THE UNIVERSITY OF READING

A Collection of MATLAB Subroutines for the Implementation of Damped Algebraic Riccati Equations

by

J.J. Hench

Numerical Analysis Report 7/97

DEPARTMENT OF MATHEMATICS

A Collection of MATLAB Subroutines for the Implementation of Damped Algebraic Riccati Equations*

J.J. Hench †

† Department of Mathematics
University of Reading
Whiteknights
Reading, Berkshire
RG6 2AX

May 1, 1997

Abstract

In this technical report, MATLAB subroutines are provided which implement various damped Riccati algorithms for standard linear systems. These subroutines can be used to synthesize dampening controllers with a degree of stability or to place closed-loop eigenvalues in a convex region in the complex plane.

Keywords: dampening feedback, damped system dynamics, damped algebraic Riccati equations, periodic Schur decomposition, periodic Hamiltonian systems, linear quadratic control.

Introduction

In this technical report, MATLAB subroutines are supplied which implement the algorithms in [1, 3, 2]. In all of these subroutines, it is assumed that

*Support for this research was provided by the Engineering and Physical Sciences Research Council of the United Kingdom, grant reference number GR/K 83618

[2] Hench, J.J., "Regional Pole Placement using Riccati Techniques," In preparation.

[3] Hench, J.J., C-Y. He, V. Kučera, and V. Mehrmann, "Dampening Controllers via a Riccati Equation Approach," Technical Report 1835, Institute of Information and Control Theory, Academy of Sciences of the Czech Republic, Pod vodárenskou věži 4, Praha 8, Czech Republic, May 1995, *To appear in IEEE Trans. Auto. Control.*

```
%

doneflag=0;

while doneflag == 0

G=J;
H=[kron(G,A)+ymax*kron(G*J,II),kron(I,-R);
   kron(I,S),-kron(G',A')-ymax*kron(J'*G',II)];
[X,f] = signric(H);
W = star(G,X);
if (max(imag(eig(A)))<ymax)
doneflag=1;
end
A=A-R*W;
Z=Z+W;

end

%
%       Next, bound the real part
%

doneflag=0;

while doneflag == 0

H=[A-xmax*II,-R;-S,-A'+xmax*II];
[X,f] = signric(H);
A=A-R*X;
Z=Z+X;


H=[-A+xmin*II,-R;-S,A'-xmin*II];
[X,f] = signric(H);
A=A+R*X;
Z=Z-X;

if (max(real(eig(A)))<xmax)&(min(real(eig(A)))>xmin)
```

Subroutine: convcomb.m

```
function[Z]=convcomb(A,B,Z1,Z2,beta,verbose)
%
%       [Z]=convcomb(A,B,Z1,Z2,beta,verbose)
%
%       This function computes the matrix Z = beta Z1 + (1-beta) Z2.
%       If verbose = 1, then it will also plot the locus of the closed
%       loop eigenvalues for all beta [0,1].
%
%       The closed-loop eigenvalues for A-BB'Z, A-BB'Z1, and A-BB;Z2
%       are denoted by *,o,+, repectively in the plot.

%       (c) 1997 John Hench.  All rights reserved.

%
%       Set up some variables
%

S=zeros(size(A));
R=B*B';
EIG=[];
Z = beta*Z1 + (1-beta)*Z2;
kk=1+sqrt(-eps^3);

if verbose == 1
subplot(1,1,1)
hold off;
plot(kk*eig(A-R*Z1),'ro')
hold on
plot(kk*eig(A-R*Z2),'r+')
plot(kk*eig(A-R*Z),'w*')

for j=0:.025:1
ZJ= j*Z1+(1-j)*Z2;
plot(kk*eig(A-R*ZJ),'r.')
end

title('Locus of Closed-loop E-vals (o,+: End Positions)')
```

