

Scilab Textbook Companion for
Modern Control Engineering
by K. Ogata¹

Created by
Brian Coutinho
Control Engineering
Electrical Engineering
IIT Rajasthan
College Teacher
Dr. Swagat Kumar
Cross-Checked by
Aditya Sengupta, IIT Bombay

August 12, 2013

¹Funded 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: Modern Control Engineering

Author: K. Ogata

Publisher: Princeton Hall Of India Private Limited, New Delhi

Edition: 5

Year: 2010

ISBN: 978-81-203-4010-7

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 Modelling of Control Systems	14
5 Transient and Steady State Response Analysis	26
6 Control Systems Analysis and Design by Root Locus Method	65
7 Control Systems Analysis and Design by Frequency Response Method	135
8 PID Controllers and Modified PID Controllers	206
9 Control Systems Analysis in State Space	245
10 Control Systems Design in State Space	256

List of Scilab Codes

Exa 2.i.1	Series Parallel Feedback connection of Systems	14
Exa 2.i.2	Transfer Function to State Space Model	15
Exa 2.b.4	Step and Ramp response of different Controllers	16
Exa 2.a.7	Transfer Function to Controllable State Space form	18
Exa 2.a.11	State space to Transfer Function model SISO system	20
Exa 2.a.12	State space to Transfer Function model MIMO system	21
Exa 2.b.14	Verifying linearization of a non linear system	22
Exa 2.4	Convert State space to Transfer Function model	23
Exa 5.a.3	Verifying design to match given response curve	26
Exa 5.a.4	Determining K and k for required step response	28
Exa 5.a.5	Verifying design to match given response	28
Exa 5.a.8	Unit step response and partial fraction expansion	29
Exa 5.a.9	Effect of zeros on step response of a system	32
Exa 5.a.10	Step response characteristics	33
Exa 5.a.11	Step Response for different zeta and wn	34
Exa 5.a.12	Response to unit ramp and exponential input	36
Exa 5.a.13	Response to input r equals 2 plus t	38
Exa 5.a.14	Response to unit acceleration input	40
Exa 5.a.15	Step Responses for different zeta	42
Exa 5.a.16	Response to initial conditions	43
Exa 5.2	Determining K and Kh for required step response	44
Exa 5.3	Step response of MIMO system	46
Exa 5.4	Second order systems with different damping ratio	47
Exa 5.5	Impulse Response of a Second order System	50
Exa 5.6	Unit Ramp response of a second order system	53
Exa 5.7	Response to step and exponential input	55
Exa 5.8	Response to initial condition	56
Exa 5.9	Response to initial conditions using state space	59

Exa 5.10	Response to initial condition using syslin x0	61
Exa 5.12	Constructing Routh array	62
Exa 5.13	Constructing Routh array	64
Exa 6.i.1	Finding the Gain K at any point on the root locus . .	65
Exa 6.i.2	Orthogonality Constant gain curves and Root Locus .	67
Exa 6.i.3	Effect of adding poles or zeros on the root locus . . .	69
Exa 6.a.6	Root locus	72
Exa 6.a.13.1	Lead Compensator Design Attempt 1	73
Exa 6.a.13.2	Lead Compensator Design Attempt 2	77
Exa 6.a.17	Design of lag lead compensator	81
Exa 6.a.18	Design of a compensator for a highly oscillatory system	85
Exa 6.1	Root Locus	89
Exa 6.2	Root Locus	89
Exa 6.3	Root Locus	91
Exa 6.4	Root Locus	94
Exa 6.5	Root locus of system in state space	94
Exa 6.6.1	Design of a lead compensator using root locus	96
Exa 6.6.2	Step and ramp response of lead compensated systems	99
Exa 6.7.1	Design of a lag compensator using root locus	102
Exa 6.7.2	Step and ramp response of lag compensated system .	107
Exa 6.8.1	Design of a lag lead compensator using root locus . .	108
Exa 6.8.2	Evaluating Lag Lead compensated system	111
Exa 6.9.1	Design of lag lead compensator using root locus 2 . .	114
Exa 6.9.2	Evaluating Lag Lead compensated system	117
Exa 6.10	Design of parallel compensation by root locus	123
Exa 6.15	Design of lag compensator	127
Exa 6.16	Design of lag lead compensator	130
Exa 7.a.1	Bode plot	135
Exa 7.i.1	Bode plot for 2nd order systems with varying zeta .	138
Exa 7.a.3	Bode plot for system in state space	139
Exa 7.a.4	Bode plot for different gain K	139
Exa 7.a.8	Stability check	141
Exa 7.a.10	Nyquist Plot with transport lag	143
Exa 7.a.11	Nyquist Plot	143
Exa 7.a.12	Nyquist plot for positive omega	145
Exa 7.a.13	Nyquist plot with points at selected frequencies . .	146
Exa 7.a.14	Nyquist plot for positive and negative feedback . . .	148
Exa 7.a.18	Verifying experimentally derived Transfer function .	150

Exa 7.a.23	Nichols plot	152
Exa 7.1	Steady state sinusoidal output	152
Exa 7.2	Steady state sinusoidal output lag and lead	154
Exa 7.3	Bode Plot in Hz	155
Exa 7.4	Bode Plot with transport lag	158
Exa 7.5	Bode Plot in rad per s	158
Exa 7.6	Bode plot in rad per s	160
Exa 7.7	Bode Plot for a system in State Space	163
Exa 7.8	Polar Plot of a linear system	164
Exa 7.9	Polar Plot with transport lag	166
Exa 7.10	Nyquist Plot	167
Exa 7.11	Nyquist Plot	169
Exa 7.12	Nyquist Plots of system in state space	170
Exa 7.13	Nyquist Plot of MIMO system	172
Exa 7.14	Nyquist Stability Check	173
Exa 7.19	Nyquist plot stability check	176
Exa 7.20	Gain and phase margins for different K	176
Exa 7.21	Stability Margins	180
Exa 7.22	Correlating bandwidth and speed of response	180
Exa 7.23	Frequency characteristics	185
Exa 7.24	Polar and Nichols plot with M circles	185
Exa 7.25	Verifying experimentally derived Transfer function	187
Exa 7.26.1	Design of Lead compensator with Bode plots	190
Exa 7.26.2	Evaluating Lead compensated system	194
Exa 7.27.1	Design of Lag compensator with Bode plots	195
Exa 7.27.2	Evaluating Lag compensated system	199
Exa 7.28.1	Design of Lag lead compensation with Bode plots	200
Exa 7.28.2	Evaluating Lag Lead compensated system	202
Exa 8.i.1	PID Design with Frequency Response	206
Exa 8.a.5	PID design	210
Exa 8.a.6	PID design	211
Exa 8.a.7.1	PID Design with Frequency Response	217
Exa 8.a.12	Computing optimal solution	218
Exa 8.a.13	Design of system with two degrees of freedom	222
Exa 8.1	Tuning a PID controller using Nichols Second Rule	227
Exa 8.2	Computation of Optimal solution 1	231
Exa 8.3	Computation of Optimal solution 2	232
Exa 8.4	Design of system with two degrees of freedom	235

Exa 8.5	Design of system with two degrees of freedom 2	239
Exa 9.b.3	Obtaining canonical form	245
Exa 9.a.5	Conversion from transfer function model to state space model	246
Exa 9.a.16	Controllability and pole zero cancellation	246
Exa 9.a.17	Controllability observability and pole zero cancellation	247
Exa 9.1	Transfer function to controllable observable and jordon canonical forms	249
Exa 9.2	Transformations in state space	249
Exa 9.3	Conversion from state space to transfer function model	250
Exa 9.4	Conversion from state space to transfer function model	251
Exa 9.5	State transition matrix	252
Exa 9.7	Finding e to the power At using laplace transforms . .	253
Exa 9.9	Linear dependence of vectors	254
Exa 9.14	State and ouput controllability and observability . . .	254
Exa 9.15	Observability	255
Exa 10.i.1	Designing a regulator using a minimum order observer	256
Exa 10.i.2	Designing a control system with a minimum order ob- server	263
Exa 10.a.5	Feedback gain for moving eigen values	264
Exa 10.a.6	Gain matrix determination	265
Exa 10.a.9	Transforming to canonical form	266
Exa 10.a.13	Designing a regulator using a minimum order observer	266
Exa 10.a.14	Designing a regulator using a minimum and full order observer	268
Exa 10.a.17	Design of quadratic optimal regulator system and find- ing the response	274
Exa 10.1	Gain matrix using characteristic eq and Ackermanns for- mula	275
Exa 10.2	Gain matrix using ppol and Ackermanns formula . . .	277
Exa 10.3	Response to initial condition	278
Exa 10.4	Design of servo system with integrator in the plant . .	280
Exa 10.5	Design of servo system without integrator in the plant	281
Exa 10.6	Observer Gain matrix using ch eq and Ackermanns for- mula	283
Exa 10.7	Designing a controller using a full order observer . . .	285
Exa 10.8	Designing a controller using a minimum order observer	288
Exa 10.9	Design of quadratic optimal regulator system	289

Exa 10.10	Design of quadratic optimal regulator system	289
Exa 10.11	Design of quadratic optimal regulator system	290
Exa 10.12	Design of quadratic optimal regulator system and finding the response	290
Exa 10.13	Design of quadratic optimal regulator system and finding the response	292
AP 1	Determine Gains and transfer function for minimal order observer	295
AP 2	Plot System Response	296
AP 3	Compute the feedback gain matrix using ackermann's formula	296
AP 4	Transfer function of A,B,C,D.	297
AP 5	Inverse Laplace transform of a rational polynomial in s	297
AP 6	Partial Fraction Residue	298
AP 7	Plot the root locus in a box	299
AP 8	Step response characteristics	299
AP 9	Polar plot of a linear system	300
AP 10	Display gain and phase margins	301
AP 11	Frequency response characteristics	301
AP 12	Gain at a point on a root locus	302

List of Figures

2.1	Step and Ramp response of different Controllers	19
2.2	Verifying linearization of a non linear system	23
2.3	Verifying linearization of a non linear system	24
5.1	Verifying design to match given response curve	27
5.2	Verifying design to match given response	30
5.3	Unit step response and partial fraction expansion	31
5.4	Effect of zeros on step response of a system	33
5.5	Step response characteristics	35
5.6	Step Response for different zeta and wn	37
5.7	Response to unit ramp and exponential input	38
5.8	Response to unit ramp and exponential input	39
5.9	Response to input r equals 2 plus t	41
5.10	Response to unit acceleration input	42
5.11	Step Responses for different zeta	43
5.12	Response to initial conditions	45
5.13	Step response of MIMO system	48
5.14	Step response of MIMO system	49
5.15	Second order systems with different damping ratio	51
5.16	Second order systems with different damping ratio	52
5.17	Impulse Response of a Second order System	54
5.18	Unit Ramp response of a second order system	55
5.19	Response to step and exponential input	57
5.20	Response to step and exponential input	58
5.21	Response to initial condition	60
5.22	Response to initial conditions using state space	61
5.23	Response to initial condition using syslin x0	63
6.1	Finding the Gain K at any point on the root locus	66

6.2	Orthogonality Constant gain curves and Root Locus	68
6.3	Effect of adding poles or zeros on the root locus	70
6.4	Root locus	73
6.5	Lead Compensator Design Attempt 1	75
6.6	Lead Compensator Design Attempt 1	76
6.7	Lead Compensator Design Attempt 2	79
6.8	Lead Compensator Design Attempt 2	80
6.9	Design of lag lead compensator	83
6.10	Design of lag lead compensator	84
6.11	Design of a compensator for a highly oscillatory system . .	87
6.12	Design of a compensator for a highly oscillatory system . .	88
6.13	Root Locus	90
6.14	Root Locus	92
6.15	Root Locus	93
6.16	Root Locus	95
6.17	Root locus of system in state space	97
6.18	Design of a lead compensator using root locus	100
6.19	Design of a lead compensator using root locus	101
6.20	Step and ramp response of lead compensated systems	103
6.21	Step and ramp response of lead compensated systems	104
6.22	Design of a lag compensator using root locus	106
6.23	Step and ramp response of lag compensated system	108
6.24	Design of a lag lead compensator using root locus	111
6.25	Design of a lag lead compensator using root locus	112
6.26	Evaluating Lag Lead compensated system	114
6.27	Evaluating Lag Lead compensated system	115
6.28	Design of lag lead compensator using root locus 2	118
6.29	Design of lag lead compensator using root locus 2	119
6.30	Evaluating Lag Lead compensated system	121
6.31	Evaluating Lag Lead compensated system	122
6.32	Design of parallel compensation by root locus	125
6.33	Design of parallel compensation by root locus	126
6.34	Design of lag compensator	129
6.35	Design of lag compensator	130
6.36	Design of lag lead compensator	133
6.37	Design of lag lead compensator	134
7.1	Bode plot	136

7.2	Bode plot for 2nd order systems with varying zeta	137
7.3	Bode plot for system in state space	140
7.4	Bode plot for different gain K	142
7.5	Nyquist Plot with transport lag	144
7.6	Nyquist Plot	145
7.7	Nyquist plot for positive omega	147
7.8	Nyquist plot with points at selected frequencies	149
7.9	Nyquist plot for positive and negative feedback	150
7.10	Verifying experimentally derived Transfer function	151
7.11	Nichols plot	153
7.12	Steady state sinusoidal output	154
7.13	Steady state sinusoidal output lag and lead	156
7.14	Bode Plot in Hz	157
7.15	Bode Plot with transport lag	159
7.16	Bode Plot in rad per s	161
7.17	Bode Plot in rad per s	162
7.18	Bode plot in rad per s	163
7.19	Bode Plot for a system in State Space	165
7.20	Polar Plot of a linear system	166
7.21	Polar Plot with transport lag	168
7.22	Nyquist Plot	169
7.23	Nyquist Plot	171
7.24	Nyquist Plots of system in state space	172
7.25	Nyquist Plot of MIMO system	174
7.26	Nyquist Stability Check	175
7.27	Nyquist plot stability check	177
7.28	Gain and phase margins for different K	178
7.29	Gain and phase margins for different K	179
7.30	Stability Margins	181
7.31	Correlating bandwidth and speed of response	183
7.32	Correlating bandwidth and speed of response	184
7.33	Frequency characteristics	186
7.34	Polar and Nichols plot with M circles	188
7.35	Polar and Nichols plot with M circles	189
7.36	Verifying experimentally derived Transfer function	190
7.37	Design of Lead compensator with Bode plots	192
7.38	Design of Lead compensator with Bode plots	193
7.39	Evaluating Lead compensated system	195

7.40	Design of Lag compensator with Bode plots	197
7.41	Design of Lag compensator with Bode plots	198
7.42	Evaluating Lag compensated system	200
7.43	Design of Lag lead compensation with Bode plots	202
7.44	Design of Lag lead compensation with Bode plots	203
7.45	Evaluating Lag Lead compensated system	205
8.1	PID Design with Frequency Response	207
8.2	PID Design with Frequency Response	208
8.3	PID design	212
8.4	PID design	213
8.5	PID design	215
8.6	PID design	216
8.7	PID Design with Frequency Response	219
8.8	PID Design with Frequency Response	220
8.9	Computing optimal solution	223
8.10	Design of system with two degrees of freedom	225
8.11	Design of system with two degrees of freedom	226
8.12	Tuning a PID controller using Nichols Second Rule	229
8.13	Tuning a PID controller using Nichols Second Rule	230
8.14	Computation of Optimal solution 1	233
8.15	Computation of Optimal solution 2	236
8.16	Design of system with two degrees of freedom	240
8.17	Design of system with two degrees of freedom	241
8.18	Design of system with two degrees of freedom 2	243
8.19	Design of system with two degrees of freedom 2	244
10.1	Designing a regulator using a minimum order observer . . .	257
10.2	Designing a regulator using a minimum order observer . . .	258
10.3	Designing a control system with a minimum order observer .	261
10.4	Designing a control system with a minimum order observer .	262
10.5	Designing a regulator using a minimum order observer . . .	269
10.6	Designing a regulator using a minimum and full order observer	272
10.7	Designing a regulator using a minimum and full order observer	273
10.8	Design of quadratic optimal regulator system and finding the response	276
10.9	Response to initial condition	279
10.10	Design of servo system with integrator in the plant	281

10.11Design of servo system without integrator in the plant	283
10.12Designing a controller using a full order observer	287
10.13Design of quadratic optimal regulator system and finding the response	292
10.14Design of quadratic optimal regulator system and finding the response	294

Chapter 2

Mathematical Modelling of Control Systems

Scilab code Exa 2.i.1 Series Parallel Feedback connection of Systems

```
1 // Illustration 2.1
2 // Section 2-3 in the book
3 // Demonstrating Series , Parallel and feedback
   connection of Linear Systems
4
5 clear; clc; close;
6
7 // Define Polynomials in variable 's'
8 // Please NOTE : The list of coeficients has to be
   given in
9 //           INCREASING powers of 's',
10
11 n1 = poly( [10] , 's' , 'c');
12 d1 = poly( [10 2 1] , 's' , 'c'); // 10 + 2*s + s^2
13
14 // Alternate method to define transfer functions in
   scilab
15 // using '%s'
16 s = %s;
```

```

17 n2 = 5;
18 d2 = 5 + s;
19
20
21 G1 = syslin('c',n1,d1); // define continuous LTI
   systems systems
22 G2 = syslin('c',n2,d2);
23
24 disp(G1,'G1 ='); disp(G2,'G2 ='); // display variables
   on the screen
25
26 series = G1 * G2;
27 parallel = G1 + G2;
28 feedback = G1 /. G2; // feedback is via G2.
29
30 disp(series,'series =');
31 disp(parallel,'parallel =');
32 disp(feedback,'feedback =');

```

Scilab code Exa 2.i.2 Transfer Function to State Space Model

```

1 // Illustration 2.2
2 // Conversion from transfer function model to state
   space model
3 // Section 2–6 of the Book
4
5 // This example demonstrates that there is no
   unique
6 // state space representation of a transfer
   function.
7
8 clear; clc; close; mode(0);
9 s = %s;
10 num = s;
11 den = 160 + 56*s + 14*s^2 + s^3;

```

```

12 Htf = syslin('c',num,den)
13
14 // There are infinite state space models for the
   same transfer
15 // function. The tf2ss() function will return one of
   them,
16
17 Hss = tf2ss(Htf);
18 ssprint(Hss);           // Print the state space model
19
20 // Alternatively: you can directly get the A,B,C,D
21 [A,B,C,D] = abcd(Htf)
22
23 // To cross check, let us find the transfer function
24 Htf2 = clean(ss2tf(Hss)) // which matches with Htf
25
26 // Now, the form given in text book is called
   controllable
27 // canonical form. It's a special form.
28 // We can directly obtain a linear system in this
   form
29 // using cont_frm(num,den) function
30
31 Hssc = cont_frm(Htf.num,Htf.den)
32 Htfc = clean(ss2tf(Hssc))
33
34 // The same transfer function again

```

Scilab code Exa 2.b.4 Step and Ramp response of different Controllers

```

1 // Exercise B-2-4
2 // Plotting the response of different types of
   controllers
3 // to unit step and unit ramp input.
4

```

```

5 clear; clc; xdel(winsid());
6
7 Kp = 4;      //proportional gain
8 Ki1 = 2;     //integral gain
9 Td = 0.8;    //differential time
10 Ti = 2;     //integral time
11 Ki2 = Kp / Ti;
12
13 s = %s;
14 Gi = syslin('c',Ki1/s);
15
16 t = 0:0.05:3;
17 ramp = t;
18 subplot(3,2,1);
19 p1 = Kp * ones(1,length(t));
20 p2 = Kp * t;
21 plot2d(t ,p1 , style=2);
22 plot2d(t ,p2 , style=3);
23 xtitle('Proportional control','t','y');
24 legend('step input','ramp input');
25 xgrid(color('gray'));
26
27 subplot(3,2,2);
28 i1 = csim("step",t, Gi);
29 i2 = csim(ramp,t, Gi);
30 plot2d(t ,i1, style=2);
31 plot2d(t ,i2, style=3) ;
32 xtitle('Integral control','t','y');
33 xgrid(color('gray'));
34 i1 = i1 * Ki2 / Ki1; //change of gain
35 i2 = i2 * Ki2 / Ki1;
36
37
38 subplot(3,2,3);
39 plot2d(t ,p1 + i1, style=2);
40 plot2d(t ,p2 + i2, style=3);
41 xtitle('Proportional integral control','t','y');
42 xgrid(color('gray'));

```

```

43
44 subplot(3,2,4);
45 pd1 = p1;
46 pd2 = p2 + Kp*Td*ones(1,length(t)); //derivative
    term
47 plot2d(t ,pd1 , style=2);
48 plot2d(t ,pd2 , style=3);
49 xtitle('Proportional plus derivative control ','t','y');
50 xgrid(color('gray'));
51
52 subplot(3,2,5);
53 plot2d(t ,pd1 + i1 , style=2);
54 plot2d(t ,pd2 + i2 , style=3,leg='ramp input') ;
55 xtitle('P.I.D. control ','t','y');
56 xgrid(color('gray'));

```

Scilab code Exa 2.a.7 Transfer Function to Controllable State Space form

```

1 // Example A-2-7
2 // Transfer function to controllable form (state
space)
3
4 clear; clc;close;mode(0);
5
6 s = %s;
7 Num = 2*s^3 + s^2 + s + 2; n = coeff(Num);
8 Den = s^3 + 4*s^2 + 5*s + 2; d = coeff(Den);
9 for i = 1:4 ; b(i) = n(5 - i); a(i) = d(5 - i); end
10
11 // Method 1
12 _beta(1) = b(1);
13 _beta(2) = b(2) - a(2)*_beta(1);

```

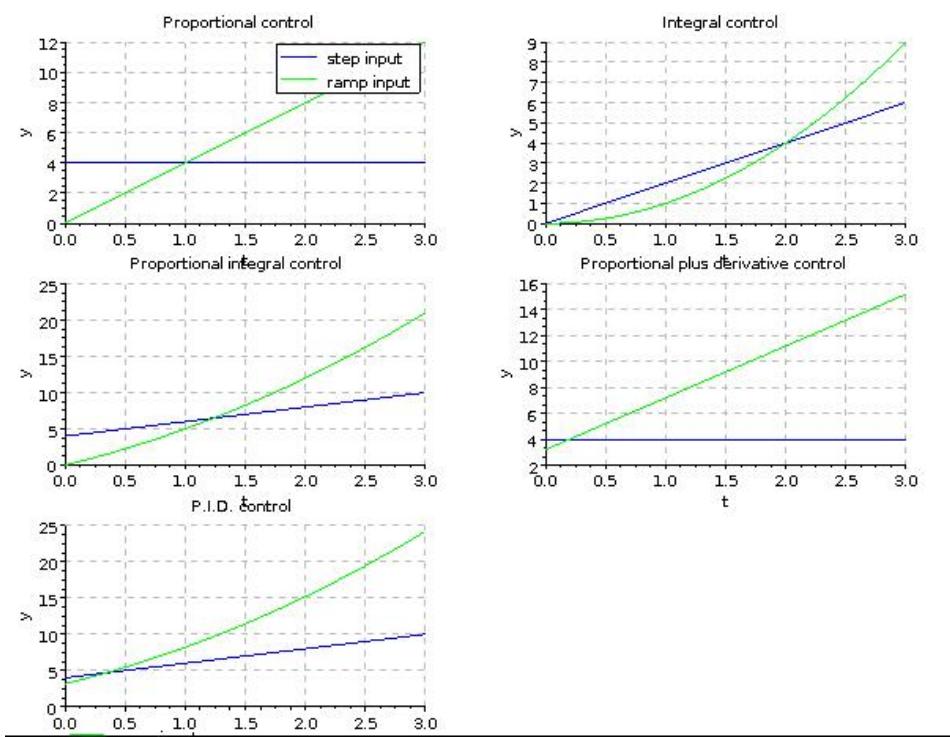


Figure 2.1: Step and Ramp response of different Controllers

```

14 _beta(3) = b(3) - a(2)*_beta(2) - a(3)*_beta(1);
15 _beta(4) = b(4) - a(2)*_beta(3) - a(3)*_beta(2) - a
    (4)*_beta(1);
16
17 A = [0 1 0; 0 0 1; -d(1:3)]
18 B = _beta(2:4)
19 C = [1 0 0]
20 D = b(1)
21
22 // method 2
23 H2 = cont_frm(Num,Den)

