FOSSEE SUMMER INTERNSHIP

IIT BOMBAY, 2017

Project Report

Control System Toolbox

Team member

Ayush Kumar

Project Mentor

1.Shamika Mohanan

2. Ashutosh Kr. Bhargava

Summer Internship 2017 Project Approval Certificate

The project entitled "Control System Toolbox" submitted by Ayush Kumar, pursuing BTech from **IIT HYDERABAD** is approved for Summer Internship 2017 programme from 22th May 2017 to 10th July 2017, at FOSSEE, IIT Bombay. During his internship, we found him hard working, sincere and diligent person and his behaviour and conduct was good. We wish him all the best for his future endeavours.

………………………….	…………………….
Prof. Kannan Moudgalya	**Shamika Mohanan**
(Professor Incharge)	(Project Mentor)

…………………………….

Ashutosh Kr. Bhargava
(Project Mentor)

ACKNOWLEDGEMENT

I, Ayush Kumar working on “Control System Toolbox”, most gratefully acknowledge my deep gratitude to all those who helped me to achieve my objective with perfection and assigned me tasks which helped me achieve my hidden potential.

I, wholeheartedly thank Prof. Kannan Moudgalya for having faith in me by giving me the opportunity to participate in the internship for this venerable institution and constantly motivating me to do better.

I would like to thank Ashutosh Kr. Bhargava Sir and Inderpreet Arora ma'am and Shamika mohanan ma'am for guiding me throughout the internship. Without your inputs and support this project could not have been possible.

ABSTRACT

Scilab is free and open source software for numerical computation providing a powerful computing environment for engineering and scientific applications. Scilab is released as open source under the CeCILL license (GPL compatible), and is available for download free of charge. It is an initiative by the FOSSEE to promote the use of scilab. This project involves around developing the “control system toolbox” which mainly includes adding new functions in CACSD module of scilab and analysing and comparing their results with existing matlab function, testing some existing functions and creating test cases.

Scilab version 6.0.0 is used.

Contents

1.1 Balred

(Model Order Reduction[1.1.1])

Significance :- Many modern mathematical models of real-life processes pose challenges when used in numerical simulations, due to complexity and large size (dimension). **Model order reduction**[1.1.2] aims to lower the computational complexity of such problems, for example, in simulations of large-scale LTI systems in control systems. By a reduction of the model's associated state space dimension or degrees of freedom, an approximation to the original model is computed. This reduced-order model (ROM) can then be evaluated with lower accuracy but in significantly less time.

Syntax :-

Sysout = balred(sysin,k)

Input argument : sysin(lti system in state-space or rational form)
k(order to which the system is to be reduced)

Output argument : sysout(reduced system of order k)

Description :- sysout = balred(sysin,k) computes a reduced-order approximation sysout of the LTI model sys. The desired order (number of states) for rsys is specified by n,it can be a number(positive) or a vector or matrix if we want to try multiple orders at once,in this case sysout returned will be a cell array of reduced-order models. Balred uses implicit balancing techniques to compute the reduced- order approximation sysout.
When sysin has unstable poles, it is first decomposed into its stable and unstable parts using stabsep, and only the stable part is approximated.

The order of the approximate model is always at least the number of unstable poles and at most the minimal order of the original model.In case input n is less than the order of unstable part,stable part of sysout is reduced to a zero order system and added with unstable part.

Methord:

Numerous algorithms are available for computing the balanced reduced system some of them which i have tried are BFSA, Balanced Truncation and BSPA, the below mentioned methord uses BSPA[1.1.3] (Balanced Singular Perturbation Approximation methord).

1.Solving the lyapunov equation to get Controllability(P) and Observability grammians(Q).

$AP + PA' + BB' = 0$ and (for continuous time)

$A'Q + QA + C'C = 0$ (P=ctr_gram(sysin),Q=obs_gram(sysin))

$APA' + BB' = P$ and (for discrete time system)

$A'QA + C'C = Q$

where (A,B,C,D are state-spaces of a continuous or discrete time system)

2.Obtaining balanced minimal realization of system[1.1.4]

2.1 Obtaining orthogonal transformation Uc and Uo such that

$P = Uc \times Sc \times Uc^{T}$ and $Q = Uo \times So \times Uo^{T}$

(Sc,So are diagonal matrices)

2.2 $H = So^{1/2} \times Uo^{T} \times Uc \times Sc^{1/2}$

2.3 Obtaining a SVD decomposition of H such that $H = U_H \times S_H \times V_H^{T}$

2.4 $T = Uo \times So^{-1/2} \times U_H \times S_H^{1/2}$

2.5 $A = T^{-1} \times A \times T$; $B = T^{-1} \times B$; $C = T \times C$; $D = D$ (balanced minimal system)

3. Partition the system such that

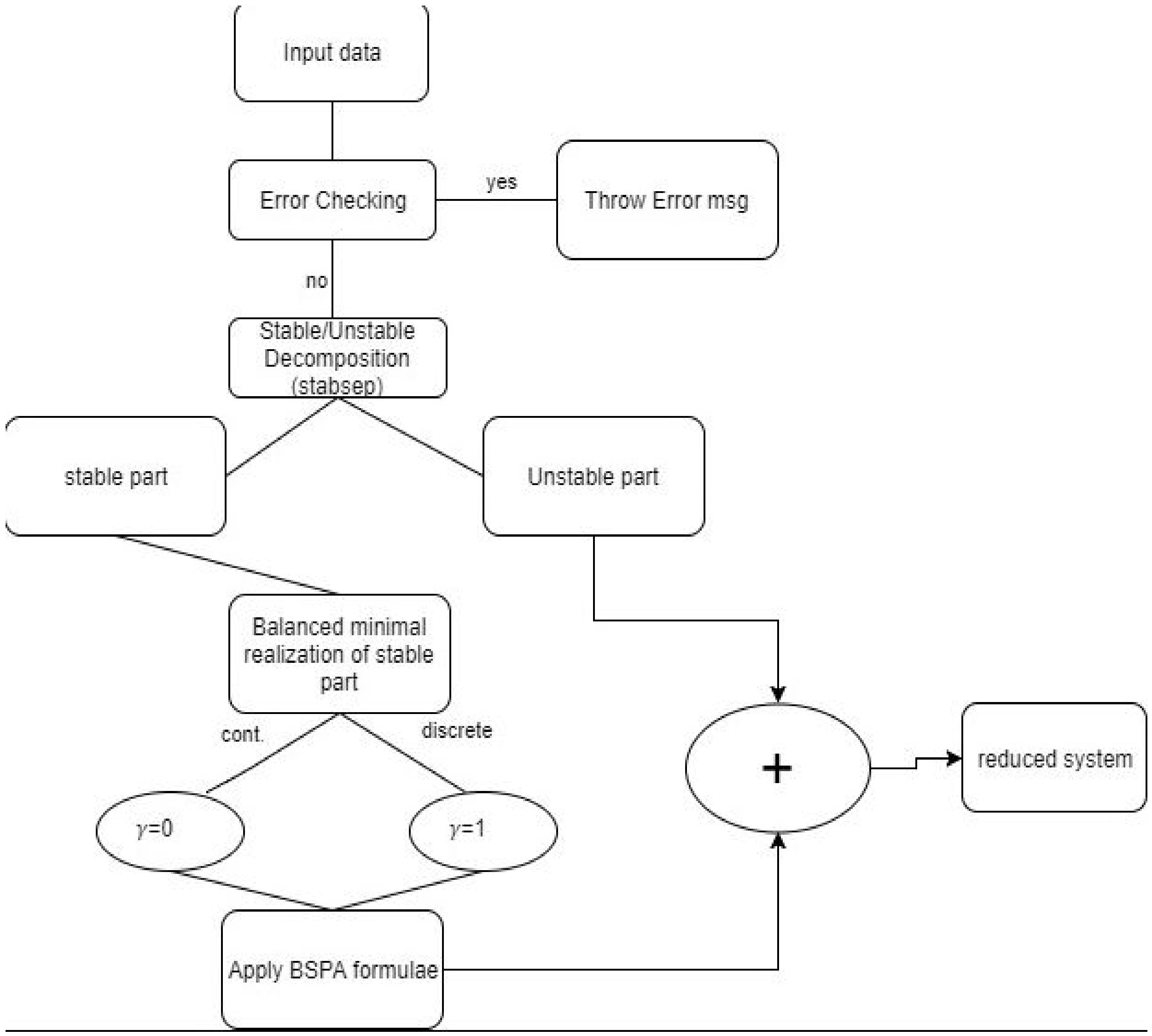
$$G = \left[\begin{array}{cc|c} \bar{A}_{11} & \bar{A}_{12} & \bar{B}_1 \\ \bar{A}_{21} & \bar{A}_{22} & \bar{B}_2 \\ \hline \bar{C}_1 & \bar{C}_2 & D \end{array}\right]$$

$$\bar{A} = \tilde{A}_{11} + \tilde{A}_{12}(\gamma I - \tilde{A}_{22})^{-1}\tilde{A}_{21}$$
$$\bar{B} = \tilde{B}_1 + \tilde{A}_{12}(\gamma I - \tilde{A}_{22})^{-1}\tilde{B}_2$$
$$\bar{C} = \tilde{C}_1 + \tilde{C}_2(\gamma I - \tilde{A}_{22})^{-1}\tilde{A}_{21}$$
$$\bar{D} = D + \tilde{C}_2(\gamma I - \tilde{A}_{22})^{-1}\tilde{B}_2$$

4.Reduced system is given by :

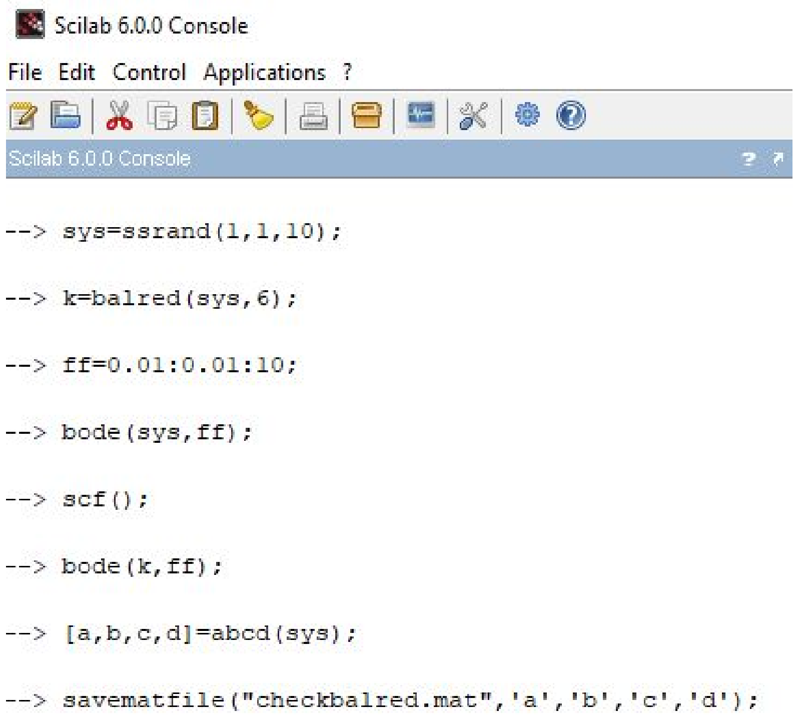
where γ=0 for continuous system and γ=1 for discrete system,above system(A,B,C,D) is the reduced order approximated system.

Flow chart :-



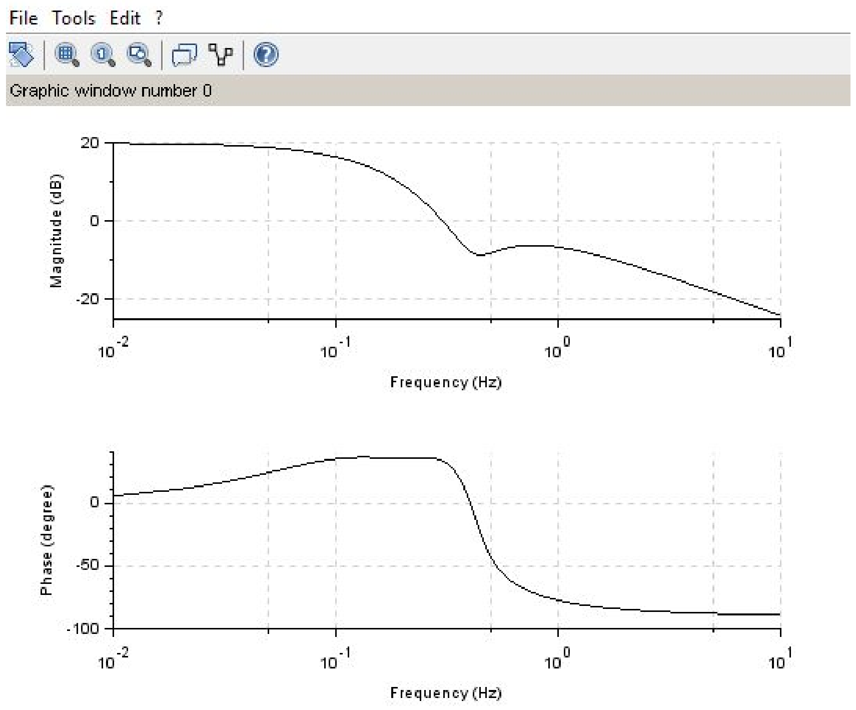
Example :-

Comparing the results in matlab and scilab

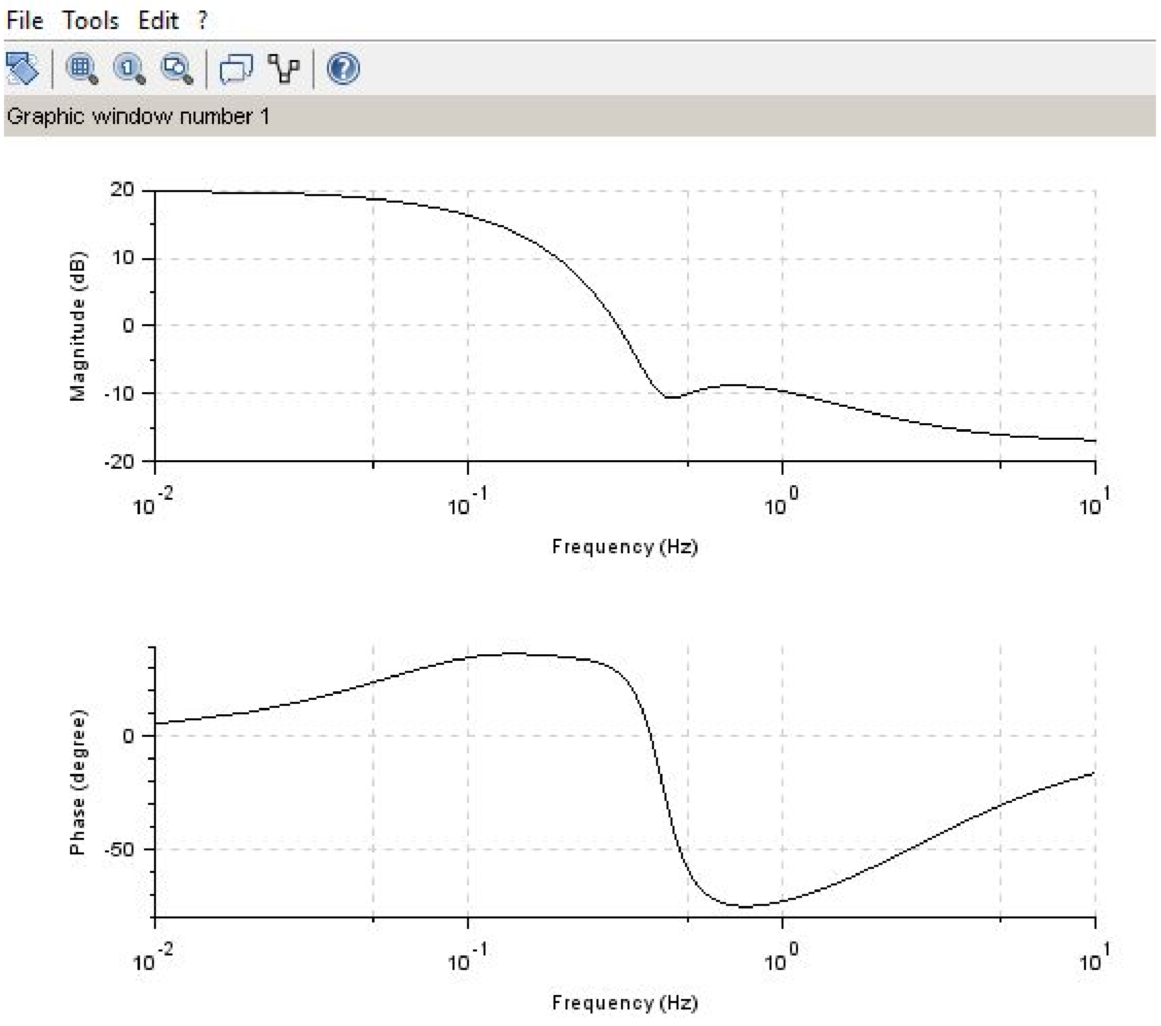


```
--> sys=ssrand(1,1,10);

--> k=balred(sys,6);

--> ff=0.01:0.01:10;

--> bode(sys,ff);

--> scf();

--> bode(k,ff);

--> [a,b,c,d]=abcd(sys);

--> savematfile("checkbalred.mat",'a','b','c','d');
```

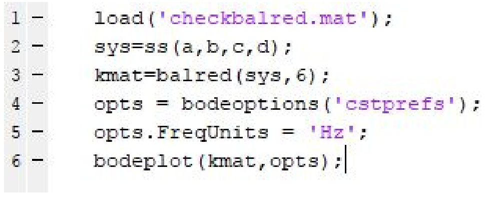
Bode plot Original 10th order system:-



Scilab result:-

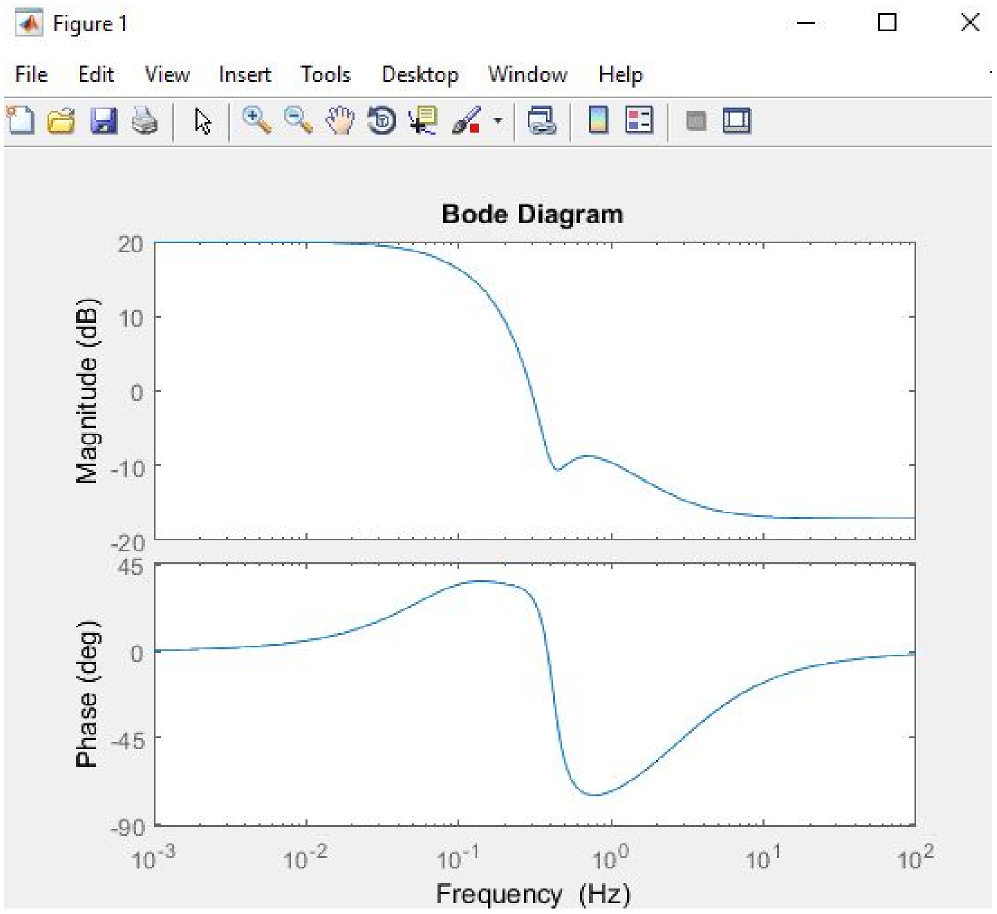


Loading data in matlab



```
load('checkbalred.mat');
sys=ss(a,b,c,d);
kmat=balred(sys,6);
opts = bodeoptions('cstprefs');
opts.FreqUnits = 'Hz';
bodeplot(kmat,opts);
```

Matlab Result :-



1.2 Stabsep

(Decomposition of a system into stable and unstable part)

Significance :- In the context of reduced order modeling generally it is required to decompose[1.2.1] an unstable system into two components. These two components separate the dynamics of the given system into stable and unstable parts.
The beauty of the algorithm is that the method presented is numerically a stable algorithm which separates into two independent (decoupled) subsystems having all the eigenvalues same as that of original system

Syntax:- [ga,gs]=stabsep(sys)

Existing function:- dtsi() [1.2.2]function exists in scilab which is equivalent to stabsep function,but it works only for continuous time state space system while implemented stabsep system works for both continuous system and discrete time state space or rational model system.

