Scilab Textbook Companion for Automatic Control Systems by B. C. Kuo And F. Golnaraghi [1]

Created by
Arpita V Huddar
B.Tech (pursuing)
Electronics Engineering
NIT Karnataka
College Teacher
S.Rekha
Cross-Checked by
Sonanya Tatikola, IITB

August 9, 2013

[1]Funded by a grant from the National Mission on Education through ICT, http://spoken-tutorial.org/NMEICT-Intro. This Textbook Companion and Scilab codes written in it can be downloaded from the "Textbook Companion Project" section at the website http://scilab.in

Book Description

Title: Automatic Control Systems

Author: B. C. Kuo And F. Golnaraghi

Publisher: Princton Hall Of India Private Limited, New Delhi

Edition: 7

Year: 1995

ISBN: 81-203-0968-5

Scilab numbering policy used in this document and the relation to the above book.

Exa Example (Solved example)

Eqn Equation (Particular equation of the above book)

AP Appendix to Example(Scilab Code that is an Appednix to a particular Example of the above book)

For example, Exa 3.51 means solved example 3.51 of this book. Sec 2.3 means a scilab code whose theory is explained in Section 2.3 of the book.

Contents

List of Scilab Codes 4

2 Mathematical Foundation 9

3 Transfer Functions Block Diagrams and Signal Flow Graphs 18

4 Mathematical Modelling of Physical Systems 26

5 State Variable Analysis 31

6 Stability of Linear Control Systems 37

7 Time Domain Analysis of Control Systems 44

8 Root Locus Technique 53

9 Frequency Domain Analysis 81

List of Scilab Codes

Exa 2.1 laplace transform of step function . . . 9
Exa 2.2 laplace transform of exponential function . . . 9
Exa 2.3 final value thereom . . . 9
Exa 2.4 inverse laplace . . . 10
Exa 2.5 partial fractions . . . 10
Exa 2.7 inverse laplace transform . . . 11
Exa 2.8 inverse laplace transform . . . 11
Exa 2.9 inverse laplace transform . . . 11
Exa 2.10 determinant of matrix . . . 12
Exa 2.12 transpose of matrix . . . 12
Exa 2.13 adjoint of matrix . . . 12
Exa 2.14 equality of matrices . . . 13
Exa 2.15 addition of matrices . . . 13
Exa 2.16 conformability for multiplication of matrices . . . 14
Exa 2.17 multiplication of matrices . . . 14
Exa 2.18 inverse of 2x2 matrix . . . 15
Exa 2.19 inverse of 3x3 matrix . . . 15
Exa 2.20 rank of a matrix . . . 16
Exa 2.21 z transform . . . 16
Exa 2.22 z transform . . . 17
Exa 2.23 z transform . . . 17
Exa 2.25 final value thereom . . . 17
Exa 3.1 closed loop transfer function matrix . . . 18
Exa 3.3 masons gain formula applied to SFG in figure 3 15 . . . 18
Exa 3.4 masons gain formula . . . 19
Exa 3.5 masons gain formula . . . 20
Exa 3.6 masons gain formula . . . 21
Exa 3.7 masons gain formula . . . 21

Exa 3.9 masons gain formula . . . 22
Exa 3.10 masons gain formula . . . 23
Exa 3.11 masons gain formula . . . 24
Exa 4.1 transfer fnuction of system . . . 26
Exa 4.2 transfer fnuction of electric network . . . 27
Exa 4.3 gear trains . . . 27
Exa 4.4 mass spring system . . . 28
Exa 4.5 mass spring system . . . 28
Exa 4.9 incremental encoder . . . 29
Exa 5.1 state transition equation . . . 31
Exa 5.7 characteristic equation from transfer function . . . 32
Exa 5.8 characteristic equation from state equation . . . 32
Exa 5.9 eigen values . . . 32
Exa 5.12 ccf form . . . 33
Exa 5.13 ocf form . . . 33
Exa 5.14 dcf form . . . 34
Exa 5.18 system with identical eigen values . . . 35
Exa 5.19 controllability . . . 35
Exa 5.20 controllability . . . 35
Exa 5.21 observability . . . 36
Exa 6.1 stability of open loop systems . . . 37
Exa 6.2 rouths tabulation to determine stability . . . 38
Exa 6.3 rouths tabulation to determine stability . . . 38
Exa 6.4 first element in any row of rouths tabulation is z . . . 39
Exa 6.5 elements in any row of rouths tabulations are all . . . 39
Exa 6.6 determining critical value of K . . . 40
Exa 6.7 determining critical value of K . . . 41
Exa 6.8 stability of closed loop systems . . . 42
Exa 6.9 bilinear transformation method . . . 42
Exa 6.10 bilinear transformation method . . . 43
Exa 7.1 type of system . . . 44
Exa 7.2 steady state errors from open loop tf . . . 44
Exa 7.3 steady state errors from closed loop tf . . . 46
Exa 7.4 steady state errors from closed loop tf . . . 47
Exa 7.5 steady state errors from closed loop tf . . . 49
Exa 7.6 steady state errors from closed loop tf . . . 51
Exa 8.1 poles and zeros . . . 53
Exa 8.2 root locus . . . 53

Exa 8.3 root locus . . . 56
Exa 8.4 root locus . . . 57
Exa 8.5 root locus . . . 58
Exa 8.8 angle of departure and angle of arrivals . . . 59
Exa 8.9 multiple order pole . . . 59
Exa 8.10 intersection of root loci with real axis . . . 61
Exa 8.11 breakaway points . . . 63
Exa 8.12 breakaway points . . . 63
Exa 8.13 breakaway points . . . 65
Exa 8.14 breakaway points . . . 67
Exa 8.15 root sensitivity . . . 67
Exa 8.16 calculation of K on root loci . . . 68
Exa 8.17 properties of root loci . . . 71
Exa 8.18 effect of addition of poles to system . . . 71
Exa 8.19 effect of addition of zeroes to system . . . 72
Exa 8.20 effect of moving poles near jw axis . . . 76
Exa 8.21 effect of moving poles awat from jw axis . . . 76
Exa 9.1 nyquist plot . . . 81
Exa 9.2 nyquist plot . . . 81
Exa 9.3 stability of non minimum phase loop tf . . . 84
Exa 9.4 stability of minimum phase loop tf . . . 86
Exa 9.5 stability of non minimum phase loop tf . . . 86
Exa 9.6 stability of non minimum phase loop tf . . . 89
Exa 9.7 stability of non minimum phase loop tf . . . 89
Exa 9.8 effect of addition of poles . . . 91
Exa 9.9 effect of addition of zeroes . . . 91
Exa 9.10 multiple loop systems . . . 94
Exa 9.14 gain margin and phase margin . . . 97
Exa 9.15 bode plot . . . 100
Exa 9.17 relative stability . . . 100

List of Figures

8.1 poles and zeros 54
8.2 root locus 55
8.3 root locus 56
8.4 root locus 57
8.5 root locus 58
8.6 angle of departure and angle of arrivals 60
8.7 intersection of root loci with real axis 61
8.8 breakaway points 62
8.9 breakaway points 64
8.10 breakaway points 65
8.11 breakaway points 66
8.12 root sensitivity 68
8.13 root sensitivity 69
8.14 calculation of K on root loci 70
8.15 effect of addition of poles to system 72
8.16 effect of addition of poles to system 73
8.17 effect of addition of zeroes to system 74
8.18 effect of addition of zeroes to system 75
8.19 effect of moving poles near jw axis 77
8.20 effect of moving poles near jw axis 78
8.21 effect of moving poles awat from jw axis 79
8.22 effect of moving poles awat from jw axis 80

9.1 nyquist plot 82
9.2 nyquist plot 83
9.3 stability of non minimum phase loop tf 84
9.4 stability of minimum phase loop tf 85
9.5 stability of non minimum phase loop tf 87
9.6 stability of non minimum phase loop tf 88

9.7 stability of non minimum phase loop tf 90
9.8 effect of addition of poles 92
9.9 effect of addition of poles 93
9.10 effect of addition of zeroes 94
9.11 effect of addition of zeroes 95
9.12 multiple loop systems 96
9.13 gain margin and phase margin 98
9.14 bode plot 99
9.15 relative stability 101

Chapter 2

Mathematical Foundation

Scilab code Exa 2.1 laplace transform of step function

```
1 //laplace transform of unit function
2 syms t s
3 y=laplace('1',t,s)
4 disp(y,"F(s)=")
```

Scilab code Exa 2.2 laplace transform of exponential function

```
1 //laplace transform of exponential function
2  syms t s;
3  y=laplace('%e^(-1*t)',t,s);
4  disp(y,"ans=")
```

Scilab code Exa 2.3 final value thereom

```
1 //final value thereom
2 syms s
```

```
3 d=poly([0 2 1 1],'s','coeff')
4 n=poly([5],'s','coeff')
5 f=n/d;
6 disp(f,"F(s)=")
7 x=s*f;
8 y=limit(x,s,0); // final value theorem
9 disp(y,"f(inf)=")
```

Scilab code Exa 2.4 inverse laplace

```
1 //inverse laplace
2 syms s
3 F=1/(s^2+1)          //w=1
4 disp(F,"F(s)=")
5 f=ilaplace(F)
6 disp(f,"f(t)=")
7 printf("since s*F(s) has two poles on imaginary axis
    of s plane,final value thereom cannot be applied
    in this case")
```

Scilab code Exa 2.5 partial fractions

```
1 //partial fractions
2 n=poly([3 5],'s','coeff')
3 d=poly([6 11 6 1],'s','coeff')
4 f=n/d;
5 disp(f,"F(s)=")
6 pf=pfss(f)
7 disp(pf)
```

Scilab code Exa 2.7 inverse laplace transform

```
1 //inverse laplace transform
2 n=poly([4],'s','coeff')
3 d=poly([4 8 1],'s','coeff')     //w=2,damping ratio=2
4 G=n/d;
5 disp(G,"G(s)=")
6 pf=pfss(G)
7 disp(pf,"G(s)=")
8 syms s t
9 g1=ilaplace(pf(1),s,t)
10 g2=ilaplace(pf(2),s,t)
11 disp(g1+g2,"g(t)=")
```

Scilab code Exa 2.8 inverse laplace transform

```
1 //inverse laplace transform
2 n=poly([5 -1 -1],'s','coeff')
3 d=poly([0 -1 -2],'s','roots')
4 Y=n/d;
5 disp(Y,"Y(s)=")
6 pf=pfss(Y)
7 disp(pf,"Y(s)=")
8 syms s t
9 y1=ilaplace(pf(1),s,t)
10 y2=ilaplace(pf(2),s,t)
11 y3=ilaplace(pf(3),s,t)
12 disp(y1+y2+y3,"g(t)=")
13 l=limit(Y*s,s,0)
14 disp(l,"limit of y(t) as t tends to infinity=")
```

Scilab code Exa 2.9 inverse laplace transform

```
1  //inverse laplace transform
2  n=poly([1000],'s','coeff')
3  d=poly([0 1000 34.5 1],'s','coeff')
4  Y=n/d;
5  disp(Y,"Y(s)=")
6  pf=pfss(Y)
7  disp(pf,"Y(s)=")
8  syms s t
9  y1=ilaplace(pf(1),s,t)
10 y2=ilaplace(pf(2),s,t)
11 y3=ilaplace(pf(3),s,t)
12 disp(y1+y2+y3,"y(t)=")
```

Scilab code Exa 2.10 determinant of matrix

```
1  //determinant of the matrix
2  A=[1 2;3 4]
3  d=det(A)
4  disp(d)
```

Scilab code Exa 2.12 transpose of matrix

```
1  //transpose of a matrix
2  A=[3 2 1;0 -1 5]
3  t=A'
4  disp(t)
```

Scilab code Exa 2.13 adjoint of matrix

```
1  //adjoint of a matrix
```

```
A=[1 2;3 4]
i=inv(A)
a=i.*det(A)
disp(a)
```

Scilab code Exa 2.14 equality of matrices

```
//equality of matrices
A=[1 2;3 4]
B=[1 2;3 4]
x=1;
for i=1:2
  for j=1:2
    if A(i,j)~=B(i,j) then
     x=0
    end
  end
end
if x==1 then
  disp("matrices are equal")
else
  disp("matrices are not equal")
end
```

Scilab code Exa 2.15 addition of matrices

```
//addition of matrices
A=[3 2;-1 4;0 -1]
B=[0 3;-1 2;1 0]
s=A+B
disp(s)
```

Scilab code Exa 2.16 conformability for multiplication of matrices

```
1 //conformablility for multiplication of matrices
2 A=[1 2 3;4 5 6]
3 B=[1 2 3]
4 C=size(A)
5 D=size(B)
6 if C(1,2)==D(1,1) then
7     disp("matrices are conformable for
        multiplication AB")
8 else
9     disp("matrices are not conformable for
        multiplication AB")
10 end
11 if D(1,2)==C(1,1) then
12     disp("matrices are conformable for
        multiplication BA")
13 else
14     disp("matrices are not conformable for
        multiplication BA")
15 end
```

Scilab code Exa 2.17 multiplication of matrices

```
1 //multiplication of matrices
2 A=[3 -1;0 1;2 0]
3 B=[1 0 -1;2 1 0]
4 C=size(A)
5 D=size(B)
6 if C(1,2)==D(1,1) then
7     AB=A*B
8     disp(AB,"AB=")
```

```
9  else
10     disp("matrices  are not conformable for
          multiplication AB")
11 end
12 if D(1,2)==C(1,1) then
13     BA=B*A
14     disp(BA,"BA=")
15 else
16     disp("matrices  are not conformable for
          multiplication BA")
17 end
```

Scilab code Exa 2.18 inverse of 2x2 matrix

```
1  //inverse of 2 X 2  matrix
2  A=[1 2;3 4]
3  d=det(A)
4  if det(A)~=0 then
5      i=inv(A)
6      disp(i,"A^-1=")
7  else
8      disp("inverse of a singular matrix doesnt exist"
          )
9  end
```

Scilab code Exa 2.19 inverse of 3x3 matrix

```
1  //inverse of a 3 X 3 matrix
2  A=[1 2 3;4 5 6;7 8 9]
3  d=det(A)
4  if det(A)~=0 then
5      i=inv(A)
6      disp(i,"A^-1=")
```

```
7 else
8     disp("inverse of a singular matrix doesnt exist"
        )
9 end
```

Scilab code Exa 2.20 rank of a matrix

```
1 //rank of a matrix
2 A=[0 1;0 1]
3 [E,Q,Z ,stair ,rk1]=ereduc(A,1.d-15)
4 disp(rk1,"rank of A=")
5 B=[0 5 1 4;3 0 3 2]
6 [E,Q,Z ,stair ,rk2]=ereduc(B,1.d-15)
7 disp(rk2,"rank of B=")
8 C=[3 9 2;1 3 0;2 6 1]
9 [E,Q,Z ,stair ,rk3]=ereduc(C,1.d-15)
10 disp(rk3,"rank of C=")
11 D=[3 0 0;1 2 0;0 0 1]
12 [E,Q,Z ,stair ,rk4]=ereduc(D,1.d-15)
13 disp(rk4,"rank of D=")
```

Scilab code Exa 2.21 z transform

```
1 //z transform
2 syms n z;
3 a=1;
4 x =%e^-(a*n);
5 X = symsum(x*(z^(-n)),n,0,%inf)
6 disp(X,"ans=")
```

Scilab code Exa 2.22 z transform

```
1 //z transform
2 syms n z;
3 x =1;
4 X = symsum(x*(z^(-n)),n,0,%inf)
5 disp(X,"ans=")
```

Scilab code Exa 2.23 z transform

```
1 //z transform
2 //t=k*T
3 syms k z;
4 a=1;
5 T=1;
6 x =%e^-(a*k*T);
7 X = symsum(x*(z^(-k)),k,0,%inf)
8 disp(X,"ans=")
```

Scilab code Exa 2.25 final value thereom

```
1 //final  value thereom
2 z=%z
3 sys=syslin('c',0.792*z^2/((z-1)*(z^2-0.416*z+0.208))
    )
4 syms z
5 l=limit(sys*(1-z^-1),z,1)
6 disp(l,"limit as k approaches infinity=")
```

Chapter 3

Transfer Functions Block Diagrams and Signal Flow Graphs

Scilab code Exa 3.1 closed loop transfer function matrix

```
1 //closed loop transfer function matrix
2 s=%s
3 G=[1/(s+1) -1/s;2 1/(s+2)]
4 H=[1 0;0 1]
5 GH=G*H
6 disp(GH,"G(s)H(s)=")
7 I=[1 0;0 1]
8 x=I+GH
9 y=inv(x)
10 M=y*G
11 disp(M,"M(s)=")
```