```

Scilab code Exa 2.a.11 State space to Transfer Function model SISO system

```

1 // Example A-2-11
2 // Conversion from state space model to transfer
   function model
3 // for a Single Input Single Output System
4
5 clear; clc; close;
6
7 // Please edit the path below
8 // cd "/your code directory /";
9 // exec("transferf.sci");
10
11 A = [-1 1 0; 0 -1 1; 0 0 -2];
12 B = [0; 0; 1];
13 C = [1 0 0];
14 D = [0];
15
16 Htf = transferf(A,B,C,D);           // Htf is the
   tranfer function
17 disp(Htf, 'Htf =');                 // polynomial. ie .
   Htf = num / den

```

check Appendix AP 4 for dependency:

transferf.sci

Scilab code Exa 2.a.12 State space to Transfer Function model MIMO system

```
1 // Example A-2-12
2 // Conversion from state space model to transfer
   function model
3 //           for a multiple input multiple output
   system
4
5 clear; clc; close;
6
7 // Please edit the path below
8 // cd "/your code directory /";
9 // exec(" transferf.sci ");
10
11 A = [0 1; -25 -4];
12 B = [1 1; 0 1];
13 C = [1 0; 0 1];
14 D = [0 0; 0 0];
15
16 Htf = transferf(A,B,C,D)      // Htf is the tranfer
   function matrix,
17 disp(Htf, 'Htf =')           // with four transfer
   functions -
18                                // Htf(1,1), Htf(1,2),
   Htf(2,1), Htf(2,2);
```

check Appendix AP 4 for dependency:

transferf.sci

Scilab code Exa 2.b.14 Verifying linearization of a non linear system

```
1 // Exercise B-2-14
2
3 // An illustration on Linearization
4 // Linearize the function y = f(x) = 0.2*x^3 at x=2
5 // SOLUTION : y = 2.4*x - 3.2
6
7 // Let us observe graphically the linear
   approximation
8 // and the error , and percentage error
9
10 clear; clc; xdel(winsid());
11
12 x = 0.05:0.05:5;
13 y = 0.2 * x .^ 3;
14
15 yl = 2.4 * x - 3.2 ;           // this is not a linear
   system!
16 err = abs(y - yl);             // Error in approximation
17 errpc = err ./ y * 100;       // Percentage error
18
19 subplot(2,1,1);
20 plot2d(x,y,style=2);
21 plot2d(x,yl,style=3,leg="linearized system");
22 xtitle('Original and linearized system','x','y');
23
24 subplot(2,1,2);
25 plot2d(x,err,style=5);
26 xtitle('Error','x','error');
27
28 scf();
29 plot2d(x,errpc,style=5,rect=[1 0 3 100]);
30 plot2d(x, 10 * ones(1,length(x)) ,style=2,leg="10%
```

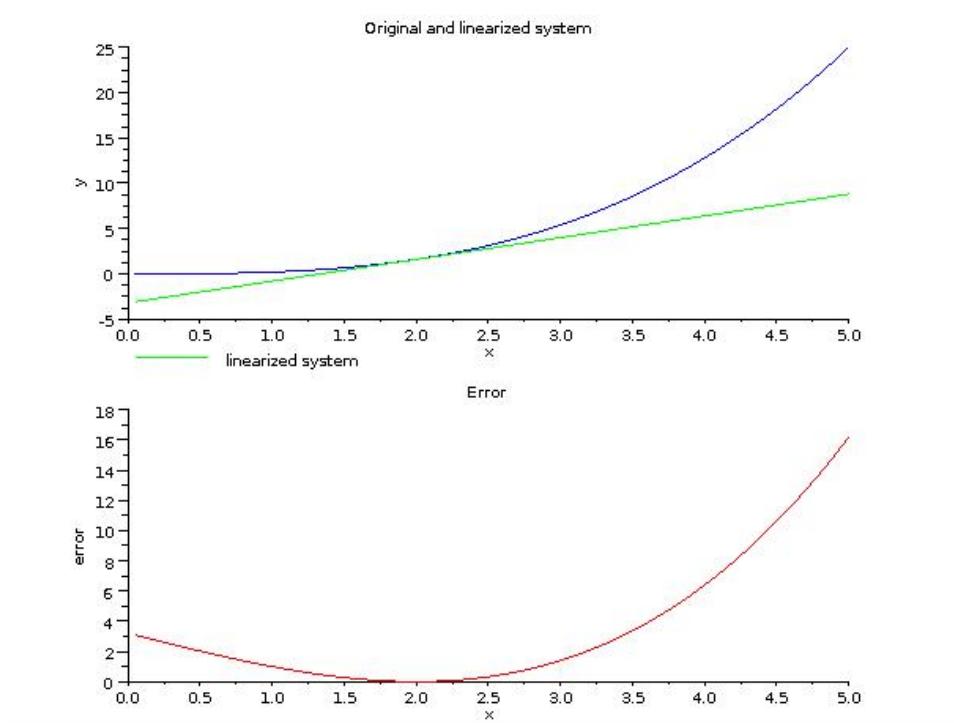


Figure 2.2: Verifying linearization of a non linear system

```

    error margin" );
31 xtitle('Percentage Error', 'x', '% error');

```

check Appendix [AP 4](#) for dependency:

`transferf.sci`

Scilab code Exa 2.4 Convert State space to Transfer Function model

```
1 // Example2-4
```

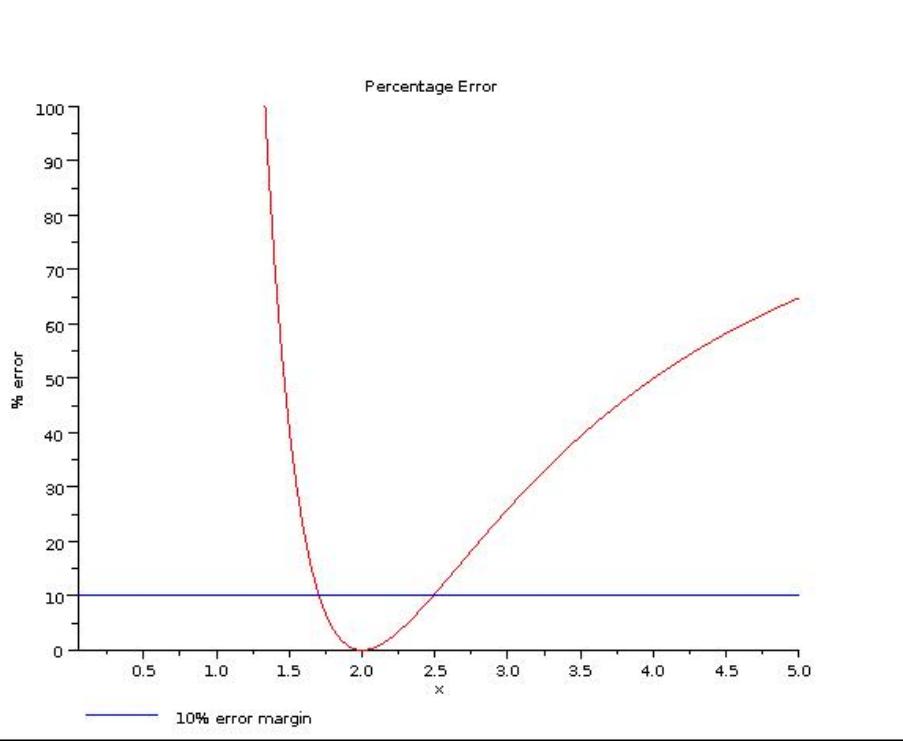


Figure 2.3: Verifying linearization of a non linear system

```
2 // Conversion from state space to transfer function
model
3
4 clear;clc;close;
5
6 // Please edit the path below
7 // cd "/your code directory/";
8 // exec("transferf.sci");
9
10 A = [0 1 0; 0 0 1;-5 -25 -5];
11 B = [0; 25; -120];
12 C = [1 0 0];
13 D = [0];
14 G = transferf(A,B,C,D);
15 disp(G,'transfer function = ');
```

Chapter 5

Transient and Steady State Response Analysis

Scilab code Exa 5.a.3 Verifying design to match given response curve

```
1 // Example A-5-3
2 // Verifying design to match given response curve
3
4 clear; clc;
5 xdel(winsid()); //close all windows
6
7 // Please edit the path
8 // cd "/<your code directory>/";
9 // exec("plotresp.sci");

10
11 s = %s;
12 K = 1.42;
13 T = 1.09;
14 K = 1.42;
15 G1 = (K/(s*(T*s + 1))) /. 1;
16 G = syslin('c',G1);
17
18 t = 0:0.1:10;
19 u = ones(1,length(t));
```

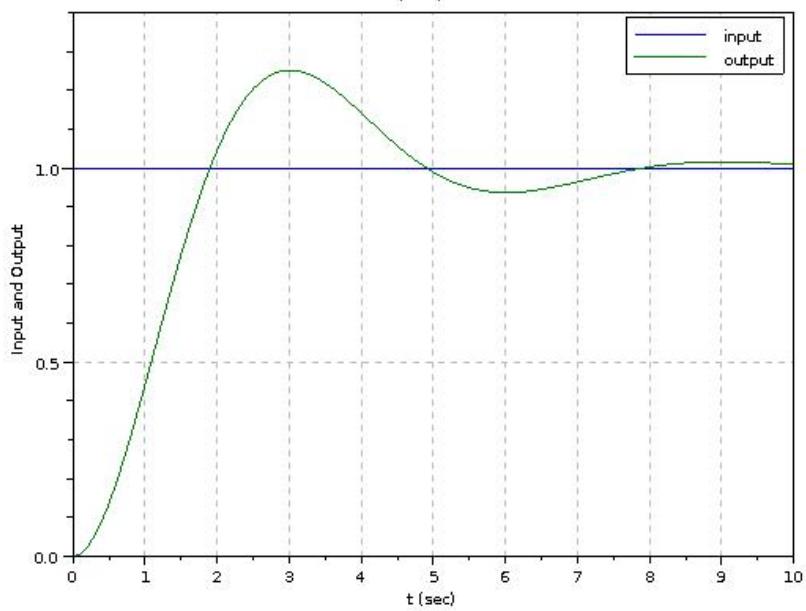


Figure 5.1: Verifying design to match given response curve

```

20 y = plotresp(u,t,G,'Step response');
21
22 [m t] = max(y);
23 Mp = m - 1;
24 tp = (t - 1) * 0.1;
25 disp(Mp,'Mp = ');
26 disp(tp,'tp = ');

```

check Appendix AP 2 for dependency:

`plotresp.sci`

Scilab code Exa 5.a.4 Determining K and k for required step response

```
1 // Example A-5-4
2 // Determining K and k for required step response
3
4 clear; clc;
5 xdel(winsid());
6 mode(0);
7
8 Mp = 0.25;
9 tp = 2;
10 J = 1; // kg.m^2
11
12 z = poly(0, 'z');
13 Eq = (z*pi)^2 - log(1/Mp)^2 * (1 - z^2);
14 x = roots(Eq);
15 zeta = abs(x(1))
16
17 wd = %pi / tp
18 wn = wd / sqrt(1 - zeta^2)
19 K = J * wn^2
20 k = 2*zeta*wn / K
```

Scilab code Exa 5.a.5 Verifying design to match given response

```
1 // Example A-5-5
2 // Verifying design to match given response curve
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 s = %s;
8 m = 5.2; // lb / ft^2
9 b = 12.2; // lb/ft/sec
```

```

10 k = 20; // lb / ft
11 G = syslin('c', 1, m*s^2 + b*s + k);
12
13 STEP = 0.05; t = 0:STEP:7;
14 u = 2 * ones(1, length(t));
15 y = csim(u, t, G);
16 plot(t, y);
17 xgrid(color('gray'));
18 xtitle('Step response', 't sec', 'Response');
19
20 [m t] = max(y);
21 Mp = (m - 0.1) / 0.1 * 100;
22 tp = (t - 1) * STEP;
23 disp(Mp, 'Mp (percent) = ');
24 disp(tp, 'tp = ');

```

Scilab code Exa 5.a.8 Unit step response and partial fraction expansion

```

1 // Example A-5-8
2 // Unit step response and partial fraction expansion
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 // Please edit path
8 // cd "<your codes path>/";
9 // exec("pf_residu.sci");
10 // exec("plotresp.sci");
11
12 s = %s ;
13 N = poly([80 72 25 3], 's', 'c');
14 D = poly([80 96 40 8 1], 's', 'c');
15 G = syslin('c', N, D)

```

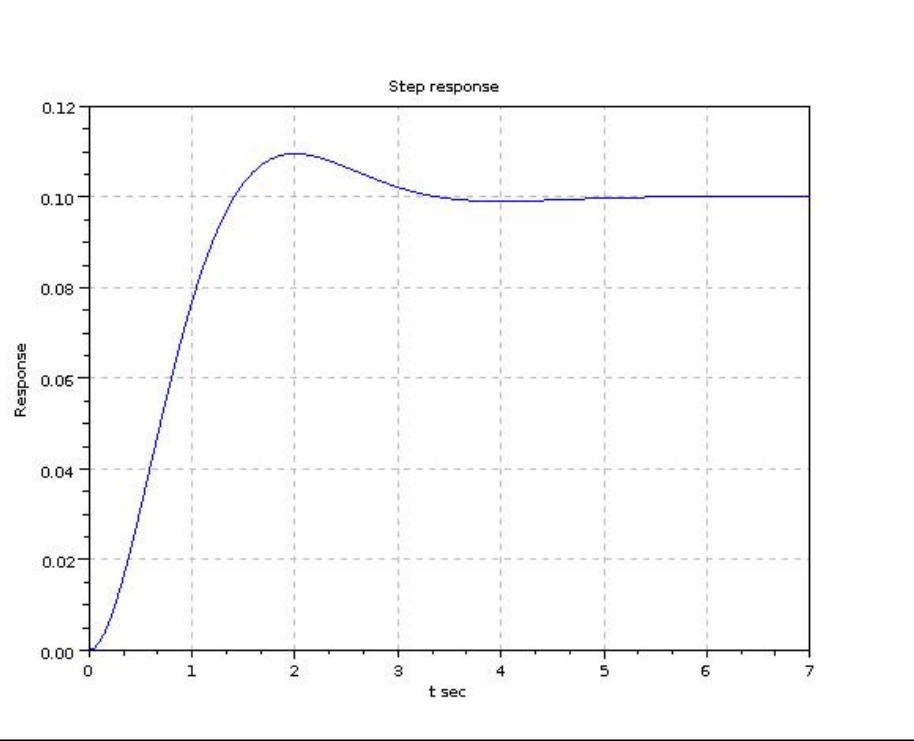


Figure 5.2: Verifying design to match given response

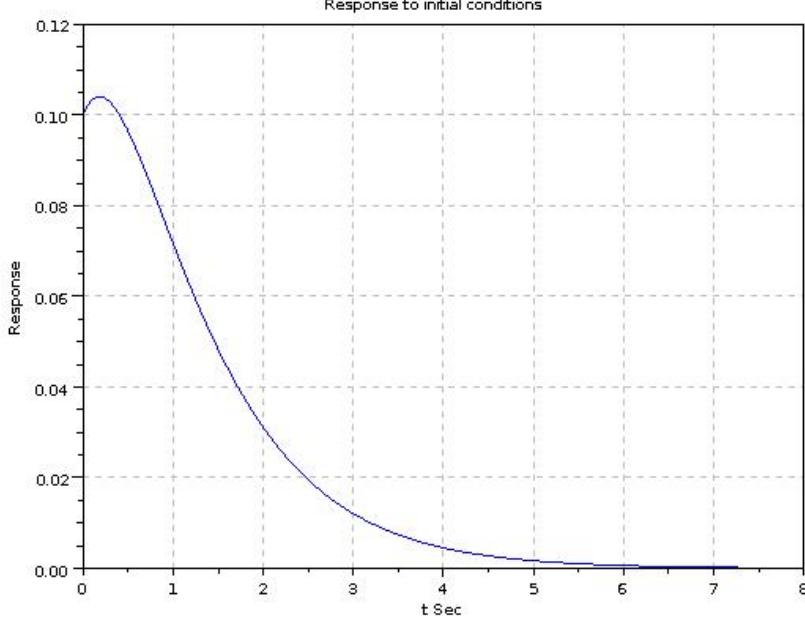


Figure 5.3: Unit step response and partial fraction expansion

```

16
17 t = 0:0.05:5;
18 u = ones(1,length(t));
19 plotresp(u,t,G, 'Unit Step Response of C(s) / D(s)');
20
21 // To find the residues of step response
22 D = D * s;
23 [r,z,p] = pf_residu(N,D);
24
25 disp(z, 'zeros = '); disp([p,r], 'poles : residues = ')
;
```

check Appendix AP 6 for dependency:

`pf_residu.sci`

check Appendix AP 2 for dependency:

`plotresp.sci`

Scilab code Exa 5.a.9 Effect of zeros on step response of a system

```
1 // Example A-5-9
2 // Effect of zeros on step response of a system
3 // Interactive program
4
5 clear; clc;
6 xdel(winsid()); // close all windows
7
8 function drawg()
9 delete(gca())
10 N = 4*(s*1/z + 1);
11 G = syslin('c',N,D);
12 ys = csim('step',t,G);
13 m = max(ys);
14 Mp = m -1;
15 plot(t,ys);
16 xtitle('Unit Step Response for zero at z = ' +
           string(z) + ' Mp = ' + string(Mp), 't (sec)', 'Output');
17 xgrid(color('gray'));
18 a = gca();
19 a.data_bounds = [0 0;10 4]
20 endfunction
21
22 s = %s;
23 z = 0.2;
24 D = s^2 + 4*s + 4;
25 t = 0:0.02:10;
26 drawg();
27 h = uicontrol('style','pushbutton','position',[250|10|60|20],'callback','z = z - 0.1;drawg()',
```

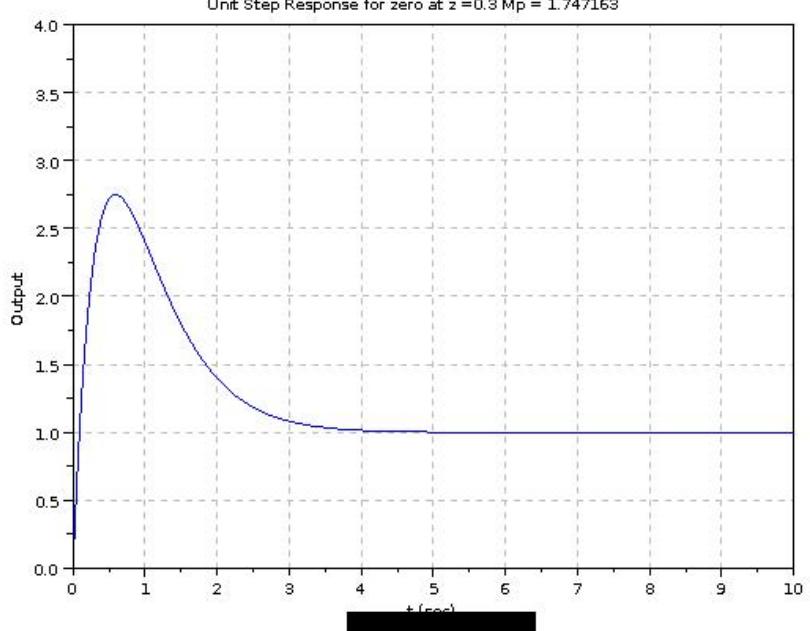


Figure 5.4: Effect of zeros on step response of a system

```

        String ', '<-');
28 j = uicontrol('style', 'pushbutton', 'position', [
310|10|60|20, 'callback', 'z = z + 0.1; drawg()', '
String ', '->');

```

Scilab code Exa 5.a.10 Step response characteristics

```

1 // Example A-5-10
2 // Plot the unit step response and find the
   transient parameters
3 // viz. - rise time, peak time , settling time and

```

```

        maximum overshoot
4
5 clear; clc;
6 xdel(winsid()); // close all windows
7 mode(0);
8
9 // Please edit path if needed
10 // cd "/<your code path>/";
11 // exec("stepch.sci");
12
13 N = poly([12.811 18 6.3223], 's', 'c') ;
14 D = poly([12.811 18 11.3223 6 1], 's', 'c');
15 G = syslin('c', N, D);
16 [Mp tp tr ts] = stepch(G, 0, 20, 0.01, 0.02)

```

check Appendix AP 8 for dependency:

stepch.sci

Scilab code Exa 5.a.11 Step Response for different zeta and wn

```

1 // Example A-5-11
2 // Unit Step Response for different systems for
   different zeta ,wn
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 zeta = [0.3 0.5 0.7 0.8];
8 wn    = [1 2 4 6];
9 n     = wn .^ 2;
10 sigma= 2 .* zeta .* wn;
11
12 s = %s;

```

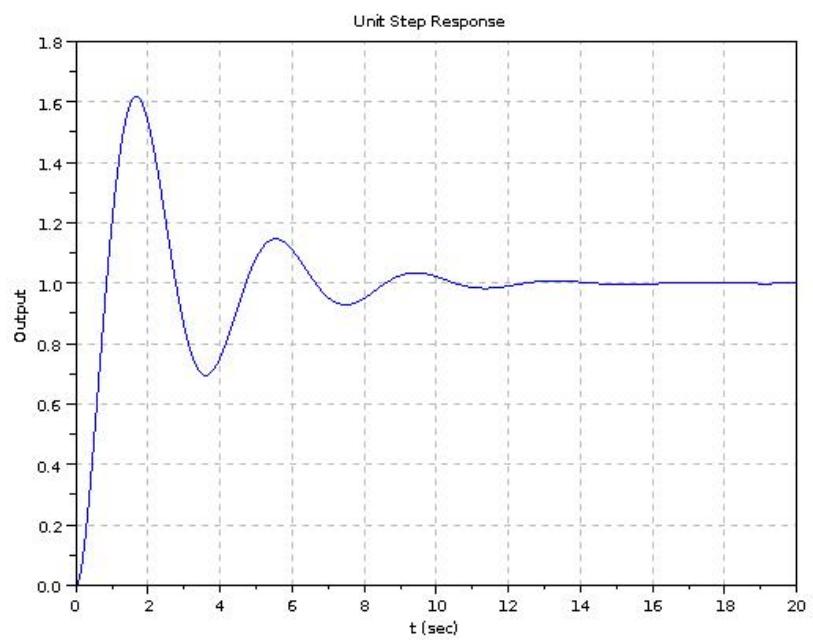


Figure 5.5: Step response characteristics

```

13 t = 0:0.1:10;
14 for i= 1:4
15 z(i,:) = csim('step',t,syslin('c', n(i), s^2 + sigma
    (i)*s + n(i)));
16 end
17
18 plot(t,z); // 2d plot of step responses
19
20 xtitle('Plot of step response curves with different
    wn and zeta','t sec','Response');
21 xgrid(color('gray'));
22 legend('(zeta,wn) = (0.3 , 1)', '(0.5 , 2)', '(0.7 ,
    4)', '(0.8 , 6)');

