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

July 31, 2019

[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

List of Figures

Chapter 2

Mathematical Foundation

Scilab code Exa 2.1 laplace transform of step function

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

Scilab code Exa 2.2 laplace transform of exponential function

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

Scilab code Exa 2.3 final value thereom

```
//final value thereom
syms s
```

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

Scilab code Exa 2.4 inverse laplace

```
//inverse laplace
syms s
F=1/(s^2+1)          //w=1
disp(F,"F(s)=")
f=ilaplace(F)
disp(f,"f(t)=")
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

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

Scilab code Exa 2.7 `inverse laplace transform`

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

Scilab code Exa 2.8 `inverse laplace transform`

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

Scilab code Exa 2.9 `inverse laplace transform`

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

Scilab code Exa 2.10 determinant of matrix

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

Scilab code Exa 2.12 transpose of matrix

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

Scilab code Exa 2.13 adjoint of matrix

```
//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`

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

Scilab code Exa 2.17 `multiplication of matrices`

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

```
else
    disp("matrices  are not conformable for
       multiplication AB")
end
if D(1,2)==C(1,1) then
    BA=B*A
    disp(BA,"BA=")
else
    disp("matrices  are not conformable for
       multiplication BA")
end
```

Scilab code Exa 2.18 inverse of 2x2 matrix

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

Scilab code Exa 2.19 inverse of 3x3 matrix

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

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

Scilab code Exa 2.20 rank of a matrix

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

Scilab code Exa 2.21 z transform

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

Scilab code Exa 2.22 `z transform`

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

Scilab code Exa 2.23 `z transform`

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

Scilab code Exa 2.25 `final value thereom`

```
//final  value thereom
z=%z
sys=syslin('c',0.792*z^2/((z-1)*(z^2-0.416*z+0.208))
    )
syms z
l=limit(sys*(1-z^-1),z,1)
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

```
//closed loop transfer function matrix
s=%s
G=[1/(s+1) -1/s;2 1/(s+2)]
H=[1 0;0 1]
GH=G*H
disp(GH,"G(s)H(s)=")
I=[1 0;0 1]
x=I+GH
y=inv(x)
M=y*G
disp(M,"M(s)=")
```

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

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

Scilab code Exa 3.4 masons gain formula

```
//masons gain formula applied to SFG in figure 3-8(d
    )
//two forward paths
syms a12 a23 a24 a25 a32 a34 a43 a44 a45
M1=a12*a23*a34*a45
M2=a12*a25
//four loops
L11=a23*a32
L21=a34*a43
L31=a24*a32*a43
L41=a44
//one pair of non touching loops
L12=a23*a32*a44
delta=1-(L11+L21+L31+L41)+(L12)
delta1=1
delta2=1-(L21+L41)
x=(M1*delta1+M2*delta2)/delta
disp(x,"y5/y1=")
//if y2 is output node
M1=a12
delta1=1-(L21+L41)
y=(M1*delta1)/delta
```

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

Scilab code Exa 3.5 masons gain formula

```
//mason's gain formula applied to SFG in figure 3-16
//y2 as output node
syms G1 G2 G3 G4 G5 H1 H2 H3 H4
M1=1
L11=-G1*H1
L21=-G3*H2
L31=G1*G2*G3*-H3
L41=-H4
L12=G1*H1*G3*H2
L22=G1*H1*H4
L32=G3*H2*H4
L42=-G1*G2*G3*H3*H4
L13=-G1*H1*G3*H2*H4
delta=1-(L11+L21+L31+L41)+(L12+L22+L32+L42)+L13
delta1=1-(L21+L41)+(L32)
x=M1*delta1/delta
disp(x,"y2/y1=")
//y4 as output node
M1=G1*G2
delta1=1-(L41)
y=M1*delta1/delta
disp(y,"y4/y1=")
//y6 or y7 as output node
M1=G1*G2*G3*G4
M2=G1*G5
delta1=1
delta2=1-(L21)
z=(M1*delta1+M2*delta2)/delta
disp(z,"y6/y1=y7/y1=")
```

Scilab code Exa 3.6 masons gain formula

```
//mason's gain formula applied to SFG in figure 3-16
//y2 as output node
syms G1 G2 G3 G4 G5 H1 H2 H3 H4
M1=1
L11=-G1*H1
L21=-G3*H2
L31=G1*G2*G3*-H3
L41=-H4
L12=G1*H1*G3*H2
L22=G1*H1*H4
L32=G3*H2*H4
L42=-G1*G2*G3*H3*H4
L13=-G1*H1*G3*H2*H4
delta=1-(L11+L21+L31+L41)+(L12+L22+L32+L42)+L13
delta1=1-(L21+L41)+(L32)
x=M1*delta1/delta
disp(x,"y2/y1=")
//y7 as output node
M1=G1*G2*G3*G4
M2=G1*G5
delta1=1
delta2=1-(L21)
y=(M1*delta1+M2*delta2)/delta
disp(y,"y7/y1=")
z=y/x                  // (y7/y2)=(y7/y1)/(y2/y1)
disp(z,"y7/y2=")
```

Scilab code Exa 3.7 masons gain formula

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

```
//mason's gain formula applied to SFG in figure 3-17
//E as output node
syms G1 G2 G3 G4 H1 H2
M1=1
L11=-G1*G2*H1
L21=-G2*G3*H2
L31=-G1*G2*G3
L41=-G1*G4
L51=-G4*H2
delta=1-(L11+L21+L31+L41+L51)
delta1=1-(L21+L51+L11)
x=M1*delta1/delta
disp(x,"E(s)/R(s)=")
//Y as output node
M1=G1*G2*G3
M2=G1*G4
delta1=1
delta2=1
y=(M1*delta1+M2*delta2)/delta
disp(y,"Y(s)/R(s)=")
```

Scilab code Exa 3.9 masons gain formula

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

Scilab code Exa 3.10 masons gain formula

```
//applying gain formula to state diagram 3−22
//r(t),x1(t) and x2(t) are input nodes
//y(t) is output node
//superposition principle holds good

syms s r x1 x2
//r(t) as input node and y(t) as output node
M1=0
delta1=1
delta=1
a=(M1*delta1)/delta
y1=a*r
disp(y1,"y1(t)=")

//x1(t) as input node and y(t) as output node
M1=1
delta1=1
b=(M1*delta1)/delta
y2=b*x1
disp(y2,"y2(t)=")

//x2(t) as input node and y(t) as output node
M1=0
delta1=1
c=(M1*delta1)/delta
y3=c*x2
disp(y3,"y3(t)=")

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

Scilab code Exa 3.11 masons gain formula

```
//applying gain formula to state diagram in figure
   3-23(b)
//r(t),x1(t),x2(t) and x3(t) are input nodes
//y(t) is output node
//superposition principle holds good

syms s a0 a1 a2 a3 r x1 x2 x3
//r(t) as input node and y(t) as output node
M1=0
delta1=1
L11=-a0*a3
delta=1-(L11)
a=(M1*delta1)/delta
y1=a*r
disp(y1,"y1(t)=")

//x1(t) as input node and y(t) as output node
M1=1
delta1=1
b=(M1*delta1)/delta
y2=b*x1
disp(y2,"y2(t)=")

//x2(t) as input node and y(t) as output node
M1=0
delta1=1
c=(M1*delta1)/delta
y3=c*x2
disp(y3,"y3(t)=")

//x3(t) as input node and y(t) as output node
M1=a0
delta1=1
d=(M1*delta1)/delta
y4=d*x3
disp(y4,"y4(t)=")
```

```

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

Chapter 4

Mathematical Modelling of Physical Systems

Scilab code Exa 4.1 `transfer fnuction of system`

```
//transfer function of the system
//from state diagram in 4-1(b)
//initial conditions are taken as zero
//considering voltage across capacitor as output
syms R L C
s=%s
M1=(1/L)*(s^-1)*(1/C)*(s^-1)
L11=-(s^-1)*(R/L)
delta=1-(L11)
delta1=1
x=M1*delta1/delta
disp(x,"Ec(s)/E(s)=")
//considering current in the circuit as output
M1=(1/L)*(s^-1)
delta1=1
y=M1*delta1/delta
disp(y,"I(s)/E(s)=")
```

Scilab code Exa 4.2 transfer fnuction of electric network

```
//transfer function of electric network
//from state diagram in 4-2(b)
//inital conditions are taken as zero
//considering i1 as output
syms R1 R2 L1 L2 C
s=%s
M1=(1/L1)*(s^-1)
L11=-(s^-1)*(R1/L1)
L21=-(s^-1)*(1/C)*(s^-1)*(1/L1)
L31=-(s^-1)*(1/L2)*(s^-1)*(1/C)
L41=-(s^-1)*(R2/L2)
L12=L11*L31
L22=L11*L41
L32=L21*L41
delta=1-(L11+L21+L31+L41)+(L12+L22+L32)
delta1=1-(L31+L41)
x=M1*delta1/delta
disp(x,"I1(s)/E(s)=")
//considering i2 as output
M1=(1/L1)*(s^-1)*(1/C)*(s^-1)*(1/L2)*(s^-1)
delta1=1
y=M1*delta1/delta
disp(y,"I2(s)/E(s)=")
//considering voltage across capacitor as output
M1=(1/L1)*(s^-1)*(1/C)*(s^-1)
delta1=1-L41
z=M1*delta1/delta
disp(z,"Ec(s)/E(s)=")
```

Scilab code Exa 4.3 gear trains

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

Scilab code Exa 4.4 mass spring system

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

Scilab code Exa 4.5 mass spring system

```
//mass-spring system
//free body diagram and state diagram are drawn as
   shown in figure 4-19(b) and 4-19(c)
//applying gain formula to state diagram
syms K M B
s=%s
//considering y1 as output
M1=(1/M)
L11=-(B/M)*(s^-1)
L21=-(K/M)*(s^-2)
L31=(K/M)*(s^-2)
delta=1-(L11+L21+L31)
delta1=1-(L11+L21)
x=M1*delta1/delta
disp(x,"Y1(s)/F(s)=")
//considering y2 as output
M1=(1/K)*(K/M)*(s^-2)
delta1=1
y=M1*delta1/delta
disp(y,"Y2(s)/F(s)=")
```

Scilab code Exa 4.9 incremental encoder

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

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

Chapter 5

State Variable Analysis

Scilab code Exa 5.1 state transition equation

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

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

Scilab code Exa 5.7 characteristic equation from transfer function

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

Scilab code Exa 5.8 characteristic equation from state equation

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

Scilab code Exa 5.9 eigen values

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

Scilab code Exa 5.12 `ccf form`

```
//OCF form
s=%s
A=[1 2 1;0 1 3;1 1 1]
B=[1;0;1]
[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]
S=[B A*B A^2*B]
disp(S,"controllability matrix=")
if (det(S)==0) then
    printf("system cannot be transformed into ccf
       form")
else
    printf("system can be transformed into ccf form"
       )
end
P=S*M
disp(P,"P=")
Accf=inv(P)*A*P
Bccf=inv(P)*B
disp(Accf,"Accf=")
disp(Bccf,"Bccf=")
```

Scilab code Exa 5.13 `ocf form`

```
//OCF form
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

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

Scilab code Exa 5.19 controllability

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

Scilab code Exa 5.20 controllability

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

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

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

Scilab code Exa 6.2 rouths tabulation to determine stability

```
//rouths tabulations to determine stability
s=%s;
m=s^3-4*s^2+s+6;
disp(m)
r=coeff(m)
n=length(r)
routh=routh_t(m) //This Function generates the Routh
    table
disp(routh,"rouths tabulation=")
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 stable")
  end
```