Scilab code Exa 3.3 masons gain formula applied to SFG in figure 3 15

```
1 //mason's gain formula applied to SFG in figure 3-15
2 syms G H
3 M1=G          //as seen from SFG there is only one
     forward path
4 L11=-G*H      //only one loop and no non touching
     loops
5 delta=1-L11
6 delta1=1
7 Y=M1*delta1/delta
8 disp(Y,"Y(s)/R(s)=")
```

Scilab code Exa 3.4 masons gain formula

```
1 //masons gain formula applied to SFG in figure 3-8(d
     )
2 //two forward paths
3 syms a12 a23 a24 a25 a32 a34 a43 a44 a45
4 M1=a12*a23*a34*a45
5 M2=a12*a25
6 //four loops
7 L11=a23*a32
8 L21=a34*a43
9 L31=a24*a32*a43
10 L41=a44
11 //one pair of non touching loops
12 L12=a23*a32*a44
13 delta=1-(L11+L21+L31+L41)+(L12)
14 delta1=1
15 delta2=1-(L21+L41)
16 x=(M1*delta1+M2*delta2)/delta
17 disp(x,"y5/y1=")
18 //if y2 is output node
19 M1=a12
20 delta1=1-(L21+L41)
21 y=(M1*delta1)/delta
```

```
22 disp(y,"y2/y1=")
```

Scilab code Exa 3.5 masons gain formula

```
1  //mason's gain formula applied to SFG in figure 3-16
2  //y2 as output node
3  syms G1 G2 G3 G4 G5 H1 H2 H3 H4
4  M1=1
5  L11=-G1*H1
6  L21=-G3*H2
7  L31=G1*G2*G3*-H3
8  L41=-H4
9  L12=G1*H1*G3*H2
10 L22=G1*H1*H4
11 L32=G3*H2*H4
12 L42=-G1*G2*G3*H3*H4
13 L13=-G1*H1*G3*H2*H4
14 delta=1-(L11+L21+L31+L41)+(L12+L22+L32+L42)+L13
15 delta1=1-(L21+L41)+(L32)
16 x=M1*delta1/delta
17 disp(x,"y2/y1=")
18 //y4 as output node
19 M1=G1*G2
20 delta1=1-(L41)
21 y=M1*delta1/delta
22 disp(y,"y4/y1=")
23 //y6 or y7 as output node
24 M1=G1*G2*G3*G4
25 M2=G1*G5
26 delta1=1
27 delta2=1-(L21)
28 z=(M1*delta1+M2*delta2)/delta
29 disp(z,"y6/y1=y7/y1=")
```

Scilab code Exa 3.6 masons gain formula

```
1  //mason's gain formula applied to SFG in figure 3-16
2  //y2 as output node
3  syms G1 G2 G3 G4 G5 H1 H2 H3 H4
4  M1=1
5  L11=-G1*H1
6  L21=-G3*H2
7  L31=G1*G2*G3*-H3
8  L41=-H4
9  L12=G1*H1*G3*H2
10 L22=G1*H1*H4
11 L32=G3*H2*H4
12 L42=-G1*G2*G3*H3*H4
13 L13=-G1*H1*G3*H2*H4
14 delta=1-(L11+L21+L31+L41)+(L12+L22+L32+L42)+L13
15 delta1=1-(L21+L41)+(L32)
16 x=M1*delta1/delta
17 disp(x,"y2/y1=")
18 //y7 as output node
19 M1=G1*G2*G3*G4
20 M2=G1*G5
21 delta1=1
22 delta2=1-(L21)
23 y=(M1*delta1+M2*delta2)/delta
24 disp(y,"y7/y1=")
25 z=y/x                    // (y7/y2)=(y7/y1)/(y2/y1)
26 disp(z,"y7/y2=")
```

Scilab code Exa 3.7 masons gain formula

```
1  //block diagram is converted to SFG
```

```
2 //mason's gain formula applied to SFG in figure 3-17
3 //E as output node
4 syms G1 G2 G3 G4 H1 H2
5 M1=1
6 L11=-G1*G2*H1
7 L21=-G2*G3*H2
8 L31=-G1*G2*G3
9 L41=-G1*G4
10 L51=-G4*H2
11 delta=1-(L11+L21+L31+L41+L51)
12 delta1=1-(L21+L51+L11)
13 x=M1*delta1/delta
14 disp(x,"E(s)/R(s)=")
15 //Y as output node
16 M1=G1*G2*G3
17 M2=G1*G4
18 delta1=1
19 delta2=1
20 y=(M1*delta1+M2*delta2)/delta
21 disp(y,"Y(s)/R(s)=")
```

Scilab code Exa 3.9 masons gain formula

```
1 //finding transfer function from state diagram by
    applying gain formula
2 //state diagram is shown in fifure 3-21
3 syms s
4 //initial conditions are sset to zero
5 M1=s^-1*s^-1
6 L11=-3*s^-1
7 L21=-2*s^-1*s^-1
8 delta=1-(L11+L21)
9 delta1=1
10 x=M1*delta1/delta
11 disp(x,"Y(s)/R(s)=")
```

Scilab code Exa 3.10 masons gain formula

```
1  //applying gain formula to state diagram 3-22
2  //r(t),x1(t) and x2(t) are input nodes
3  //y(t) is output node
4  //superposition principle holds good
5
6  syms s r x1 x2
7  //r(t) as input node and y(t) as output node
8  M1=0
9  delta1=1
10 delta=1
11 a=(M1*delta1)/delta
12 y1=a*r
13 disp(y1,"y1(t)=")
14
15 //x1(t) as input node and y(t) as output node
16 M1=1
17 delta1=1
18 b=(M1*delta1)/delta
19 y2=b*x1
20 disp(y2,"y2(t)=")
21
22 //x2(t) as input node and y(t) as output node
23 M1=0
24 delta1=1
25 c=(M1*delta1)/delta
26 y3=c*x2
27 disp(y3,"y3(t)=")
28
29 disp(y1+y2+y3,"y(t)=")
```

Scilab code Exa 3.11 masons gain formula

```
1 //applying gain formula to state diagram in figure
    3-23(b)
2 //r(t),x1(t),x2(t) and x3(t) are input nodes
3 //y(t) is output node
4 //superposition principle holds good
5
6 syms s a0 a1 a2 a3 r x1 x2 x3
7 //r(t) as input node and y(t) as output node
8 M1=0
9 delta1=1
10 L11=-a0*a3
11 delta=1-(L11)
12 a=(M1*delta1)/delta
13 y1=a*r
14 disp(y1,"y1(t)=")
15
16 //x1(t) as input node and y(t) as output node
17 M1=1
18 delta1=1
19 b=(M1*delta1)/delta
20 y2=b*x1
21 disp(y2,"y2(t)=")
22
23 //x2(t) as input node and y(t) as output node
24 M1=0
25 delta1=1
26 c=(M1*delta1)/delta
27 y3=c*x2
28 disp(y3,"y3(t)=")
29
30 //x3(t) as input node and y(t) as output node
31 M1=a0
32 delta1=1
33 d=(M1*delta1)/delta
34 y4=d*x3
35 disp(y4,"y4(t)=")
```

```
36
37 disp(y1+y2+y3+y4,"y(t)=")
```

Chapter 4

Mathematical Modelling of Physical Systems

Scilab code Exa 4.1 transfer fnuction of system

```
1  //transfer function of the system
2  //from state diagram in 4-1(b)
3  //initial conditions are taken as zero
4  //considering voltage across capacitor as output
5  syms R L C
6  s=%s
7  M1=(1/L)*(s^-1)*(1/C)*(s^-1)
8  L11=-(s^-1)*(R/L)
9  delta=1-(L11)
10 delta1=1
11 x=M1*delta1/delta
12 disp(x,"Ec(s)/E(s)=")
13 //considering current in the circuit as output
14 M1=(1/L)*(s^-1)
15 delta1=1
16 y=M1*delta1/delta
17 disp(y,"I(s)/E(s)=")
```

Scilab code Exa 4.2 transfer fnuction of electric network

```
1  //transfer function of electric network
2  //from state diagram in 4-2(b)
3  //inital conditions are taken as zero
4  //considering i1 as output
5  syms R1 R2 L1 L2 C
6  s=%s
7  M1=(1/L1)*(s^-1)
8  L11=-(s^-1)*(R1/L1)
9  L21=-(s^-1)*(1/C)*(s^-1)*(1/L1)
10 L31=-(s^-1)*(1/L2)*(s^-1)*(1/C)
11 L41=-(s^-1)*(R2/L2)
12 L12=L11*L31
13 L22=L11*L41
14 L32=L21*L41
15 delta=1-(L11+L21+L31+L41)+(L12+L22+L32)
16 delta1=1-(L31+L41)
17 x=M1*delta1/delta
18 disp(x,"I1(s)/E(s)=")
19 //considering i2 as output
20 M1=(1/L1)*(s^-1)*(1/C)*(s^-1)*(1/L2)*(s^-1)
21 delta1=1
22 y=M1*delta1/delta
23 disp(y,"I2(s)/E(s)=")
24 //considering voltage across capacitor as output
25 M1=(1/L1)*(s^-1)*(1/C)*(s^-1)
26 delta1=1-L41
27 z=M1*delta1/delta
28 disp(z,"Ec(s)/E(s)=")
```

Scilab code Exa 4.3 gear trains

```
1  //gear trains
2  printf("Given \n inertia(J2)=0.05oz-in.-sec^2 \n
       frictional torque(T2)=2oz-in. \n N1/N2(r)=1/5")
3  J2=0.05;
4  disp(J2,"J2=")
5  T2=2;
6  disp(T2,"T2=")
7  r=1/5
8  disp(r,"N1/N2=")
9  printf("J1=(N1/N2)^2*J2 \n T1=(N1/N2)*T2")
10 J1=(r)^2*J2;
11 disp(J1,"The reflected inertia on side of N1=")
12 T1=(r)*T2
13 disp(T1,"The reflected coulumb friction is=")
```

Scilab code Exa 4.4 mass spring system

```
1  //mass-spring system
2  //free body diagram and state diagram are drawn as
       shown in figure 4-18(b) and 4-18(c)
3  //applying gain formula to state diagram
4  syms K M B
5  s=%s
6  M1=(1/M)*(s^-2)
7  L11=-(B/M)*(s^-1)
8  L21=-(K/M)*(s^-2)
9  delta=1-(L11+L21)
10 delta1=1
11 x=M1*delta1/delta
12 disp(x,"Y(s)/F(s)=")
```

Scilab code Exa 4.5 mass spring system

```
1  //mass-spring system
2  //free body diagram and state diagram are drawn as
       shown in figure 4-19(b) and 4-19(c)
3  //applying gain formula to state diagram
4  syms K M B
5  s=%s
6  //considering y1 as output
7  M1=(1/M)
8  L11=-(B/M)*(s^-1)
9  L21=-(K/M)*(s^-2)
10 L31=(K/M)*(s^-2)
11 delta=1-(L11+L21+L31)
12 delta1=1-(L11+L21)
13 x=M1*delta1/delta
14 disp(x,"Y1(s)/F(s)=")
15 //considering y2 as output
16 M1=(1/K)*(K/M)*(s^-2)
17 delta1=1
18 y=M1*delta1/delta
19 disp(y,"Y2(s)/F(s)=")
```

Scilab code Exa 4.9 incremental encoder

```
1  //incremental encoder
2  //2 sinusoidal signals
3  //generates four zero crossings per cycle(zc)
4  //printwheel has 96 characters on its pheriphery(ch)
       and encoder has 480 cycles(cyc)
5  zc=4
6  ch=96
7  cyc=480
8  zcpr=cyc*zc  //zero crossings per revolution
9  disp(zcpr,"zero_crossings_per_revolution=")
10 zcpc=zcpr/ch //zreo crossings per character
11 disp(zcpc,"zero_crossings_per_character=")
```

```
12 //500khz clock is used
13 //500 pulses/zero crossing
14 shaft_speed=500000/500
15 x=shaft_speed/zcpr
16 disp(x,"ans=")   //in rev per sec
```

Chapter 5

State Variable Analysis

Scilab code Exa 5.1 state transition equation

```
1 //state transition equation
2 //as seen from state equation A=[0 1;-2 -3] B=[0;1]
    E=0;
3 A=[0 1;-2 -3]
4 B=[0;1]
5 s=poly(0,'s');
6 [Row Col]=size(A)     //Size of a matrix
7 m=s*eye(Row,Col)-A    //sI-A
8 n=det(m)              //To Find The Determinant of si
    -A
9 p=inv(m) ;            // To Find The Inverse Of sI-A
10 U=1/s
11 p=p*U
12 syms t s;
13 disp(p,"phi(s)=")    //Resolvent Matrix
14 for i=1:Row
15 for j=1:Col
16 //Taking Inverse Laplace of each element of Matrix
    phi(s)
17 q(i,j)=ilaplace(p(i,j),s,t);
18 end;
```

```
19 end;
20 disp(q,"phi(t)=")//State Transition Matrix
21 y=q*B; //x(t)=phi(t)*x(0)
22 disp(y,"Solution To The given eq.=")
```

Scilab code Exa 5.7 characteristic equation from transfer function

```
1 //characteristic equation from transfer function
2 s=%s
3 sys=syslin('c',1/(s^3+5*s^2+s+2))
4 c=denom(sys)
5 disp(c,"characteristic equation=")
```

Scilab code Exa 5.8 characteristic equation from state equation

```
1 //characteristic equation from state equation
2 A=[0 1 0;0 0 1;-2 -1 -5]
3 B=[0;0;1]
4 C=[1 0 0]
5 D=[0]
6 [Row Col]=size(A)
7 Gr=C*inv(s*eye(Row,Col)-A)*B+D
8 c=denom(Gr)
9 disp(c,"characteristic equation=")
```

Scilab code Exa 5.9 eigen values

```
1 //eigen values
2 A=[0 1 0;0 0 1;-2 -1 -5]
3 e=spec(A)     //spec gives eigen values of matrix
4 disp(e,"eigen values=")
```

Scilab code Exa 5.12 ccf form

```
1  //OCF form
2  s=%s
3  A=[1 2 1;0 1 3;1 1 1]
4  B=[1;0;1]
5  [row,col]=size(A)
6  c=s*eye(row,col)-A
7  x=det(c)
8  r=coeff(x)
9  M=[r(1,2) r(1,3) 1;r(1,3) 1 0;1 0 0]
10 S=[B A*B A^2*B]
11 disp(S,"controllability matrix=")
12 if (det(S)==0) then
13     printf("system cannot be transformed into ccf
       form")
14 else
15     printf("system can be transformed into ccf form"
       )
16 end
17 P=S*M
18 disp(P,"P=")
19 Accf=inv(P)*A*P
20 Bccf=inv(P)*B
21 disp(Accf,"Accf=")
22 disp(Bccf,"Bccf=")
```

Scilab code Exa 5.13 ocf form

```
1  //OCF form
2  A=[1 2 1;0 1 3;1 1 1]
```

```
B=[1;0;1]
C=[1 1 0]
D=0
[row,col]=size(A)
c=s*eye(row,col)-A
x=det(c)
r=coeff(x)
M=[r(1,2) r(1,3) 1;r(1,3) 1 0;1 0 0]
V=[C;C*A;C*A^2]
disp(V,"observability matrix=")
if (det(V)==0) then
    printf("system cannot be transformed into ocf
       form")
else
    printf("system can be transformed into ocf form"
       )
end
Q=inv(M*V)
disp(Q,"Q=")
Aocf=inv(Q)*A*Q
Cocf=C*Q
B=inv(Q)*B
disp(Aocf,"Aocf=")
disp(Cocf,"Cocf=")
```

Scilab code Exa 5.14 dcf form

```
//DCF form
A=[0 1 0;0 0 1;-6 -11 -6]
x=spec(A)
T=[1 1 1;x(1,1) x(2,1) x(3,1);(x(1,1))^2 (x(2,1))^2
   (x(3,1))^2]
Adcf=inv(T)*A*T
disp(Adcf,"Adcf=")
```

Scilab code Exa 5.18 system with identical eigen values

```
1 //system with identical eigen values
2 A=[1 0;0 1]    //lamda1=1
3 B=[2;3]        //b11=2 b21=3
4 S=[B A*B]
5 if det(S)==0 then
6     printf("system is uncontrollable")
7 else
8     printf("system is contollable")
9     end
```

Scilab code Exa 5.19 controllability

```
1 //controllability
2 A=[-2 1;0 -1]
3 B=[1;0]
4 S=[B A*B]
5 if det(S)==0 then
6     printf("system is uncontrollable")
7 else
8     printf("system is contollable")
9     end
```

