State Space Object and Functions

Manual and Tutorial

Contents

1 The stsp Class

$$\frac{dx}{dt} = \mathbf{A}x + \mathbf{B}u$$
$$y = \mathbf{C}x + \mathbf{D}u$$

The MATLAB© class stsp defines a linear system in state space form. It has associated functions for performing a number of common operations on a linear state space system. Each stsp object is created from the state space matrices of a linear system having the form:
x - a vector of system states of length n
A - a square (nxn) state matrix
u - a vector of inputs of length ni
B - the input matrix of size nxni
y - a vector of outputs of length no
C - output matrix of size noxn
D - feed forward matrix of size no by ni

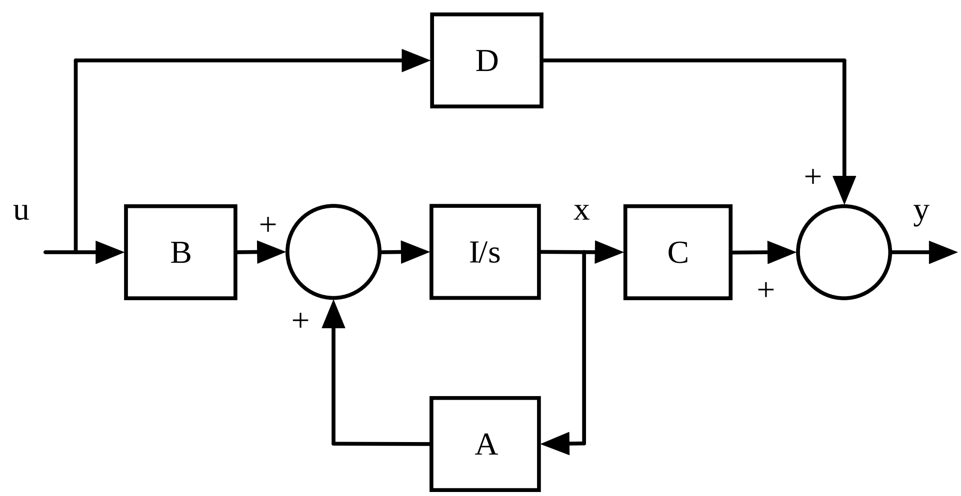


Figure 1 Linear System Model

A state space object is formed using the state space constructor

```
s = stsp(A,B,C,D);
```

s is a state space object with the following structure
s.a, s.b, s.c, s.d, s.NumStates, s.NumInputs, s.NumOutputs, where
s.a = A; s.b = B;, s.c = C and s.d = D.

Warnings are given if the dimensions of A,B,C,D are not consistent.

The following are special forms of state space objects:

> `s = stsp` forms an empty state space object, i.e., all structure fields are []
> `s = stsp([],[],[],d)` forms a state space system with empty a,b and c fields
> `s1 = stsp(s)` sets `s1=s` if s is a state space object: Note: the command `s1 = s` produces the same result

The matrices A,B,C and D may be either all full or all sparse.

The MATLAB **get** function is overloaded for use with state space objects, i.e.,
`sv = get(s)`, will give a structure having the same form as s, i.e.
sv.a, sv.b, sv.c, sv.d, sv.NumStates, sv.NumInputs, sv.NumOutputs

Typing s invokes the class' overloaded **display** function, which calls **get** and writes the structure (sv) on the command screen

Functions **subsref** and **subsasgn** are overloaded to allow access to the class fields outside the class directory. For example

```
s.d = [];
```

calls the subsasgn function and sets s.d to an empty matrix.

```
a=s.a;
```

calls the subsref function and sets a to the class state matriix.

1.1 The stsp Directory

The state space class constructor and functions which operate on state space objects are included in the directory named @stsp. The directory contains the following functions:

stsp	constructor for state space object
add_in	adds inputs to a state space object
add_out	adds outputs to a state space object
combine_in	sums selected inputs and retains selected inputs
cpc_stsp	robust control design based on co-prime uncertainty
display	displays the fields of a state space object
dompole	computes the dominant poles
dstate	determines the rate of change of states for the state space object's states non-windup limited, output limited
eig	eigenvalues of a state space object, overrides eig
eq	logical comparison s1==s2 ; 1 if true, 0 if false
eval	evaluates a state space object, overrides eval
exres	calculation of time response to a random input
fb_aug	connects two state space objects in feed back: feedback inputs and outputs may be specified; retained inputs and outputs may be specified
fr_stsp	frequency response of single input single output state space object
fr_mstsp	frequency response of multivariable state space object

full	converts a,b,c,d of a state space object to full matrix storage
get	creates a structure with the stsp fields
init_stsp	finds the initial input vector and the initial states to satisfy an initial output y
inv	finds the inverse of a state space object, if one exists
minus	subtracts two consistent state space objects sm = s1-s2
nlsr	nonlinear response to step, ramp or impulse
norm_copr	forms the normalized co-prime factors of a state space object
order_in	selects specified inputs and forms a modified stsp object
order_out	selects specified outputs and forms a new state space object
order_state	reorders the states of a stsp object
parallel	combines two state space objects with same number of inputs and outputs into one with the input applied equally to both subsystems and the output being the sum of the outputs from the two subsystems
parin	parallels the inputs of two stsp objects
plus	adds two state space objects sp = s1+s2 may be augmented
power	sp = s.^n n may be positive, negative or zero
reduss	balanced residual reduction of an unstable system
residue	calculates the residues of a state space object
response	calulates response to a user defined function of time
rtlocus	calculates the root locus of two state space objects connected in feedback
sdecomp	decomposes a system into the sum of two systems,used by reduss
sparse	converts a,b,c and d matrices to sparse storeage
stabred	balanced residual reduction for stable state space object
sresid	forms a reduced order system by eliminating n-ord states by assuming that their rates of change are zero
stepres	step response
sparse	converts a,b,c and d matrices to sparse storeage
stabred	balanced residual reduction for stable state space object
sresid	forms a reduced order system by eliminating n-ord states by assuming that their rates of change are zero
ss2tf	converts state space object to a trasfer function object
stack	combines a number of stsp objects into an equivalent single stsp oblect
stepres	calculates the response to a step input
stsp2ss	coverts state space object to control system toolbox object
sum_out	sums selected outputs and retains selected outputs

sysbal	performs balanced reduction which retains modes with Hankel singular values greater than a specified tolerance
times	multiples two state space objects st=s1.*s2 may be augmented, i.e., st = times(s1,s2,n_c,o_idx,i_idx)
tr_stsp	calculates the response of a state space object to a user defined time function
transpose	transpose of a state space object
uiminus	changes the sign of the output of a state space object
verstcat	combines state space objects into a single stacked state space object
zeros	calculates the transmission zeros of a full state space system

1.2 The statesp directory

This contains the @stsp directory, which defines the class and its functions. Also within the statesp directory are a number of functions that form state space objects corresponding to useful transfer functions and to useful actions in state space analysis.

bode_lab	labels axes of bode diagram
cll_stsp	forms complex lead lag state space object
dif_stsp	forms an imperfect differentiator state space object
dr_plot	plots a line which indicates a particular damping ratio on an Argand diagram
int_stsp	forms an integrator state space object
labxyarg	labels Argand diagram axes
lag_stsp	forms a lag state space object
ldlg_stsp	forms a lead/lag state space object
pid_stsp	forms a proportional + integral + derivative state space object
plot_bode	plots a bode diagram given frequency and response vectors
ric_eig	Performs matrix recatti reduction using eigenvalues
ric_schr	Performs matrix recatti reduction using schur decomposition
wo_stsp	forms a washout state space object

The statesp directory must be added to the MATLAB© path to allow MATLAB© access to the stsp object and its functions.

2 Examples of Use

2.1 Power System Stabilizer State Space Object

Power system stabilizers commonly have transfer functions of the form

$$P(s) = Ks\frac{sT_w}{1+sT_w}\frac{(1+sT_1)(1+sT_2)}{(1+sT_3)(1+sT_4)}$$

A state space object representing this transfer function may be formed by multiplying state space objects for a scalar, a washout and two lead/lags, i.e.,

```
sp = Ks.*wo_stsp(Tw).*ldlg_stsp(1,T3,T1).*ldlg_stsp(1,T4,T2);
```

For T_w = 10s; T_1 = 0.1s; T_3 = 0.02s; T_4 = 0.02s, T_2 = 0.08s and K_s = 10,

```
sp = 10.*wo_stsp(10).*ldlg_stsp(1,.02,.1).*ldlg_stsp(1,.02,.08)
ans =
             a: [3x3 double]
             b: [3x1 double]
             c: [-20 4 1]
             d: 200
     NumStates: 3
     NumInputs: 1
    NumOutputs: 1

sp.a
ans =
         -0.1            0            0
          200          -50            0
          750         -150          -50
sp.b
ans =
           1
       -2000
       -7500
sp.c
ans =
   -20     4     1
sp.d
ans =
   200
```

2.2 Frequency response

The frequency response of the stabilizer in 2.1 may be calculated using fr_stsp.

```
[f,ym,ya] = fr_stsp(sp,'log',0.01,1.1,100);
```

With this function call, f is a row vector of frequencies starting at 0.01 Hz and increasing in multiples of 1.1 until f(n) becomes greater than 100 Hz. The gain is ym and the phase, in degrees, is ya.

Alernatively, once the vector f exists,
```
[f,ym,ya]=fr_stsp(sp,f);
```
f may be defined by MATLAB functions logspace or linspace.

With f logarithmic, a Bode diagram may be plotted using

```
plot_bode(f,ym,ya)
```

The result is shown in Figure 2.

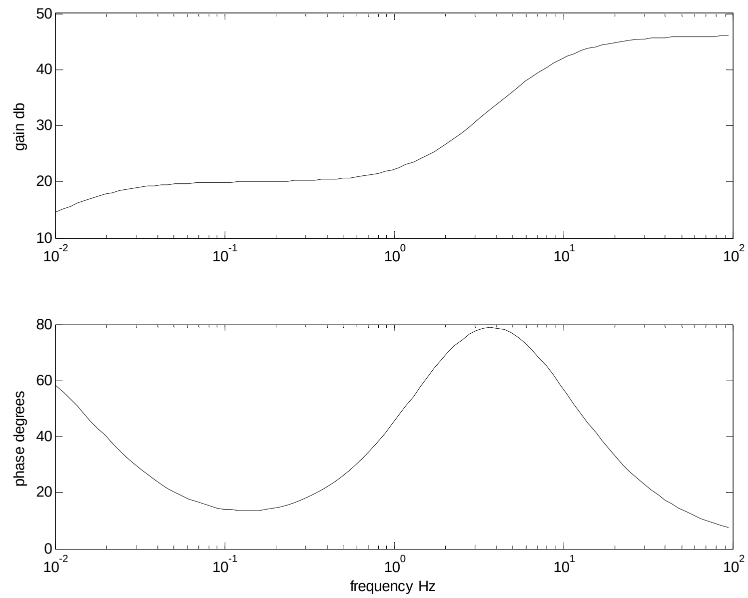


Figure 2 Bode diagram of a power system stabilizer's frequency response

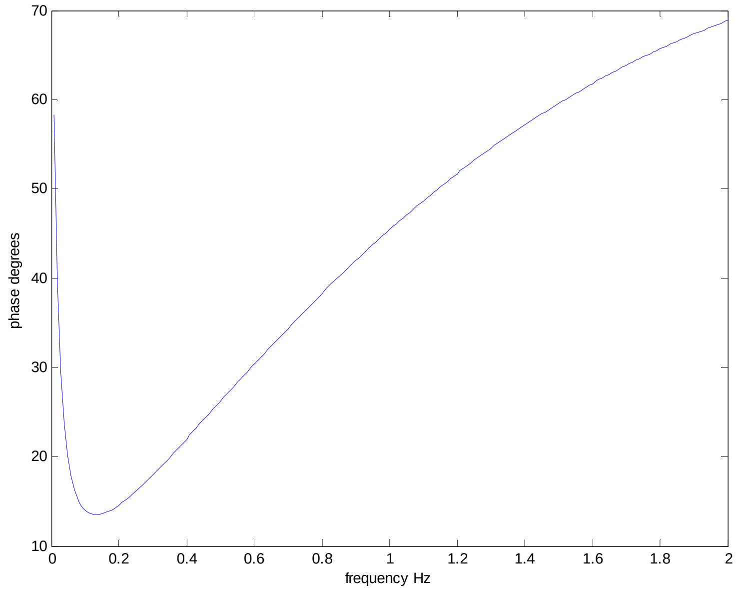


Figure 3 Linear frequency response of power system stabilizer phase

When tuning a power system stabilizer it is normal to display the phase on a linear scale. The frequency response function allows this, i.e.,

```
[f,ym,ya]=fr_stsp(sp,'lin',0.01,0.01,2);
```

The frequency f is now the vector f = 0.01:0.01:2. A plot of the stabilizer phase over this frequency range is shown in Figure 3.