Subroutine: place.m

```
function[Z,X,Y]=place(A,B,theta,sigma,maxop,verbose)
%
%       [Z,X,Y]=place(A,B,theta,sigma,maxop,verbose)
%
%       This algorithm places poles via a Riccati tensor
%       formulation for a linear system with state
%       matrix A and input matrix B. The stable eigenvalues
%       of A-B*B'*Z are within an angle to the REAL
%       axis of theta (in radians), and are at least sigma
%       to the left of the imaginary axis.
%
%       The variable maxop limits the amount of operations
%       (folds and pushes) which, if unlimited, make this
%       algorithm O(n^4).  For the maximum amount of operations,
%       set maxop to 0 (or n). For the minimum amount, set
%       maxop to 1.
%
%       The variable verbose allows you to see what this
%       algorithm is doing. Following convention, verbose != 1
%       is quiet.
%
%       The program also plots the closed-loop eigenvalues
%       as they progress the the Pole Placement via Riccati
%       (PPR) algorithm, and compares the result with the
%       Damped Riccati Algorithm (DRA). Note that the comparison
%       is only valid with theta = pi/4. (X is the symmetric
%       and Y is the non-symmetric output of the DRA)
%
%       Finally, a note: By replacing A with A+rho*I, the
%       whole problem can be shifted to the left by rho.
%       Also, these controllers may be used in convex combination
%       with controllers from other Riccati based algorithms.


%       Subroutines needed: rotate.m, signric.m, star.m
%       (c) 1996 John Hench  All rights reserved.
```

```
figure(1)
subplot(2,1,1)
title('Progression of E-vals in PPR algorithm')
mx=1.2*(max(max(abs(kk*eig(A))))+sigma);
plot([-sigma,-mx*cos(t)],[ sigma*tan(t), mx*sin(t)   ],'w:')
hold on
plot([-sigma,-mx*cos(t)],[-sigma*tan(t),-mx*sin(t)   ],'w:')
plot([-sigma,-sigma    ],[-sigma*tan(t), sigma*tan(t)],'w:')

plot(kk*eig(A),'wo')
disp(' ')
disp('White o_s show where almost open--loop matrices are.')
disp('(The system has already been stabilized by a deg. ARE)')
disp(' ')
disp('Blue dots show where the e-vals will end up ')
disp('after the algorithm is finished, hopefully.')
disp('<<PAUSE>>'),pause
clc

end


%
%       Now, move all e-vals so that they are at least sigma
%       away from the imaginary axis by the same method
%
%       First, create a list of real values of the e-vals,
%       not counting twice for complex conjugate pairs
%

ea = eig(A);
eap=[];
for j=1:n
if imag(ea(j))>0
else
eap=[eap,ea(j)];
end
end
reals = sort(real(-eap));
```

```
for j = 1:min(push_count,maxop)

[X,f]= signric([(A+pushes(j)*I),-R;S,-(A+pushes(j)*I)']);
A=A-R*X;
Z=Z+X;
%
%       Plot a few things
%
if verbose == 1
Eigs = [Eigs,kk*eig(A)];
plot(Eigs,'o')
disp(' ')
disp('Pause to show how e-vals line up vertically.')
disp(' ')
disp('Circles show where the eigenvalues are or have been.')
disp(' ')
disp('Colors denote which iteration as PPR progresses.')
disp('Yellow, first iteration, and going forwards')
disp('with Magenta, Cyan, Red, Green, Blue, and Yellow again.')
disp('<<PAUSE>>'),pause
clc
end
end
%
%       Now, create a list of phases of the e-vals
%

phases = -sort(angle(-eig(A)));

%
%       Angles of pole placement are half way between
%       each of the phases up to theta...
%
%       Count how many folds are needed in the complex
%       rotate subroutine.
%
```

```
for j = 1:min(fold_count,maxop)

X=rotate(A,B,angles(j));
A=A-R*X;
Z=Z+X;
%
%       Plot a few things
%
if verbose ==1
Eigs = [Eigs,kk*eig(A)];
plot(Eigs,'o')
disp(' ')
disp('Pause to show how e-vals line up anglewise.')
disp(' ')
disp('Circles show where the eigenvalues are or have been.')
disp(' ')
disp('Colors denote which iteration.')
disp('Yellow, first iteration, and going forwards')
disp('with Magenta, Cyan, Red, Green, Blue, and Yellow again.')
disp('<<PAUSE>>'),pause
clc
end
end

%
%       One last stabilization with shift in in case of
%       sensitive e-vals (A problem if theta < pi/4)
%

HL    = [A0+sigma*I-R*Z,-R;-S,-(A0+sigma*I-R*Z)'];
[X,f] = signric(HL);
Z     = Z+X;
A     = A0-R*Z;

if verbose == 1
plot(kk*eig(A),'w+')
disp('White +_s denote closed-loop e-vals')
disp('produced by this algorithm')
```

```
if verbose == 1
subplot(2,1,1)
title('Evolution of Closed-loop E-vals (+: Final Position)')
axisval=axis;
subplot(2,1,2)
hold off
clc
disp(' ')
disp('Now compare with Damped Riccati equation (theta=pi/4)')
disp(' ')
plot(kk*eig(A0-R*Z),'w+')
disp('White +_s denote closed-loop e-vals from PPR algorithm.')
disp('(Pole Placement via Riccati Tensors)')
disp(' ')
axis(axisval)
axis(axis)
hold on
plot(kk*[eps,eps],'k.')
mx=1.2*(max(max(abs(kk*eig(A0-R*Z))))+sigma);
plot([-sigma,-mx*cos(t)],[ sigma*tan(t), mx*sin(t)   ],'w:')
plot([-sigma,-mx*cos(t)],[-sigma*tan(t),-mx*sin(t)   ],'w:')
plot([-sigma,-sigma    ],[-sigma*tan(t), sigma*tan(t)],'w:')
eigs_x=eig(A0-R*X);
plot(kk*eigs_x,'cx')
plot(kk*eig(A0-R*Y),'mo')
if min(real(eigs_x)) < axisval(1)
text(.9*axisval(1),.8*axisval(4),'<-- Some DRA evals are off-plot.')
end
disp('Cyan x_s denote closed-loop e-vals from DRA (sym soln)')
disp('Magenta o_s denote closed-loop e-vals from DRA (non-sym soln)')
disp(' ')
disp('Now compare norm of feeback solutions from')
disp('Pole Placement Via Riccati Algorithm (Z) and')
disp('Damped Riccati Algorithm (X), with the norm')
disp('being || B^T*Z ||_fro and || B^T*X ||_fro,')
disp('and || B^T*Y ||_fro respectively.')
disp(' ')
norm_PPR = norm(B'*Z,'fro')
norm_DRA_X     = norm(B'*X,'fro')
```