```

Scilab code Exa 5.a.12 Response to unit ramp and exponential input

```

1 // Example A-5-12
2 // Response to unit ramp and exponential input
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 // Please edit path if needed
8 // cd "/<your code folder>/"
9 // exec("plotresp.sci");
10
11 s = %s;
12 G = syslin('c', s + 10, s^3 + 6*s^2 + 9*s + 10);
13
14 t = 0:0.05:10;
15 e = exp(-0.5 * t);
16 plotresp(t,t,G,'Response to unit ramp input');
17 scf();

```

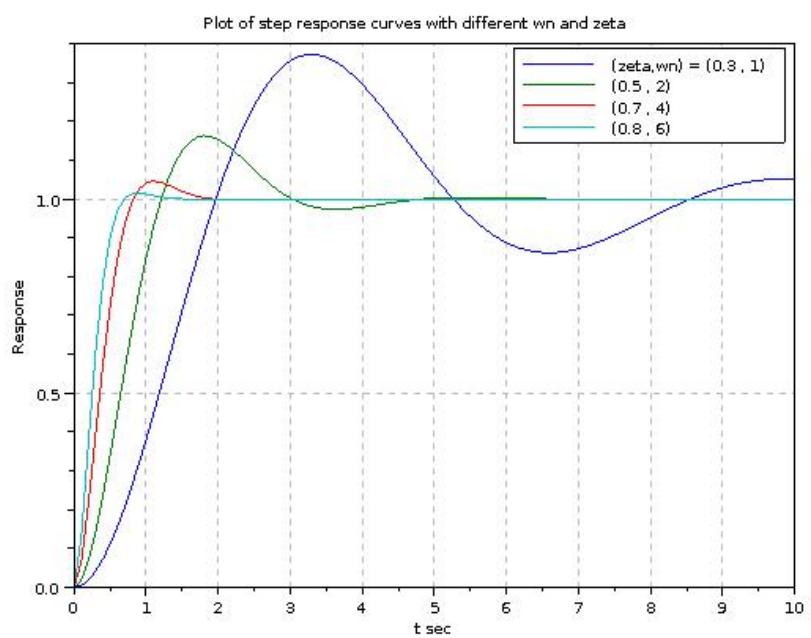


Figure 5.6: Step Response for different zeta and wn

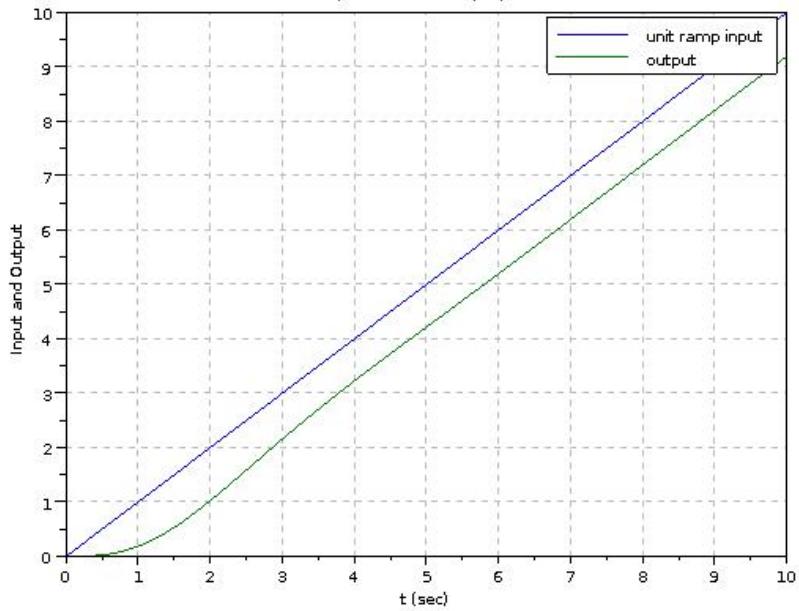


Figure 5.7: Response to unit ramp and exponential input

```
18 plotresp(e,t,G,'Response to exponential input');
```

check Appendix AP 2 for dependency:

`plotresp.sci`

Scilab code Exa 5.a.13 Response to input r equals 2 plus t

```
1 // Example A-5-13
2 // Response to input r = 2 + t
```

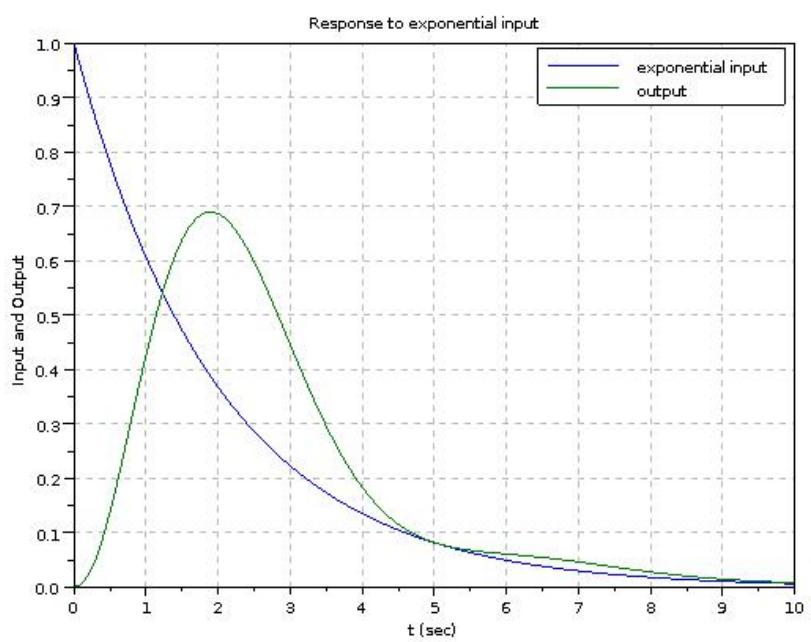


Figure 5.8: Response to unit ramp and exponential input

```

3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 // Please edit the path
8 // cd "/<your code folder>/Codes/chapter_5";
9 // exec("plotresp.sci")
10
11 s = %s;
12 G = syslin('c', 5, s^2 + s + 5);
13 t = 0:0.05:10;
14 r = 2 + t;
15 plotresp(r,t,G,'Response to input r = 2 + t');

```

check Appendix AP 2 for dependency:

plotresp.sci

Scilab code Exa 5.a.14 Response to unit acceleration input

```

1 // Example A-5-14
2 // Response to unit acceleration r = (1/2) * t^2
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 // please edit the path
8 // cd "/<your code folder>/Codes/chapter_5"
9 // exec("plotresp.sci")
10
11 s = %s;
12 G = syslin('c', 2, s^2 + s + 2);
13 t = 0:0.05:10;
14 r = (1/2) * t.^2;

```

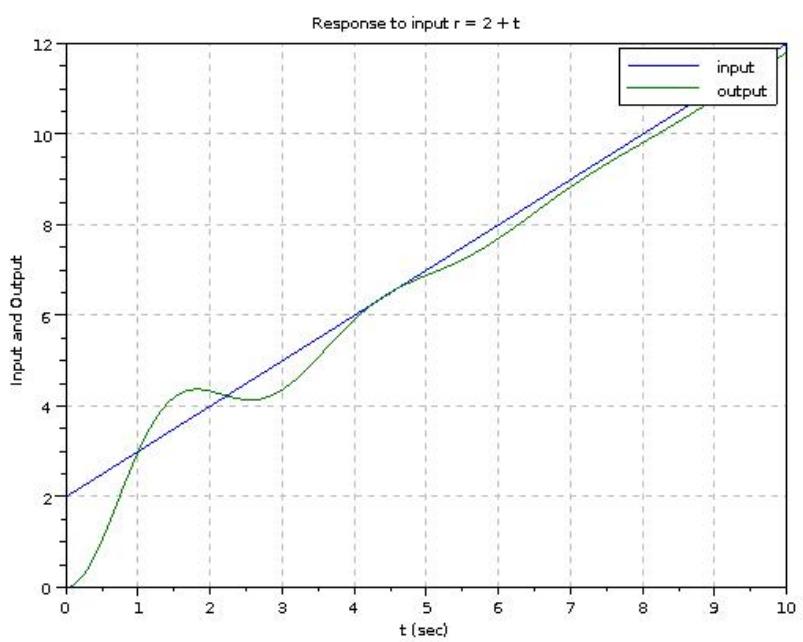


Figure 5.9: Response to input r equals 2 plus t

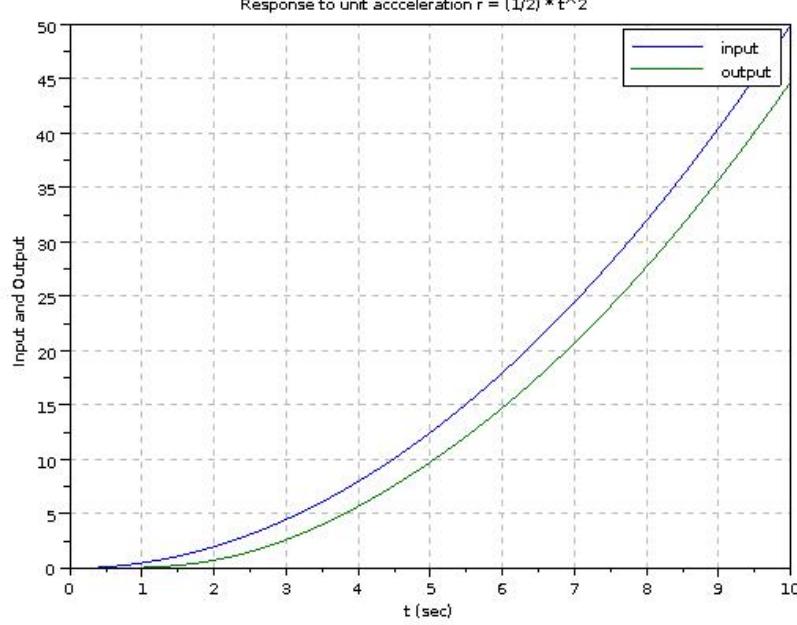


Figure 5.10: Response to unit acceleration input

```
15 plotresp(r,t,G,'Response to unit acceleration r =
(1/2) * t^2');
```

check Appendix AP 2 for dependency:

`plotresp.sci`

Scilab code Exa 5.a.15 Step Responses for different zeta

```
1 // Example A-5-15
2 // 2d and 3d plot for various values of zeta
3
```

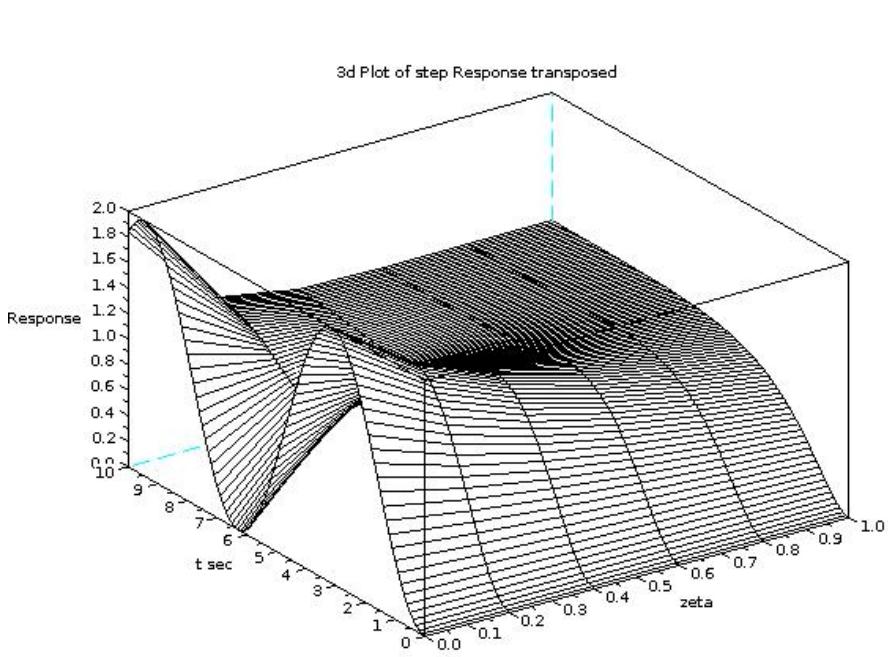


Figure 5.11: Step Responses for different zeta

```

4 // Please refer to example 5-4
5
6 // To get the trasnposed plot please add the lines
7
8 scf();
9 mesh(y,x,z);
10 xtitle(' 3d Plot of step Response transposed ','zeta',
      't sec ','Response');

```

Scilab code Exa 5.a.16 Response to initial conditions

```

1 // Example A-5-16
2 // Response to initial conditions
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 A = [0 1 0; 0 0 1; -10 -17 -8];
8 C = [1 0 0];
9 x0 = [2; 1; 0.5];
10 G = syslin('c',A,[0; 0; 0],C,0,x0);
11
12 t = 0:0.05:10;
13 u = zeros(1,length(t));
14 y = csim(u,t,G);
15
16 plot(t,y);
17 xgrid(color('gray'));
18 xtitle('Response to initial condition','t (sec)', 'output');

```

Scilab code Exa 5.2 Determining K and Kh for required step response

```

1 // Example 5-2
2 // Determining K and Kh for required step response
   characteristics
3
4 clear; clc;
5 xdel(winsid());
6 mode(0);
7
8 Mp = 0.2;
9 tp = 1;
10 J = 1; // kg.m^2

```

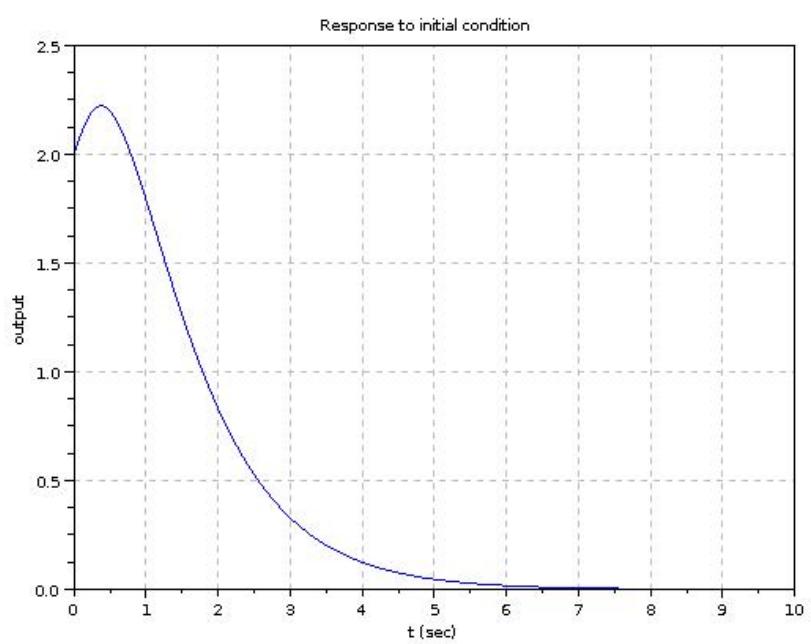


Figure 5.12: Response to initial conditions

```

11 B = 1; // N-/rad/sec
12
13 z = poly(0, 'z');
14 Eq = (z*pi)^2 - log(1/Mp)^2 * (1 - z^2);
15 x = roots(Eq);
16 zeta = abs(x(1))
17
18 wd = %pi / tp
19 wn = wd / sqrt(1 - zeta^2)
20 K = J * wn^2
21 Kh = (2*sqrt(K*J)*zeta - B) / K
22
23 sigma = wn*zeta;
24 _beta = atan(wd/sigma)
25 tr = (%pi - _beta) / wd
26 ts_2percent = 4 / sigma
27 ts_5percent = 3 / sigma

```

Scilab code Exa 5.3 Step response of MIMO system

```

1 // Example 5-3
2 // Step response of a linear System given in State
   Space
3 // Model (Multiple Input Multiple Output System)
4
5 clear; clc;
6 xdel(winsid()); // close all windows
7
8 A = [-1 -1; 6.5 0];
9 B = [1 1; 1 0];
10 C = [1 0; 0 1];
11 D = [0 0; 0 0];
12 G = syslin('c', A, B, C, D);
13 Gtf = clean(ss2tf(G));
14 disp(Gtf, 'Gtf = '); // transfer function matrix

```

```

15
16 N = 200;                                //No of points
17 t = linspace(0,10,N);
18 u1 = [ones(1,N) ; zeros(1,N)];
19 u2 = [zeros(1,N); ones(1,N) ];
20
21 y1 = csim(u1,t,G);                      // find system response
22 y2 = csim(u2,t,G);
23
24 plot(t,y1);
25 xtitle('Unit Step Response: input = u1 (u2 = 0)', 't
    Sec', 'Response');
26 xgrid(color('gray'));                     // grid
27 legend('output: y1', 'output: y2');
28
29 scf(1);                                 // new window
30 plot(t,y2);
31 xtitle('Unit Step Response: input = u2 (u1 = 0)', 't
    Sec', 'Response');
32 xgrid(color('gray'));
33 legend('output: y1', 'output: y2');
34
35 // We cannot use csim('step', , ) because this
    option is only available
36 // for SISO systems

```

Scilab code Exa 5.4 Second order systems with different damping ratio

```

1 // Example 5-4
2 // 2d and 3d plots of standard second order systems
3 // with wn = 1 and different damping ratios

```

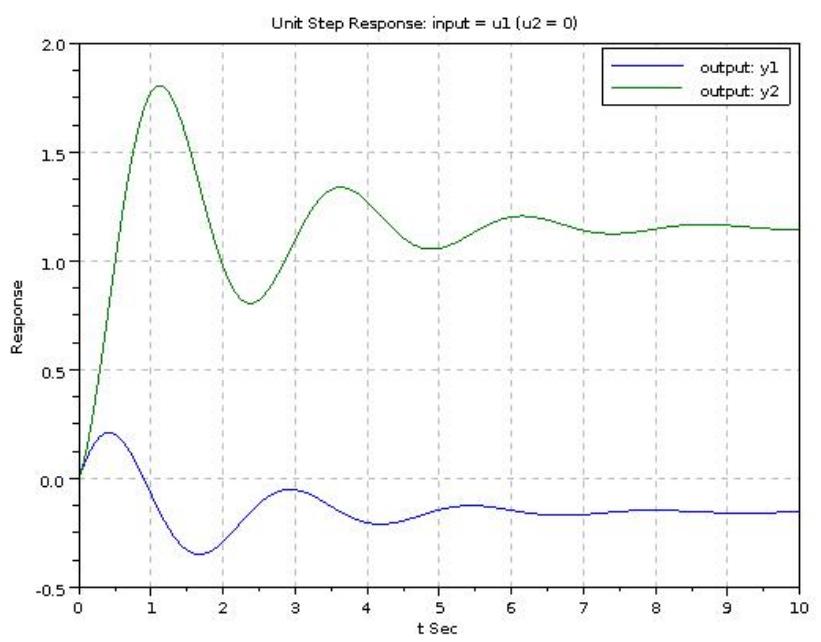


Figure 5.13: Step response of MIMO system

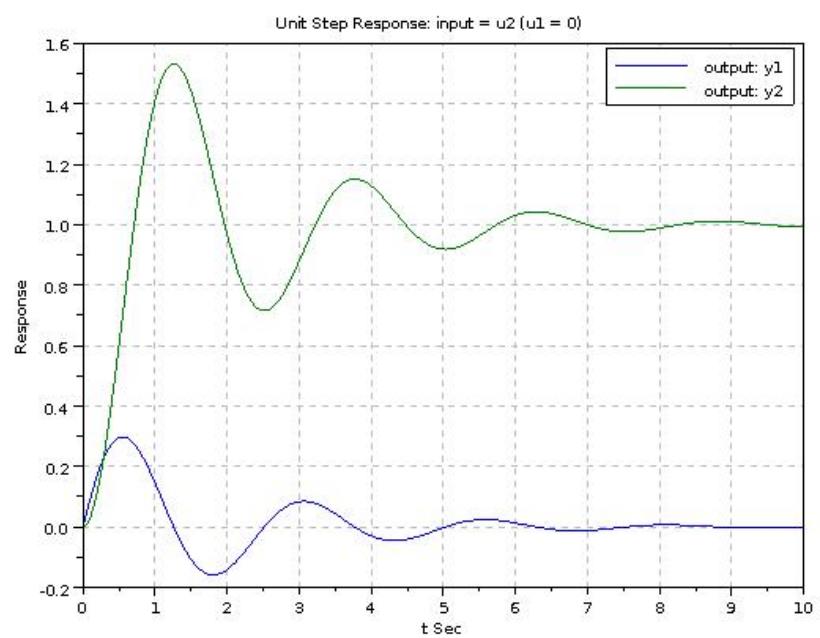


Figure 5.14: Step response of MIMO system

```

4
5 clear; clc;
6 xdel(winsid()); // close all windows
7
8 s = %s;
9 t = 0:0.1:10;
10 zeta = 0:0.2:1;
11
12 for n = 1:6
13     z(n,:) = csim('step',t,syslin('c', 1,s^2 + 2*
zeta(n)*s + 1));
14 end
15
16 plot(t,z); // 2d plot of step responses
17 xtitle('Plot of step response curves with wn = 1 and
different zeta','t sec','Response');
18 xgrid(color('gray'));
19 legend('zeta = 0','0.2','0.4','0.6','0.8','1.0');
20
21 scf(); // new window
22
23 [x,y] = meshgrid(0:0.1:10 , 0:0.2:1); //needed by
the mesh command
24 mesh(x,y,z);
25 xtitle(' 3d Plot of step Response','t sec','zeta',
'Response');

```

Scilab code Exa 5.5 Impulse Response of a Second order System

```

1 // Example 5-5
2 // Impulse Response of a Second Order System