Once the frequency vector is defined, it may be used to plot other responses over the same frequency range. For example if we modify the power system stabilizer parameters

```
sp2 = 10.*wo_stsp(1).*ldlg_stsp(1,.02,.1).*ldlg_stsp(1,.02,.08);
```

We can obtain the response, shown in Figure 4, using

```
[f,ym,ya] = fr_stsp(sp2,f);
hold
plot(f,ya,'r')
```

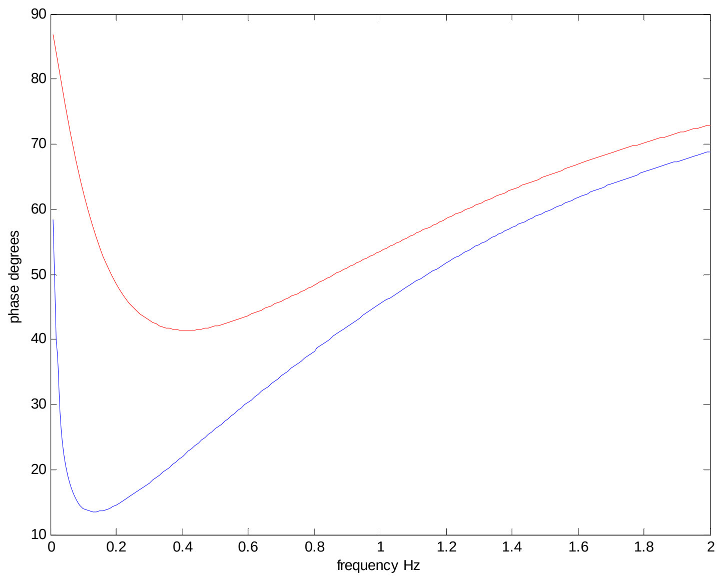


Figure 4 Frequency response of two power system stabilizers

2.3 Multiplying State Space Objects

Two state space objects with the number of outputs of the first equal to the number of inputs of the second may be multiplied by the command

```
s3 = times(s1,s2);
```

This function overrides the MATLAB times function for state space objects, and is equivalent to

```
s3 = s1.*s2;
```

An example is the formulation of the power system stabilizer model

```
sp2 = 10.*wo_stsp(1).*ldlg_stsp(1,.02,.1).*ldlg_stsp(1,.02,.08);
```

The first element, a scalar (10) is automatically converted to
`ssc = stsp([],[],[],10);`
within the times routine.

More general multiplication may be performed using

```
[s3, in1, in2, out1, out2] = times(s1, s2, n_c, out_idx, in_idx);
```

where
s3 is the new state space object
in1 is the order of the inputs of s1 in the s3

in2 is the order of the inputs of s2 in s3
out1 is the order of the outputs of s1 in the output of s3
out2 is the order of the outputs of s2 in the output of s3
n_c is the number of interconnections between s1 and s2
out_idx is the index of the outputs of s1 to be connected to s2
in_idx is the index of the inputs of s2 to be connected to s1

2.3.1 Example: Comparing the response of generator speed and terminal bus frequency

The state space matrices of the two-area system with exciters and thermal governors may be found in the chapter3, results directory of the cd ‘Power System Oscillations’ (sbegstsp.mat).

A state space object with input equal to the voltage reference at generator 1 and outputs equal to that generators speed and terminal bus angle is formed using

```
sgvrsa1 = stsp(a_mat,b_vr(:,1),[c_spd(1,:);c_ang(1,:)],zeros(2,1));
```

To get the per unit frequency, the bus angle is differentiated and divided by the systems angular frequency base (120π). The function **dif_stsp** gives a state space object for an imperfect differentiator with transfer function

$$d(s) = \frac{s}{1 + sT_d}$$

```
sd = dif_stsp(0.01);
[sgvrsf1,in1,in2,out1,out2] = times(sgvrsa1,(1/120/pi).*sd,1,2,1);
```

gives a new state space object that retains the first output of `sgvrsa1` and differentiates and normalizes the second input .

The response to a step change in voltage reference may be obtained using

```
[r, t] = stepres(sgvrsf1,0.05,10,0.01);
```

It is shown in Figure 5.

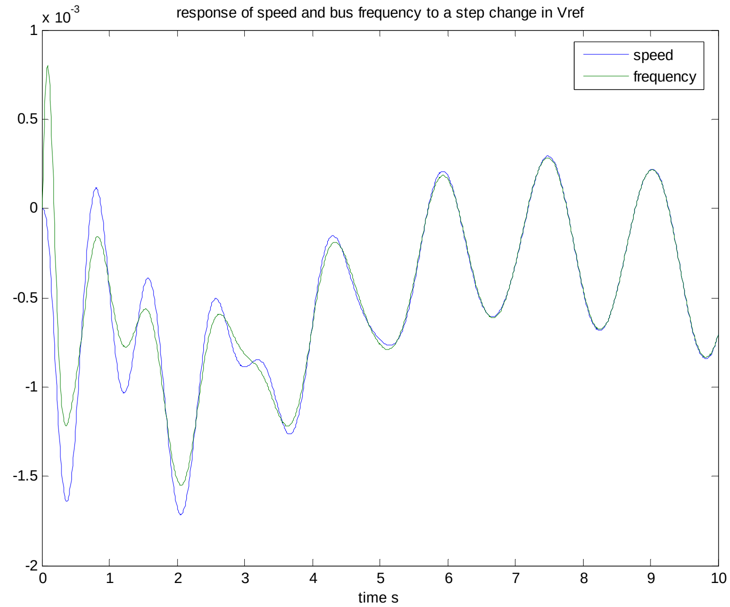


Figure 5 Step response of generator speed and terminal bus frequency to a step change of 0.05 in the exciter voltage reference

2.4 State space objects in feedback

State space objects may be connected in feedback using the function **fb_aug**.

For example, a power system stabilizer may be connected between a generator's speed and its exciter voltage reference. Taking the state space objects sgvrsf1 and sp, the closed loop system may be found using

```
sgvrspd1 = stsp(a_mat,b_vr(:,1),c_spd(1,:),0);
sgvrsps = fb_aug(sgvrspd1,sp,fb1,fb2);
```

fb1 and fb2 are MATLAB structures which define the feedback.
In this case

```
fb1.NumOutputs = 1; fb1.NumInputs = 1; fb1.i_idx = 1;
fb1.o_idx = 1;fb1.ri_idx = 1; fb1.ro_idx=2;
fb2.NumOutputs = 1; fb2.NumInputs = 1; fb2.i_idx = 1;
fb2.o_idx = 1;fb2.ri_idx = []; fb2.ro_idx=[];
```

The eigenvalues of the system without power system stabilizer may be found using the eig function overloaded for stsp objects

```
l = eig(sgvrspd1);
```

and for the closed loop system,

```
lcl = eig(sgvrsps);
```

The eigenvalues of the electromechanical modes are compared in Table 1.

Table 1 Eigenvalues of electromechanical modes with and without Power System Stabilizer

Without PSS	With PSS
0.044387 ∓ 4.0309i	-0.090599 ∓ 4.0112i
-0.55258 ∓ 7.302i	-0.56461 ∓ 7.302i
-0.55317 ∓ 7.383i	-1.7102 ∓ 7.383i

2.5 Root Loci

The root locus as the gain of a feedback control is increased may be determined using

```
rl = rtlocus(sf,sb,gs,gi,ge);
```

where `gs` is the starting gain, `gi` is the gain increment, and `ge` is the finishing gain.

For example, for the system shown in Figure 6

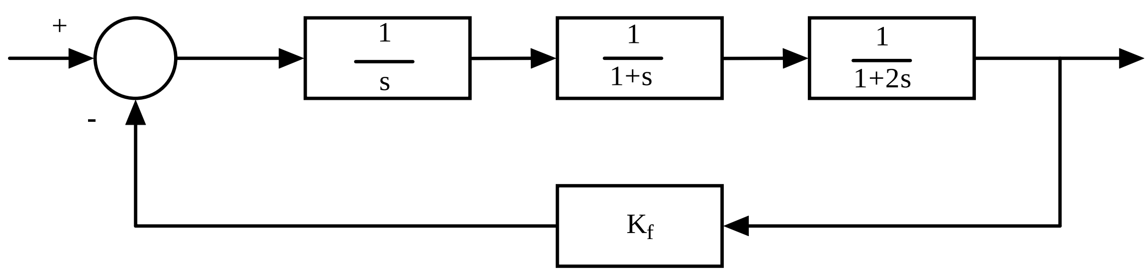


Figure 6 Feed back system

```
sf = int_stsp(1).*lag_stsp(1,1).*lag_stsp(1,2);
sb = stsp([],[],[],1);
gs = 0; gi = 0.1; ge = 20;
```

The root locus with $K_f = 0:0.1:20$ is produced using

```
rl = rtlocus(sf,-sb,0,0.1,20);
plot(rl(:,1),zeros(3,1),'k+');
hold
plot(rl,'k.')
plot(rl(:,[11:10:201]),'r*')
```

The grid, x and y axes and the title were added using

```
grid
labxyarg
title('root locus with feedback gain')
```

The result is shown in Figure 7. The red asterisks indicate feed back gains of 1:1:20.

