CODEN: LUTFD2/(TFRT-7392)/1-10/(1988)

Calculation of a Real Sorted Schur Decomposition

Michael Lundh

Department of Automatic Control
Lund Institute of Technology
June 1988

Department of Automatic Control **Lund Institute of Technology** P.O. Box 118 S-221 00 Lund Sweden	*Document name* Internal Report
	Date of issue June 1988
	Document Number CODEN: LUTFD2/(TFRT-7392)/1-10/(1988)
Author(s) Michael Lundh	*Supervisor*
	Sponsoring organisation

Title and subtitle
Calculation of a Real Sorted Schur Decomposition

Abstract

An algorithm is presented for calculation of a real sorted Schur form of a matrix with real entries.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title		*ISBN*
Language English	*Number of pages* 10	*Recipient's notes*
Security classification		

Introduction

Any matrix $A \in C^{n \times n}$ can be decomposed as

$$B = T^H A T$$

where $T \in C^{n \times n}$ is a unitary matrix, i.e. $T^H T = I$, and B upper triangular with eigenvalues on the diagonal. The use of unitary matrices give nice numerical properties. If the matrix $A \in R^{n \times n}$ the matrix B could be an upper block triangular real matrix and the corresponding $T \in R^{n \times n}$ is orthogonal. For a matrix on real Schur form it is often required that the blocks are sorted such that the spectral abscissas for the blocks are descending or ascending.
Schur factorization of a matrix is a powerful tool in system theory. Sorted real Schur forms can be used in many applications, e.g. solution of Riccati equations (Laub, 1985), spectral decomposition (Nichols, 1987). However, despite their excellent numerical properties they are not used as often as desirable, e.g. the Algebraic Riccati equation solver in MATLAB does not use real Schur forms.
This paper provides some MATLAB functions for computing a real sorted Schur form of a matrix with real entries. By itself MATLAB provides a function that calculates a real Schur form that however is unsorted. The important algorithm in this paper sorts the blocks of a real Schur matrix.

Method

MATLAB provides a routine SCHUR for real Schur factorization. This yields a $n \times n$ matrix

$$B_0 = \begin{pmatrix} b_{11} & \dots & \dots & b_{1n} \\ 0 & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & b_{nn} \end{pmatrix}$$

with eigenvalues of the blocks b_{ii} in any order. It is necessary to sort the blocks.

Sorting in Ascending Order

The sorting is done by applying a number of orthogonal transformations T_i to B_0. Each of these is a Givens rotation. A Givens rotation is defined by

$$T_j = \begin{pmatrix} c & -s \\ s & c \end{pmatrix}$$

where $c^2 + s^2 = 1$. Using this type of transformation matrix two subsequent blocks b_{ii-1} and b_{ii} are interchanged.

Algorithm for Sorting: First the block with the smallest spectral abscissa is moved to the upper left corner. This normally requires several transformations, each interchanging two subsequent blocks. Then neglect the upper left block and the corresponding rows and columns. Apply the algorithm on the reduced matrix. After n steps the blocks are sorted in ascending order.

Sorting in Descending Order

After the algorithm above has been applied a final transformation

$$T_f = \begin{pmatrix} 0 & \dots & 1 \\ \vdots & \iddots & \vdots \\ 1 & \dots & 0 \end{pmatrix}$$

reverts the rows and columns of the sorted matrix.

MATLAB Functions

A Real Sorted Schur Form is computed as follows: First the MATLAB routine SCHUR computes an unsorted real Schur decomposition. Then blocks are sorted in ascending order. The blocks have size 1×1 or 2×2. Four different cases may then occur when permuting two subsequent blocks. Changing a block of size i with a block of size j is done by the routine $CHij$. The transformation matrices are calculated using function GIV. The method for these transformations are found in van Doren (1980). Finally, if desired, the rows and columns are reverted to yield a matrix sorted in descending order.