```

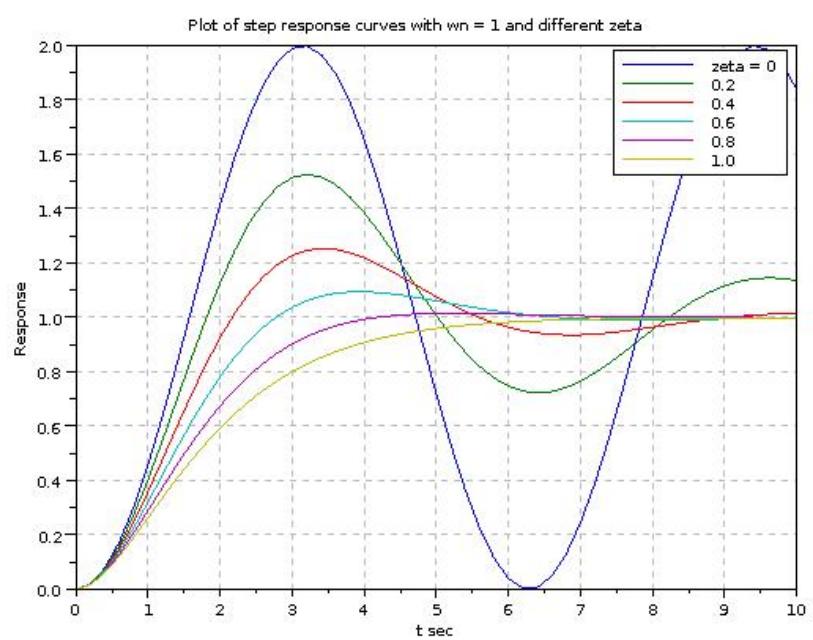


Figure 5.15: Second order systems with different damping ratio

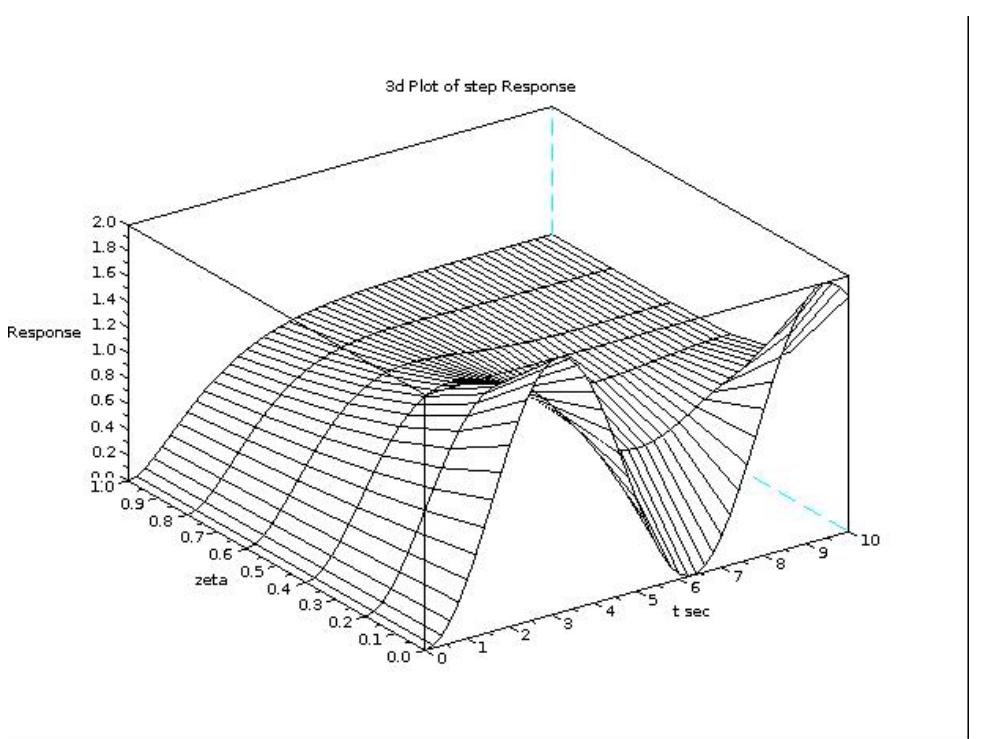


Figure 5.16: Second order systems with different damping ratio

```

3
4 clear; clc;
5 xdel(winsid()); //close all windows
6
7 s = %s;
8 G = syslin('c', 1, s^2 + 0.2*s + 1);
9
10 t = 0:0.5:50;
11 y = csim('impuls', t, G);
12 plot(t, y);
13 xtitle('Impulse Response of 1/ (s^2 + 0.2*s + 1)', 't
sec', 'Response');
14 xgrid(color('gray'));

```

check Appendix AP 2 for dependency:

plotresp.sci

Scilab code Exa 5.6 Unit Ramp response of a second order system

```

1 // Example 5-6
2 // Unit Ramp response of a second order system
3
4 clear; clc;
5 xdel(winsid()); //close all windows
6
7 // Please edit the path
8 // cd "/<your code directory>/";
9 // exec("plotresp.sci");
10
11 s = %s
12 G = syslin('c', 2*s + 1, s^2 + s + 1);
13
14 t = 0:0.05:10;

```

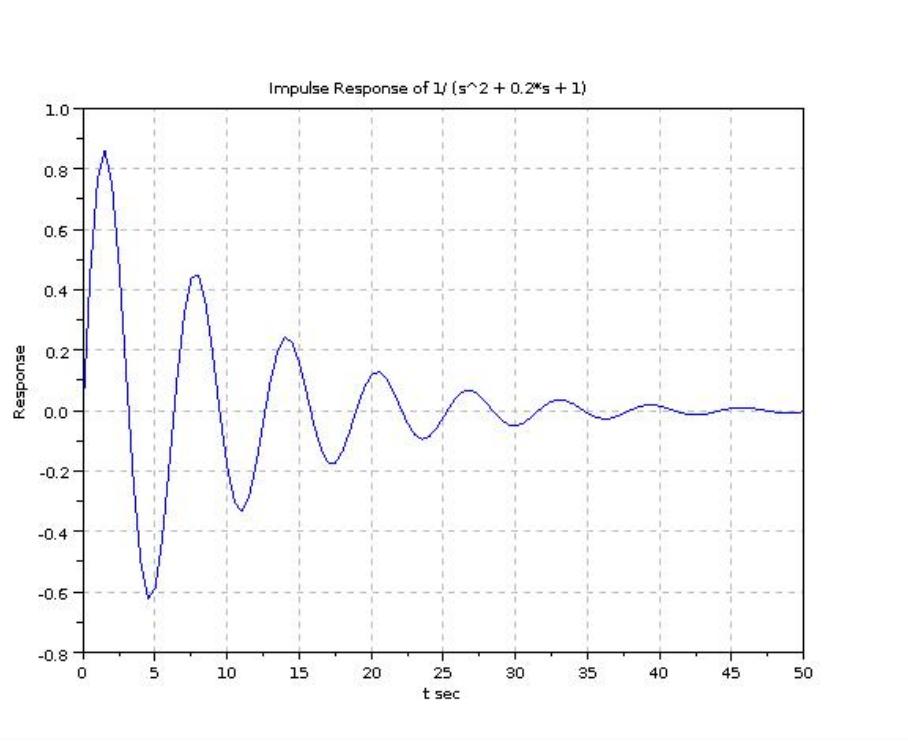


Figure 5.17: Impulse Response of a Second order System

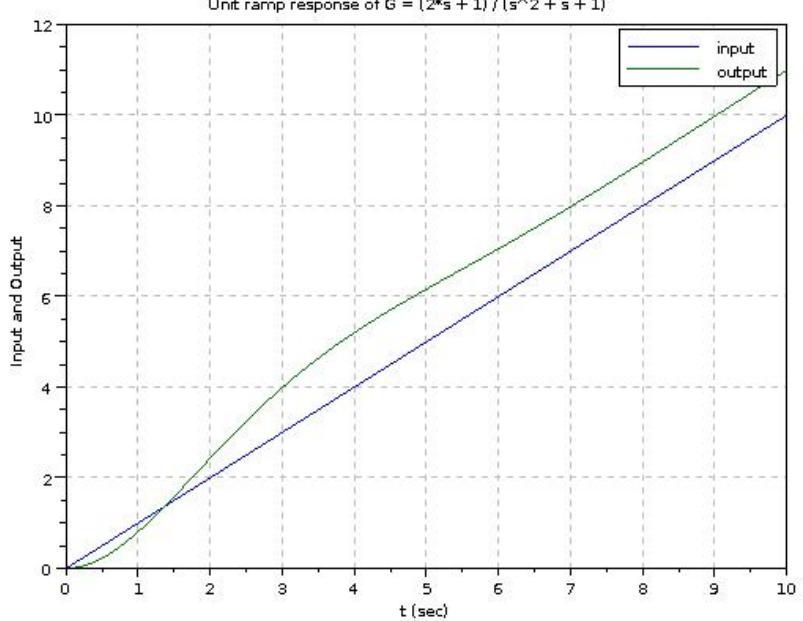


Figure 5.18: Unit Ramp response of a second order system

```
15 plotresp(t,t,G,'Unit ramp response of G = (2*s + 1)
/ (s^2 + s + 1)');
```

check Appendix AP 2 for dependency:

`plotresp.sci`

Scilab code Exa 5.7 Response to step and exponential input

```
1 // Example 5-7
2 // Response to step and exponential input
3
```

```

4 clear; clc;
5 xdel(winsid()); //close all windows
6
7 // Please edit the path
8 // cd "/<your code directory >/";
9 // exec("plotresp.sci");
10
11 t = 0:0.1:16;
12 A = [-1 0.5; -1 0];
13 B = [0; 1];
14 C = [1 0];
15 D = [0];
16 G = syslin('c',A,B,C,D);
17
18 // unit step response
19 u = ones(1,length(t));
20 plotresp(u,t,G,'Unit-Step Response');
21 scf();
22 // response to exponential input = e^(-t)
23 u = exp(-t);
24 plotresp(u,t,G,'Response to exponential input');

```

Scilab code Exa 5.8 Response to initial condition

```

1 // Example 5-8
2 // Response to initial condition (Transfer Function)
3
4 clear; clc;
5 xdel(winsid()); //close all windows
6
7 s = %s;

```

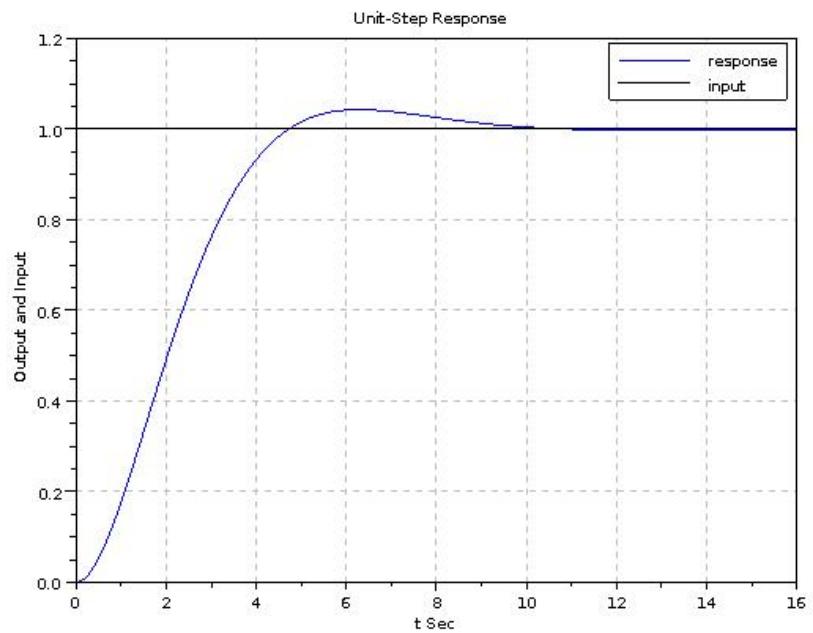


Figure 5.19: Response to step and exponential input

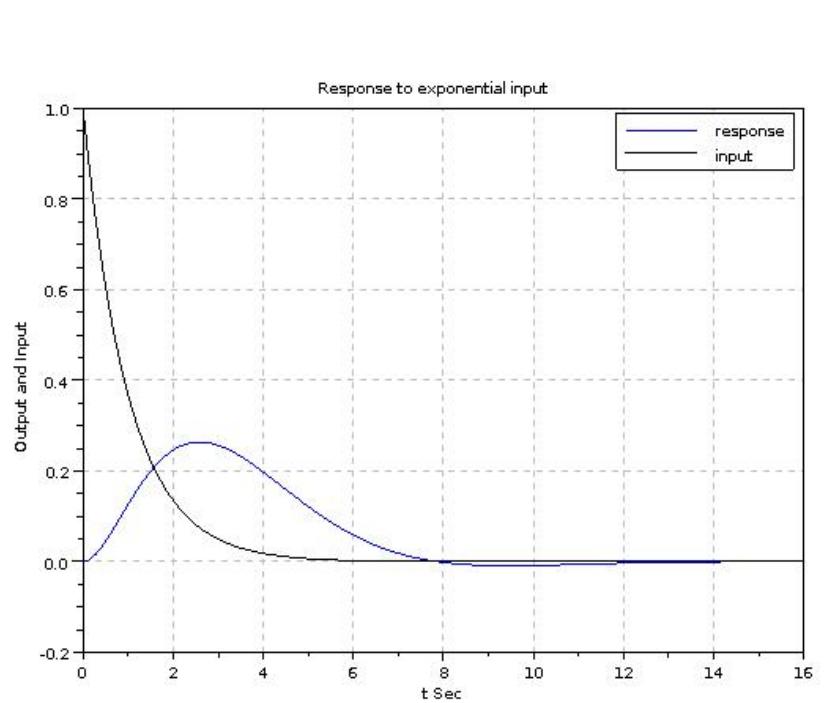


Figure 5.20: Response to step and exponential input

```

8 N = 0.1*s^2 + 0.35*s ;
9 D = s^2 + 3*s + 2;
10 G = syslin('c',N,D);
11
12 t = linspace(0,8,200);
13 u = ones(1,200);
14 y = csim(u,t,G);
15
16 plot(t,y);
17 xtitle('Response to initial conditions','t Sec',
         'Response');
18 xgrid(color('gray'));
19 // We cannot use the 'step' version of csim directly
20 // as direct feedback sets to zero for the 'step'
   option

```

Scilab code Exa 5.9 Response to initial conditions using state space

```

1 // Example 5-9
2 // Response to initial conditions using state space
   approach
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 A = [0 1; -10 -5];
8 x0 = [2; 1];
9 G = syslin('c',A,x0,[0 0],[0]); // use dummy C and D
   variables
10
11 t = 0:0.01:3;
12 [y,x] = csim('impuls',t,G);
13

```

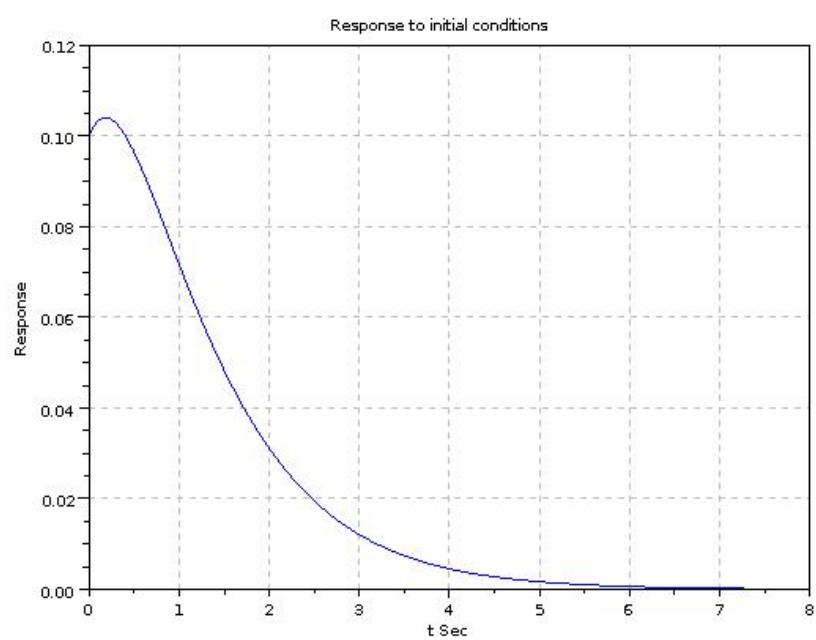


Figure 5.21: Response to initial condition

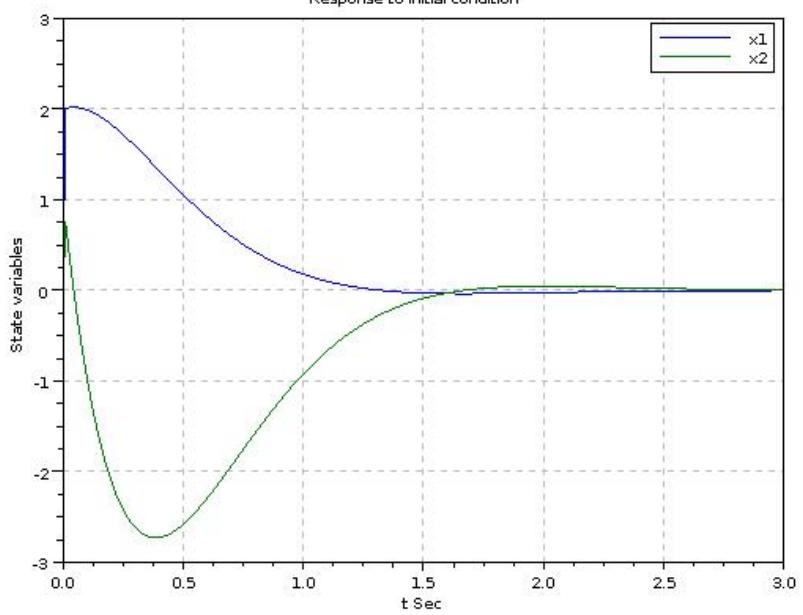


Figure 5.22: Response to initial conditions using state space

```

14 plot(t, x(1,:), t, x(2,:));
15 xtitle('Response to initial condition', 't Sec', 
          State variables');
16 xgrid(color('gray'));
17 legend('x1', 'x2');
18 // The State variables x, respond only to A,B
      matrices
19 // changing C and D will make no difference.

```

Scilab code Exa 5.10 Response to initial condition using syslin x0

```

1 // Example 5-10
2 // Response to initial condition ( differential
   equation)
3 // Solution of differential equation with initial
   conditions
4
5 clear; clc;
6 xdel(winsid()); // close all windowss
7
8 t = 0:0.05:10;
9 s = %s;
10 G1 = cont_frm(1, s^3 + 8*s^2 + 17*s + 10); //get the
      state space model
11 ssprint(G1);
12
13 x0 = [2; 1; 0.5]; // initial states of the system
14 G = syslin('c', G1.A, G1.B, G1.C, G1.D, x0);
15
16 y = csim( zeros(1,length(t)) , t, G);
17          // response to zero input will give response
          to initial state
18 plot(t,y);
19 xgrid(color('gray'));
20 xtitle('Response to initial conditions','t Sec','y')
;
```

Scilab code Exa 5.12 Constructing Routh array

```

1 // Example 5-12
2 // Constructing Routh array in scilab
3
4 clear; clc;
5 xdel(winsid()); // close all windows
```

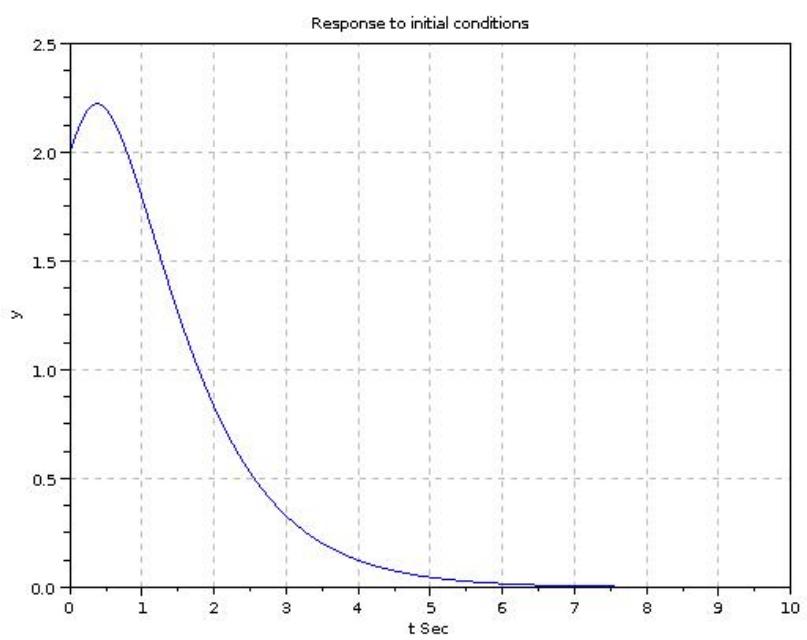


Figure 5.23: Response to initial condition using syslin x0

```
6 mode(0);
7
8 s = %s;
9 H = s^4 + 2*s^3 + 3*s^2 + 4*s + 5;
10 routh_t(H) // display the routh table
```

Scilab code Exa 5.13 Constructing Routh array

```
1 // Example 5-13
2 // Constructing Routh array in scilab
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6 mode(0);
7
8 s = %s;
9 H = s^5 + 2*s^4 + 24*s^3 + 48*s^2 - 25*s - 50;
10 routh_t(H)
11
12 // In this example a zero row forms at s^3
13 // the function automatically computes the
14 // derivative of the
15 // auxilliary polynomial 2s^4 + 48s^2 - 50
16 // viz = 8*s^3 + 96s^2
```

Chapter 6

Control Systems Analysis and Design by Root Locus Method

check Appendix AP 12 for dependency:

gainat.sci

Scilab code Exa 6.i.1 Finding the Gain K at any point on the root locus

```
1 // Illustration 6.1
2 // Finding the Gain K at any point on the root locus
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 // please set the path
8 // cd "/<your code directory>/"
9 // exec("rootl.sci");
10 // exec("gainat.sci");
11
12 function drawr()
```

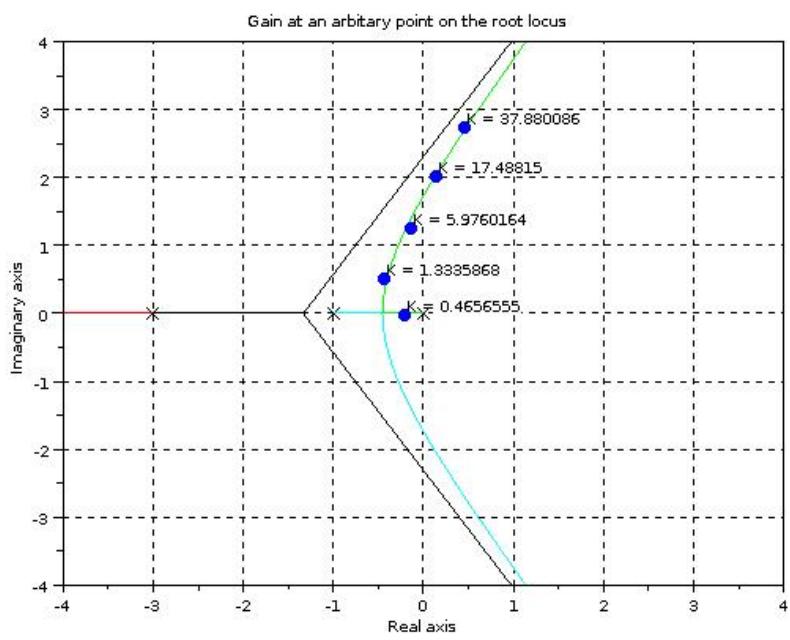


Figure 6.1: Finding the Gain K at any point on the root locus

```

13   rootl(G, [-4 -4; 4 4], 'Gain at an arbitrary point on
      the root locus');
14 endfunction
15
16 s = %s;
17 G = syslin('c', 1, s * (s + 1) * (s + 3));
18 drawr();
19 addmenu(0, 'Gain', ['Select Point 5 points', 'clear']);
20 Gain_0 = ['for i = 1:5; gainat(G); end;', 'delete(gca()
()); drawr();'];
21
22 // click on the Gain menu in the menu bar
23 // clear will restore your rootlocus

```

check Appendix AP 7 for dependency:

`rootl.sci`

Scilab code Exa 6.i.2 Orthogonality Constant gain curves and Root Locus

```

1 // Illustration 6.2
2 // Orthogonality of constant gain curves and root
   locus
3 // and the root locus
4
5 // Section 6.3 Figure 6-29 in the book
6
7 clear; clc;
8 xdel(winsid()); // close all windows
9
10 // please set the path
11 // cd "/<your code directory>/"
12 // exec("rootl.sci");

