := 1, \; i = 1, 2, \ldots$$

$n = 16.000$, $\text{cond}(A) \approx$ `1E16`

Intlab: NaN

XSC: $x_1 = x_n =$ `[1.7069866799E11, 1.7069866801E11]`
after 4 iteration steps

Intlab approach: Cholesky factorization, lower bounds for smallest singular values, norm estimations to bound $||\hat{x} - \tilde{x}||_\infty$

Krawczyk/Rump iteration

$A = LU$ without pivoting ==> L and U preserve structure of A

$$K([X]) := U^{-1}L^{-1}\left(\underbrace{b - A\tilde{x}}_{\text{defect}} + (LU - A)[X]\right) \subseteq \text{ interior}([X])$$

implies

$$\hat{x} = A^{-1}b \in \tilde{x} + K([X])$$

Essential step: compute validated inclusion of the solution of a triangular linear system (non trivial task!)

Solving a triangular system

$$[z^0] = \begin{pmatrix} [-1,1] \\ [-1,1] \end{pmatrix}$$

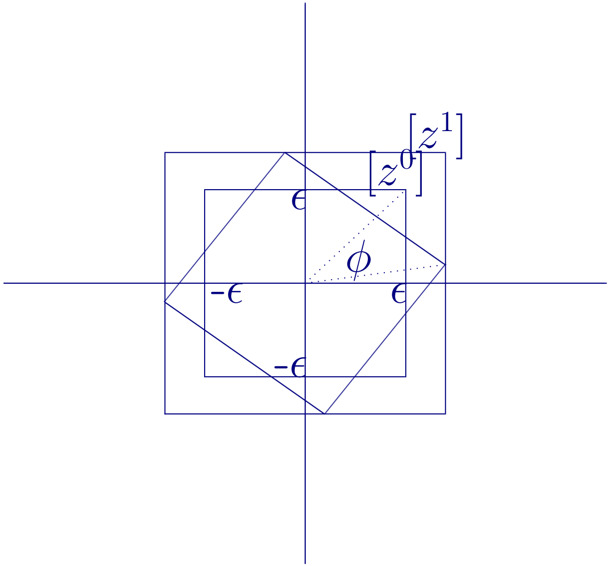
Linear vector iteration $z^{k+1} := Az^k, k = 0, 1, 2 \dots$

We are looking for enclosures of $S_k := \{ A^k z^0 \mid z^0 \in [z^0] \}$

Using interval operations, i.e. $[z^{k+1}] := A \cdot [z^k]$:

With $\phi := \frac{\pi}{4}, A := \begin{pmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{pmatrix}$ each step corresponds to the inclusion of an interval vector rotated by ϕ (wrapping).

Wrapping effect



Solving a triangular system

Define lower triangular linear system with coefficient matrix

$$M := \begin{pmatrix} I & & & & & \\ A & -I & & & & \\ & & \ddots & \ddots & \ddots & \\ & & & A & -I & \\ & & & & A & -I \end{pmatrix}$$

and right hand side b with $b_0 = [z^0], b_k = (0,0)^T, k > 1$.

$==>$ Forward substitution in interval arithmetic is equivalent to the interval computation of the vector iteration described above, i.e. it holds $[x_k] = [z^k], k = 0, 1, 2 \ldots$

$==>$ Breakdown of interval forward substitution after a few steps due to overestimation (wrapping effect) leading to overflow

Neumaier's example

$$A := \begin{pmatrix} 1 & & & & & \\ 1 & 1 & & & & \\ 1 & 1 & 1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & 1 & 1 & 1 & \\ & & & 1 & 1 & 1 \end{pmatrix}, b_i := [-1, 1], i = 1, 2, \ldots$$

For the true solution $x = (x_k)$ it holds $x_k \subseteq [-n, n], k = 1, 2, \ldots n$.

But interval forward substitution (based on IEEE double numbers) fails for $n = 41$.

Idea to reduce wrapping effect in vector iteration

If the vector iteration results in a rotation by the angle ϕ of the initial set, the local coordinate system can be the original coordinate system rotated by ϕ. Intermediate results are represented as the (unevaluated) product of a real matrix by an ordinary interval vector.

More general: use parallel-epipeds (Lohner)

$$P = P(B, [z], y) := \{\, y + Bz \mid z \in [z] \,\}$$

B regular basis matrix, $[z]$ interval vector, y point vector.

How to find basis matrix B?

