Contents

List of Scilab Code

Chapter 1

Implementing 'Two Degrees of Freedom'Controller for First order systems on a Single Board Heater System

The aim of this experiment is to implement a 2DOF controller on a single board heater system. The target group is anyone who has basic knowledge of Control Engineering.

1.1 About this Experiment

We have used Scilab with Scicos as an interface for sending and receiving data. This interface is shown in Fig.1.1. Fan speed and Heater current are the two inputs to the system. For this experiment, the heater current is used as a control effort generated by inputting the various 2-DOF controller parameters like Rc, Sc, Tc and gamma. The fan input could be thought of as an external disturbance.

1.2 Theory

Degree of freedom as far as the control theory is concerned is the number of parameters on which the plant is no more dependent or the number of parameters that are free to vary. This means that a higher degree of freedom controller makes

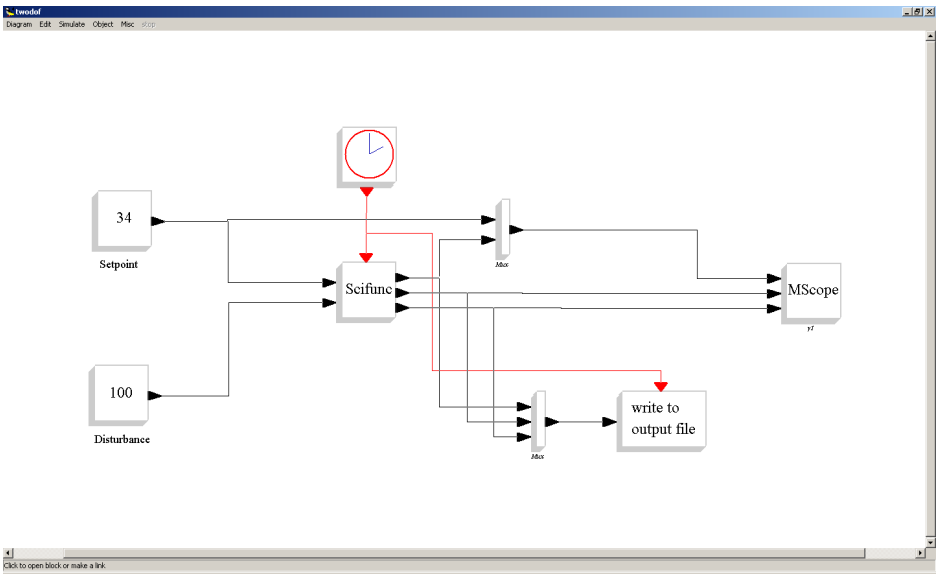

Figure 1.1: Scicos interface for this experiment

the plant less susceptible to disturbances. Controllers are broadly classified as feedback and feed forward controllers. Feedback controllers are further classified as 'One Degree of Freedom controller 'and 'Two Degree of Freedom controller '. Feed forward controllers are those who take the control action before a disturbance disturbs the plant. But this implies an ability to sense the disturbance. Moreover, exact knowledge about the plant is also needed. Nevertheless, due to these restrictions, it is rarely used alone. A feedback control strategy is as shown in figure 1.2. The reference and the output is continuously compared to generate error which is fed to the controller to take the appropriate control action. Here, exact knowledge about the plant, $G(z)$ and the disturbance, v is not necessary. Solving for y(n), we get

$$y(n) = \frac{G(z)G_c(z)}{1 + G(z)G_c(z)} r(n) + \frac{1}{1 + G(z)G_c(z)} v(n) \tag{1.1}$$

let,

$$T(z) = \frac{G(z)G_c(z)}{1 + G(z)G_c(z)} \tag{1.2}$$

$$S(z) = \frac{1}{1 + G(z)G_c(z)} \tag{1.3}$$

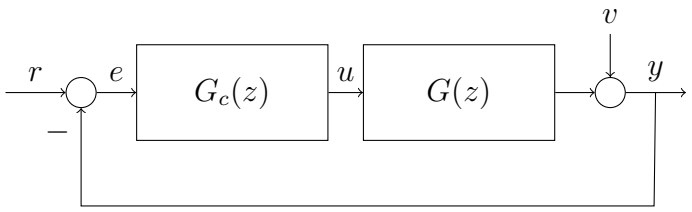

Figure 1.2: Feed back control strategy

this implies

$$y(n) = T(z)r(n) + S(z)v(n) \tag{1.4}$$

Here it could be seen that the controller has to track the reference input as well as eliminate the effect of external disturbance. But, however from the above equation it could be seen that

$$S + T = 1 \tag{1.5}$$

Hence it is not possible to achieve both of the requirements, simultaneously in this particular control arrangement. This control arrangement is called `One Degree of Freedom`, abbreviated as 1-DOF. A `Two Degrees of Freedom` strategy is as shown in figure 1.3. Here, G_b and G_f together constitute the controller. G_b is in the feedback path and is used to eliminate the effect of disturbances, whereas,G_f is in the feed forward path and is used to help the output track reference input. We need a control law something of the form,

$$R_c(z)u(n) = T_c(z)r(n) - S_c(z)y(n) \tag{1.6}$$

The terms R_c, S_c and T_c are all in polynomials of z^{-1}.