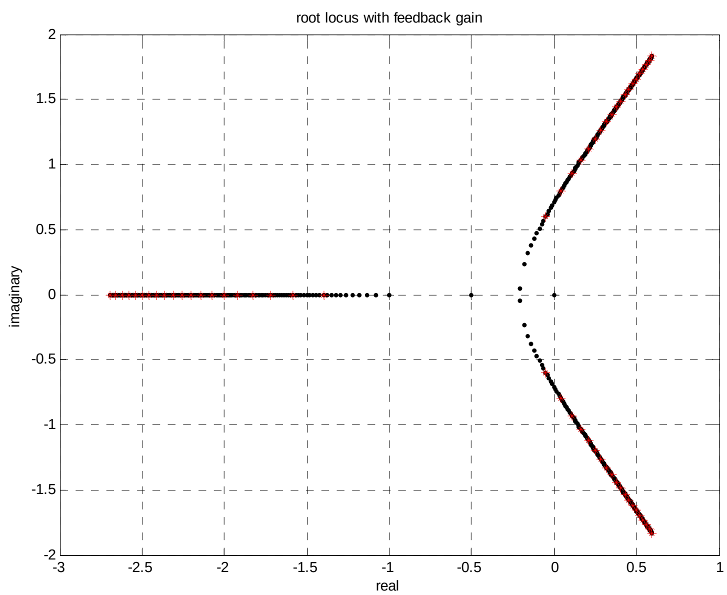


Figure 7 Root locus with feedback gain

2.6 Poles and Zeros

The poles of s may be found using the overloaded **eig** function

```
lp = eig(s);
```

The transmission zeros may be found using **zeros**

```
lz = zeros(s);
```

For the two-area system model used in example 2.4:

```
lp = eig(sgvrspd1);
lz = zeros(sgvrspd1);
```

The dominant poles and zeros are shown in Figure 8.

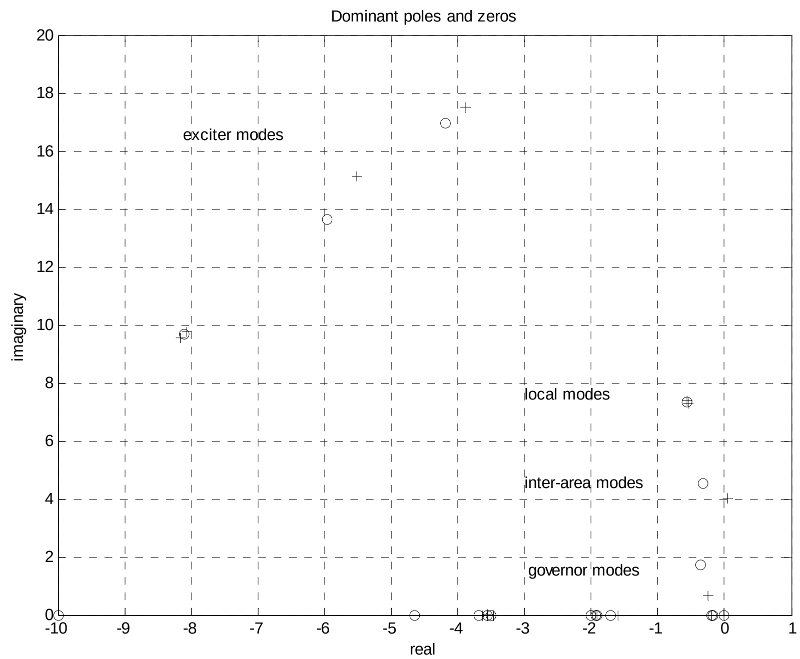


Figure 8 Dominant Poles and Transmission zeros

2.7 Residues

The residues of a state space object may be determined using

```
r = residue(s);
```

r is a MATLAB cell with each element giving the residues for all the inputs and outputs of s associated with one eigenvalue of the state matrix

2.8 Nonlinear Divisors

In linear power system models without governors, generator damping or infinite buses, there are two theoretically zero eigenvalues. The state matrix in such a system cannot be diagonalized but it may be reduced to a Jordan Canonical Form.

Example: Two-area system with classical generator models

The state space matrix is

0	376.99	0	0	0	0	0	0
-0.074585	0	0.067735	0	0.0032971	0	0.0035524	0
0	0	0	376.99	0	0	0	0
0.07199	0	0.086612	0	0.0075941	0	0.0070283	0
0	0	0	0	0	376.99	0	0
0.0064902	0	0.011573	0	-0.077948	0	0.059884	0
0	0	0	0	0	0	0	376.99
0.012327	0	0.016269	0	0.067112	0	0.095708	0

The eigenvalues calculated using the MATLAB **eig** function are

0
0
-0.0000 - 3.5319i
-0.0000 + 3.5319i
-0.0000 - 7.5092i
-0.0000 + 7.5092i
-0.0000 - 7.5746i
-0.0000 + 7.5746i

The first two eigenvalues are both zero.

The eigenvectors corresponding to the two zero eigenvalues are practically identical, and the eigenvector matrix is close to singular. This indicates that the zero eigenvalues are nonlinear divisors.

With

```
sgem = stsp(a_mat,b_pm(:,1),c_spd(1,:),0);
```

The eigenvalues and eigenvectors of the state space object are obtained by

```
[lsgem ,usgem] = eig(sgem);
```

lsgem is not a diagonal matrix of eigenvalues, but a matrix with the eigenvalues on the diagonal, and with unity in the off diagonal element connecting the two equal eigenvalues. This is the Jordan Canonical Form.

0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	-3.5319i	0	0	0	0	0
0	0	0	3.5319i	0	0	0	0
0	0	0	0	-7.5092i	0	0	0
0	0	0	0	0	7.5092i	0	0
0	0	0	0	0	0	-7.5746i	0
0	0	0	0	0	0	0	7.5746i

The first two eigenvectors are

.5	0
0	0.0013263
.5	0
0	0.0013263
.5	0
0	0.0013263
.5	0
0	0.0013263

3 State Space Function Descriptions

3.1 stsp

syntax: `s = stsp(a,b,c,d)`
purpose: the constructor function for a stsp object
input: state space matrices
- a state matrix (nxn)
- b input matrix (nxni)
- c output matrix (noxn)
- d feed forward matrix (noxni)

output: state space object s, a member of class stsp, with the following structure
- s.a
- s.b
- s.c
- s.d
- s.NumStates
- s.NumInputs
- s.NumOutputs

3.2 add_in

syntax: `sa=add_in(s,b,d)`
purpose: adds specified additional inputs to a stsp object
inputs
- s original state space object
- b additional input matrix; must have rows equal to the number of states in s
- d additional d matrix; must have number of rows equal to number of outputs of s, and columns equal to number of columns of b

Note:
if b is an empty matrix the original input or output matrix is unaltered
if d is an empty matrix the original d is augmented by zeros to be consistent with the new b matrix

output sa new state space object with added inputs following the original inputs

3.3 add_out

syntax: `sa=add_out(s,c,d)`
purpose: adds soecified outputs to a stsp object
inputs
- s original state space object
- c additional output matrix; must have columns equal to the number of states in s
- d additional d matrix; must have number of rows equal to number of columns of c , and columns equal to number of inputs of s

Note:
if c is an empty matrix the original input or output matrix is unaltered
if d is an empty matrix the original d is augmented by zeros to be consistent with the new c matrix

output sa   new state space object with added outputs following the original outputs

3.4 combine_in

syntax: `si=combine_in(s,s1_idx,s2_idx,r_idx)`
purpose: sums specified inputs, and retains specified inputs

inputs	s	original state space object
	s1_idx	index of the first inputs to be summed
	s2_idx	index of the second inputs to be summed
	r_idx	index of the inouts to be retained
output	si	modified state space object

In si, the first inputs correspond to the summed inputs and the retained inputs folllow.

3.5 cpc_stsp

syntax: `[sc,gmin] = cpc_stsp(s,grel);`
`purpose:` Finds a robust controller based on co-prime uncertainty

input:	s	shaped open loop state space object
	grel	multiplier for optimum gamma
output:	sc	optimal positive feedback control
	gmin	gamma min (1/stability margin)

Modified from *coprimeunc*: Table 9.70 Skogestad and Postlewhwaite, *'Multivariable Feedback Control'*, John Wiley and Sons, 1997.

3.6 display

The overloaded dispay function for the stsp class

3.7 dompole

syntax: `[l,xs,vs,ps,rs] = dompole(s,ls)`
purpose: computes the most dominant pole closest to the supplied pole estimate

inputs:	s	a state space object
	ls	initial pole estimate
output :	l	dominant pole
	xs	right eigenvector
	vs	left eigenvector
	ps	paticipation factor
	rs	residue

3.8 dstate

syntax: `[y,xn,dx] = dstate(s,x,u,xmax,xmin,ymax,ymin);`
purpose: Determines the rate of change of states for the state space objects, and implements output limits and non-windup state limits.
inputs:

s	state space object

x	starting states (length equal to the number of states of s)
u	input vector (length equal to the number of inputs of s)
xmax	maximum value of x (length equal to the number of states of s)
xmin	minimum value of x (length equal to the number of states of s)
ymax	maximum value of y (length equal to the number of outputs of s)
ymin	minimum value of y (length equal to the number of outputs of s)

outputs:

y - output vector
xn - limited state vector
dx - rate of change of state vector

Note: a non-windup limit on state x is defined by:

```
if x>xmax; x=xmax and if dx/dt>0; dx/dt=0
if x<xmin;  x=xmin and if dx/dt<0; dx/dt=0
```

3.9 eig

syntax: `[l,u,v,p,sm] = eig(s,tol);`
purpose: performs eigenvalue analysis of a stsp object
inputs:

s	state space object
tol	tolerance for determining whether eigenvectors are equal – default 1e-6

outputs:

l	if there are no equal eigenvalues, l is the diagonal matrix of eigenvalues if there are nonlinear divisors, l is the Jordan canonical form
u	the matrix of right eigenvectors
v	the matrix of left eigenvectors
p	the participation matrix p = u.*v.’
sm	a modified state space object - am=l; bm=vb;cm=cu;dm=d

When the state matrix has equal eigenvalues, which are nonlinear divisors, the QR algorithm used in the standard MATLB eig function gives either equal eigenvectors or pathalogically equal eigenvectors, i.e., the eigenvector matrix is singular. This situation may be detected detected using the MATLAB condeig function. This gives the condition numbers of the eigenvectors which are compared to tol. If the condition numbers are greater than tol, the state matrix is assumed to have nonlinear divisors.

The eigenvectors for nonlinear divisors are modified so that the eigenvalue matrix is in Jordan canonical form with unity in the upper off diagonal between the equal eigenvalues.

Overrides the MATLAB eig function for state space objects

3.10 eq

syntax: `e = eq(s1,s2);` or `s1==s2`
purpose: checks the equality of two stsp objects
input: two state space objects
output:

if a,b,c,and d are identical for both s1 and s2; e = 1
otherwise; e = 0
example

```
sf = int_stsp(1).*lag_stsp(1,1).*lag_stsp(1,2);
sb = -stsp([],[],[],1);
sf==sb

ans =

     0
sf==sf

ans =

     1
```

Overrides MATLAB eq for state space objects.