Riccati Equations

Linear quadratic regulator design implies solving an algebraic Riccati equation. The continuous time algebraic Riccati equation

$$SA + A^TS - SBQ_2^{-1}B^TS + Q_1 = 0$$

is solved using the method in Laub (1985).
Form the Hamiltonian matrix M of size $2n \times 2n$.

$$M = \begin{pmatrix} A & -BQ_2^{-1}B^T \\ -Q_1 & -A^T \end{pmatrix}$$

Determine an orthogonal T_1 such that

$$M_s = T_1^T M T_1$$

is on real Schur form. Sort the blocks with an orthogonal transformation T_2 such that

$$\begin{aligned} M_{ss} = \begin{pmatrix} M_{11} & M_{12} \\ 0 & M_{22} \end{pmatrix} &= T_2^T M_s T_2 \\ &= (T_1T_2)^T M (T_1T_2) \end{aligned}$$

where M_{11} has stable eigenvalues. The eigenvalues of M_{22} will be unstable. Partition T into four $n \times n$ blocks

$$T = T_1T_2 = \begin{pmatrix} T_{11} & T_{12} \\ T_{21} & T_{22} \end{pmatrix}$$

then the solution of the Riccati equation is found with

$$S = T_{21}T_{11}^{-1}$$

and the state feedback gain is

$$L = Q_2^{-1}B^TS$$

Implementation

All MATLAB functions are listed in appendix 1. They are also found in directory `bode::use:[michael.packages.matlab.realschur]`, and will be submitted to "PACLIB".

Example 1

The first example shows how the algorithm works. Compute a real sorted Schur form of a matrix with eigenvalues

$$\lambda = \begin{cases} -1.0000 \\ -0.7071 \pm 0.7071i \\ 1.4142 \pm 1.4142i \\ 2.0000 \end{cases}$$

The different steps are displayed in appendix 2. (b_i displays intermediate results) First a real Schur form is calculated (b_0). The block sizes are 1, 2, 2 and 1. Then the following blocks are interchanged (3,4), (2,3) and (1,2) yielding b_1, b_2 and b_3. Now the block with the most negative spectral abscissa is in the upper left corner. Change blocks 3 and 4 (b_4), 2 and 3 (b_5), and finally blocks 3 and 4. Now the matrix S is a real sorted Schur decomposition of the initial matrix A and Q is the orthogonal transformation matrix.

Example 2

This example demonstrates how the routine can be used to solve a LQ design problem. Determine the optimal state feedback gain matrix L such that the feedback law $u = -Lx$ minimizes the cost function

$$J = \int x^T Q_1 x + u^T Q_2 u \, \mathrm{dt}$$

for the laboratory servo used in the course Sampled Data Control.

$$\dot{x} = \begin{pmatrix} 0 & 9.25 \\ 0 & -0.12 \end{pmatrix} x + \begin{pmatrix} 0 \\ 2.66 \end{pmatrix} u$$

With weight matrices $Q_1 = \mathrm{diag}(2 \quad 0)$ and $Q_2 = 0.1$ the design is performed by `[l,s]=lqrx(a,b,Q1,Q2)` yielding

$$L = \begin{pmatrix} 4.47 & 6.37 \end{pmatrix}$$
$$S = \begin{pmatrix} 0.31 & 0.17 \\ 0.17 & 0.24 \end{pmatrix}$$

The closed loop system will have eigenvalues $\lambda = -8.53 \pm 6.11i$.

Conclusions

An algorithm for computation of a real sorted Schur form of a matrix with real entries is presented. The result from the MATLAB function SCHUR is sorted by a new algorithm.

References

VAN DOREN, P. (1980): "A Generalized Eigenvalue Approach for Solving Riccati Equations," Numerical Analysis Project, Stanford.

LAUB, A. (1985): "Lectures on Numerical Linear Algebra in Control," Report CODEN: LUTFD2/TFRT-7312, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

NICHOLS, N. K. (1987): "Robustness in Partial Pole Placement," *IEEE, Transactions on Automatic Control*, **32**, 728–732.