It could be seen that,

$$G_b = \frac{S_c}{R_c} \tag{1.7}$$

and

$$G_f = \frac{T_c}{R_c} \tag{1.8}$$

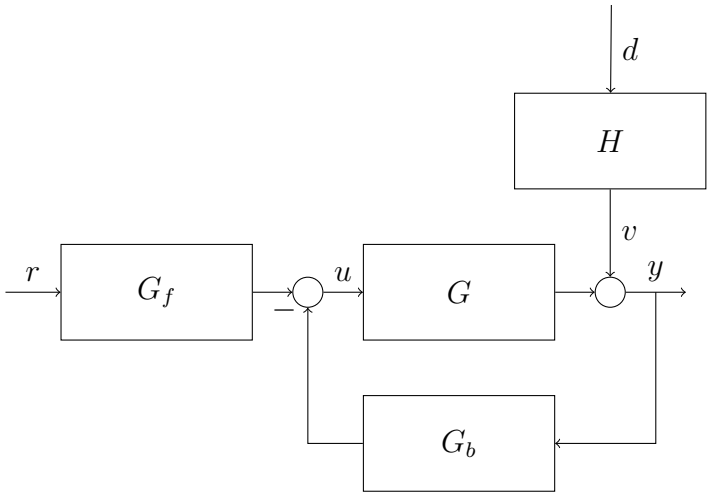

Figure 1.3: 2DOF Feed back control strategy

Consider a plant with model

$$A(z)y(n) = z^{-k}B(z)u(n) + v(n) \tag{1.9}$$

Substituting equation 1.6 in equation 1.9, we get

$$Ay(n) = z^{-k}\frac{B}{R_c}\Big[T_c r(n) - S_c y(n)\Big] + v(n) \tag{1.10}$$

solving for $y(n)$, we get

$$\left(\frac{R_c A + z^{-k}BS_c}{R_c}\right)y(n) = z^{-k}\frac{BT_c}{R_c}r(n) + v(n) \tag{1.11}$$

This can also be written as

$$y(n) = z^{-k}\frac{BT_c}{\phi_{cl}}r(n) + \frac{R_c}{\phi_{cl}}v(n) \tag{1.12}$$

where

$$\phi_{cl} = R_c(z)A(z) + z^{-k}B(z)S_c(z) \tag{1.13}$$

and is known as the closed-loop characteristic polynomial.

Now, we want the following conditions to be satisfied.

1. The zeros of ϕ_{cl} should be inside the unit circle, so that the closed-loop system becomes stable.

2. The value of $z^{-k}\frac{BT_c}{\phi_{cl}}$ must be close to unity so that reference tracking is achieved

3. The value of $\frac{R_c}{\phi_{cl}}$ must be as small as possible to achieve disturbance rejection

We would now see the pole placement controller approach to design a 2DOF controller.[1]

1.3 Designing 2-DOF controller using pole placement control approach

A 2DOF pole placement controller is as shown in the figure 1.4 It should be noted

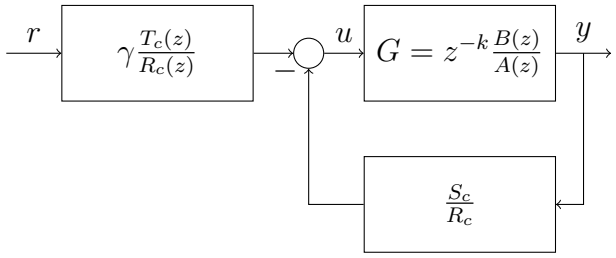

Figure 1.4: 2-DOF pole placement controller

that the effect of external disturbance will not be considered for this section. We want the closed loop transfer function to behave in such a way so that the output y is related to the setpoint r in the following manner

$$Y_m(z) = \gamma z^{-k}\frac{B_r}{\phi_{cl}}R(z) \tag{1.14}$$

Here, $Y_m(z)$ means the model output. ϕ_{cl} is nothing but the closed loop characteristic polynomial obtained by the desired location analysis.

The value of gamma is chossen in such a way so that at steady-state the output of the model is equal to the setpoint.

$$\gamma = \frac{\phi_{cl(1)}}{B_r(1)} \tag{1.15}$$

Simplifying the block diagram shown in figure 1.4 yields

$$Y = \gamma z^{-k} \frac{BT_c}{AR_c + z^{-k}BS_c} R \tag{1.16}$$

Here we have dropped the argument of z for convineance

On comparing equation 1.14 and 1.16 we can see that

$$\frac{BT_c}{AR_c + z^{-k}BS_c} = \frac{B_r}{\phi_{cl}} \tag{1.17}$$

Here after factorisation of the LHS we can expect some cancellations between the numerator and the denominator thereby making the $degB_r < degB$. But the cancellations ,if any, must be between $stable$ poles and zeros. One should avoid the cancellation of an unstable pole with a zero.

Hence, we differenciate the factors as $good$ and bad factors. Therefore we write A and B as

$$A = A^g A^b \tag{1.18}$$

$$B = B^g B^b \tag{1.19}$$

We also split R_c, S_c and T_c as shown

$$R_c = B^g R_1 \tag{1.20}$$

$$S_c = A^g S_1 \tag{1.21}$$

$$T_c = A^g T_1 \tag{1.22}$$

Hence, the equation 1.17 becomes

$$\frac{B^g B^b A^g T_1}{A^g A^b B^g R_1 + z^{-k} B^g B^b A^g S_1} = \frac{B_r}{\phi_{cl}} \tag{1.23}$$

After appropriate cancellations, we obtain

$$\frac{B^b T_1}{A^b R_1 + z^{-k} B^b S_1} = \frac{B_r}{\phi_{cl}} \tag{1.24}$$

Equating the LHS and RHS of equation 1.24 we obtain

$$B^b T_1 = B_r \tag{1.25}$$

$$A^b R_1 + z^{-k} B^b S_1 = \phi_{cl} \tag{1.26}$$

Equation 1.26 is known as the aryabhatta's identity and can be used to solve for R_1 and S_1. There are many options to choose for the value of T_1. By choosing T_1 to be equal to S_1 the 2-DOF controller is reduced to 1-DOF controller. We usually choose T_1=1.

Equation 1.25 becomes

$$B^b = B_r \tag{1.27}$$

hence the expression of gamma is now changed to

$$\gamma = \frac{\phi_{cl(1)}}{B^b(1)} \tag{1.28}$$

and the desired closed loop transfer function now becomes

$$Y_m(z) = \gamma z^{-k} \frac{B^b}{\phi_{cl}} R(z) \tag{1.29}$$

This implies that the open loop model imposes two limitations on the closed loop model.