Subroutine: place2.m

```
function[Z,X,Y]=place2(A,B,theta,sigma,maxop,verbose)
%
%       [Z,X,Y]=place2(A,B,theta,sigma,maxop,verbose)
%
%       This algorithm places poles via a Riccati tensor
%       formulation for a linear system with state
%       matrix A and input matrix B. The stable eigenvalues
%       of A-B*B'*Z are within an angle to the REAL
%       axis of theta (in radians), and are at least sigma
%       to the left of the imaginary axis.
%
%       The variable maxop limits the amount of operations
%       (folds and pushes) which, if unlimited, make this
%       algorithm O(n^4).   For the maximum amount of operations,
%       set maxop to 0 (or n). For the minimum amount, set
%       maxop to 1.
%
%       The variable verbose allows you to see what this
%       algorithm is doing. Following convention, verbose != 1
%       is quiet.
%
%       The program also plots the closed-loop eigenvalues
%       as they progress the the Pole Placement via Riccati
%       (PPR) algorithm, and compares the result with the
%       Damped Riccati Algorithm (DRA). Note that the comparison
%       is only valid with theta = pi/4. (X is the symmetric
%       and Y is the non-symmetric output of the DRA)
%
%       This program differs from place in that it dampens
%       first, then shifts second.
%
%       Finally, a note: By replacing A with A+rho*I, the
%       whole problem can be shifted to the left by rho.
%       Also, these controllers may be used in convex combination
%       with controllers from other Riccati based algorithms.
```

```
if verbose == 1

figure(1)
subplot(2,1,1)
title('Progression of E-vals in PPR algorithm')
mx=1.2*(max(max(abs(kk*eig(A))))+sigma);
plot([-sigma,-mx*cos(t)],[ sigma*tan(t), mx*sin(t)   ],'w:')
hold on
plot([-sigma,-mx*cos(t)],[-sigma*tan(t),-mx*sin(t)   ],'w:')
plot([-sigma,-sigma    ],[-sigma*tan(t), sigma*tan(t)],'w:')

plot(kk*eig(A),'wo')
disp(' ')
disp('White o_s show where almost open--loop matrices are.')
disp('(The system has already been stabilized by a deg. ARE)')
disp(' ')
disp('Blue dots show where the e-vals will end up ')
disp('after the algorithm is finished, hopefully.')
disp('<<PAUSE>>'),pause
clc

end

%
%       Now, create a list of phases of the e-vals
%

phases = -sort(angle(-eig(A)));

%
%       Angles of pole placement are half way between
%       each of the phases up to theta...
%
%       Count how many folds are needed in the complex
%       rotate subroutine.
%

if max(phases)>0
```

```
for j = 1:min(fold_count,maxop)

X=rotate(A,B,angles(j));
A=A-R*X;
Z=Z+X;
%
%       Plot a few things
%

if verbose == 1
Eigs = [Eigs,kk*eig(A)];
plot(Eigs,'o')
disp(' ')
disp('Pause to show how e-vals line up anglewise.')
disp(' ')
disp('Circles show where the eigenvalues are or have been.')
disp(' ')
disp('Colors denote which iteration.')
disp('Yellow, first iteration, and going forwards')
disp('with Magenta, Cyan, Red, Green, Blue, and Yellow again.')
disp('<<PAUSE>>'),pause
clc
end
end

%
%       Now, move all e-vals so that they are at least sigma
%       away from the imaginary axis by the same method
%
%       First, create a list of real values of the e-vals,
%       not counting twice for complex conjugate pairs
%

ea = eig(A);
eap=[];
for j=1:n
if imag(ea(j))>0
else
```

```
%
%       Execute each push
%


for j = 1:min(push_count,maxop)

[X,f]= signric([(A+pushes(j)*I),-R;S,-(A+pushes(j)*I)']);
A=A-R*X;
Z=Z+X;
%
%       Plot a few things
%
if verbose ==1

Eigs = [Eigs,kk*eig(A)];
plot(Eigs,'o')
disp(' ')
disp('Pause to show how e-vals line up vertically.')
disp(' ')
disp('Circles show where the eigenvalues are or have been.')
disp(' ')
disp('Colors denote which iteration as PPR progresses.')
disp('Yellow, first iteration, and going forwards')
disp('with Magenta, Cyan, Red, Green, Blue, and Yellow again.')
disp('<<PAUSE>>'),pause
clc
end
end

%
%       One last stabilization with shift in in case of
%       sensitive e-vals (A problem if theta < pi/4)
%

HL    = [A0+sigma*I-R*Z,-R;-S,-(A0+sigma*I-R*Z)'];
[X,f] = signric(HL);
Z     = Z+X;
```

```
A2Y=A1-R*Y1;
H2Y=[A2Y+sigma*I, -R;-S,-(A2Y+sigma*I)'];
[Y2,f]=signric(H2Y);

X=X0+X1+X2;
Y=X0+Y1+Y2;

if verbose == 1
subplot(2,1,1)
title('Evolution of Closed-loop E-vals (+: Final Position)')
axisval=axis;
subplot(2,1,2)
hold off
clc
disp(' ')
disp('Now compare with Damped Riccati equation (theta=pi/4)')
disp(' ')
plot(kk*eig(A0-R*Z),'w+')
disp('White +_s denote closed-loop e-vals from PPR algorithm.')
disp('(Pole Placement via Riccati Tensors)')
disp(' ')
axis(axisval)
axis(axis)
hold on
plot(kk*[eps,eps],'k.')
mx=1.2*(max(max(abs(kk*eig(A0-R*Z))))+sigma);
plot([-sigma,-mx*cos(t)],[ sigma*tan(t), mx*sin(t)   ],'w:')
plot([-sigma,-mx*cos(t)],[-sigma*tan(t),-mx*sin(t)   ],'w:')
plot([-sigma,-sigma    ],[-sigma*tan(t), sigma*tan(t)],'w:')
eigs_x=eig(A0-R*X);
plot(kk*eigs_x,'cx')
plot(kk*eig(A0-R*Y),'mo')
if min(real(eigs_x)) < axisval(1)
text(.9*axisval(1),.8*axisval(4),'<-- Some DRA evals are off-plot.')
end
disp('Cyan x_s denote closed-loop e-vals from DRA (sym soln)')
disp('Magenta o_s denote closed-loop e-vals from DRA (non-sym soln)')
disp(' ')
```

Subroutine: rotate.m

```
function[Z]=rotate(A,B,theta)
%
%        [Z]=rotate(A,B,theta)
%
%       This routine performes the degenerate rotated
%       Riccati equation using the real tensor form.
%       The matrix A is the state matrix, the
%       matrix B is the input matrix, and the index
%       of rotation is theta.

%       (c) 1996 John J. Hench  All rights reserved.

n=max(size(A));
m=n+1;
p=2*n;

R=B*B';
S=zeros(n);
I=eye(2);

t=theta;
G=[cos(t),-sin(t);sin(t),cos(t)];
H=[kron(G,A),kron(I,-R);kron(I,S),kron(G',-A')];
[X,f] = signric(H);
Z = star(G,X);
```