Appendix 1

MATLAB functions

```
function [q,b] = rssf(x,updown)
%RSSF [Q,B] = RSSF(X,UPDOWN)
%       Real Sorted Schur Form of a matrix X with real elements.
%       Produces a matrix B with the real eigenvalues on the
%       diagonal and the complex eigenvalues in 2-by-2 blocks on
%       the diagonal. Also an orthogonal transformation matrix Q
%       is produced so that B = Q'*X*Q and Q'*Q = EYE(Q).
%       The optional parameter UPDOWN determines whether the real
%       part of the eigenvalues of B are increasing (UPDOWN>=0 default)
%       or decreasing (UPDOWN<0).

%       Michael Lundh
%       LastEditDate : Mon Jun  6 13:28:06 1988

if any(any(imag(x))),  error(' X not real '); end;

%-------------------------- Schur decomposition
[q0,a]=schur(x);

%-------------------------- Determine blocksizes and real part of eigenvalues
[n,n]=size(a)       ;
as=[diag(a,-1) ;0] ;
i=1;  s=1;
while i<=n,
  if abs(as(i))>1.0E-10,
    stru(s)=2;
    rela(s)=( a(i,i) + a(i+1,i+1) )/2;
  else
    stru(s)=1;
    rela(s)=a(i,i);
  end;
  i=i+stru(s);
  s=s+1;
end;

%-------------------------- Sort eigenvalues using orthogonal tranformations
nbl= max(size(stru));
q  = eye(a);
b  = a;

for bl=1:nbl,
  [m,mii]=min(rela(bl:nbl));  % search min eigenvalue in below actual point
  mii=bl-1+mii;
  while mii>bl,               % change min eigen value in actual point
    ihi=sum( stru(1:mii) );
    ilo=ihi-stru(mii-1)-stru(mii)+1;
    comp=10*stru(mii-1)+stru(mii);
    if comp<11.5,
      z=ch11( b(ilo:ihi,ilo:ihi) );
    elseif comp<12.5,
      z=ch12( b(ilo:ihi,ilo:ihi) );
    elseif comp<21.5,
      z=ch21( b(ilo:ihi,ilo:ihi) );
    elseif comp<22.5,
      z=ch22( b(ilo:ihi,ilo:ihi) );
    else
      error('stupidity');
    end
    zz=eye(n);
    zz(ilo:ihi,ilo:ihi) = z;
    q=q*zz;
    b=zz'*b*zz;
    tmp = rela(mii);  rela(mii) = rela(mii-1);  rela(mii-1) = tmp;
    tmp = stru(mii);  stru(mii) = stru(mii-1);  stru(mii-1) = tmp;
    [m,mii]=min(rela(bl:nbl));
    mii=bl-1+mii;
  end
```

```
end

%--- Total transformation matrix
q=q0*q;

%--- Sort if required in descending order
if nargin==2,
  if updown<0,
    nq=max(size(q));
    q1=zeros(nq);
    for i=1:nq,
      q1(i,nq-i+1)=1;
    end;
    q=q*q1;
    b=q1*b*q1;
  end;
end;
```

```
function [c,s] = giv(x,y)
%GIV  [C,S] = GIV(X,Y)
%     Sine and Cosine for Givens Rotation

%     Michael Lundh
%     LastEditDate : Mon Jun  6 10:01:44 1988
den = sqrt( x*x + y*y );
% if den=0.0, c=1; s=0; else
c = x/den;
s = y/den;
```

```
function q = ch11(a)
%CH11 Q = CH11(A)
%     Change 1x1 and 1x1 blocks by Givens Rotation

%     Michael Lundh
%     LastEditDate : Mon Jun  6 10:11:33 1988

[c,s]=giv( a(1,2) , a(1,1)-a(2,2) );
q    = [ c s ; -s c ];
```

```
function q = ch12(a)
%CH12 Q = CH12(A)
%     Change 1x1 and 2x2 blocks by Givens Rotation

%     Michael Lundh
%     LastEditDate : Mon Jun  6 10:10:40 1988

h = a(1,1)*eye(3) - a                  ;
[c,s]=giv( h(3,3) , h(3,2) )           ;
r=[1 0 0 ; 0 c s ; 0 -s c ]            ;
hr=h*r                                 ;

[c,s]=giv( hr(1,2) ,hr(2,2) )          ;  % z1