- The bad portion of the open loop model cannot be cancelled out and it appears in the closed loop model.
- The open loop plant delay cannot be removed or minimized,i.e. the closed loop model cannot be made faster then the open loop model.

1.4 Step by step procedure to design and implement a 2-DOF controller

We obtain a first order transfer function of the plant using the step test approach.The model so obtained is

$$G(s) = \frac{0.5}{58s + 1} \tag{1.30}$$

with time constant $\tau = 58sec$ and gain $K = 0.5$

After discretisation with sampling time = 1 second, we obtain

$$G(z) = \frac{0.0085468}{z - 0.982906} \tag{1.31}$$

$$= \frac{0.0085468z^{-1}}{1 - 0.982906z^{-1}} \tag{1.32}$$

Discretisation can be done using the scilab code `c2d.sce`. We would now define good and bad terms

$$\begin{aligned} A^g &= 1 - 0.982906z^{-1} \\ A^b &= 1 \\ B^g &= 0.0085468 \\ B^b &= 1 \end{aligned}$$

Let us now define the transient specifications. We choose,

$$\text{Rise time} = 102 \text{ seconds}$$

No. of samples per rise time (N_r) is calculated as

$$\begin{aligned} N_r &\leq \frac{\text{Rise time}}{\text{Sampling time}} \\ &= 127 \end{aligned}$$

next

$$\begin{aligned} \omega &= \frac{\pi}{2N_r} \\ &= 0.0123 \end{aligned}$$

We choose,

$$\begin{aligned} Overshoot(\epsilon) &= 0.05.....i.e5\% \\ \rho &\leq \epsilon^{\omega/\pi} \\ &= 0.988 \end{aligned}$$

Let us now calculate 2DOF Controller parameters. The closed loop characteristic polynomial is given by

$$\begin{aligned} \phi_{cl} &= 1 - z^{-1}2\rho cos\omega + \rho^2 z^{-2} \\ &= 1 - 1.97639z^{-1} + 0.97668z^{-2} \end{aligned}$$

But according to equation 1.26

$$A^b R_1 + z^{-k} B^b S_1 = \phi_{cl}$$

Recall that we had not considered external disturbance in the block diagram shown in figure1.4. However, we can still, up to some extent, take care of the disturbances. This is achieved by using the internal model principle. If a model of step is present inside the loop, step disturbances can be rejected. We can apply this by forcing R_c to have this term. A step model is given by

$$1(z) = \frac{1}{1 - z^{-1}}$$

Let the denominator of the step model be denoted as Δ

$$\Delta = 1 - z^{-1}$$

Therefore,

$$R_c = B^g \Delta R_1$$

Δ has a root which lies on the unit circle. Hence it has to be treated as a bad part and should not be cancelled out. Hence, we should make sure that all of the occurences of R_1 have this term.

Therefore,

$$\phi_{cl} = A^b \Delta R_1 + z^{-k} B^b S_1 \tag{1.33}$$

Hence,

$$A^b \Delta R_1 + z^{-k} B^b S_1 = 1 - 1.97639 z^{-1} + 0.97668 z^{-2}$$

The expression, however, does not satisfy the conditions required for solving the Aryabhatta Identity.

Let,

$$R_1 = 1 - 0.97668 z^{-1}$$

therefore

$$S_1 = 0.00029$$
$$R_c = B^g \Delta R_1$$

therefore

$$R_c = 0.009 - 0.01779 z^{-1} + 0.00879 z^{-2}$$
$$S_c = A^g S_1$$

hence

$$S_c = 0.0002887 - 0.0002838 z^{-1}$$
$$T_c = A^g T_1$$

therefore

$$T_c = 1 - 0.983 z^{-1}$$
$$\gamma = \frac{\phi_{cl(1)}}{B^b(1)}$$
$$= 0.0002887$$