```

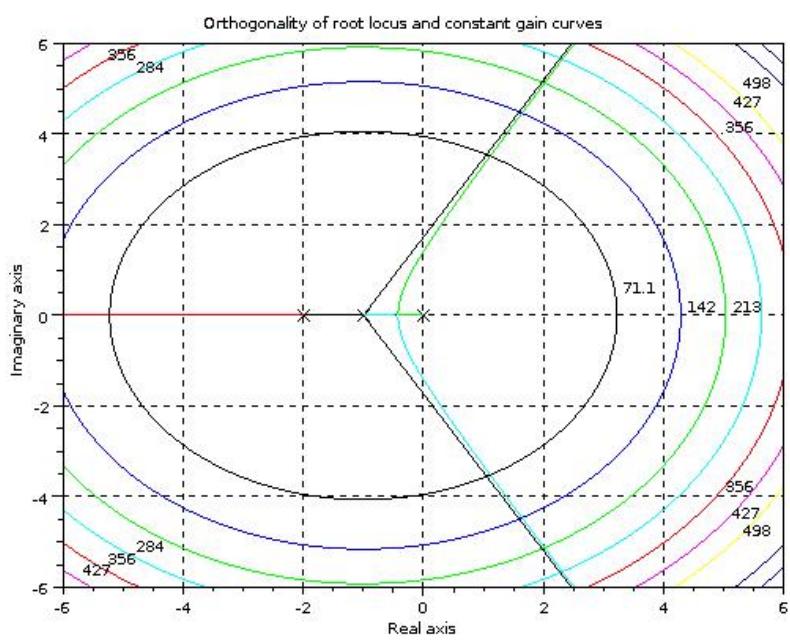


Figure 6.2: Orthogonality Constant gain curves and Root Locus

```

13
14 s = %s;
15 P = 1 / ( s * (s + 1) * (s + 2) );
16 G = syslin('c',P);
17
18 rootl(G, [ -6 -6; 6 6 ], 'Orthogonality of root locus
    and constant gain curves');
19
20 P = 1 / P;
21 v = -6:0.1:6;
22 [X,Y] = ndgrid(v,v); // prepares a grid to compute
    the gain
23 S = X + %i * Y;
24 K = abs(horner(P,S)); // Gain evaluated over the
    grid
25
26 contour(v,v,K,10); // plot lines of constant gain

```

check Appendix AP 7 for dependency:

`rootl.sci`

Scilab code Exa 6.i.3 Effect of adding poles or zeros on the root locus

```

1 // Illustration 6.3
2 // Effect of adding poles or zeros on the root locus
    of the system
3 // (section 6-5). (fig 6-35)
4 // Interactive Program
5
6 // A MENU called "Add" will be added to the window
7
8 clear; clc;
9 xdel(winsid()); // close all windows

```

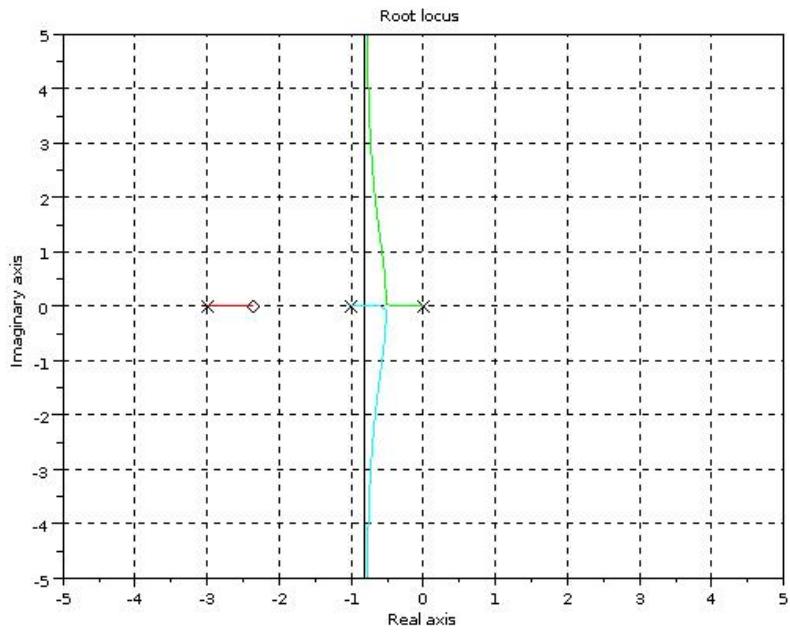


Figure 6.3: Effect of adding poles or zeros on the root locus

```

10
11 // please set the path
12 // cd "<your code directory>/"
13 // exec("rootl.sci");
14
15 function J = add(n,H)
16
17 z = locate(1,1);
18 x = z(1);y = z(2);
19 N = H.num;
20 D = H.den;
21 if abs(y) <= 0.2 then
22   if n == 1 then D = D * (s-x);
23   else N = N * (s-x);
24 end
25 zp = x;
26 else
27   if n == 1 then D = D * (s^2 - 2*x*s + x^2 + y
28     ^2);
29   else N = N * (s^2 - 2*x*s + x^2 + y^2);
30 end
31 zp = x + %i * y;
32 end
33 J = syslin('c',N,D);
34 draws(J);
35 if(n == 1) then disp(zp,"p = "); else disp(zp,"z
36 = ");end
37 disp(J,"G = ");
38 endfunction
39
40 function draws(P)
41   delete(gca());
42   rootl(P,[-5 -5; 5 5], 'Root locus'); //you can
43   change the range :[-20,-20;20,20];
44
45 // Main Program

```

```

45 s = %s;
46 N = 1;
47 D = s * (s + 1) * (s + 3);
48 G = syslin('c', 1, D);
49 H = G;
50
51 draws(G);
52 addmenu(0, 'Add', [ 'Pole ', 'zero ', 'Reset ']);
53 Add_0 = [ 'H = add(1,H)', 'H = add(2,H);', 'draws(G);H='
      G; ];
54
55 // place a zero close to the pole at -3
56 // first place it to the right then , to the left
57 // Then mover farther to the right.[-5 -5; 5 5]

```

check Appendix [AP 7](#) for dependency:

`rootl.sci`

Scilab code Exa 6.a.6 Root locus

```

1 // Example A-6-6
2 // Root locus
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 // please edit the path
8 // cd "/<your code directory>/";
9 // exec("rootl.sci");
10
11 s = %s;
12 G = syslin('c', 1, s * (s + 1) * (s^2 + 4*s + 13));
13 rootl(G, [-6 -5; 6 5], 'Root locus plot for 1/ [s * (s
      + 1) * (s^2 + 4*s + 13)]');
14

```

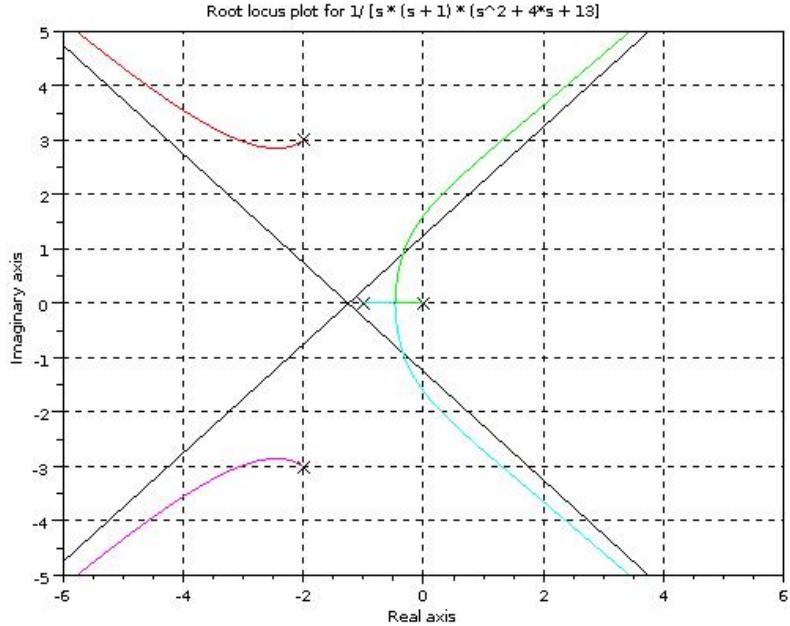


Figure 6.4: Root locus

```

15 // the same method may be employed to plot root loci
   in examples
16 // A=[-6,-1,2,3,8,10,
17 // simply write the transfer function and choose
   suitable range
18 // [xmin ymin; xmax ymax]

```

check Appendix AP 7 for dependency:

`rootl.sci`

Scilab code Exa 6.a.13.1 Lead Compensator Design Attempt 1

```

1 // Example A-6-13-1
2 // Lead Compensator Design Attempt 1
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 // please edit the path
8 // cd "/<your code directory>/";
9 // exec("rootl.sci");
10 // exec("plotresp.sci");
11
12 s = %s;
13 G = syslin('c',1,s^2);
14 H = syslin('c',1,0.1*s + 1);
15
16 R = [-1 -1];
17 I = [1.73205 -1.73205];
18 dp = R(1) + %i*I(1);
19
20 subplot(1,2,1);
21 rootl(G*H,[-15 -15; 5 15], 'Root locus plot for
    uncompensated system');
22 plot(R,I,'x');
23 angdef = 180 - phasemag(horner(G*H,dp));
24 disp(angdef, 'angle deficiency =');
25
26 z = 1; // zero at -1;
27 p = 1.73205 / tand(90 - angdef) + 1 ;
28 Gc = (s + z) / (s + p);
29 disp(Gc, 'lead compensator =');
30
31 Kc = abs(1/ horner(G*Gc*H,dp));
32 disp(Kc, 'Kc =');
33 O = Kc*Gc*G*H; disp(O, 'open loop Transfer function
    =');
34 C = Kc*Gc*G /. H; disp(C, 'closed loop Transfer
    function =');
35 disp(roots(C.den), 'closed loop poles =');

```

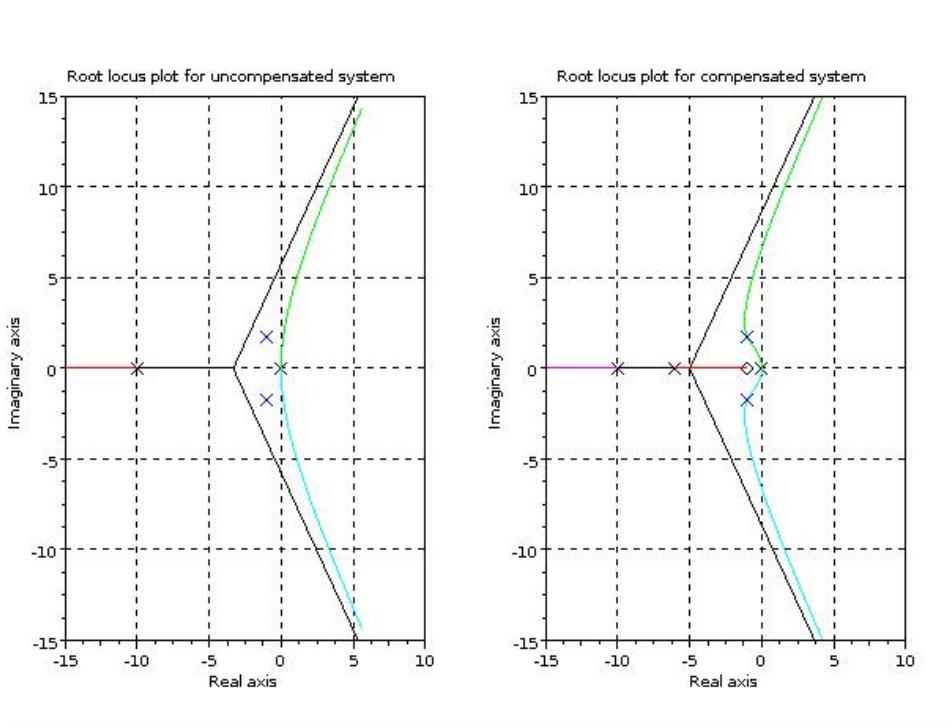


Figure 6.5: Lead Compensator Design Attempt 1

```

36
37 subplot(1,2,2);
38 rootl(0,[-15 -15; 5 15], 'Root locus plot for
            compensated system');
39 plot(R,I,'x');
40
41 scf();
42 t = 0:0.05:10;
43 u = ones(1,length(t)); //step response
44 plotresp(u,t,C, 'Unit step response');
45 xstring(1,0.95, 'compensated system');

```

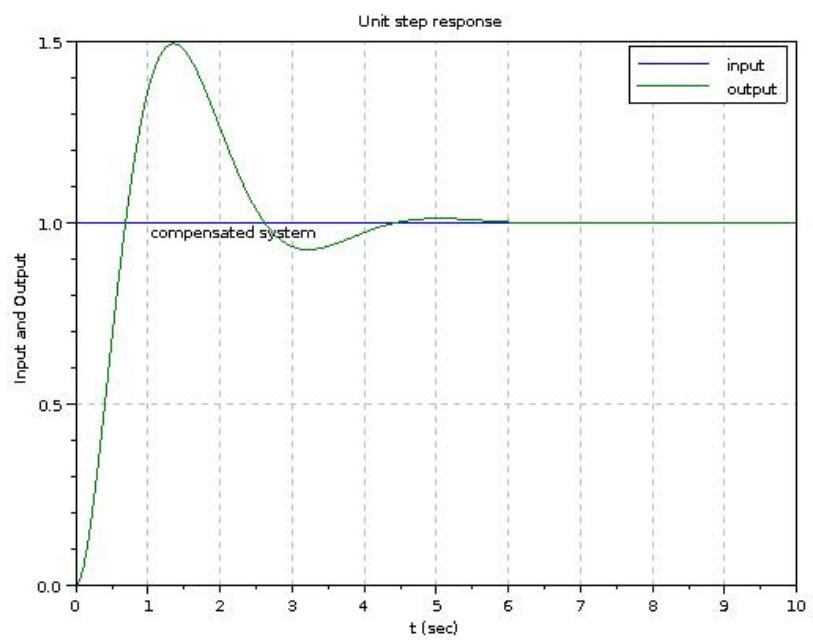


Figure 6.6: Lead Compensator Design Attempt 1

check Appendix AP 2 for dependency:

`plotresp.sci`

check Appendix AP 7 for dependency:

`rootl.sci`

Scilab code Exa 6.a.13.2 Lead Compensator Design Attempt 2

```
1 // Example A-6-13-2
2 // Lead Compensator Design Attempt 2
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 // please edit the path
8 // cd "/<your code directory>/";
9 // exec("rootl.sci");
10 // exec("plotresp.sci");
11
12 s = %s;
13 G = syslin('c', 1, s^2);
14 H = syslin('c', 1, 0.1*s + 1);
15
16 R = [-1 -1];
17 I = [1.73205 -1.73205];
18 dp = R(1) + %i*I(1);
19
20 subplot(1,2,1);
21 rootl(G*H, [-15 -15; 5 15], 'Root locus plot for
uncompensated system');
22 plot(R, I, 'x');
23 angdef = 180 - phasemag(horner(G*H, dp));
24 disp(angdef, 'angle deficiency =');
```

```

26 z = 3; // zeros at -3;
27 p = 1.73205 / tand(40.89334 - angdef/2) + 1 ; disp(p
    , 'p =' );
28 Gc = ((s + z) / (s + p)) ^2;
29 disp(Gc, 'lead compensator =' );
30
31 Kc = abs(1/ horner(G*Gc*H,dp));
32 disp(Kc, 'Kc =' );
33 O = Kc*Gc*G*H; disp(O, 'open loop Transfer function
    =' );
34 C = Kc*Gc*G /. H; disp(C, 'closed loop Transfer
    function =' );
35 disp(roots(C.den), 'closed loop poles =' );
36
37 subplot(1,2,2);
38 rootl(O,[-15 -15; 5 15], 'Root locus plot for
    compensated system');
39 plot(R,I, 'x');
40
41 scf();
42 t = 0:0.05:10;
43 u = ones(1,length(t)); //step response
44 plotresp(u,t,C, 'Unit step response');
45 xstring(1,0.95, 'compensated system');

```

check Appendix AP 2 for dependency:

`plotresp.sci`

check Appendix AP 7 for dependency:

`rootl.sci`

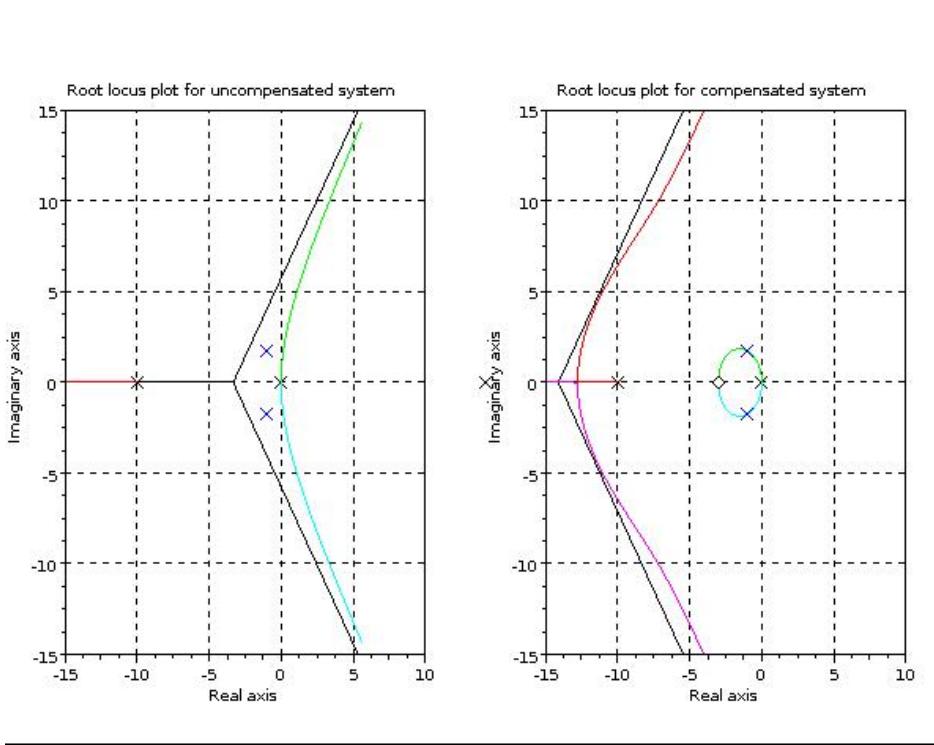


Figure 6.7: Lead Compensator Design Attempt 2

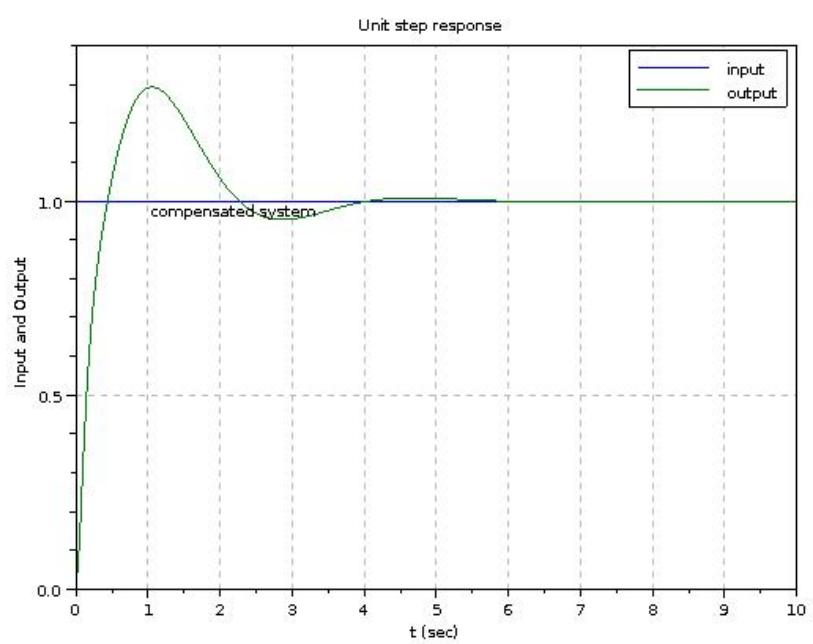


Figure 6.8: Lead Compensator Design Attempt 2

Scilab code Exa 6.a.17 Design of lag lead compensator

```
1 // Example A-6-17
2 // Design of lag lead compensator
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6 mode(0);
7
8 // please edit the path
9 // cd "/<your code directory>/";
10 // exec("rootl.sci");
11 // exec("plotresp.sci");
12
13 s = %s;
14 G = syslin('c', 1 , s * (s + 1) * (s + 5));
15
16 Kv = 50; // desired velocity constant
17 disp(horner(s*G,0), 'Kv (uncompensated system) = ');
18
19 // designing lead part
20 Kc = Kv / abs(horner(s*G,0))
21 z1 = 1 //to cancel the pole s = -1 of the plant
22
23 _beta = 16.025; disp(_beta, 'beta = ');
24 x = 1.9054 // beta and x are found analytically
25
26 dp = -x + sqrt(3)*%i*x
27 R = [-x -x]; I = [imag(dp) -imag(dp)];
28 p1 = z1 * _beta
29
30 Gc1 = Kc * (s + z1)/(s + p1); disp(Gc1, 'Lead
    compensator Gc1 = ');
31
32 // Lag compensator design
33 p2 = 0.01 //say
34 z2 = p2 * _beta
35 Gc2 = (s + z2)/(s + p2);
```

```

36 disp(Gc2, 'Lag compensator Gc2 =');
37 disp(abs(horner(Gc2,dp)), 'magnitude contribution of
      lag part =');
38 disp(phasemag(horner(Gc2,dp)), 'angle contribution of
      lag part =');
39 // these are acceptable
40
41 Gc = Gc1 * Gc2
42 H = G * Gc;           // compensated system
43 H = syslin('c',numer(H),denom(H));
44
45 subplot(1,2,1);
46 rootl(G,[-20 -15; 10 15], 'Uncompensated system');
47 plot(R,I,'x');
48 xgrid(color('gray'));
49 subplot(1,2,2);
50 rootl(H,[-20 -15; 10 15], 'Compensated system');
51 plot(R,I,'x');
52 xgrid(color('gray'));
53 xstring(R(1),I(1), 'Desired closed loop poles');
54
55 G1 = syslin('c',G /. 1);
56 C = syslin('c',H /. 1);    // final closed loop
      system
57 disp(C, 'closed loop system =');
58 disp(roots(C.den), 'closed loop poles = ');
59 disp(horner(s*H,0), 'velocity error constant Kv =')
60
61 scf();
62 subplot(2,1,1);
63 t = 0:0.05:10;
64 u = ones(1,length(t));
65 plotresp(u,t,G1, '');
66 plotresp(u,t,C, 'Unit step response');
67 xstring(1,0.1, 'uncompensated system');
68 xstring(0.7,1.12, 'compensated system');
69
70 subplot(2,1,2);