Subroutine: star.m

```
function[C]=star(A,B)
%
%       C = star(A,B)
%
%       This routine computes the MacRae star product (v.
%       "Matrix Derivatives" by Gerald Rogers).  It more
%       or less is a tensor contraction: it reduces a mp by
%       nq matrix B to a p by q matrix C by means of summing
%       all of the i,jth m by n blocks of C with the coefficient
%       of summation taken from the i,jth element of A.

%       (c) 1996 John J. Hench  All rights reserved.

[m,n]   = size(A);
[mp,nq] = size(B);

if (rem(mp,m)~=0)|(rem(nq,n)~=0)
error('A and B are not *-conformable')
end

p = mp/m;
q = nq/n;

C = zeros(p,q);

for j=1:m
for k=1:n
C = C+A(j,k)*B( (j-1)*p+1:j*p , (k-1)*q+1:k*q );
end
end
```

A Collection of MATLAB Subroutines for the Implementation of Damped Algebraic Riccati Equations*

J.J. Hench †

† Department of Mathematics
University of Reading
Whiteknights
Reading, Berkshire
RG6 2AX

May 1, 1997

Abstract

In this technical report, MATLAB subroutines are provided which implement various damped Riccati algorithms for standard linear systems. These subroutines can be used to synthesize dampening controllers with a degree of stability or to place closed-loop eigenvalues in a convex region in the complex plane.

Keywords: dampening feedback, damped system dynamics, damped algebraic Riccati equations, periodic Schur decomposition, periodic Hamiltonian systems, linear quadratic control.

Introduction

In this technical report, MATLAB subroutines are supplied which implement the algorithms in [1, 3, 2]. In all of these subroutines, it is assumed that

*Support for this research was provided by the Engineering and Physical Sciences Research Council of the United Kingdom, grant reference number GR/K 83618

the matrices A, B, and C are taken from a standard linear time–invariant system of the form

$$\begin{array}{rcl} \dot{x} & = & Ax + Bu \\ y & = & Cx \end{array} \tag{1}$$

In these routines, a number of damped algebraic Riccati equations are solved, including the stabilizing solution to the symmetric Damped Algebraic Riccati Equation (SDARE)

$$X(A^2 - RS) + (A^{2T} - SR)X - X(AR + RA^T)X + (A^T S + SA) = 0, \tag{2}$$

the Skew-Symmetric Damped Algebraic Riccati Equation (SSDARE)

$$Y(A^2 + RS) + (A^{2T} + SR)Y - Y(AR - RA^T)Y + (A^T S - SA) = 0, \tag{3}$$

and the real–valued Tensor Algebraic Riccati Equation (TARE):

$$\mathcal{A}^T \mathcal{X} + \mathcal{X}\mathcal{A} - \mathcal{X}\mathcal{R}\mathcal{X} = 0. \tag{4}$$

The additional matrices above are defined as $\mathcal{R} = I \otimes R$, $J = G(\pi/2)$, and $\mathcal{A} = (G \otimes II)\,(I \otimes A + (\varsigma I + \omega J) \otimes II)$, where

$$G(\phi) = \begin{bmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{bmatrix} = \begin{bmatrix} \mathrm{Re}(\alpha) & \mathrm{Im}(\alpha) \\ -\mathrm{Im}(\alpha) & \mathrm{Re}(\alpha) \end{bmatrix}, \tag{5}$$

and $\alpha = e^{\imath\phi}$ for $\phi \in [0, \pi/2]$. In order to avoid confusion between the 2×2 identity $I = G(0)$ and the $n \times n$ identity, the latter is written as "double I" II.

In the appendix, there are the listings to seven MATLAB .m files: `box.m`, `place2.m`, `signric.m`, `convcomb.m`, `place.m`, `rotate.m`, `star.m`.

These subroutines are highly commented, with an explanation of their functionality at the beginning of the file; therefore, we refrain from describing them again here.

References

[1] He, C-Y., J.J. Hench, and V. Mehrmann, "A Note on Damped Algebraic Riccati Equations," University of Reading, Department of Mathematics Technical Report NA 5/96, 1996, *to appear in IEEE Trans. Auto. Control.*

[2] Hench, J.J., "Regional Pole Placement using Riccati Techniques," In preparation.

[3] Hench, J.J., C-Y. He, V. Kučera, and V. Mehrmann, "Dampening Controllers via a Riccati Equation Approach," Technical Report 1835, Institute of Information and Control Theory, Academy of Sciences of the Czech Republic, Pod vodárenskou věži 4, Praha 8, Czech Republic, May 1995, *To appear in IEEE Trans. Auto. Control.*

Appendix

The MATLAB subroutines listings mentioned above are provided here.

Subroutine: box.m

```
function[Z]=box(A,B,xmin,xmax,ymin,ymax)
%
%       [Z]=box(A,B,xmin,xmax,ymin,ymax)
%
%       This function restricts the closed-loop eigenvalues
%       of A-B*B'*Z to be within a rectangle in the complex
%       plane with vertices [xmin,ymin], [xmin,ymax],
%       [xmax,ymin], and [xmax,ymax].  The algorithm uses
%       a Tensor Riccati formulation.

%       (c) 1997 John Hench.  All rights reserved.

%
%       Set up some variables
%

S=zeros(size(A));
R=B*B';
II=eye(size(A));
I=eye(2);
J=[0,1;-1,0];
Z=zeros(size(A));

%
%       Give the obvious error message
%

if (ymin>0)|(ymax<0)|(ymin~=-ymax)
error('For complex conjugacy, ymin = -ymax. Also, ymax > 0.')
end

%
%       First bound the imaginary part
```

```
%

doneflag=0;

while doneflag == 0

G=J;
H=[kron(G,A)+ymax*kron(G*J,II),kron(I,-R);
   kron(I,S),-kron(G',A')-ymax*kron(J'*G',II)];
[X,f] = signric(H);
W = star(G,X);
if (max(imag(eig(A)))<ymax)
doneflag=1;
end
A=A-R*W;
Z=Z+W;

end

%
%       Next, bound the real part
%

doneflag=0;

while doneflag == 0

H=[A-xmax*II,-R;-S,-A'+xmax*II];
[X,f] = signric(H);
A=A-R*X;
Z=Z+X;


H=[-A+xmin*II,-R;-S,A'-xmin*II];
[X,f] = signric(H);
A=A+R*X;
Z=Z-X;

if (max(real(eig(A)))<xmax)&(min(real(eig(A)))>xmin)
```

```
doneflag=1;
end
end
```