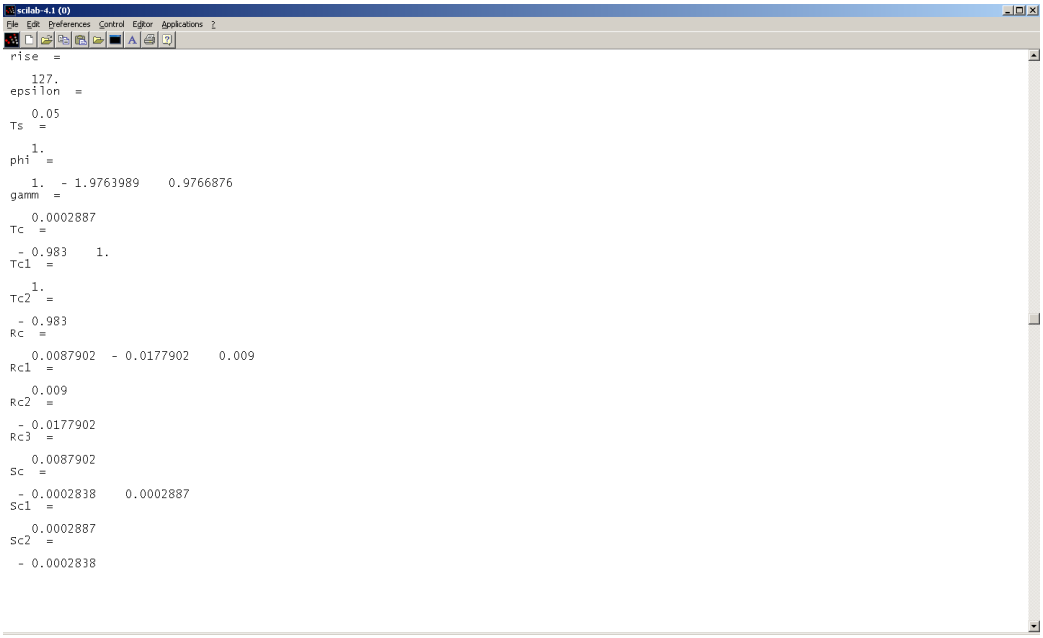

Figure 1.5: Scilab output for `2DOF_para.sce`

$\phi_{cl(1)}$ means for $z = 1$, steady-state. So, we get

$$\begin{aligned} R_c &= R_{c1} + R_{c2}z^{-1} + R_{c3}z^{-2} \\ &= 0.009 - 0.01779z^{-1} + 0.0087z^{-2} \\ S_c &= S_{c1} + S_{c2}z^{-1} \\ &= 0.000288 - 0.0002838z^{-1} \\ T_c &= T_{c1} + T_{c2}z^{-1} \\ &= 1 - 0.983z^{-1} \\ \gamma &= 0.0002887 \end{aligned}$$

Scilab code `2-DOF_para.sce` does these caluclations(Use value of `A` and `B` obtained from scilab code c2d.sce). This code utilizes various other scilab codes provided at the end of this document. [1] After execution of `2-DOF_para.sce`, execute the `2-DOF.sce` code in scilab. Now Run the Scicos code `2-DOF.cos` with required setpoint value and observe the temperature profile. Make sure that

[1] NOTE:- The scilab codes are given at the end of this document.

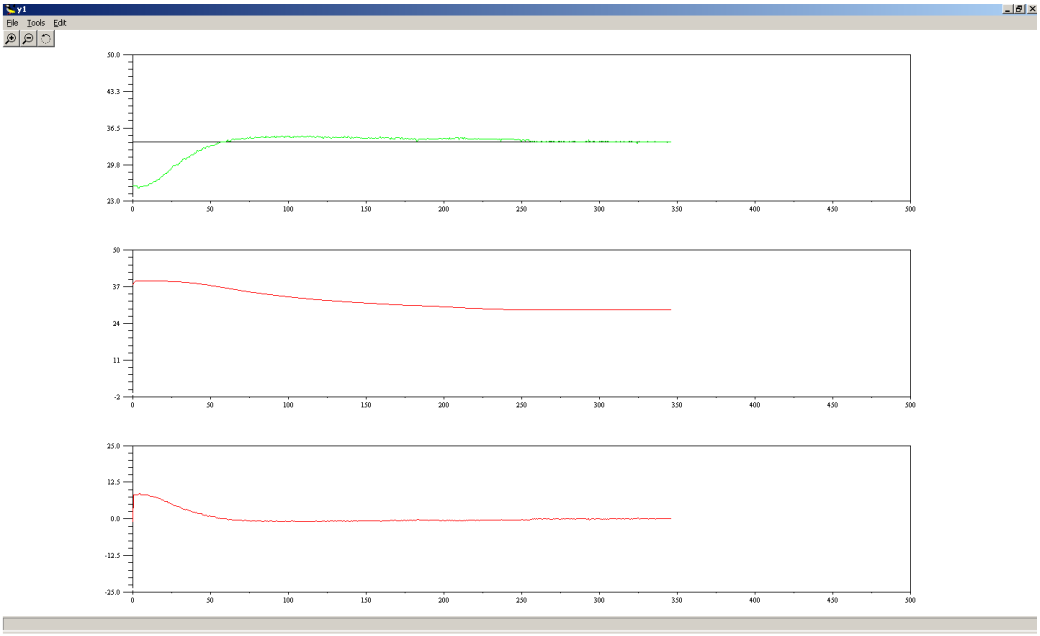

Figure 1.6: Implementation of 2DOF controller

you input the sampling time(Clock period) same as the one you used for discretisation of your plant. It could be seen that the output (temperature) tracks the setpoint irrespective of the step changes in the fan speed. We can see that the Over shoot turns out to be 6% and rise time turns out to be 60 seconds, which is acceptable.

1.5 Scilab Codes

Scilab Code 1.5.1 `c2d.sce`

```
1 s=poly(0,'s'); // Defines s to be a polynomial variable
2 TFcont = syslin('c',[0.5/(58*s+1)]) // Creating cont-
    time transfer function
3 SScont = tf2ss(TFcont); // Converting cont-time
    transfer function to state-space model
4 Ts=1; // Sampling time
5 SSdisc=dscr(SScont,Ts); // Discretizing cont-time state
```

```
    -space model
TFdisc=ss2tf(SSdisc) // Converting discr-time ss model
    to tf
```

Scilab Code 1.5.2 *2-DOF_para.sce*

```
global temp heat_in fan_in CO u_new u_old u_old_old
    r_old y_old Rc1 Rc2 Rc3 Sc1 Sc2 Tc1 Tc2 gamm

// Transfer function
B = 0.0085468; A = [1 -0.9829064]; k=1;

// Transient specifications
rise = 127
epsilon = 0.05
Ts = 1
phi = desired(Ts,rise,epsilon);

// Controller design
Delta = [1 -1]; // internal model of step used
[Rc,Sc,Tc,gamm] = pp_im(B,A,k,phi,Delta);

// parameters for twodof.cos
gamm
[Tcp1,Tcp2] = cosfil_ip(Tc,1); // Tc/1
Tc=coeff(Tcp1)
Tc1=Tc(1,2)
Tc2=Tc(1,1)
[Rcp1,Rcp2] = cosfil_ip(1,Rc); // 1/Rc
Rc=coeff(Rcp2)
Rc1=Rc(1,3)
Rc2=Rc(1,2)
Rc3=Rc(1,1)
[Scp1,Scp2] = cosfil_ip(Sc,1); // Sc/1
Sc=coeff(Scp1)
Sc1=Sc(1,2)
Sc2=Sc(1,1)
```