Scilab code Exa 5.20 controllability

```
1 //controllability
2 A=[1 2 -1;0 1 0;1 -4 3]
3 B=[0;0;1]
```

```
S=[B A*B A^2*B]
if det(S)==0 then
    printf("system is uncontrollable")
else
    printf("system is contollable")
    end
```

Scilab code Exa 5.21 observability

```
//observability
A=[-2 0;0 -1]
B=[3;1]
C=[1 0]
V=[C;C*A]
if det(V)==0 then
    printf("system is unobservable")
else
    printf("system is observable")
    end
```

Chapter 6

Stability of Linear Control Systems

Scilab code Exa 6.1 stability of open loop systems

```
1  //stability of open loop systems
2  s=%s
3  sys1=syslin('c',20/((s+1)*(s+2)*(s+3)))
4  disp(sys1,"M(s)=")
5  printf("sys1 is stable as there are no ploes or
       zeroes in RHP")
6  sys2=syslin('c',20*(s+1)/((s-1)*(s^2+2*s+2)))
7  disp(sys2,"M(s)=")
8  printf("sys2 is unstable due to pole at s=1")
9  sys3=syslin('c',20*(s-1)/((s+2)*(s^2+4)))
10 disp(sys3,"M(s)=")
11 printf("sys3 is marginally stable or marginally
       unstable due to s=j2 and s=-j2")
12 sys4=syslin('c',10/((s+10)*(s^2+4)^2))
13 disp(sys4,"M(s)=")
14 printf("sys4 is unstable due to multiple order pole
       at s=j2 and s=-j2")
15 sys5=syslin('c',10/(s^4+30*s^3+s^2+10*s))
16 disp(sys5,"M(s)=")
```

```
17 printf("sys5 is stable if pole at s=0 is placed
     intentionally")
```

Scilab code Exa 6.2 rouths tabulation to determine stability

```
1  //rouths tabulations to determine stability
2  s=%s;
3  m=s^3-4*s^2+s+6;
4  disp(m)
5  r=coeff(m)
6  n=length(r)
7  routh=routh_t(m) //This Function generates the Routh
       table
8  disp(routh,"rouths tabulation=")
9  c=0;
10 for i=1:n
11 if (routh(i,1)<0)
12 c=c+1;
13  end
14   end
15   if(c>=1)
16     printf("system is unstable")
17   else printf("system is stable")
18   end
```

Scilab code Exa 6.3 rouths tabulation to determine stability

```
1  //rouths tabulations to determine stability
2  s=%s;
3  m=2*s^4+s^3+3*s^2+5*s+10;
4  disp(m)
5  r=coeff(m)
6  n=length(r)
```

```
7  routh=routh_t(m) //This Function generates the Routh
       table
8  disp(routh,"rouths tabulation=")
9  c=0;
10 for i=1:n
11 if (routh(i,1)<0)
12 c=c+1;
13  end
14   end
15   if(c>=1)
16     printf("system is unstable")
17   else printf("system is stable")
18   end
```

Scilab code Exa 6.4 first element in any row of rouths tabulation is z

```
1 //first element in any row of rouths tabulation is
     zero
2 s=%s
3 m=s^4+s^3+2*s^2+2*s+3
4 r=coeff(m);  //Extracts the coefficient of the
     polynomial
5 n=length(r);
6 routh=routh_t(m)
7 disp(routh,"routh=")
8 printf("since there are two sign changes in the
     rouths tabulation,sys is unstable")
```

Scilab code Exa 6.5 elements in any row of rouths tabulations are all

```
1 //elements in one row of rouths tabulations are all
     zero
2 s=%s;
```

```
m=s^5+4*s^4+8*s^3+8*s^2+7*s+4;
disp(m)
r=coeff(m)
n=length(r)
routh=routh_t(m)
disp(routh,"rouths tabulations=")
c=0;
for i=1:n
if (routh(i,1)<0)
c=c+1;
end
end
if(c>=1)
printf("system is unstable")
else printf("system is marginally stable")
end
```

Scilab code Exa 6.6 determining critical value of K

```
//determining critical value of K
s=%s
syms K
m=s^3+3408.3*s^2+1204000*s+1.5*10^7*K
cof_a_0 = coeffs(m,'s',0);
cof_a_1 = coeffs(m,'s',1);
cof_a_2 = coeffs(m,'s',2);
cof_a_3 = coeffs(m,'s',3);

r=[cof_a_0 cof_a_1 cof_a_2 cof_a_3]

n=length(r);
routh=[r([4,2]);r([3,1])];
routh=[routh;-det(routh)/routh(2,1),0];
t=routh(2:3,1:2); //extracting the square sub block
   of routh matrix
```

```
16 routh=[routh;-det(t)/t(2,1),0]
17 disp(routh,"rouths tabulation=")
18 routh(3,1)=0  //For marginaly stable system
19 sys=syslin('c',1.5*10^7/(s^3+3408.3*s^2+1204000*s))
20 k=kpure(sys)
21 disp(k,"K(marginal)=")
22 disp('=0',routh(2,1)*(s^2)+1.5*10^7*k,"auxillary
      equation")
23 p=poly([1.5*10^7*k,0,3408.3],'s','coeff')
24 s=roots(p)
25 disp(s,"Frequency of oscillation(in rad/sec)=")
```

Scilab code Exa 6.7 determining critical value of K

```
1  //determining critical value of K
2  s=%s
3  syms K
4  m=s^3+3*K*s^2+(K+2)*s+4
5  cof_a_0 = coeffs(m,'s',0);
6  cof_a_1 = coeffs(m,'s',1);
7  cof_a_2 = coeffs(m,'s',2);
8  cof_a_3 = coeffs(m,'s',3);
9
10 r=[cof_a_0 cof_a_1 cof_a_2 cof_a_3]
11
12 n=length(r);
13 routh=[r([4,2]);r([3,1])];
14 routh=[routh;-det(routh)/routh(2,1),0];
15 t=routh(2:3,1:2); //extracting the square sub block
      of routh matrix
16 routh=[routh;-det(t)/t(2,1),0]
17 disp(routh,"rouths tabulation=")
18 routh(3,1)=0  //For marginaly stable system
19 sys=syslin('c',s*(3*s+1)/(s^3+2*s+4))
20 k=kpure(sys)
```

```
21 disp(k,"K(marginal)=")
```

Scilab code Exa 6.8 stability of closed loop systems

```
1  //stability of closed loop systems
2  z=%z
3  sys1=syslin('c',5*z/((z-0.2)*(z-0.8)))
4  disp(sys1,"M(z)=")
5  printf("sys1 is stable")
6  sys2=syslin('c',5*z/((z+1.2)*(z-0.8)))
7  disp(sys2,"M(z)=")
8  printf("sys2 is unstable due to pole at z=-1.2")
9  sys3=syslin('c',5*(z+1)/(z*(z-1)*(z-0.8)))
10 disp(sys3,"M(z)=")
11 printf("sys3 is marginally stable due to z=1")
12 sys4=syslin('c',5*(z+1.2)/(z^2*(z+1)^2*(z+0.1)))
13 disp(sys4,"M(z)=")
14 printf("sys4 is unstable due to multiple order pole
     at z=-1")
```

Scilab code Exa 6.9 bilinear transformation method

```
1  //bilinear transformation method
2  r=%s
3  //p=z^3+5.94*z^2+7.7*z-0.368
4  //substituting z=(1+r)/(1-r) we get
5  m=3.128*r^3-11.47*r^2+2.344*r+14.27
6  x=coeff(m)
7  n=length(x)
8  routh=routh_t(m)
9  disp(routh,"rouths tabulations")
10 c=0;
11 for i=1:n
```

```
12 if (routh(i,1)<0) then
13 c=c+1
14 end
15 end
16 if (c>=1) then
17 printf("system is unstable")
18 else printf("system is stable")
19 end
```

Scilab code Exa 6.10 bilinear transformation method

```
1 //bilinear transformation method
2 s=%s
3 syms K
4 //p=z^3+z^2+z+K
5 //substituting z=(1+r)/(1-r) we get
6 m=(1-K)*s^3+(1+3*K)*s^2+3*(1-K)*s+3+K
7 cof_a_0 = coeffs(m,'s',0);
8 cof_a_1 = coeffs(m,'s',1);
9 cof_a_2 = coeffs(m,'s',2);
10 cof_a_3 = coeffs(m,'s',3);
11
12 r=[cof_a_0 cof_a_1 cof_a_2 cof_a_3]
13
14 n=length(r);
15 routh=[r([4,2]);r([3,1])];
16 routh=[routh;-det(routh)/routh(2,1),0];
17 t=routh(2:3,1:2); //extracting the square sub block
     of routh matrix
18 routh=[routh;-det(t)/t(2,1),0]
19 disp(routh,"rouths tabulation=")
```

Chapter 7

Time Domain Analysis of Control Systems

Scilab code Exa 7.1 type of system

```
1 //type of system
2 s=%s
3 G1=syslin('c',(1+0.5*s)/(s*(1+s)*(1+2*s)*(1+s+s^2)))
4 disp(G1,"G(s)=")
5 printf("type 1 as it has one s term in denominator")
6 G2=syslin('c',(1+2*s)/s^3)
7 disp(G2,"G(s)=")
8 printf("type 3 as it has 3 poles at origin")
```

Scilab code Exa 7.2 steady state errors from open loop tf

```
1 //steady state errors from open loop transfer
     function
2 s=%s;
3 //type 1 system
4 G=syslin('c',(s+3.15)/(s*(s+1.5)*(s+0.5)))//K=1
```

```
5  disp(G,"G(s)=")
6  H=1;
7  y=G*H;
8  disp(y,"G(s)H(s)=")
9  syms s;
10 Kv=limit(s*y,s,0); //Kv= velocity error coefficient
11 Ess=1/Kv
12 //Refering the table 7.1 given in the book ,For type
       1 system  Kp=%inf,Ess=0 & Ka=0,Ess=%inf
13 printf("For type1 system \n step input Kp=inf Ess=0
      \n \n parabolic input Ka=0 Ess=inf \n ")
14 disp(Kv,"ramp input Kv=")
15 disp(Ess,"Ess=")
16 //type 2 system
17 p=poly([1],'s','coeff');
18 q=poly([0 0 12 1],'s','coeff');
19 G=p/q;//K=1
20 disp(G,"G(s)=")
21 H=1;
22 y=G*H;
23 disp(y,"G(s)H(s)=")
24 Ka=limit(s^2*y,s,0); //Ka= parabolic error
      coefficient
25 Ess=1/Ka
26 //Refering the table 7.1 given in the book ,For type
       2 system  Kp=%inf,Ess=0 & Kv=inf,Ess=0
27 printf("For type2 system \n step input Kp=inf Ess=0
      \n ramp input Kv=inf Ess=0 \n ")
28 disp(Ka,"parabolic input Ka=")
29 disp(Ess,"Ess=")
30 //type 2 system
31 p=poly([5 5],'s','coeff');
32 q=poly([0 0 60 17 1],'s','coeff');
33 G=p/q;//K=1
34 disp(G,"G(s)=")
35 H=1;
36 y=G*H;
37 disp(y,"G(s)H(s)=")
```

```
38 Ka=limit(s^2*y,s,0); //Ka= parabolic error
      coefficient
39 Ess=1/Ka
40 //Refering the table 7.1 given in the book ,For type
      2 system  Kp=%inf,Ess=0 & Kv=inf,Ess=0
41 printf("For type2 system \n step input Kp=inf Ess=0
      \n ramp input Kv=inf Ess=0 \n ")
42 disp(Ka,"parabolic input Ka=")
43 disp(Ess,"Ess=")
```

Scilab code Exa 7.3 steady state errors from closed loop tf

```
1 //steady state errors from closed loop transfer
     functions
2 s=%s
3 p=poly([3.15 1 0],'s','coeff');//K=1
4 q=poly([3.15 1.75 2 1],'s','coeff');
5 M=p/q
6 disp(M,"M(s)=")
7 H=1;
8 R=1;
9 b=coeff(p)
10 a=coeff(q)
11
12 //step input
13 if (a(1,1)==b(1,1)) then
14     printf("for unit step input Ess=0" )
15 else
16     Ess=1/H*(1-(b(1,1)*H/a(1,1)))*R
17     disp(Ess,"for unit step input Ess=")
18 end
19
20 //ramp input
21 c=0
22 for i=1:2
```

```
23      if(a(1,i)-b(1,i)*H==0) then
24          c=c+1
25      end
26  end
27  if(c==2)
28          printf("for unit ramp input Ess=0")
29      else if(c==1) then
30              Ess=(a(1,2)-b(1,2)*H)/a(1,1)*H
31              disp(Ess,"for unit ramp input Ess=")
32          else printf("for unit ramp input Ess=inf")
33       end
34   end
35
36  //parabolic input
37  c=0
38  for i=1:3
39      if(a(1,i)-b(1,i)*H==0) then
40          c=c+1
41      end
42  end
43  if(c==3)
44          printf("for unit parabolic input Ess=0")
45      else if(c==2) then
46              Ess=(a(1,3)-b(1,3)*H)/a(1,1)*H
47              disp(Ess,"for unit parabolic input Ess="
                    )
48          else printf("for unit parabolic input Ess=
                inf")
49       end
50   end
```

Scilab code Exa 7.4 steady state errors from closed loop tf

```
1  //steady state errors from closed loop transfer
     functions
```

```
2  s=%s
3  p=poly([5 5 0],'s','coeff');
4  q=poly([5 5 60 17 1],'s','coeff');
5  M=p/q
6  disp(M,"M(s)=")
7  H=1;
8  R=1;
9  b=coeff(p)
10 a=coeff(q)
11
12 //step input
13 if (a(1,1)==b(1,1)) then
14     printf("for unit step input Ess=0 \n" )
15 else
16     Ess=1/H*(1-(b(1,1)*H/a(1,1)))*R
17     disp(Ess,"for unit step input Ess=")
18 end
19
20 //ramp input
21 c=0
22 for i=1:2
23     if(a(1,i)-b(1,i)*H==0) then
24         c=c+1
25     end
26 end
27 if(c==2)
28         printf("for unit ramp input Ess=0 \n")
29     else if(c==1) then
30             Ess=(a(1,2)-b(1,2)*H)/a(1,1)*H
31             disp(Ess,"for unit ramp input Ess=")
32         else printf("for unit ramp input Ess=inf \n"
               )
33      end
34  end
35
36 //parabolic input
37 c=0
38 for i=1:3
```

```
39      if(a(1,i)-b(1,i)*H==0) then
40          c=c+1
41      end
42  end
43  if(c==3)
44          printf("for unit parabolic input Ess=0 \n")
45      else if(c==2) then
46              Ess=(a(1,3)-b(1,3)*H)/a(1,1)*H
47              disp(Ess,"for unit parabolic input Ess="
                    )
48          else printf("for unit parabolic input Ess=
                inf \n")
49       end
50   end
```

Scilab code Exa 7.5 steady state errors from closed loop tf

```
1  //steady state errors from closed loop transfer
      functions
2  s=%s
3  p=poly([5 1 0],'s','coeff');
4  q=poly([5 5 60 17 1],'s','coeff');
5  M=p/q
6  disp(M,"M(s)=")
7  H=1;
8  R=1;
9  b=coeff(p)
10 a=coeff(q)
11
12 //step input
13 if (a(1,1)==b(1,1)) then
14      printf("for unit step input Ess=0 \n" )
15 else
16      Ess=1/H*(1-(b(1,1)*H/a(1,1)))*R
17      disp(Ess,"for unit step input Ess=")
```

```
18 end
19
20 //ramp input
21 c=0
22 for i=1:2
23     if(a(1,i)-b(1,i)*H==0) then
24         c=c+1
25     end
26 end
27 if(c==2)
28         printf("for unit ramp input Ess=0 \n")
29     else if(c==1) then
30             Ess=(a(1,2)-b(1,2)*H)/a(1,1)*H
31             disp(Ess,"for unit ramp input Ess=")
32         else printf("for unit ramp input Ess=inf \n"
               )
33      end
34  end
35
36 //parabolic input
37 c=0
38 for i=1:3
39     if(a(1,i)-b(1,i)*H==0) then
40         c=c+1
41     end
42 end
43 if(c==3)
44         printf("for unit parabolic input Ess=0 \n")
45     else if(c==2) then
46             Ess=(a(1,3)-b(1,3)*H)/a(1,1)*H
47             disp(Ess,"for unit parabolic input Ess="
                  )
48         else printf("for unit parabolic input Ess=
               inf \n")
49      end
50  end
```