```

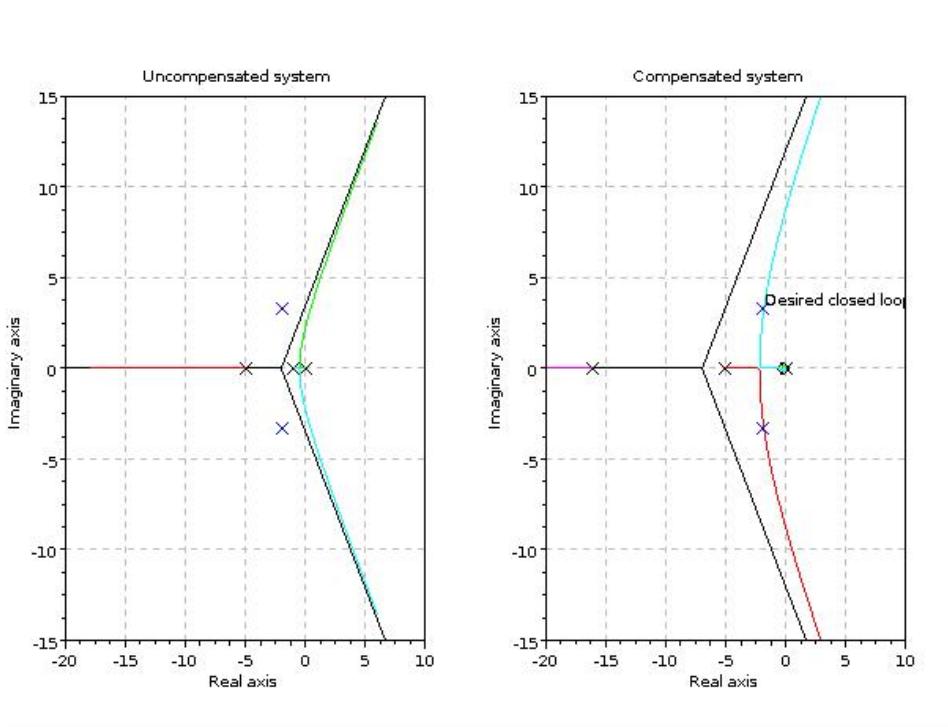


Figure 6.9: Design of lag lead compensator

```

71 plotresp(t,t,G1,'');
72 plotresp(t,t,C,'Unit ramp response');
73 xstring(3,0.9,'uncompensated system');
74 xstring(0.7,2,'compensated system');

```

check Appendix AP 2 for dependency:

`plotresp.sci`

check Appendix AP 7 for dependency:

`rootl.sci`

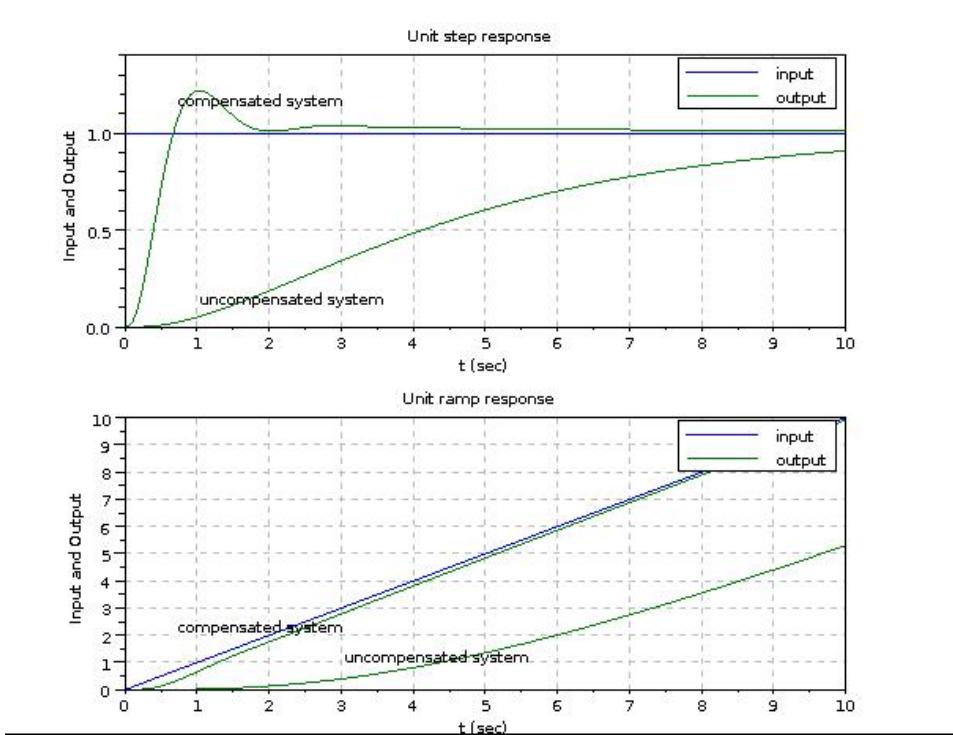


Figure 6.10: Design of lag lead compensator

Scilab code Exa 6.a.18 Design of a compensator for a highly oscillatory system

```
1 // Example A-6-18
2 // Design of a compensator for an highly
   oscillatory system
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6 mode(0);
7
8 // please edit the path
9 // cd "/<your code directory>/";
10 // exec("rootl.sci");
11 // exec("plotresp.sci");
12
13 s = %s;
14 G = syslin('c', 2*s + 0.1, s * (s^2 + 0.1*s + 4));
15
16 R = [-2 -2];
17 I = 2*sqrt(3) * [1 -1];
18 dp = R(1) + %i*I(1)
19
20 // Cancel the zero at -0.1
21 Gc2 = (s + 4)/(2*s + 0.1)
22 G1 = G*Gc2
23
24 angdef = 180 - phasemag(horner(G1, dp));
25 disp(angdef, 'angle deficiency =')
26
27 // Designing two lead compensators in series
28 angdefby2 = angdef / 2
29 z = 2 // say
30 p = 2 + 2 * sqrt(3) * cotd(90 - angdefby2)
```

```

31
32 Gc1 = ((s + z)/(s + p))^2
33 G2 = Gc1 * G1;
34 Kc = 1 / abs(horner(G2, dp))
35 Gc = Kc * Gc1 * Gc2
36
37 H = Kc * G2; disp(H, 'Gc*G = ');
38 C = H /. 1; disp(C, 'closed loop Transfer function
= ');
39 disp(roots(C.den), 'closed loop poles = ');
40
41 subplot(1,2,1);
42 rootl(G, [-15 -15; 15 15], 'Root locus plot for
uncompensated system');
43 plot(R,I,'x');
44 xgrid(color('gray'));
45 subplot(1,2,2);
46 rootl(H, [-15 -15; 15 15], 'Root locus plot for
compensated system');
47 plot(R,I,'x');
48 xgrid(color('gray'));
49
50 scf();
51 subplot(2,1,1);
52 t = 0:0.02:5;
53 u = ones(1,length(t)); //step response
54 plotresp(u,t,C, 'Unit step response of the
compensated system');
55
56 subplot(2,1,2);
57 t = 0:0.02:8;
58 plotresp(t,t,C, 'Unit step response of the
compensated system');

```

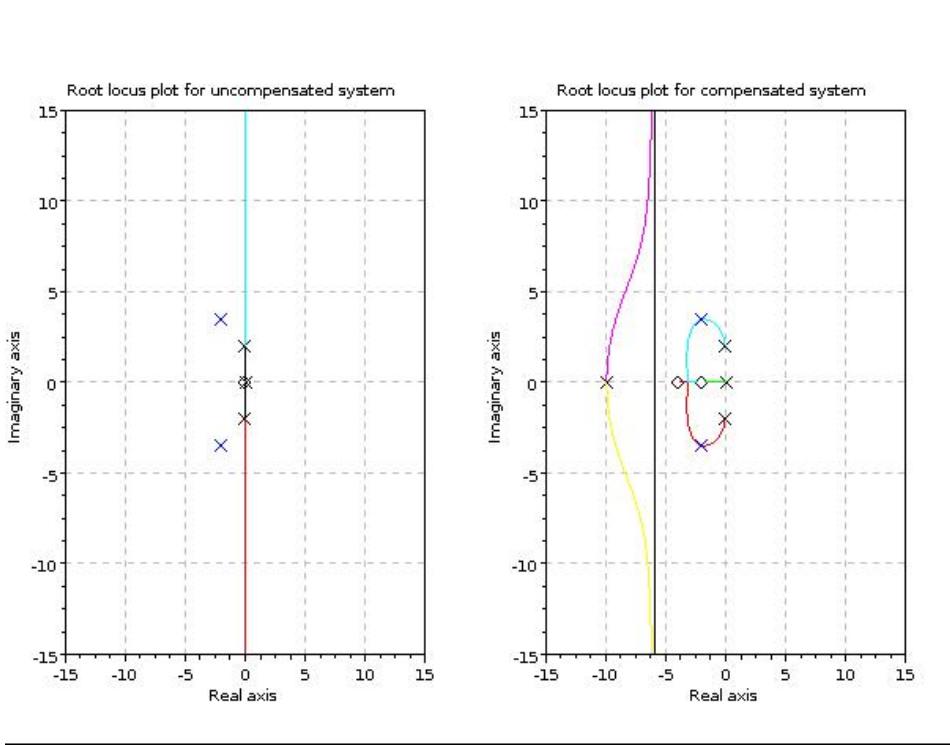


Figure 6.11: Design of a compensator for a highly oscillatory system

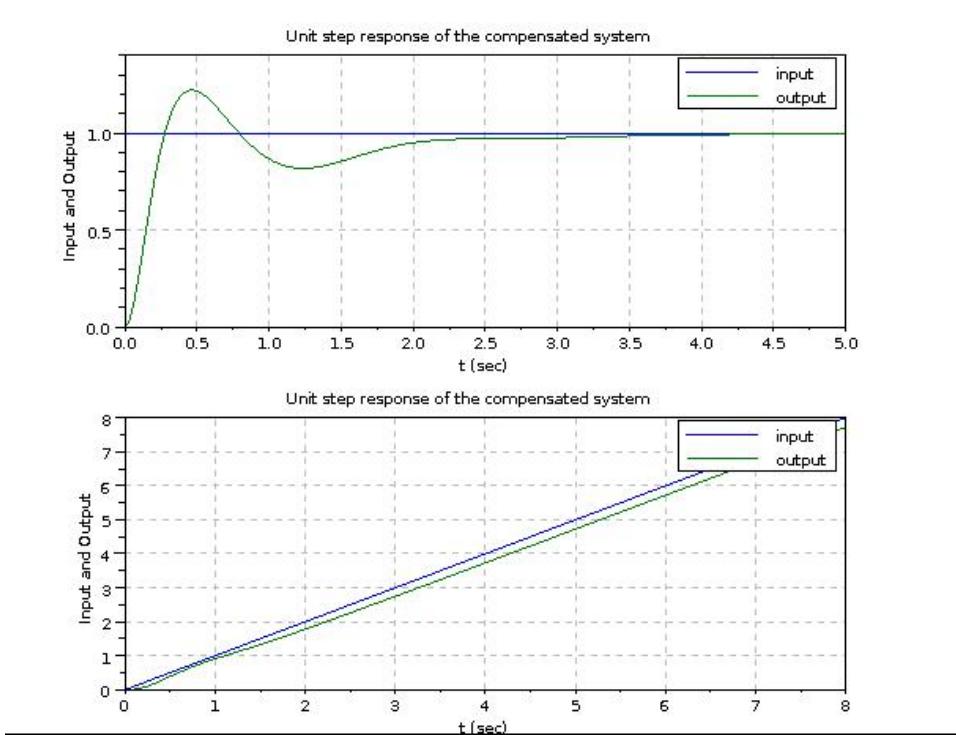


Figure 6.12: Design of a compensator for a highly oscillatory system

check Appendix AP 2 for dependency:

`plotresp.sci`

check Appendix AP 7 for dependency:

`rootl.sci`

check Appendix AP 7 for dependency:

`rootl.sci`

Scilab code Exa 6.1 Root Locus

```
1 // Example 6-1
2 // Root Locus
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 // please edit the path
8 // cd "/<your code directory>/";
9 // exec("rootl.sci");
10
11 s = %s;
12 D = s * (s + 1) * (s + 2);
13 H = syslin('c',1,D);
14
15 rootl(H,[-4 -3; 2 3], 'Root locus of G(s) = 1/(s*(s +
1)*(s + 2))');
```

Scilab code Exa 6.2 Root Locus

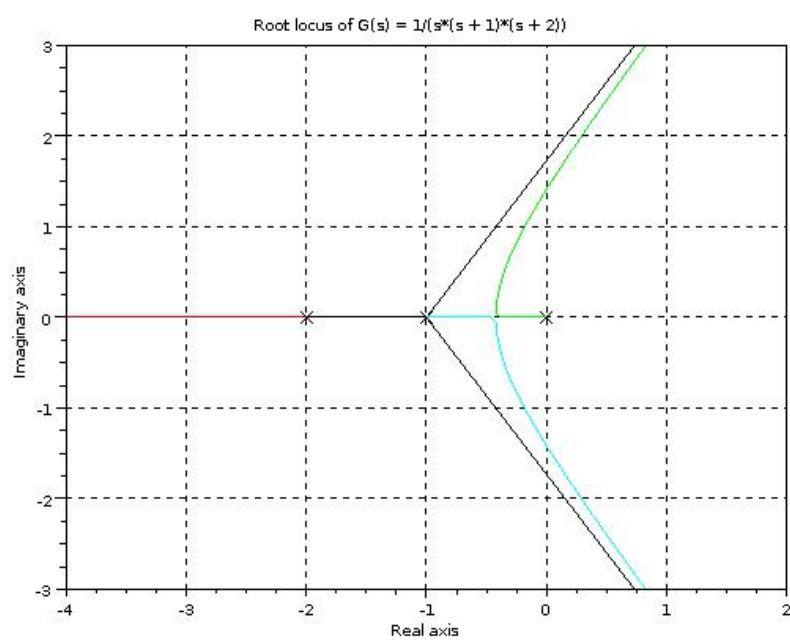


Figure 6.13: Root Locus

```

1 // Example 6-2
2 // Root Locus
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 s = %s;
8 H = syslin('c', s + 2, s^2 + 2*s + 3);
9
10 evans(H,10);
11 xgrid();
12 a = gca();
13 a.box = "on";
14 a.data_bounds = [-6 -3; 2 3];
15 a.children(1).visible = 'off';
16 xtitle('Root locus of G(s) = (s + 2)/ (s^2 + 2*s +
3)');

```

check Appendix AP 7 for dependency:

`rootl.sci`

Scilab code Exa 6.3 Root Locus

```

1 // Example 6-3
2 // Root locus
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 // please edit the path
8 // cd "/<your code directory>/";
9 // exec("rootl.sci");
10

```

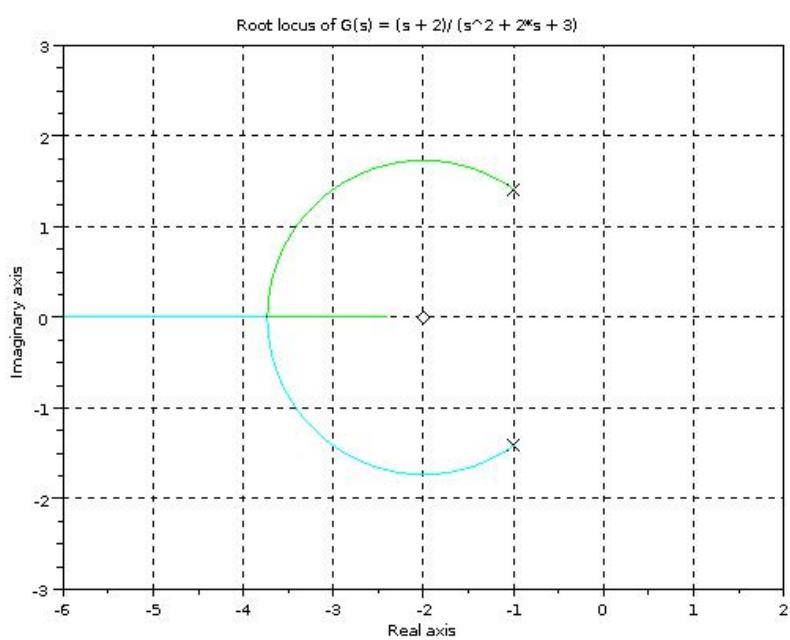


Figure 6.14: Root Locus

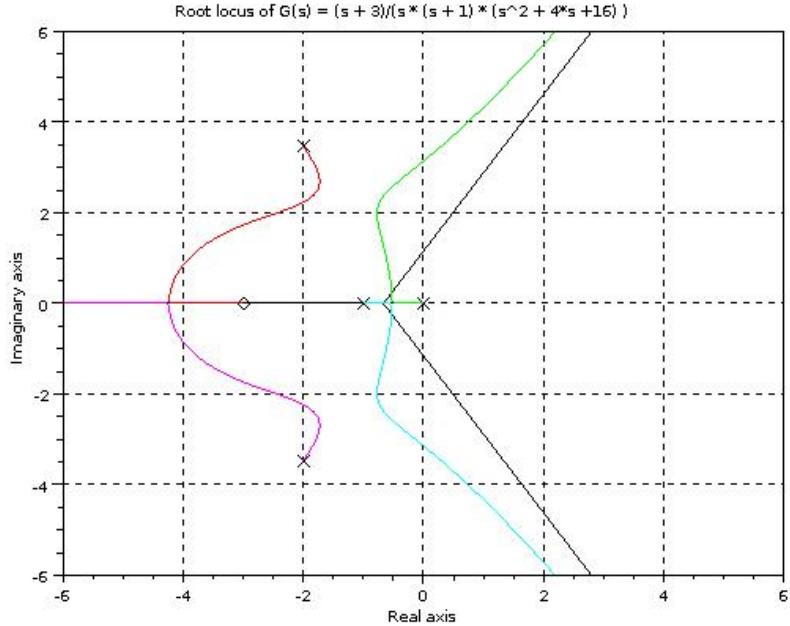


Figure 6.15: Root Locus

```

11 s = %s;
12 N = s + 3;
13 D = s * (s + 1) * (s^2 + 4*s + 16);
14 H = syslin('c',N,D);
15 disp(roots(D), 'open loop poles = ');
16 disp(roots(N), 'open loop zeros = ');
17
18 rootl(H, [-6 -6; 6 6], 'Root locus of G(s) = (s + 3)/(
    s * (s + 1) * (s^2 + 4*s + 16) )');

```

check Appendix AP 7 for dependency:

`rootl.sci`

Scilab code Exa 6.4 Root Locus

```
1 // Example 6-4
2 // Root locus
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 // please edit the path
8 // cd "/<your code directory>/";
9 // exec("rootl.sci");
10
11 s = %s;
12 D = s*(s + 0.5)*(s^2 + 0.6*s + 10);
13 H = syslin('c', 1, D);
14 disp(roots(D), 'open loop poles =');
15
16 rootl(H, [-6 -6; 6 6], 'Root locus of G(s) = 1/(s*(s
+ 0.5)*(s^2 + 0.6*s + 10))');
```

check Appendix AP 7 for dependency:

rootl.sci

Scilab code Exa 6.5 Root locus of system in state space

```
1 // Example 6_5
2 // Root locus of system in state space
3
4 clear; clc;
```

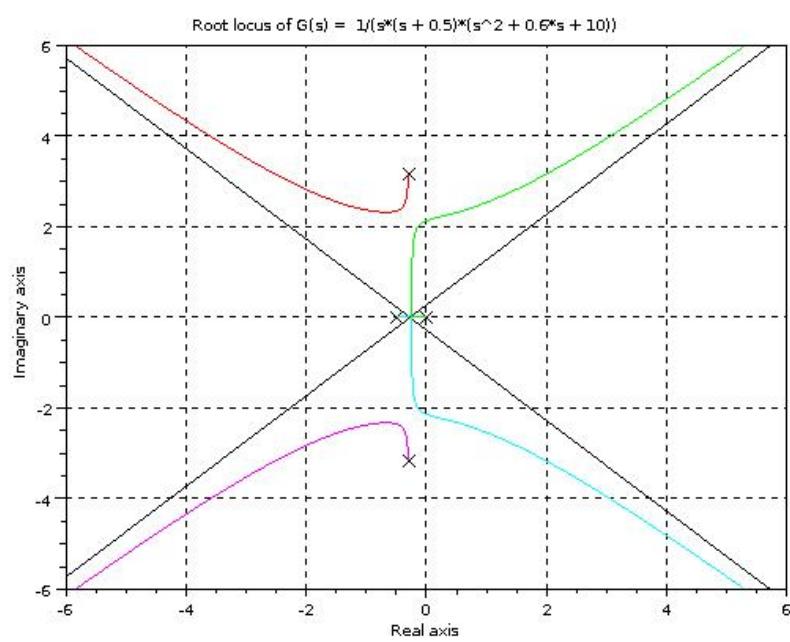


Figure 6.16: Root Locus

```

5 xdel(winsid()); // close all windows
6
7 // please edit the path
8 // cd "/<your code directory>/";
9 exec("rootl.sci");
10
11 A = [0 1 0; 0 0 1; -160 -56 -14];
12 B = [0; 1; -14];
13 C = [1 0 0];
14 D = [0];
15 G = syslin('c',A,B,C,D);
16 H = clean(ss2tf(G));
17 disp(H, ' transfer function = ');
18
19 rootl(G, [-20 -20; 20 20], 'Root locus plot of State
Space model');

```

Scilab code Exa 6.6.1 Design of a lead compensator using root locus

```

1 // Example 6-6-1
2 // Design of a lead compensator using root locus
3
4
5 clear; clc;
6 xdel(winsid()); // close all windows
7
8 // please edit the path
9 // cd "/<your code directory>/";
10 // exec("rootl.sci");
11
12 s = %s;
13 G = syslin('c',10 , s*(s+1) ); //open loop system
14

```

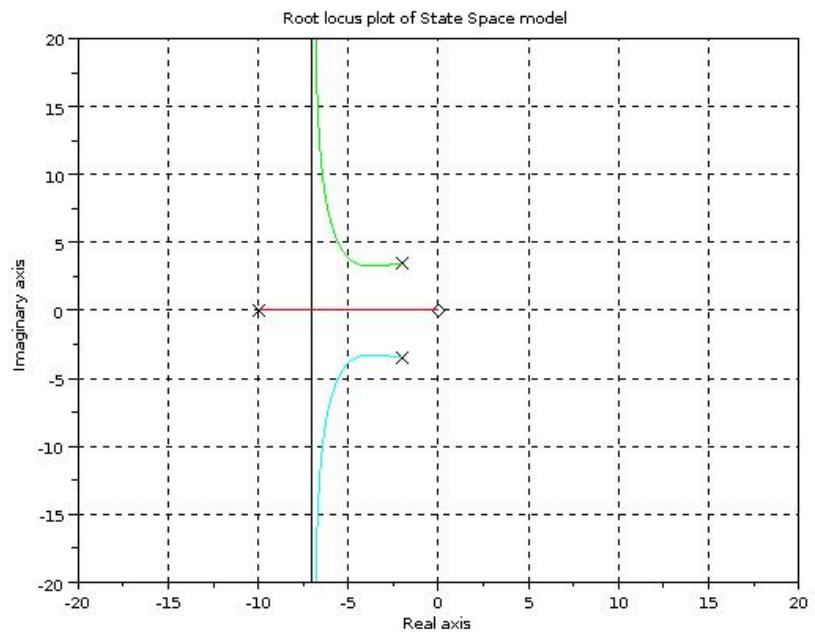


Figure 6.17: Root locus of system in state space

```

15 R = [ -1.5 -1.5];
16 I = [2.5981 -2.5981]; // desired closed loop poles
17 dp = R(1) + %i*I(1);
18
19 rootl(G,[-5 -5; 1 5], 'Uncompensated system');
20 xgrid(color('gray'));
21 plot(R,I,'x'); // A gain adjustment is not enough
as the
22 // desired poles do not lie on the
roor locus
23
24 [phi1 db] = phasemag(horner(G,dp));
25 angdef = 180 - phi1;
26 disp(angdef, 'Angle deficiency = ');
27
28 // Lead compensator for Maximum Kv
29 // here we will find the pole-zero of the
compensator
30 // using the prescirbed method
31
32 [phi2 dbi] = phasemag(dp);
33 angOPA = phi2;
34 angPOD = 180 - phi2;
35 angOPD = (angOPA - angdef) / 2;
36 angOPC = (angOPA + angdef) / 2;
37
38 angPDO = (180 - angPOD - angOPD);
39 angPCO = (180 - angPOD - angOPC);
40
41 // using the sine rule of triangles
42 D0 = sind(angOPD) * abs(dp) / sind(angPDO);
43 C0 = sind(angOPC) * abs(dp) / sind(angPCO);
44
45 Gc = (s + D0)/(s + C0);
46 disp(Gc, 'compensator = ');
47 H = G.num * Gc / G.den ; // compensated
system
48 H = syslin('c',numer(H),denom(H));