Description:- It decomposes the given lti system sys into stable and unstable part such that sys=ga+gs [1.2.3],where ga is unstable part and gs is stable part of the system.

Implementation methord :-

1. Initial system : $G = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right]$

2. Obtaining a unitary matrix U such that such that the system is transformed in upper block diagonal schur form.

i.e A=U, x Ax U ; B = U' x B ; C=C x U ; D=D;

the resultant transformation matrix U will transform the state matrix A into upper block diagonal schur form.

First stage transformation:-

$$G_t = \left[\begin{array}{c|c} U'AU & U'B \\ \hline CU & D \end{array}\right] = \left[\begin{array}{c|c} A_t & B_t \\ \hline C_t & D \end{array}\right] = \left[\begin{array}{cc|c} A_{t11} & A_{t12} & B_{t1} \\ O & A_{22} & B_{t2} \\ \hline C_{t1} & C_{t2} & D \end{array}\right]$$

where, m (rows in A_{t11}) denotes the number of stable eigenvalues, and n(rows in A_{t22}) denotes the number of unstable eigenvalues.

Here the transformed matrix accommodates in such a way that all the stable eigenvalues of the system are contained in left upper block (A_{t11}) and all the unstable eigenvalues are contained in left lower block(A_{t22}).

2.The transformed system in, contains a coupling term(At12) . Therefore to bring it into completely decoupled form, we solve the general form of lyapunov equation

Solving further the general form of lyapunov equation to obtain S

$$A_{t11}S - SA_{t22} + A_{t12} = 0$$

3.Then we proceed to second stage of transformation,where we create W matrix as

$$W = \begin{bmatrix} I_m & : & S \\ \text{....} & : & \text{....} \\ O & : & I_n \end{bmatrix}$$

Where I_m and I_m are identity matrices of size m and n.

4.The completely decoupled system is then given as

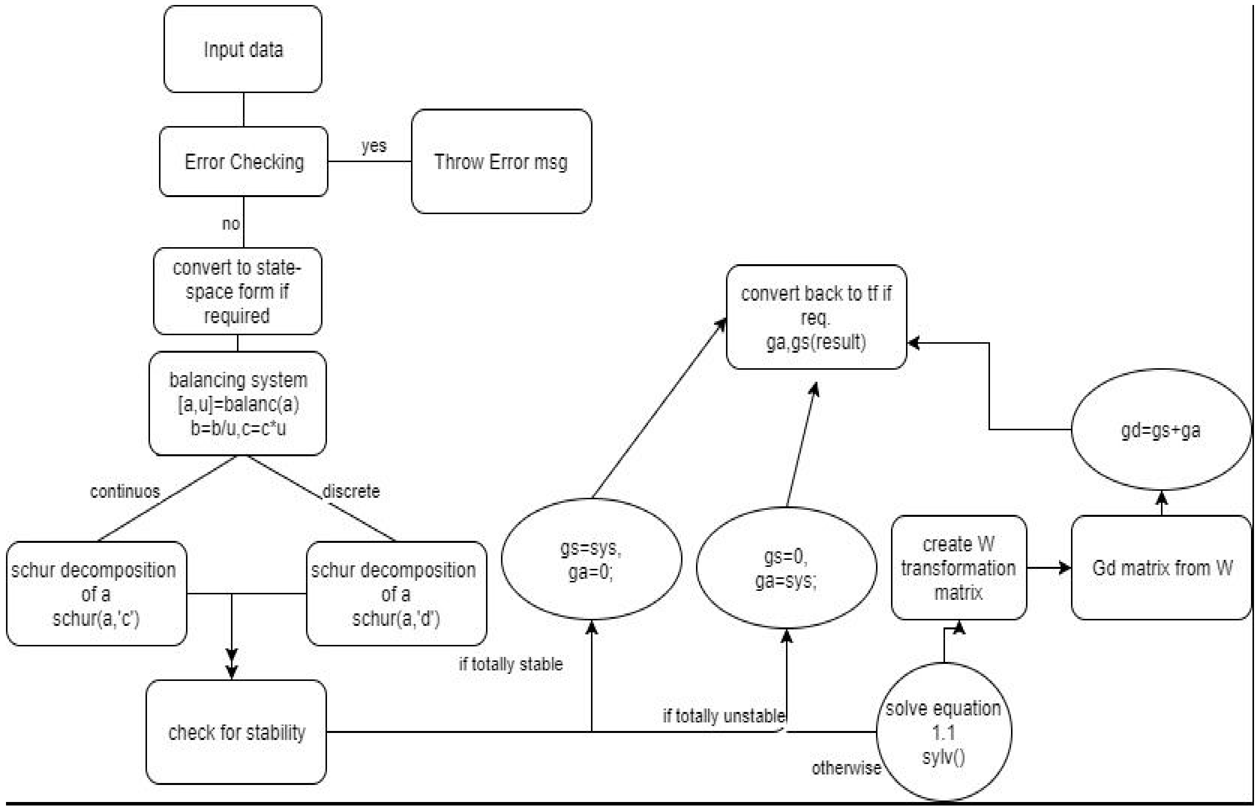
$$G_d = \left[\begin{array}{c|c} W^{-1}A_tW & W^{-1}B_t \\ \hline C_tW & D \end{array}\right] = \begin{array}{c} \begin{array}{cc} \overset{m}{\longleftrightarrow} & \overset{n}{\longleftrightarrow} \end{array} \\ \left[\begin{array}{cc|c} a_{11} & O & b_1 \\ O & a_{22} & b_2 \\ \hline c_1 & c_2 & D \end{array}\right] \end{array} \begin{array}{l} \updownarrow m \\ \updownarrow n \\ \\ \end{array}$$

5.Which can be further decomposed into stable & unstable subsystems as :

$$G_d = \left[\begin{array}{c|c} a_{11} & b_1 \\ \hline c_1 & D \end{array}\right] + \left[\begin{array}{c|c} a_{22} & b_2 \\ \hline c_2 & 0 \end{array}\right]$$

Stable part unstable part

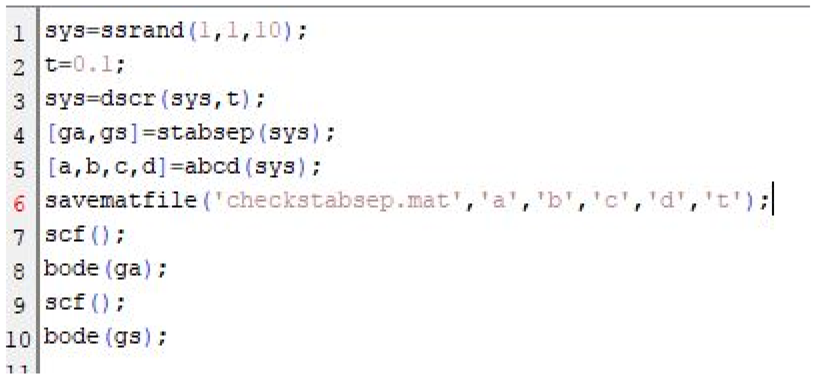
Program flow :-



Input data
Error Checking
yes
Throw Error msg
no
convert to state-space form if required
balancing system [a,u]=balanc(a) b=b/u,c=c*u
continuos
discrete
schur decomposition of a schur(a,'c')
schur decomposition of a schur(a,'d')
check for stability
if totally stable
gs=sys, ga=0;
if totally unstable
gs=0, ga=sys;
otherwise
solve equation 1.1 sylv()
create W transformation matrix
Gd matrix from W
gd=gs+ga
convert back to tf if req. ga,gs(result)

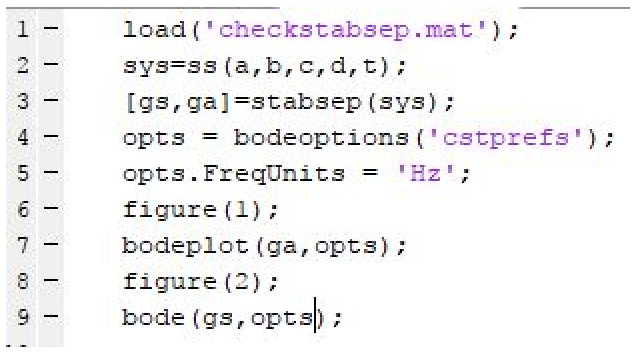
Example :

Creating a random 10th order discrete lti system in scilab and comparing the results of matlab and scilab from bode plot.



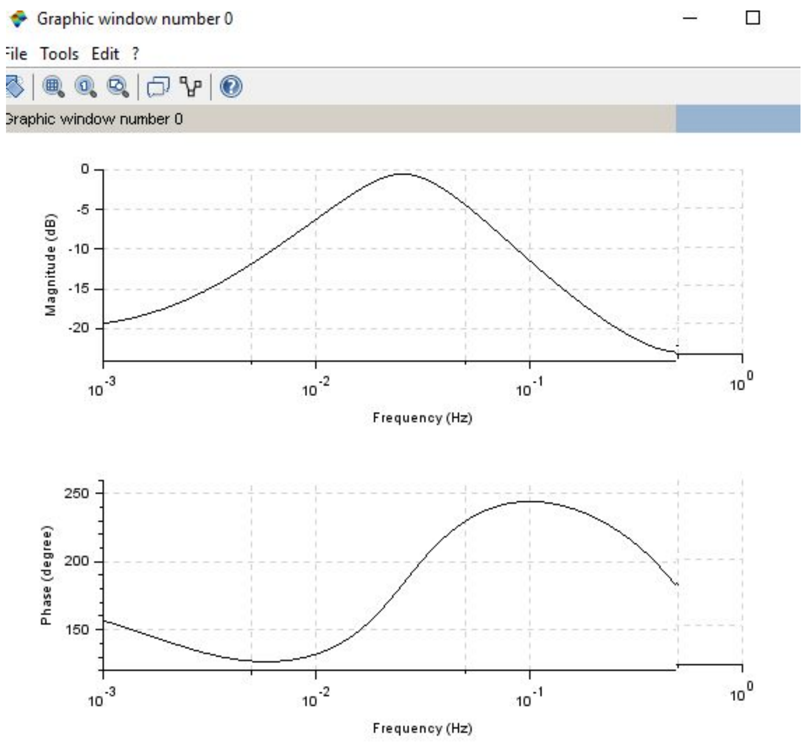
```
sys=ssrand(1,1,10);
t=0.1;
sys=dscr(sys,t);
[ga,gs]=stabsep(sys);
[a,b,c,d]=abcd(sys);
savematfile('checkstabsep.mat','a','b','c','d','t');
scf();
bode(ga);
scf();
bode(gs);
```

Loading data and plotting bode graphs in matlab :

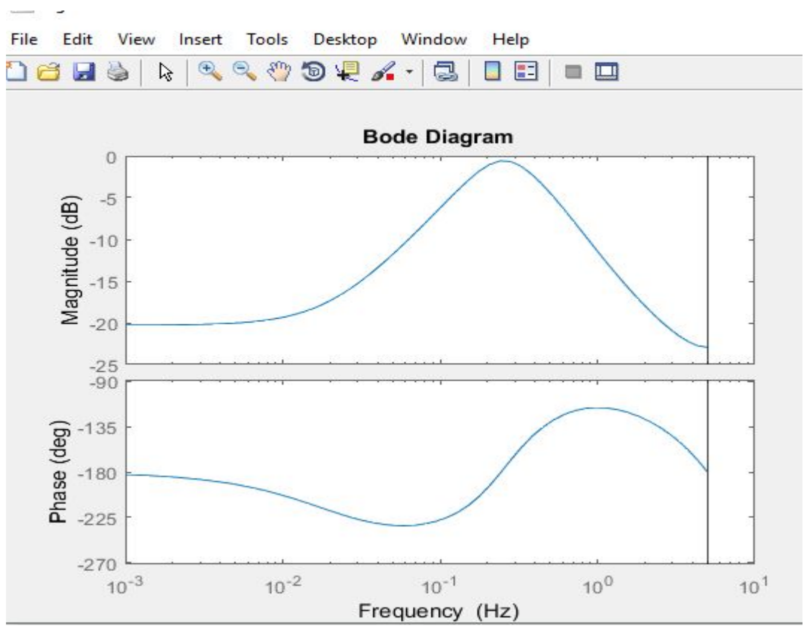


```
load('checkstabsep.mat');
sys=ss(a,b,c,d,t);
[gs,ga]=stabsep(sys);
opts = bodeoptions('cstprefs');
opts.FreqUnits = 'Hz';
figure(1);
bodeplot(ga,opts);
figure(2);
bode(gs,opts);
```

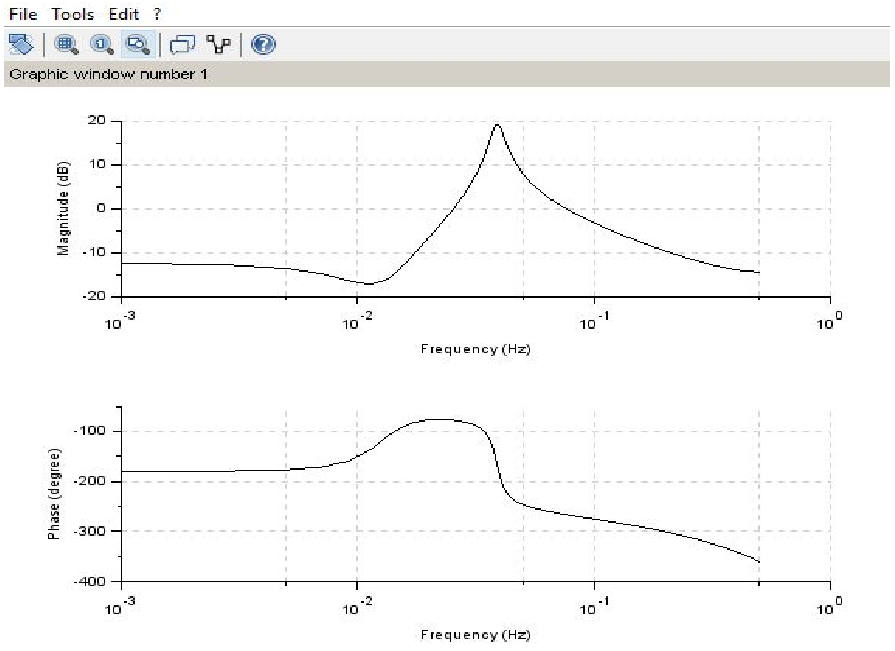
Bode plot of unstable part scilab



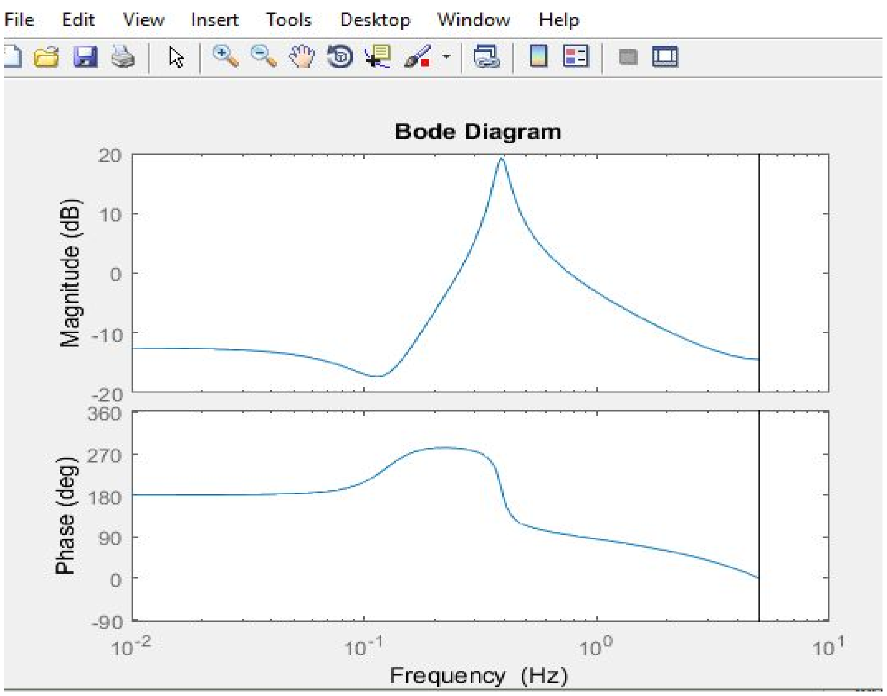
bode plot of unstable part matlab



Bode plot of stable part scilab



Bode plot of stable part matlab



1.3 Hankel

(Squared Hankel Singular values)

Significance :-

In control theory, **Hankel singular values[1.3.1]**, named after Hermann Hankel, provide a measure of energy for each state in a system. They are the basis for balanced model reduction, in which high energy states are retained while low energy states are discarded. The reduced model retains the important features of the original model.

Hankel singular values are calculated as the square roots, $\{\sigma_i \geq 0, i = 1,\ldots,n\}$, of the eigenvalues, $\{\lambda_i \geq 0, i = 1,\ldots,n\}$, for the product of the controllability Gramian, and the observability Gramian.

Syntax :- [nk,W]=hankel(sys,tol)

Input argument : sys :- linear system

Tol:tolerance parameter for detecting imaginary axis modes (default value is 1000*%eps)

Output argument :- nk :- squared hankel singular values

W=Product of grammians = P x Q

Description :- Returns the squared hankel singular[1.3.2] values of the system sys (a stable or unstable,continuous or discrete time system) as a vector in ascending order and W as Product of the controllability and observability grammians of the system stable part of the system.