Subroutine: convcomb.m

```
function[Z]=convcomb(A,B,Z1,Z2,beta,verbose)
%
%       [Z]=convcomb(A,B,Z1,Z2,beta,verbose)
%
%       This function computes the matrix Z = beta Z1 + (1-beta) Z2.
%       If verbose = 1, then it will also plot the locus of the closed
%       loop eigenvalues for all beta [0,1].
%
%       The closed-loop eigenvalues for A-BB'Z, A-BB'Z1, and A-BB;Z2
%       are denoted by *,o,+, repectively in the plot.

%       (c) 1997 John Hench.  All rights reserved.

%
%       Set up some variables
%

S=zeros(size(A));
R=B*B';
EIG=[];
Z = beta*Z1 + (1-beta)*Z2;
kk=1+sqrt(-eps^3);

if verbose == 1
subplot(1,1,1)
hold off;
plot(kk*eig(A-R*Z1),'ro')
hold on
plot(kk*eig(A-R*Z2),'r+')
plot(kk*eig(A-R*Z),'w*')

for j=0:.025:1
ZJ= j*Z1+(1-j)*Z2;
plot(kk*eig(A-R*ZJ),'r.')
end

title('Locus of Closed-loop E-vals (o,+: End Positions)')
```

```
hold off
end
```