Scilab code Exa 6.3 rouths tabulation to determine stability

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

```
routh=routh_t(m) //This Function generates the Routh
    table
disp(routh,"rouths tabulation=")
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 stable")
  end
```

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

```
//first element in any row of rouths tabulation is
   zero
s=%s
m=s^4+s^3+2*s^2+2*s+3
r=coeff(m);  //Extracts the coefficient of the
   polynomial
n=length(r);
routh=routh_t(m)
disp(routh,"routh=")
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

```
//elements in one row of rouths tabulations are all
   zero
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
```

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

Scilab code Exa 6.7 determining critical value of K

```
//determining critical value of K
s=%s
syms K
m=s^3+3*K*s^2+(K+2)*s+4
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
routh=[routh;-det(t)/t(2,1),0]
disp(routh,"rouths tabulation=")
routh(3,1)=0  //For marginaly stable system
sys=syslin('c',s*(3*s+1)/(s^3+2*s+4))
k=kpure(sys)
```

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

Scilab code Exa 6.8 stability of closed loop systems

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

Scilab code Exa 6.9 bilinear transformation method

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

```
if (routh(i,1)<0) then
c=c+1
end
end
if (c>=1) then
printf("system is unstable")
else printf("system is stable")
end
```

Scilab code Exa 6.10 `bilinear transformation method`

```
//bilinear transformation method
s=%s
syms K
//p=z^3+z^2+z+K
//substituting z=(1+r)/(1-r) we get
m=(1-K)*s^3+(1+3*K)*s^2+3*(1-K)*s+3+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
routh=[routh;-det(t)/t(2,1),0]
disp(routh,"rouths tabulation=")
```

Chapter 7

Time Domain Analysis of Control Systems

Scilab code Exa 7.1 type of system

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

Scilab code Exa 7.2 steady state errors from open loop tf

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

```
disp(G,"G(s)=")
H=1;
y=G*H;
disp(y,"G(s)H(s)=")
syms s;
Kv=limit(s*y,s,0); //Kv= velocity error coefficient
Ess=1/Kv
//Refering the table 7.1 given in the book ,For type 1 system  Kp=%inf,Ess=0 & Ka=0,Ess=%inf
printf("For type1 system \n step input Kp=inf Ess=0 \n \n parabolic input Ka=0 Ess=inf \n ")
disp(Kv,"ramp input Kv=")
disp(Ess,"Ess=")
//type 2 system
p=poly([1],'s','coeff');
q=poly([0 0 12 1],'s','coeff');
G=p/q;//K=1
disp(G,"G(s)=")
H=1;
y=G*H;
disp(y,"G(s)H(s)=")
Ka=limit(s^2*y,s,0); //Ka= parabolic error coefficient
Ess=1/Ka
//Refering the table 7.1 given in the book ,For type 2 system  Kp=%inf,Ess=0 & Kv=inf,Ess=0
printf("For type2 system \n step input Kp=inf Ess=0 \n ramp input Kv=inf Ess=0 \n ")
disp(Ka,"parabolic input Ka=")
disp(Ess,"Ess=")
//type 2 system
p=poly([5 5],'s','coeff');
q=poly([0 0 60 17 1],'s','coeff');
G=p/q;//K=1
disp(G,"G(s)=")
H=1;
y=G*H;
disp(y,"G(s)H(s)=")
```

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

Scilab code Exa 7.3 steady state errors from closed loop tf

```
//steady state errors from closed loop transfer functions
s=%s
p=poly([3.15 1 0],'s','coeff');//K=1
q=poly([3.15 1.75 2 1],'s','coeff');
M=p/q
disp(M,"M(s)=")
H=1;
R=1;
b=coeff(p)
a=coeff(q)

//step input
if (a(1,1)==b(1,1)) then
    printf("for unit step input Ess=0" )
else
    Ess=1/H*(1-(b(1,1)*H/a(1,1)))*R
    disp(Ess,"for unit step input Ess=")
end

//ramp input
c=0
for i=1:2
```

```
    if(a(1,i)-b(1,i)*H==0) then
        c=c+1
    end
end
if(c==2)
        printf("for unit ramp input Ess=0")
    else if(c==1) then
            Ess=(a(1,2)-b(1,2)*H)/a(1,1)*H
            disp(Ess,"for unit ramp input Ess=")
        else printf("for unit ramp input Ess=inf")
     end
 end

//parabolic input
c=0
for i=1:3
    if(a(1,i)-b(1,i)*H==0) then
        c=c+1
    end
end
if(c==3)
        printf("for unit parabolic input Ess=0")
    else if(c==2) then
            Ess=(a(1,3)-b(1,3)*H)/a(1,1)*H
            disp(Ess,"for unit parabolic input Ess="
               )
        else printf("for unit parabolic input Ess=
           inf")
     end
 end
```

Scilab code Exa 7.4 steady state errors from closed loop tf

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

```
s=%s
p=poly([5 5 0],'s','coeff');
q=poly([5 5 60 17 1],'s','coeff');
M=p/q
disp(M,"M(s)=")
H=1;
R=1;
b=coeff(p)
a=coeff(q)

//step input
if (a(1,1)==b(1,1)) then
    printf("for unit step input Ess=0 \n" )
else
    Ess=1/H*(1-(b(1,1)*H/a(1,1)))*R
    disp(Ess,"for unit step input Ess=")
end

//ramp input
c=0
for i=1:2
    if(a(1,i)-b(1,i)*H==0) then
        c=c+1
    end
end
if(c==2)
        printf("for unit ramp input Ess=0 \n")
    else if(c==1) then
            Ess=(a(1,2)-b(1,2)*H)/a(1,1)*H
            disp(Ess,"for unit ramp input Ess=")
        else printf("for unit ramp input Ess=inf \n"
           )
     end
 end

//parabolic input
c=0
for i=1:3
```

```
    if(a(1,i)-b(1,i)*H==0) then
        c=c+1
    end
end
if(c==3)
        printf("for unit parabolic input Ess=0 \n")
    else if(c==2) then
            Ess=(a(1,3)-b(1,3)*H)/a(1,1)*H
            disp(Ess,"for unit parabolic input Ess="
               )
        else printf("for unit parabolic input Ess=
           inf \n")
     end
 end
```

Scilab code Exa 7.5 steady state errors from closed loop tf

```
//steady state errors from closed loop transfer
   functions
s=%s
p=poly([5 1 0],'s','coeff');
q=poly([5 5 60 17 1],'s','coeff');
M=p/q
disp(M,"M(s)=")
H=1;
R=1;
b=coeff(p)
a=coeff(q)

//step input
if (a(1,1)==b(1,1)) then
    printf("for unit step input Ess=0 \n" )
else
    Ess=1/H*(1-(b(1,1)*H/a(1,1)))*R
    disp(Ess,"for unit step input Ess=")
```

```
end

//ramp input
c=0
for i=1:2
    if(a(1,i)-b(1,i)*H==0) then
        c=c+1
    end
end
if(c==2)
        printf("for unit ramp input Ess=0 \n")
    else if(c==1) then
            Ess=(a(1,2)-b(1,2)*H)/a(1,1)*H
            disp(Ess,"for unit ramp input Ess=")
        else printf("for unit ramp input Ess=inf \n"
           )
     end
 end

//parabolic input
c=0
for i=1:3
    if(a(1,i)-b(1,i)*H==0) then
        c=c+1
    end
end
if(c==3)
        printf("for unit parabolic input Ess=0 \n")
    else if(c==2) then
            Ess=(a(1,3)-b(1,3)*H)/a(1,1)*H
            disp(Ess,"for unit parabolic input Ess="
               )
        else printf("for unit parabolic input Ess=
           inf \n")
     end
 end
```

Scilab code Exa 7.6 steady state errors from closed loop tf

```
//steady state errors from closed loop transfer
   functions
s=%s
p=poly([5 1 0],'s','coeff');
q=poly([10 10 60 17 1],'s','coeff');
M=p/q
disp(M,"M(s)=")
H=2;
R=1;
b=coeff(p)
a=coeff(q)

//step input
if (a(1,1)==b(1,1)) then
    printf("for step input Ess=0 \n" )
else
    Ess=1/H*(1-(b(1,1)*H/a(1,1)))*R
    disp(Ess,"for step input Ess=")
end

//ramp input
c=0
for i=1:2
    if(a(1,i)-b(1,i)*H==0) then
        c=c+1
    end
end
if(c==2)
        printf("for ramp input Ess=0 \n")
    else if(c==1) then
            Ess=(a(1,2)-b(1,2)*H)/a(1,1)*H
            disp(Ess,"for ramp input Ess=")
```

```
        else printf("for ramp input Ess=inf \n")
      end
  end

//parabolic input
c=0
for i=1:3
    if(a(1,i)-b(1,i)*H==0) then
        c=c+1
    end
end
if(c==3)
        printf("for parabolic input Ess=0 \n")
    else if(c==2) then
            Ess=(a(1,3)-b(1,3)*H)/a(1,1)*H
            disp(Ess,"for parabolic input Ess=")
        else printf("for parabolic input Ess=inf \n"
            )
      end
  end
```

Chapter 8

Root Locus Technique

Scilab code Exa 8.1 poles and zeros

```
//poles and zeroes
s=%s
sys=syslin('c',(s+1)/(s*(s+2)*(s+3)))
plzr(sys)
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