Controllability grammian :- It is a way to measure whether a given system sys is controllable or not ,where controllability is defined as it describes (as quoted in wikipedia)the ability of an external input (the vector of control variables) to move the internal state of a system from any initial state to any other final state in a finite time interval.The controllability Gramian can be found as the solution of the Lyapunov equation given by :-

AP + PA’+ BB’= 0 (continuous)

APA’ + BB’ = P (discrete)

Observability grammian :It is a way to measure whether a given system sys is observable or not,A system with an initial state, x(t0) is **observable** if and only if the value of the initial state can be determined from the system output y(t) that has been observed through the time interval t0 < t < <t(f). If the initial state cannot be so determined, the system is **unobservable**.The observability Gramian can be found as the solution of the Lyapunov equation given by

A’Q + QA + C’C = 0 (Continuous system)

A’QA + C’C = Q (discrete time system)

Implementation :

1.Decompose the input system to stable and unstable part using stabsep function.

2.Obtain a balanced minimal realization of the stable part of the system.(balreal(gs))

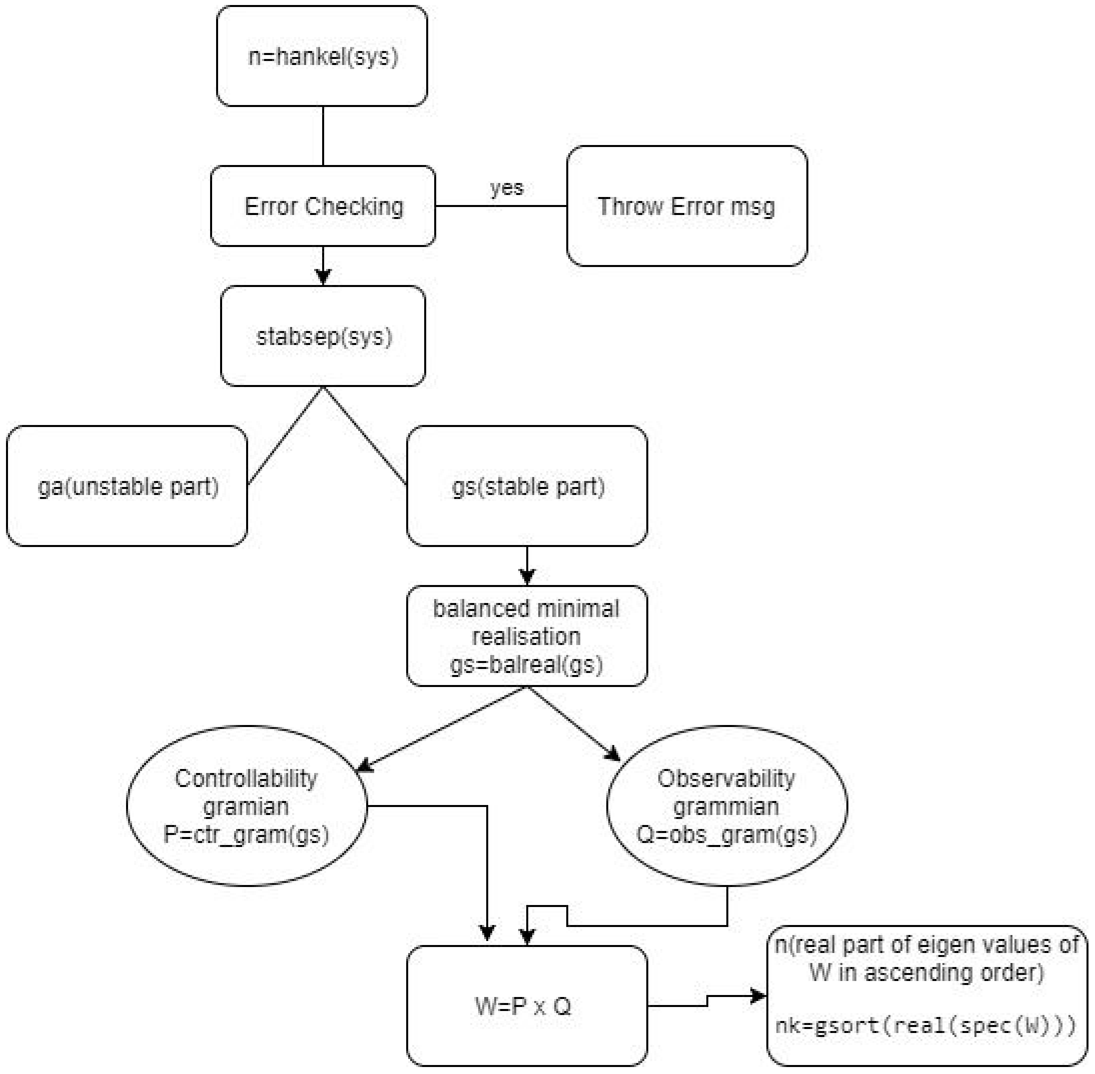
3.Compute the controllability grammian P and Observability grammian Q.

4.W=P xQ

5.squared hankel singular values will be the eigenvalues of W in sorted order.

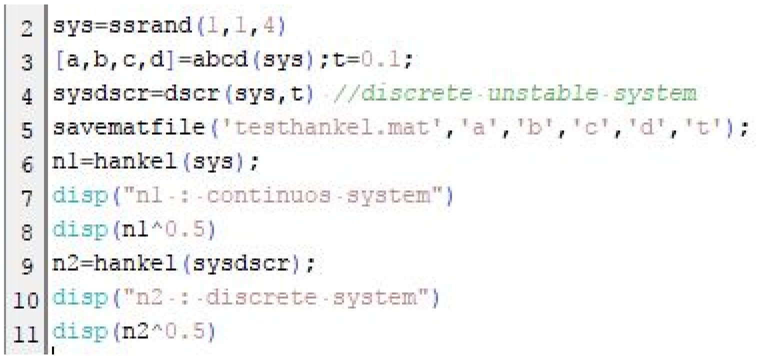
Note : Existing hankelsv() [1.3.3] function in scilab computes squared hankel singular values for a stable continuous time system while above mentioned hankel () function works fine for both stable as well as unstable ,continuous or discrete time system,as shown in the example.

Function flowchart :-


n=hankel(sys)
Error Checking
yes
Throw Error msg
stabsep(sys)
ga(unstable part)
gs(stable part)
balanced minimal realisation gs=balreal(gs)
Controllability gramian P=ctr_gram(gs)
Observability grammian Q=obs_gram(gs)
W=P x Q
n(real part of eigen values of W in ascending order)
nk=gsort(real(spec(W)))

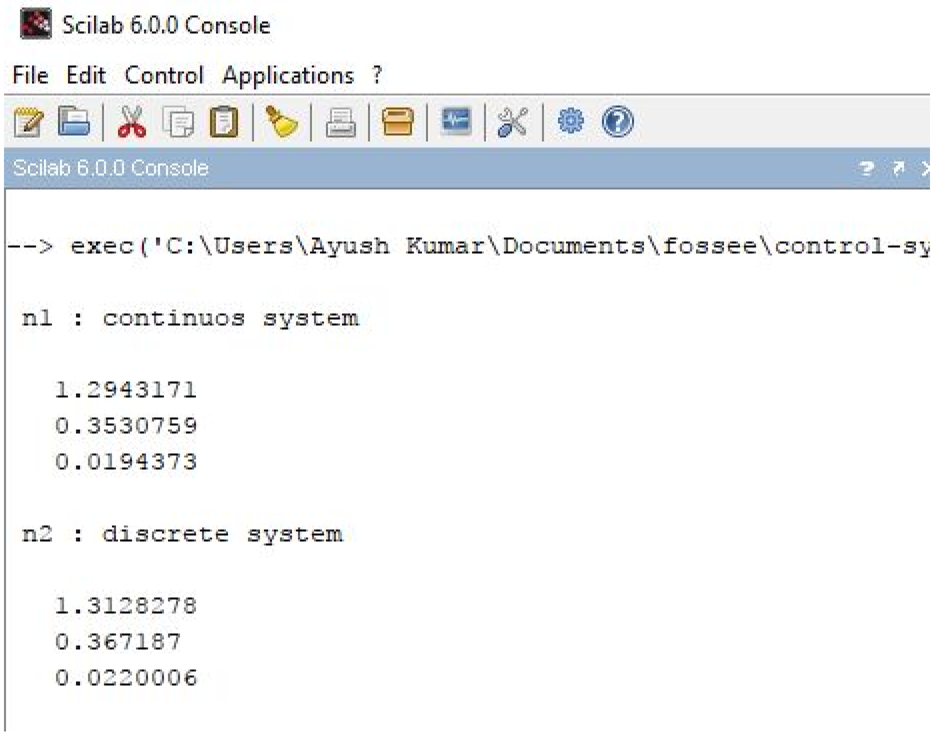
Example :-

Here we are comparing the hankel singular value for a unstable system and discretizing it further and comparing the results again.Square root is taken as hankel( ) mentioned above returns squared hankel values.



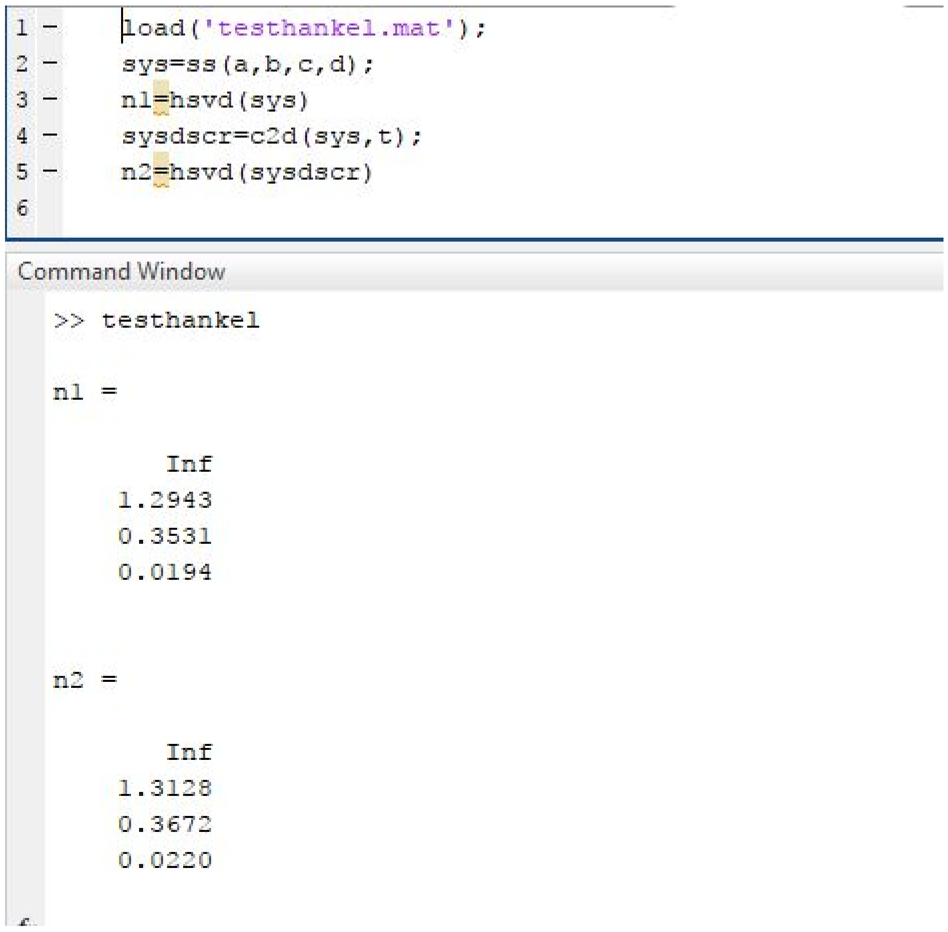
```
sys=ssrand(1,1,4)
[a,b,c,d]=abcd(sys);t=0.1;
sysdscr=dscr(sys,t) //discrete unstable system
savematfile('testhankel.mat','a','b','c','d','t');
n1=hankel(sys);
disp("n1 : continuos system")
disp(n1^0.5)
n2=hankel(sysdscr);
disp("n2 : discrete system")
disp(n2^0.5)
```

Scilab result



```
--> exec('C:\Users\Ayush Kumar\Documents\fossee\control-sy

 n1 : continuos system

   1.2943171
   0.3530759
   0.0194373

 n2 : discrete system

   1.3128278
   0.367187
   0.0220006
```

Matlab result :



```
load('testhankel.mat');
sys=ss(a,b,c,d);
n1=hsvd(sys)
sysdscr=c2d(sys,t);
n2=hsvd(sysdscr)
```

Command Window

```
>> testhankel

n1 =

       Inf
    1.2943
    0.3531
    0.0194


n2 =

       Inf
    1.3128
    0.3672
    0.0220
```

1.4 HSVPLOT :-

(Plot hankel singular values and return plot handles)

Syntax :

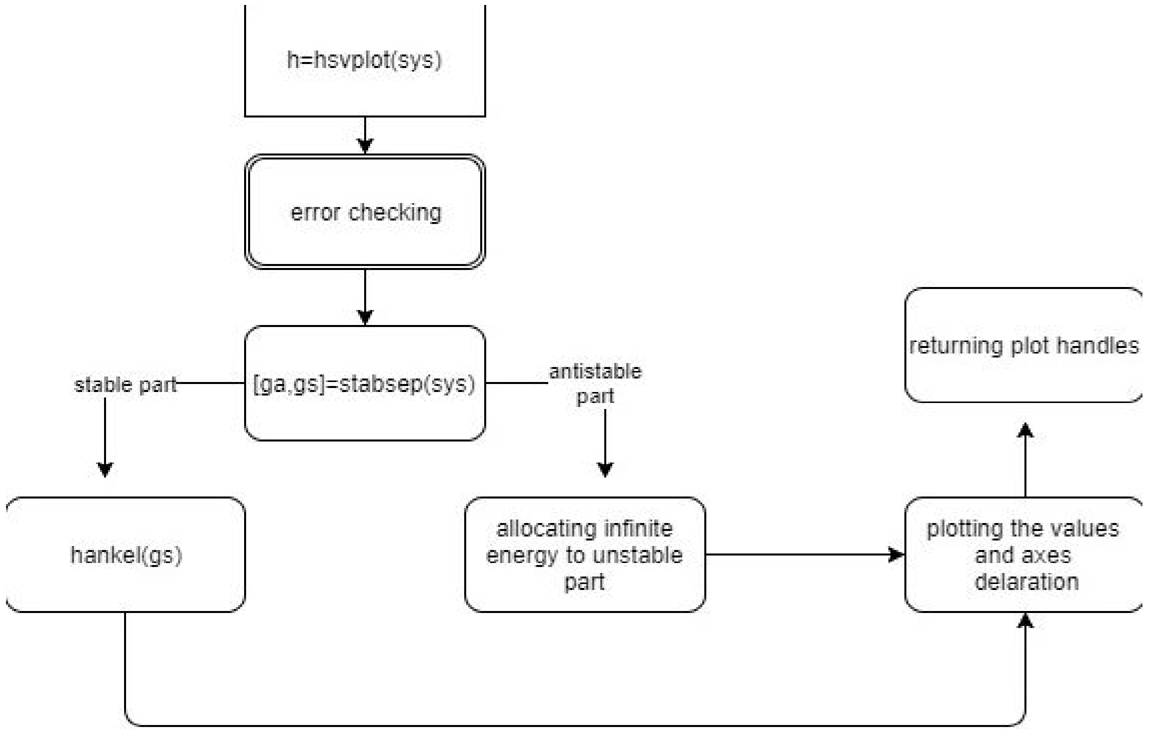
[h]=hsvplot(sys)

hsvplot(sys)

Description :

h=hsvplot(sys)[1.4.1] plots the Hankel singular values of a given lti system and returns the plot handles h which can be used to further customize the plot,in case system has unstable part,the energies associated with the unstable part has relatively much larger energies than that associated with stable part and hence are allocated infinite energy,these infinite energies are shown as zeroes left to the first bar of stable modes in the plot.

Function flowchart :

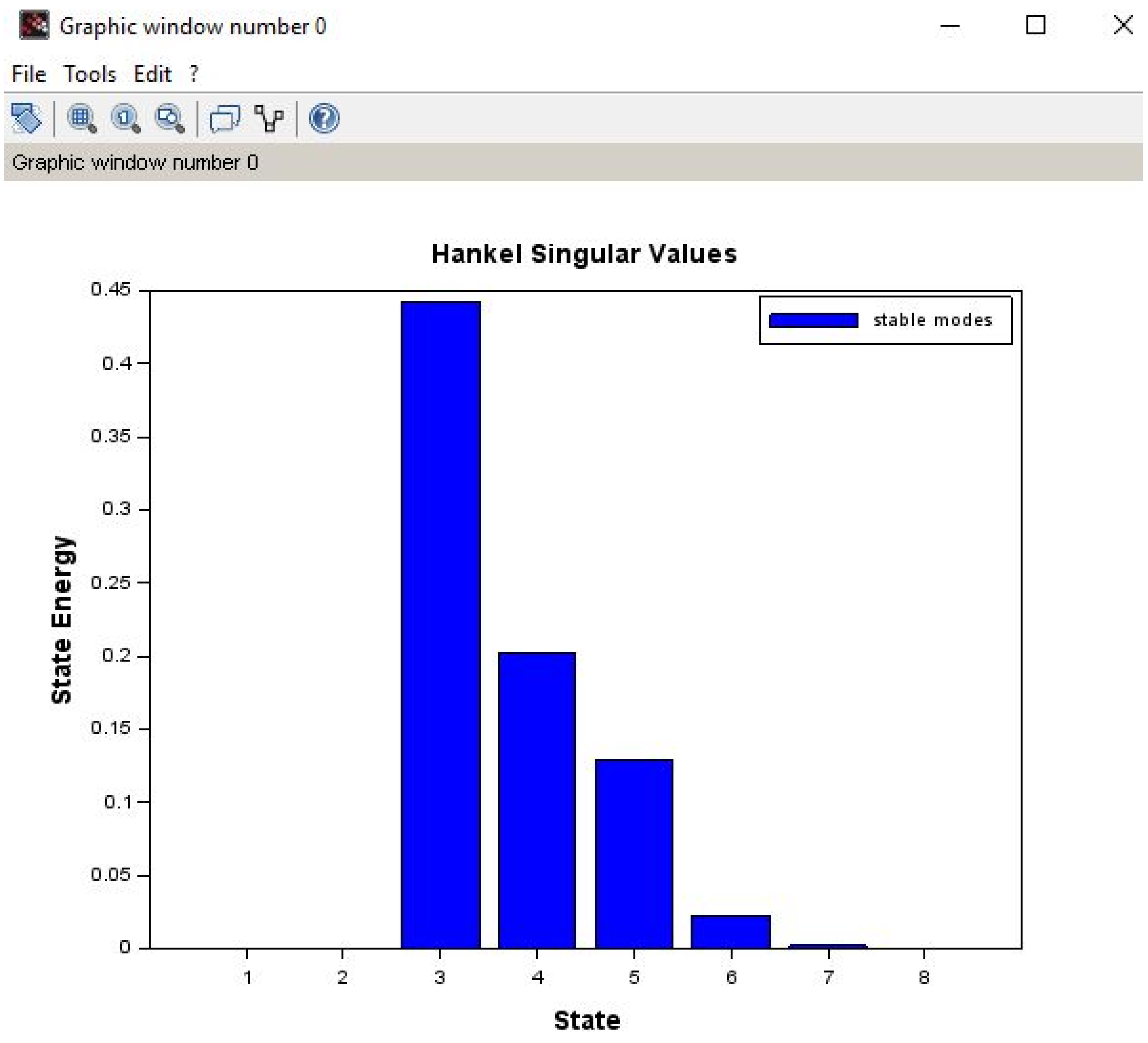


Example :

A discrete 8th order unstable system.

```
sys=ssrand(1,1,8)
sys=dscr(sys,0.1);
h=hsvplot(sys)
```

hsvplot:



1.5.1 RBS :-

(random binary signal generator)

Significance :- A **random binary sequence[1.5.1]** (RBS) is a binary sequence that, while generated with a deterministic algorithm, is difficult to predict and exhibits statistical behavior similar to a truly random sequence. RBS are used in telecommunication, encryption, simulation, correlation technique and time-of-flight spectroscopy.In control systems this generated signal can be used as input signal to determine the output behaviour of a given system or an unknown system model,or in system identification.