Scilab code Exa 7.6 steady state errors from closed loop tf

```
1  //steady state errors from closed loop transfer
   functions
2  s=%s
3  p=poly([5 1 0],'s','coeff');
4  q=poly([10 10 60 17 1],'s','coeff');
5  M=p/q
6  disp(M,"M(s)=")
7  H=2;
8  R=1;
9  b=coeff(p)
10 a=coeff(q)
11
12 //step input
13 if (a(1,1)==b(1,1)) then
14     printf("for step input Ess=0 \n" )
15 else
16     Ess=1/H*(1-(b(1,1)*H/a(1,1)))*R
17     disp(Ess,"for step input Ess=")
18 end
19
20 //ramp input
21 c=0
22 for i=1:2
23     if(a(1,i)-b(1,i)*H==0) then
24         c=c+1
25     end
26 end
27 if(c==2)
28         printf("for ramp input Ess=0 \n")
29     else if(c==1) then
30             Ess=(a(1,2)-b(1,2)*H)/a(1,1)*H
31             disp(Ess,"for ramp input Ess=")
```

```
32          else printf("for ramp input Ess=inf \n")
33       end
34   end
35
36  //parabolic input
37  c=0
38  for i=1:3
39      if(a(1,i)-b(1,i)*H==0) then
40          c=c+1
41      end
42  end
43  if(c==3)
44          printf("for parabolic input Ess=0 \n")
45      else if(c==2) then
46              Ess=(a(1,3)-b(1,3)*H)/a(1,1)*H
47              disp(Ess,"for parabolic input Ess=")
48          else printf("for parabolic input Ess=inf \n"
                )
49       end
50   end
```

Chapter 8

Root Locus Technique

Scilab code Exa 8.1 poles and zeros

```
1 //poles and zeroes
2 s=%s
3 sys=syslin('c',(s+1)/(s*(s+2)*(s+3)))
4 plzr(sys)
5 printf("three points on the root loci at which K=0
    and those at which K=inf are shown in fig")
```

Scilab code Exa 8.2 root locus

```
1 //root locus
2 s=%s
3 sys=syslin('c',(s+1)/(s*(s+2)*(s+3)))
4 evans(sys)
5 printf("number of branches of root loci is 3 as
    equation is of 3rd order")
```

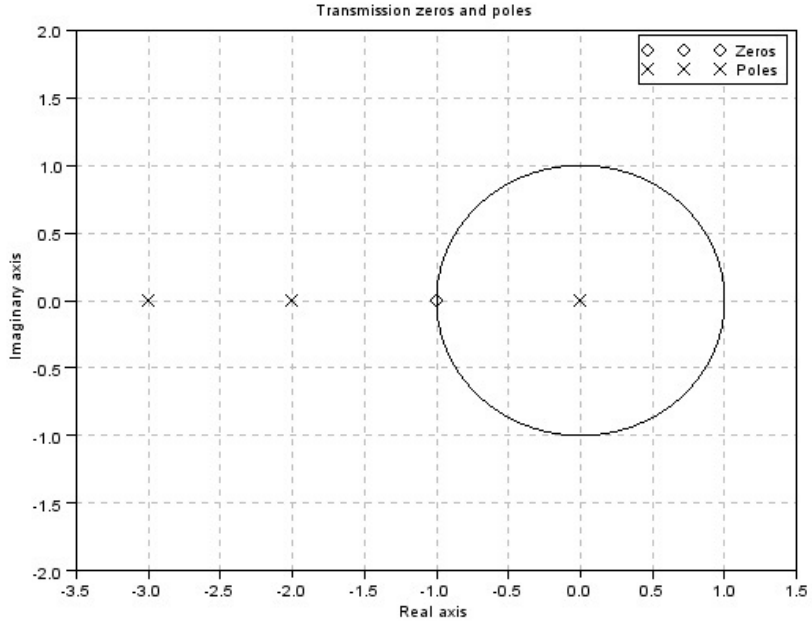

Figure 8.1: poles and zeros

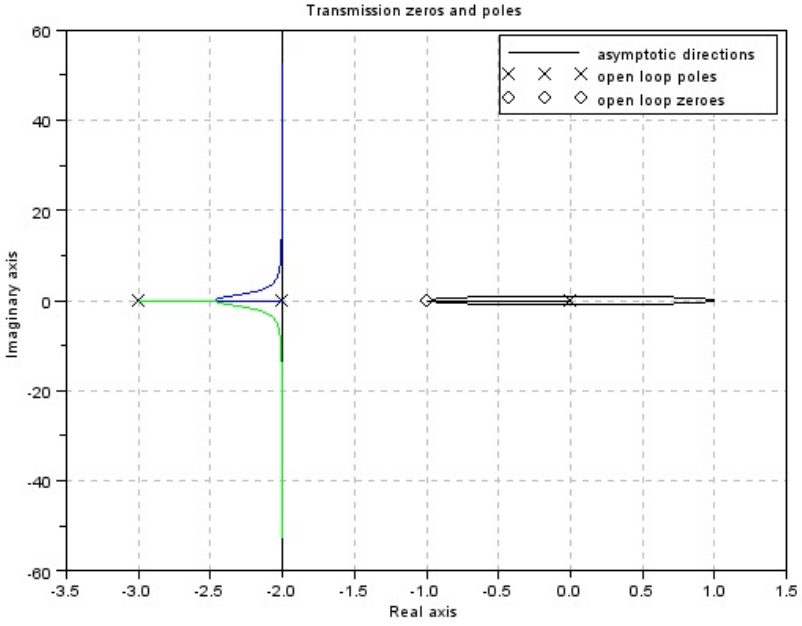

Figure 8.2: root locus

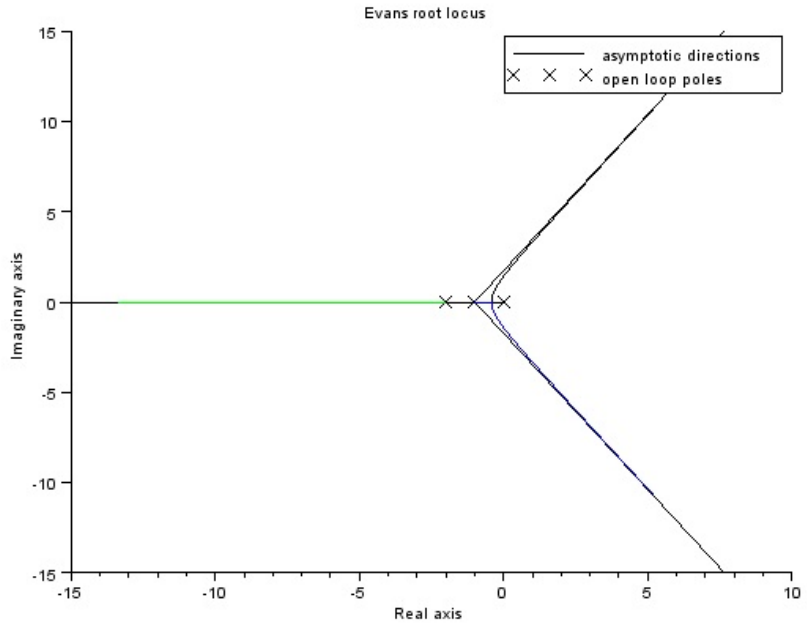

Figure 8.3: root locus

Scilab code Exa 8.3 root locus

```
1 //root locus
2 s=%s
3 sys=syslin('c',1/(s*(s+2)*(s+1)))
4 clf
5 evans(sys)
6 printf("root loci is symmetical to both axis")
```

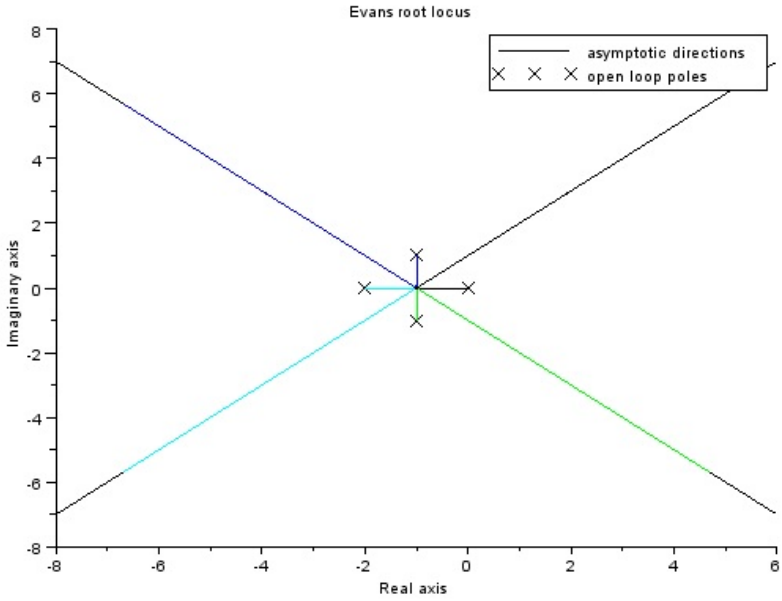

Figure 8.4: root locus

Scilab code Exa 8.4 root locus

```
1 //root locus
2 s=%s
3 sys=syslin('c',1/(s*(s+2)*(s^2+2*s+2)))
4 clf
5 evans(sys)
6 printf("when pole zero configuration is symmetrical
    wrt a point in s plane,then root loci is
    symmetrical to that point")
```

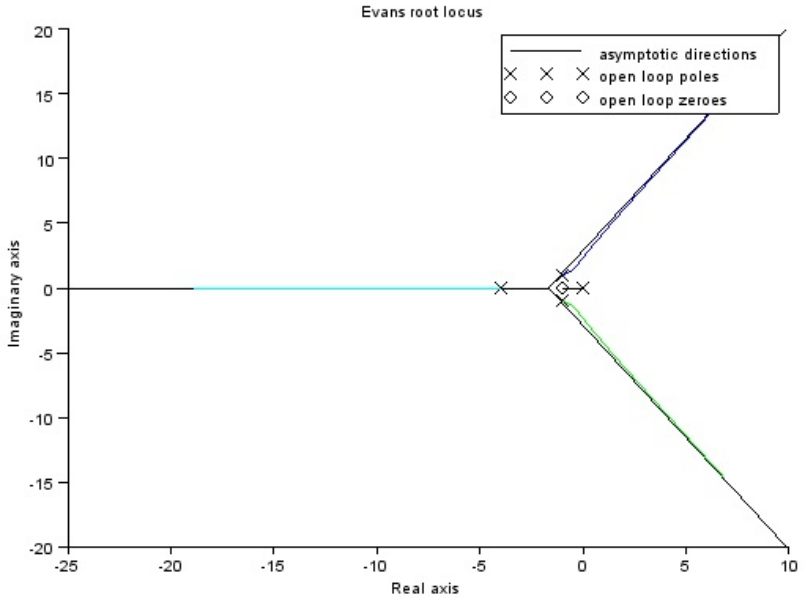

Figure 8.5: root locus

Scilab code Exa 8.5 root locus

```
1 //root locus
2 s=%s
3 sys=syslin('c',(s+1)/(s*(s+4)*(s^2+2*s+2)))
4 clf
5 evans(sys)
6 n=4;
7 disp(n,"no of poles=")
8 m=1;
9 disp(m,"no of poles=")
```

```
10 //angle of asymptotes
11 printf("angle of asymptotes of RL")
12 for i=0:(n-m-1)
13     O=((2*i)+1)/(n-m)*180
14     disp(O,"q=")
15     end
16 printf("angle of asymptotes of CRL")
17 for i=0:(n-m-1)
18     O=(2*i)/(n-m)*180
19     disp(O,"q=")
20 end
21 //centroid
22 printf("Centroid=((sum of all real part of poles of
     G(s)H(s))-(sum of all real part of zeros of G(s)H
     (s))/(n-m) \n")
23 C=((0-4-1-1)-(-1))/(n-m);
24 disp(C,"centroid=")
```

Scilab code Exa 8.8 angle of departure and angle of arrivals

```
1 //angle of departure and angle of arrivals
2 s=%s
3 sys=syslin('c',1/(s*(s+3)*(s^2+2*s+2)))
4 clf
5 evans(sys)
6 printf("angle of arrival and departure of root loci
     on the real axis are not affected by complex
     poles and zeroes of G(s)H(s)")
```

Scilab code Exa 8.9 multiple order pole

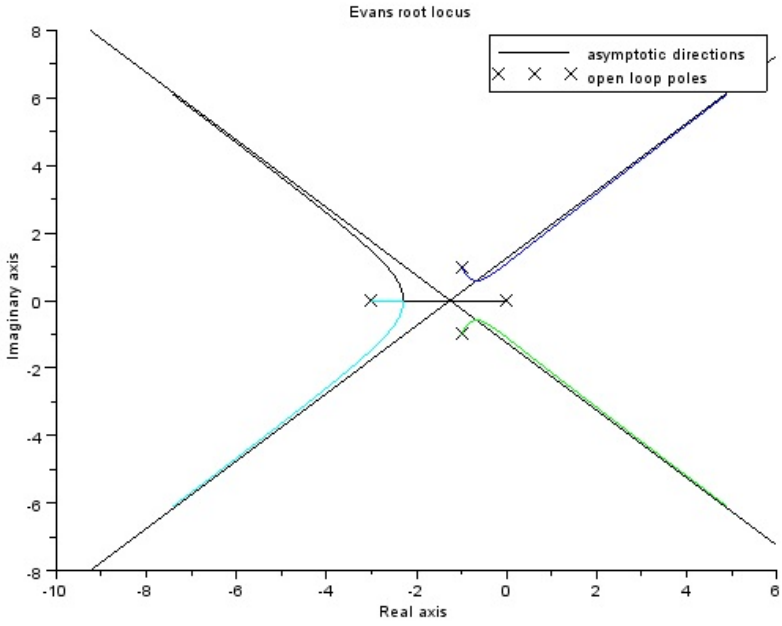

Figure 8.6: angle of departure and angle of arrivals

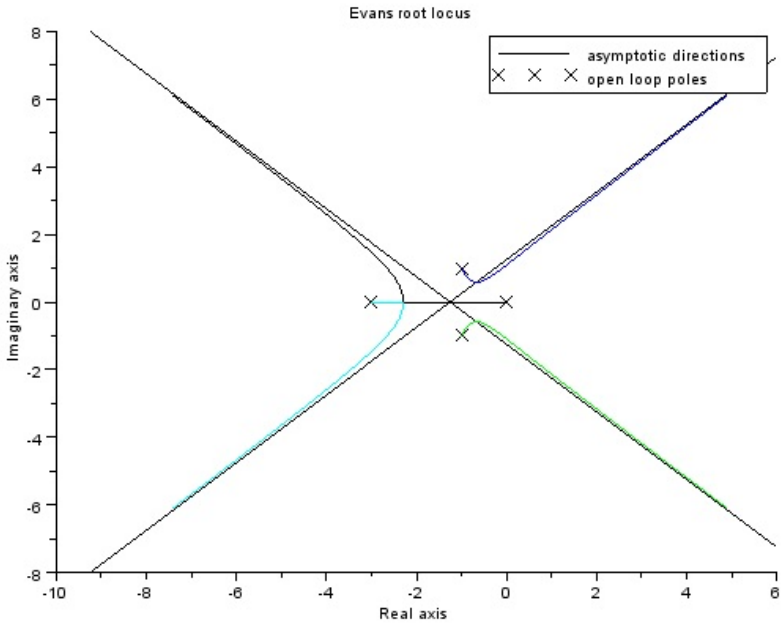

Figure 8.7: intersection of root loci with real axis

```
1 //multiple order pole
2 s=%s
3 sys=syslin('c',(s+3)/(s*(s+2)^3))
4 clf
5 evans(sys)
6 printf("this shows that whole real axis is occupied
    by RL and CRL")
```

Scilab code Exa 8.10 intersection of root loci with real axis

```
1 //intersection of root loci with real axis
```