3.11 eval

syntax: `[v,x] = eval(s,l,u);`
pupose: computes the output value of the stsp object s at a specified point and input
inputs:

s	a state space object
l	a scalar which may be complex
u	input column vector (length s.NumInputs)

outputs:

v	the value of the state space output evaluated at l v = s.c*inv(lI-s.a)*s.b*u + s.d*u
x	the value of the states evaluated at l x = inv(lI-s.a)*s.b*u

Overrides the eval function for state space objects

example:

```
sf = int_stsp(1).*lag_stsp(1,1).*lag_stsp(1,2);
[v,x] = eval(sf,i*2*pi*10,1)
v =

-4.8107e-008 +2.0148e-006i
x =

           0 -   0.015915i
 -0.00025324 -4.0304e-006i
-4.8107e-008 +2.0148e-006i
```

3.12 exres

syntax:
```
[rs,ts,x_fin]=exres(s,vm,t_int,t_samp,t_noise,t_fin,x_start)
```
purpose: calculates response to a random input
inputs:

s	– stsp object
vm	– input magnitude
t_init	– time step used for simulation
t_samp	– time step used for output
t_noise	– time step at which random input is changed
t_fin	– finish time of simulation
x_start	– initial states

outputs:

rs	– response sampled at t_so = t_fin/n_so
ts	– time vector corresponding to rs
x_fin	– final output

3.13 fb_aug

syntax:
```
[sfba,in1,in2,out1,out2] = fb_aug(s1,s2,fb1,fb2);
```
purpose: connects two stsp objects in feedback.
inputs:

s1	forward loop state space object
s2	feedback loop state space object
fb1	structure defining input and output indexes which define the feedback for s1
fb2	structure defining input and output indexes which define the feedback for s2

Structure format for fb1 and fb2:

fb.NumInputs fb.NumOutputs fb.i_idx fb.o_idx fb.ri_idx fb.ro_idx

outputs:

sfba	closed loop state space object
in1	correspondence between inputs to sfba and inputs to s1
in2	correspondence between inputs to sfba and inputs to s2
out1	correspondence between outputs of sfba and outputs of s1
out2	correspondence between outputs of sfba and outputs of s2

Note:
The number of outputs defined in fb1 must be the same as the number of inputs defined in fb2
The number of inputs defined in fb1 must be the same as the number of outputs defined in fb2
The feedback inputs and outputs are defined by i_idx and o_idx
The inputs of sfba are the inputs of s1 defined by fb1.ri_idx
The inputs of s2 defined by fb2.ri_idx
The outputs of sfba are the outputs of s1 defined by fb1.ro_idx together with the outputs of s2 defined by fb2.ro_idx

If fb1 and fb2 are not supplied
The inputs and outputs of s1 are retained
The input to s2 is the output of s1
The outputs of s2 plus the original inputs of s1 are the inputs to s1
The number of outputs of s1 must equal the number of inputs of s2
The number of inputs of s1 must equal the number of outputs of s2

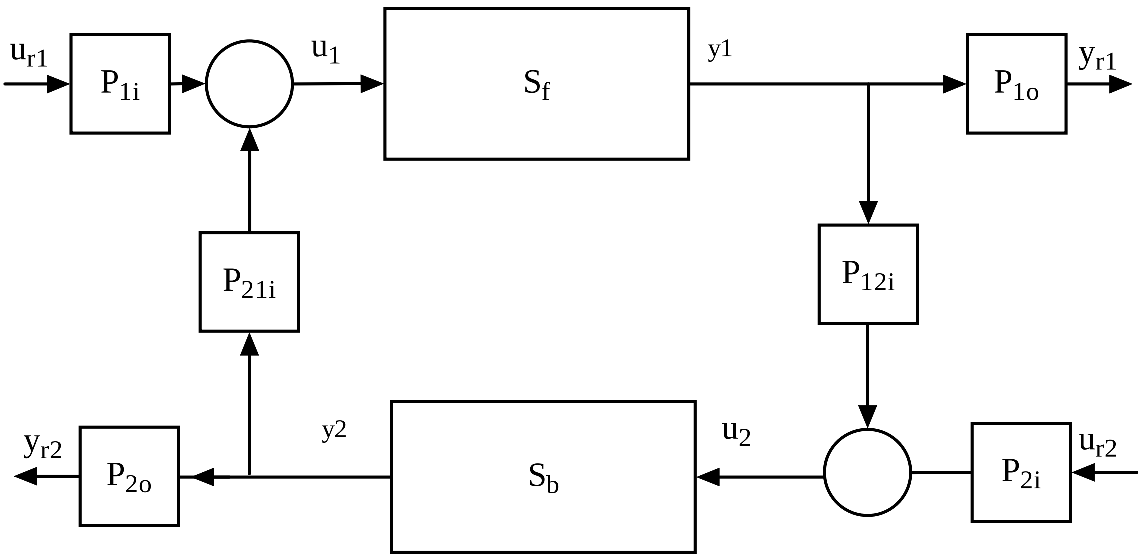


Figure 9 General feedback model

The basic state space model is

$$\begin{aligned}
\frac{dx_1}{dt} &= a_1x_1 + b_1u_1 \\
y_1 &= c_1x_1 + d_1u_1 \\
S_f &= \mathrm{stsp}(a_1, b_1, c_1, d_1) \\
\frac{dx_2}{dt} &= a_2x_1 + b_2u_2 \\
y_1 &= c_2x_2 + d_2u_2 \\
S_b &= \mathrm{stsp}(a_2, b_2, c_2, d_2)
\end{aligned} \qquad \textbf{(1.1)}$$

The interconnection equations are

$$\begin{aligned}
u_1 &= P_{1i}u_{r1} + P_{21i}y_2 \\
u_2 &= P_{2i}u_{r2} + P_{12i}y_1 \\
y_{r1} &= P_{1o}y_1 \\
y_{r2} &= P_{2o}y_2
\end{aligned} \qquad \textbf{(1.2)}$$

Thus u_1 and u_2 are

$$\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} I & -P_{21i}d_2 \\ -P_{12i}d_1 & I \end{bmatrix}^{-1} \left(\begin{bmatrix} P_{i1} & 0 \\ 0 & P_{i2} \end{bmatrix} \begin{bmatrix} u_{r1} \\ u_{r2} \end{bmatrix} + \begin{bmatrix} 0 & P_{21i} \\ P_{12i} & 0 \end{bmatrix} \begin{bmatrix} c_1 & 0 \\ 0 & c_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right) \tag{1.3}$$

The retained outputs are given by

$$\begin{aligned} \begin{bmatrix} y_{r1} \\ y_{r2} \end{bmatrix} &= \begin{bmatrix} P_{10} & 0 \\ 0 & P_{20} \end{bmatrix} \left(\begin{bmatrix} c_1 & 0 \\ 0 & c_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} d_1 & 0 \\ 0 & d_2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \right) \\ &= \begin{bmatrix} P_{10} & 0 \\ 0 & P_{20} \end{bmatrix} \left(\left\{ \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix} + \begin{bmatrix} d_1 & 0 \\ 0 & d_2 \end{bmatrix} \begin{bmatrix} I & -P_{21i}d_2 \\ -P_{12i}d_1 & I \end{bmatrix}^{-1} \begin{bmatrix} 0 & P_{21i} \\ P_{12i} & 0 \end{bmatrix} \right\} \begin{bmatrix} c_1 & 0 \\ 0 & c_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right) \\ &+ \begin{bmatrix} P_{10} & 0 \\ 0 & P_{20} \end{bmatrix} \begin{bmatrix} d_1 & 0 \\ 0 & d_2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \end{aligned} \tag{1.4}$$

The state dynamic equations are

$$\begin{aligned} \frac{d}{dt} \begin{bmatrix} x1 \\ x2 \end{bmatrix} &= \left(\begin{bmatrix} a1 & 0 \\ 0 & a2 \end{bmatrix} + \begin{bmatrix} b1 & 0 \\ 0 & b2 \end{bmatrix} \begin{bmatrix} I & -P_{21i}d_2 \\ -P_{12i}d_1 & I \end{bmatrix}^{-1} \begin{bmatrix} 0 & P_{21i} \\ P_{12i} & 0 \end{bmatrix} \begin{bmatrix} c_1 & 0 \\ 0 & c_2 \end{bmatrix} \right) \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \\ &+ \begin{bmatrix} b1 & 0 \\ 0 & b2 \end{bmatrix} \begin{bmatrix} I & -P_{21i}d_2 \\ -P_{12i}d_1 & I \end{bmatrix}^{-1} \begin{bmatrix} P_{1i} & 0 \\ 0 & P_{2i} \end{bmatrix} \begin{bmatrix} u_{r1} \\ u_{r2} \end{bmatrix} \end{aligned} \tag{1.5}$$

3.13.1 Example 1

With

```
sf = int_stsp(1).*lag_stsp(100,0.05)
```

and

```
sb = stsp([],[],[],10)
sfb=fb_aug(sf,-sb)
sfb =
ans =

             a: [2x2 double]
             b: [2x1 double]
             c: [0 100]
             d: 0
     NumStates: 2
     NumInputs: 1
    NumOutputs: 1
sfb.a
ans =

           0       -1000
          20         -20
sfb.b
ans =

     1
     0

sfb.c
ans =

     0   100
```

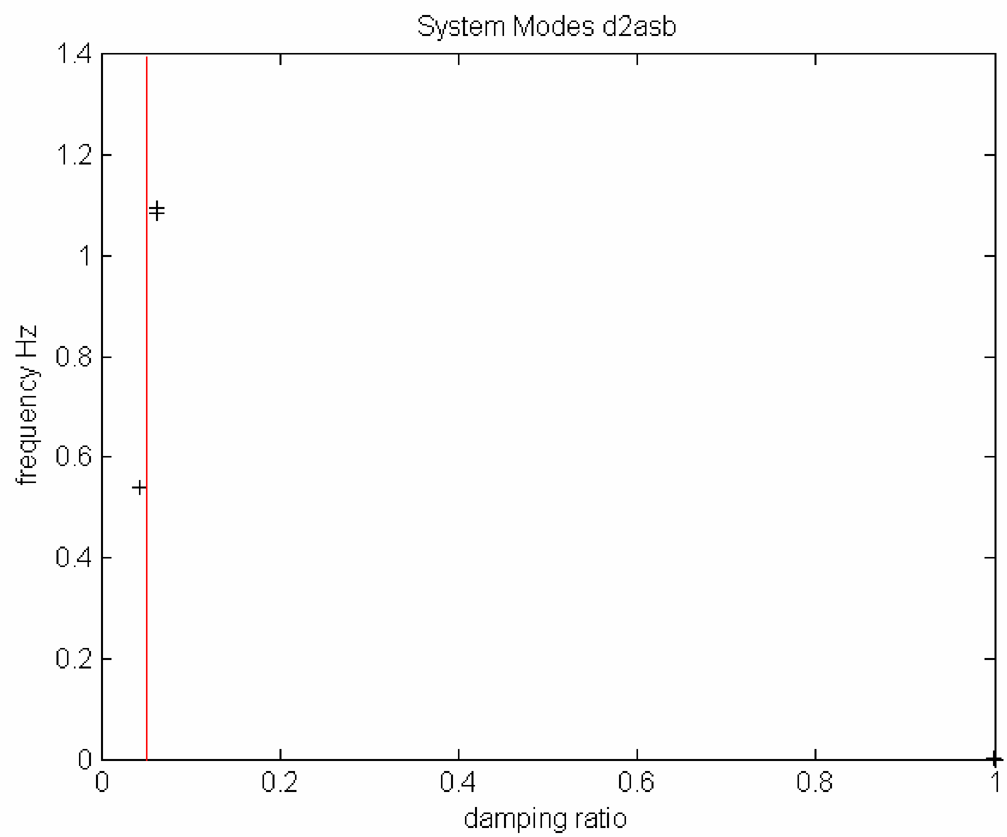
```
sfb.d
ans =

     0
l=eig(sfb)
l =

          -10 -     141.07i
          -10 +     141.07i
```

3.13.2 Example 2

For a two area power system with four generators with no controls, the modes are



To examine the effect of an exciter on generator 1, an exciter model is formed using

```
sexc = lag_stsp(100,0.05)
```

A state space model of the system with input the generator field voltages; and output the generator terminal voltage magnitudes is formed using

```
sefdvm = stsp(sps.a,b,c,d)
```