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

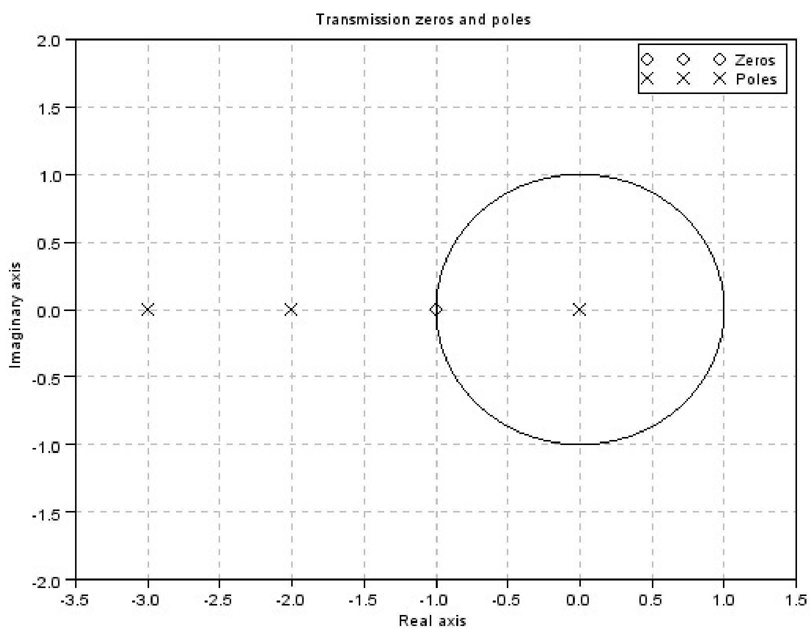

Figure 8.1: poles and zeros

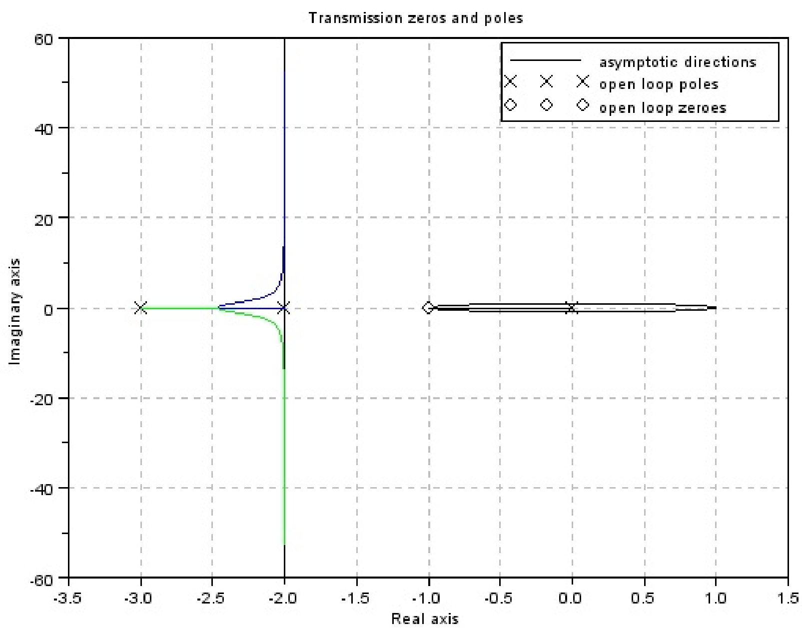

Figure 8.2: root locus

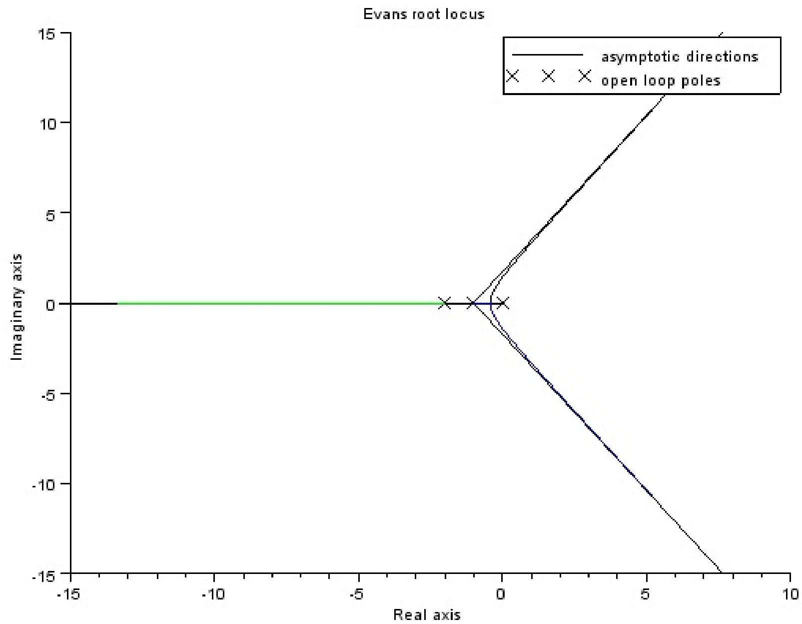

Figure 8.3: root locus

Scilab code Exa 8.3 `root locus`

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

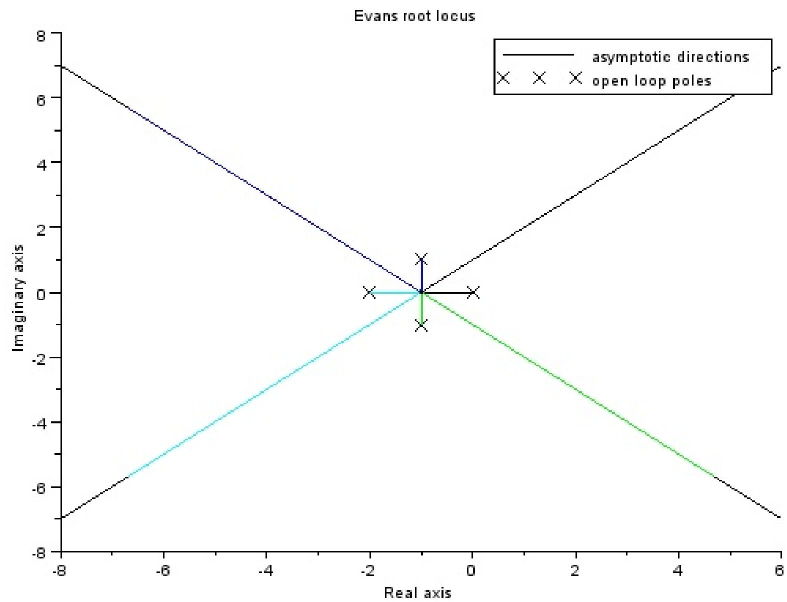

Figure 8.4: root locus

Scilab code Exa 8.4 `root locus`

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

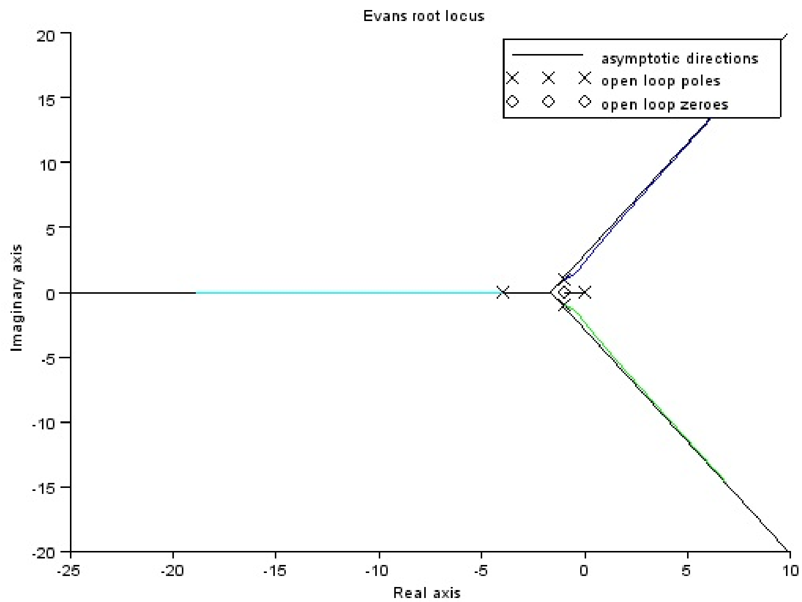

Figure 8.5: root locus

Scilab code Exa 8.5 `root locus`

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

```
//angle of asymptotes
printf("angle of asymptotes of RL")
for i=0:(n-m-1)
    O=((2*i)+1)/(n-m)*180
    disp(O,"q=")
    end
printf("angle of asymptotes of CRL")
for i=0:(n-m-1)
    O=(2*i)/(n-m)*180
    disp(O,"q=")
end
//centroid
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")
C=((0-4-1-1)-(-1))/(n-m);
disp(C,"centroid=")
```

Scilab code Exa 8.8 angle of departure and angle of arrivals

```
//angle of departure and angle of arrivals
s=%s
sys=syslin('c',1/(s*(s+3)*(s^2+2*s+2)))
clf
evans(sys)
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

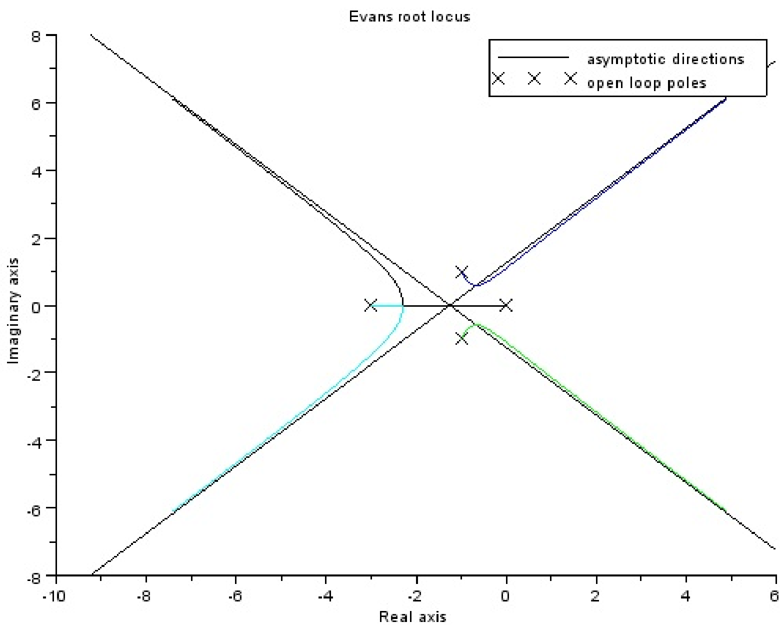

Figure 8.6: angle of departure and angle of arrivals

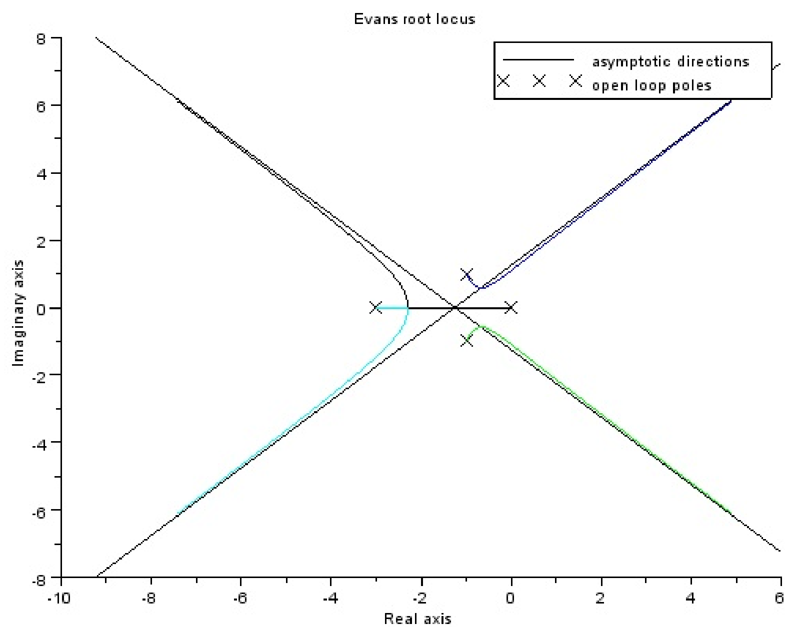

Figure 8.7: intersection of root loci with real axis

```
//multiple order pole
s=%s
sys=syslin('c',(s+3)/(s*(s+2)^3))
clf
evans(sys)
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`

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

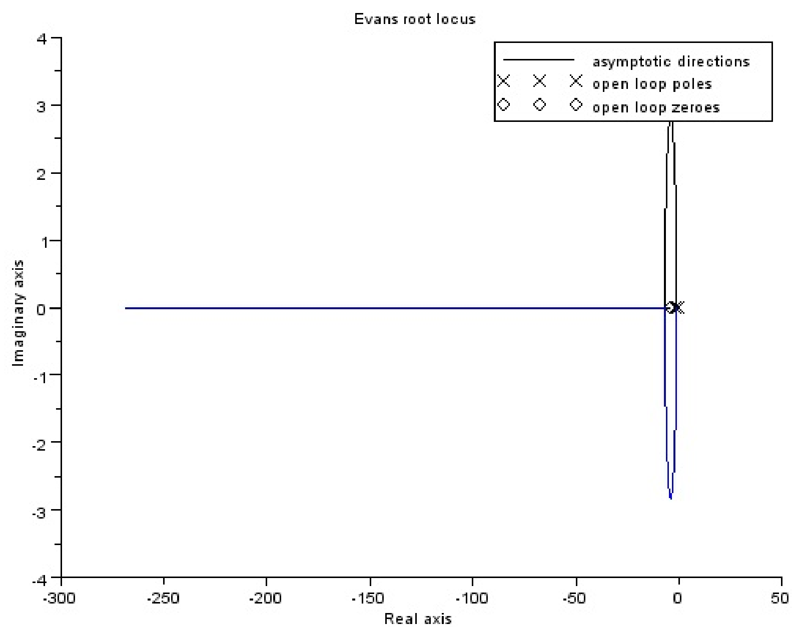

Figure 8.8: breakaway points

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

Scilab code Exa 8.11 `breakaway points`

```
//breakaway points
s=%s
sys=syslin('c',(s+4)/(s*(s+2)))
evans(sys)
syms s
d=derivat(sys)
n=numer(d)
a=roots(n)      //a=breakaway points
disp(a,"breakaway points=")
for i=1:2
    K=-a(i,1)*(a(i,1)+2)/(a(i,1)+4)
    disp(a(i,1),"s=")
    disp(K,"K=")
end
printf("if K is positive breakaway point lies on RL
   or else on CRL")
```