Scilab Code 1.5.3 *2-DOF.sci*

```
1  //2DOF Controller
2  //Heater input is passed as input argument to
       introduce control effort 'CO'
3  //Fan input is passed as input argument which is kept
       at constant level(disturbance)
4  //Range of Fan input :60 to 252
5  //Temperature is read
6
7  function [temp,CO,et] = twodof(setpoint,disturbance)
8  global temp heat_in fan_in CO u_new u_old u_old_old
       r_old y_old Rc1 Rc2 Rc3 Sc1 Sc2 Tc1 Tc2 gamm
9
10 e_new = setpoint - temp;
11 r_new = setpoint;
12 y_new = temp;
13
14 et=setpoint-temp;
15
16
17 u_new = (1/Rc1)*(gamm*Tc1*r_new + gamm*Tc2*r_old-Sc1*
       y_new -Sc2*y_old-Rc2*u_old - Rc3*u_old_old);
18
19 CO = u_new;
20
21    if CO>39
22      CO = 39;
23    end;
24
25    if CO<0
26      CO =0;
27    end;
28
29 u_new = CO;
30
31 u_old_old = u_old;
32 u_old = u_new;
```

```
r_old = r_new;
y_old = y_new;

heat_in = CO;
fan_in = disturbance;

ok = writebincom(handl,[254]); // heater
ok = writebincom(handl,[heat_in]);
ok = writebincom(handl,[253]); // fan
ok = writebincom(handl,[fan_in]);
ok = writebincom(handl,[255]);
sleep(1);
[temp,ok,nbytes] = readbincom(handl,2);
temp = temp(1) + 0.1*temp(2);


endfunction;
```

Scilab Code 1.5.4 *cindep.sci*

```
// Updated ----No change
// function b = cindep( S, gap )
// used in XD + YN = C. all rows except the last of
    are assumed to
// be independent. The aim is to check if the last
    row is dependent on the
// rest and if so how. The coefficients of dependence
     is sent in b
function b = cindep( S,gap)

if argn(2) == 1
        gap = 1.0e8;
end
eps = 2.2204e-016;
[rows,cols] = size(S);
if rows > cols
    ind = 0;
else
```

```
16      sigma = svd(S);
17      len = length(sigma);
18      if (sigma(len)/sigma(1) <= (eps*max(i,cols)))
19          ind = 0;                        // not independent
20      else
21          if or(sigma(1:len-1) ./sigma(2:len)>=gap)
22              ind = 0;                    // not dependent
23          else
24              ind = 1;                    // independent
25          end
26      end
27  end
28  if ind
29      b = [];
30  else
31      b = S(rows,:)/S(1:rows-1,:);
32      b = makezero(b,gap);
33  end
34  endfunction
```

Scilab Code 1.5.5 `clcoef.sci`

```
1   // Updated -----No change
2   // H. Kwakernaak, July, 1990
3   // Modified by Kannan Moudgalya in Nov. 1992
4
5   function [P,degP] = clcoef(Q,degQ)
6
7   [rQ,cQ] = polsize(Q,degQ);
8
9   if and(and(Q==0))
10    P = zeros(rQ,cQ);
11    degP = 0;
12  else
13    P = Q; degP = degQ; rP = rQ; cP = cQ;
14    j = degP+1;
15    while j >= 0
16    X = P(:,(j-1)*cP+1:j*cP)
```

```
17        if max(sum(abs(X'))) < (1e-8)*max(sum(abs(P)))
18            P = P(:,1:(j-1)*cP);
19            degP = degP-1;
20        else
21            j = 0;
22        end
23        j = j-1;
24    end
25  end
26  endfunction
```

Scilab Code 1.5.6 `clcoef.sci`

```
1   // Updated -----No change
2   // H. Kwakernaak, July, 1990
3   // Modified by Kannan Moudgalya in Nov. 1992
4
5   function [P,degP] = clcoef(Q,degQ)
6
7   [rQ,cQ] = polsize(Q,degQ);
8
9   if and(and(Q==0))
10    P = zeros(rQ,cQ);
11    degP = 0;
12  else
13    P = Q; degP = degQ; rP = rQ; cP = cQ;
14    j = degP+1;
15    while j >= 0
16    X = P(:,(j-1)*cP+1:j*cP)
17        if max(sum(abs(X'))) < (1e-8)*max(sum(abs(P)))
18            P = P(:,1:(j-1)*cP);
19            degP = degP-1;
20        else
21            j = 0;
22        end
23        j = j-1;
24    end
25  end
```

```
endfunction
```

Scilab Code 1.5.7 cosfil_ip.sci

```
// Updated(31-7-07)
// Input arguments are numerator and denominator
// polynomials' co efficients in ascending
// powers of z^-1

// Scicos blocks need input polynomials
// with positive powers of z

function [nume,deno] = cosfil_ip(num,den)

[Nn,Nd] = polyno(num,'z');
[Dn,Dd] = polyno(den,'z');
nume = Nn*Dd;
deno = Nd*Dn;

endfunction;
```

Scilab Code 1.5.8 desired.sci

```
// Updated(26-7-07)
// 9.4
function [phi,dphi] = desired(Ts,rise,epsilon)

Nr = rise/Ts; omega = %pi/2/Nr; rho = epsilon^(omega/
    %pi);
phi = [1 -2*rho*cos(omega) rho^2]; dphi = length(phi)
    -1;
endfunction;
```

Scilab Code 1.5.9 indep.sci

```
// Updated ----No change
// function b = indep(S,gap)
```

```
3  // determines the first row that is dependent on the
       previous rows of S.
4  // The coefficients of dependence is returned in b
5  function b = indep( S,gap)
6
7  if argn(2) == 1
8           gap = 1.0e8;
9           end
10 [rows,cols] = size(S);
11 ind = 1;
12 i = 2;
13 eps = 2.2204e-016;
14 while ind & i <= rows
15        sigma = svd(S(1:i,:));
16        len = length(sigma);
17        if(sigma(len)/sigma(1) < (eps*max(i,cols)))
18          ind =0;
19        else
20          shsig = [sigma(2:len);sigma(len)];
21          if or( (sigma ./shsig) > gap)
22              ind = 0;
23          else
24              ind = 1;
25              i = i+1;
26          end
27        end
28
29 end
30 if ind
31          b =[];
32
33 else
34          c = S(i,:)/S(1:i-1,:);
35          c = makezero(c,gap);
36          b = [-c 1];
37 end
38 endfunction
```