Subroutine: place.m

```
function[Z,X,Y]=place(A,B,theta,sigma,maxop,verbose)
%
%       [Z,X,Y]=place(A,B,theta,sigma,maxop,verbose)
%
%       This algorithm places poles via a Riccati tensor
%       formulation for a linear system with state
%       matrix A and input matrix B. The stable eigenvalues
%       of A-B*B'*Z are within an angle to the REAL
%       axis of theta (in radians), and are at least sigma
%       to the left of the imaginary axis.
%
%       The variable maxop limits the amount of operations
%       (folds and pushes) which, if unlimited, make this
%       algorithm O(n^4).  For the maximum amount of operations,
%       set maxop to 0 (or n). For the minimum amount, set
%       maxop to 1.
%
%       The variable verbose allows you to see what this
%       algorithm is doing. Following convention, verbose != 1
%       is quiet.
%
%       The program also plots the closed-loop eigenvalues
%       as they progress the the Pole Placement via Riccati
%       (PPR) algorithm, and compares the result with the
%       Damped Riccati Algorithm (DRA). Note that the comparison
%       is only valid with theta = pi/4. (X is the symmetric
%       and Y is the non-symmetric output of the DRA)
%
%       Finally, a note: By replacing A with A+rho*I, the
%       whole problem can be shifted to the left by rho.
%       Also, these controllers may be used in convex combination
%       with controllers from other Riccati based algorithms.


%       Subroutines needed: rotate.m, signric.m, star.m
%       (c) 1996 John Hench  All rights reserved.
```

```
n=max(size(A));
m=n+1;
p=2*n;

A0=A;
R=B*B';
S=zeros(n);
Z=zeros(n);
I=eye(n);

t=theta;
s=sigma;

hold off
kk= 1+sqrt(-1)*eps;

Eigs=[];
clc

if maxop == 0
maxop = n;
end

%
%       First, stablize A with a degenerate ARE
%

H = [A,-R;-S,-A'];
[X,f] = signric(H);

A = A-R*X;
Z = Z+X;

%
%       This is for display purposes (plots)
%

if verbose == 1
```

```
figure(1)
subplot(2,1,1)
title('Progression of E-vals in PPR algorithm')
mx=1.2*(max(max(abs(kk*eig(A))))+sigma);
plot([-sigma,-mx*cos(t)],[ sigma*tan(t), mx*sin(t)   ],'w:')
hold on
plot([-sigma,-mx*cos(t)],[-sigma*tan(t),-mx*sin(t)   ],'w:')
plot([-sigma,-sigma    ],[-sigma*tan(t), sigma*tan(t)],'w:')

plot(kk*eig(A),'wo')
disp(' ')
disp('White o_s show where almost open--loop matrices are.')
disp('(The system has already been stabilized by a deg. ARE)')
disp(' ')
disp('Blue dots show where the e-vals will end up ')
disp('after the algorithm is finished, hopefully.')
disp('<<PAUSE>>'),pause
clc

end


%
%       Now, move all e-vals so that they are at least sigma
%       away from the imaginary axis by the same method
%
%       First, create a list of real values of the e-vals,
%       not counting twice for complex conjugate pairs
%

ea = eig(A);
eap=[];
for j=1:n
if imag(ea(j))>0
else
eap=[eap,ea(j)];
end
end
reals = sort(real(-eap));
```

```
%
%       Pushes are at the real values of the e-vals
%
%       Count how many pushes are needed.
%

if min(reals) < sigma
push_count=0;
for j=1:max(size(reals))
if reals(j) < sigma
push_count=push_count+1;
end
end
end

reals=reals(1:push_count);
reals=[reals,sigma];

%
%       Pushes go between the real vales of the eigenvalues to be pushed
%


for j=1:push_count
pushes(j) = (reals(j)+reals(j+1))/2;
end

%
%       Limit amounts of pushes to maxop
%

if push_count > maxop
pushes = pushes(push_count-maxop+1:push_count);
end

%
%       Execute each push
%
```

```
for j = 1:min(push_count,maxop)

[X,f]= signric([(A+pushes(j)*I),-R;S,-(A+pushes(j)*I)']);
A=A-R*X;
Z=Z+X;
%
%       Plot a few things
%
if verbose == 1
Eigs = [Eigs,kk*eig(A)];
plot(Eigs,'o')
disp(' ')
disp('Pause to show how e-vals line up vertically.')
disp(' ')
disp('Circles show where the eigenvalues are or have been.')
disp(' ')
disp('Colors denote which iteration as PPR progresses.')
disp('Yellow, first iteration, and going forwards')
disp('with Magenta, Cyan, Red, Green, Blue, and Yellow again.')
disp('<<PAUSE>>'),pause
clc
end
end
%
%       Now, create a list of phases of the e-vals
%

phases = -sort(angle(-eig(A)));

%
%       Angles of pole placement are half way between
%       each of the phases up to theta...
%
%       Count how many folds are needed in the complex
%       rotate subroutine.
%
```

```
if max(phases)>0
fold_count=0;
for j=1:floor(n/2)
if phases(j) > theta
fold_count=fold_count+1;
end
end
end

%
%       Folds go at the between the phases of the eigenvalues to be moved
%

phases= phases(1:fold_count);
phases=[phases;theta];

for j=1:fold_count
folds(j) = (phases(j)+ phases(j+1))/2;
end

%
%       Include last fold at desired theta point,
%       and tranlate to angles for the rotate algorithm
%

angles = abs(folds - (pi/2)*ones(size(folds)));

%
%       Limit amounts of folds to maxop.
%

if fold_count > maxop
angles = angles(fold_count-maxop+1:fold_count);
end

%
%       Execute each fold
%
```

```
for j = 1:min(fold_count,maxop)

X=rotate(A,B,angles(j));
A=A-R*X;
Z=Z+X;
%
%       Plot a few things
%
if verbose ==1
Eigs = [Eigs,kk*eig(A)];
plot(Eigs,'o')
disp(' ')
disp('Pause to show how e-vals line up anglewise.')
disp(' ')
disp('Circles show where the eigenvalues are or have been.')
disp(' ')
disp('Colors denote which iteration.')
disp('Yellow, first iteration, and going forwards')
disp('with Magenta, Cyan, Red, Green, Blue, and Yellow again.')
disp('<<PAUSE>>'),pause
clc
end
end

%
%       One last stabilization with shift in in case of
%       sensitive e-vals (A problem if theta < pi/4)
%

HL    = [A0+sigma*I-R*Z,-R;-S,-(A0+sigma*I-R*Z)'];
[X,f] = signric(HL);
Z     = Z+X;
A     = A0-R*Z;

if verbose == 1
plot(kk*eig(A),'w+')
disp('White +_s denote closed-loop e-vals')
disp('produced by this algorithm')
```

```
hold off
end

%
%       Now, compare with Damped Riccati Algorithm
%
%       First stabilize system with small offset del
%

del = sqrt(eps);
H0 = [A0+del*I,-R;S,-(A0+del*I)'];
[X0,f]=signric(H0);

%
%       Next, Damped (with Symmetric Solution) system
%

A1=A0-R*X0;
H12=[A1,-R;S,A1'];
H22=[-A1,R;S,A1'];
[X1,f]=signric(H22*H12);
[Y1,f]=signric(H12*H22);

%
%       Stabilize again with stability margin sigma
%

A2X=A1-R*X1;
H2X=[A2X+sigma*I, -R;-S,-(A2X+sigma*I)'];
[X2,f]=signric(H2X);
A3=A2X-R*X2;

A2Y=A1-R*Y1;
H2Y=[A2Y+sigma*I, -R;-S,-(A2Y+sigma*I)'];
[Y2,f]=signric(H2Y);

X=X0+X1+X2;
Y=X0+Y1+Y2;
```

```
if verbose == 1
subplot(2,1,1)
title('Evolution of Closed-loop E-vals (+: Final Position)')
axisval=axis;
subplot(2,1,2)
hold off
clc
disp(' ')
disp('Now compare with Damped Riccati equation (theta=pi/4)')
disp(' ')
plot(kk*eig(A0-R*Z),'w+')
disp('White +_s denote closed-loop e-vals from PPR algorithm.')
disp('(Pole Placement via Riccati Tensors)')
disp(' ')
axis(axisval)
axis(axis)
hold on
plot(kk*[eps,eps],'k.')
mx=1.2*(max(max(abs(kk*eig(A0-R*Z))))+sigma);
plot([-sigma,-mx*cos(t)],[ sigma*tan(t), mx*sin(t)   ],'w:')
plot([-sigma,-mx*cos(t)],[-sigma*tan(t),-mx*sin(t)   ],'w:')
plot([-sigma,-sigma    ],[-sigma*tan(t), sigma*tan(t)],'w:')
eigs_x=eig(A0-R*X);
plot(kk*eigs_x,'cx')
plot(kk*eig(A0-R*Y),'mo')
if min(real(eigs_x)) < axisval(1)
text(.9*axisval(1),.8*axisval(4),'<-- Some DRA evals are off-plot.')
end
disp('Cyan x_s denote closed-loop e-vals from DRA (sym soln)')
disp('Magenta o_s denote closed-loop e-vals from DRA (non-sym soln)')
disp(' ')
disp('Now compare norm of feeback solutions from')
disp('Pole Placement Via Riccati Algorithm (Z) and')
disp('Damped Riccati Algorithm (X), with the norm')
disp('being || B^T*Z ||_fro and || B^T*X ||_fro,')
disp('and || B^T*Y ||_fro respectively.')
disp(' ')
norm_PPR = norm(B'*Z,'fro')
norm_DRA_X     = norm(B'*X,'fro')
```

```
norm_DRA_Y      = norm(B'*Y,'fro')
title('E-vals of PPR vs. DRA (X:x,Y:o,Z:+)')
end
```