Use approximately orthogonal matrix $B = Q$ with Q coming from a QR-decomposition.

Vector iteration using QR-decomposition

Algorithm 1: $[y^0] = [z^0], B_0 = I$

$$\begin{aligned} B_j &\approx A \cdot B_{j-1} \\ Q_j R_j &\approx B_j \quad \text{approximate QR-decomposition} \\ B_j &= Q_j \\ [y^j] &= [(B_j)^{-1}] \cdot A \cdot B_{j-1} \cdot [y^{j-1}] \\ [z^j] &= B_j \cdot [y^j] \end{aligned}$$

Theorem 1: If Algorithm 1 works, it holds for $k > 0$

$$z^k = A \cdot z^{k-1} = A^k z^0 \in [z^k] \text{ for all } z^0 \in [z^0].$$

Inverse of an approximately orthogonal matrix

Remark: Q orthogonal $==> Q^{-1} = Q^T$

Theorem 2: If $||I - QQ^T||_\infty =: s < 1$, it holds

$$Q^{-1} \in [Q^{-1}] := Q^T + [-\epsilon, \epsilon] \begin{pmatrix} 1 & \dots & 1 \\ \vdots & & \vdots \\ 1 & \dots & 1 \end{pmatrix} \text{ with } \epsilon := \frac{s}{1-s} ||Q^T||_\infty .$$

Ideas used to solve triangular systems $Ax = b$

Lower triangular system $Ax = b$:

Dimension n, lower bandwidth l.

Interval forward substitution using parallel-epipeds
based on QR-decompositions.

QR-decompositions have to be done for l-by-l square matrices.

Computation of l-by-l inverse matrices.

XSC source code to solve lower triangular systems

```
procedure lss_lower( var A : rmatrix; var b,x : ivector; l : integer );
{ Forward substitution using coordinate transformations }
var i,j,n,err: integer;
    ...
begin
   cOld:= id(cOld);
   bi:= 0;
   for i:= 1 to l do begin
       x[i]:= ##( b[i] - for j:= 1 to i-1 sum (A[i,j-i]*x[j]) ) / A[i,0];
       y[i]:= x[i];
   end;
   for i:= l+1 to n do begin
      for j:= 1 to l do af[j]:= -A[i,j-l-1] / A[i,0];
      bi[l]:= b[i] / A[i,0];
      ai:= af*cOld;
      c:= QR( mid(ai), y );
      inv( c, cInv, err );
      y:= (cInv*ai)*y + cInv*bi;
      x[i]:= c[l]*y;
      cOld:= c;
   end;
end;
```

XSC source code to solve triangular systems

```
procedure lss( var A : rmatrix; b : ivector; var x : ivector );
const eps = 0.1;
var Lo          : rmatrix[lb(A,1)..ub(A,1),lb(A,2)..0      ];
    Up          : rmatrix[lb(A,1)..ub(A,1),      0..ub(A,2)];
    LU_A        : imatrix[lb(A,1)..ub(A,1),lb(A,2)..ub(A,2)];
    b_app,x_app : rvector[lb(A,1)..ub(A,1)];
    defect,z,za : ivector[lb(A,1)..ub(A,1)];
    i,j,k,l,n,m : integer;
begin
   n:=      ub(A,1);
   l:= abs( lb(A,2) );
   k:=      ub(A,2);
   if ( l = 0 ) or ( k = 0 ) then
   begin
      lss_triangular( A, mid(b), x_app );
      for i:= 1 to n do
         defect[i]:= ##( b[i] - for j:= max(1,i-l) to min(n,i+k)
                                   sum ( A[i,j-i]*x_app[j] )      );
      lss_triangular( A, defect, z );
      x:= x_app + z;
   end else
```

Reconsidering Neumaier's example

$$A := \begin{pmatrix} 1 & & & & & \\ 1 & 1 & & & & \\ 1 & 1 & 1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & 1 & 1 & 1 & \\ & & & 1 & 1 & 1 \end{pmatrix}, b_i := [-1, 1], i = 1, 2, \dots$$

For the true solution $x = (x_k)$, it holds $x_k \subseteq [-n, n], k = 1, 2, \dots n$.