Scilab Code 1.5.10 *left_prm.sci*

```
1  //  function [B,degB,A,degA,Y,degY,X,degX] = ...
2  //  left_prm(N,degN,D,degD,job,gap)
3  //
4  //  does three different things according to integers that 'job' takes
5  //  job = 1.
6  //  this is the default. It is always done for all jobs.
7  //                 -1                                    -1
       -1
8  //  Given ND , returns coprime B and A where ND = A
       B
9  //  It is enough if one sends the first four input arguments
10 //  If gap is required to be sent, then one can send either 1 or a null
11 //  entry for job
12 //  job = 2.
13 //  first solve for job = 1 and then solve XA + YB = I
14 //  job = 3.
15 //  used in solving XD + YN = C
16 //  after finding coprime factorization, data are returned
17 //
18 //  convention: the variable with prefix deg stand for degrees
19 //  of the corresponding polynomial matrices
20 //
21 //  input:
22 //  N: right fraction numerator polynomial matrix
23 //  D: right fraction denominator polynomial matrix
24 //  N and D are not neccessarily coprime
25 //  gap: variable used to zero entries; default value is 1.0e+8
26 //
27 //  output
```

```
28 // b and A are left coprime num. and den. polynomial
       matrices
29 // X and Y are solutions to Aryabhatta identity, only
       for job = 2
30
31 function [B,degB,A,degA,Y,degY,X,degX] = left_prm(N,
       degN,D,degD,job,gap)
32 if argn(2) == 4 | argn(2) == 5
33    gap = 1.0e8 ;
34 end
35 // pause
36 if argn(2) == 4,
37 job = 1; end
38 [F,degF] = rowjoin(D,degD,N,degN);
39 [Frows,Fbcols] = polsize(F,degF);          // Fbcols =
       block columns
40 Fcols = Fbcols * (degF+1) ;                // actual
       columns of F
41 T1 = []; pr =[]; degT1 = 0; T1rows = 0; shft = 0;
42 S=F; sel = ones(Frows,1); T1bcols =1;
43 abar = (Fbcols + 1):Frows;                 // a_super_bar
       of B-C. Chang
44 while isempty(T1) | T1rows < Frows - Fbcols
45       Srows = Frows*T1bcols; // max actual columns of
             result
46       [T1,T1rows,sel,pr] = ...
47             t1calc(S,Srows,T1,T1rows,sel,pr,Frows,
                  Fbcols,abar,gap);
48       [T1rows,T1cols] = size(T1);
49       if T1rows < Frows - Fbcols
50         T1 = [T1 zeros(T1rows,Frows)];
51         T1bcols = T1bcols + 1;          // max. block
             columns of result
52         degT1 = degT1 + 1;              // degree of
             result
53         shft = shft +Fbcols;
54         S = seshft(S,F,shft);
55         sel = [sel;sel(Srows-Frows+1:Srows)];
```

```
56          rowvec = (T1bcols-1)*Frows+(Fbcols+1):T1bcols
                * Frows;
57          abar = [abar rowvec];              // A_super_bar
                of B-C. chang
58      end
59  end
60
61  [B,degB,A,degA] = colsplit(T1,degT1,Fbcols,Frows-
        Fbcols);
62  [B,degB] = clcoef(B,degB);
63  B = -B;
64  [A,degA] = clcoef(A,degA);
65  // pause
66  if job == 2
67      S = S(mtlb_logical(sel),:);
                                        // columns
68      [redSrows,Scols] = size(S);
69      C = [eye(Fbcols,Fbcols) zeros(Fbcols,Scols-
            Fbcols)];     // append with zeros
70      T2 = C/S;
71      T2 = makezero(T2,gap);
72      T2 = move_sci(T2,find(sel),Srows);
73      [X,degX,Y,degY] = colsplit(T2,degT1,Fbcols,Frows
            - Fbcols);
74      [X,degX] = clcoef(X,degX);
75      [Y,degY] = clcoef(Y,degY);
76  elseif job == 3
77      Y = S;
78      degY = sel;
79      X = degT1;
80      degX = Fbcols;
81  else
82      if job ~= 1
83          error('Message from left_prm:no legal job
                number specified')
84      end
85  end
86  endfunction
```

Scilab Code 1.5.11 *makezero.sci*

```
// Updated
// function B = makezero(B, gap)
// where B is a vector and gap acts as a tolerance

function B = makezero(B, gap)

if argn(2) == 1
    gap = 1.0e8;
end
temp = B(find(B));              // non zero entries of B
temp = -sort(-abs(temp));       // absolute values sorted in
     descending order
len = length(temp);
ratio = temp(1:len-1) ./temp(2:len); // each ratio >1
min_ind = min(find(ratio>gap));
if ~isempty(min_ind)
    our_eps = temp(min_ind+1);
    zeroind = find(abs(B)<=our_eps);
    B(zeroind) = zeros(1,length(zeroind));
end
endfunction
```

Scilab Code 1.5.12 *move_sci.sci*

```
// function result = move_sci(b, nonred, max_sci)
// Moves matrix b to matrix result with the
    information on where to move,
// decided by the indices of nonred.
// The matrix result will have as many rows as b has
    and max number of columns.
// b is augumented with zeros to have nonred number of
     columns;
// The columns of b put into those of result as
    decided by nonred.
```

```
7
8  function result = move_sci(b,nonred,max_sci)
9  [brows,bcols] = size(b);
10 b = [b zeros(brows,length(nonred)-bcols)];
11 result = zeros(brows,max_sci);
12 result(:,nonred') = b;
13 endfunction
```