Where

```
b=sps.b(:,1:4);
c=sps.c([1 2 6 7],:);
d=zeros(4)
```

The feedback definition structures are

```
fb1 =

    NumOutputs: 1
```

```
     NumInputs: 1
         o_idx: 1
         i_idx: 1
        ri_idx: [2 3 4]
        ro_idx: [1 2 3 4]
fb2 =

    NumOutputs: 1
     NumInputs: 1
         o_idx: 1
         i_idx: 1
        ri_idx: []
        ro_idx: []
```

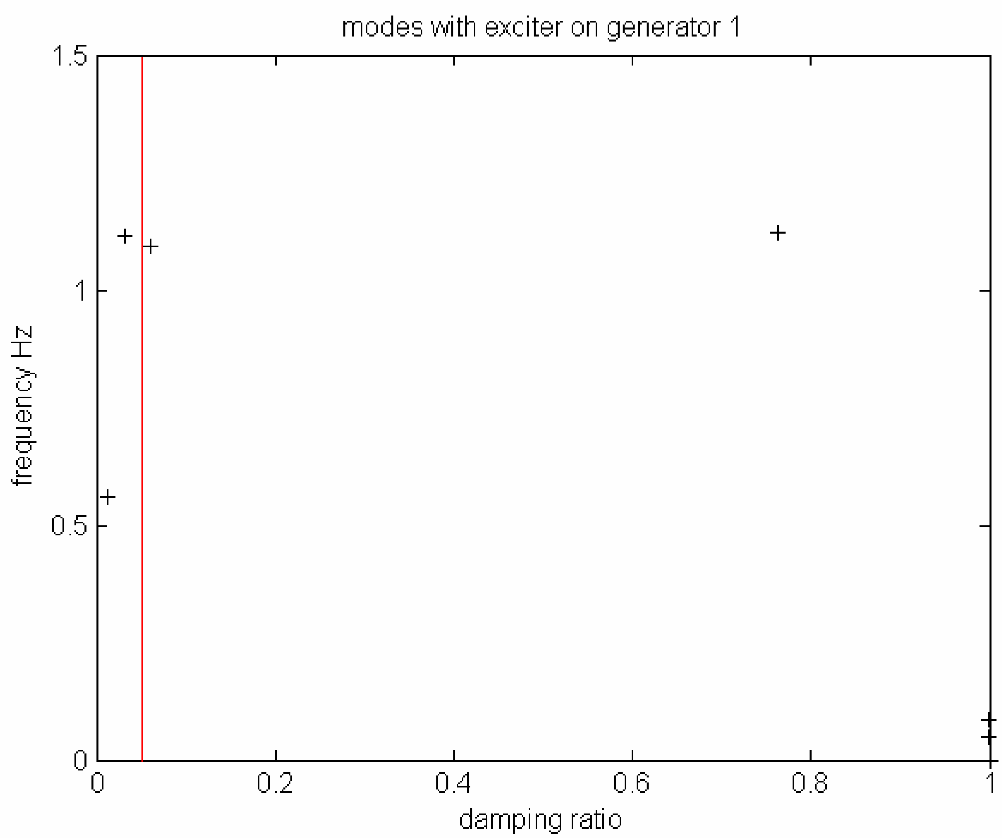
The feedback is implemented using

```
[sbexc1,in1,in2,out1,out2] = fb_aug(sefdvm,-sexc,fb1,fb2)
```

The eigenvalues recalculated with the exciter on generator 1 are obtained using

```
[l,u,v,p,sm] = eig(sbexc1);
damp = real(l)./abs(l);
freq = imag(l)/2/pi
```

The frequency against damping ratio plot with feedback is



The effect is to reduce the damping of the local mode associated with generators 1 and 2, and the interarea mode and to introduce a well damped exciter mode with damping ratio `0.76298` and frequency `1.1219` Hz.

3.14 fr_mstsp

syntax:

```
[f,y,smn,smx,cn]=fr_mstsp(s,ftype,fstart,fstep,fend);
```

purpose: calculates the multivariable frequency response of a stsp object

inputs:

s a state space object - which must have a non-empty state matrix

ftype 'lin' gives a linear set of frequencies

'log' gives a logarithmic range of frequencies

in this case fstep is treated as a multiplier - minimum value 1.01

fstart the start frequency in Hz

fstep the frequency step in Hz

fend the end frequency in Hz

Note: if fstart, fstep and fend are not entered, ftype must be a vector of frequencies in Hz

outputs:

f the frequency vector used for plotting

y a cell object

y{k}(m,n) gives the frequency response of the mth output to the nth input corresponding to the frency f(k)

smn the minimum singular value vector

smx the maximum singular value vector

cn the condition number vector (smx./smn)

3.14.1 Example

For the two-area system with static exciters, thermal governors and power system stabilizers

```
slmvm = stsp(sps.a,sps.b(:,15:16),sps.c(1:13,:),zeros(13,2));
```

The multivariable frequency response is calculated using

```
[f,y,smn,smx,cn] = fr_mstsp(slmvm,'log',0.01,1.1,100);
```

The response is plotted using

```
subplot(3,1,1);semilogx(f,smx);
subplot(3,1,2);semilogx(f,smn);
subplot(3,1,3);semilogx(f,cn);
xlabel('frequency Hz')
ylabel('smx')
ylabel('smn')
ylabel('cn')
```

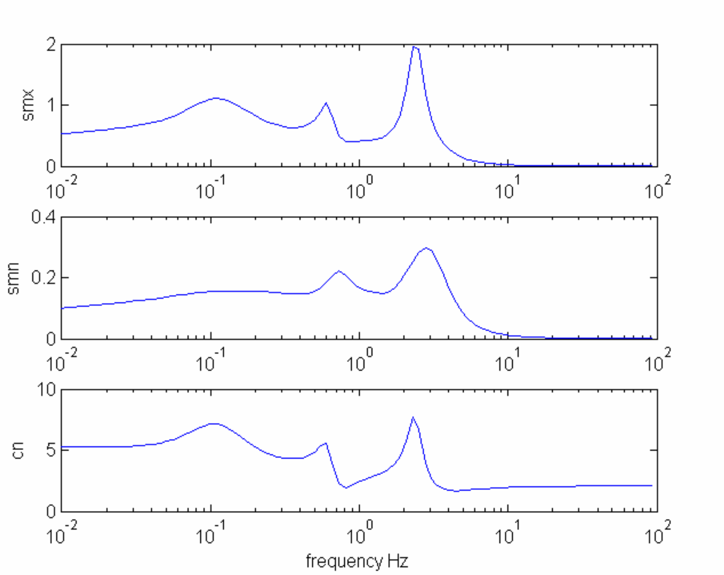


Figure 10 Multivariable frequency response two-area system

3.15 fr_stsp

syntax: `[f,ymag,yphase]=fr_stsp(s,ftype,fstart,fstep,fend);`
purpose: calculates the frequency reponse of a single inputpsingle output stsp object
inputs:

s	a single input, single output state space object - which must have a non-empty state matrix
ftype	‘lin’ gives a linear set of frequencies
ftype	‘log’ gives a logarithmic range of frequencies in this case fstep is treated as a multiplier - minimum value 1.01
fstart	the start frequency in Hz
fstep	the frequency step in Hz
fend	the end frequency in Hz if fstart, fstep and fend are not entered, ftype is a vector of frequencies in Hz

outputs:

f	the frequency vector used for plotting
ymag	the magnitude vector
yphase	the phase vector in degrees

To plot a Bode diagram, form the frequency vector using ftype = ‘log’, then plot using plot_bode.
To plot a Nyquist diagram use:

```
plot(ym.*cos(ya*pi/180),ym.*sin(ya*pi/180));
```

3.15.1 Example

For the two area system

```
slm1vm1 = stsp(sps.a,sps.b(:,15),sps.c(1,:),0);
[f,ym,ya] = fr_stsp(slm1vm1,'log',0.01,1.1,100);
plot_bode(f,ym,ya);
```

The bode diagram is shown in Figure 11

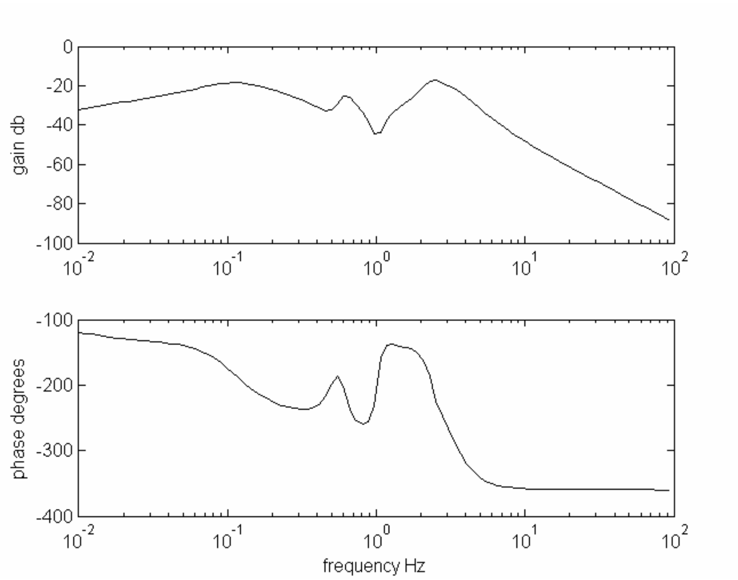


Figure 11 Bode diagram of the response of the voltage at bus 1 to active load modulation at bus 4.

3.16 full

syntax `sf = full(s);`

purpose: converts the state matrices of s (s.a,s.b,s.c,and s.d) to full storage, i.e.,

```
sf.a = full(s.a);sf.b=full(s.b);sf.c=full(s.c);sf.d=full(s.d);
```

3.17 get

syntax `s_v = get(s)`

purpose: overloads get for state space objects. Allows state space object fields to be accessed outside the @stsp directory and is used by display.

input:

s state space object

output:

s_v MATLAB structure with the same fields as s
sv.a
sv.b
sv.c
sv.d
sv.NumStates
sv.NumInputs
sv.NumOutputs

3.18 init_stsp

syntax: `[x,u]=init_stsp(s,y);`

purpose: finds the initial input vector and the initial states to satisfy an initial output y
input:
- s state space object
- y initial output vector

outputs:
- x initial state vector
- u initial input vector

3.19 inv

syntax: `si=inv(s);`
purpose: finds inverse of a state space system provided s.d has an inverse
input: s a state space object, s.d must be square and non-singular
output: si a state space object equal to the inverse of s

```
si.a = s.a-s.b*inv(s.d)*s.c
si.b = s.b*inv(s.d)
si.c = -inv(s.d)*s.c
si.d = inv(s.d)
```

3.20 minus

syntax: `st = minus(s1,s2);` or `st = s1-s2;`
purpose: overrides minus for state space objects.
Subtracts the outputs of two state space objects and creates a new state space object
inputs: two state space objects with the same number of ouputs
output: a new state space object representing the difference

3.21 nlsr

syntax: `[y,x,t] = nlsr(s,i_type,u,h,tmax,xmax,xmin,ymax,ymin);`
purpose: Calculates response of a state space object to a step, impulse or ramp input using predictor-corrector integration. Each state may be non-windup limited between xmax and xmin. Each output may be limited to between ymax and ymin.
input:

s	state space object
i_type	'st' - step response
	'im; - impulse response
	'ra' - ramp response
u	input vector
h	time step
tmax	maximum response time
xmax	vector of maximum state limits
xmin	vector of minimum state limits
ymax	vector of maximum output limits
ymin	vector of minimum output limits

outputs:

y	output time response
x	state time response
t	time vector

3.22 norm_copr

syntax: `[sn,sm] = norm_copr(s);`
Forms the normalized coprime factors of the state space object s