Forward substitution using coordinate transformations (QR-decompositions):

$n = 2.000.000$

$x_1 =$`[ -1.0, 1.0 ]`, and $x_n =$`[ -2.8E11, 2.8E11 ]`

XSC source code to solve banded systems $Ax = b$

```
{ Compute the LU-factorization and the defect LU-A : }
lu_decomp( A,Lo,Up,LU_A );

{ Compute an approximate solution x_app : }
lss_triangular( Lo,mid(b),b_app ); lss_triangular( Up,b_app,x_app );

{ Compute the defect := b - A*x_app of the approx. solution x_app : }
for i:= 1 to n do
   defect[i]:= ##( b[i] - for j:= max(1,i-l) to min(n,i+k)
                           sum ( A[i,j-i]*x_app[j] )       );
z:= defect;
m := 0;                      { Iteration until inclusion is obtained }
repeat                       { or max. iteration count is exceeded : }
   za:= blow( z, eps );
   for i:= 1 to n do
      z[i]:= ##( defect[i] + for j:= max(1,i-l) to min(n,i+k)
                              sum ( LU_A[i,j-i]*za[j] )    );
   lss_triangular( Lo,z,x );
   lss_triangular( Up,x,z );
   m:= m + 1;
until ( z in za ) or ( m = 10 );
x:= x_app + z;
```

M-matrix

$$A = \begin{pmatrix} 1 & -1 & -1 & & & & & \\ -1 & 2 & 0 & -1 & & & & \\ -1 & 0 & 3 & 0 & -1 & & & \\ & -1 & 0 & 3 & 0 & -1 & & \\ & & \ddots & \ddots & \ddots & & & \\ & & & -1 & 0 & 3 & 0 & -1 \\ & & & & -1 & 0 & 3 & 0 \\ & & & & & -1 & 0 & 3 \end{pmatrix}$$

$n = 30,\ b_k =$ `[-1,1]`, $k = 1, 2 \ldots :$

Intlab: $x_1 =$ `[ -1.3E13, 1.3E13 ]`
XSC: $x_1 =$ `[ -3.0E12, 3.0E12 ]`
Using a solver for dense matrices: $x_1 =$ `[ -2.94E12, 2.94E12 ]`

M-matrix

$$A = \begin{pmatrix} 1 & -1 & -1 & & & & & & \\ -1 & 2 & 0 & -1 & & & & & \\ -1 & 0 & 3 & 0 & -1 & & & & \\ & -1 & 0 & 3 & 0 & -1 & & & \\ & & \ddots & \ddots & \ddots & & & & \\ & & & -1 & 0 & 3 & 0 & -1 \\ & & & & -1 & 0 & 3 & 0 \\ & & & & & -1 & 0 & 3 \end{pmatrix}$$

$n = 35, b_k =$ `[-1,1]`, $k = 1, 2 \ldots :$

Intlab: $x_n =$ `[ -3.0437e15, 3.0437e15]`
XSC: $x_n =$ `[ -2.9e7, 2.9e7 ]`
Using a solver for dense matrices: $x_n =$ `[ -2.4158e7, 2.4158e7]`

M-matrix, XSC, exact scalar products

```
Dimension  n: 300
Bandwidths l, k: 2 2
A:  -1  0  3  0  -1
Change elements ? (y/n) y
  row, col, new value : 1 1  1
  row, col, new value : 1 2 -1
  row, col, new value : 2 1 -1
  row, col, new value : 2 2  2
  row, col, new value : 0 0
b: =[-1,1]
Norm(LU-A):  0  <== !
Iteration: 0
Iteration: 1
Result validated: TRUE
x =
  1: [              -2.2E+125,                 2.2E+125 ]
  2: [              -1.4E+125,                 1.4E+125 ]
299: [              -9.5E+062,                 9.5E+062 ]
300: [              -5.9E+062,                 5.9E+062 ]
```

Componentwise good error bounds

$$A = \begin{pmatrix} 5 & -4 & 1 & & & & \\ -4 & 6 & -4 & 1 & & & \\ 1 & -4 & 6 & -4 & 1 & & \\ & & \ddots & \ddots & \ddots & & \\ & & 1 & -4 & 6 & -4 & 1 \\ & & & 1 & -4 & 6 & -4 \\ & & & & 1 & -4 & 5 \end{pmatrix}$$

$n = 1000$, cond$(A) \approx$ `1.65E11`, exponent range $-5..5$

```
A: 1 -4  6 -4  1
row, col, new value :    1    1  5
row, col, new value : 1000 1000  5

Solution will be close to x[i] = 10^( m*(n+1-2*i)/(n-1) )
Enter exponent range m: 5
Norm(LU-A):  7.77E-16
Iteration: 0
Iteration: 1
Iteration: 2
Result validated: TRUE
```