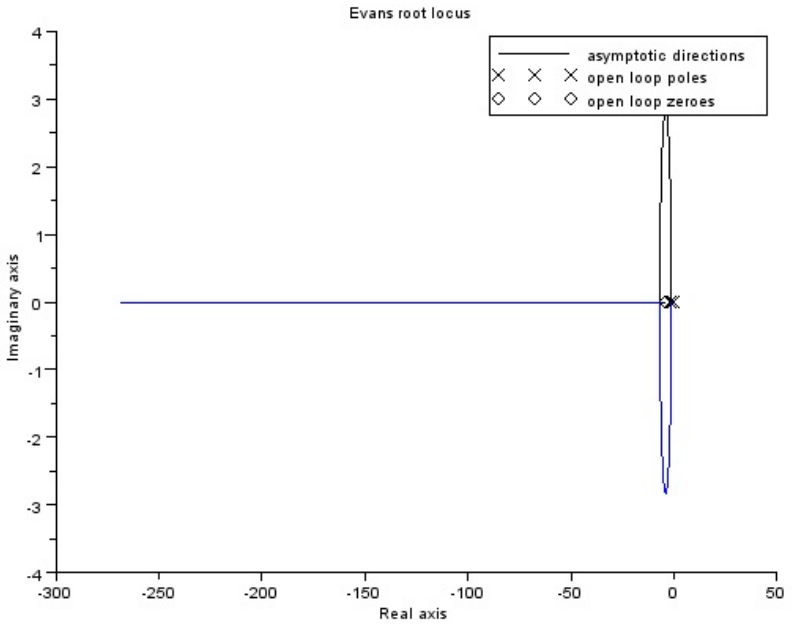

Figure 8.8: breakaway points

```
2  s=%s
3  sys=syslin('c',1/(s*(s+3)*(s^2+2*s+2)))
4  clf
5  evans(sys)
6  K=kpure(sys)
7  disp(K,"value of K where RL crosses jw axis=")
8  p=poly([K 6 8 5 1],'s','coeff')
9  x=roots(p)
10 x1=clean(x(1,1))
11 x2=clean(x(2,1))
12 disp(x2,x1,"crossover points on jw axis=")
```

Scilab code Exa 8.11 breakaway points

```
1 //breakaway points
2 s=%s
3 sys=syslin('c',(s+4)/(s*(s+2)))
4 evans(sys)
5 syms s
6 d=derivat(sys)
7 n=numer(d)
8 a=roots(n)     //a=breakaway points
9 disp(a,"breakaway points=")
10 for i=1:2
11     K=-a(i,1)*(a(i,1)+2)/(a(i,1)+4)
12     disp(a(i,1),"s=")
13     disp(K,"K=")
14 end
15 printf("if K is positive breakaway point lies on RL
   or else on CRL")
```

Scilab code Exa 8.12 breakaway points

```
1 //breakaway points
2 s=%s
3 sys=syslin('c',(s+2)/(s^2+2*s+2))
4 evans(sys)
5 syms s
6 d=derivat(sys)
7 n=numer(d)
8 a=roots(n)     //a=breakaway points
9 disp(a,"breakaway points=")
10 for i=1:2
11     K=-(a(i,1)^2+2*a(i,1)+2)/(a(i,1)+2)
12         disp(a(i,1),"s=")
```

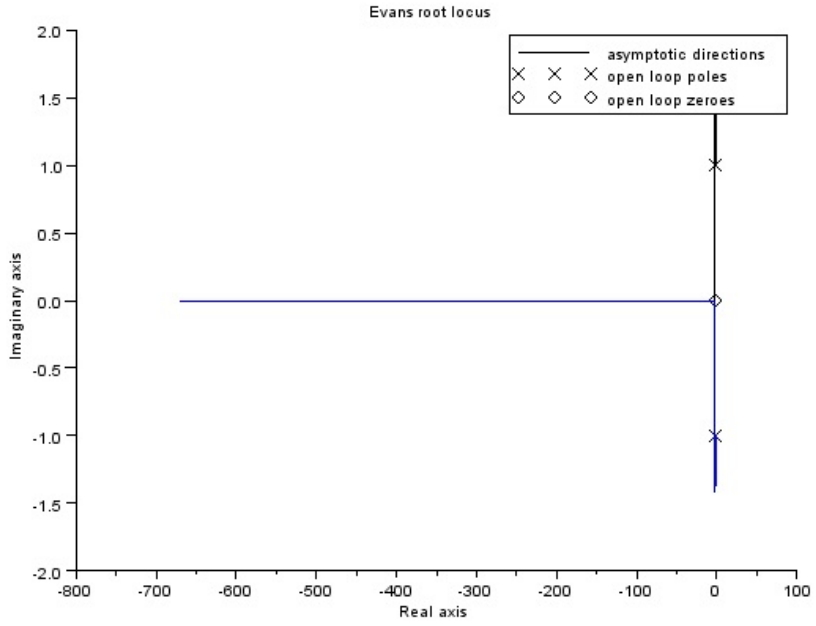

Figure 8.9: breakaway points

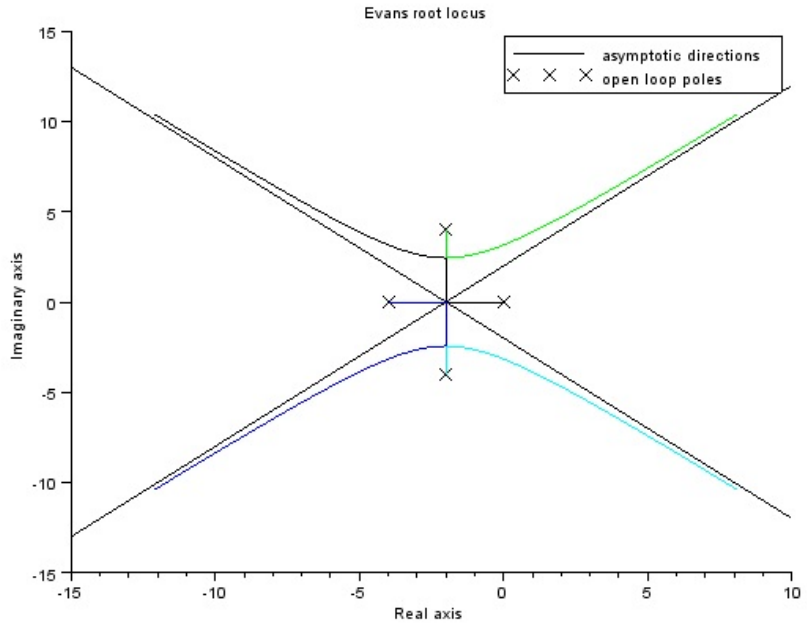

Figure 8.10: breakaway points

```
13          disp(K,"K=")
14  end
15  printf("if K is positive breakaway point lies on RL
       or else on CRL")
```

Scilab code Exa 8.13 breakaway points

```
1  //breakaway points
2  s=%s
3  sys=syslin('c',1/(s*(s+4)*(s^2+4*s+20)))
4  evans(sys)
```

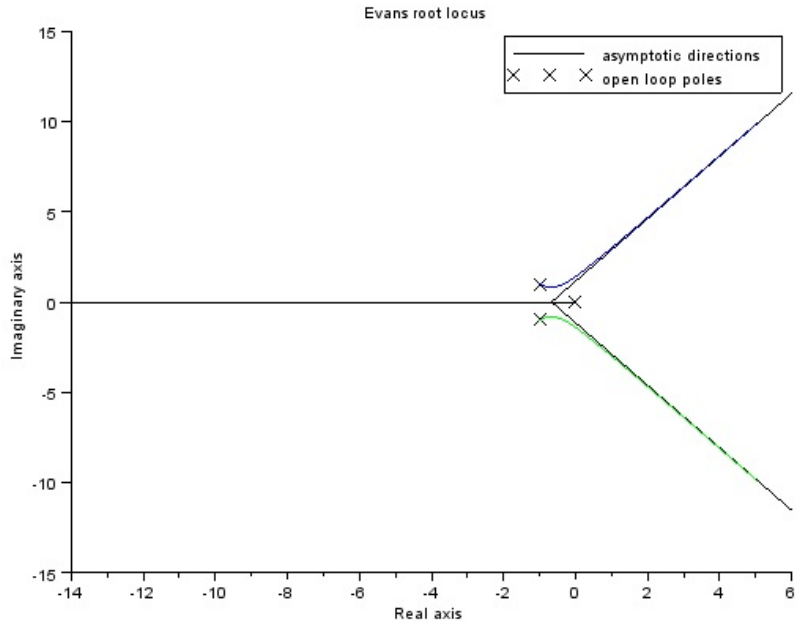

Figure 8.11: breakaway points

```
5  syms s
6  d=derivat(sys)
7  n=numer(d)
8  a=roots(n)     //a=breakaway points
9  disp(a,"breakaway points=")
10 for i=1:3
11     K=-a(i,1)*(a(i,1)+4)*(a(i,1)^2+4*a(i,1)+20)
12         disp(a(i,1),"s=")
13         disp(K,"K=")
14 end
15 printf("if K is positive breakaway point lies on RL
   or else on CRL")
```

Scilab code Exa 8.14 breakaway points

```
1 //breakaway points
2 s=%s
3 sys=syslin('c',1/(s*(s^2+2*s+2)))
4 evans(sys)
5 syms s
6 d=derivat(sys)
7 n=numer(d)
8 a=roots(n)     //a=breakaway points
9 disp(a,"breakaway points=")
10 for i=1:2
11     K=-a(i,1)^2+2*a(i,1)+2
12         disp(a(i,1),"s=")
13         disp(K,"K=")
14 end
15 printf("if K is complex then point is not a break
   away point")
```

Scilab code Exa 8.15 root sensitivity

```
1 //root sensitivity
2 s=%s
3 sys1=syslin('c',1/(s*(s+1)))
4 evans(sys1)
5
6 sys2=syslin('c',(s+2)/(s^2*(s+1)^2))
7 evans(sys2)
8
9 printf("root densitivity at breakaway points is
   infinite")
```

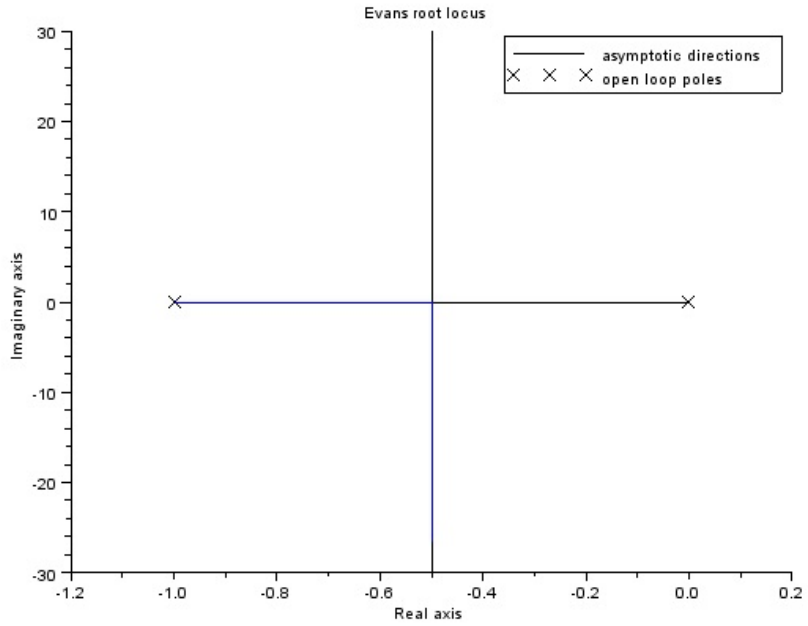

Figure 8.12: root sensitivity

Scilab code Exa 8.16 calculation of K on root loci

```
1 //calculation of K on root loci
2 s=%s
3 sys=syslin('c',(s+2)/(s^2+2*s+2))
4 evans(sys)
5 //value of K at s=0
```

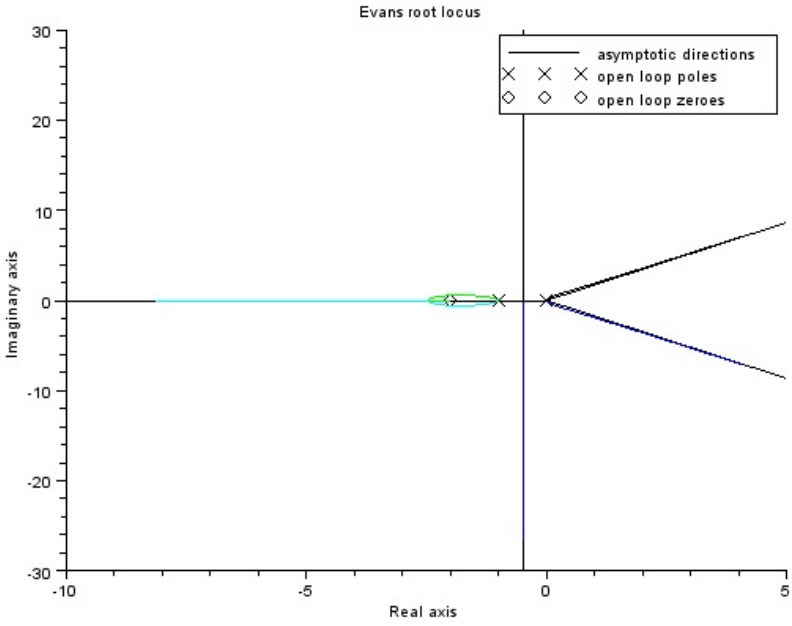

Figure 8.13: root sensitivity

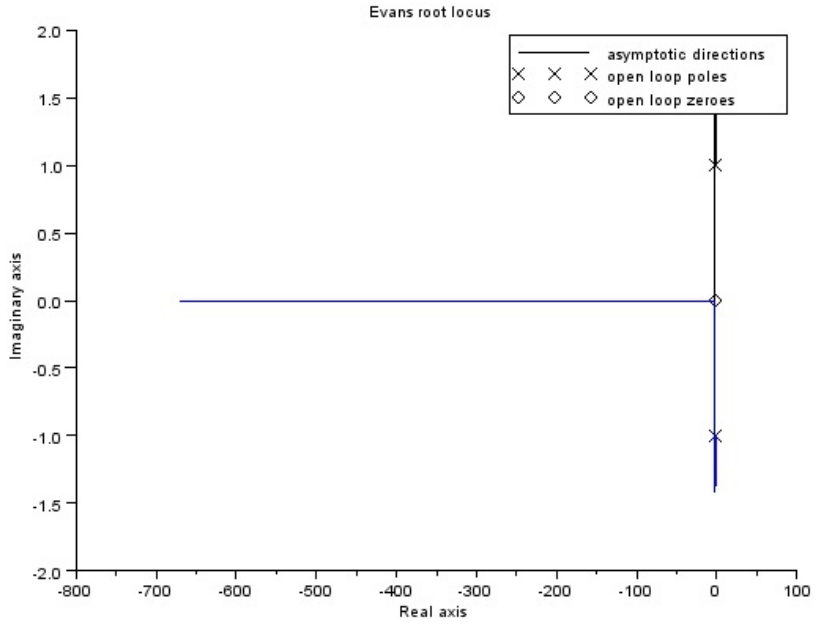

Figure 8.14: calculation of K on root loci

```
6  printf("K=A*B/C \n A and B are lenths of vectors
      drawn from poles of sys \n C is lenths of vector
      drawn from zero of sys")
7  A=sqrt((-1)^2+1^2)
8  B=sqrt((-1)^2+(-1)^2)
9  C=-2
10 K=A*B/C
11 disp(K,"value of K at s=0 is")
```

Scilab code Exa 8.17 properties of root loci

```
1  //properties of root loci
2  s=%s
3  sys=syslin('c',(s+3)/(s*(s+5)*(s+6)*(s^2+2*s+2)))
4  d=denom(sys)
5  n=numer(sys)
6  p=roots(d)
7  z=roots(n)
8  disp(p,"poles of sys=")
9  disp(z,"zeroes of sys=")
10 n=length(p)
11 m=length(z)
12 disp(n,"no of poles=")
13 disp(m,"no of zeroes=")
14 if (n>m) then
15     disp(n,"no of branches of RL=")
16 else
17     disp(m,"no of branches of CRL=")
18 end
19 printf("the root loci are symmetrical with respect
      to the real axis of the plane")
```

Scilab code Exa 8.18 effect of addition of poles to system

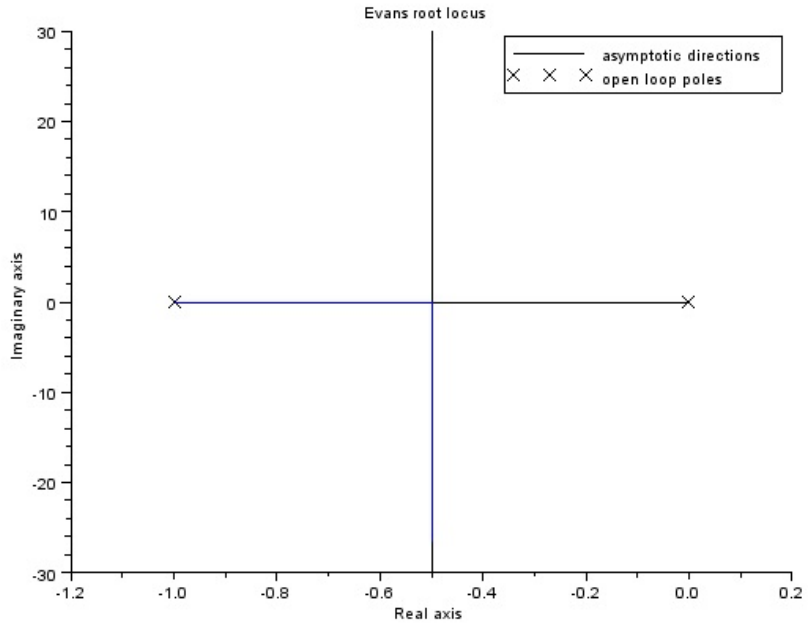

Figure 8.15: effect of addition of poles to system

```
1 //effect of addition of poles to sys
2 s=%s
3 sys=syslin('c',1/(s*(s+1)))     //a=1
4 evans(sys)
5 sys1=syslin('c',1/(s*(s+1)*(s+2)))    //b=2
6 evans(sys1)
7 printf("adding a pole to sys has effect of pushing
    the root loci towards the RHP")
```