z1=[c s 0 ; -s c 0 ; 0 0 1]            ;
z1hr=z1*hr                             ;

[c,s]=giv( z1hr(2,3) , z1hr(3,3) );  % z2
z2=[1 0 0 ; 0 c s ; 0 -s c]            ;
% rrz=z2*z1hr

q=(z2*z1)';
```

```
function q = ch21(a)
%CH21 Q = CH21(A)
%     Change 2x2 and 1x1 blocks by Givens Rotation

%     Michael Lundh
%     LastEditDate : Mon Jun  6 10:10:22 1988

h = a(3,3)*eye(3) - a              ;
[c,s]=giv( h(1,1) , -h(2,1) )      ;
r=[c -s 0 ; s c 0 ; 0 0 1]'        ;
rth=r'*h                           ;

[c,s]=giv( rth(2,3) , rth(2,2) )  ;   % z1
z1=[1 0 0 ; 0 c s ; 0 -s c]        ;
rthz=rth*z1                        ;

[c,s]=giv( rthz(1,2) , rthz(1,1) );   % z2
z2=[c s 0 ; -s c 0 ; 0 0 1]        ;
%rrz=rthz*z2

q=z1*z2;
```

```
function q = ch22(a)
%CH22 Q = CH22(A)
%     Change 2x2 and 2x2 blocks by Givens Rotation

%     Michael Lundh
%     LastEditDate : Mon Jun  6 10:10:11 1988

lr=( a(3,3)+a(4,4) )/2                            ;
li=sqrt(a(3,4)*a(4,3) + ((a(3,3)-a(4,4))^2)/4 ) ;

h = ((lr+li)*eye(4) - a)*((lr-li)*eye(4) - a)     ;

[c,s]=giv( h(1,1) , -h(2,1) )          ;
r=[c -s 0 0;s c 0 0;0 0 1 0;0 0 0 1]' ;
rh=r'*h                                ;

[c,s]=giv( rh(2,3) ,rh(2,2) )          ;    % q1
q1=[1 0 0 0;0 c s 0;0 -s c 0;0 0 0 1] ;
rh=rh*q1                               ;

[c,s]=giv( rh(1,2) ,rh(1,1) )          ;    % q2
q2=[c s 0 0;-s c 0 0;0 0 1 0;0 0 0 1] ;
rh=rh*q2                               ;

[c,s]=giv( rh(2,4) ,rh(2,3) )          ;    % q3
q3=[1 0 0 0;0 1 0 0;0 0 c s;0 0 -s c] ;
rh=rh*q3                               ;

[c,s]=giv( rh(1,3) ,rh(1,2) )          ;    % q4
q4=[1 0 0 0;0 c s 0;0 -s c 0;0 0 0 1] ;
%rh=rh*q4

q=q1*q2*q3*q4;
```

```
function [k,s]=lqrx(a,b,q,r)
%LQRX   Linear quadratic regulator design for continuous-time systems.
%       [K,S] = LQR(A,B,Q,R)  calculates the optimal feedback gain matrix K
%       such that the feedback law  u = -Kx  minimizes the cost function:
%
%               J = Integral {x'Qx + u'Ru} dt
%                                                .
%       subject to the constraint equation:   x = Ax + Bu
%       Also returned is S, the steady-state solution to the associated
%       algebraic Riccati equation.
%       Real sorted Schur decomposition is used.

%       Michael Lundh
%       LastEditDate : Wed Jun  8 10:49:14 1988

error(nargchk(4,4,nargin));
error(abcdchk(a,b));

[m,n] = size(a);
[mb,nb] = size(b);
[mq,nq] = size(q);
if (m ~= mq) | (n ~= nq)
        error('A and Q must be the same size')
end
[mr,nr] = size(r);
if (mr ~= nr) | (nb ~= mr)
        error('B and R must be consistent')
end

% Check if q is positive semi-definite and symmetric
if any(eig(q) < 0) | (norm(q'-q,1)/norm(q,1) > eps)
        error('Q must be symmetric and positive semi-definite')
end
% Check if r is positive definite and symmetric
if any(eig(r) <= 0) | (norm(r'-r,1)/norm(r,1) > eps)
        error('R must be symmetric and positive definite')
end

% Solve Riccati equation using real sorted schur form
[v,d] = rssf([a -b/r*b';-q, -a']);
s = v(n+1:n+n,1:n)/v(1:n,1:n);
k = r\b'*s;
```