Subroutine: place2.m

```
function[Z,X,Y]=place2(A,B,theta,sigma,maxop,verbose)
%
%       [Z,X,Y]=place2(A,B,theta,sigma,maxop,verbose)
%
%       This algorithm places poles via a Riccati tensor
%       formulation for a linear system with state
%       matrix A and input matrix B. The stable eigenvalues
%       of A-B*B'*Z are within an angle to the REAL
%       axis of theta (in radians), and are at least sigma
%       to the left of the imaginary axis.
%
%       The variable maxop limits the amount of operations
%       (folds and pushes) which, if unlimited, make this
%       algorithm O(n^4).   For the maximum amount of operations,
%       set maxop to 0 (or n). For the minimum amount, set
%       maxop to 1.
%
%       The variable verbose allows you to see what this
%       algorithm is doing. Following convention, verbose != 1
%       is quiet.
%
%       The program also plots the closed-loop eigenvalues
%       as they progress the the Pole Placement via Riccati
%       (PPR) algorithm, and compares the result with the
%       Damped Riccati Algorithm (DRA). Note that the comparison
%       is only valid with theta = pi/4. (X is the symmetric
%       and Y is the non-symmetric output of the DRA)
%
%       This program differs from place in that it dampens
%       first, then shifts second.
%
%       Finally, a note: By replacing A with A+rho*I, the
%       whole problem can be shifted to the left by rho.
%       Also, these controllers may be used in convex combination
%       with controllers from other Riccati based algorithms.
```

```
%       Subroutines needed: rotate.m, signric.m, star.m
%       (c) 1996 John Hench All rights reserved.

n=max(size(A));
m=n+1;
p=2*n;

A0=A;
R=B*B';
S=zeros(n);
Z=zeros(n);
I=eye(n);

t=theta;
s=sigma;

hold off
kk= 1+sqrt(-1)*eps;

Eigs=[];
clc

if maxop == 0
maxop = n;
end

%
%       First, stablize A with a degenerate ARE
%

H = [A,-R;-S,-A'];
[X,f] = signric(H);

A = A-R*X;
Z = Z+X;

%
%       This is for display purposes (plots)
%
```

```
if verbose == 1

figure(1)
subplot(2,1,1)
title('Progression of E-vals in PPR algorithm')
mx=1.2*(max(max(abs(kk*eig(A))))+sigma);
plot([-sigma,-mx*cos(t)],[ sigma*tan(t), mx*sin(t)   ],'w:')
hold on
plot([-sigma,-mx*cos(t)],[-sigma*tan(t),-mx*sin(t)   ],'w:')
plot([-sigma,-sigma    ],[-sigma*tan(t), sigma*tan(t)],'w:')

plot(kk*eig(A),'wo')
disp(' ')
disp('White o_s show where almost open--loop matrices are.')
disp('(The system has already been stabilized by a deg. ARE)')
disp(' ')
disp('Blue dots show where the e-vals will end up ')
disp('after the algorithm is finished, hopefully.')
disp('<<PAUSE>>'),pause
clc

end

%
%       Now, create a list of phases of the e-vals
%

phases = -sort(angle(-eig(A)));

%
%       Angles of pole placement are half way between
%       each of the phases up to theta...
%
%       Count how many folds are needed in the complex
%       rotate subroutine.
%

if max(phases)>0
```

```
fold_count=0;
for j=1:floor(n/2)
if phases(j) > theta
fold_count=fold_count+1;
end
end
end

%
%       Folds go at the between the phases of the eigenvalues to be moved
%

phases= phases(1:fold_count);
phases=[phases;theta];

for j=1:fold_count
folds(j) = (phases(j)+ phases(j+1))/2;
end


%
%       Include last fold at desired theta point,
%       and tranlate to angles for the rotate algorithm
%

angles = abs(folds - (pi/2)*ones(size(folds)));

%
%       Limit amounts of folds to maxop.
%

if fold_count > maxop
angles = angles(fold_count-maxop+1:fold_count);
end

%
%       Execute each fold
%
```

```
for j = 1:min(fold_count,maxop)

X=rotate(A,B,angles(j));
A=A-R*X;
Z=Z+X;
%
%       Plot a few things
%

if verbose == 1
Eigs = [Eigs,kk*eig(A)];
plot(Eigs,'o')
disp(' ')
disp('Pause to show how e-vals line up anglewise.')
disp(' ')
disp('Circles show where the eigenvalues are or have been.')
disp(' ')
disp('Colors denote which iteration.')
disp('Yellow, first iteration, and going forwards')
disp('with Magenta, Cyan, Red, Green, Blue, and Yellow again.')
disp('<<PAUSE>>'),pause
clc
end
end

%
%       Now, move all e-vals so that they are at least sigma
%       away from the imaginary axis by the same method
%
%       First, create a list of real values of the e-vals,
%       not counting twice for complex conjugate pairs
%

ea = eig(A);
eap=[];
for j=1:n
if imag(ea(j))>0
else
```

```
eap=[eap,ea(j)];
end
end
reals = sort(real(-eap));

%
%       Pushes are at the real values of the e-vals
%
%       Count how many pushes are needed.
%

if min(reals) < sigma
push_count=0;
for j=1:max(size(reals))
if reals(j) < sigma
push_count=push_count+1;
end
end
end

reals=reals(1:push_count);
reals=[reals,sigma];

%
%       Pushes go between the real vales of the eigenvalues to be pushed
%


for j=1:push_count
pushes(j) = (reals(j)+reals(j+1))/2;
end

%
%       Limit amounts of pushes to maxop
%

if push_count > maxop
pushes = pushes(push_count-maxop+1:push_count);
end
```

```
%
%       Execute each push
%


for j = 1:min(push_count,maxop)

[X,f]= signric([(A+pushes(j)*I),-R;S,-(A+pushes(j)*I)']);
A=A-R*X;
Z=Z+X;
%
%       Plot a few things
%
if verbose ==1

Eigs = [Eigs,kk*eig(A)];
plot(Eigs,'o')
disp(' ')
disp('Pause to show how e-vals line up vertically.')
disp(' ')
disp('Circles show where the eigenvalues are or have been.')
disp(' ')
disp('Colors denote which iteration as PPR progresses.')
disp('Yellow, first iteration, and going forwards')
disp('with Magenta, Cyan, Red, Green, Blue, and Yellow again.')
disp('<<PAUSE>>'),pause
clc
end
end

%
%       One last stabilization with shift in in case of
%       sensitive e-vals (A problem if theta < pi/4)
%

HL    = [A0+sigma*I-R*Z,-R;-S,-(A0+sigma*I-R*Z)'];
[X,f] = signric(HL);
Z     = Z+X;
```

```
A     = A0-R*Z;

if verbose ==1

plot(kk*eig(A0-R*Z),'w+')
disp('White +_s denote closed-loop e-vals')
disp('produced by this algorithm')
hold off

end

%
%       Now, compare with Damped Riccati Algorithm
%
%       First stabilize system with small offset del
%

del = sqrt(eps);
H0 = [A0+del*I,-R;S,-(A0+del*I)'];
[X0,f]=signric(H0);

%
%       Next, Damped (with Symmetric Solution) system
%

A1=A0-R*X0;
H12=[A1,-R;S,A1'];
H22=[-A1,R;S,A1'];
[X1,f]=signric(H22*H12);
[Y1,f]=signric(H12*H22);

%
%       Stabilize again with stability margin sigma
%

A2X=A1-R*X1;
H2X=[A2X+sigma*I, -R;-S,-(A2X+sigma*I)'];
[X2,f]=signric(H2X);
A3=A2X-R*X2;
```

```
A2Y=A1-R*Y1;
H2Y=[A2Y+sigma*I, -R;-S,-(A2Y+sigma*I)'];
[Y2,f]=signric(H2Y);

X=X0+X1+X2;
Y=X0+Y1+Y2;

if verbose == 1
subplot(2,1,1)
title('Evolution of Closed-loop E-vals (+: Final Position)')
axisval=axis;
subplot(2,1,2)
hold off
clc
disp(' ')
disp('Now compare with Damped Riccati equation (theta=pi/4)')
disp(' ')
plot(kk*eig(A0-R*Z),'w+')
disp('White +_s denote closed-loop e-vals from PPR algorithm.')
disp('(Pole Placement via Riccati Tensors)')
disp(' ')
axis(axisval)
axis(axis)
hold on
plot(kk*[eps,eps],'k.')
mx=1.2*(max(max(abs(kk*eig(A0-R*Z))))+sigma);
plot([-sigma,-mx*cos(t)],[ sigma*tan(t), mx*sin(t)   ],'w:')
plot([-sigma,-mx*cos(t)],[-sigma*tan(t),-mx*sin(t)   ],'w:')
plot([-sigma,-sigma    ],[-sigma*tan(t), sigma*tan(t)],'w:')
eigs_x=eig(A0-R*X);
plot(kk*eigs_x,'cx')
plot(kk*eig(A0-R*Y),'mo')
if min(real(eigs_x)) < axisval(1)
text(.9*axisval(1),.8*axisval(4),'<-- Some DRA evals are off-plot.')
end
disp('Cyan x_s denote closed-loop e-vals from DRA (sym soln)')
disp('Magenta o_s denote closed-loop e-vals from DRA (non-sym soln)')
disp(' ')
```

```
disp('Now compare norm of feeback solutions from')
disp('Pole Placement Via Riccati Algorithm (Z) and')
disp('Damped Riccati Algorithm (X), with the norm')
disp('being || B^T*Z ||_fro and || B^T*X ||_fro,')
disp('and || B^T*Y ||_fro respectively.')
disp(' ')
norm_PPR = norm(B'*Z,'fro')
norm_DRA_X     = norm(B'*X,'fro')
norm_DRA_Y     = norm(B'*Y,'fro')
title('E-vals of PPR vs. DRA (X:x,Y:o,Z:+)')
end
```