Syntax :-

u=idinput(N);

u=idinput([n,nu])

u=idinput([n,nu,m])

u=idinput(__,type)

u=idinput(__,type,band)

u=idinput(__,type,band,levels)

Description :-

u = idinput(n) returns a single-channel random binary signal u of length N. By default it's values are either -1 or 1.

u = idinput([n,nu]) returns an nu-channel random binary input signal, where each channel signal has length n. The signals in each channel are independent from each other.

u = idinput([n,nu,m]) returns an Nu-channel periodic random binary input signal with specified period and number of periods. Each input channel signal is of length m*n.

u=idinput(___,type) specifies the type of input to be generated as one of the following:

'rbs' — Random binary signal

Currently it's implemented for rbs only can be further extended to 'rgs','prbs' etc.

u =idinput(**___**,type,band) specifies the frequency band of the signal.

u =idinput(**___**,type,band,levels) specifies the level of the binary generated signal

Implementation :-

Several algorithms are available for generating a random normally distributed signal.

I tried two of them :-

1. LCG algorithm[1.5.2] :- X(n+1)=(aX(n)+c)% m.

Here in our case m=2^32,a=1664525,c=1013904223 ,

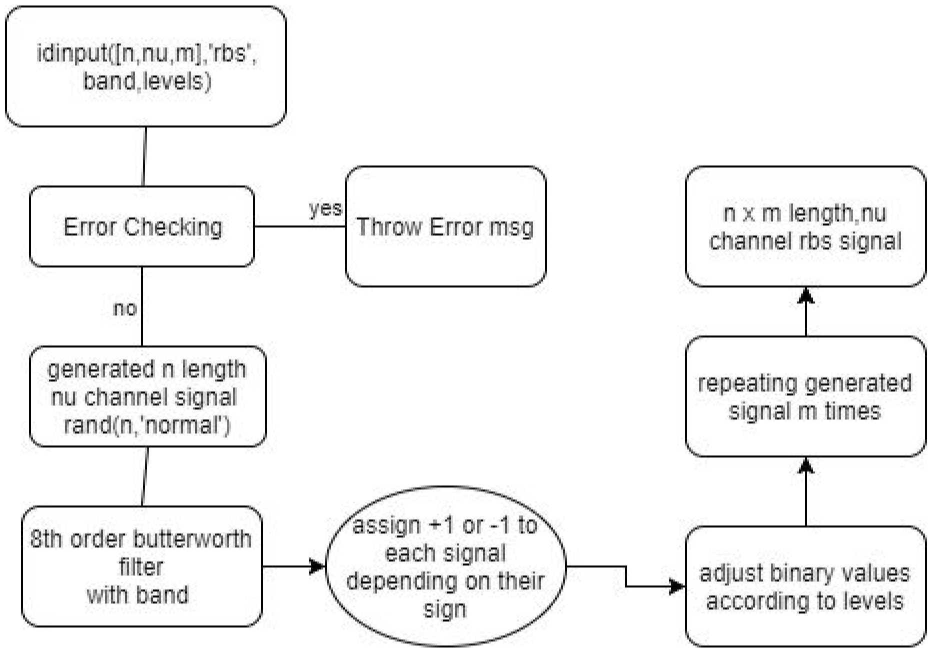
X(0) current time in millisec taken as seed.

Although this methord also generate random binary signals which are almost close to a random process and the generated binary signals are totally independent of each other,but taking into consideration the calculation complexity this approach is far more slower and takes a lot more time in case we want n to be >1000.

2.Using rand function with current time in millisec as seed to generate normally distributed signal,so that each time it generates a different signal.This implementation methord computes results quicker.

The generated signal is then passed through an '8th' order butterworth filter which acts as a low pass high pass or bandpass filter as the case may be.Butterworth filter is defined as another function idfilt(data,band,nu) ,where data is the signal,band :-band of the filter ,nu:-no of channels in the signal.

Function flowchart :-


idinput([n,nu,m],'rbs', band,levels)
Error Checking
yes
Throw Error msg
no
generated n length nu channel signal rand(n,'normal')
8th order butterworth filter with band
assign +1 or -1 to each signal depending on their sign
adjust binary values according to levels
repeating generated signal m times
n x m length,nu channel rbs signal

1.5.2 Butterworth filter :-

Syntax :

u=filtbutter(data,n,band)

Arguments :

data : input signal

n : order of the butterworth filter

band : pass band of the filter

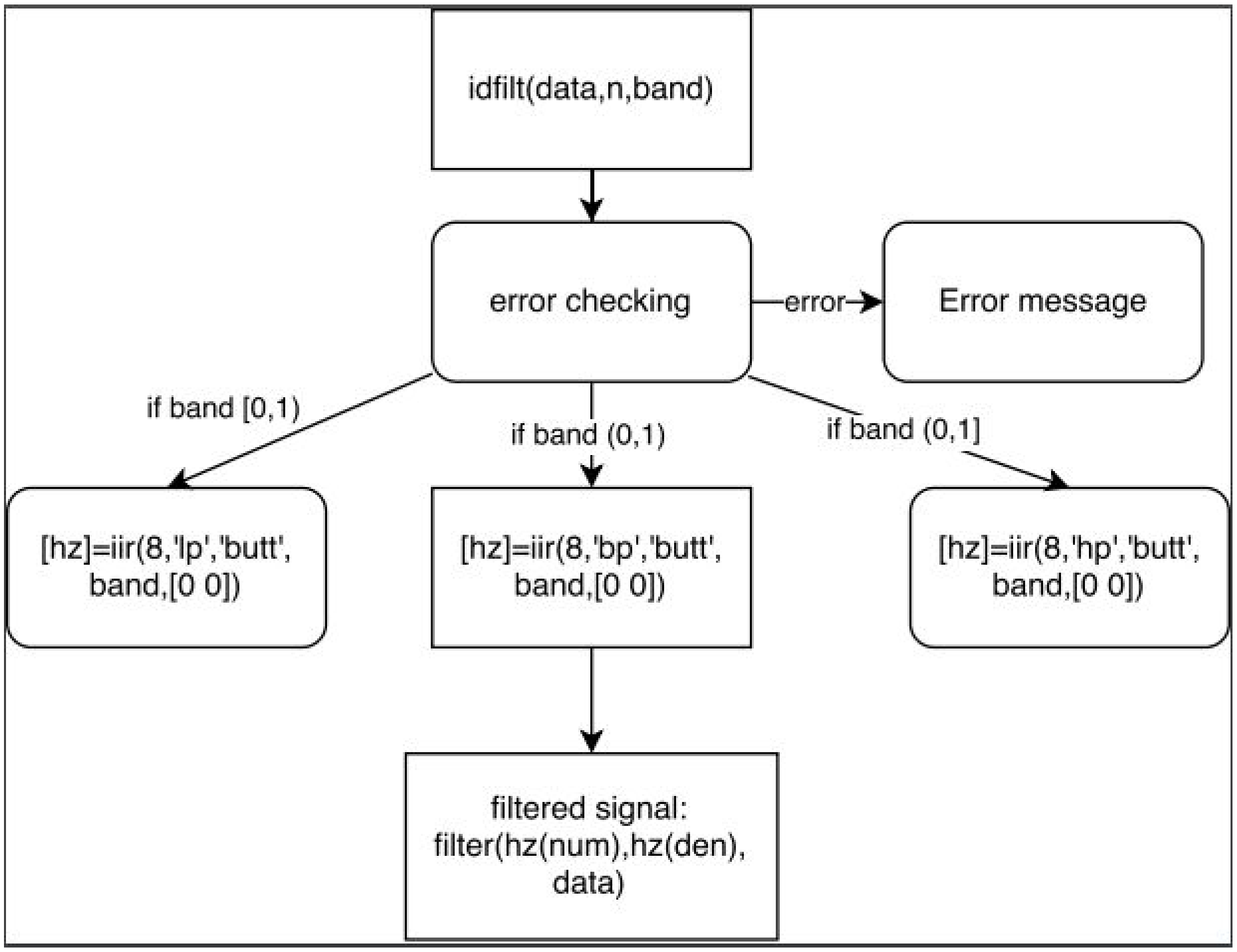
Description :-

Butterworth filter[1.5.3] is a type of signal processing filter designed to have as flat a frequency response as possible in the passband. It is also referred to as a **maximally flat magnitude filter**.

u=filtbutter(data,n,band) passes the input signal data through a nth order butterworth filter of passband band.

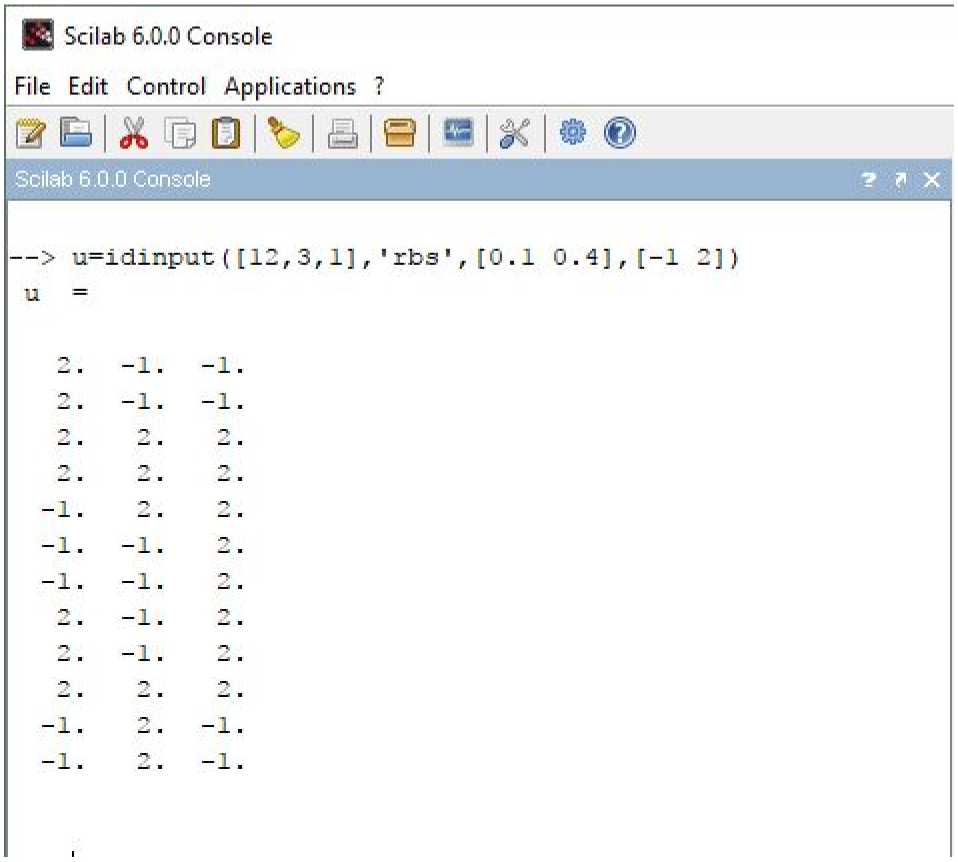
Note : in idinput(...) we are passing the randomly generated signal with a 8th order butterworth filter to get the final random binary signal.

Butterworth filter flowChart :-


idfilt(data,n,band)
error checking
error
Error message
if band [0,1)
if band (0,1)
if band (0,1]
[hz]=iir(8,'lp','butt', band,[0 0])
[hz]=iir(8,'bp','butt', band,[0 0])
[hz]=iir(8,'hp','butt', band,[0 0])
filtered signal: filter(hz(num),hz(den), data)

Example :-

Binary 3 channel signal of length 12.



```
--> u=idinput([12,3,1],'rbs',[0.1 0.4],[-1 2])
 u  =


   2.  -1.  -1.
   2.  -1.  -1.
   2.   2.   2.
   2.   2.   2.
  -1.   2.   2.
  -1.  -1.   2.
  -1.  -1.   2.
   2.  -1.   2.
   2.  -1.   2.
   2.   2.   2.
  -1.   2.  -1.
  -1.   2.  -1.
```

1.6 Lqry :-

(linear-quadratic (LQ) state-feedback regulator with output weighting[1.6.1])

Syntax :

[K,S,e] = lqry(sys,Q,R,N)

Input Arguments :-

sys :- A state space representation of a linear dynamical system.

Q :- Real symmetric matrix, with same dimensions as sys.A

R:- full rank real symmetric matrix (size(sys.B(2)))

N :- real matrix, the default value is zeros(size(R,1),size(Q,2))

K :- a real matrix,the optimal gain

Output Arguments :-

X :- stabilizing solution of the riccati equation

e :- closed loop eigenvalues

Description :- Given the plant

sys $\begin{cases} \dot{x} = Ax + Bu \\ z = Cx + Du \end{cases}$ in continuous time or $\begin{cases} x^{+} = Ax + Bu \\ z = Cx + Du \end{cases}$ in discrete time.

It computes a state feedback control u= -Kx ,that minimizes the quadratic cost function[1.6.1]

$$\int_0^\infty y^T Q y + u^T R u + 2 y^T N u dt$$

with output weighting.

Implementation methord :-

1.[Q N^T;N R]=[C^T 0;D^T I] x [Q N^T;N R]x[C D;0 I]

2. For a continuous plant, X is calculated by the stabilizing solution of the Riccati equation:

$(A-BR^{-1}N)^TX+X(A-BR^{-1}N)-XBR^{-1}B^TX+Q-N^TR^{-1}N=0$

$K=R^{-1}(B^TX+N^T)$

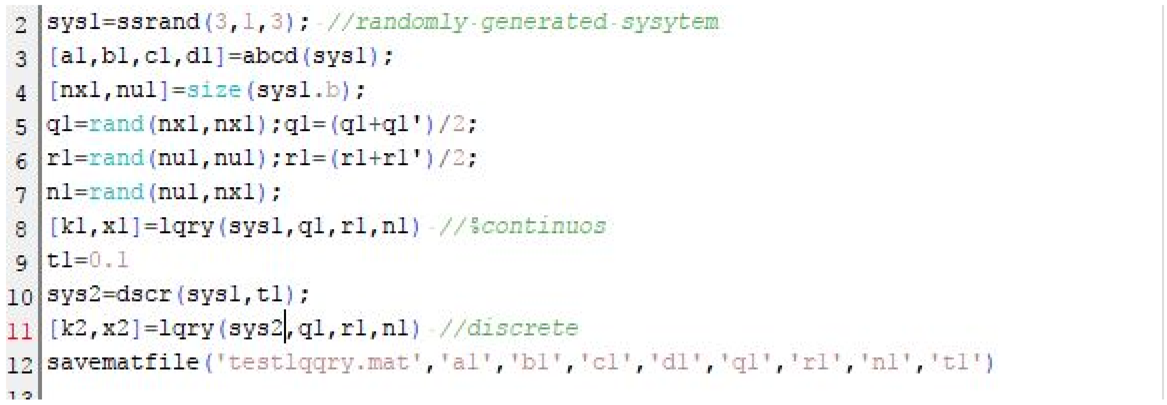
For a discrete time plant X is calculated by stabilizing solution of the riccati eqn:

$A^TXA -X -(A^TXB+N^T)(B^TXB+R)^{-1}(B^TXA+N)+Q=0$
$K=(B^TXB+R^{-1})(B^TXA+N)$

e=eigenvalues(A-BK)

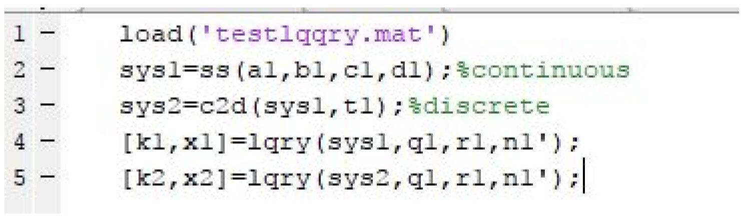
Example :-

Generating random sys1 and discrete time sys2 and q1, r1, n1 matrices.



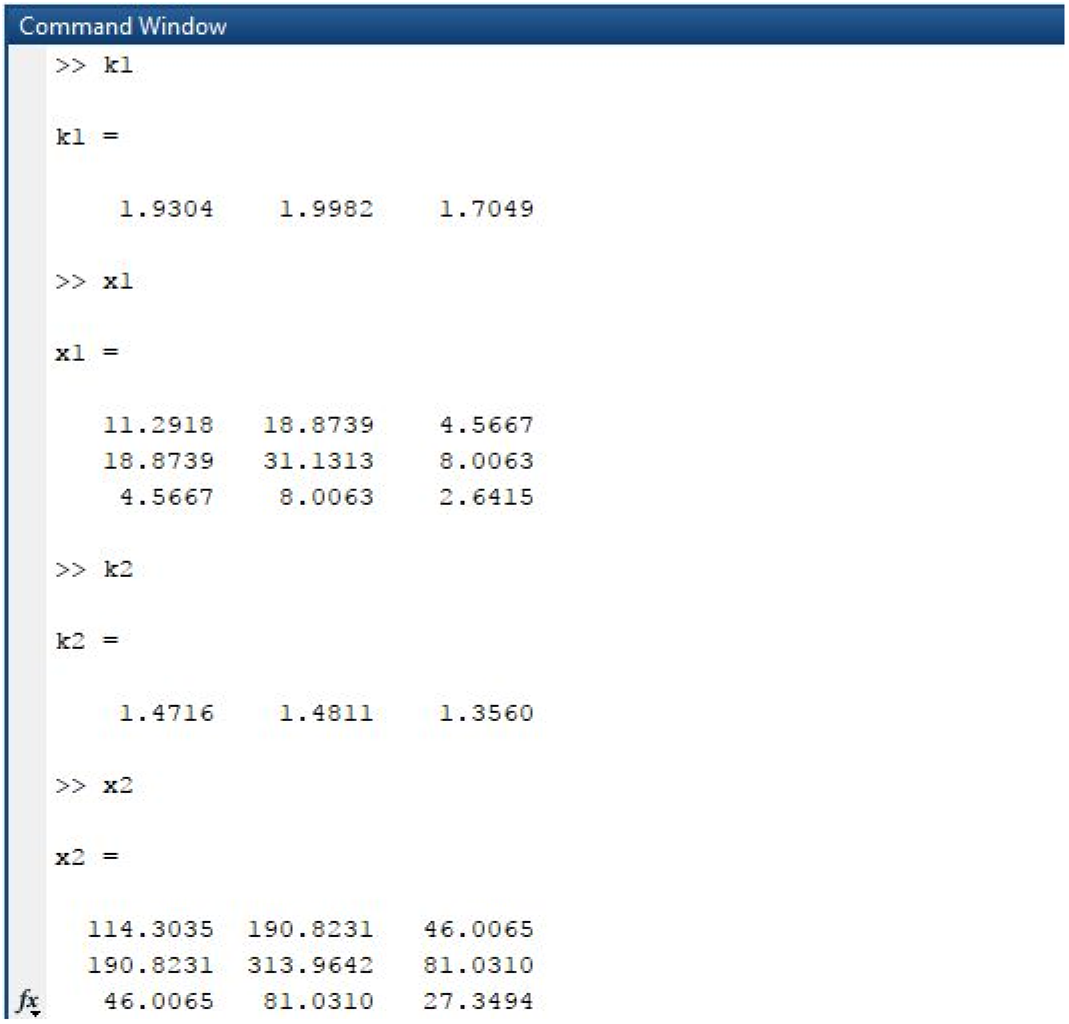
```
sys1=ssrand(3,1,3); //randomly generated sysytem
[a1,b1,c1,d1]=abcd(sys1);
[nx1,nu1]=size(sys1.b);
q1=rand(nx1,nx1);q1=(q1+q1')/2;
r1=rand(nu1,nu1);r1=(r1+r1')/2;
n1=rand(nu1,nx1);
[k1,x1]=lqry(sys1,q1,r1,n1) //%continuos
t1=0.1
sys2=dscr(sys1,t1);
[k2,x2]=lqry(sys2,q1,r1,n1) //discrete
savematfile('testlqqry.mat','a1','b1','c1','d1','q1','r1','n1','t1')
```