Scilab code Exa 8.12 `breakaway points`

```
//breakaway points
s=%s
sys=syslin('c',(s+2)/(s^2+2*s+2))
evans(sys)
syms s
d=derivat(sys)
n=numer(d)
a=roots(n)      //a=breakaway points
disp(a,"breakaway points=")
for i=1:2
    K=-(a(i,1)^2+2*a(i,1)+2)/(a(i,1)+2)
        disp(a(i,1),"s=")
```

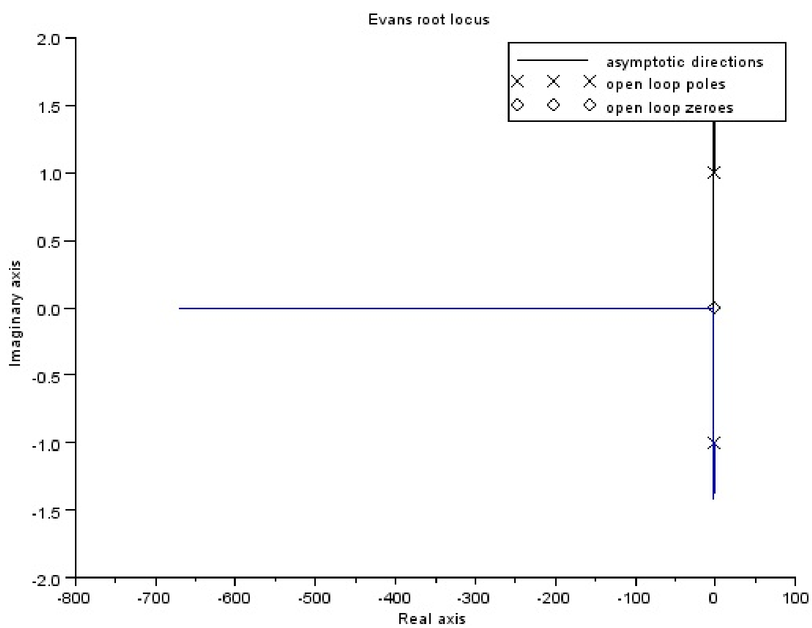

Figure 8.9: breakaway points

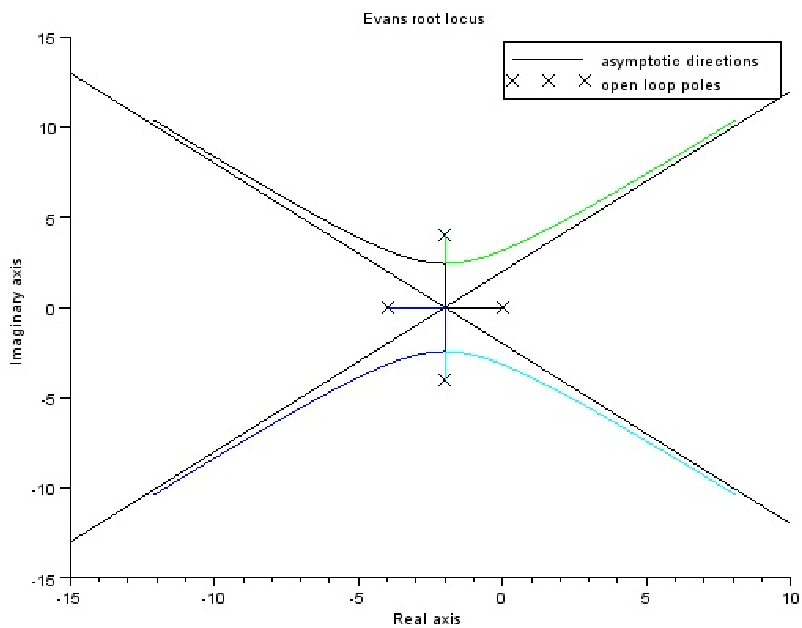

Figure 8.10: breakaway points

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

Scilab code Exa 8.13 breakaway points

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

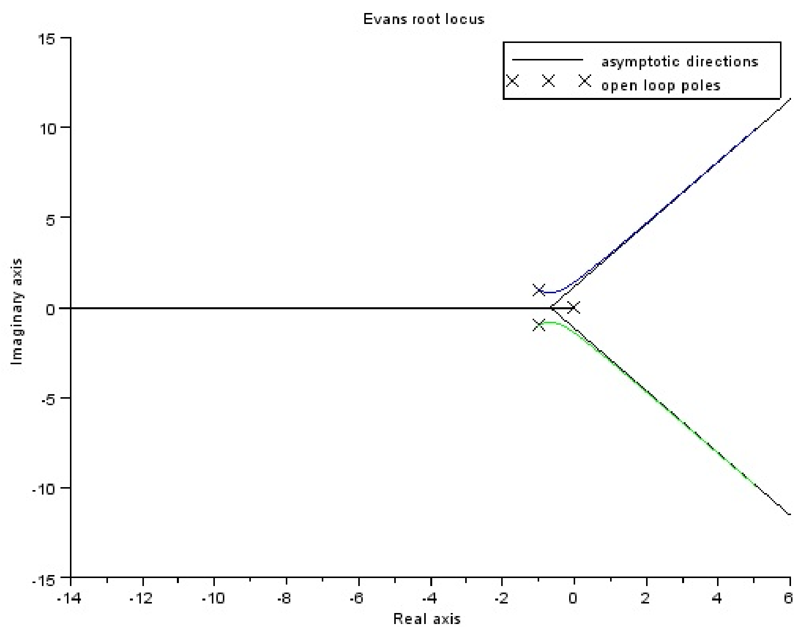

Figure 8.11: breakaway points

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

Scilab code Exa 8.14 breakaway points

```
//breakaway points
s=%s
sys=syslin('c',1/(s*(s^2+2*s+2)))
evans(sys)
syms s
d=derivat(sys)
n=numer(d)
a=roots(n)     //a=breakaway points
disp(a,"breakaway points=")
for i=1:2
    K=-a(i,1)^2+2*a(i,1)+2
        disp(a(i,1),"s=")
        disp(K,"K=")
end
printf("if K is complex then point is not a break
   away point")
```

Scilab code Exa 8.15 root sensitivity

```
//root sensitivity
s=%s
sys1=syslin('c',1/(s*(s+1)))
evans(sys1)

sys2=syslin('c',(s+2)/(s^2*(s+1)^2))
evans(sys2)

printf("root densitivity at breakaway points is
   infinite")
```

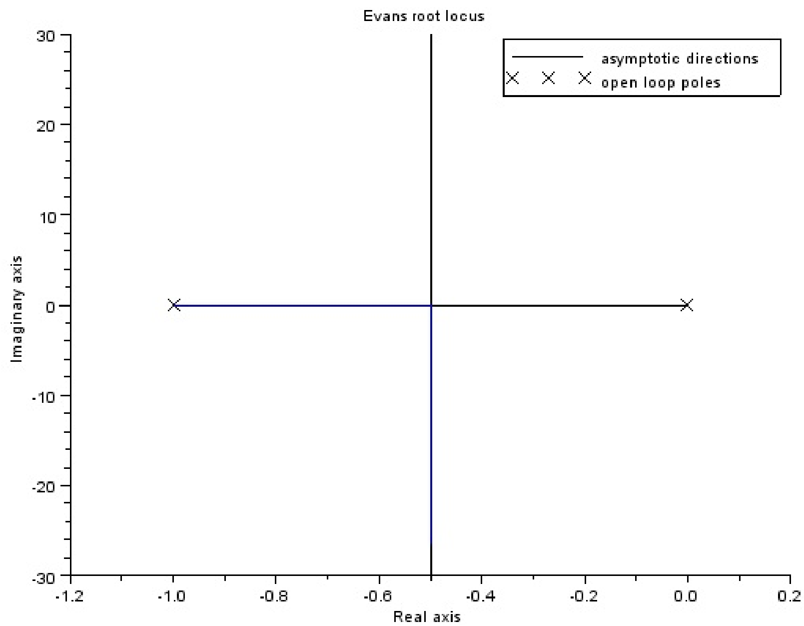

Figure 8.12: root sensitivity

Scilab code Exa 8.16 `calculation of K on root loci`

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

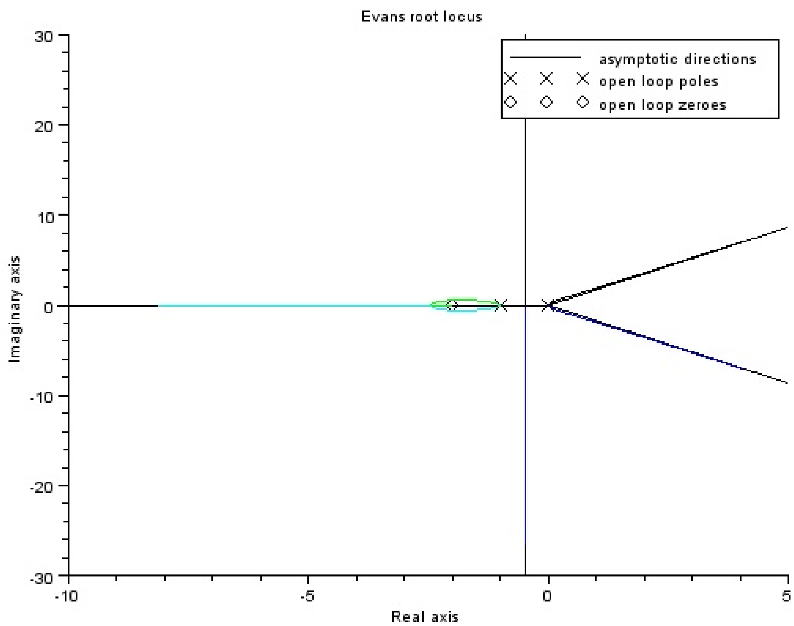

Figure 8.13: root sensitivity

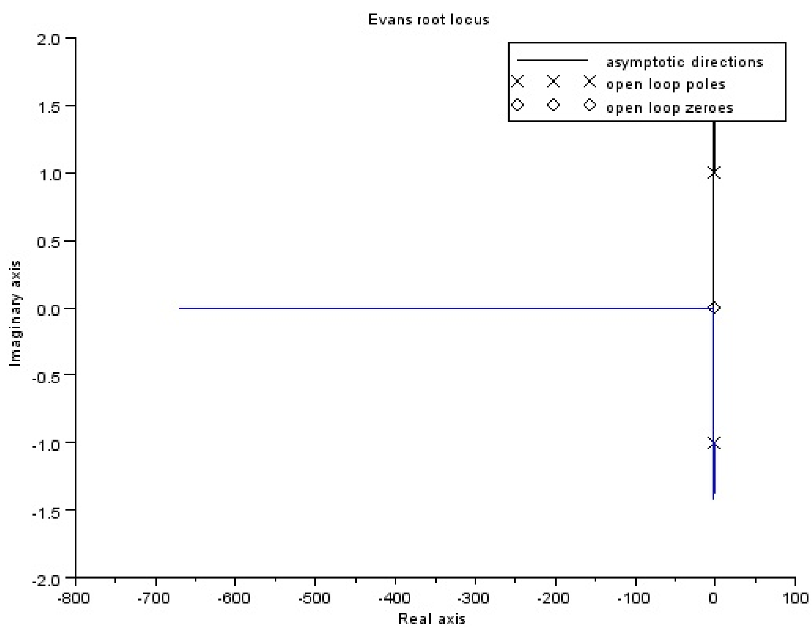

Figure 8.14: calculation of K on root loci

```
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")
A=sqrt((-1)^2+1^2)
B=sqrt((-1)^2+(-1)^2)
C=-2
K=A*B/C
disp(K,"value of K at s=0 is")
```