```

```

49
50 scf();
51 rootl(H, [-5 -5; 1 5], 'Compensated system');
52 xgrid(color('gray'));
53 plot(R,I,'x');
54
55 // Final system passes through the desired poles
56 // required gain for the system
57 Kc = abs(1 / horner(H,dp));
58 disp(Kc, 'required gain Kc = ');
59 C = H*Kc /. 1;           // final closed loop system
60 disp(C, 'closed loop system = ');
61 disp(roots(C.den), 'closed loop poles = ');
62 disp(horner(s*H*Kc,0), 'velocity error constant Kv = '
)

```

check Appendix AP 7 for dependency:

`rootl.sci`

Scilab code Exa 6.6.2 Step and ramp response of lead compensated systems

```

1 // Example 6-6-2
2 // Step and ramp response of lead compensated
   systems
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 function Gc = leadcomp(Kc,z,p);
8   Gc = Kc* ((s + z)/(s + p));

```

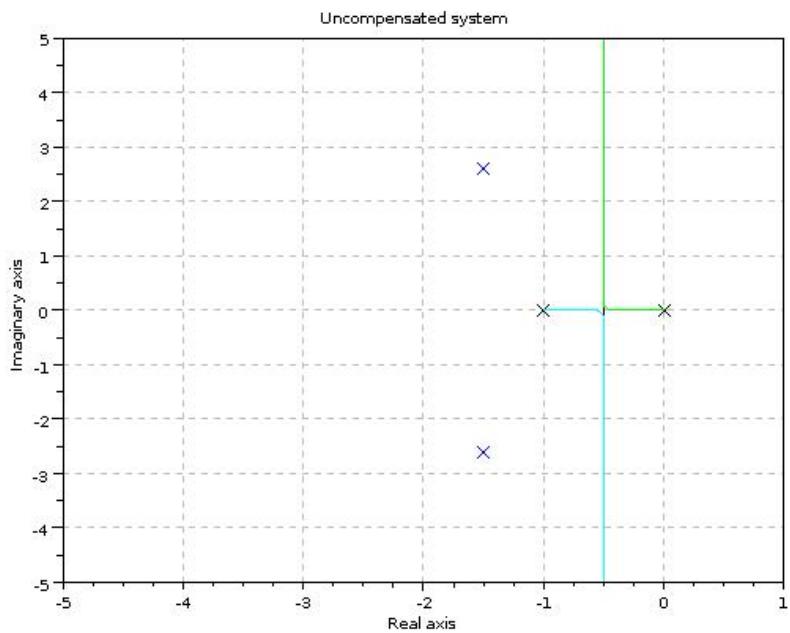


Figure 6.18: Design of a lead compensator using root locus

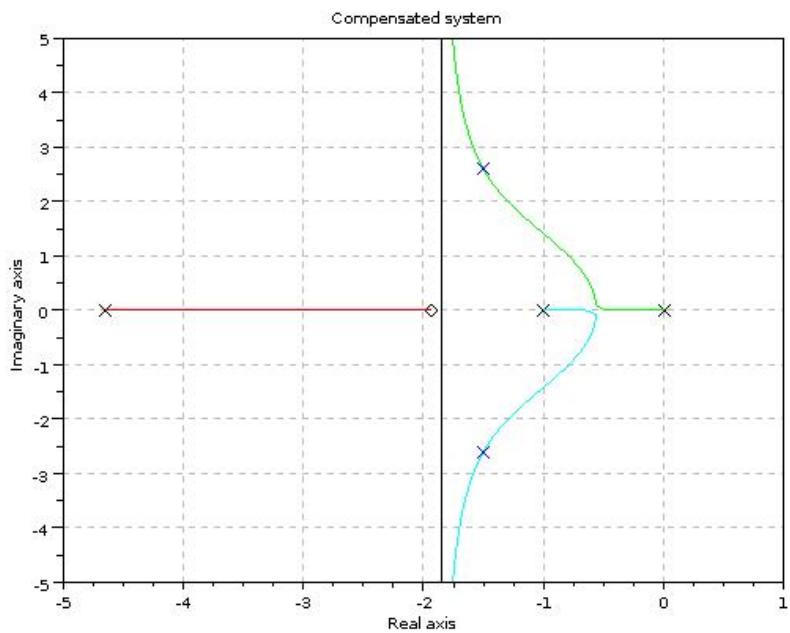


Figure 6.19: Design of a lead compensator using root locus

```

9  endfunction
10
11 function plotall(u,t,text)
12     y    = csim(u,t,H );
13     yc1 = csim(u,t,H1);
14     yc2 = csim(u,t,H2);
15
16     plot(t,y,t,yc1,t,yc2);
17     xgrid(color('gray'));
18     xtitle(text + ' Response of compensated and
19         uncompensated systems ','t sec ','Output');
20     legend('Uncompensated System','Compensated System
21             Method 1','Compensated System Method 2');
22 endfunction
23
24 s = %s;
25 G = 10 / ( s*(s+1) ); //open loop system
26 Gc1 = leadcomp(1.2292,1.9373,4.6458);
27 Gc2 = leadcomp(0.9,1,3);
28
29 H = syslin('c',G /. 1);
30 H1 = syslin('c', ( G * Gc1 ) /. 1);
31 H2 = syslin('c', ( G * Gc2 ) /. 1);
32
33 t = 0:0.05:5;
34 u = ones(1,length(t));
35 plotall(u,t,'Step');scf();
36 t = 0:0.05:9;
37 plotall(t,t,'Ramp');
38 plot(t,t,'k');

```

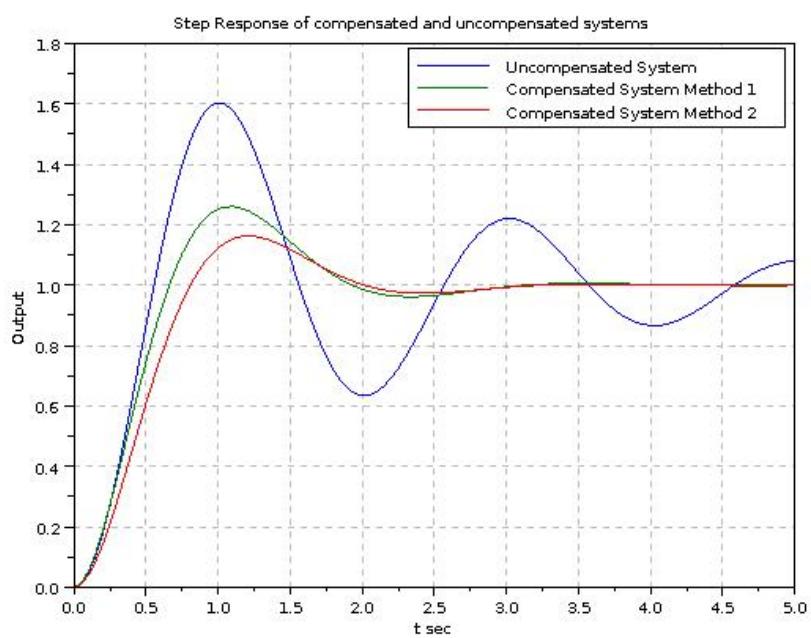


Figure 6.20: Step and ramp response of lead compensated systems

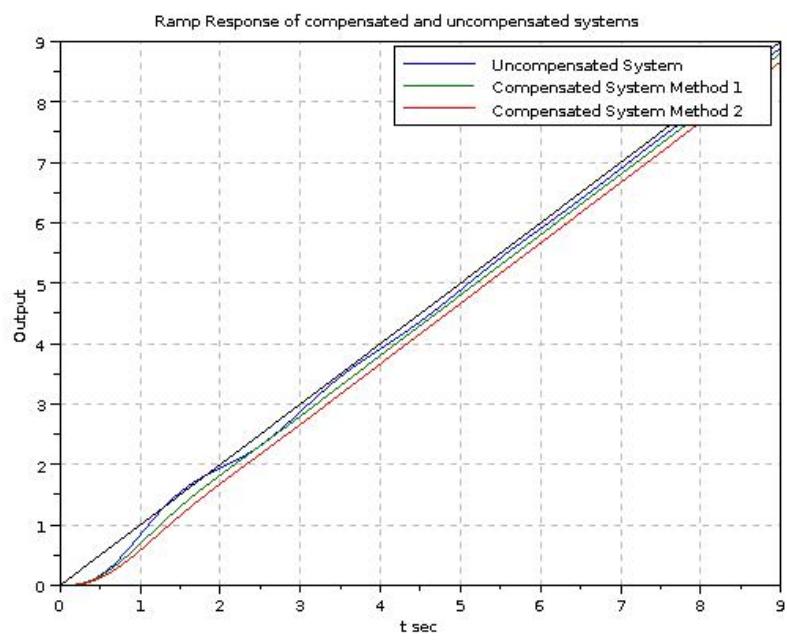


Figure 6.21: Step and ramp response of lead compensated systems

Scilab code Exa 6.7.1 Design of a lag compensator using root locus

```
1 // Example 6-7-1
2 // Design of a lag compensator using root locus
3
4
5 clear; clc;
6 xdel(winsid()); // close all windows
7
8 // please edit the path
9 // cd "/<your code directory>/";
10 // exec("rootl.sci");
11
12 s = %s;
13 G = syslin('c', 1.06, s*(s+1)*(s+2)); //open loop
    system
14 R = [-0.31 -0.31];
15 I = [0.55 -0.55]; // desired closed loop poles
16 dp = R(1) + %i*I(1);
17 disp(roots(G.den + 1.06), 'Closed loop poles (
    uncompensated)=');
18 disp(horner(s*G, 0), 'Kv (uncompensated system = ');
19
20 rootl(G, [-3 -2; 1 2], '');
21 plot(R, I, 'x');
22
23 // Lag compensator for Kv = 5 sec.
24
25 _beta = 10; // taking beta as 10
26 z = 0.05;
27 p = z / _beta;
28
29 Gc = (s + z)/(s + p);
30 disp(Gc, 'compensator = ');
31 H = G.num * Gc / G.den; // compensated
    system
32 H = syslin('c', numer(H), denom(H));
33
```

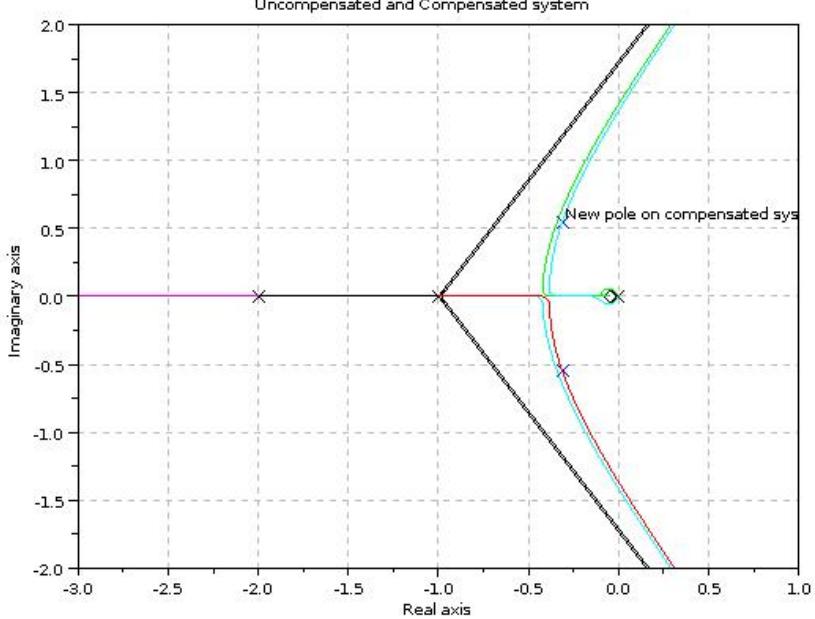


Figure 6.22: Design of a lag compensator using root locus

```

34 rootl(H,[-3 -2; 1 2], 'Uncompensated and Compensated
    system');
35 xgrid(color('gray'));
36 xstring(R(1),I(1),'New pole on compensated sys');
37
38 Kc = abs(1 / horner(H,dp));
39 disp(Kc,'required controller gain Kc = ');
40 C = H*Kc /. 1;           // final closed loop system
41 disp(C,'closed loop system =');
42 disp(roots(C.den),'closed loop poles =');
43 disp(horner(s*C,0),'velocity error constant Kv ='
    )

```

check Appendix AP 7 for dependency:

rootl.sci

Scilab code Exa 6.7.2 Step and ramp response of lag compensated system

```
1 // Example 6-7-2
2 // Step and ramp response of lag compensated system
3
4 clear; clc;
5 xdel(winsid()); //close all windows
6
7 // please edit the path
8 // cd "/<your code directory>/";
9 // exec("plotresp.sci");
10
11 s = %s;
12 G = 1.06 / (s * (s + 1) * (s + 2));
13
14 Kc = 0.9956;
15 z = 0.05;
16 p = 0.005;
17 Gc = Kc * (s + z)/(s + p);
18 GGc = G*Gc;
19
20 H = syslin('c',G /. 1);
21 Hc = syslin('c',GGc /. 1);
22
23 t = 0:0.5:40;
24 u1 = ones(1,length(t)); //step response
25
26 subplot(2,1,1);plotresp(u1,t,H,'');
27 plotresp(u1,t,Hc,'Unit step response');
28 xstring(5,0.9,'uncompensated system');
29 xstring(0.1,1.2,'compensated system');
30
31 t = 0:0.5:50;
```

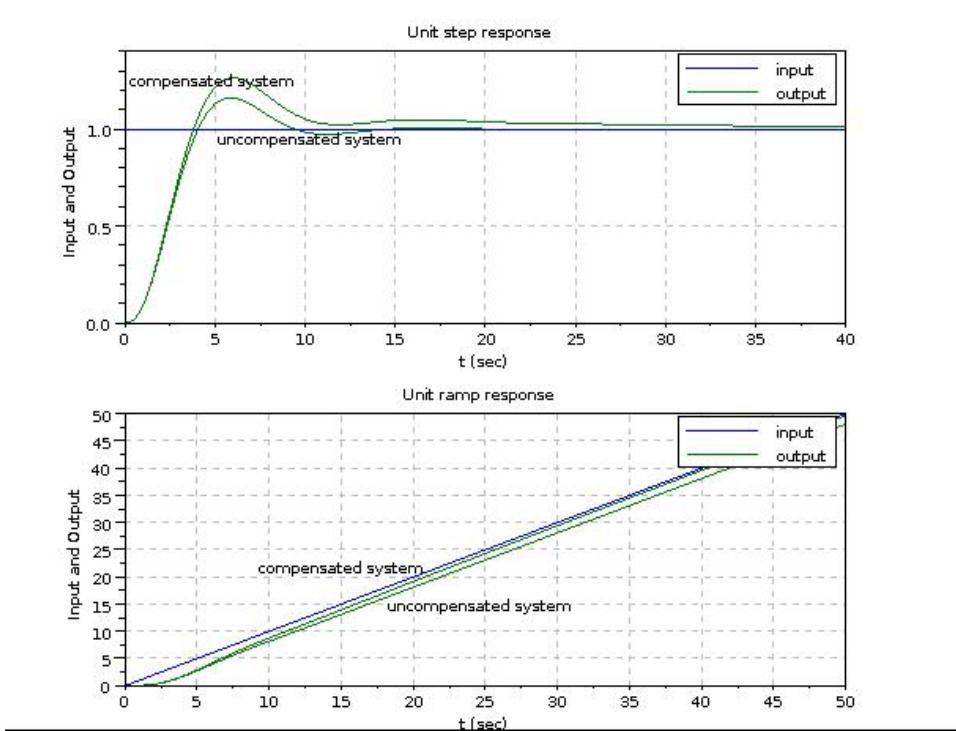


Figure 6.23: Step and ramp response of lag compensated system

```

32 u2 = t;                                //ramp response
33 subplot(2,1,2);plotresp(u2,t,H,'');
34 plotresp(u2,t,Hc,'Unit ramp response');
35 xstring(18,13,'uncompensated system');
36 xstring(9,20,'compensated system');

```

check Appendix AP 2 for dependency:

`plotresp.sci`

Scilab code Exa 6.8.1 Design of a lag lead compensator using root locus

```

1 // Example 6-8-1
2 // Design of a lag lead compensator using root locus
3 // zeta ~= gamma (not equal to)
4
5 clear; clc;
6 xdel(winsid()); //close all windows
7 // please edit the path
8 // cd "/<your code directory>/";
9 // exec("rootl.sci");
10
11 s = %s;
12 G = syslin('c', 4, s * (s + 0.5)); //open loop
    system
13
14 Kv = 80; // desired velocity constant
15 wn = 5; // desired natural frequency and
    damping
16 _zeta = 0.5;
17 sigma = -1*wn * _zeta;
18 wd = wn * sqrt(1 - _zeta^2);
19
20 dp = sigma + %i*wd; // desired closed loop poles
21 disp(roots(G.den + 4), 'Closed loop poles (
    uncompensated)=');
22 disp(horner(s*G, 0), 'Kv (uncompensated system = ');
23
24 rootl(G, [-5 -2; 1 2], 'Uncompensated system');
25 xgrid(color('gray'));
26 plot([sigma sigma], [wd -wd], 'x');
27 xstring(sigma, wd, 'Desired CL poles');
28
29 // Designing Lead Part
30 [phi1 db] = phasemag(horner(G, dp));
31 angdef = 180 - phi1;
32 disp(angdef, 'Angle deficiency = ');
33

```

```

34 z1 = 0.5 //Make the lead compensator zero cancel
            the system zero
35 // To determin p1;
36 // Gc1 = [0.5 +(-2.5 + 4.33j)] / [(p1 -2.5) + 4.33j]
37 [theta m2] = phasemag(-2.0 + 4.33*i);
38 p1 = 2.5 + 4.33*cotd(theta - angdef); // so that it
            contributes 'angdef'
39
40 Gc1 = (s + z1)/(s + p1);           disp(Gc1, 'Lead
            compensator Gc1 =');
41 _gamma = p1 / z1;                  disp(_gamma, 'gamma = ');
42 Kc = abs(1/horner(G*Gc1,dp));   disp(Kc, 'Kc = ');
43
44 // Lag compensator design
45 _beta = Kv * _gamma / Kc / horner(s*G,0); disp(_beta
            , 'beta');
46
47 T2 = 5; //say
48 z2 = 1 / T2; p2 = z2 / _beta;
49 Gc2 = (s + z2)/(s + p2);
50 disp(Gc2, 'Lag compensator Gc2 =');
51 disp(abs(horner(Gc2,dp)), 'magnitude contribution of
            lag part =');
52 disp(phasemag(horner(Gc2,dp)), 'angle contribution of
            lag part =');
53 // these are acceptable
54
55 Gc = Kc*Gc1*Gc2;
56 disp(Gc, 'final lag lead controller = ');
57 scf()
58 rootl(Gc*G,[-5 -2; 1 2], 'Compensated system');
59 xgrid(color('gray'));
60 plot([sigma sigma],[wd -wd], 'x');
61
62 C = Gc*G /. 1;
63 disp(C, 'closed loop system =');
64 disp(roots(C.den), 'closed loop poles = ');

```

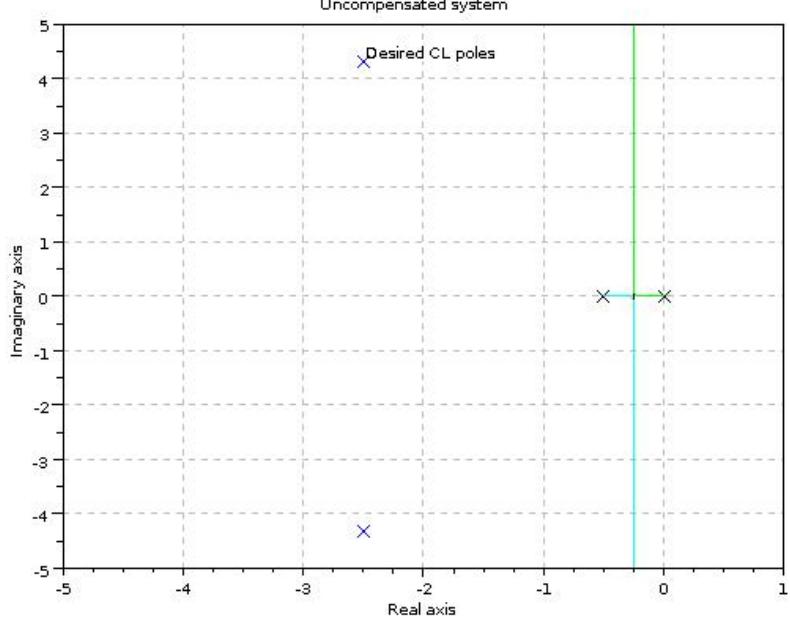


Figure 6.24: Design of a lag lead compensator using root locus

```
65 disp(horner(s*Gc*G,0), 'velocity error constant Kv =' )
)
```

check Appendix [AP 7](#) for dependency:

`rootl.sci`

Scilab code Exa 6.8.2 Evaluating Lag Lead compensated system

```
1 // Example 6-8-2
```