Loading scilab data in matlab :



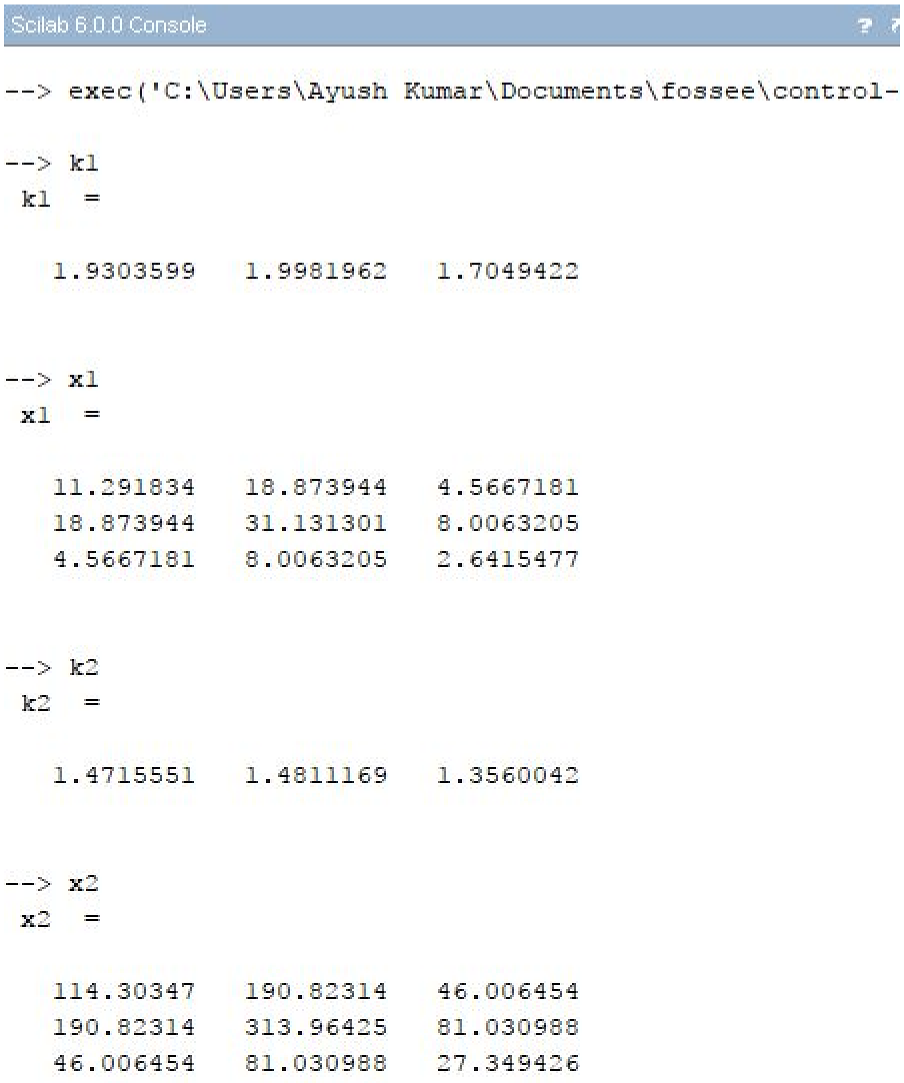
```
load('testlqry.mat')
sys1=ss(a1,b1,c1,d1);%continuous
sys2=c2d(sys1,t1);%discrete
[k1,x1]=lqry(sys1,q1,r1,n1');
[k2,x2]=lqry(sys2,q1,r1,n1');
```

matlab result :



```
>> k1

k1 =

    1.9304    1.9982    1.7049

>> x1

x1 =

   11.2918   18.8739    4.5667
   18.8739   31.1313    8.0063
    4.5667    8.0063    2.6415

>> k2

k2 =

    1.4716    1.4811    1.3560

>> x2

x2 =

  114.3035  190.8231   46.0065
  190.8231  313.9642   81.0310
   46.0065   81.0310   27.3494
```

Scilab result :



```
--> exec('C:\Users\Ayush Kumar\Documents\fossee\control-

--> k1
 k1  =

   1.9303599   1.9981962   1.7049422


--> x1
 x1  =

   11.291834   18.873944   4.5667181
   18.873944   31.131301   8.0063205
   4.5667181   8.0063205   2.6415477


--> k2
 k2  =

   1.4715551   1.4811169   1.3560042


--> x2
 x2  =

   114.30347   190.82314   46.006454
   190.82314   313.96425   81.030988
   46.006454   81.030988   27.349426
```

1.7 LQRD :-

(discrete linear-quadratic (LQ) regulator for continuous plant[1.7.1])

Syntax:-

[Kd,Xd]=lqrd(A,B,Q,R,N,ts)

Input arguments : A,B:- continuous plant state space

Q,r,n:- weighting matrices, ts:- sampling time of the discrete regulator

Description:- lqrd designs a discrete full-state-feedback regulator that has response characteristics similar to a continuous state-feedback regulator designed using lqr. This command is useful to design a gain matrix for digital implementation.

calculates the discrete state-feedback law

$u[n]=-K_d x[n]$

that minimizes a discrete cost function equivalent to the continuous cost function :-

$$\int_0^\infty x^T Q x + u^T R u + 2x^T N u dt$$

Implementation:- The equivalent discrete gain matrix Kd is determined by discretizing the continuous plant and weighting matrices using the sample time Ts and the zero-order hold approximation.

$$\Phi(\tau) = e^{A\tau},$$
$$\Gamma(\tau) = \int_0^\tau e^{A\eta} B d\eta,$$

The weighting matrix for the equivalent discrete cost function:-

$$\begin{bmatrix} Q_d & N_d \\ N_d^T & R_d \end{bmatrix} = \int_0^{T_s} \begin{bmatrix} \Phi^T(\tau) & 0 \\ \Gamma^T(\tau) & I \end{bmatrix} \begin{bmatrix} Q & N \\ N^T & R \end{bmatrix} \begin{bmatrix} \Phi(\tau) & \Gamma(\tau) \\ 0 & I \end{bmatrix} d\tau$$

Determine discrete equivalent of continuous cost function along with Ad,Bd matrices of discretized system,The discrete equivalent is calculated using matrix exponential formulas due to Van Loan [1.7.2].

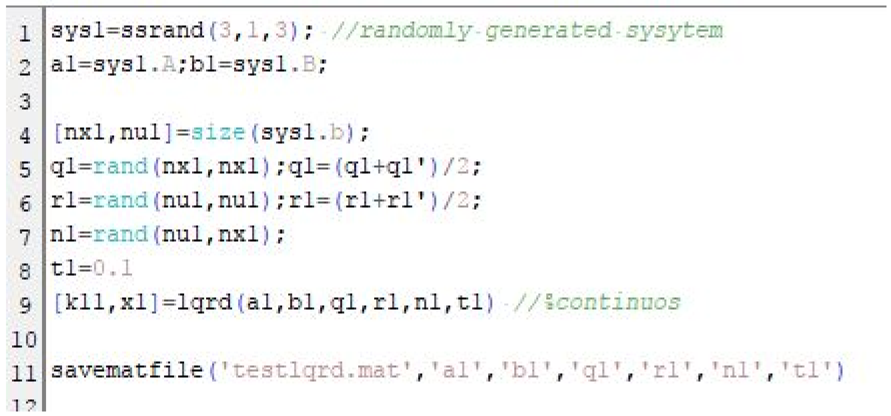
Xd is calculated by solving the General discrete time algebraic riccati equation.

$A^TXA-X-(A^TXB+N_d)(B^TXB+R_d)^{-1}(B^TXA+N_d^T)+Q_d=0$

$Kd=(B^TX_dB+R)^{-1}(B^TX_dA+N_d^T)$

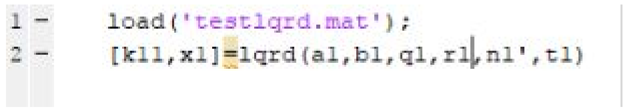
Example :-

scilab



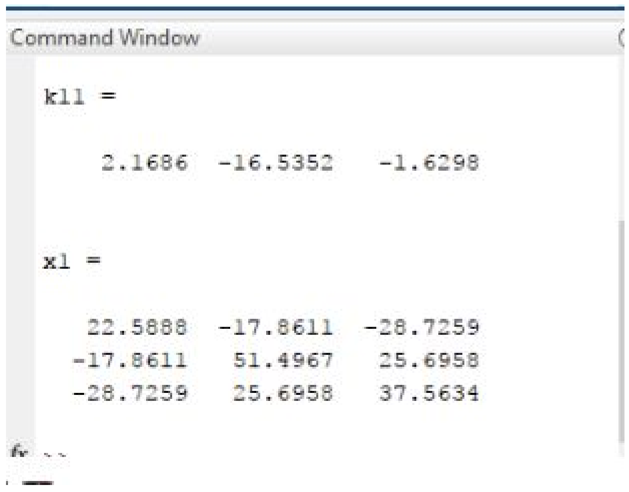
```
sysl=ssrand(3,1,3); //randomly generated sysytem
al=sysl.A;bl=sysl.B;

[nxl,nul]=size(sysl.b);
ql=rand(nxl,nxl);ql=(ql+ql')/2;
rl=rand(nul,nul);rl=(rl+rl')/2;
nl=rand(nul,nxl);
tl=0.1
[kll,xl]=lqrd(al,bl,ql,rl,nl,tl) //%continuos

savematfile('testlqrd.mat','al','bl','ql','rl','nl','tl')
```

matlab



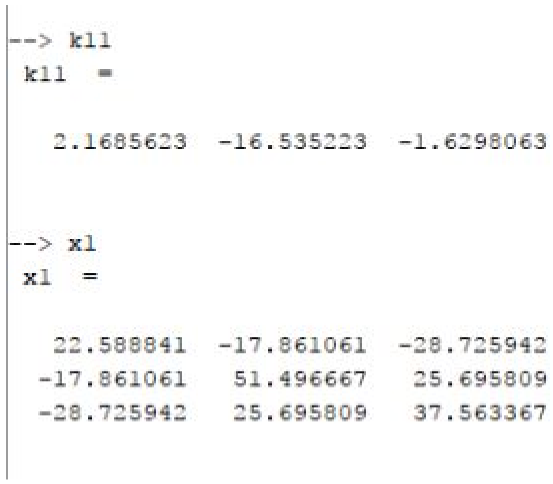
```
load('testlqrd.mat');
[kll,xl]=lqrd(al,bl,ql,rl,nl',tl)
```

Matlab result :



```
k11 =

    2.1686  -16.5352   -1.6298


x1 =

   22.5888  -17.8611  -28.7259
  -17.8611   51.4967   25.6958
  -28.7259   25.6958   37.5634

fx >>
```

Scilab result :



```
--> k11
 k11  =

   2.1685623  -16.535223  -1.6298063


--> x1
 x1  =

   22.588841  -17.861061  -28.725942
  -17.861061   51.496667   25.695809
  -28.725942   25.695809   37.563367
```

1.8 estim :-

(state spaces of a state estimator for a given estimator gain.[1.8.1])

Syntax :-

[ae,be,ce,de]=estim(sys,l)
[ae,be,ce,de]=estim(sys,l,sensors,known)

Input argument :

sys:lti model
l :state feedback matrix.
sensors : Indices of measured output signals y,if omitted all outputs are measured.
known : Indices of known input signals u (deterministic) to sys. All other inputs to sys are assumed stochastic.Default known=[ ]

Output arguments :

[ae,be,ce,de]:state spaces of the estimator

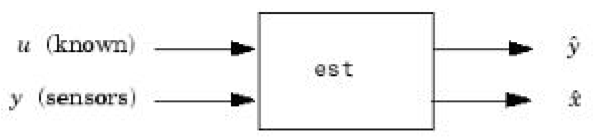
Description :

estim(sys,l)produces a state estimator est given the plant state-space model sys and the estimator gain L.All inputs of sys are assumed stochastic and all outputs are measured. The estimator state-spaces[1.8.2] are returned.
For a continuous system
dx/dt=Ax+Bw, y=Cx+Dw
estim uses the following equation to generate plant output estimate $\hat{y}$ and a state estimate $\hat{x}$:

$$d\hat{x}/dt=A\hat{x}+L(\hat{y}-C\hat{x})$$
$$[\,|\hat{y}|;|\hat{x}|\,] = [C\ ;\ I]\hat{x}$$

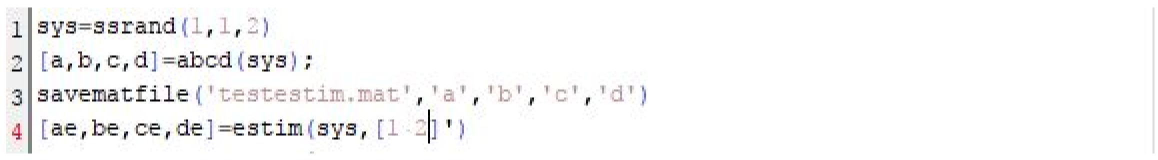
est = estim(sys,L,sensors,known) handles more general plants sys with both known (deterministic) inputs u and stochastic inputs w, and both measured outputs y and non measured outputs z,in this case the estimates are given by:

$$d\hat{x}/dt=A\hat{x}+B_2u+L(\hat{y}-C2\hat{x}-D_{22}u)$$
$$[\,|\hat{y}|;|\hat{x}|\,] = [C_2\ ;\ I]\hat{x} + [D_{22}\ ;\ 0]u$$



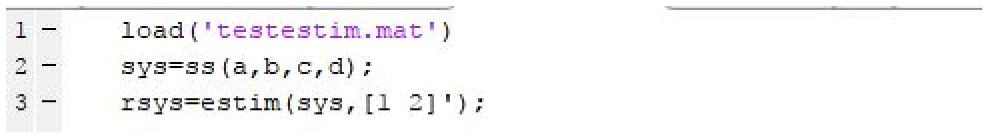
Example :

sys a random 2nd order system ,gain matrix l=[1 2]’



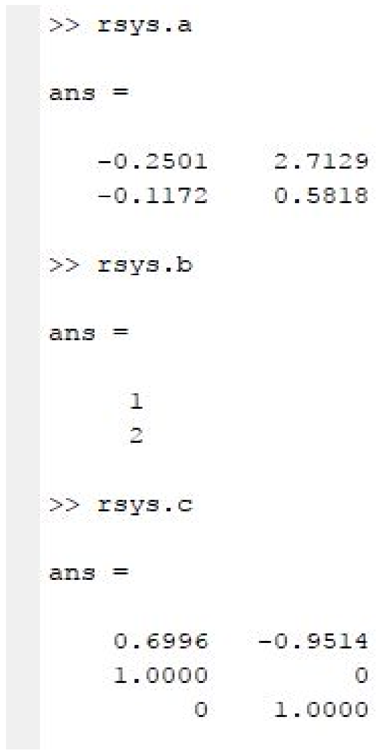
```
sys=ssrand(1,1,2)
[a,b,c,d]=abcd(sys);
savematfile('testestim.mat','a','b','c','d')
[ae,be,ce,de]=estim(sys,[1 2]')
```

Loading data in matlab:



```
load('testestim.mat')
sys=ss(a,b,c,d);
rsys=estim(sys,[1 2]');
```

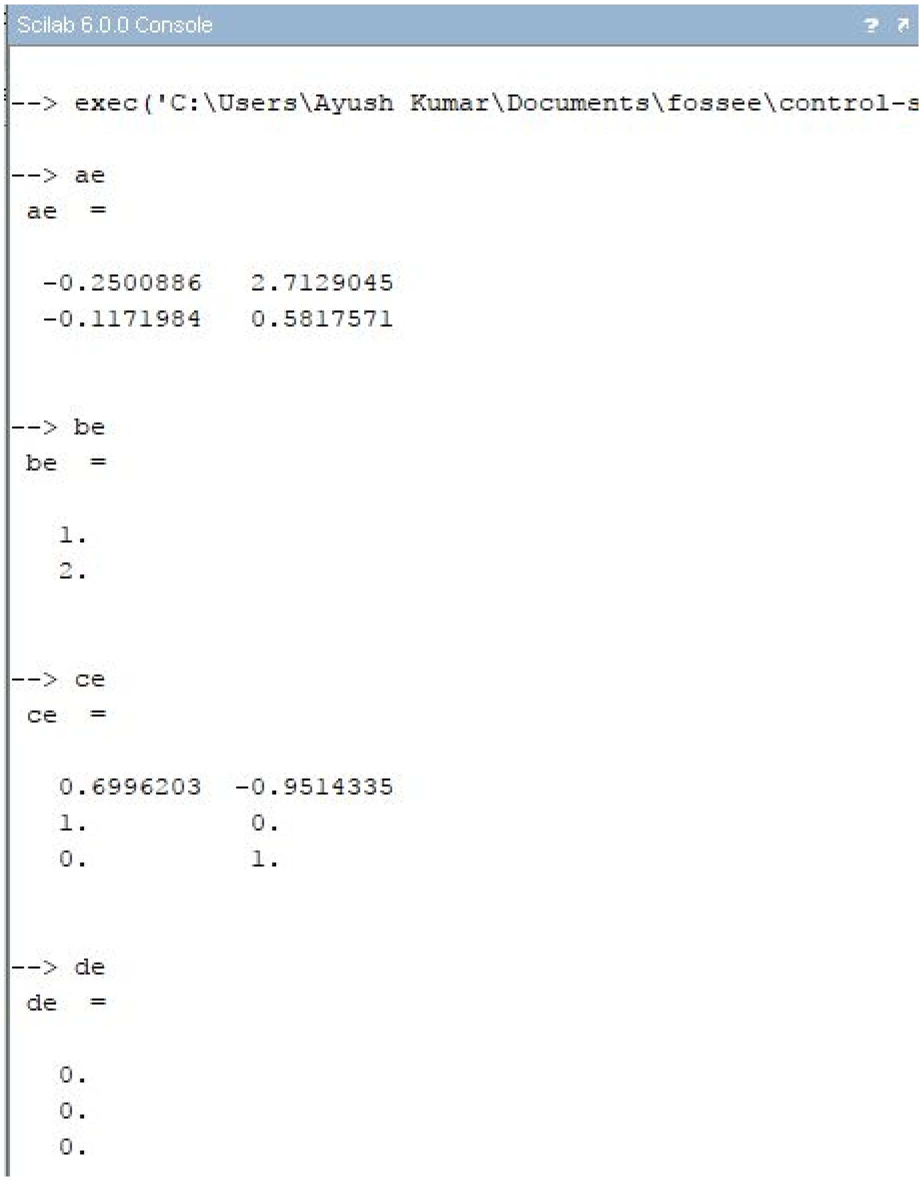
Matlab result:



```
>> rsys.a

ans =

   -0.2501    2.7129
   -0.1172    0.5818

>> rsys.b

ans =

     1
     2

>> rsys.c

ans =

    0.6996   -0.9514
    1.0000         0
         0    1.0000
```

```
>> rsys.d

ans =

     0
     0
     0
```

Scilab result:



Scilab 6.0.0 Console

```
--> exec('C:\Users\Ayush Kumar\Documents\fossee\control-s

--> ae
 ae  =

  -0.2500886   2.7129045
  -0.1171984   0.5817571


--> be
 be  =

   1.
   2.


--> ce
 ce  =

   0.6996203  -0.9514335
   1.          0.
   0.          1.


--> de
 de  =

   0.
   0.
   0.
```

1.9 Reg:

(returns state spaces of a regulator given state-feedback and estimator gain)

Syntax:

[ac,bc,cc,dc]=reg(sys,k,l)
[ac,bc,cc,dc]=reg(sys,k,l,sensors,controls)

Description:

[ac,bc,cc,dc] =reg(sys,K,L) returns the state spaces of a dynamic regulator[1.9.1] rsys given a state-space model sys of the plant, a state-feedback gain matrix K, and an estimator gain matrix L. The gains K and L are typically designed using pole placement or LQG techniques[1.9.2]. The function reg can be used for both continuous and discrete time plants.This syntax assumes that all inputs of sys are controls, and all outputs are measured.