Appendix 2

Example 1 with intermediate results

```
A = 2.4142   -0.4142   -4.6569    0.8284    9.6569    8.0000
    1.0000         0         0         0         0         0
         0    1.0000         0         0         0         0
         0         0    1.0000         0         0         0
         0         0         0    1.0000         0         0
         0         0         0         0    1.0000         0

[Q,S]=rssf(A)

b0= 2.0000    2.1827   -6.3928   -3.8480    2.7600   -8.2922
         0   -0.0255    2.5430    1.2737   -0.8878    2.8143
         0   -1.6016    2.8540    1.7449   -1.2888    3.6594
         0         0         0   -0.7582   -0.6185    0.1061
         0         0         0    0.8127   -0.6560    1.0704
         0         0         0         0         0   -1.0000

b1= 2.0000    2.1827   -6.3928   -2.8518   -9.1082    0.3040
         0   -0.0255    2.5430    0.9831    3.0596   -0.0563
         0   -1.6016    2.8540    1.2365    4.0653   -0.2021
         0         0         0   -1.0000   -0.5062    0.3064
         0         0         0    0.0000   -1.0209   -0.5053
         0         0         0    0.0000    1.1844   -0.3933

b2= 2.0000   -0.7157    2.1338   -6.9785   -9.1082    0.3040
         0   -1.0000    0.4603   -1.5603   -2.0683    0.3597
         0    0.0000    0.0160    2.6125    2.9083   -0.0285
         0         0   -1.5138    2.8124    3.6617   -0.0880
         0    0.0000    0.0000    0.0000   -1.0209   -0.5053
         0    0.0000    0.0000    0.0000    1.1844   -0.3933

b3=-1.0000   -0.7157   -0.9429    3.1370    4.1254   -0.4204
         0    2.0000    1.9687   -6.4260   -8.3796    0.2123
    0.0000    0.0000    0.0160    2.6125    2.9083   -0.0285
         0         0   -1.5138    2.8124    3.6617   -0.0880
    0.0000    0.0000    0.0000    0.0000   -1.0209   -0.5053
    0.0000    0.0000    0.0000    0.0000    1.1844   -0.3933

b4=-1.0000   -0.7157    0.7545    0.1122   -0.7825    5.1703
         0    2.0000   -1.5067   -0.8701    1.5251  -10.4919
    0.0000    0.0000   -1.0469   -0.7614    0.7135   -4.6187
    0.0000    0.0000    0.8083   -0.3673   -0.0094    0.4949
    0.0000    0.0000    0.0000    0.0000    0.2629    3.0621
    0.0000    0.0000    0.0000    0.0000   -1.0860    2.5655

b5=-1.0000    0.3196   -0.0540   -0.9944   -0.7825    5.1703
    0.0000   -0.9816   -0.7995    1.7842    1.3567   -9.0789
    0.0000    0.7196   -0.4326    0.0122    0.1566   -0.6760
    0.0000    0.0000    0.0000    2.0000    0.9849   -6.9838
    0.0000    0.0000    0.0000    0.0000    0.2629    3.0621
    0.0000    0.0000    0.0000    0.0000   -1.0860    2.5655

S =-1.0000    0.3196   -0.0540    0.6959    2.4405   -4.6789
    0.0000   -0.9816   -0.7995   -1.2635   -4.2890    8.2133
    0.0000    0.7196   -0.4326    0.0380   -0.3081    0.6207
    0.0000    0.0000    0.0000    1.5307    3.6968   -6.7809
    0.0000    0.0000    0.0000   -0.5447    1.2978   -1.7261
    0.0000    0.0000    0.0000    0.0000    0.0000    2.0000

Q = 0.4082   -0.7070   -0.0839    0.5465    0.1226   -0.1128
   -0.4082    0.3257   -0.3570    0.7349   -0.2398    0.0467
    0.4082    0.1566    0.4848    0.1480   -0.7414    0.0467
   -0.4082   -0.4572   -0.2248   -0.2862   -0.5527   -0.4318
    0.4082    0.4001   -0.2709   -0.0063    0.0824   -0.7701

   -0.4082   -0.0188    0.7118    0.2395    0.2560   -0.4511
```