Subroutine: rotate.m

```
function[Z]=rotate(A,B,theta)
%
%        [Z]=rotate(A,B,theta)
%
%       This routine performes the degenerate rotated
%       Riccati equation using the real tensor form.
%       The matrix A is the state matrix, the
%       matrix B is the input matrix, and the index
%       of rotation is theta.

%       (c) 1996 John J. Hench  All rights reserved.

n=max(size(A));
m=n+1;
p=2*n;

R=B*B';
S=zeros(n);
I=eye(2);

t=theta;
G=[cos(t),-sin(t);sin(t),cos(t)];
H=[kron(G,A),kron(I,-R);kron(I,S),kron(G',-A')];
[X,f] = signric(H);
Z = star(G,X);
```

Subroutine: signric.m

```
function [S,f]=signric(H)
%
%        Linear quadratic regulator design for continuous-time systems.
%        [S,f] = signric(H)  calculates the Riccati solution to
%
%               0 = SA + A'S - SRS + Q
%
%        via the Matrix Sign Function.

%       (c) 1996 John J. Hench  All rights reserved.

n2 = max(size(H));
n=n2/2;
I=eye(n);
f=0;

stp=1;
Z=H;
rcn=rcond(Z);
count=0;
while (stp>eps)&(count<21)
c=exp(log(abs(det(Z)))/n2);
Znew=(Z+c^2*inv(Z))/(2*c);
stp=rcn*norm((Z-Znew),'fro');
count=count+1;
Z=Znew;
end

Z11=Z(1:n,1:n);
Z12=Z(1:n,n+1:n2);
Z21=Z(n+1:n2,1:n);
Z22=Z(n+1:n2,n+1:n2);
L=[Z12;Z22+I];
R=[Z11+I;Z21];
S=-L\R;
f=rcond(Z);
```

Subroutine: star.m

```
function[C]=star(A,B)
%
%       C = star(A,B)
%
%       This routine computes the MacRae star product (v.
%       "Matrix Derivatives" by Gerald Rogers).  It more
%       or less is a tensor contraction: it reduces a mp by
%       nq matrix B to a p by q matrix C by means of summing
%       all of the i,jth m by n blocks of C with the coefficient
%       of summation taken from the i,jth element of A.

%       (c) 1996 John J. Hench  All rights reserved.

[m,n]   = size(A);
[mp,nq] = size(B);

if (rem(mp,m)~=0)|(rem(nq,n)~=0)
error('A and B are not *-conformable')
end

p = mp/m;
q = nq/n;

C = zeros(p,q);

for j=1:m
for k=1:n
C = C+A(j,k)*B( (j-1)*p+1:j*p , (k-1)*q+1:k*q );
end
end
```