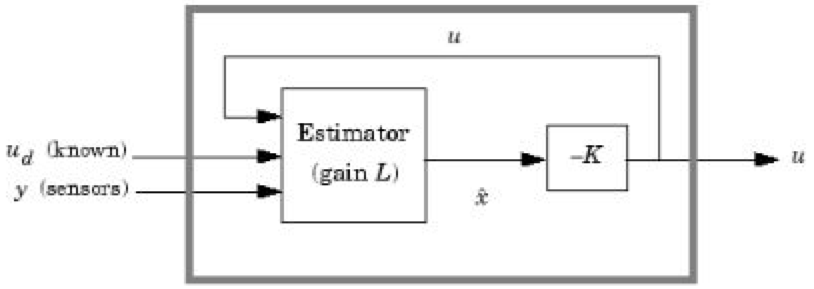
The regulator rsys is obtained by connecting the state-feedback law u = –Kx and the state estimator with gain matrix L which can be created by the state spaces ac,bc,cc,dc.This regulator should be connected to the plant using positive feedback.

For a plant:
dx/dt=Ax+Bu,
y=Cx+Du

The regulator is given by:
$d\hat{x}/dt = (A-LC-(B-LD)K)\hat{x} + Ly$
$u = -K\hat{x}$

[ac,bc,cc,dc] = reg(sys,K,L,sensors,known,controls) handles more general regulation problems where: The plant inputs consist of controls u, known inputs u_d, and stochastic inputs w.Only a subset y of the plant outputs is measured.

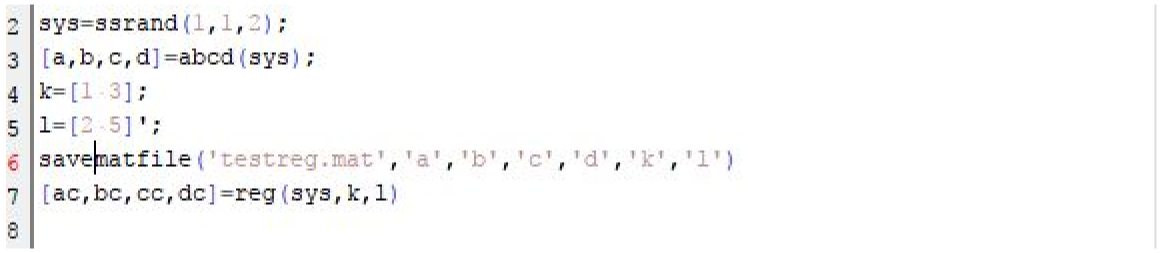
The index vectors sensors, known, and controls specify y, u_d, and u as subsets of the outputs and inputs of sys.



Regulator

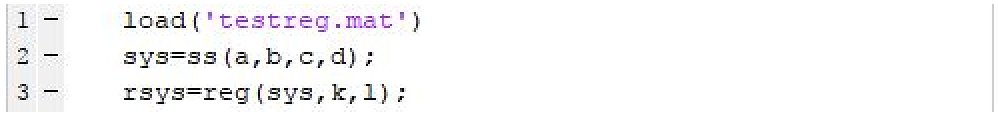
Example:

generating a random 2nd order system and k and l matrices.



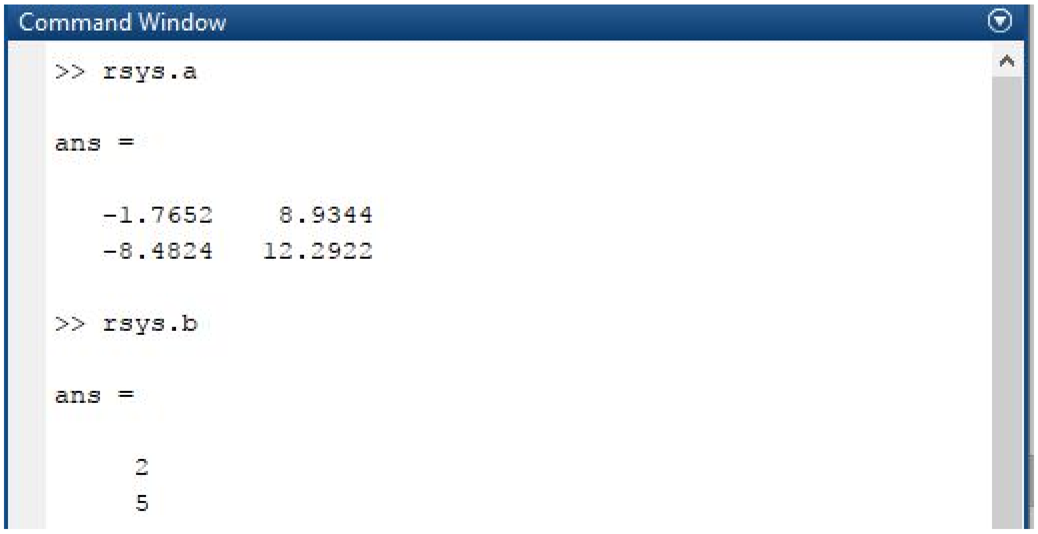
```
sys=ssrand(1,1,2);
[a,b,c,d]=abcd(sys);
k=[1 3];
l=[2 5]';
savematfile('testreg.mat','a','b','c','d','k','l')
[ac,bc,cc,dc]=reg(sys,k,l)

```

Loading data in matlab:



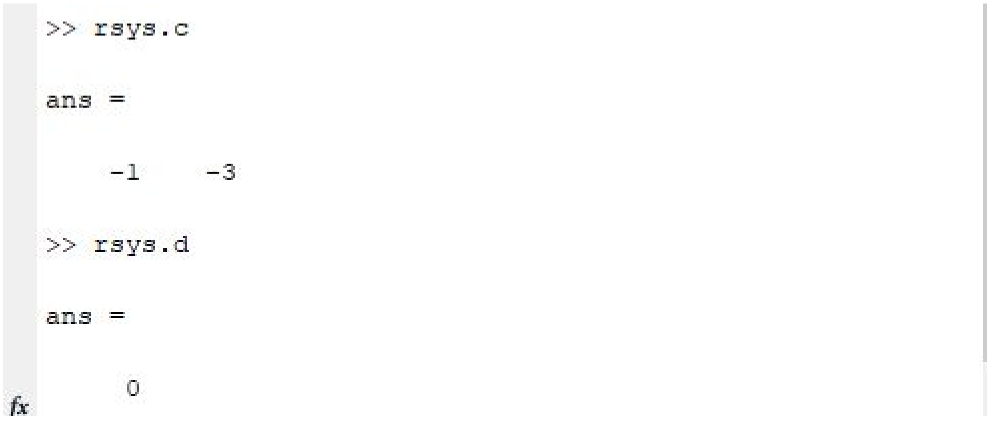
```
load('testreg.mat')
sys=ss(a,b,c,d);
rsys=reg(sys,k,l);
```

Matlab result:



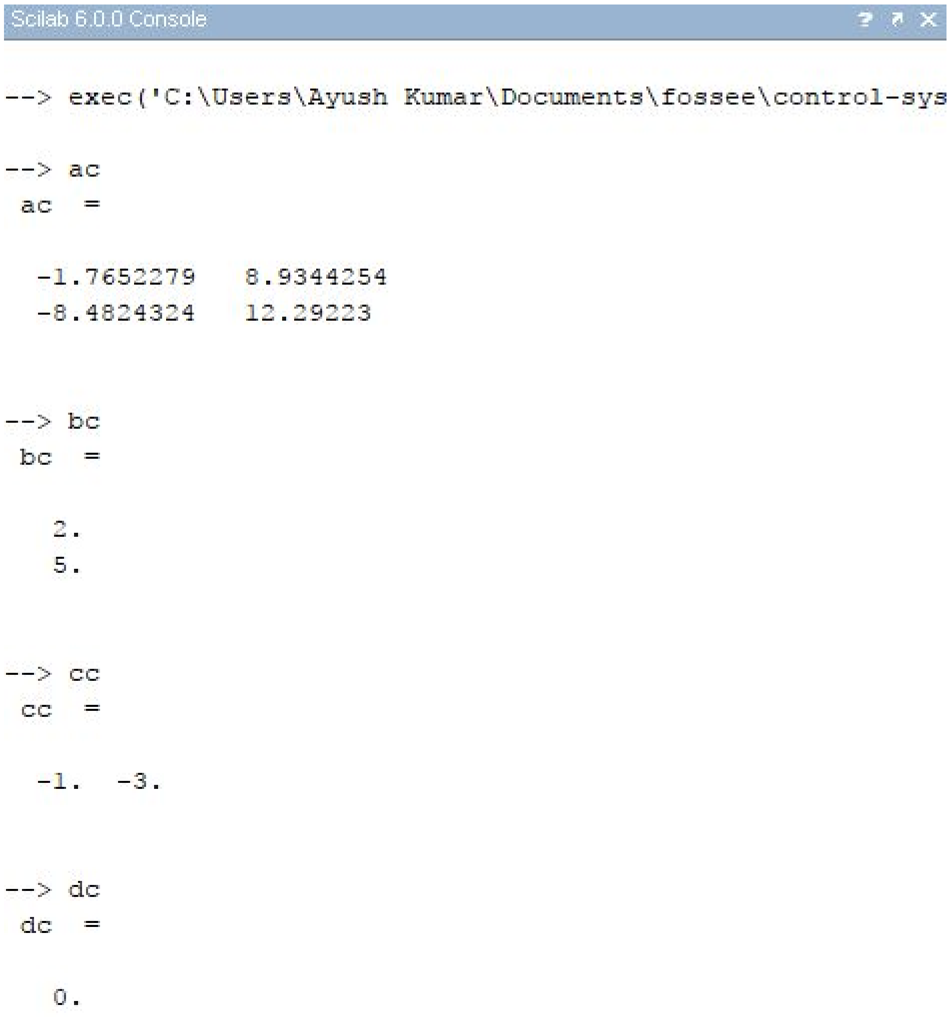
Command Window

```
>> rsys.a

ans =

   -1.7652    8.9344
   -8.4824   12.2922

>> rsys.b

ans =

     2
     5
```



```
>> rsys.c

ans =

    -1    -3

>> rsys.d

ans =

     0
```

Scilab result



```
Scilab 6.0.0 Console

--> exec('C:\Users\Ayush Kumar\Documents\fossee\control-sys

--> ac
 ac  =

  -1.7652279   8.9344254
  -8.4824324   12.29223


--> bc
 bc  =

   2.
   5.


--> cc
 cc  =

  -1.  -3.


--> dc
 dc  =

   0.
```

2.0 nmodels :-

(Number of models in model array[2.0.1])

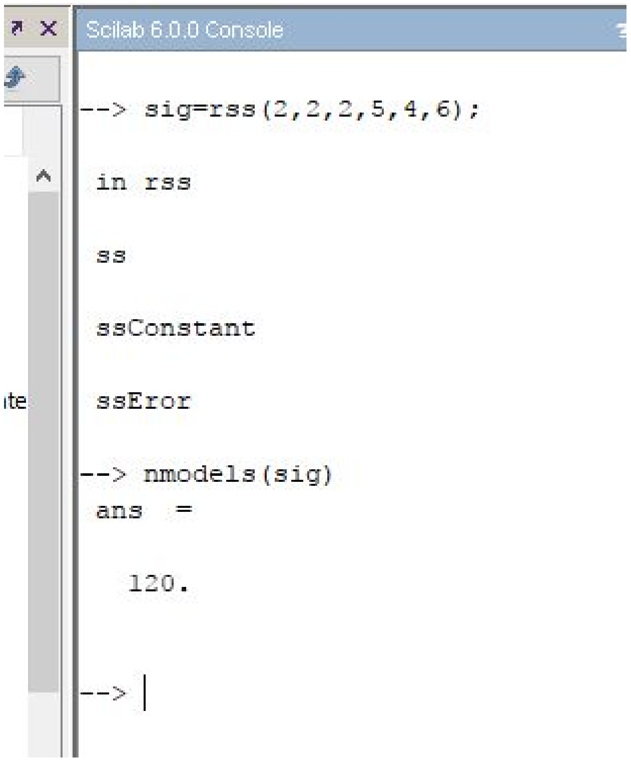
Syntax :- n = nmodels(sys)

Input argument : sys :- Input model array, specified as an array of input-output models such as numeric LTI models, generalized models..

Output argument :- N :- no of models in the given model array.

Description :- n = nmodels(sys)[2.0]returns the number of models in an array of dynamic system models or static models.

Example :



```
--> sig=rss(2,2,2,5,4,6);

 in rss

 ss

 ssConstant

 ssEror

--> nmodels(sig)
 ans  =

   120.


-->
```

2.1 Rlocusplot :-

(plots root locus and returns plot handle)

Syntax:-

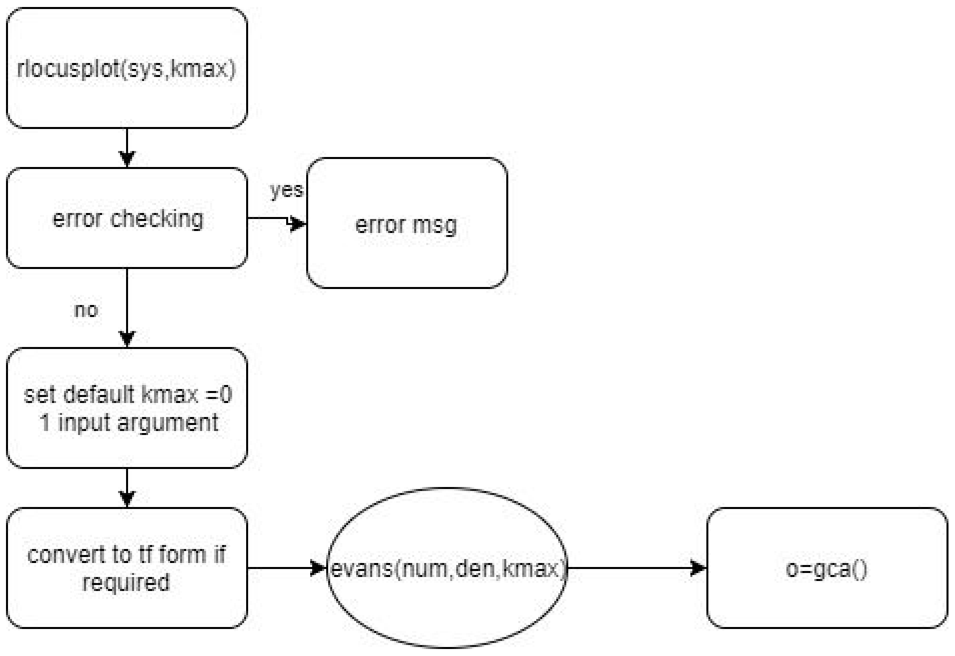
h=rlocusplot(H,Kmax)

Input argument :-
H :- Siso linear system given by a transfer function or a state space representation.
Kmax real(maximum gain desired for the plot)

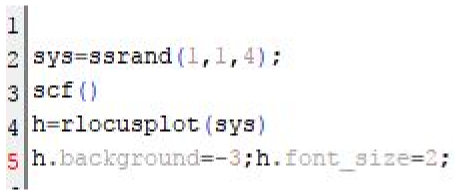
Output :
h :- plot handle,which can be used to further customize the plot.

Description :- Plots the root locus[2.1.1] for a SISO linear system in state-space or transfer form H(s) . This is the locus of the roots of 1+k*H(s)=1+k*N(s)/D(s), in the complex plane. For a selected sample of gains k <= kmax ,the imaginary part of the roots of D(s)+k*N(s) is plotted vs the real part, and returns the plot handles[2.1.2] which can be used to customize the plot.

Program flow :-

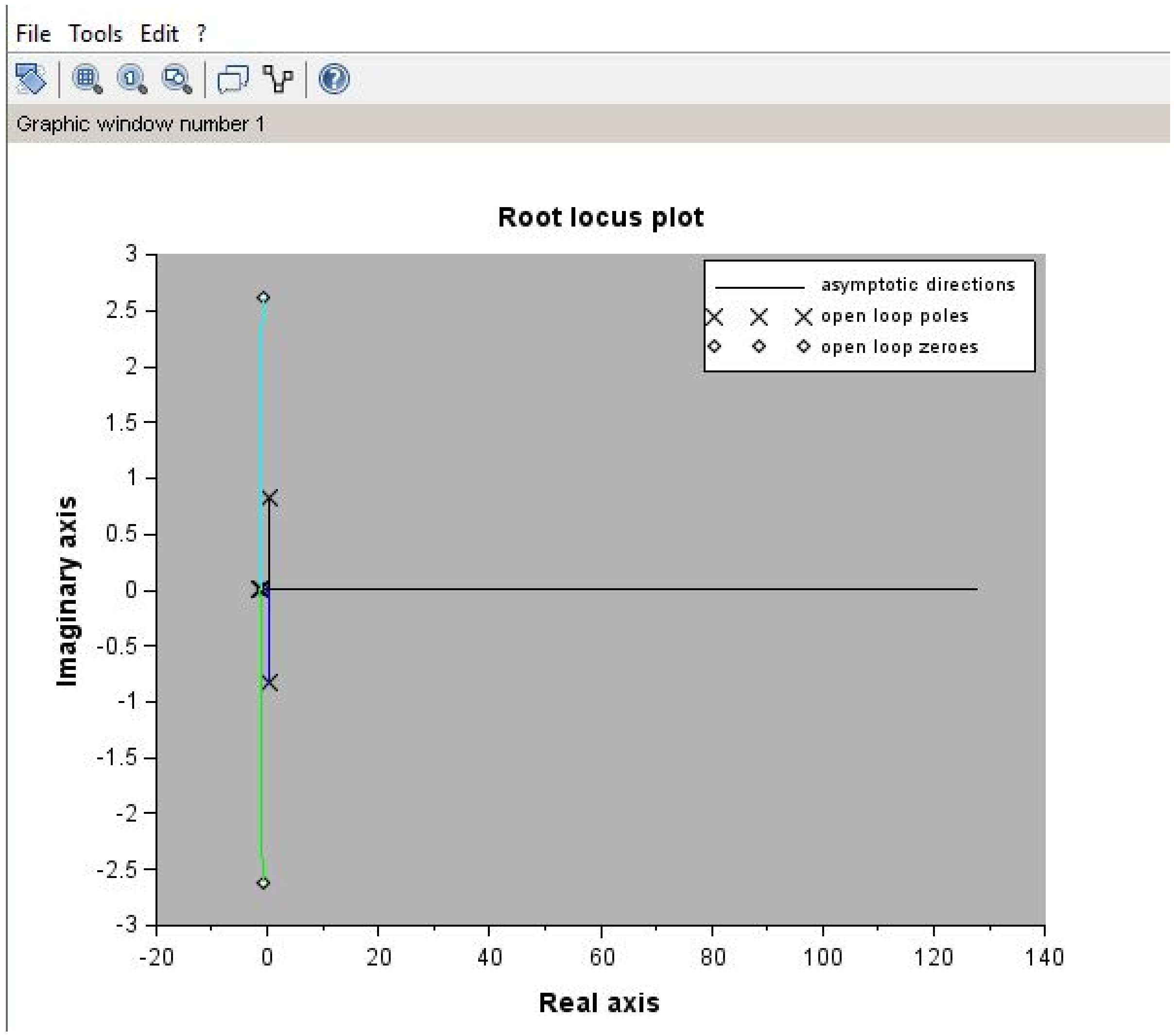


Example :-



```
sys=ssrand(1,1,4);
scf()
h=rlocusplot(sys)
h.background=-3;h.font_size=2;
```

rlocusplot:



2.2 Dare :-

(solve general discrete algebraic riccati equation)

Syntax:- [X]=dare(A,B,Q)

[X]=dare(A,B,Q,R)

[X]=dare(A,B,Q,R,S)

Arguments :-

A,B,Q,R,S :- A real square matrix,B real matrices,Q real symmetric matrix of size(A) and R symmetric matrix of size(b,2) default identity matrix,S real matrix of size(b)

[X]:-solution of the general discrete riccati equation

Description :-

[X]=dare(A,B,Q,R) computes the unique stabilizing solution X of the discrete-time algebraic Riccati equation

$$A^TXA-X-A^TXB(B^TXB+R)^{-1}(B^TXA)+Q=0$$

[X] =dare(A,B,Q,R,S) solves the more general discrete-time algebraic Riccati equation[2.2.1]

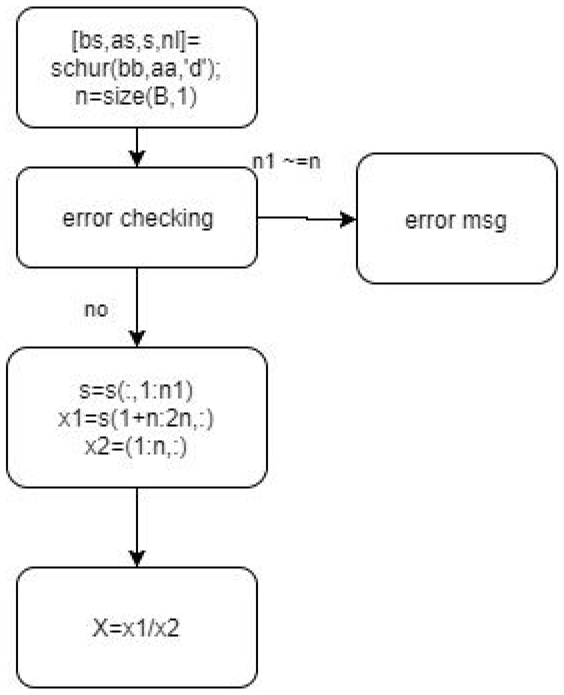
$$A^TXA-X-(A^TXB+S)(B^TXB+R)^{-1}(B^TXA+S^T)+Q=0$$

Methord :-

The general solution of the riccati equation is obtained by schur factorisation of the matrix pencils associated with these Riccati equations[2.2.2] :

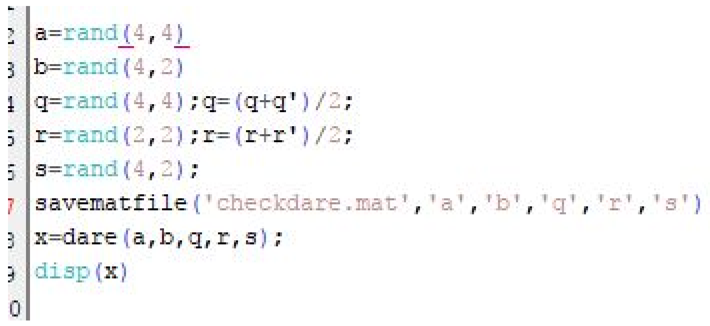
$$z\underbrace{[I\ BR^{-1}B^T\ ;\ 0\ \ (A-BR^{-1}S^T)^T]}_{aa}\ -\ \underbrace{[A-BR^{-1}S^T\ 0\ ;\ SR^{-1}S^T-Q]}_{bb}$$

Function flowChart:



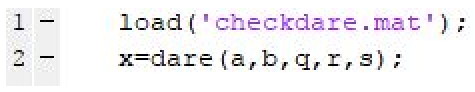
Example (Result comparison) :-

Generating random a,b,q,r and s matrices (scilab)



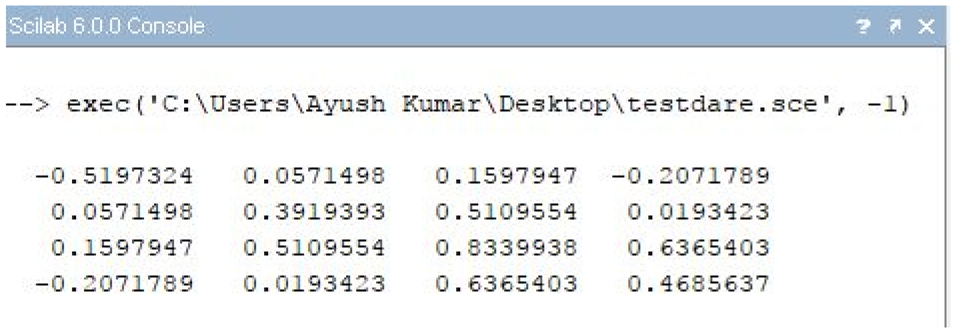
```
a=rand(4,4)
b=rand(4,2)
q=rand(4,4);q=(q+q')/2;
r=rand(2,2);r=(r+r')/2;
s=rand(4,2);
savematfile('checkdare.mat','a','b','q','r','s')
x=dare(a,b,q,r,s);
disp(x)
```

Loading data in matlab:



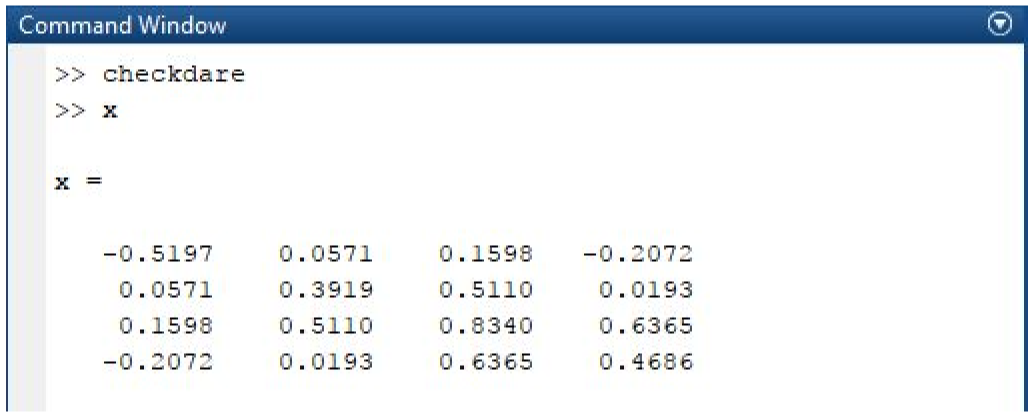
```
load('checkdare.mat');
x=dare(a,b,q,r,s);
```

Scilab result:


Scilab 6.0.0 Console

```
--> exec('C:\Users\Ayush Kumar\Desktop\testdare.sce', -1)

  -0.5197324   0.0571498   0.1597947  -0.2071789
   0.0571498   0.3919393   0.5109554   0.0193423
   0.1597947   0.5109554   0.8339938   0.6365403
  -0.2071789   0.0193423   0.6365403   0.4685637
```

Matlab result :


Command Window

```
>> checkdare
>> x

x =

   -0.5197    0.0571    0.1598   -0.2072
    0.0571    0.3919    0.5110    0.0193
    0.1598    0.5110    0.8340    0.6365
   -0.2072    0.0193    0.6365    0.4686
```

2.3 Stack :-

(build model array by stacking models along array dimensions)

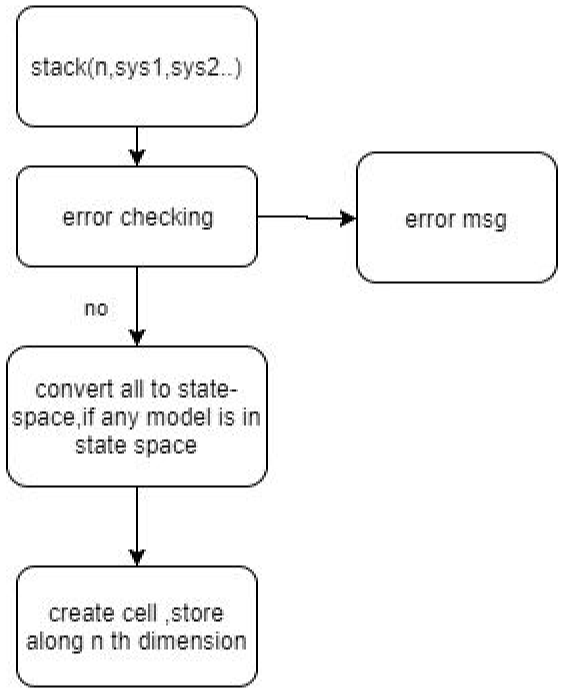
Syntax:-

sys=stack(n,sys1,sys2,.....)

Description :-

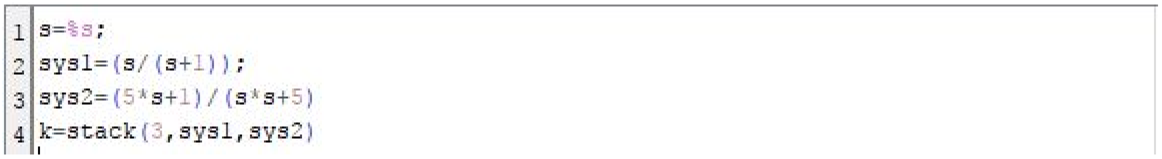
sys = stack(n,sys1,sys2,...)[2.3.1] produces an array of dynamic system models sys by stacking (concatenating) the models (or arrays) sys1,sys2,... along the array dimension. All models must have the same number of inputs and outputs (the same I/O dimensions), but the number of states can vary.The resulting array dimensions are stored in a cell which can further be accessed.

Program flow :-

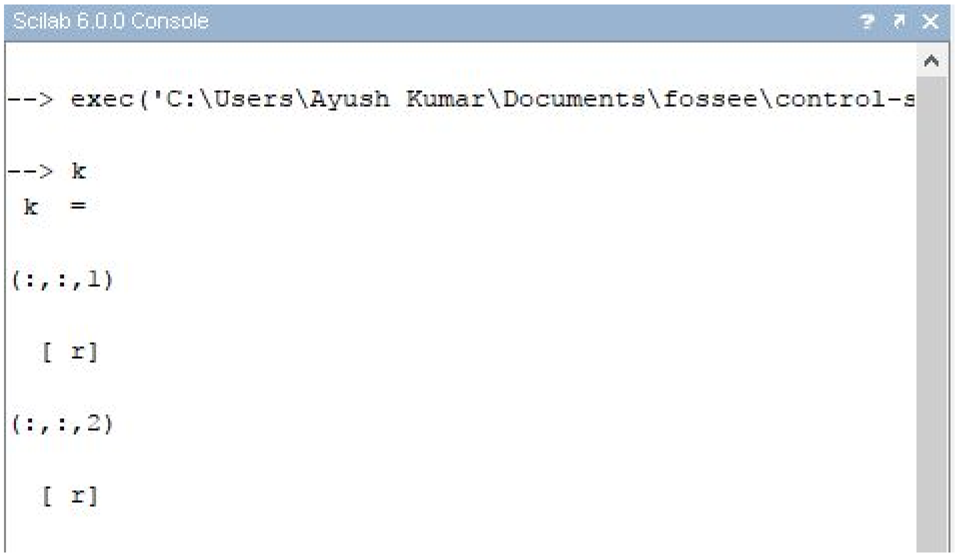

stack(n,sys1,sys2..)
error checking
error msg
no
convert all to state-space,if any model is in state space
create cell ,store along n th dimension

Example:-

Stacking 2 systems sys1 and sys2 in k along 3rd dimension



```
s=%s;
sys1=(s/(s+1));
sys2=(5*s+1)/(s*s+5)
k=stack(3,sys1,sys2)
```



```
--> exec('C:\Users\Ayush Kumar\Documents\fossee\control-s

--> k
 k  =

(:,:,1)

  [ r]

(:,:,2)

  [ r]
```

```
--> k{:,:}
 ans  =


       ans(1)


    s
  ------

  1 + s


       ans(2)


  1 + 5s
  -------
       2
  5 + s
```

Tunable models :-

2.4 Realp :-

(real tunable parameter)

Syntax :-

P=realp(name,initialvalue)

Description :-

It creates a tunable real-valued parameter[2.4.1] with name specified by name and initial value initvalue. it can be scalar- or matrix- valued

Input argument :-

name :- name of realp parameter p,specified as a character or string.it sets the value of the name property of p.

initvalue :- initial numeric value of the parameter p,can be a real scalar vector or a marix

Output argument :

p : real parameter object.

Properties of created realp object ;

name :name of realp object

value : Value of the realp object ,it can be a real scalar value or a 2-dimensional matrix. The initial value is set by the initvalue input argument. The dimensions of Value are fixed on creation of the realp object.

Max: Upper bound for the parameter value. The dimension of the max property matches the dimension of the Value property.(default + inf)

Min : Lower bound for the parameter value. The dimension of the min property matches the dimension of the Value property.(default - inf)

Free :- Boolean value specifying whether the parameter is free to be tuned. Set the Free property to 1 (true) for tunable parameters, and 0 (false) or ~=1 for fixed parameters.The dimension of the Free property matches the dimension of the Value property.Default: 1 (true) for all entries

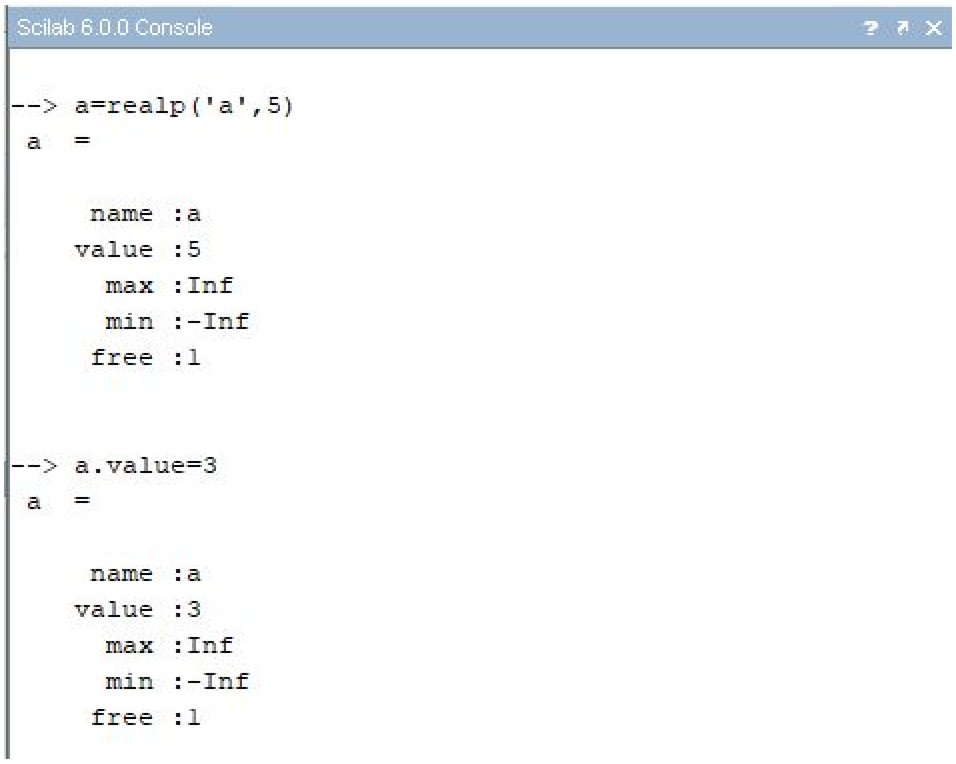
Implementation :

a tlist object [2.4.2]is created of type “realp” to implement the realp functionality (the created parameter will be an object having the properties name,value,max,min,free).

Overloading function[2.4.3] %realp_p(**var**) is defined so as to display the result in a presentable way.

Example:

creating a realp named a and initial value 5 and further changing it to 3.



```
--> a=realp('a',5)
 a   =

     name :a
    value :5
      max :Inf
      min :-Inf
     free :1


--> a.value=3
 a   =

     name :a
    value :3
      max :Inf
      min :-Inf
     free :1
```

2.5 Genmat :

Generalized matrices (genmat) are matrices[2.5.1] that depend on tunable parameters.we can also use generalized matrices for building generalized LTI models that represent control systems having a mixture of fixed and tunable components.It can be used for parameter studies,how the system behaviour changes with change in parameter etc.

Syntax :

M =genmat(A) converts the numeric array or tunable parameter A into a object.

A :- a realp object.

M: created generalized matrix.

It can also be created by combining numeric values with realp object,their combinations using any of the arithmetic operators +, -, *, /, \, and ^.

For example, if a and b are tunable parameters, the expression a + b is represented as a generalized matrix.

Description :- a generalized matrix [2.5.1] is a structure having attributes blocks,SamplingGrid and doublem.

where

Blocks : a structure which keep tracks of the realp objects which are used in creating the generalized matrix.

SamplingGrid : Sampling grid for model arrays

doublem : stores the current value of created generalized matrix .

Implementation :

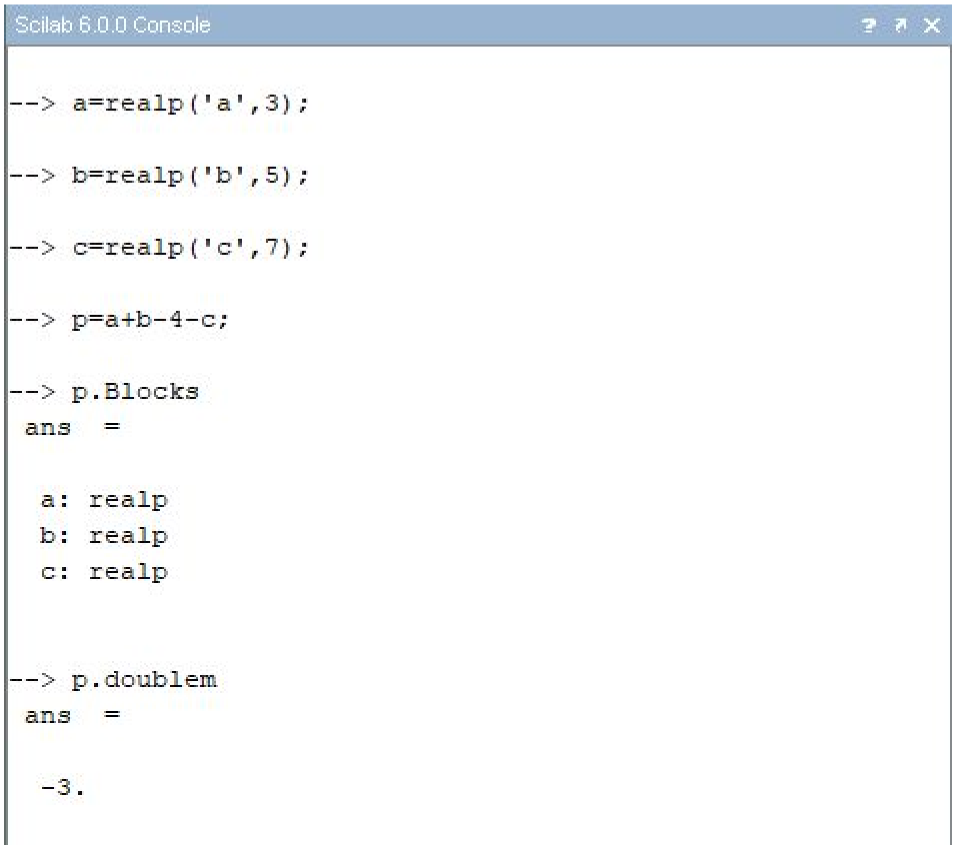
1.Creating a structure[2.5.2] with attributes blocks,SamplingGrid and doublem such that whenever genmat function is called with a realp object it creates a generalized matrix.