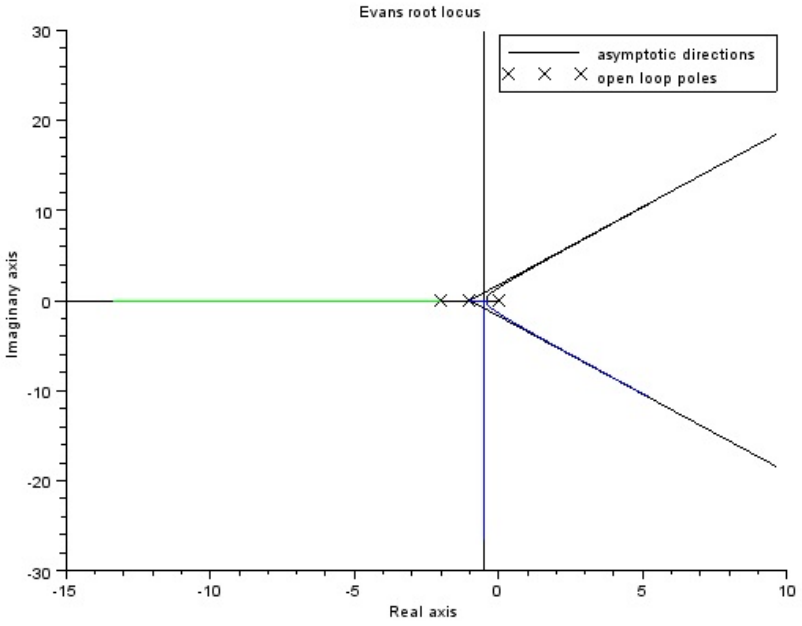

Figure 8.16: effect of addition of poles to system

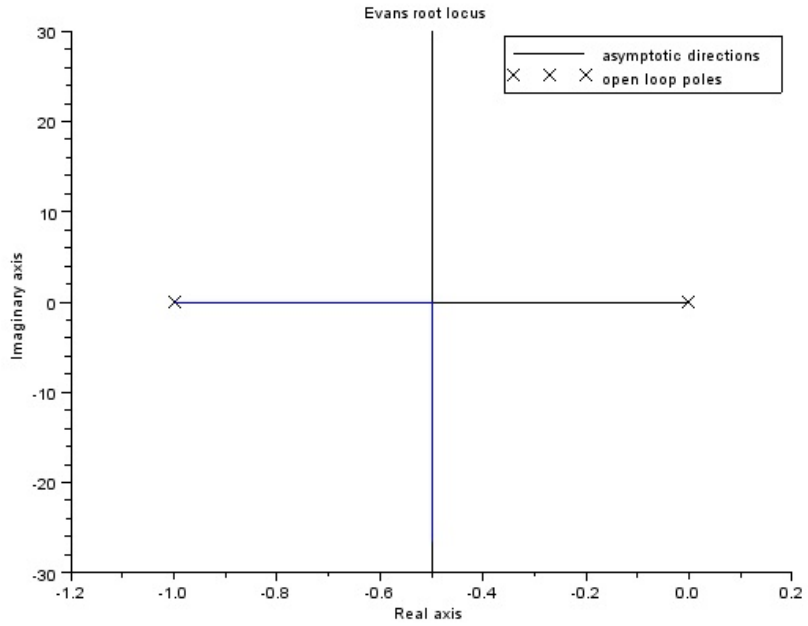

Figure 8.17: effect of addition of zeroes to system

Scilab code Exa 8.19 effect of addition of zeroes to system

```
1 //effect of addition of zeroes to sys
2 s=%s
3 sys=syslin('c',1/(s*(s+1)))     //a=1
4 evans(sys)
5 sys1=syslin('c',(s+2)/(s*(s+1)))    //b=2
6 //evans(sys1)
7 printf("adding a LHP zero to sys has effect of
    moving and bending the root loci towards the LHP"
    )
```

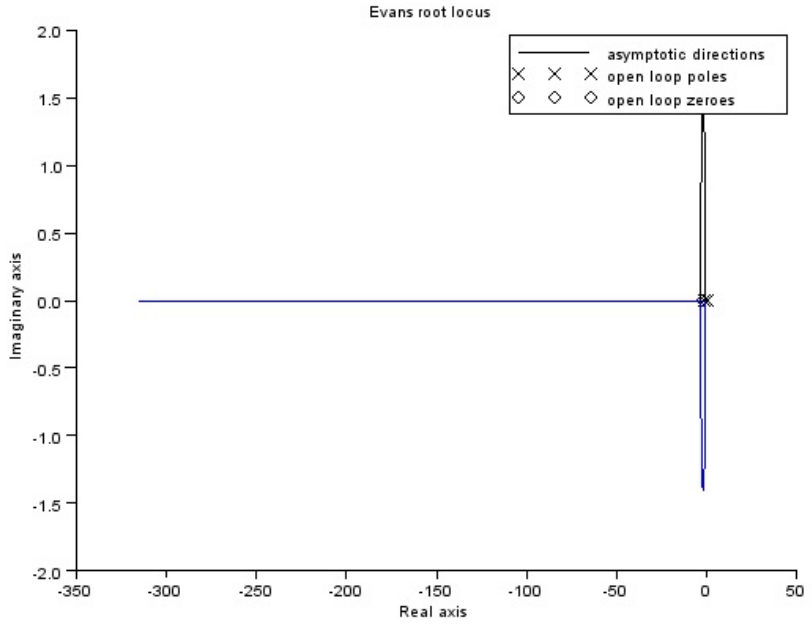

Figure 8.18: effect of addition of zeroes to system

Scilab code Exa 8.20 effect of moving poles near jw axis

```
1 //effect of moving pole near jw axis
2 s=%s
3 sys1=syslin('c',(s+1)/(s^2*(s+10)))     //a=10 b=1
4 evans(sys1)
5 sys2=syslin('c',(s+1)/(s^2*(s+9)))    //a=9
6 evans(sys2)
7 sys3=syslin('c',(s+1)/(s^2*(s+8)))     //a=8
8 evans(sys3)
9 sys4=syslin('c',(s+1)/(s^2*(s+3)))    //a=3
10 evans(sys4)
11 sys5=syslin('c',(s+1)/(s^2*(s+1)))    //a=1
12 evans(sys5)
13 printf("as pole is moved towards jw axis RL also
    moves towards jw axis")
```

Scilab code Exa 8.21 effect of moving poles awat from jw axis

```
1 //effect of moving pole away from jw axis
2 s=%s
3 sys1=syslin('c',(s+2)/(s*(s^2+2*s+1)))     //a=1
4 evans(sys1)
5 sys2=syslin('c',(s+1)/(s*(s^2+2*s+1.12)))    //a=1.12
6 evans(sys2)
7 sys3=syslin('c',(s+1)/(s*(s^2+2*s+1.185)))     //a
    =1.185
8 evans(sys3)
```

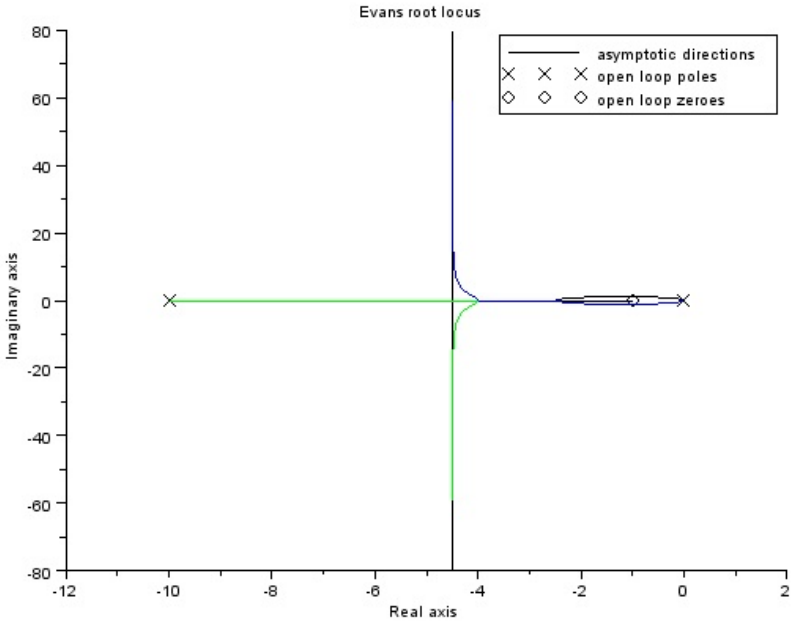

Figure 8.19: effect of moving poles near jw axis

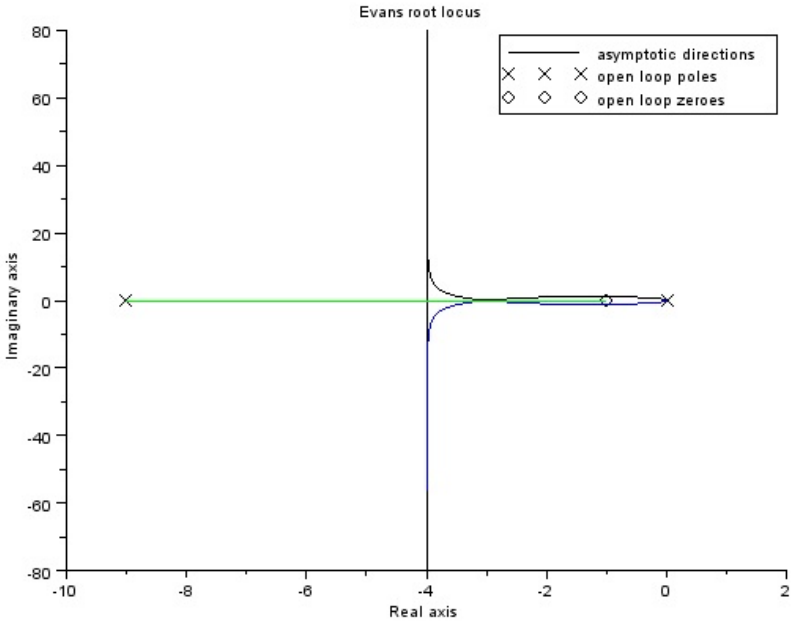

Figure 8.20: effect of moving poles near jw axis

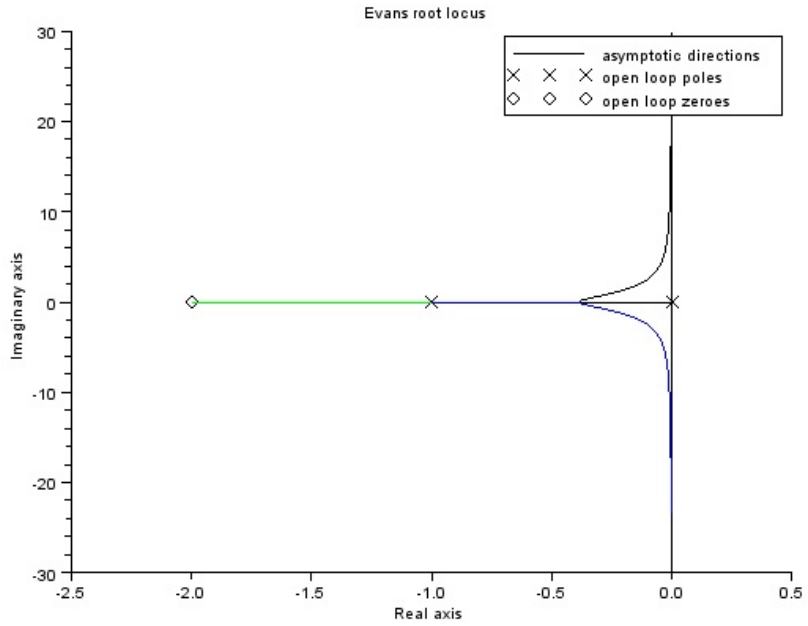

Figure 8.21: effect of moving poles awat from jw axis

```
9  sys4=syslin('c',(s+1)/(s*(s^2+2*s+3)))     //a=3
10 evans(sys4)
11 printf("as pole is moved away from jw axis RL also
     moves away from jw axis")
```

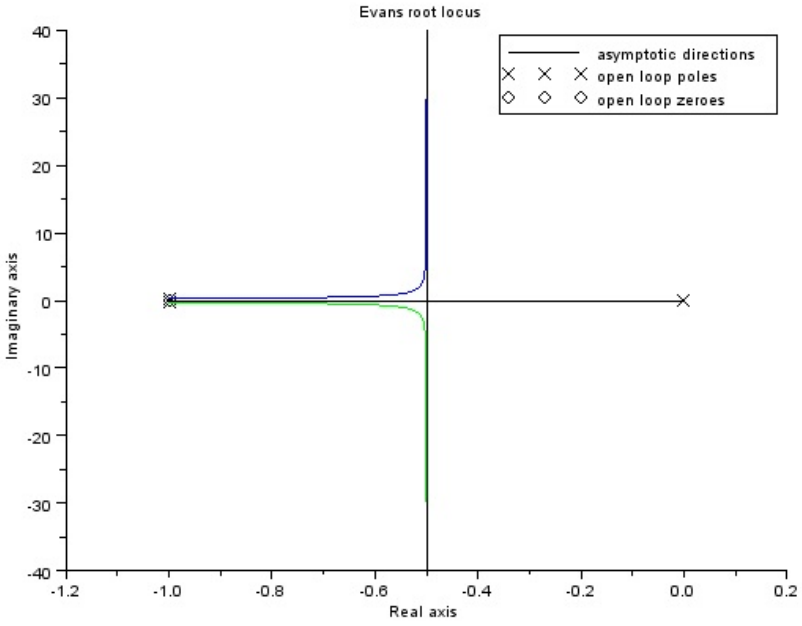

Figure 8.22: effect of moving poles awat from jw axis

Chapter 9

Frequency Domain Analysis

Scilab code Exa 9.1 nyquist plot

```
1 //nyquist plot
2 s=%s;
3 sys=syslin('c',1/(s*(s+2)*(s+10)))
4 nyquist(sys)
5 show_margins(sys,'nyquist')
6 K=kpure(sys)
7 disp(K,"system is stable for 0<K<")
```

Scilab code Exa 9.2 nyquist plot

```
1 //nyquist plot
2 s=%s;
3 sys=syslin('c',s*(s^2+2*s+2)/(s^2+5*s+1))
4 nyquist(sys)
5 show_margins(sys,'nyquist')
```