Componentwise good error bounds

XSC result:

```
   1: [ 9.999999999999895E+4, 9.999999999999897E+4 ]
   2: [ 9.772146969726118E+4, 9.772146969726121E+4 ]

 500: [   1.0115920238409E+0,   1.0115920238411E+0 ]

 999: [     1.02380212529E-5,     1.02380212531E-5 ]
1000: [     1.00024277420E-5,     1.00024277422E-5 ]

Max. rel. error =  1.9E-10  at i = 958
```

Intlab results:

```
n: 1000
m: 5
x(1): 1.0e+005 *
         [   2.39998562840655,   2.40001437159339]

x(n/2):  [   0.96283958680437,   3.83715827001644]
x(n):    [  -1.43713534471308,   1.43718333849899]
```

WISSENSCHAFTLICHES RECHNEN/
SOFTWARETECHNOLOGIE
WALTER KRÄMER

Solving large banded systems of linear equations with componentwise small error bounds

Thank you!

El Paso, October 2, 2008

M-matrix, XSC

```
Dimension  n: 35
Bandwidths l, k:
2 2
A:
-0.1 0 0.3 0 -0.1
Change elements of A? (y/n) y
row, col, new value : 1 1 0.1
row, col, new value : 1 2 -0.1
row, col, new value : 2 1 -0.1
row, col, new value : 2 2 0.2
row, col, new value : 0 0
b:
=[-1,1]
Change elements of b? (y/n) n
Norm(LU-A):  3.053113335516752E-017
Iteration: 0
Iteration: 1
Result validated: TRUE
x =
  1: [              -3.8E+015,                3.8E+015 ]
  2: [              -2.3E+015,                2.3E+015 ]
  3: [              -1.5E+015,                1.5E+015 ]
  4: [              -8.8E+014,                8.8E+014 ]
  5: [              -5.5E+014,                5.5E+014 ]


 15: [              -4.4E+012,                4.4E+012 ]
 16: [              -2.8E+012,                2.8E+012 ]
 17: [              -1.7E+012,                1.7E+012 ]
 18: [              -1.1E+012,                1.1E+012 ]
```

```
 19: [               -6.5E+011,              6.5E+011 ]

 30: [               -3.3E+009,              3.3E+009 ]
 31: [               -2.0E+009,              2.0E+009 ]
 32: [               -1.4E+009,              1.4E+009 ]
 33: [               -7.5E+008,              7.5E+008 ]
 34: [               -4.0E+008,              4.0E+008 ]
 35: [               -2.5E+008,              2.5E+008 ]

max. rel. error =  0.000000000000000E+000  at i = 0
max. abs. error =  7.402832252635707E+015  at i = 1
min. abs. x[35] = [               -2.5E+008,              2.5E+008 ]
max. abs. x[1] =  [               -3.8E+015,              3.8E+015 ]
```

M-matrix, Intlab

```
n=35, t=-1;
u=sparse(3:n,1:n-2,t); u(n,n)=0;
v=sparse(2:n,1:n-1,t); v(n,n)=0;
w=sparse(1:n,1:n,1); a=(u+v+w);
a=0.1*full(a*a');
b=ones(n,1)*intval('[-1,1]');
bm1= -1*ones(n,1);
bp1= +1*ones(n,1);
full(a(1:7,1:7))
tic; x=verifylss(a,b); toc
x(1:1); x(n:n)
tic; x=verifylss(a,bm1); toc
x(1:1); x(n:n)
tic; x=verifylss(a,bp1); toc
x(1:1); x(n:n)
disp('Conditin number: '), disp(cond(a))
a=sparse(a);
tic; x=verifylss(a,b); toc
x(1:1); x(n:n)
n =
    35
ans =
    0.1   -0.1   -0.1      0      0      0      0
   -0.1    0.2      0   -0.1      0      0      0
   -0.1      0    0.3      0   -0.1      0      0
      0   -0.1      0    0.3      0   -0.1      0
      0      0   -0.1      0    0.3      0   -0.1
      0      0      0   -0.1      0    0.3      0
      0      0      0      0   -0.1      0    0.3
Elapsed time is 0.041048 seconds.
```

```
intval ans =
  1.0e+008 *
[   -2.5920,    2.5920]
Elapsed time is 0.036202 seconds.
intval ans =
  1.0e+008 *
[   -2.3942,   -2.3941]
Elapsed time is 0.014014 seconds.
intval ans =
  1.0e+008 *
    2.3941
Conditin number:
   1.0133e+15
Elapsed time is 0.010214 seconds.
intval ans =
  1.0e+016 *
[   -3.2784,    3.2784]
```