Scilab code Exa 8.17 `properties of root loci`

```
//properties of root loci
s=%s
sys=syslin('c',(s+3)/(s*(s+5)*(s+6)*(s^2+2*s+2)))
d=denom(sys)
n=numer(sys)
p=roots(d)
z=roots(n)
disp(p,"poles of sys=")
disp(z,"zeroes of sys=")
n=length(p)
m=length(z)
disp(n,"no of poles=")
disp(m,"no of zeroes=")
if (n>m) then
    disp(n,"no of branches of RL=")
else
    disp(m,"no of branches of CRL=")
end
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`

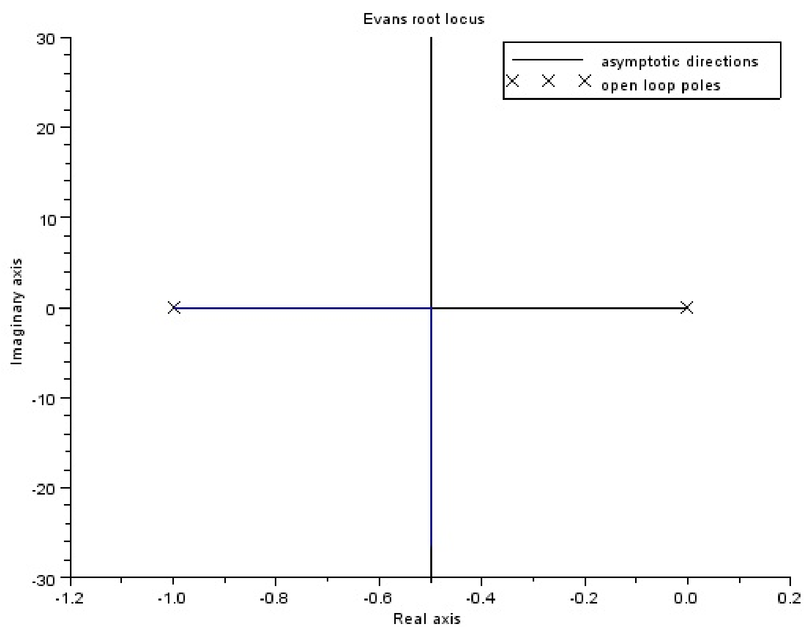

Figure 8.15: effect of addition of poles to system

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

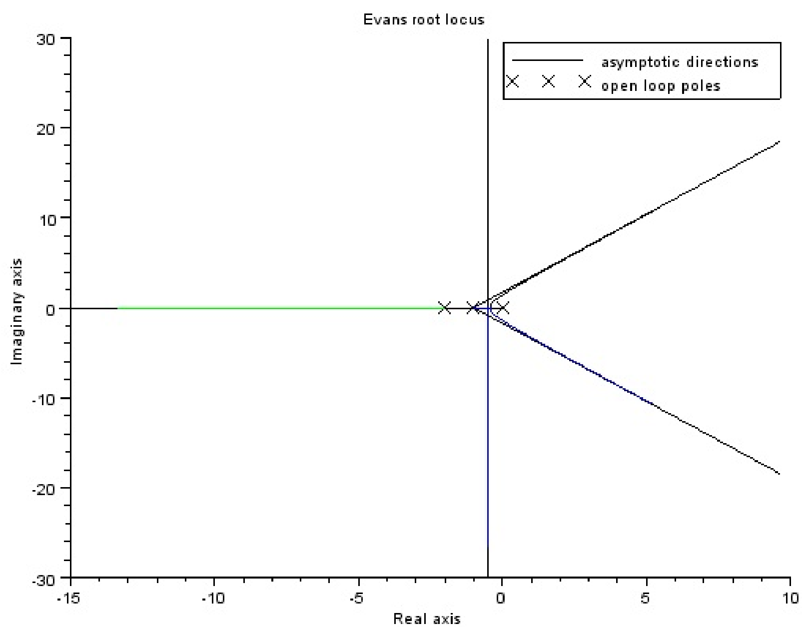

Figure 8.16: effect of addition of poles to system

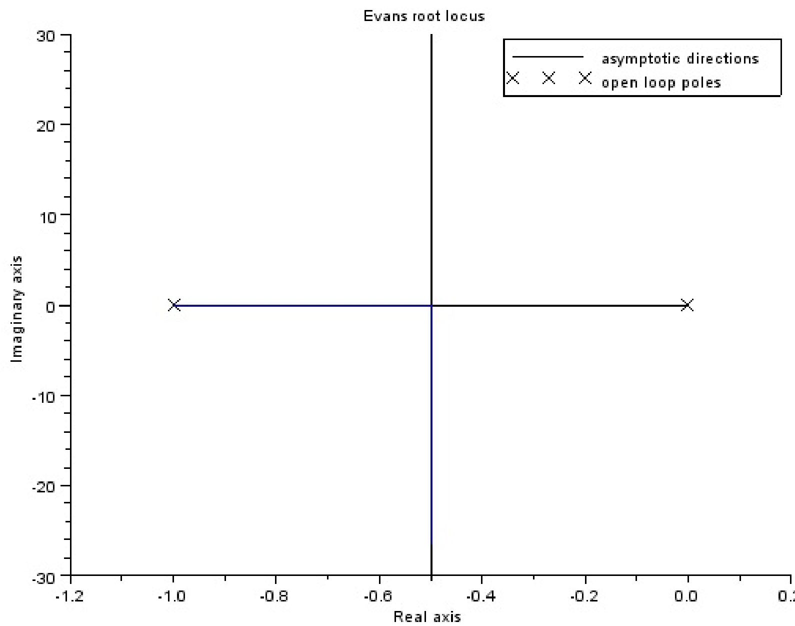

Figure 8.17: effect of addition of zeroes to system

Scilab code Exa 8.19 `effect of addition of zeroes to system`

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

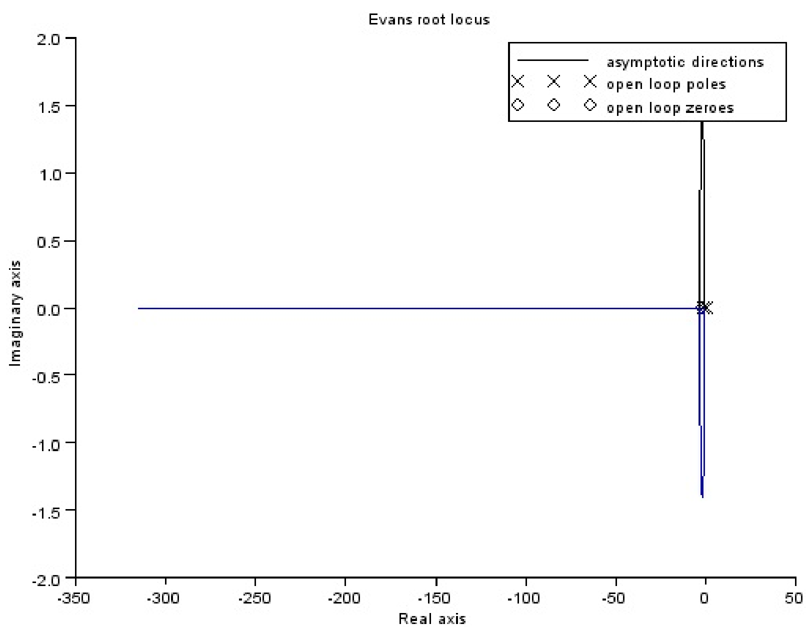

Figure 8.18: effect of addition of zeroes to system

Scilab code Exa 8.20 `effect of moving poles near jw axis`

```
//effect of moving pole near jw axis
s=%s
sys1=syslin('c',(s+1)/(s^2*(s+10)))     //a=10 b=1
evans(sys1)
sys2=syslin('c',(s+1)/(s^2*(s+9)))    //a=9
evans(sys2)
sys3=syslin('c',(s+1)/(s^2*(s+8)))     //a=8
evans(sys3)
sys4=syslin('c',(s+1)/(s^2*(s+3)))    //a=3
evans(sys4)
sys5=syslin('c',(s+1)/(s^2*(s+1)))    //a=1
evans(sys5)
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`