Scilab Code 1.5.13 `polmul.sci`

```
1  // Updated ------No change
2  // polmul
3  // The command
4  //     [C,degA] = polmul(A,degA,B,degB)
5  // produces the polynomial matrix C that equals the
       product A*B of the
6  // polynomial matrices A and B.
7  //
8  // H. Kwakernaak, July, 1990
9
10
11 function [C,degC] = polmul(A,degA,B,degB)
12 [rA,cA] = polsize(A,degA);
13 [rB,cB] = polsize(B,degB);
14 if cA ~= rB
15     error('polmul: Inconsistent dimensions of input
          matrices');
16 end
17
18 degC = degA+degB;
19 C = [];
20 for k = 0:degA+degB
21     mi = 0;
22     if k-degB > mi
23         mi = k-degB;
24     end
25     ma = degA;
26     if k < ma
```

```
27          ma = k;
28      end
29      Ck = zeros(rA,cB);
30      for i = mi:ma
31          Ck = Ck + A(:,i*cA+1:(i+1)*cA)*B(:,(k-i)*cB
                +1:(k-i+1)*cB);
32      end
33      C = [C Ck];
34  end
35  endfunction
```

Scilab Code 1.5.14 `polsize.sci`

```
1  // Updated ---- No change
2  // function [rQ,cQ] = polsize(Q,degQ)
3  // FUNCTION polsize TO DETERMINE THE DIMENSIONS
4  // OF A POLYNOMIAL MATRIX
5  //
6  // H. Kwakernaak, August, 1990
7
8  function [rQ,cQ] = polsize(Q,degQ)
9
10 [rQ,cQ] = size(Q); cQ = cQ/(degQ+1);
11 if abs(round(cQ)-cQ) > 1e-6
12     error('polsize: Degree of input inconsistent with
           number of columns');
13 else
14     cQ = round(cQ);
15 end
16 endfunction
```

Scilab Code 1.5.15 `polsplit3.sci`

```
1  // Updated(18-7-07)
2  // 9.11
3  // function [goodpoly,badpoly] = polsplit3(fac,a)
4  // Splits a scalar polynomial of z^{-1} into good and
       bad
```

```
5  // factors. Input is a polynomial in increasing degree
       of
6  // z^{-1}. Optional input is a, where a <= 1.
7  // Factors that have roots outside a circle of radius
       a or
8  // with negative roots will be called bad and the rest
9  // good. If a is not specified, it will be assumed as
       1.
10
11 function [goodpoly,badpoly] = polsplit3(fac,a)
12 if argn(2) == 1, a = 1; end
13 if a>1 error('good polynomial also is unstable'); end
14 fac1 = poly(fac(length(fac):-1:1),'z','coeff');
15 rts = roots(fac1);
16 rts = rts(length(rts):-1:1);
17
18 // extract good and bad roots
19 badindex = mtlb_find((abs(rts)>=a-1.0e-5)|(real(rts)
       <-0.05));
20 badpoly = coeff(poly(rts(badindex),'z'));
21 goodindex = mtlb_find((abs(rts)<a-1.0e-5)&(real(rts)
       >=-0.05));
22 goodpoly = coeff(poly(rts(goodindex),'z'));
23
24 // scale by equating the largest terms
25 [m,index] = max(abs(fac));
26 goodbad = convol(goodpoly,badpoly);
27 goodbad = goodbad(length(goodbad):-1:1);
28 factor1 = fac(index)/goodbad(index);
29 goodpoly = goodpoly * factor1;
30 goodpoly = goodpoly(length(goodpoly):-1:1);
31 badpoly = badpoly(length(badpoly):-1:1);
32 endfunction;
```

Scilab Code 1.5.16 `polyno.sci`

```
1  // Updated(1-8-07)
2  // Operations:
```

```
3  // Polynomial definition
4  // Flipping of coefficients
5  // Variable ------- passed as input argument (either 's' or 'z')
6  // Both num and den are used mostly used in scicos files,
7  // to get rid of negative powers of z
8
9  // Polynomials with powers of s need to
10 // be flipped only
11
12 function [polynu,polyde] = polyno(zc,a)
13 zc = clean(zc);
14 polynu = poly(zc(length(zc):-1:1),a,'coeff');
15   if a == 'z'
16   polyde = %z^(length(zc) - 1);
17   else
18   polyde = 1;
19   end
20
21 // Scicos(4.1) Filter block shouldn't have constant/constant
22   if type(polynu)==1 & type(polyde)==1
23     if a == 'z'
24       polynu = %z; polyde = %z;
25     else
26       polynu = %s; polyde = %s;
27     end;
28   end;
29
30 endfunction
```

Scilab Code 1.5.17 *pp_im.sci*

```
1  // Updated(27-7-07)
2  // 9.8
3  // function [Rc,Sc,Tc,gamma,phit] = pp_im(B,A,k,phi,Delta)
```

```
// Calculates 2-DOF pole placement controller.
// ------
function [Rc,Sc,Tc,gamm] = pp_im(B,A,k,phi,Delta)

// Setting up and solving Aryabhatta identity
[Ag,Ab] = polsplit3(A); dAb = length(Ab) - 1;
[Bg,Bb] = polsplit3(B); dBb = length(Bb) - 1;

[zk,dzk] = zpowk(k);

[N,dN] = polmul(Bb,dBb,zk,dzk);
dDelta = length(Delta)-1;
[D,dD] = polmul(Ab,dAb,Delta,dDelta);
dphi = length(phi)-1;

[S1,dS1,R1,dR1] = xdync(N,dN,D,dD,phi,dphi);

// Determination of control law
Rc = convol(Bg,convol(R1,Delta)); Sc = convol(Ag,S1);
Tc = Ag; gamm = sum(phi)/sum(Bb);
endfunction;
```