M-matrix XSC

```
Dimension  n: 39
Bandwidths l, k:
2 2
A:
-0.1 0 0.3 0 -0.1
Change elements of A? (y/n) y
row, col, new value : 1 1 0.1
row, col, new value : 1 2 -0.1
row, col, new value : 2 1 -0.1
row, col, new value : 2 2 0.2
row, col, new value : 0 0
b:
=[-1,1]
Change elements of b? (y/n) n
Norm(LU-A):  3.053113335516752E-017
Iteration: 0
Iteration: 1
Iteration: 2
Result validated: TRUE
x =
  1: [              -4.5E+017,              4.5E+017 ]
  2: [              -2.8E+017,              2.8E+017 ]
  3: [              -1.8E+017,              1.8E+017 ]
  4: [              -1.1E+017,              1.1E+017 ]
  5: [              -6.6E+016,              6.6E+016 ]


 17: [              -2.1E+014,              2.1E+014 ]
 18: [              -1.3E+014,              1.3E+014 ]
 19: [              -7.8E+013,              7.8E+013 ]
```

```
 20: [              -4.8E+013,              4.8E+013 ]
 21: [              -3.0E+013,              3.0E+013 ]

 34: [              -5.8E+010,              5.8E+010 ]
 35: [              -3.6E+010,              3.6E+010 ]
 36: [              -2.4E+010,              2.4E+010 ]
 37: [              -1.4E+010,              1.4E+010 ]
 38: [              -7.1E+009,              7.1E+009 ]
 39: [              -4.4E+009,              4.4E+009 ]

max. rel. error =  0.000000000000000E+000  at i = 0
max. abs. error =  8.971519104780430E+017  at i = 1
min. abs. x[39] = [              -4.4E+009,              4.4E+009 ]
max. abs. x[1]  = [              -4.5E+017,              4.5E+017 ]
```