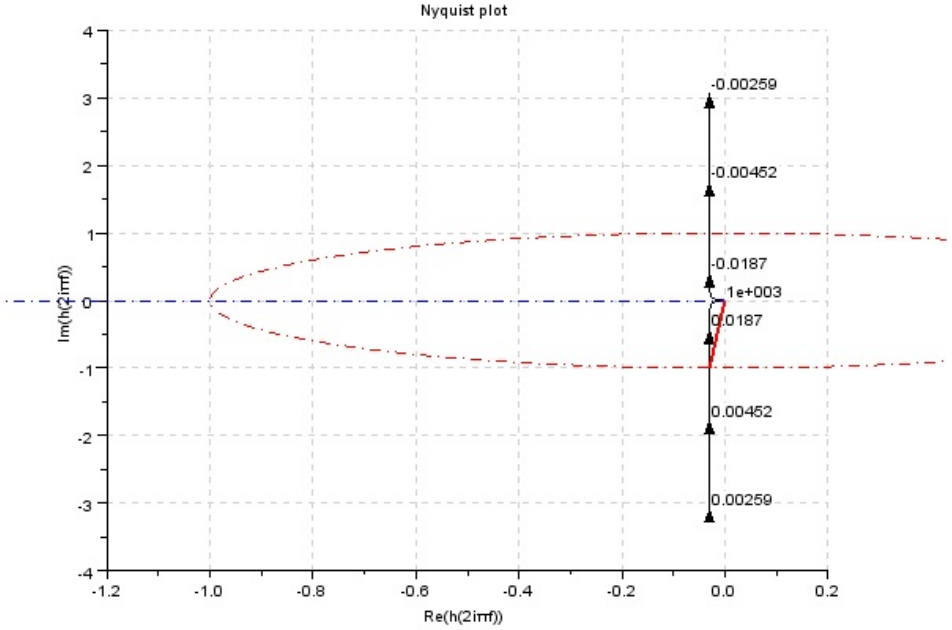

Figure 9.1: nyquist plot

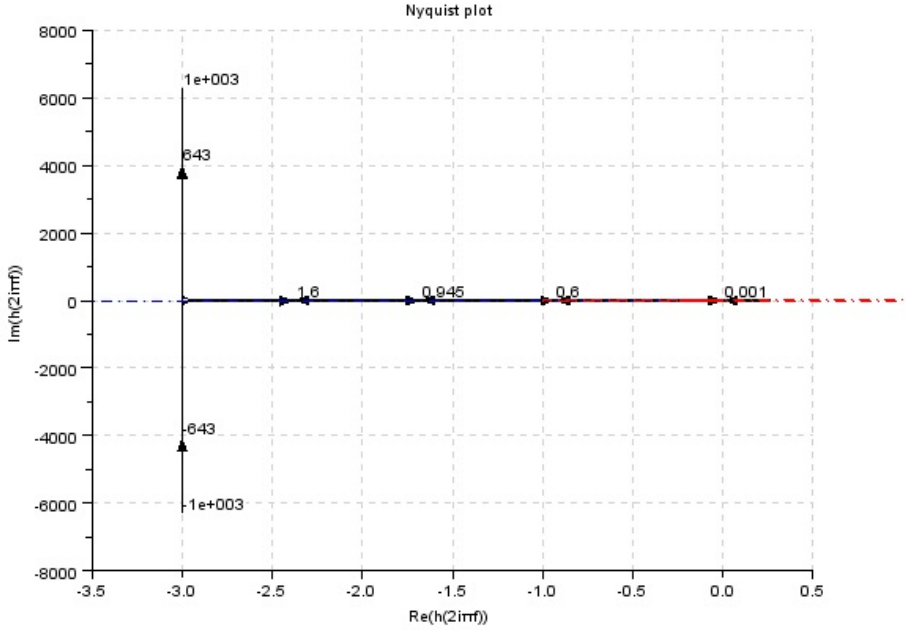

Figure 9.2: nyquist plot

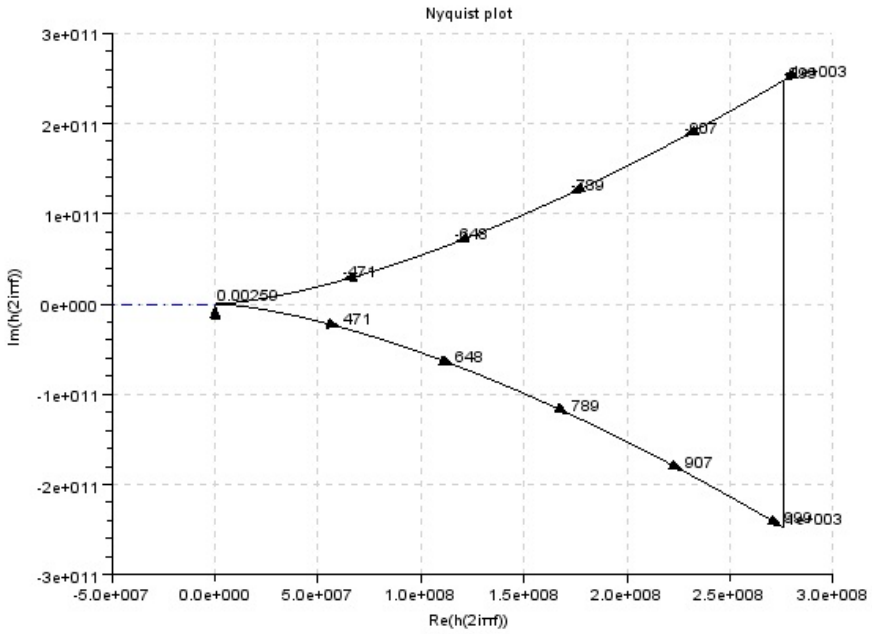

Figure 9.3: stability of non minimum phase loop tf

```
6 printf("Since P=0(no of poles in RHP)=Poles of G(s)H
    (s) \n here the number of zeros of 1+G(s)H(s) in
    the RHP is N>0 \n hence the system is unstable")
```

Scilab code Exa 9.3 stability of non minimum phase loop tf

```
1 //stability of non minimum phase loop
    transfer_function
2 s=%s;
3 sys=syslin('c',(s^2-s+1)/s*(s^2-6*s+5))
4 nyquist(sys)
```

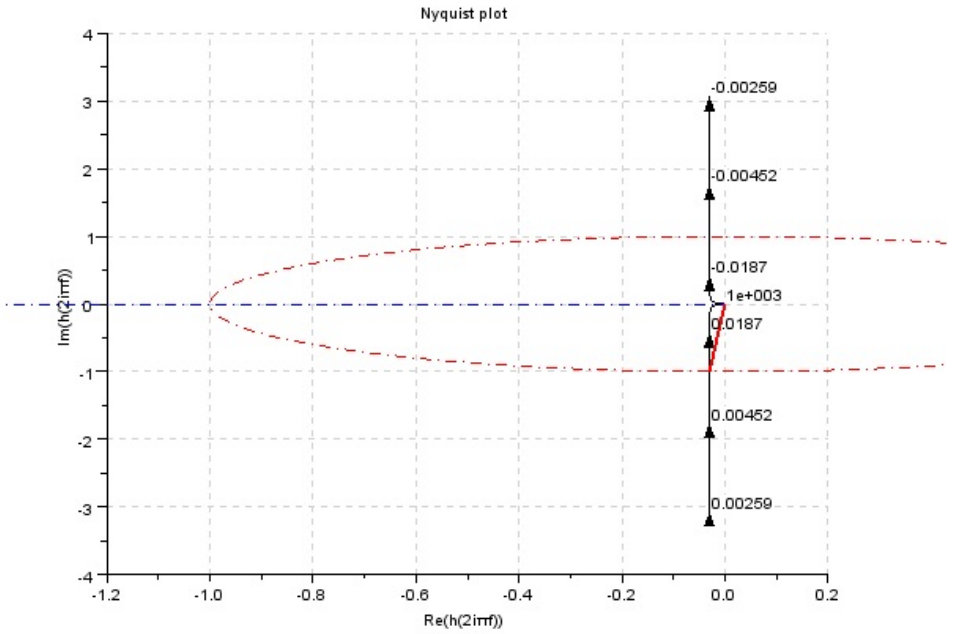

Figure 9.4: stability of minimum phase loop tf

```
5  show_margins(sys,'nyquist')
6  printf("Z=0 hence sys is closed loop stable but as
       it is a non minimum phase loop_function it should
        satisfy angle criterion")
7  Z=0//no of zeroes of 1+G(s)H(s) in RHP
8  P=2//no of poles in RHP
9  Pw=1//no of poles on jw axis including origin
10 theta=(Z-P-0.5*Pw)*180
11 disp(theta,"theta=")
12 printf("theta from nyquist_plot = -90 \n hence
       system is unstabe")
```

Scilab code Exa 9.4 stability of minimum phase loop tf

```
1 //stability of minimum phase loop transfer function
2 s=%s;
3 sys=syslin('c',1/(s*(s+2)*(s+10)))
4 nyquist(sys)
5 show_margins(sys,'nyquist')
6 Z=0//no of zeroes of 1+G(s)H(s) in RHP
7 P=0//no of poles in RHP
8 Pw=1//no of poles on jw axis including origin
9 theta=(Z-P-0.5*Pw)*180
10 disp(theta,"theta=")
11 printf("theta from nyquist_plot = -90 \n hence
   system is stabe")
```

Scilab code Exa 9.5 stability of non minimum phase loop tf

```
1 //stability of non minimum phase loop
   transfer_function
2 s=%s;
3 sys=syslin('c',(s-1)/s*(s+1))
4 nyquist(sys)
5 show_margins(sys,'nyquist')
6 P=0//no of poles in RHP
7 Pw=1//no of poles on jw axis including origin
8 theta=90//as seen from nyquist plot
9 Z=(theta/180)+0.5*Pw+P
10 disp(Z,"Z=")
11 printf("Z is not equal to 0. \n hence system is
   unstabe")
```

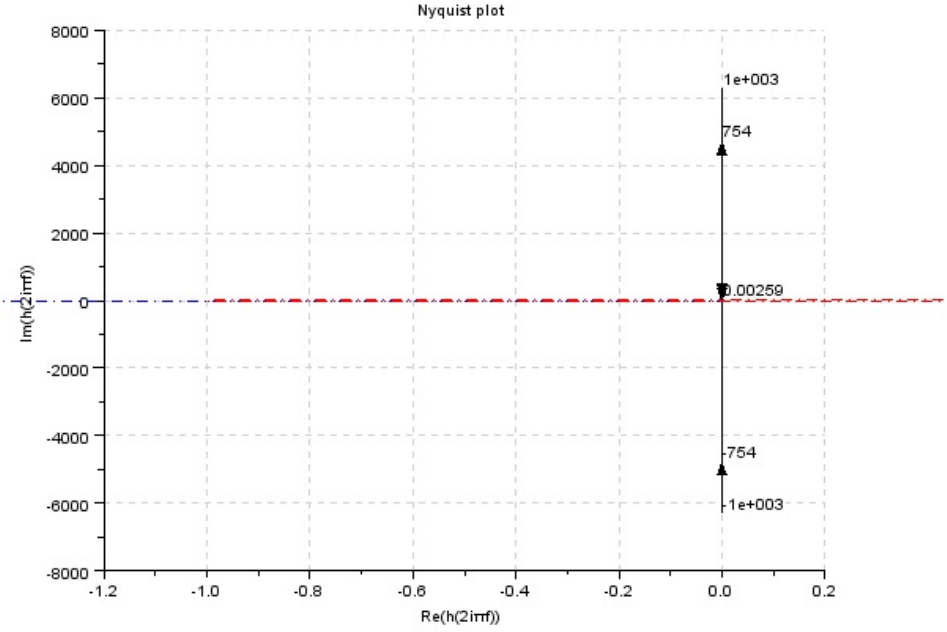

Figure 9.5: stability of non minimum phase loop tf

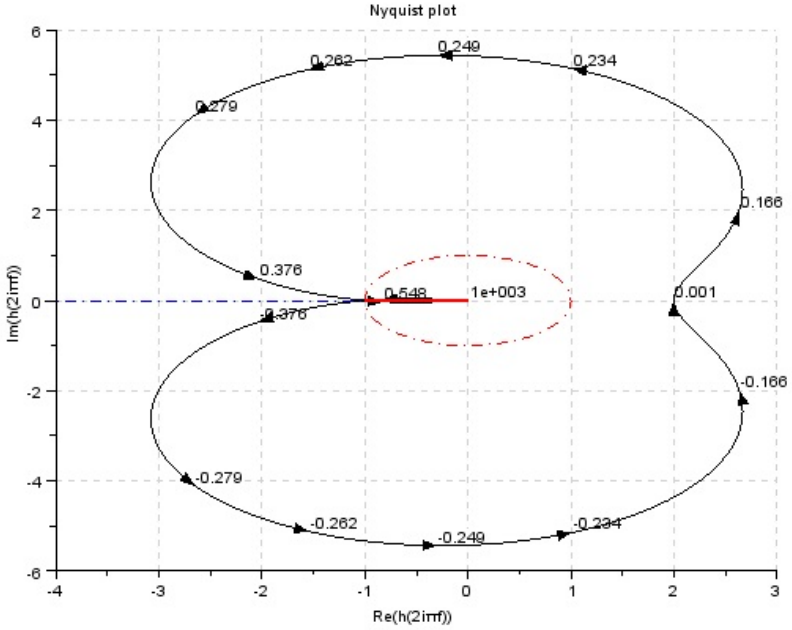

Figure 9.6: stability of non minimum phase loop tf

Scilab code Exa 9.6 stability of non minimum phase loop tf

```
1 //stability of non minimum phase loop
    transfer_function
2 s=%s;
3 sys=syslin('c',10*(s+2)/(s^3+3*s^2+10))
4 nyquist(sys)
5 show_margins(sys,'nyquist')
6 printf("Z=0 hence sys is closed loop stable but as
    it is a non minimum phase loop_function it should
     satisfy angle criterion")
7 Z=0//no of zeroes of 1+G(s)H(s) in RHP
8 P=2//no of poles in RHP
9 Pw=0//no of poles on jw axis including origin
10 theta=(Z-P-0.5*Pw)*180
11 disp(theta,"theta for stability=")
12 printf("theta from nyquist_plot = -360 \n hence
    system is stabe")
```

Scilab code Exa 9.7 stability of non minimum phase loop tf

```
1 //stability of non minimum phase loop
    transfer_function
2 s=%s;
3 sys=syslin('c',1/(s+2)*(s^2+4))
4 nyquist(sys)
5 show_margins(sys,'nyquist')
6 Z=0//no of zeroes of 1+G(s)H(s) in RHP(for sys to be
    stable)
7 P=0//no of poles in RHP
```

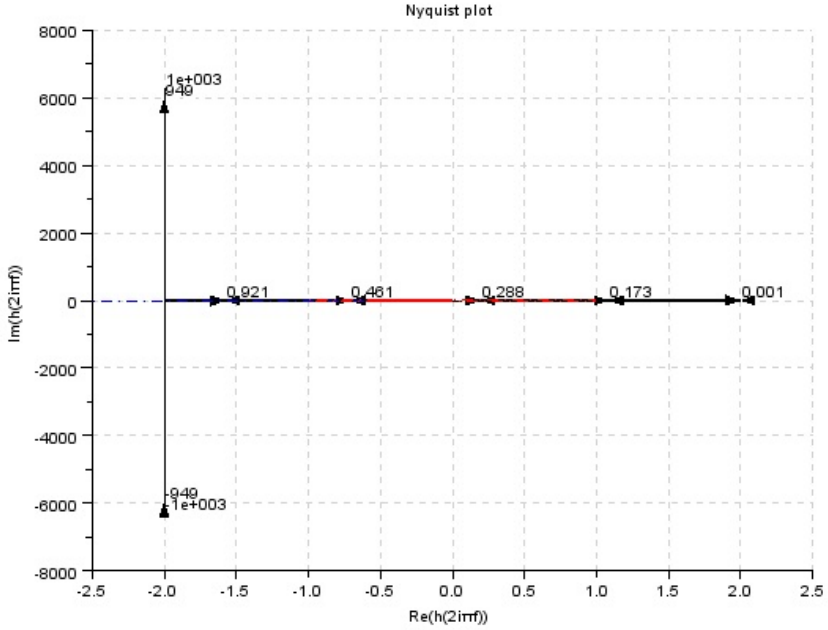

Figure 9.7: stability of non minimum phase loop tf

```
8  Pw=2//no of poles on jw axis including origin
9  theta=(Z-P-0.5*Pw)*180
10 disp(theta,"for stability theta=")
11 printf("theta from nyquist_plot = 135 \n hence
      system is unstabe")
```

Scilab code Exa 9.8 effect of addition of poles

```
1  //effect of addition of poles
2  s=%s;
3  sys1=syslin('c',1/(s^2*(s+1)))//taking T1=1
4  nyquist(sys1)
5  show_margins(sys1,'nyquist')
6  sys2=syslin('c',1/(s^3*(s+1)))
7  //nyquist(sys2)
8  //show_margins(sys2,'nyquist')
9  printf("these two plots show that addition of poles
      decreases stability")
```

Scilab code Exa 9.9 effect of addition of zeroes

```
1  //effect of addition of zeroes
2  s=%s;
3  sys1=syslin('c',1/(s*(s+1)*(2*s+1)))//taking T1=1,T2
      =2
4  nyquist(sys1)
5  show_margins(sys1,'nyquist')
6  sys2=syslin('c',(3*s+1)/(s*(s+1)*(2*s+1)))//Td=3
7  //nyquist(sys2)
```