Based on Table 4.1 Multivariable Feedback Control, Skogestad and Postlethwaite, John Wiley and Sons, 1997

3.23 order_in

syntax: `sm = order_in(s,b_idx);`
purpose: modifies the order of the inputs of s
inputs: s state space object
b_idx index of inputs required in new state space object
output: sm state space object with modified input order

3.24 order_out

syntax: `sm = order_out(s,c_idx);`
purpose: modifies the order of the outputs of s
inputs: s state space object
c_idx index of outputs required in new state space object
output sm state space object with modified output order

3.25 order_st

syntax: `sm = order_st(s,x_idx)`
purpose: modifies the order of the states of s
inputs: s state space object
x_idx index of state order required in new state space object
output sm state space object with modified state order

3.26 parallel

syntax: `st = parallel(s1,s2,s3,......sn);`
purpose:: parallels state space objects and creates a new state space object.
inputs: a set of state space objects each having the same number of outputs and inputs
output: a new state space object representing the parallel combination, i.e.,
inputs to st are the inputs s1 to sn
output of st is the summed outputs of s1 to sn

3.27 parin

syntax: `st = parin(s1,s2,s3,......sn);`
purpose:: parallels the inputs ofstate space objects and creates a new state space object.
inputs: a set of state space objects each having the same number of inputs
output: a new state space object with a single set of inputs representing the parallel combination, of the original inputs

3.28 plus

syntax `st = plus(s1,s2,so1,so2,ro1,ro2);`

purpose: sums the outputs of two state space objects to create a new state space object.

inputs: s1 and s2, state space objects to be summed

so1 and so2 indices of the outputs of s1 and s2 to be summed - must be the same length

ro1 and ro2 indices of outputs of s1 and s2 to be retained

output: st a new state space object representing the sum, with the input the inputs of s1 and s2 with outputs the sum of the so1 outputs of s1 and so2 outputs of s2, and with the ro1 outputs of s1 and the ro2 outputs of s2

Note: if there only two inputs the output of st is the sum of the outputs of s1 and s2, and s1 and s2 must have the same number of outputs.

Overrides plus for state space objects.

3.29 power

syntax: `sp = power(s,b);`

`or`

`sp = s.^b`

purpose: overloads MATLAB's element by element power function for state space objects

input: s a state space object

b an integer which may be positive, negative or zero

output: sp a state space object formed as follows

if b >0

sp = s.*s.*s…..*s b times

if b is equal to 0, s must have the same number of inputs as outputs, then

`sp = stsp([],[],[],eye(s.NumOutputs))`

if b<0

s must be invertible (s.d not singular)

sp = inv(s).*inv(s).*inv(s)………*inv(s) abs(b) times

3.30 randres

syntax: `[r,t,xf,u] = randres(s,x0,ts,ns,dur)`

purpose: calculates the response to a random inputs applied to all inuts of s

inputs: s stsp object

x0 initial state vector

ts solution time step s

ns number of time steps at which the random input is changed

dur total simulation time s

output :r output resposne

t time vector

xf final state vector

u generated random input

3.31 reduss

syntax: `sred = reduss(s,k);`

purpose: balanced residual reduction of an unstable system
retains k states in the stable part
based on
Table 11.30 Skogestad and Postlewhwaite, 'Multivariable Feedback Control', John Wiley and Sons, 1996

input: s a state space object with some eigenvalues having positive real parts
k the number of stable states to be retained

output: sred a reduced order state space object

3.31.1 Example

For the two-area system with exciters and governors, a reduced system model between the voltage reference and the terminal voltage magnitude at generator 1 is obtained using

```
svrvm1 = stsp(a_mat,b_vr(:,1),c_v(1,:),0);
sred = reduss(svrvm1,10);
```

The frequency responses of the full system model and the reduced system model are compared in Figure 12.

The eigenvalues of the reduced system include the unstable modes of the full system together with the 10 most significant poles of the full system.

```
lr =
  -1.0192e-006
      -2.7533
      0.12833 -      3.9591i
      0.12833 +      3.9591i
     -0.38213 -      7.1739i
     -0.38213 +      7.1739i
      -17.512
      -5.7846 -      17.342i
      -5.7846 +      17.342i
      -13.268 -      14.324i
      -13.268 +      14.324i
      -9366.1
```

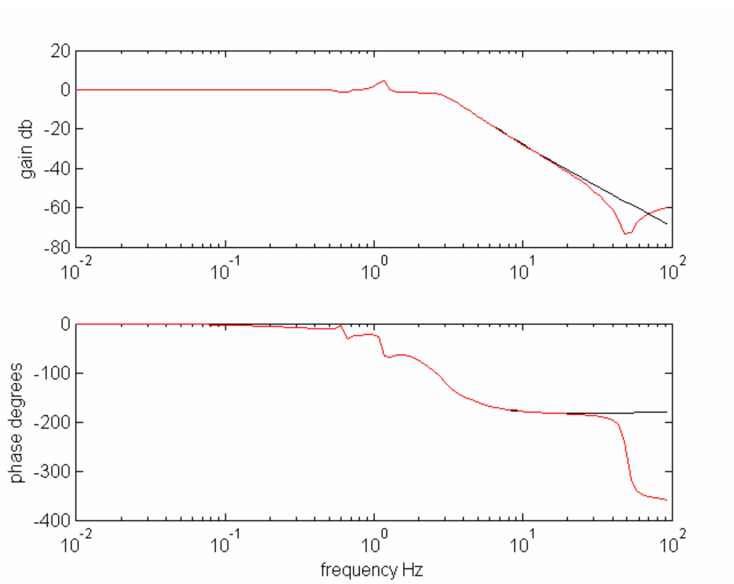


Figure 12 Frequency responses of full and reduced system models

Step responses of the full and reduced systems are shown in Figure 13.

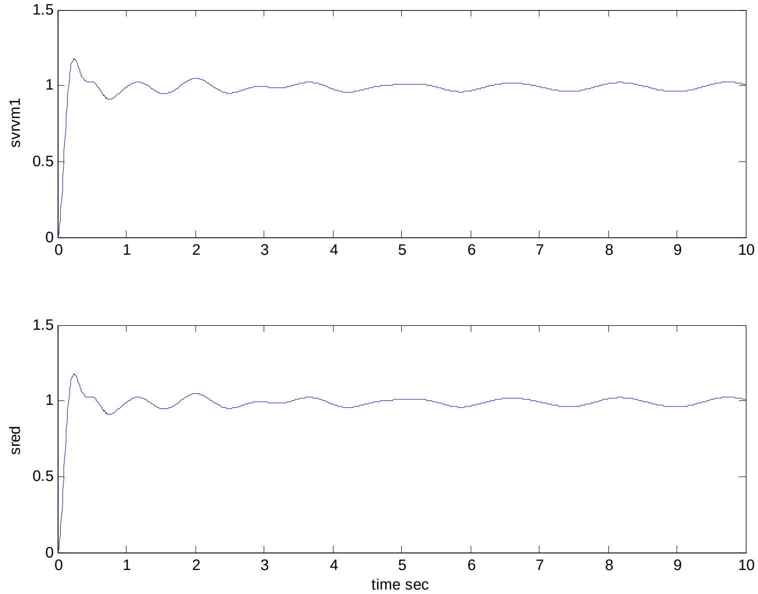


Figure 13 Responses of full and reduced order systems to a unit step input

3.32 residue

syntax: `r = residue(s);`
input: s a state space object
output: r a cell with r{k} containing the residues associated with the k^{th} eigenvalue of s.a. The eigenvalues are sorted in ascending order of magnitude

```
r(k)={(s.c*u(:,k))*(v(k,:)*s.b)};
```

3.33 response

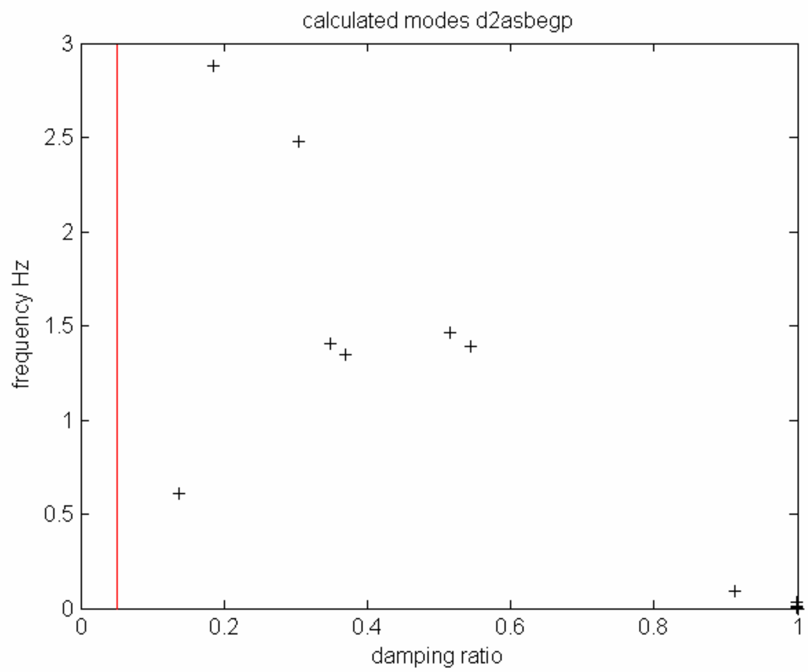
syntax: [res, t] = response(s,v_in,t,x0);
inputs:

- s state space object
- v_in the input matrix, the k^{th} column is the value of the input at t(k)
- t a row vector of time corresponding to the input
- x0 initial value of x - zero if not specified

outputs: res - time response matrix
Note: v_in must be a matrix with the number of rows equal to the number of inputs of s, and number of columns equal to the length of t
Use `plot(t,res)` to view the response

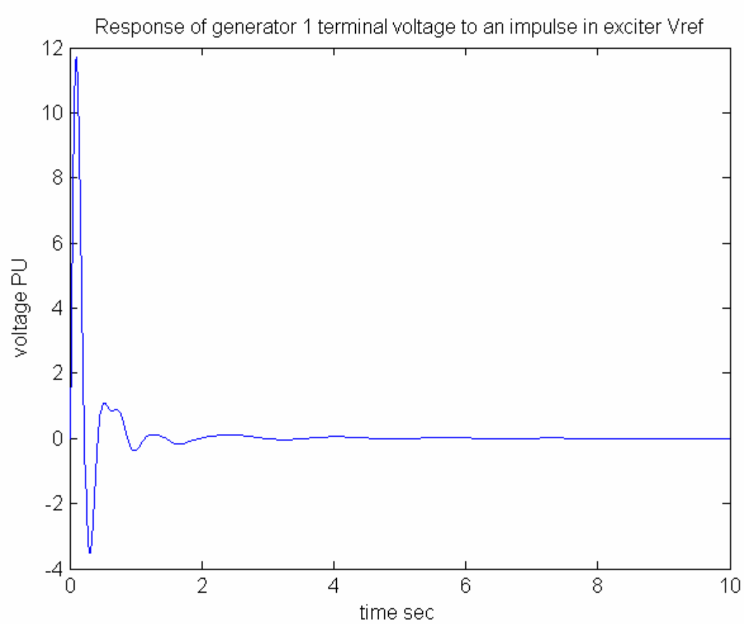
3.33.1 Example

For the two-area power system with exciters, governors and power system stabilizers modelled using the Power System Toolbox the modes are




```
svrvm1 = stsp(a_mat,b_vr(:,1),c_v(1,:),0);
x0 = zeros(NumStates,1);
t = 0:0.01:10;
v_in = 1;
f_in = zeros(1,length(t));
f_in(1) = 100;f_in(2)=100;
[res,t,x] = response(svrvm1,v_in,t,x0,f_in);
plot(t,res);
```

Note: f_in is 100 for 0.01 seconds and zero thereafter.