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

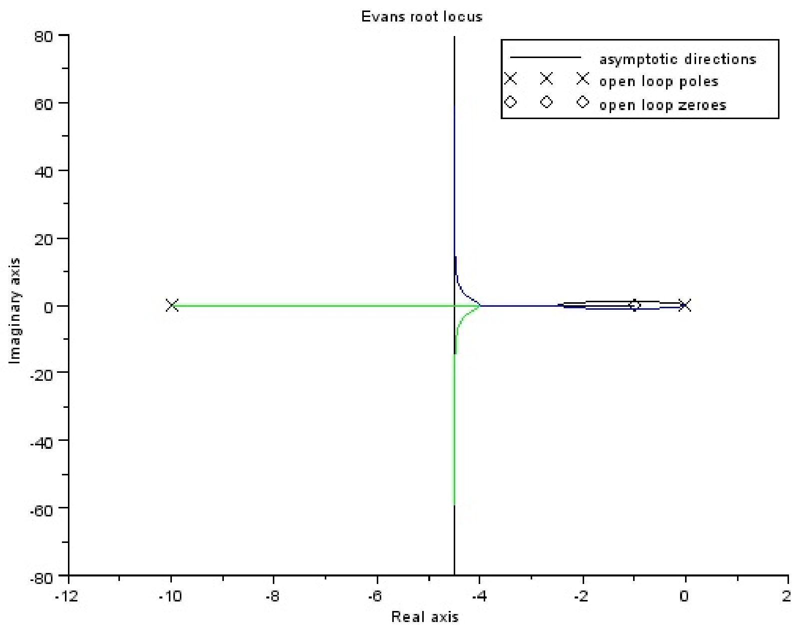

Figure 8.19: effect of moving poles near jw axis

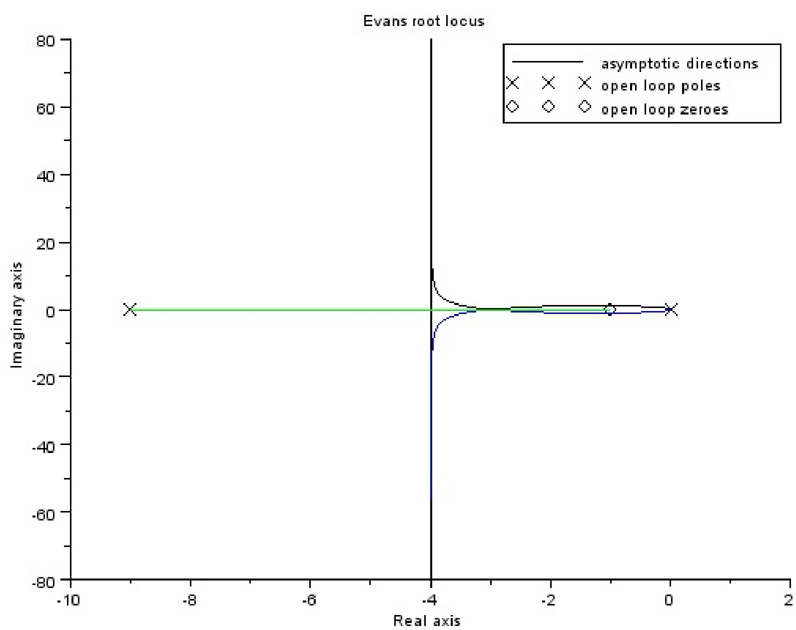

Figure 8.20: effect of moving poles near jw axis

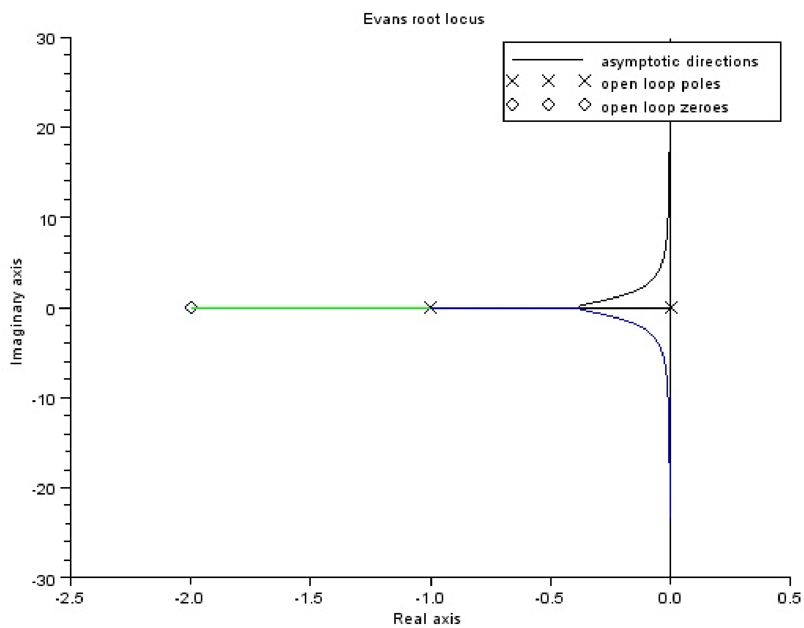

Figure 8.21: effect of moving poles awat from jw axis

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

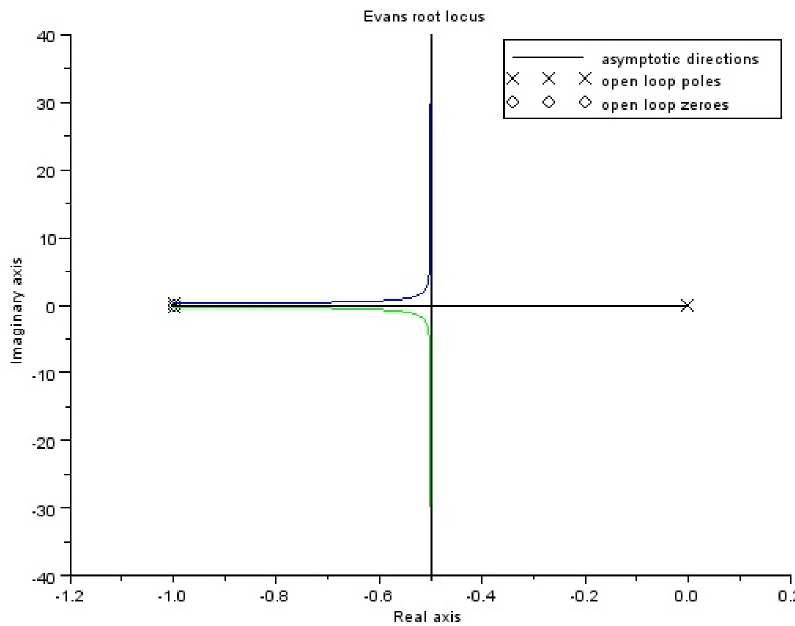

Figure 8.22: effect of moving poles awat from jw axis

Chapter 9

Frequency Domain Analysis

Scilab code Exa 9.1 nyquist plot

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

Scilab code Exa 9.2 nyquist plot

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

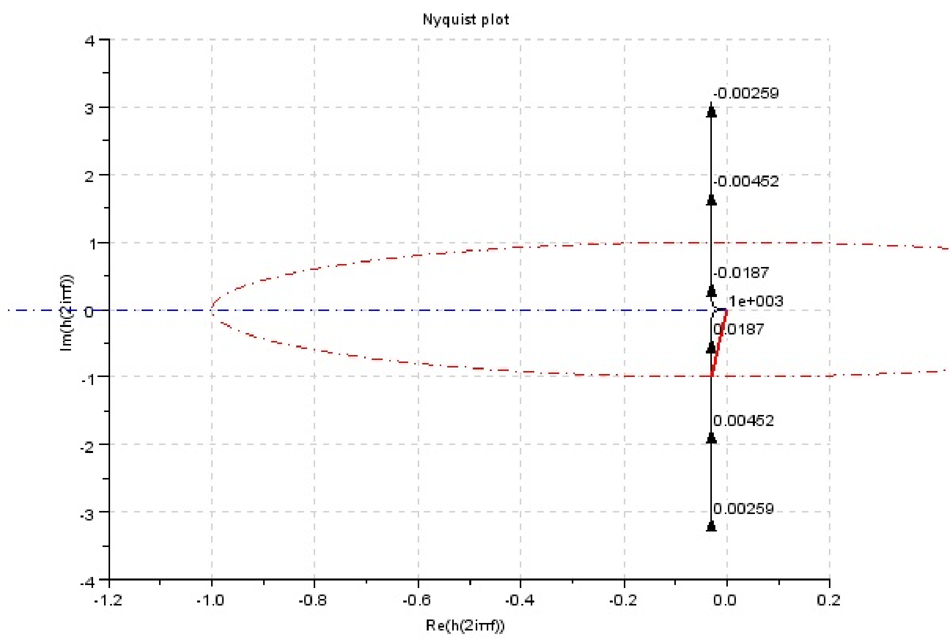

Figure 9.1: nyquist plot

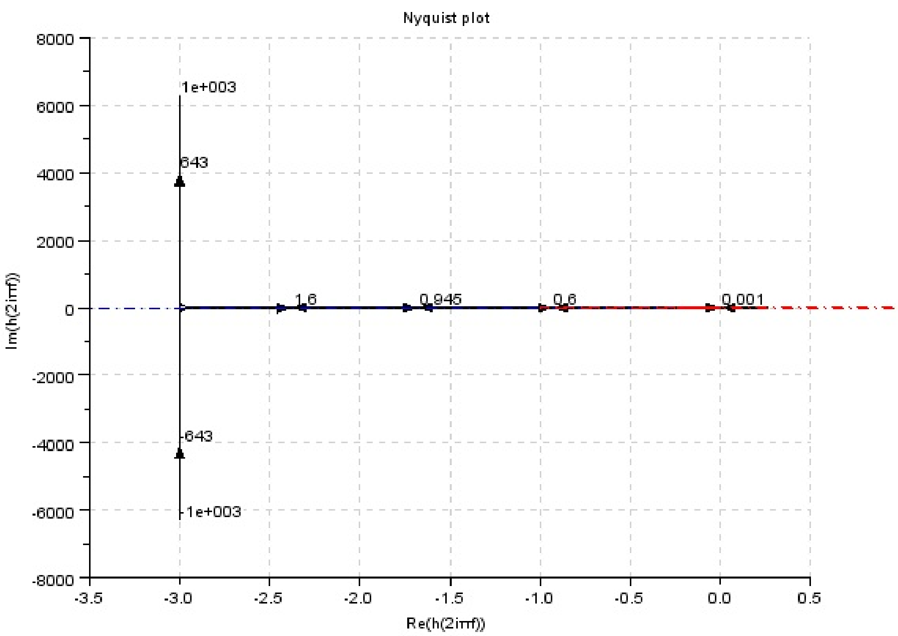

Figure 9.2: nyquist plot

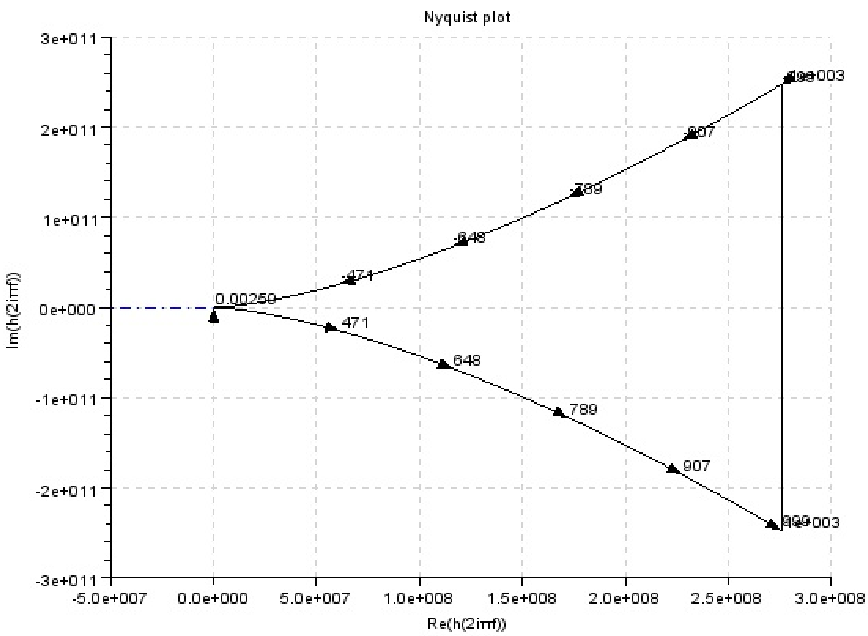

Figure 9.3: stability of non minimum phase loop tf

```
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`

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

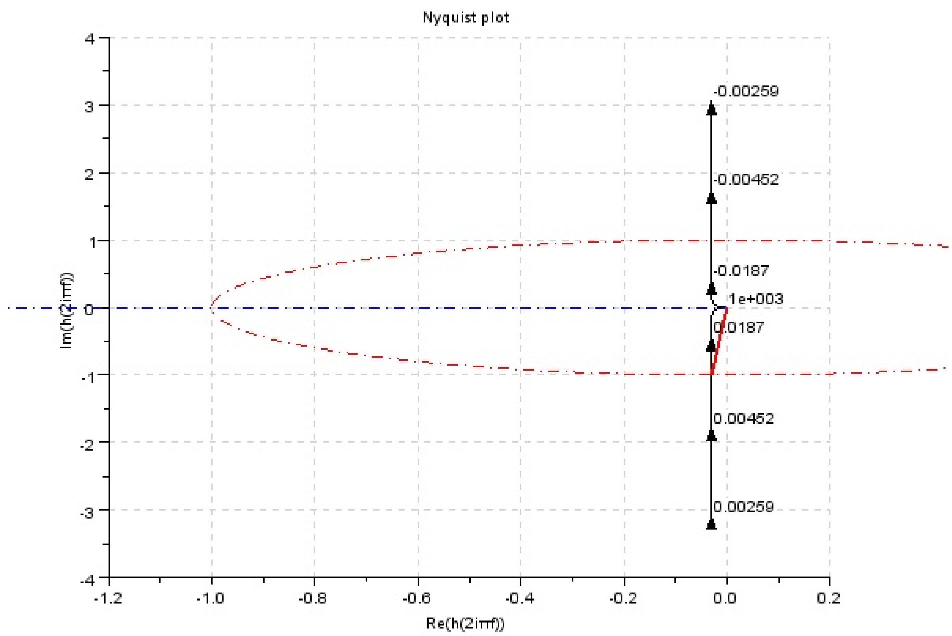

Figure 9.4: stability of minimum phase loop tf

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

Scilab code Exa 9.4 `stability of minimum phase loop tf`