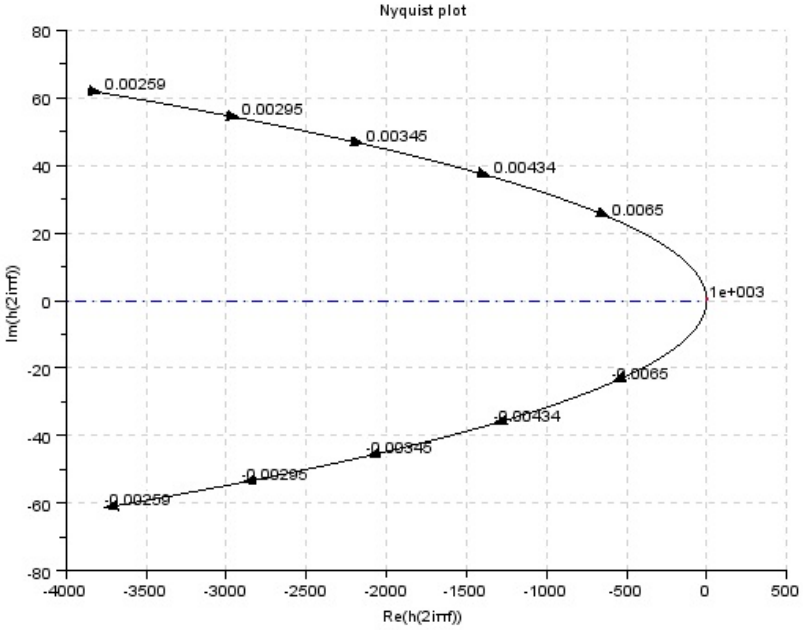

Figure 9.8: effect of addition of poles

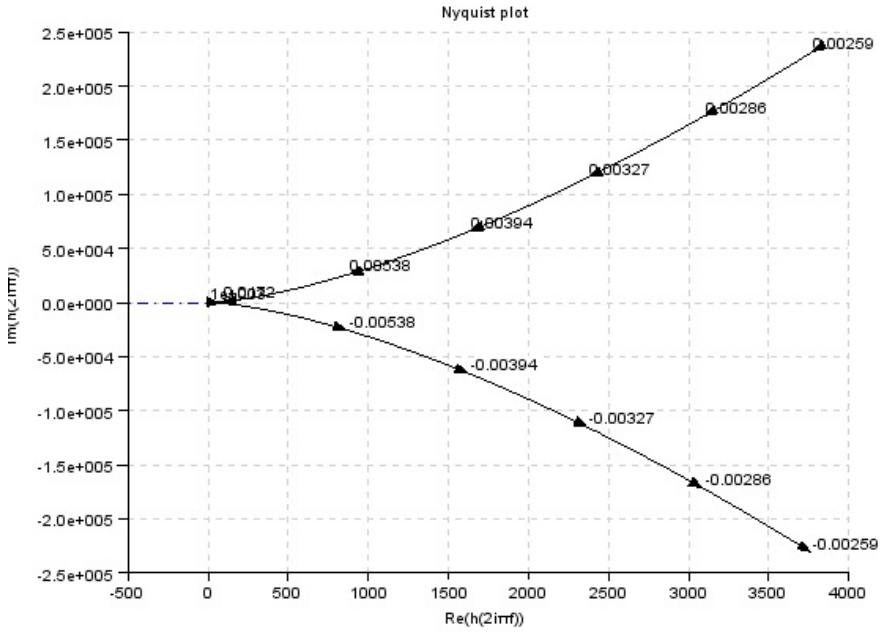

Figure 9.9: effect of addition of poles

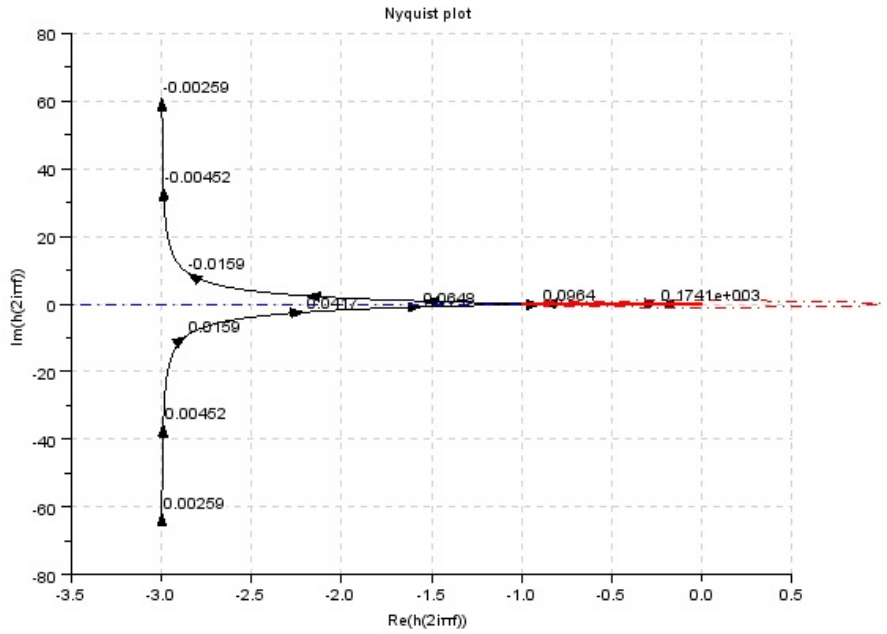

Figure 9.10: effect of addition of zeroes

```
8 //show_margins(sys2,'nyquist')
9 printf("these two plots show that addition of poles
    increases stability")
```

Scilab code Exa 9.10 multiple loop systems

```
1 //multiple loop systems
```

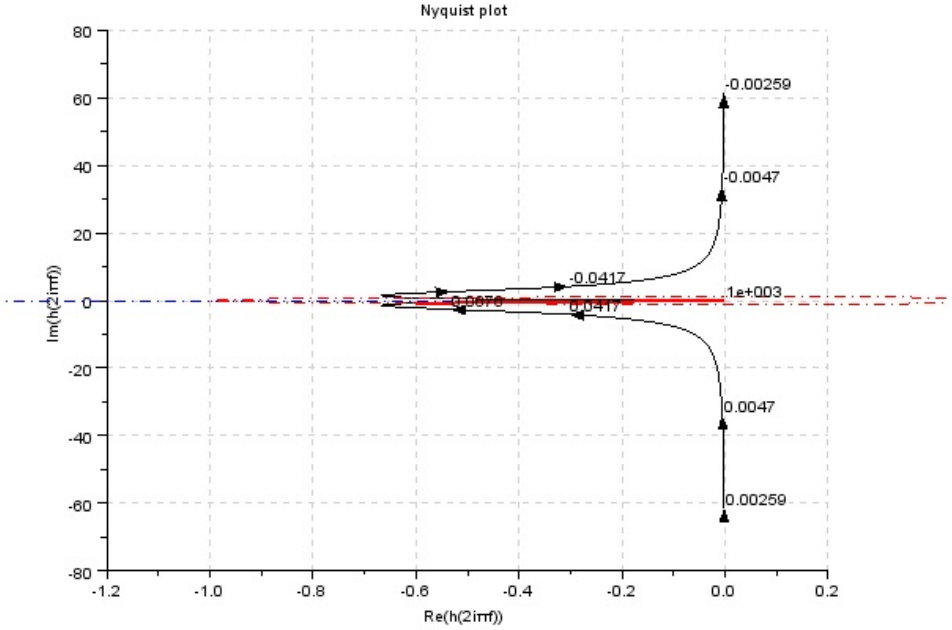

Figure 9.11: effect of addition of zeroes

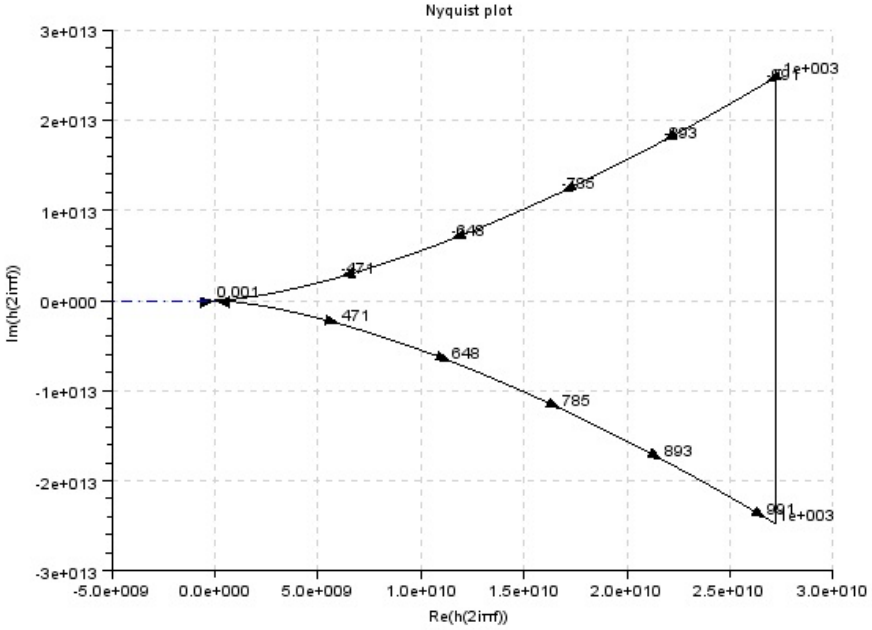

Figure 9.12: multiple loop systems

```
2 s=%s
3 innerloop=syslin('c',6/s*(s+1)*(s+2))
4 nyquist(innerloop)
5 show_margins(innerloop,'nyquist')
6 printf("nyquist plot intersects jw axis at -1 so
    innerloop is marginally stable")
7 outerloop=syslin('c',100*(s+0.1)/(s+10)*(s^3+3*s
    ^2+2*s+6))
8 //nyquist(outerloop)
9 show_margins(outerloop,'nyquist')
10 P=0//no of poles on RHP
11 Pw=2//no of poles on jw axis
12 theta=-(Z-P-0.5*Pw)*180
13 Z=0//for outer loop to be stable
14 disp(theta,"theta for stability=")
15 printf("theta as seen from nyquist plot is same as
    that required for stability \n hence outer loop
    is stable")
```

Scilab code Exa 9.14 gain margin and phase margin

```
1 //gain margin and phase margin
2 s=%s;
3 sys=syslin('c',(2500)/(s*(s+5)*(s+50)))
4 nyquist(sys)
5 show_margins(sys,'nyquist')
6 gm=g_margin(sys)
7 pm=p_margin(sys)
8 disp(gm,"gain margin=")
9 disp(pm,"phase margin=")
```

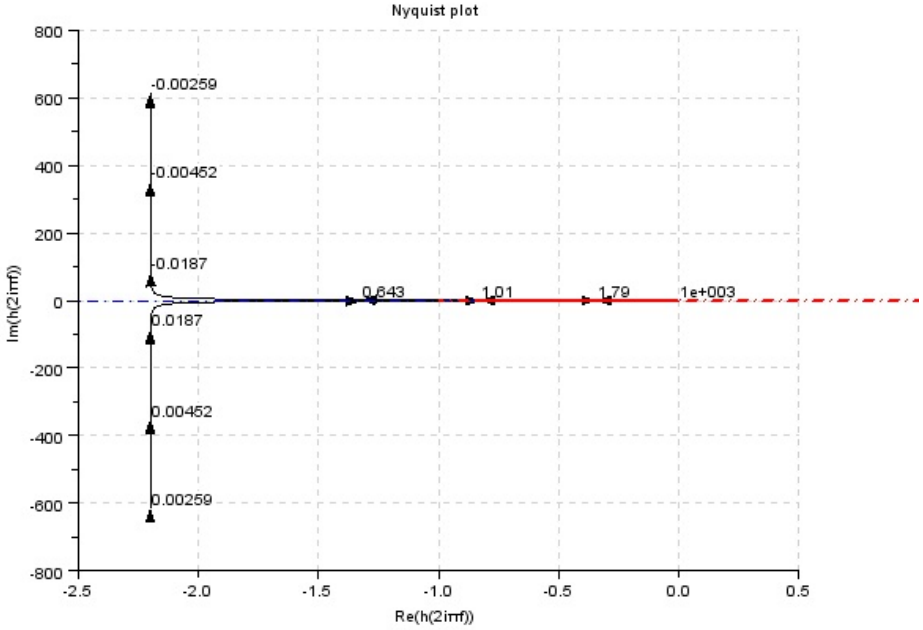

Figure 9.13: gain margin and phase margin

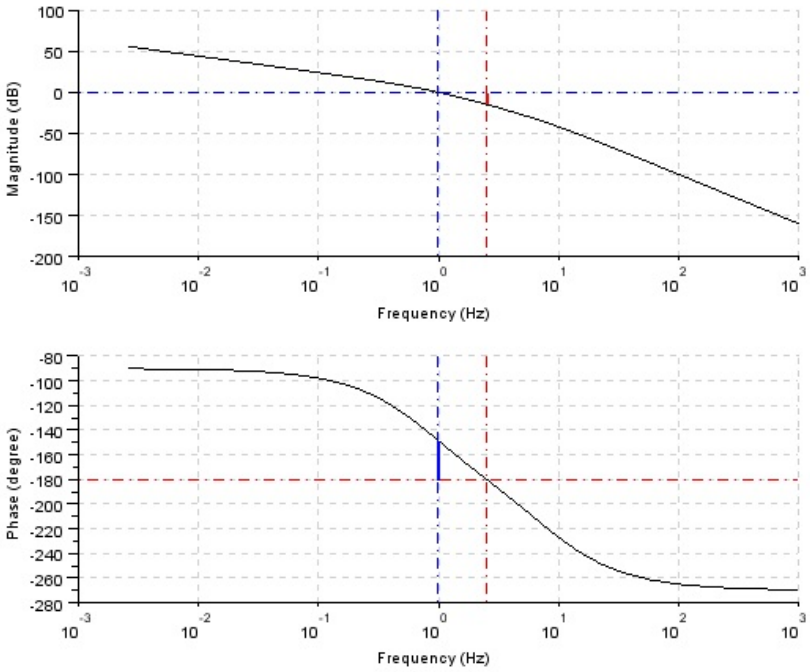

Figure 9.14: bode plot

Scilab code Exa 9.15 bode plot

```
1  //bode plot
2  s=%s;
3  sys=syslin('c',(2500)/(s*(s+5)*(s+50)))
4  bode(sys)
5  show_margins(sys,'bode')
6  gm=g_margin(sys)
7  pm=p_margin(sys)
8  disp(gm,"gain margin=")
9  disp(pm,"phase margin=")
10 if (gm<=0 | pm<=0)
11   printf("system is unstable")
12 else
13   printf("system is stable")
14   end
```

Scilab code Exa 9.17 relative stability

```
1  //relative stability
2  s=%s;
3  sys=syslin('c',(100)*(s+5)*(s+40)/(s^3*(s+100)*(s
     +200)))//K=1
4  bode(sys)
5  show_margins(sys,'bode')
6  gm=g_margin(sys)
7  pm=p_margin(sys)
8  disp(gm,"gain margin=")
9  disp(pm,"phase margin=")
10 if (gm<=0 | pm<=0)
```

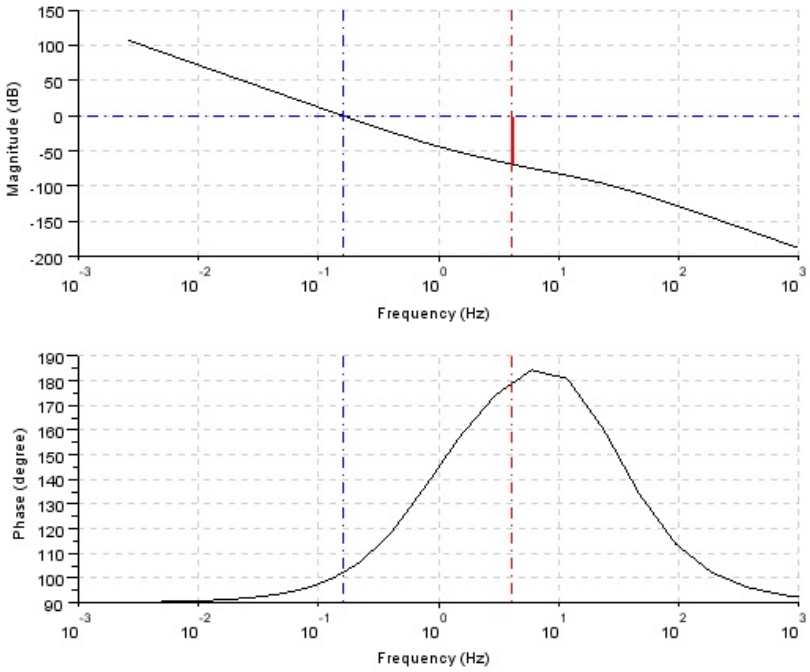

Figure 9.15: relative stability

```
11    printf("system is unstable")
12  else
13    printf("system is stable")
14    end
```