3.34 rtlocus

syntax: `rl = rtlocus(s1,s2,gstart,gstep,gend,flag);`
purpose: calculates the eigenvalue locus with feedback gain
inputs: s1 forward loop state space object
s2 fedback state space object
gstart initial gain
gstep gain step
gend final gain
flag optional: if not supplied flag is set to 1
1 : the gain multiplies s2
2 : the gain multiplies s1
output: rl matrix of eigenvalues of the closed loop system at feedback gains of gstart to gend at steps gstep.
Note:

1. If s1 is single-input-single-output, then s2 may be a scalar.
2. Positive feedback is assumed . For negative feedback use
   `r = rtlocus(s1,-s2,gstart,gstep,gend,flag);`

Example

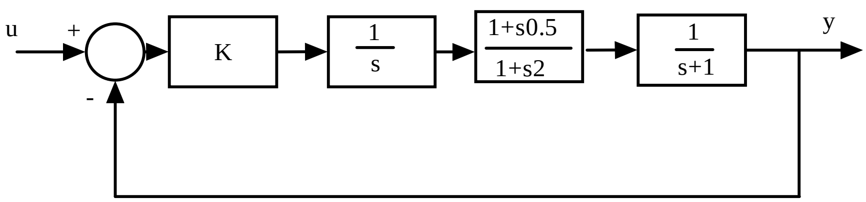


Figure 14 Example System

The forward loop state space object is formed using
```
sf = int_stsp.*ldlg_stsp(1,2,.5).*lag_stsp(1,1);
```
The zeros are
```
lz =
    -2
   Inf
   Inf
   Inf
```
There is one real zero at -2. Three zeros are at infinity. This indicates that three branches of the root locus will tend to infinity at very high values of K.
The poles of the rootlocus are the eigenvalues of sf and give the starting points of the locus.
```
l =
           -1
         -0.5
            0
```
The root locus for this system is calculated using
```
rl = rtlocus(sf,-1,0,.1,10);
```
The locus is shown in Figure 15

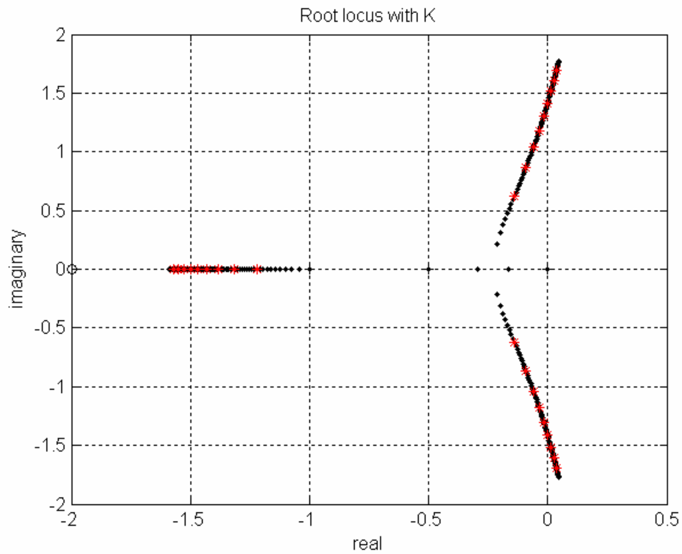


Figure 15 Root locus with K black gain interval 0.01, red gain interval 0.1

There are three locus braches which start from the systems open loop poles. The locus starting at -1 remains real and is terminated by the zero at -2. The locii which start from 0 and -.5 converge as the gain is increased, and at gains greater 0.15 there are two complex conjugate eigenvalues. At a gains higher than 6, the complex eigenvalues have a positive real part, i.e., the system is unstable.

3.35 sdecomp

syntax: `[sst,sun] = sdecomp(s,bord,fl)`
purpose: decomposes a state space system into the sum of two state space systems
inputs: s stsp object
bord defines the system partition
f1 if fl is ‘f’ the d matrix of sun is zero
output: sst has eigenvalues with real parts less than bord
sun has eigenvalues with real parts greater than bord

3.36 sparse

syntax `ss = sparse(s);`
purpose: converts the matrices s.a,s.b,s.c,and s.d to sparse storage, i.e.,

```
ss.a = sparse(s.a); ss.b = sparse(s.b); ss.c = sparse(s.c);
ss.d = sparse(s.d);
```

3.37 spy

syntax `spy(s)`
purpose: Draws a spy plot of a stsp objects a,b,c and d matrices.
example:
For the 16 machine system, spy plots of the sparse system model suc, and the full system model sps are shown in Figures 16 and 17.

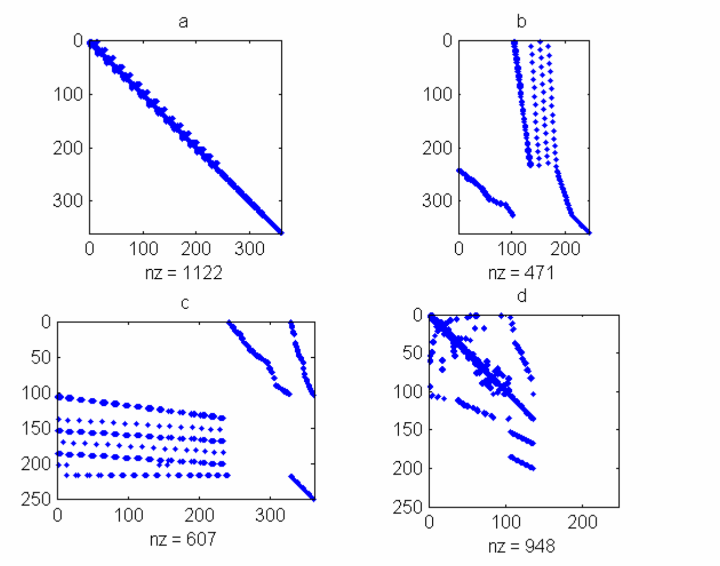


Figure 16 Spy plot of sparse model of 16 machine power system

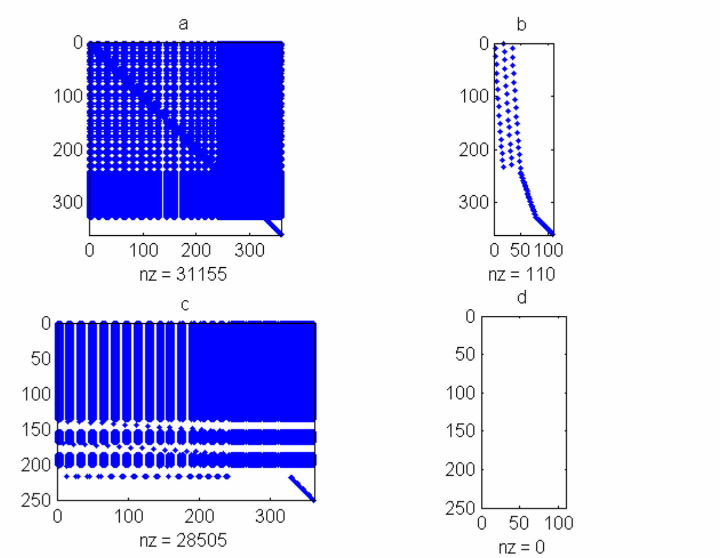


Figure 17 Spy plot of full model of 16 machine power system

3.38 stabred

syntax: `sred = stabred(s,k);`
purpose: f orms balanced residual reduction of a stable stsp object s, retaining k modes.
inputs:

	s	state space object to be reduced
	k	number of retained states
output:	sred	reduced state space object

3.39 stepres

syntax: `[res,t] = stepres(s,v_in,t_max,t_step,no_plot);`
purpose: calulates the response to a step input
inputs:

s	state space object
v_in	the magnitude of the input disturbance v_in is a column vector of disturbance input measurements with length equal to the number of inputs of s
t_max	the maximum value of the response in sec
t_step	the time step to be used in the response calculation
no_plot	1 if no plot is required. If not specified a plot is produced

outputs:

res	step response matrix of the outputs of s
t	time vector

A new figure is opened and res is plotted against t, if no_plot is not equal to 1.
With `s = lag_stsp(1,1); v_in = 1; tmax=5; t_step = 0.1`, the response is shown in Figure 16.

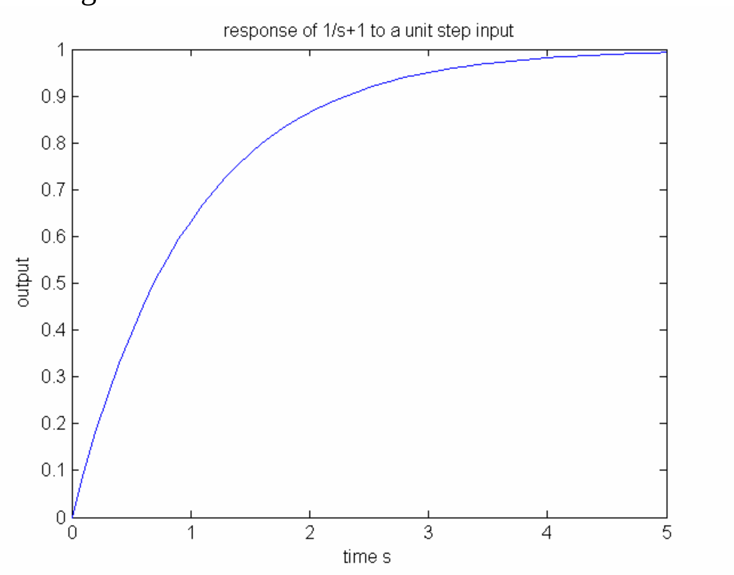


Figure 18 Calculated Response

3.40 sum_in

syntax: `si=sum_in(s,s1_idx,s2_idx,r_idx);`
purpose: sums selected inputs and retains specified inputs
inputs: s state space object
s1_idx first index of the inputs to be summed
s2_idx second index of inputs to be summed
if an entry is negative the corresponding input is subtracted
r_idx index of inputs of s to be retained
output: si modified state space object
the summed inputs of s is the first input to si
the retained inputss follow in the order of r_idx
outputs are unchanged

The inputs to si are
`[sign(s1_idx)*u(s1_idx)+sign(s2_idx)*u(s2_idx);u(r_idx)]`

3.41 sum_out

syntax: `so=sum_out(s,s1_idx,s2_idx,r_idx)`
purpose: sums selected outputs and retains specified outputs
inputs: s state space object
s1_idx first index of the outputs to be summed
s2_idx second index of outputs to be summed
if an entry is negative the corresponding output is subtracted
r_idx index of outputs of s to be retained
output: so modified state space object
the summed outputs of s are the first outputs of so
the retained outputs follow in the order of r_idx
inputs are unchanged

3.42 sysbal

syntax: `[sbal,sig] = sysbal(s,tol)`
purpose: finds a truncated balanced realization of the stsp object s.
inputs: s state space object
tol tolerance (optional)
output: sbal reduced state space object

3.43 times

syntax: `[st,in1,in2,out1,out2] = times(s1,s2,n_ic,ico_idx,ici_idx);`
purpose: over rides times for stsp objects
inputs:
s1 and s2 state space objects
n_ic number of interconnections
ico_idx index of the outputs of s1 to be connected to s2
ici_idx index of the inputs of s2 to be connected
outputs:

st	a state space object with output equal to output of s2 and any outputs of s1 not interconnected output order : retained outputs of s1;outputs of s2 input equal to the input of s1 and the inputs of s2 not interconnected input order inputs of s1; retained inputs of s2
in1	correspondence between inputs to st and inputs to s1
in2	correspondence between inputs to st and inputs to s2
out1	correspondence between outputs of st and outputs of s1
out2	correspondence between outputs of st and outputs of s2

For state space objects having the same number of inputs and the same number of outputs, the syntax $st = s1.*s2$ may be used.