Scilab Code 1.5.18 *rowjoin.sci*

```
// Updated --------No change
// function [P,degP] = rowjoin(P1,degP1,P2,degP2)
// MATLAB FUNCTION rowjoin TO SUPERPOSE TWO POLYNOMIAL
// MATRICES

// H. Kwakernaak, July, 1990

function [P,degP] = rowjoin(P1,degP1,P2,degP2)

[rP1,cP1] = polsize(P1,degP1);
[rP2,cP2] = polsize(P2,degP2);
if cP1 ~= cP2
  error('rowjoin: Inconsistent numbers of columns');
end
```

```
15
16 rP = rP1+rP2; cP = cP1;
17 if degP1 >= degP2
18     degP = degP1;
19 else
20     degP = degP2;
21 end
22
23 if isempty(P1)
24     P = P2;
25 elseif isempty(P2)
26     P = P1;
27 else
28     P = zeros(rP,(degP+1)*cP);
29     P(1:rP1,1:(degP1+1)*cP1) = P1;
30     P(rP1+1:rP,1:(degP2+1)*cP2) = P2;
31 end
32 endfunction
```

Scilab Code 1.5.19 `seshft.sci`

```
1 // Updated ---------No change
2 // function C = seshft(A,B,N)
3 // given A and B matrices, returns C = [<-A-> 0
4 //                                      0   <-B->] with B
      shifted east by N cols
5
6 function C = seshft(A,B,N)
7 [Arows,Acols] = size(A);
8 [Brows,Bcols] = size(B);
9 if N >= 0
10     B = [zeros(Brows,N) B];
11     Bcols = Bcols + N;
12 elseif N < 0
13     A = [zeros(Arows,abs(N)) A];
14     Acols = Acols +abs(N);
15 end
16   if Acols < Bcols
```

```
17          A = [A zeros(Arows,Bcols-Acols)];
18      elseif Acols > Bcols
19          B = [B zeros(Brows,Acols-Bcols)];
20      end
21      C = [A
22           B];
23  endfunction
```

Scilab Code 1.5.20 `t1calc.sci`

```
1  // Updated
2  // function [T1,T1rows,sel,pr] = ...
3  // t1calc(S,Srows,T1,T1rows,sel,pr,Frows,Fbcols,abar,
       gap)
4  // calculates the coefficient matrix T1
5  // redundant row information is kept in sel: redundant
       rows are marked
6  // with zeros. The undeleted rows are marked with
       ones.
7
8  function [T1,T1rows,sel,pr] = t1calc(S,Srows,T1,T1rows
       ,sel,pr,Frows,Fbcols,abar,gap)
9  b = 1;                                    // vector of
       primary red. rows
10
11 while (T1rows < Frows - Fbcols) & or(sel==1) & ~
       isempty(b)
12      S = clean(S);
13      b = indep(S(mtlb_logical(sel),:),gap); // send
            selected rows of S
14      if ~isempty(b)
15          b = clean(b);
16          b = move_sci(b,find(sel),Srows);
17          j = length(b);
18          while ~(b(j) & or(abar==j))     // pick largest
                nonzero entry
19              j = j-1;                    // of coeff.
                    belonging to abar
```

```
20                  if ~j
21                      fprintf('\nMessage from t1calc,
                            called from left_prm\n\n')
22                      error('Denominator is noninvertible
                            ')
23                  end
24              end
25              if ~or(j<pr & pmodulo(pr,Frows) == pmodulo(j,
                    Frows)) // pr(2),pr(1)
26                  T1 = [T1; b];                   // condition
                        is not violated
27                  T1rows = T1rows +1;             // accept this
                        vector
28              end                                 // else don't
                    accept
29              pr = [pr; j];                       // update
                    prime red row info
30              while j <= Srows
31                  sel(j) = 0;
32                  j = j + Frows;
33              end
34          end
35  end
36  endfunction
```

Scilab Code 1.5.21 *xdync.sci*

```
1  // Updated ----No change
2  // function [Y,degY,X,degX,B,degB,A,degA] = xdync(N,
       degN,D,degD,C,degC,gap)
3  // given coefficient matrix in T1, primary redundant
       row information sel,
4  // solves XD + YN = C
5
6  // calling order changed on 16 April 2005. Old order:
7  // function [B,degB,A,degA,Y,degY,X,degX] = xdync(N,
       degN,D,degD,C,degC,gap)
8
```

```
function [Y,degY,X,degX,B,degB,A,degA] = xdync(N,degN,
    D,degD,C,degC,gap)
if argn(2) == 6
        gap = 1.0e+8;
end

[F,degF] = rowjoin(D,degD,N,degN);

[Frows,Fbcols] = polsize(F,degF); // Fbcols = block
    columns

[B,degB,A,degA,S,sel,degT1,Fbcols] = left_prm(N,degN,D
    ,degD,3,gap);
// if issoln(D,degD,C,degC,B,degB,A,degA)
        [Crows,Ccols] = size(C);
        [Srows,Scols] = size(S);
        S = clean(S);
        S = S(mtlb_logical(sel),:);
        T2 =[];

        for i = 1:Crows,
                Saug = seshft(S,C(i,:),0);
                b = cindep(Saug);
                b = move_sci(b,find(sel),Srows);
                T2 =[T2; b];
        end

    [X,degX,Y,degY] = colsplit(T2,degT1,Fbcols,Frows-
        Fbcols);

    [X,degX] = clcoef(X,degX);
    [Y,degY] = clcoef(Y,degY);
     Y = clean(Y); X = clean(X);
endfunction
```

Scilab Code 1.5.22 *zpowk.sci*

```
// Updated(26-7-07)
```

```
2  //  9.6
3  //  ------
4
5  function [zk,dzk] = zpowk(k)
6  zk = zeros(1,k+1); zk(1,k+1) = 1;
7  dzk = k;
8  endfunction
```