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

Scilab code Exa 9.5 `stability of non minimum phase loop tf`

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

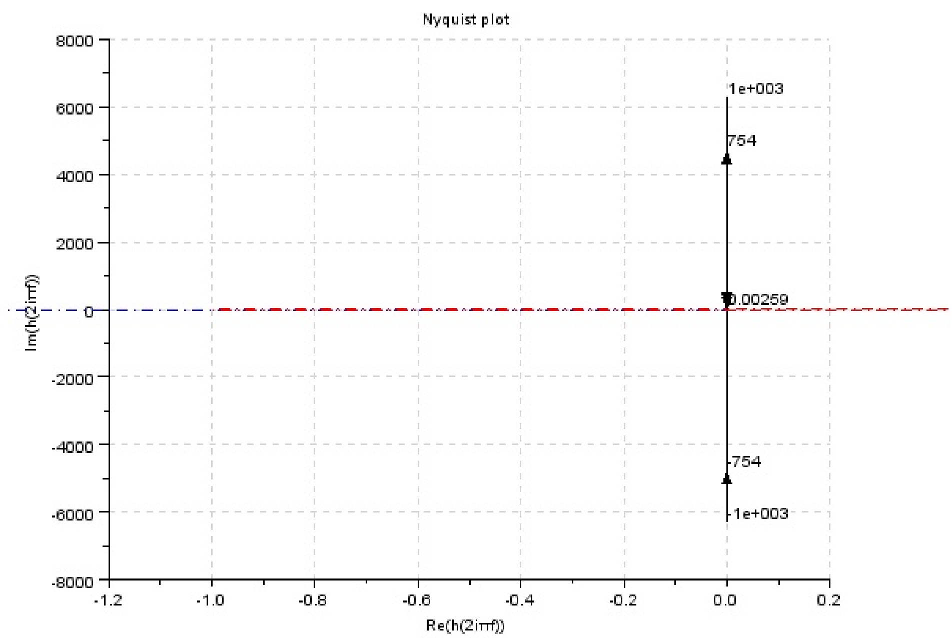

Figure 9.5: stability of non minimum phase loop tf

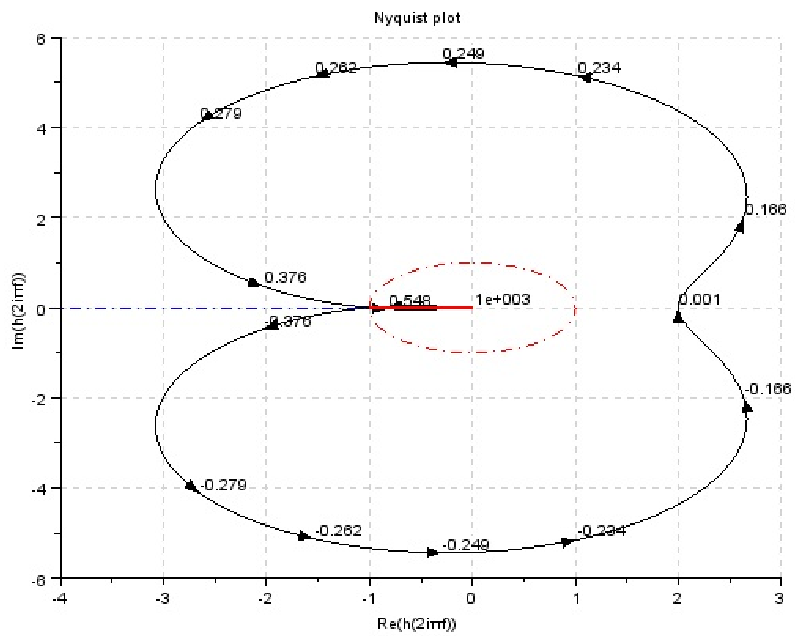

Figure 9.6: stability of non minimum phase loop tf

Scilab code Exa 9.6 stability of non minimum phase loop tf

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

Scilab code Exa 9.7 stability of non minimum phase loop tf

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

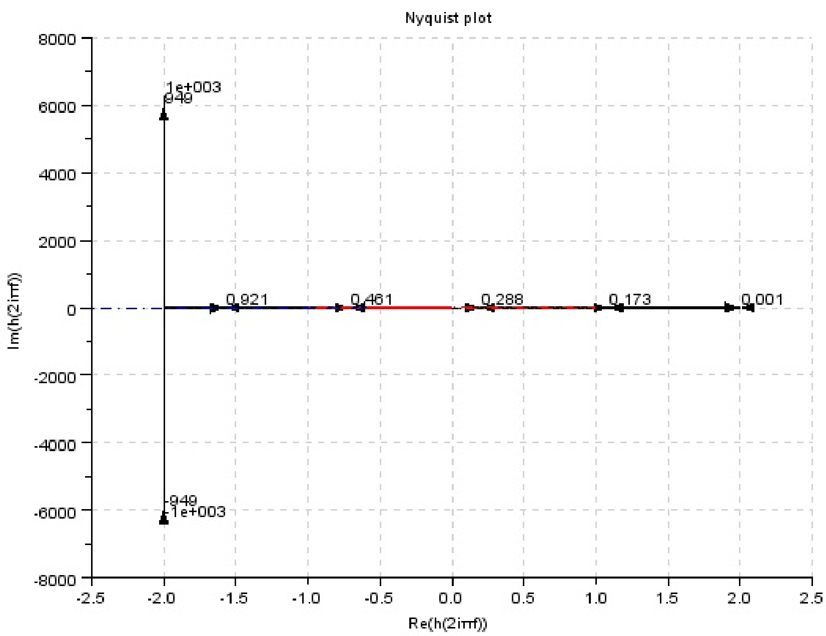

Figure 9.7: stability of non minimum phase loop tf

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

Scilab code Exa 9.8 effect of addition of poles

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

Scilab code Exa 9.9 effect of addition of zeroes

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

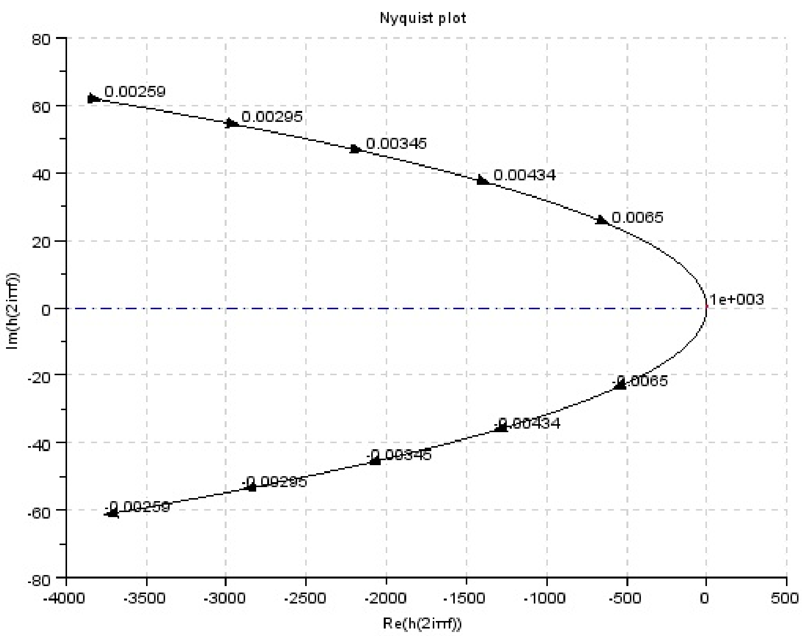

Figure 9.8: effect of addition of poles

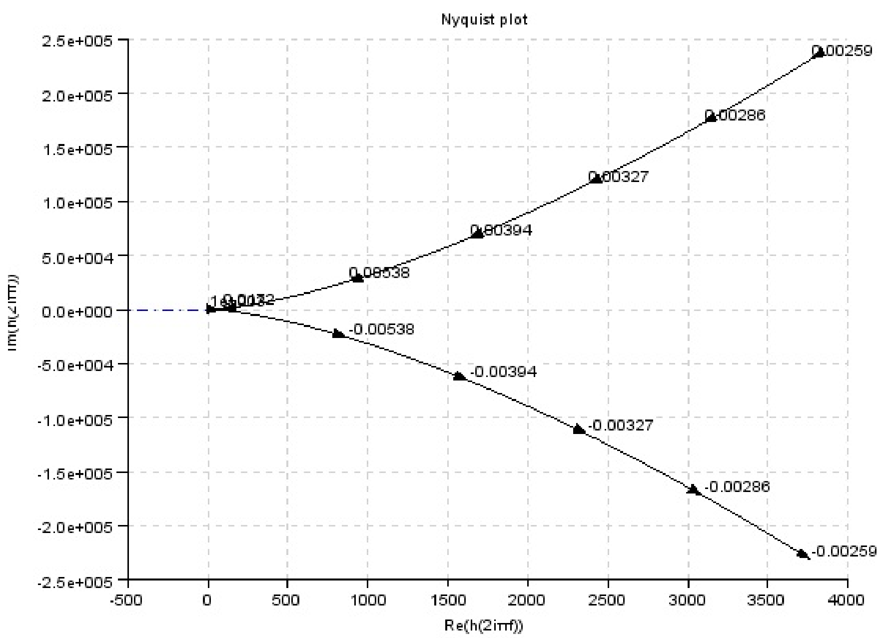

Figure 9.9: effect of addition of poles

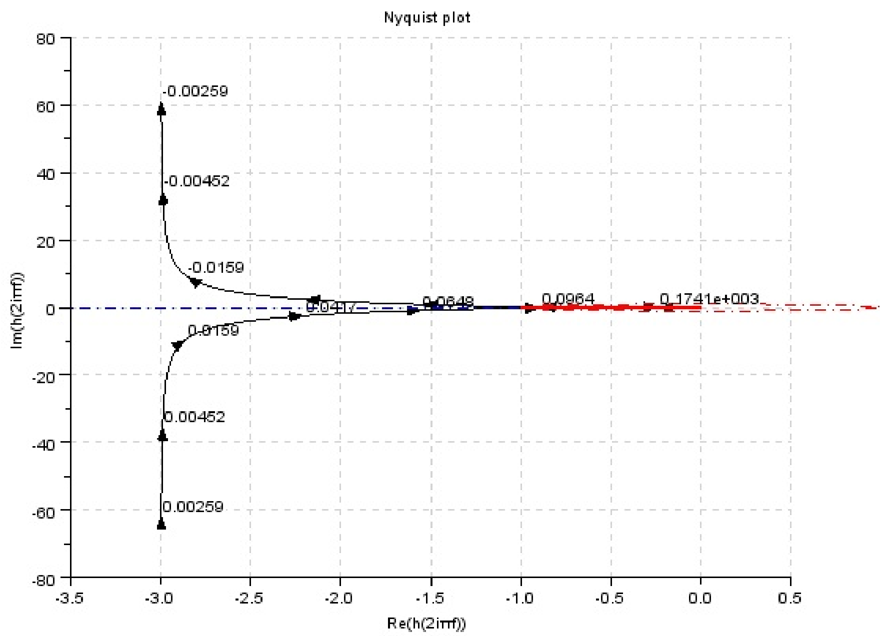

Figure 9.10: effect of addition of zeroes

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

Scilab code Exa 9.10 `multiple loop systems`