2.Defining overloading functions[2.5.3] like %realp_a_s(a,b), %realp_a_realp(a,b) etc so that a genmat is returned whenever an arithmetic object of a realp object with a numeral or another realp object is called (example if we try to calculate a+b+1 where both a and b are realp object it will return a generalized matrix structure, whose blocks will contain a and b objects and doublem will have the value a.value+b.value+1).

Note:i implemented about 35 overloading functions but this function still needs some more overloading functions like %s_c_st,%s_f_st to define so as to make it work for the case of matrices like [1 a+b;c 0],which i was not able to finish as nothing much information is given about them and methord of their implementation.

Example:

Creating 3 realp object a,b,c.p is a genmat object in which p.Blocks tracks the tunable parameters and p.doublem stores p current value(i.e a.value+b.value-4-c.value)



```
--> a=realp('a',3);

--> b=realp('b',5);

--> c=realp('c',7);

--> p=a+b-4-c;

--> p.Blocks
 ans  =

  a: realp
  b: realp
  c: realp


--> p.doublem
 ans  =

  -3.
```

2.6 Function checking :-

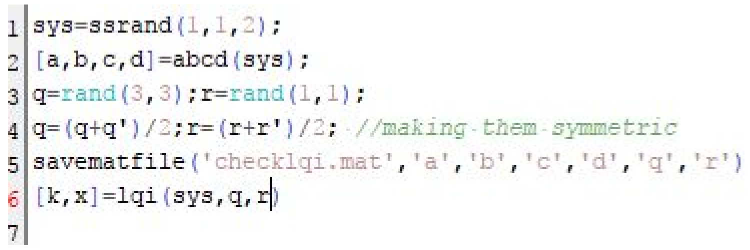
Lqi(matlab) : equivalent function lqi[2.6.1] is present in scilab

Syntax: [k,x]=lqi(sys,q,r)

Sys: lti model system with nx states,nu inputs and ny outputs.
q:Real nx+ny by nx+ny symmetric matrix,
r:Real nu by nu symmetric matrix,

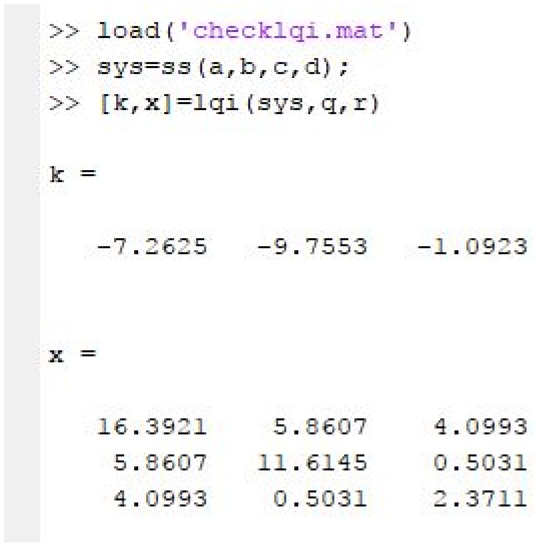
Description :- This function computes the linear quadratic integral full-state gain K for the plant .

Test Example: scilab



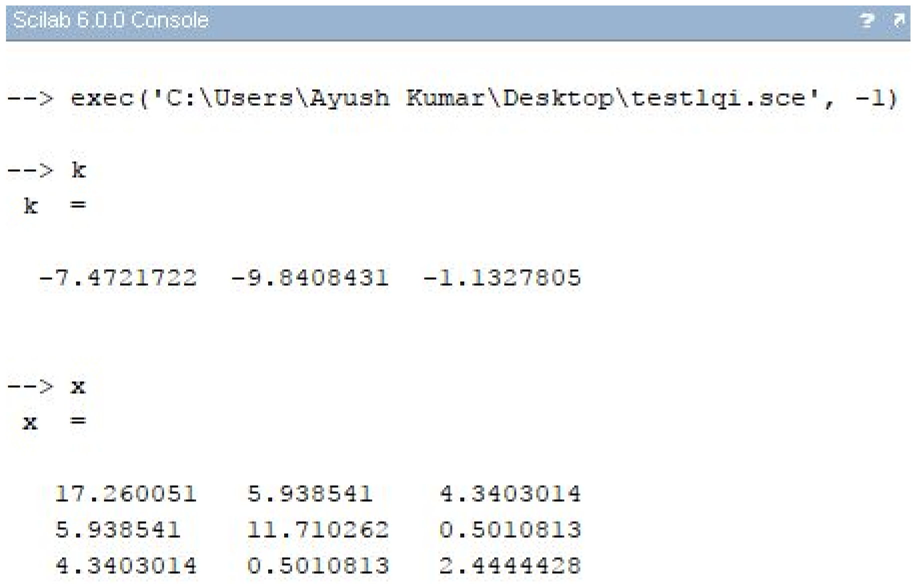
```
sys=ssrand(1,1,2);
[a,b,c,d]=abcd(sys);
q=rand(3,3);r=rand(1,1);
q=(q+q')/2;r=(r+r')/2; //making them symmetric
savematfile('checklqi.mat','a','b','c','d','q','r')
[k,x]=lqi(sys,q,r)
```

Loading data matlab and matlab result :



```
>> load('checklqi.mat')
>> sys=ss(a,b,c,d);
>> [k,x]=lqi(sys,q,r)

k =

   -7.2625   -9.7553   -1.0923


x =

   16.3921    5.8607    4.0993
    5.8607   11.6145    0.5031
    4.0993    0.5031    2.3711
```

Scilab result:

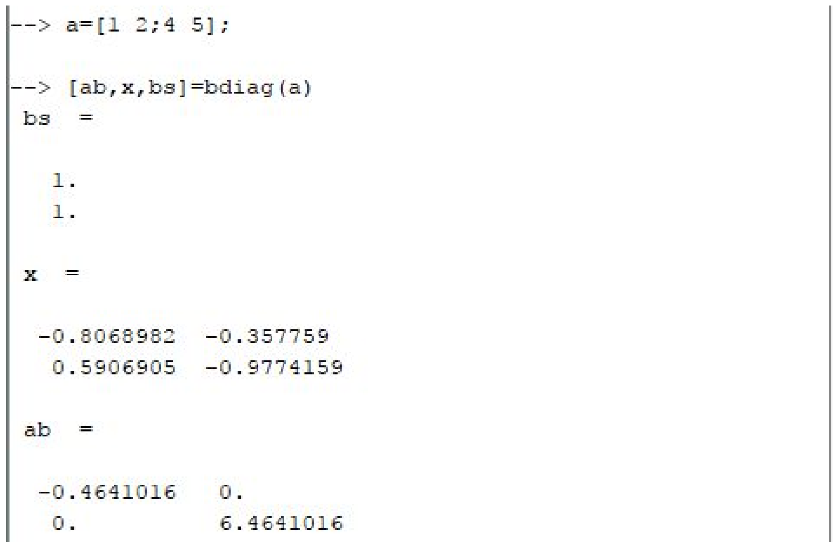


Scilab 6.0.0 Console

```
--> exec('C:\Users\Ayush Kumar\Desktop\testlqi.sce', -1)

--> k
 k  =

  -7.4721722  -9.8408431  -1.1327805


--> x
 x  =

   17.260051   5.938541    4.3403014
   5.938541    11.710262   0.5010813
   4.3403014   0.5010813   2.4444428
```

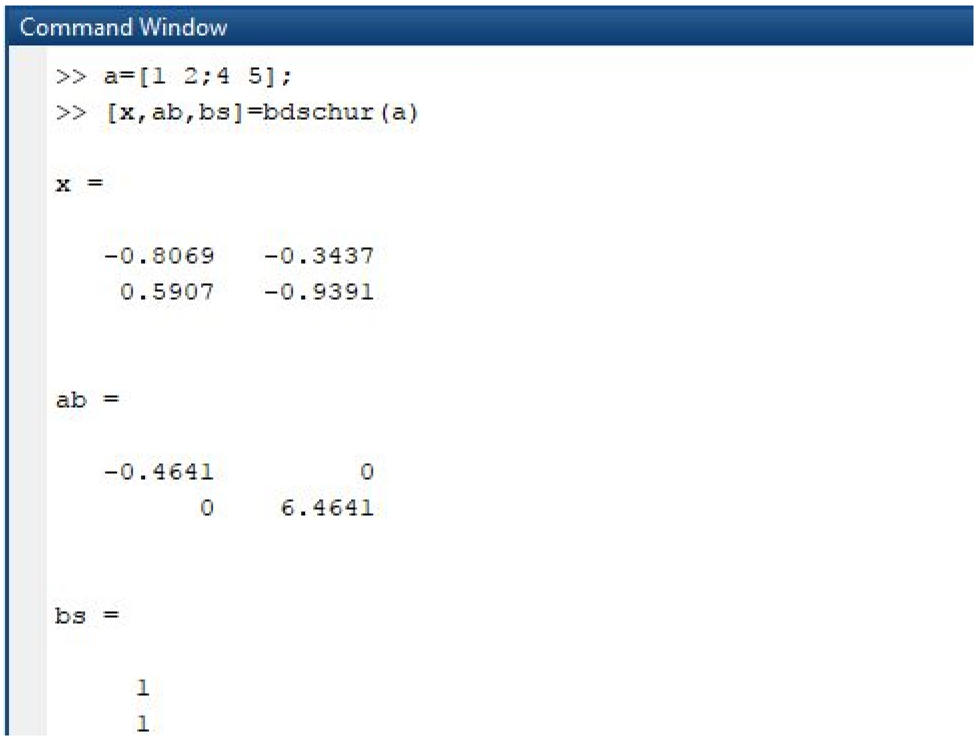
Bdschur(matlab):- bdiag(scilab)[2.6.2] :- syntax:- [ab,x,bs]=bdiag(a)
Description :- It performs the block-diagonalization of matrix a, bs gives the structure of the blocks (respective sizes of the blocks). X is the change of basis i.e ab = inv(x)ax is block diagonal.

Example:scilab



```
--> a=[1 2;4 5];

--> [ab,x,bs]=bdiag(a)
 bs  =

   1.
   1.

 x  =

  -0.8068982  -0.357759
   0.5906905  -0.9774159

 ab  =

  -0.4641016   0.
   0.          6.4641016
```

Matlab result:



```
>> a=[1 2;4 5];
>> [x,ab,bs]=bdschur(a)

x =

   -0.8069   -0.3437
    0.5907   -0.9391


ab =

   -0.4641         0
         0    6.4641


bs =

     1
     1
```

Permute(matlab):- permute(scilab) [2.6.3]

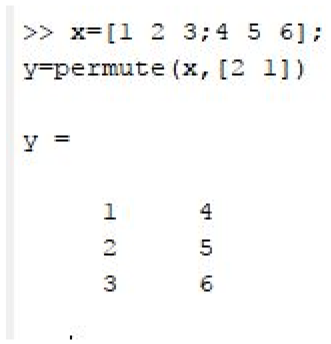
syntax: y=permute(x,dims)

Description Permutes the dimensions of an array according to dims.

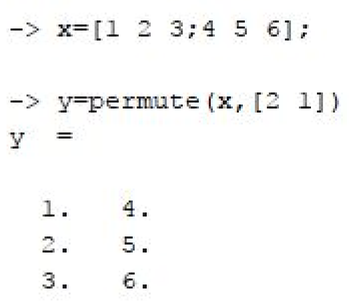
Input argument dims must be a valid permutation of 1:n>=nmin where n is the number of dimensions of the desired array, at least as many as nmin = ndims(x).

example:-

Matlab result   scilab result



```
>> x=[1 2 3;4 5 6];
y=permute(x,[2 1])

y =

     1     4
     2     5
     3     6
```



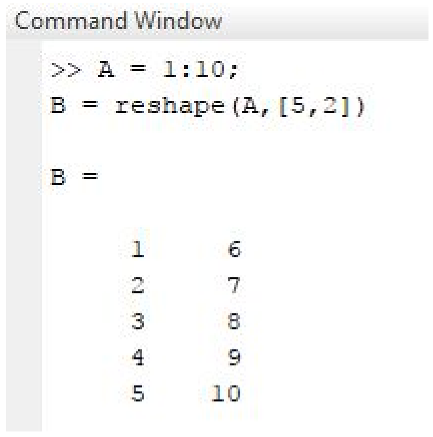
```
-> x=[1 2 3;4 5 6];

-> y=permute(x,[2 1])
y  =

  1.   4.
  2.   5.
  3.   6.
```

Reshape(matlab):-matrix(scilab)[2.6.4] :- syntax :y=matrix(v,n,m)

Description :- For a vector or a matrix with n x m entries, the command y=matrix(v,n,m) or similarly y=matrix(v,[n,m]) transforms the v vector (or matrix) into an nxm matrix by stacking column wise the entries of v.

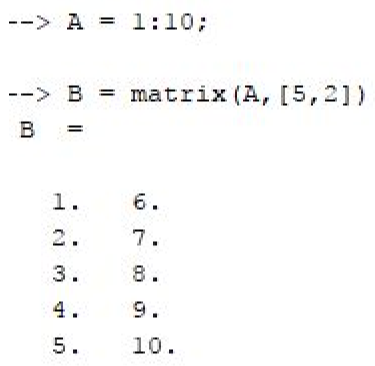
Example:

Matlab result   Scilab result



Command Window

```
>> A = 1:10;
B = reshape(A,[5,2])

B =

     1     6
     2     7
     3     8
     4     9
     5    10
```



```
--> A = 1:10;

--> B = matrix(A,[5,2])
 B  =

   1.   6.
   2.   7.
   3.   8.
   4.   9.
   5.   10.
```

Literature review:-

First i get familiar with state-space models and transfer function model ,bode plots gain and phase margins from“control system and engineering by Norman s nise”.as these are the basic thing in control systems,further i worked on stable unstable decomposition and pade approx-mation although i was not able to completely understand pade approximation and how   to implement it, a good explanation of stable unstable decomposition of a system was given in the paper which was used in stabsep function,then i studied about hankel singular values and balanced minimal realization from which was used in hankel function ,i read about controll-ability and observability grammians and how they are evaluated using lyapunov equation and further the above concepts were used in model order reduction of a system using BPSA approximation methord.For random binary signal generation several approach are available one was lcg algorithm further i studied about butterworth filter,low pass high pass and band pass filter and analog filter and conversion from analog to digital filter in scilab and iir function of scilab and how to use it as a butterworth filter.For lqrd and lqry , estim , reg and dare,   i studied about linear quadratic regulator,kalman filter,general algebraic riccati equations in continuous and discrete time domain feedback control from matlab and scilab documentation and above mentioned book,also some information about it was given in “V.Golub”,numerous information about how to implement objects,,use of cell array in scilab and structure and concept about overloading function and its use was available on scilab help page and information from matlab documentation were used in implementing nmodels,realp, genmat,stack functions.

Problems faced and suggestions:

As a lot of resources are available on net for any topic it becomes harder to find the relevant source,also as i was working alone on control system toolbox i think a team of 2 or 3 people would have worked more productively and efficiently,as in that case apart from mentors we could approach team members also in case of any doubt or question.

BIBLIOGRAPHY:-

[1.1.1]http://in.mathworks.com/help/ident/ref/balred.html?s_tid=doc_ta
[1.1.2]https://en.wikipedia.org/wiki/Model_order_reduction
[**1.1.3**]http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=261486
[**1.1.4**]http://web.iitd.ac.in/~janas/courses/material/eel879/sp_topics_01.pdf

[1.2.1]http://in.mathworks.com/help/control/ref/stabsep.html?searchHighlight=stabsep&s_tid=doc_srchtitle
[1.2.2]https://help.scilab.org/docs/5.3.2/fr_FR/dtsi.html
[1.2.3]http://www.incois.gov.in/proceedings/4.4%20backupofsksingsknagar.pdf

[1.3.1]https://en.wikipedia.org/wiki/Hankel_singular_value
[1.3.2]http://web.iitd.ac.in/~janas/courses/material/eel879/sp_topics_01.pdf
[1.3.3]https://help.scilab.org/doc/5.3.3/en_US/hankelsv.html

[1.4.1]http://in.mathworks.com/help/control/ref/hsvplot.html?searchHighlight=hsvplot&s_tid=doc_srchtitle

[1.5.1]https://en.wikipedia.org/wiki/Binary_sequence
[1.5.2]https://en.wikipedia.org/wiki/Linear_congruential_generator
[1.5.3]http://in.mathworks.com/help/signal/ref/butter.html?searchHighlight=butter&s_tid=doc_srchtitle

[1.6.1]http://in.mathworks.com/help/control/ref/lqry.html?searchHighlight=lqry&s_tid=doc_srchtitle

[1.7.1]http://in.mathworks.com/help/control/ref/lqrd.html?searchHighlight=lqrd&s_tid=doc_srchtitle
[1.7.2]http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1101743&tag=1

[1.8.1]http://in.mathworks.com/help/control/ref/estim.html?searchHighlight=estim&s_tid=doc_srchtitle
[1.8.2]https://octave.sourceforge.io/control/function/estim.html

[1.9.1] http://in.mathworks.com/help/control/ref/reg.html?searchHighlight=reg&s_tid=doc_srchtitle
[1.9.2]http://web.mit.edu/16.31/www/Fall06/1631_topic17.pdf

[2.0.1]http://in.mathworks.com/help/control/ref/nmodels.html?searchHighlight=nmodels&s_tid=doc_srchtitle

[2.1.1]https://help.scilab.org/doc/5.5.2/en_US/evans.html
[2.1.2]https://help.scilab.org/doc/5.5.2/en_US/gca.html

[2.2.1]http://in.mathworks.com/help/control/ref/dare.html?searchHighlight=dare&s_tid=doc_srchtitle
[2.2.2]http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1457351&tag=1

[2.3.1]http://in.mathworks.com/help/control/ref/stack.html?searchHighlight=stack&s_tid=doc_srchtitle

[2.4.1]http://in.mathworks.com/help/control/ref/realp.html?searchHighlight=realp&s_tid=doc_srchtitle
[2.4.2]https://help.scilab.org/doc/5.3.3/en_US/tlist.html
[2.4.3]https://help.scilab.org/doc/5.5.2/en_US/overloading.html

[2.5.1]http://in.mathworks.com/help/control/ref/genmat.html?searchHighlight=genmat&s_tid=doc_srchtitle
[2.5.2]https://help.scilab.org/doc/5.5.2/en_US/struct.html
[2.5.3]https://help.scilab.org/doc/5.5.2/en_US/overloading.html

[2.6.1]https://help.scilab.org/docs/6.0.0/en_US/lqi.html

[2.6.2]https://help.scilab.org/docs/6.0.0/en_US/bdiag.html

[2.6.3]https://help.scilab.org/docs/6.0.0/en_US/permute.html

[2.6.4]https://help.scilab.org/docs/6.0.0/en_US/matrix.html

*****https://wiki.scilab.org/Guidelines%20To%20Design%20a%20Module