3.44 tr_stsp

syntax: `res = tr_stsp(s,v_in,t,x0)`
purpose: calculates time response of a stsp object
inputs:

s	-	stsp object
v_in	-	input
t	-	time vector
x0	-	initial state vector

output res - system response to the specified input

3.45 transpose

syntax: `st = transpose(s);`
or
`st = s.';`
purpose: overrides transpose for state space objects
input: s state space object
output: st modified state space object with

```
st.a=s.a.'
st.b = s.c'
st.c = st.b'
st.d = s.d'
```

3.46 uminus

syntax: `sm = minus(s);`
or
`sm = -s;`
purpose: overrides uminus for state space objects.
input: s a state space object
output: sm a state space object having the same a,b matrices as s
the c & d matrices are the negative of those of s

3.47 vertcat

syntax: `ss = vertcat(s1,s2,s3,s4,.....sn);`
or
ss = [s1;s2;s3;s4;s5;.....sn];

purpose: forms a single stsp object from several input stsp objects
inputs: s1,s2,s3 ....,sn a set of state space objects
output: ss single state space object with
output equal to outputs of the state space objects
input equal to the inputs of the state space objects
ss.a = diag(s1.a;s2.a;s3.a;s4.a .....) etc.
Overrides vertcat for state space objects.

3.48 Zeros

syntax: `[lz,uz,vz] = zeros(s);`
purpose: calculates the transmission zeros of a stsp object
input: s a statespace object, if s is sparse it is converted to full
outputs:

lz - transmission zeros of s
uz - right eigenvectors associated with the transmission zeros
vz - left eigenvectors corresponding to transmission zeros

Forms the A and B matrices for use in **eig** in the form
[uz,lz]=eig(A,B);
where

$$A = \begin{bmatrix} s.a & s.b \\ s.c & s.d \end{bmatrix}$$

$$B = \begin{bmatrix} \mathrm{eye(s.NumStates)} & \mathrm{zeros(s.NumStates, s.NumInputs)} \\ \mathrm{zeros(s, NumOutputs, s, NumStates)} & \mathrm{zeros(s.NumOutputs, s.NumInputs)} \end{bmatrix}$$

$$\mathrm{vz} = \mathrm{inv(uz)}$$

4 State Space Constructor and Utility Functions

4.1 bode_lab

4.1.1 Purpose

Labels a bode plot

4.2 but_stsp

4.2.1 Syntax

```
s = but_stsp(fb,n)
n is the number of filter stages
```

4.2.2 Purpose

Forms a state space object representing a single input, single output, n^{th} order Butterworth filter.

4.2.3 Basis

.Butterworth filters have poles placed on a semi circle of radius f_b. The number of poles is set by the number of filter stages.
If n = 1, the filter is a lag having a pole at –2πfb, i.e.

$$\frac{dx}{dt} = -2\pi f_b x + u$$
$$y = x$$

If n = 2, the the filter has two poles: $s = -2\pi f_b(-1 \pm j1)/\sqrt{2}$
If n=5, the filter has five poles : $s_1 = -2\pi f_b;\ s_{23} = -\pi f_b\left(\sqrt{3} \pm j\right);\ s_{45} = -\pi f_b\left(1 \pm j\sqrt{3}\right)$

4.2.4 Inputs

fb - filter natural frequency
n - number of stages

4.3 cll_stsp

4.3.1 Syntax

```
s = cll_stsp(K,an,bn,ad,bd)
```

4.3.2 Purpose

Forms a state space model of a complex lead-lag transfer function

$$y = K\frac{(1+ans+bns^2)}{(1+ads+bds^2)}u$$
$$\frac{d}{dt}\begin{bmatrix}x_1\\x_2\end{bmatrix} = \begin{bmatrix}-\frac{ads}{bds} & -\frac{1}{bds}\\ 1 & 0\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix} + \begin{bmatrix}\frac{K}{bds}\\ 0\end{bmatrix}u$$
$$y = \begin{bmatrix}ans - adsbns/bds & 1 - bns/bds\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix} + \frac{Kbns}{bds}u$$

4.4 dif_stsp

4.4.1 Syntax

```
s=dif_stsp(T);
```

4.4.2 Purpose

Forms a state space model of an imperfect differentator with the transfer function

$$y = \frac{s}{(1+sT)}u$$
$$T\frac{dx}{dt} = -x + u$$
$$y = \frac{dx}{dt} = \frac{1}{T}(u-x)$$

If T is not specified, or is specified to be zero: T is assumed to be 0.01s.

4.5 dr_plot

4.5.1 Syntax

```
[x,y]=dr_plot(ws,wf,dr,col)
```

4.5.2 Purpose

Plots a line equivalent to a specified damping ratio on an argand diagram.

ws - the starting point on the y (imaginary) axis
wf - the finishing point on the y (imaginary) axis
dr - the damping ratio
col - a colour specifier: one of (‘k’, ‘r’, ‘b’, ‘g’, ‘c’, ‘m’, or ‘y’)

4.6 int_stsp

4.6.1 Syntax

```
si=int_stsp(K)
```

4.6.2 Purpose

Forms the state space model of an integrator

$$y = \frac{K}{s}u$$
$$\frac{dx}{dt} = Ku$$
$$y = x$$

4.7 labxyarg

Labels an argand diagram

4.8 lag_stsp

4.8.1 Syntax

```
sl = lag_stsp(K,Tc)
```

4.8.2 Purpose

Forms a state space model of a lag, having the transfer function

$$y = \frac{K}{(1+sT_c)}u$$
$$T_c\frac{dx}{dt} = -x + u$$
$$y = Kx$$

4.9 ldlg_stsp

4.9.1 Syntax

```
sll = ldlg_stsp(K,Td,Tn)
```

4.9.2 Purpose

Forms a state space model of a lead-lag element having the transfer function

$$y = K\frac{(1+sT_n)}{(1+sT_d)}$$

$$\frac{dx}{dt} = -\frac{1}{T_d}(x+u)$$

$$y = \frac{KT_n}{T_d}u - K\left(\frac{T_n}{T_d} - 1\right)x$$

4.10 orsf

4.10.1 Syntax

```
[u,t,k]=orsf(u,a,fl,bord)
```

4.10.2 Purpose

Given a matrix a in real Schur form, orsf finds

t = u'*a*u

with u unitary and the eigenvalues of

t are ordered

when fl= ‘o’

to have increasing real parts,

when fl = ‘s’

sorted into two groups

with the eigenvalues of the first group having

real part < bord.

The default value of bord is zero.

k is the number of poles with real part < bord

4.11 pid_stsp

4.11.1 Syntax

```
spid = pid_stsp(kp,ki,kd,Td)
```

4.11.2 Purpose

Forms a state space model of a proportional plus integral plus derrivative controller having the transfer function

$$y = k_p + \frac{k_i}{s} + \frac{k_d s}{(1 + sT_d)}$$

4.12 plot_bode

4.12.1 Syntax

```
p = plot_bode(f,ym,ya,col,uw)
```

4.12.2 Purpose

Plots a bode diagram of the frequency response defined by

f	-	vector of frequencies (Hz)
y_m	-	vector of magnitude of frequency response
y_a	-	vector of angle of frequency response (degrees)
col	-	choice of ‘k’,’b’,’r’,’g’,’y’,’m’,’c’
uw	-	if 1, unwraps phase

4.13 randngen

4.13.1 Syntax

```
x = randngen(n,m,tag)
```

4.13.2 Purpose

Generates a random number sequence using the MATLAB© functions rand or randn.

4.14 ric_eig

4.14.1 Syntax

```
[x1,x2,fail,reig_min,epkgdif] = ric_eig(ham,epp,balflg)
```

4.14.2 Purpose

Solves the eigenvalue problem associated with the stabilizing solution (A+R*X stable) of the Riccati equation

A'*X + X*A + X*R*X - Q = 0.

An eigenvalue decomposition is used to obtain the stable
invariant subspace of the Hamiltonian matrix, ham,
which contains the above variables in the following format:

ham = [A R; Q -A'].
If ham has no eigenvalues on the imaginary axis, there exist n x n matrices
x1 and x2 such that [x1 ; x2] spans the n-dimensional
stable-invariant subspace of ham.
If x1 is invertible, then
X = x2/x1 satisfies the Riccati equation,
and A+RX stable.
fail is returned with a value of 1 if there are eigenvalues of ham on the imaginary axis, or having a real part less than epp (default 1e-10)

reig_min is the minimum real part of the eigenvalues.
balflg = 0 balances ham , while if balflg is non-zero ham is unchanged.setting.The default is 0.
If ham is not diagonalizable, it is better to use ric_schr.

epkgdif is a comparison of two different imaginary axis tests.

4.15 ric_schur

4.15.1 Syntax

```
[r1,r2,fail,reig_min,epkgdif] = ric_schr(ham,epp,balflg)
```

4.15.2 Purpose

Solves the eigenvalue problem associated with the stabilizing solution (A+R*X stable) of the Riccati equation

A'*X + X*A + X*R*X - Q = 0.

A real Schur decomposition is used to obtain the stable
invariant subspace of the Hamiltonian matrix, HAM,
which contains the above variables in the following format:
ham = [A R; Q -A'].

If ham has no imaginary axis eigenvalues, there exist n x n matrices
x1 and x2 such that [x1 ; x2] spans the n-dimensional
stable-invariant subspace of ham. If x1 is invertible, then
x = x2/x1 satisfies the Riccati equation, and A+Rx is stable.
fail = 1, if all eigenvalues have non-zero real parts
fail = 2, if there are an unequal number of positive and negative eigenvalues
fail = 3, if both conditions occur
reig_min - minimum absolute value of the real parts of the eigenvalues
balflg is set to 0 to balance ham and set to 1 otherwise: the default is 0.
epkgdif gives a comparison of two different imaginary axis tests.

4.16 sjh6

4.16.1 Syntax

```
u = sjh6(a,c)
```

4.16.2 Purpose

Solves the Lyapunov equation

$$a'x + xa + c'c = 0$$

u is the Cholesky factor, i.e. u is upper triangular and x = u'u

4.17 stsp2ss

4.17.1 Syntax

```
ss = stsp2ss(s)
```

4.17.2 Purpose

Converts a stsp object to a MATLAB© Control System Toolbox ss object

4.18 stsp2tf

4.18.1 Syntax

```
t = stsp2tf(s)
```

4.18.2 Purpose

Converts a stsp object to a MATLAB© Control System Toolbox transfer function object

4.19 svp_lab

4.19.1 Syntax

```
s=svp_lab
```

4.19.2 Purpose

Labels a singular value frequency response plot.

4.20 wo_stsp

4.20.1 Syntax

```
sw=wo_stsp(T)
```

4.20.2 Purpose

Forms a state space model of a washout filter having the transfer function

$$y = \frac{sT}{1+sT}u$$