M-matrix XSC

```
intvalinit('DisplayInfSup')
n=1000; u=sparse(2:n,1:n-1,-1); u(n,n)=1; u(n,n)=0; %full(u)
o=sparse(1:n-1,2:n,-1); o(n,n)=1; o(n,n)=0; % full(o)
a= u +  sparse(1:n,1:n,2.0) + o;
a=full(a*a');
a(1:4,1:4)
a(n-3:n,n-3:n)
x=ones(n,1);
disp('n:'); disp(n);
disp('m:'); m=5; disp(m);
for i=1:n
  x(i)= 2.4*10^floor( (m*(n+1-2*i))/(n-1) );
end
b=a*x;
%b=intval('[-1,1]')*b;
%a=sparse(a);
if issparse(a)
  disp('SPARSE matrix');
else
  disp('FULL matrix');
end

tic; x=verifylss(a,b); toc
for i=1:n
  x(i);
end
disp('x(1):  '), x(1)
disp('x(n/2):'), x(n/2)
disp('x(n):  '), x(n)
===> Default display of intervals by infimum/supremum (e.g. [ 3.14 , 3.15 ])
```

```
ans =
     5    -4     1     0
    -4     6    -4     1
     1    -4     6    -4
     0     1    -4     6
ans =
     6    -4     1     0
    -4     6    -4     1
     1    -4     6    -4
     0     1    -4     5
n:
        1000
m:
     5
FULL matrix
Elapsed time is 10.335566 seconds.
x(1):
intval ans =
  1.0e+005 *
[   2.39999999999996,   2.39999999999997]
x(n/2):
intval ans =
[   2.39999892841040,   2.39999892841041]
x(n):
intval ans =
  1.0e-004 *
[   0.23996892951158,   0.23996892951159]
>> intvalinit('DisplayInfSup')
n=1000; u=sparse(2:n,1:n-1,-1); u(n,n)=1; u(n,n)=0; %full(u)
o=sparse(1:n-1,2:n,-1); o(n,n)=1; o(n,n)=0; % full(o)
a= u +  sparse(1:n,1:n,2.0) + o;
a=full(a*a');
a(1:4,1:4)
a(n-3:n,n-3:n)
x=ones(n,1);
disp('n:'); disp(n);
```

```
disp('m:'); m=5; disp(m);
for i=1:n
  x(i)= 2.4*10^floor( (m*(n+1-2*i))/(n-1) );
end
b=a*x;
%b=intval('[-1,1]')*b;
a=sparse(a);
if issparse(a)
  disp('SPARSE matrix');
else
  disp('FULL matrix');
end

tic; x=verifylss(a,b); toc
for i=1:n
  x(i);
end
disp('x(1):  '), x(1)
disp('x(n/2):'), x(n/2)
disp('x(n):  '), x(n)
ans =
     5    -4     1     0
    -4     6    -4     1
     1    -4     6    -4
     0     1    -4     6
ans =
     6    -4     1     0
    -4     6    -4     1
     1    -4     6    -4
     0     1    -4     5
n:
        1000
m:
     5
SPARSE matrix
Elapsed time is 0.061001 seconds.
```

```
x(1):
intval ans =
  1.0e+005 *
[   2.39998562840655,   2.40001437159339]
x(n/2):
intval ans =
[   0.96283958680437,   3.83715827001644]
x(n):
intval ans =
[  -1.43713534471308,   1.43718333849899]
>> intvalinit('DisplayInfSup')
n=1000; u=sparse(2:n,1:n-1,-1); u(n,n)=1; u(n,n)=0; %full(u)
o=sparse(1:n-1,2:n,-1); o(n,n)=1; o(n,n)=0; % full(o)
a= u +  sparse(1:n,1:n,2.0) + o;
a=full(a*a');
a(1:4,1:4)
a(n-3:n,n-3:n)
x=ones(n,1);
disp('n:'); disp(n);
disp('m:'); m=5; disp(m);
for i=1:n
  x(i)= 2.4*10^floor( (m*(n+1-2*i))/(n-1) );
end
b=a*x;
b=intval('[-1,1]')*b;
%a=sparse(a);
if issparse(a)
  disp('SPARSE matrix');
else
  disp('FULL matrix');
end

tic; x=verifylss(a,b); toc
for i=1:n
  x(i);
end
```

```
disp('x(1):  '), x(1)
disp('x(n/2):'), x(n/2)
disp('x(n):  '), x(n)
ans =
     5    -4     1     0
    -4     6    -4     1
     1    -4     6    -4
     0     1    -4     6
ans =
     6    -4     1     0
    -4     6    -4     1
     1    -4     6    -4
     0     1    -4     5
n:
        1000
m:
     5
FULL matrix
Elapsed time is 9.733518 seconds.
x(1):
intval ans =
  1.0e+009 *
[  -7.08020096789044,   7.08020096789044]
x(n/2):
intval ans =
  1.0e+012 *
[  -1.53368179029783,   1.53368179029783]
x(n):
intval ans =
  1.0e+009 *
[  -4.10526884606013,   4.10526884606013]
>> intvalinit('DisplayInfSup')
n=1000; u=sparse(2:n,1:n-1,-1); u(n,n)=1; u(n,n)=0; %full(u)
o=sparse(1:n-1,2:n,-1); o(n,n)=1; o(n,n)=0; % full(o)
a= u +  sparse(1:n,1:n,2.0) + o;
a=full(a*a');
```

```
a(1:4,1:4)
a(n-3:n,n-3:n)
x=ones(n,1);
disp('n:'); disp(n);
disp('m:'); m=5; disp(m);
for i=1:n
  x(i)= 2.4*10^floor( (m*(n+1-2*i))/(n-1) );
end
b=a*x;
b=intval('[-1,1]')*b;
a=sparse(a);
if issparse(a)
  disp('SPARSE matrix');
else
  disp('FULL matrix');
end

tic; x=verifylss(a,b); toc
for i=1:n
  x(i);
end
disp('x(1):   '), x(1)
disp('x(n/2):'), x(n/2)
disp('x(n):   '), x(n)
ans =
     5    -4     1     0
    -4     6    -4     1
     1    -4     6    -4
     0     1    -4     6
ans =
     6    -4     1     0
    -4     6    -4     1
     1    -4     6    -4
     0     1    -4     5
n:
        1000
```

```
m:
     5
SPARSE matrix
Elapsed time is 0.049785 seconds.
x(1):
intval ans =
  1.0e+016 *
[  -1.87460102308363,   1.87460102308363]
x(n/2):
intval ans =
  1.0e+016 *
[  -1.87460102308363,   1.87460102308363]
x(n):
intval ans =
  1.0e+016 *
[  -1.87460102308363,   1.87460102308363]
```