```
//multiple loop systems
```

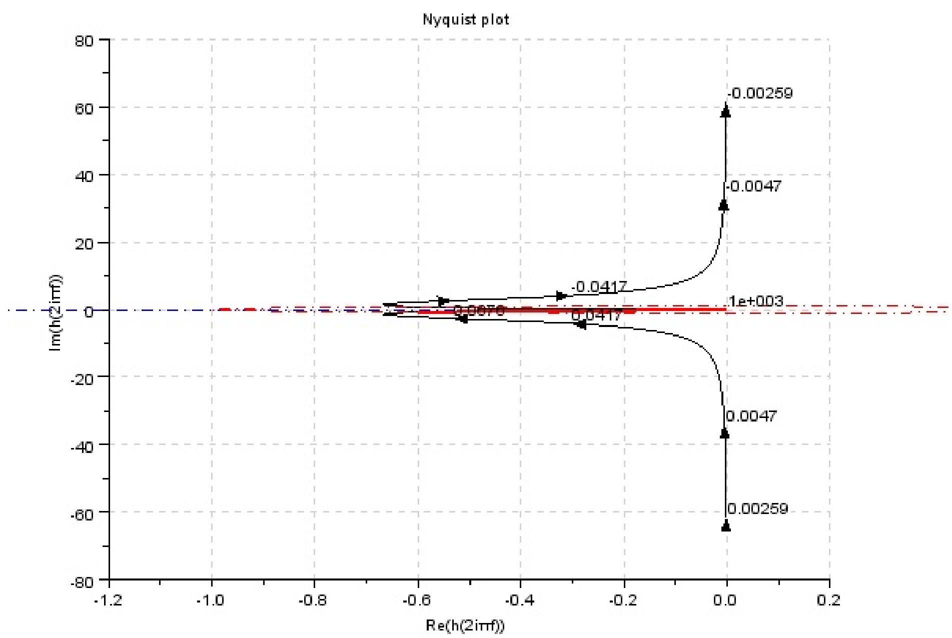

Figure 9.11: effect of addition of zeroes

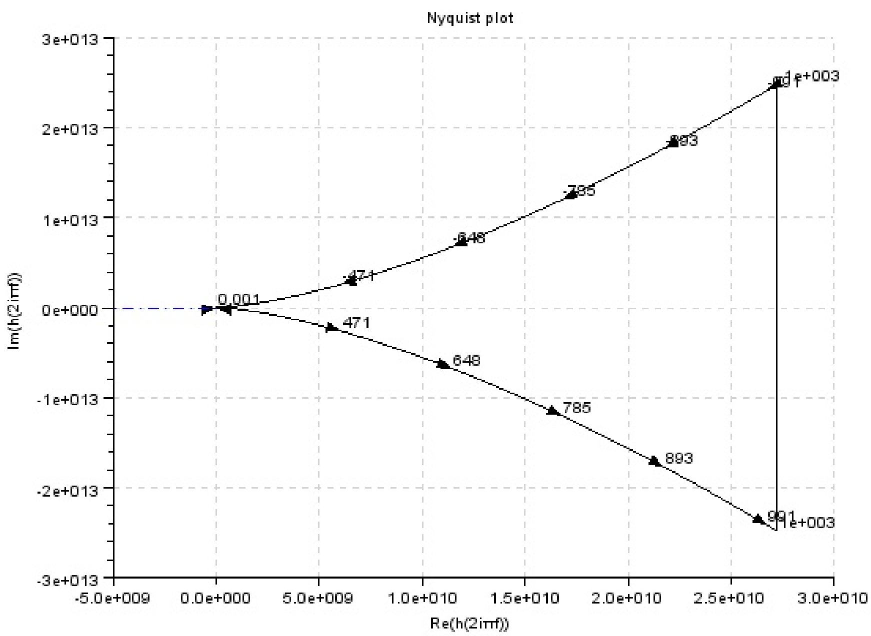

Figure 9.12: multiple loop systems

```
s=%s;
Z = 0;
innerloop=syslin('c',6/s*(s+1)*(s+2))
nyquist(innerloop)
show_margins(innerloop,'nyquist')
printf("nyquist plot intersects jw axis at -1 so
   innerloop is marginally stable")
outerloop=syslin('c',100*(s+0.1)/(s+10)*(s^3+3*s
   ^2+2*s+6))
//nyquist(outerloop)
show_margins(outerloop,'nyquist')
P=0//no of poles on RHP
Pw=2//no of poles on jw axis
theta=-(Z-P-0.5*Pw)*180
Z=0//for outer loop to be stable
disp(theta,"theta for stability=")
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

```
//gain margin and phase margin
s=%s;
sys=syslin('c',(2500)/(s*(s+5)*(s+50)))
nyquist(sys)
show_margins(sys,'nyquist')
gm=g_margin(sys)
pm=p_margin(sys)
disp(gm,"gain margin=")
disp(pm,"phase margin=")
```

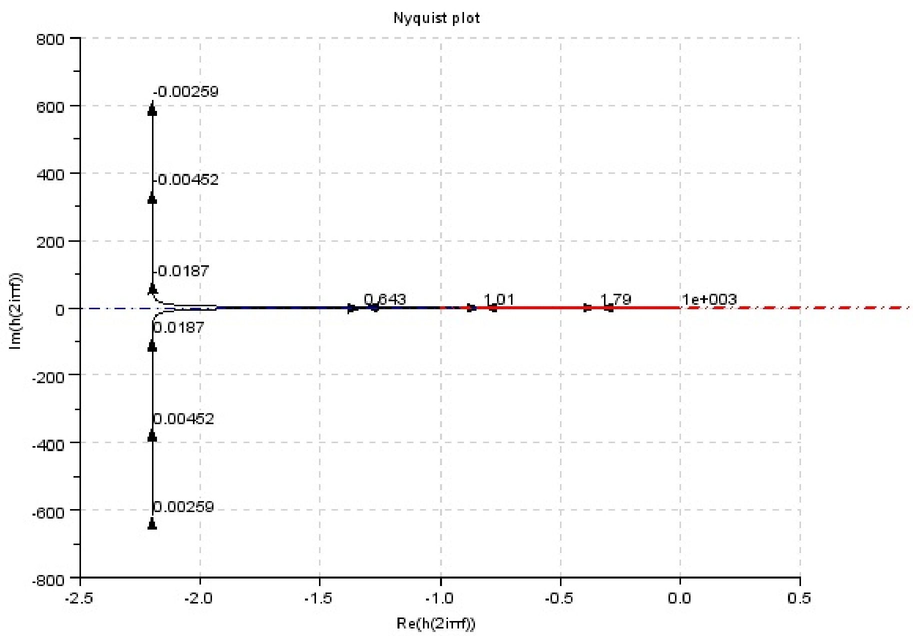

Figure 9.13: gain margin and phase margin

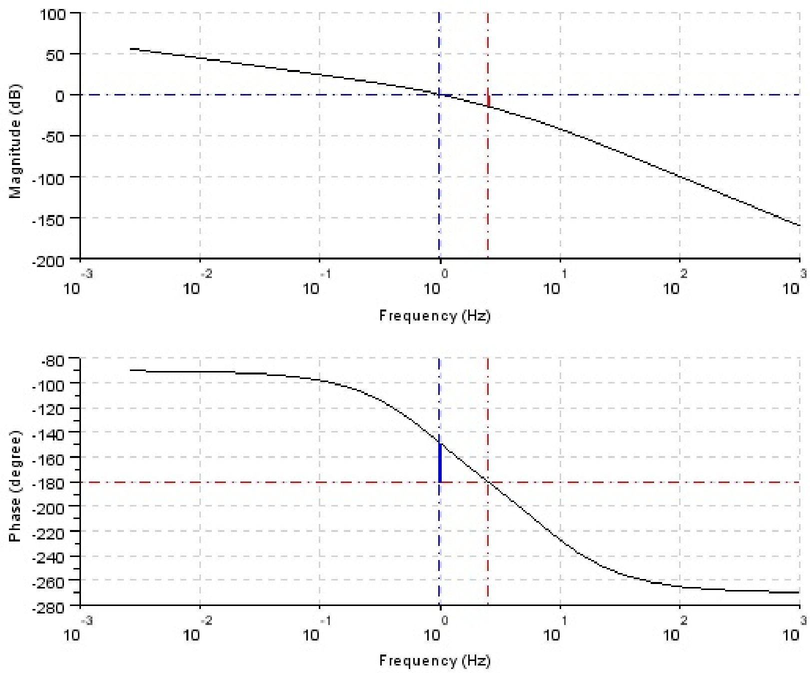

Figure 9.14: bode plot

Scilab code Exa 9.15 `bode plot`

```
//bode plot
s=%s;
sys=syslin('c',(2500)/(s*(s+5)*(s+50)))
bode(sys)
show_margins(sys,'bode')
gm=g_margin(sys)
pm=p_margin(sys)
disp(gm,"gain margin=")
```

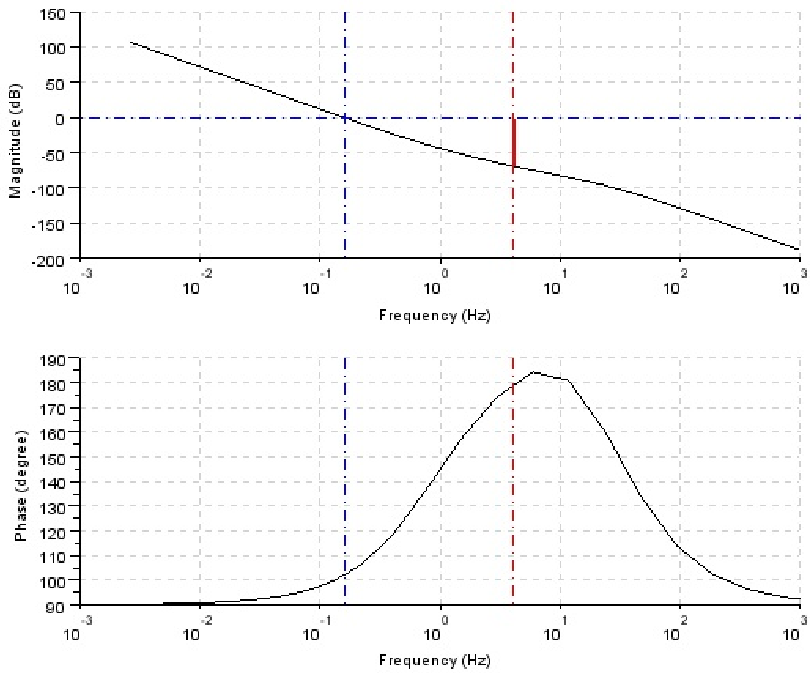

Figure 9.15: relative stability

```
disp(pm,"phase margin=")
if (gm<=0 | pm<=0)
  printf("system is unstable")
else
  printf("system is stable")
  end
```

Scilab code Exa 9.17 `relative stability`

```
//relative stability
s=%s;
```

```
sys=syslin('c',(100)*(s+5)*(s+40)/(s^3*(s+100)*(s+200)))//K=1
bode(sys)
show_margins(sys,'bode')
gm=g_margin(sys)
pm=p_margin(sys)
disp(gm,"gain margin=")
disp(pm,"phase margin=")
if (gm<=0 | pm<=0)
  printf("system is unstable")
else
  printf("system is stable")
  end
```