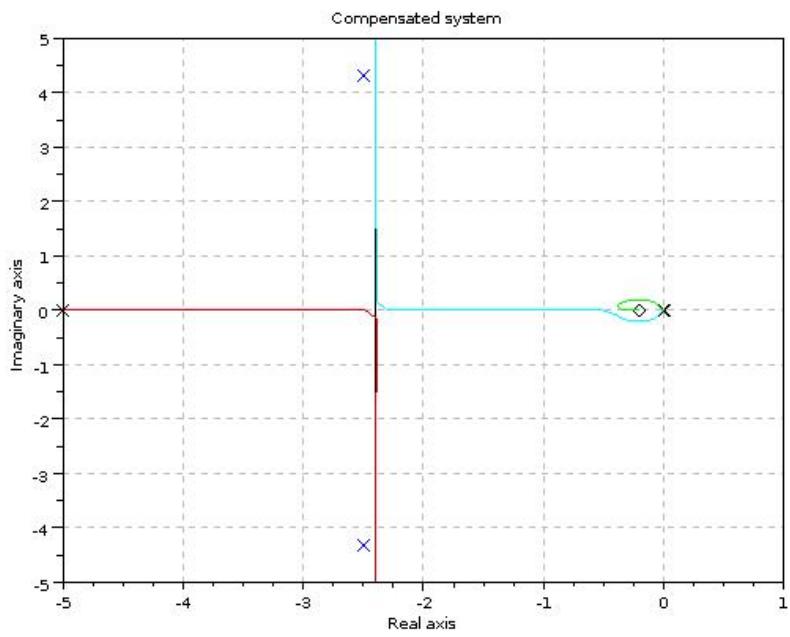


Figure 6.25: Design of a lag lead compensator using root locus

```

2 // Evaluating Lag Lead compensated system
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 // please edit the path
8 // cd "/<your code directory>/";
9 // exec("plotresp.sci");
10
11 s = %s;
12 G = 4 / (s * (s + 0.5));
13
14 Gc = 6.25 * (s + 0.5) * (s + 0.2) / (s + 5) / (s +
    0.125);
15 GGc = G*Gc;
16
17 H = syslin('c',G /. 1);
18 Hc = syslin('c',GGc /. 1);
19
20 t = 0:0.05:20;
21 u1 = ones(1,length(t)); // step response
22 plotresp(u1,t,H,'');
23 plotresp(u1,t,Hc,'Unit step response');
24 xstring(0.5,1.7,'uncompensated system');
25 xstring(1,0.95,'compensated system');
26
27 scf()
28 t = 0:0.05:10;
29 plotresp(t,t,H,'');
30 y2 = plotresp(t,t,Hc,'Unit ramp response');a = gca()
31 delete(a.children(2)); // deleting the drawn graph
    and redrawing
32 // with a different colour
33 plot(t,y2,'r');
34 legend('ramp input','uncompensated system',
    'compensated system');

```

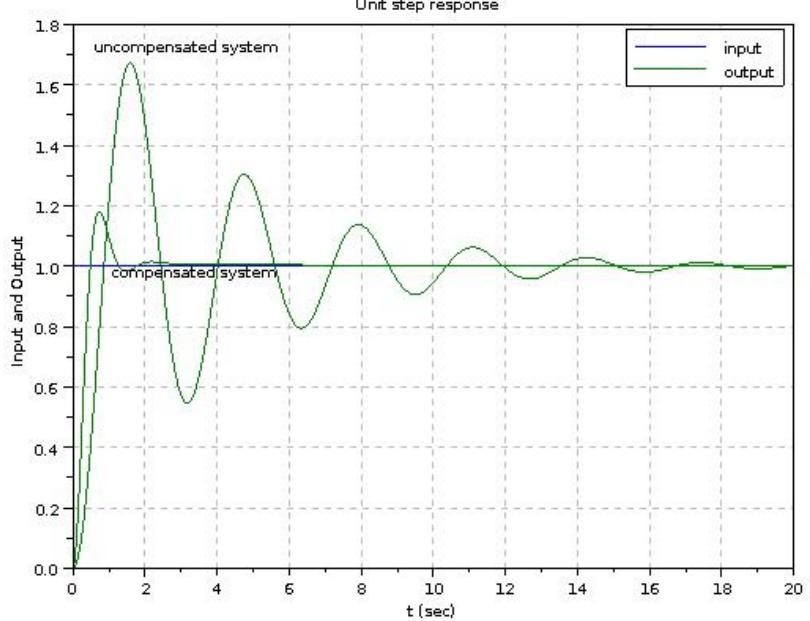


Figure 6.26: Evaluating Lag Lead compensated system

check Appendix AP 2 for dependency:

`plotresp.sci`

Scilab code Exa 6.9.1 Design of lag lead compensator using root locus 2

```

1 // Example 6-9-1
2 // Design of a lag lead compensator using root locus
  2

```

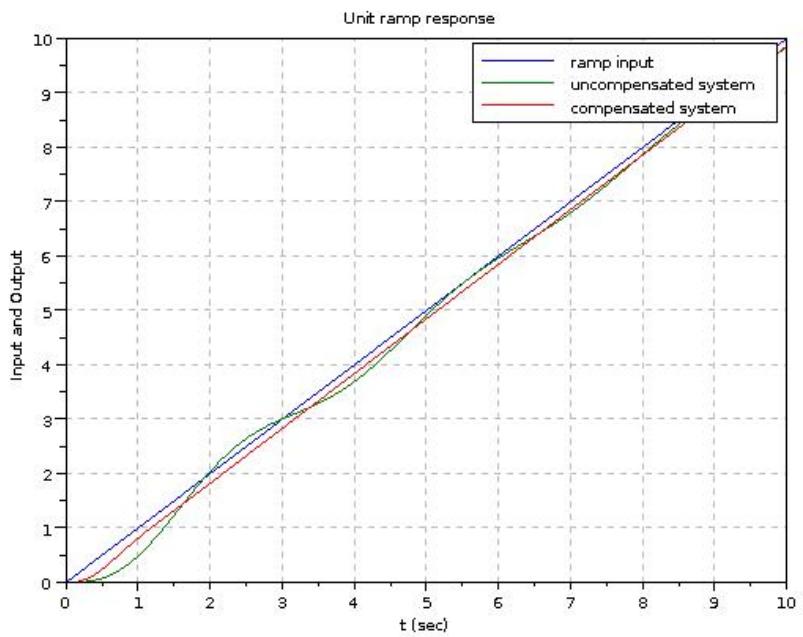


Figure 6.27: Evaluating Lag Lead compensated system

```

3 // gamma = beta case
4
5 clear; clc;
6 xdel(winsid()); //close all windows
7
8 // please edit the path
9 // cd "/<your code directory>/";
10 // exec("rootl.sci");
11
12 s = %s;
13 G = syslin('c',4 , s * (s + 0.5)); //open loop
    system
14
15 Kv = 80;           // desired velocity constant
16 wn = 5;           // desired natural frequency and
    damping
17 _zeta = 0.5;
18 sigma = -1*wn * _zeta;
19 wd = wn * sqrt(1 - _zeta^2);
20 dp = sigma + %i*wd; // desired closed loop poles
21 disp(roots(G.den + 4), 'Closed loop poles (
    uncompensated)=');
22 disp(horner(s*G,0), 'Kv (uncompensated system = ');
23
24 rootl(G,[-5 -2; 1 2], 'Uncompensated system');
25 xgrid(color('gray'));
26 plot([sigma sigma],[wd -wd], 'x');
27 xstring(sigma,wd, 'Desired CL poles');
28
29 // Designing Lead Part
30 Kc = Kv / horner(s*G,0); disp(Kc, 'Kc = ');
31 z1 = 2.38; //z1 and p1 determined graphically
32 p1 = 8.34;
33 T1 = 1 / z1;           disp(T1, 'T1');
34 _beta = T1 * p1;       disp(_beta, 'beta = ');
35
36 Gc1 =Kc * (s + z1)/(s + p1); disp(Gc1, 'Lead
    compensator Gc1 =');

```

```

37
38 // Lag compensator design
39 T2 = 10; //say
40 z2 = 1 / T2; p2 = z2 / _beta;
41 Gc2 = (s + z2)/(s + p2);
42 disp(Gc2, 'Lag compensator Gc2 =');
43 disp(abs(horner(Gc2,dp)), 'magnitude contribution of
    lag part =');
44 disp(phasemag(horner(Gc2,dp)), 'angle contribution of
    lag part =');
45 // these are acceptable
46
47 Gc = Gc1*Gc2;
48 disp(Gc, 'final lag lead controller =');
49 scf()
50 rootl(Gc*G, [-5 -2; 1 2], 'Compensated system');
51 xgrid(color('gray'));
52 plot([sigma sigma], [wd -wd], 'x');
53
54 C = Gc*G /. 1;
55 disp(C, 'closed loop system =');
56 disp(roots(C.den), 'closed loop poles =');
57 disp(horner(s*Gc*G,0), 'velocity error constant Kv =
    ')
58 disp(dp, 'desired poles =');

```

check Appendix AP 7 for dependency:

`rootl.sci`

Scilab code Exa 6.9.2 Evaluating Lag Lead compensated system

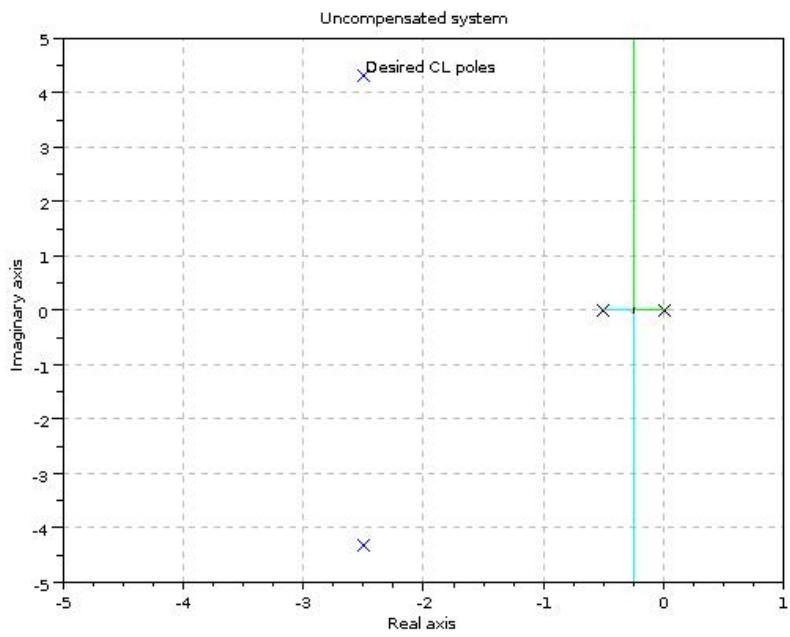


Figure 6.28: Design of lag lead compensator using root locus 2

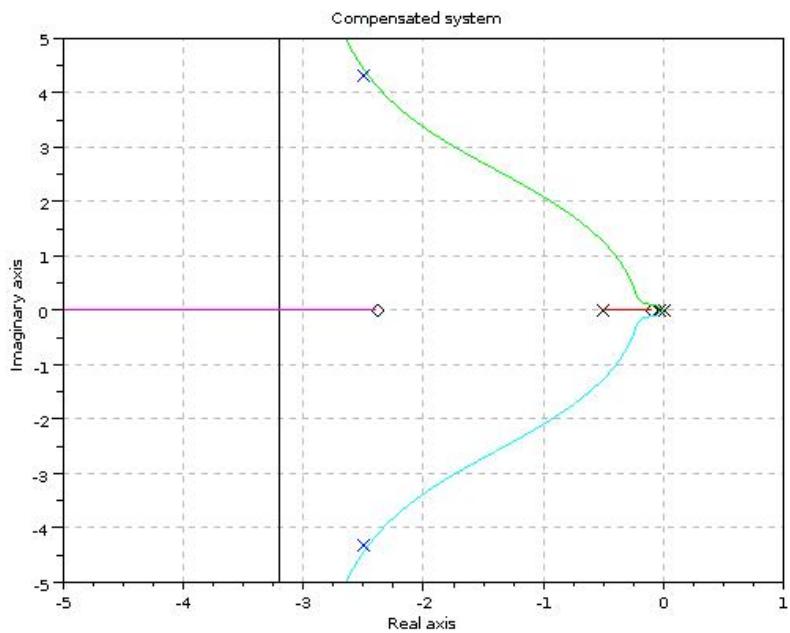


Figure 6.29: Design of lag lead compensator using root locus 2

```

1 // Example 6-9-2
2 // Evaluating Lag Lead compensated system
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 // please edit the path
8 // cd "/<your code directory>/";
9 // exec("plotresp.sci");
10
11 s = %s;
12 G = 4 / (s * (s + 0.5));
13
14 Gc = 10 * (s + 2.38) * (s + 0.1) / (s + 8.34) / (s +
    0.0285);
15 GGc = G*Gc;
16
17 H = syslin('c',G /. 1);
18 Hc = syslin('c',GGc /. 1);
19
20 t = 0:0.05:20;
21 u1 = ones(1,length(t)); //step response
22 plotresp(u1,t,H,'');
23 plotresp(u1,t,Hc,'Unit step response');
24 xstring(0.5,1.7,'uncompensated system');
25 xstring(1,0.95,'compensated system');
26
27 scf()
28 t = 0:0.05:10;
29 plotresp(t,t,H,'');
30 plotresp(t,t,Hc,'Unit ramp response');
31 xstring(1.4,0.9,'uncompensated system');
32 xstring(0,1.5,'compensated system');

```

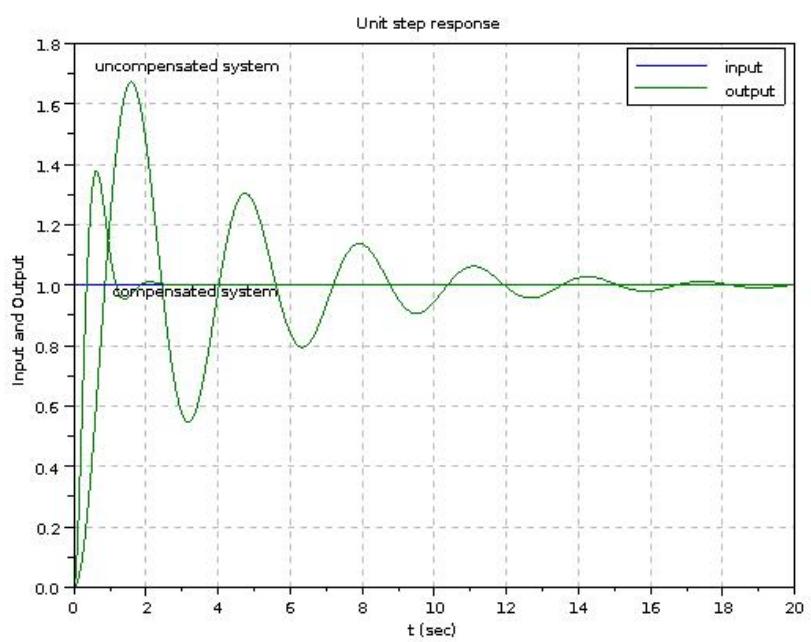


Figure 6.30: Evaluating Lag Lead compensated system

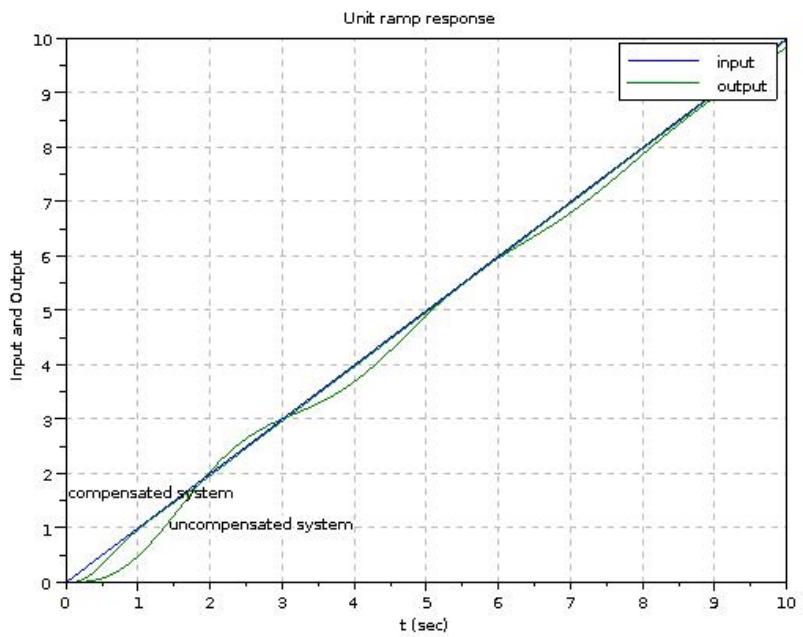


Figure 6.31: Evaluating Lag Lead compensated system

check Appendix AP 2 for dependency:

plotresp.sci

check Appendix AP 2 for dependency:

plotresp.sci

Scilab code Exa 6.10 Design of parallel compensation by root locus

```
1 // Example 6-10
2 // Design of parallel compensation by root locus
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 // please edit the path
8 // cd "/<your code directory>/";
9 // exec("plotresp.sci");
10
11 function [G,C] = getsystem(K)
12     G = 20 / ( s*(s+1)*(s+4) + K*s ); //open loop
13     system
14     C = syslin('c',G /. 1); // closed loop system
15 endfunction
16
17 s = %s;
18
19 // Root locus of the denominator polynomial (
20 // modified)
21 H = syslin('c',s , s^3 + 5*s^2 + 4*s + 20);
22 evans(H);
23 a= gca();a.children(1).visible = 'off';
24 sgrid([0.4],[]); // draw zeta = 0.4 line
25 a.box = "on";
26 a.data_bounds = [-6 -6;1 6];
```

```

25 xgrid(color('gray'));
26
27 r = [ -2.1589 ; -1.049 ]; i =[4.9652; 2.4065];
28 p = r + %i * i;
29 K = [1; 1] ./ abs(horner(H,p));
30 plot(r,i,'.');
31 xstring(r,i,['K =' + string(K(1)), 'K =' + string(K
    (2))] );
32
33 k = K ./ 20;
34 disp([K k], 'K : k = ');
35 [G1 C1] = getsystem(K(1));
36 [G2 C2] = getsystem(K(2));
37
38 disp(roots(C1.den), 'closed loop poles of system with
    k = ' + string(k(1)));
39 disp(roots(C2.den), 'closed loop poles of system with
    k = ' + string(k(2)));
40 disp(C1, 'C1 ='); disp(C2, 'C2 =');
41
42 scf();
43 t = 0:0.05:10;
44 u = ones(1,length(t));
45 plotresp(u,t,C1,'');
46 plotresp(u,t,C2,'Step response of parallel
    compensated systems');
47 xstring(1.3,1.1,'k = ' + string(k(1)));
48 xstring(2,0.8,'k = ' + string(k(2)));

```

check Appendix AP 2 for dependency:

`plotresp.sci`

check Appendix AP 7 for dependency:

`rootl.sci`

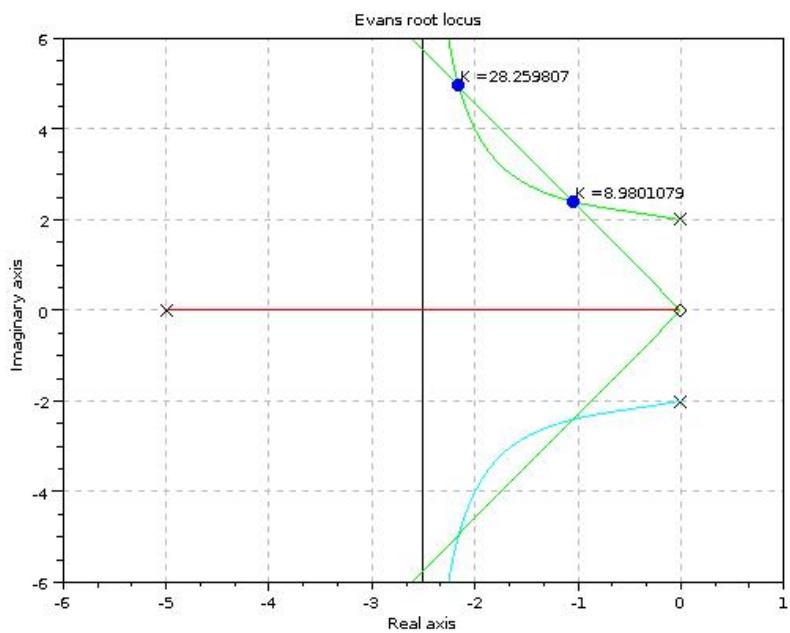


Figure 6.32: Design of parallel compensation by root locus

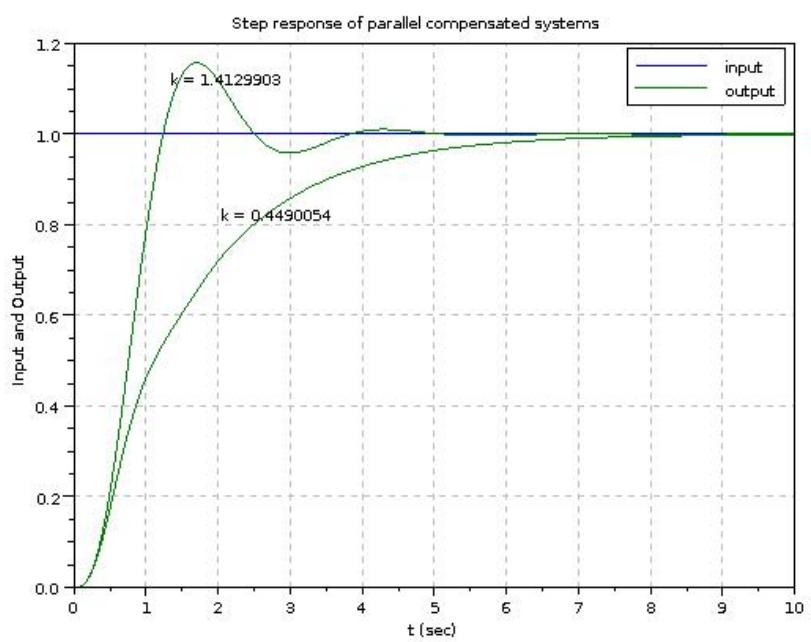


Figure 6.33: Design of parallel compensation by root locus

Scilab code Exa 6.15 Design of lag compensator

```
1 // Example A-6-15
2 // Design of lag compensator
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6 mode(0);
7
8 // please edit the path
9 // cd "/<your code directory>/";
10 // exec("rootl.sci");
11 // exec("plotresp.sci");
12
13 s = %s;
14 G = syslin('c', 10, s * (s + 4));
15
16
17 Kv = 80; // desired velocity constant
18 R = [-2 -2];
19 I = [sqrt(6) -sqrt(6)];
20 dp = R(1) + %i*I(1)
21
22 disp(horner(s*G, 0), 'Kv (uncompensated system) = ');
23 _beta = 20; // taking Kc = 1 we get beta as 10
24 z = 0.1; // choose z = 0.1
25 p = z / _beta;
26 Gc = (s + z)/(s + p);
27 disp(Gc, 'compensator = ');
28 H = G * Gc; // compensated system
29 H = syslin('c', numer(H), denom(H));
30 Gdp = horner(Gc, dp);
31 disp(abs(Gdp), 'Magnitude contribution of controller
=');
```

```

32 disp(phasemag(Gdp), 'Angle contribution of controller
   =');
33
34 rootl(G, [-3 -4; 1 4], '');
35 rootl(H, [-3 -4; 1 4], 'Uncompensated and Compensated
   system');
36 xgrid(color('gray'));
37 plot(R, I, 'x');
38 xstring(R(1), I(1), 'Original pole on uncompensated
   sys');
39
40 G1 = syslin('c', G /. 1);
41 C = syslin('c', H /. 1);           // final closed loop
   system
42 disp(C, 'closed loop system =');
43 disp(roots(C.den), 'closed loop poles = ');
44 disp(horner(s*H, 0), 'velocity error constant Kv =')
45
46 scf();
47 subplot(2, 1, 1);
48 t = 0:0.05:10;
49 u = ones(1, length(t));
50 plotresp(u, t, G1, '');
51 plotresp(u, t, C, 'Unit step response');
52 xstring(1, 0.9, 'uncompensated system');
53 xstring(0.7, 1.12, 'compensated system');
54
55
56 t = 0:0.5:20;
57 subplot(2, 1, 2);
58 plotresp(t, t, G1, '');
59 plotresp(t, t, C, 'Unit ramp response');
60 xstring(2, 0.9, 'uncompensated system');
61 xstring(0.1, 4, 'compensated system');

```

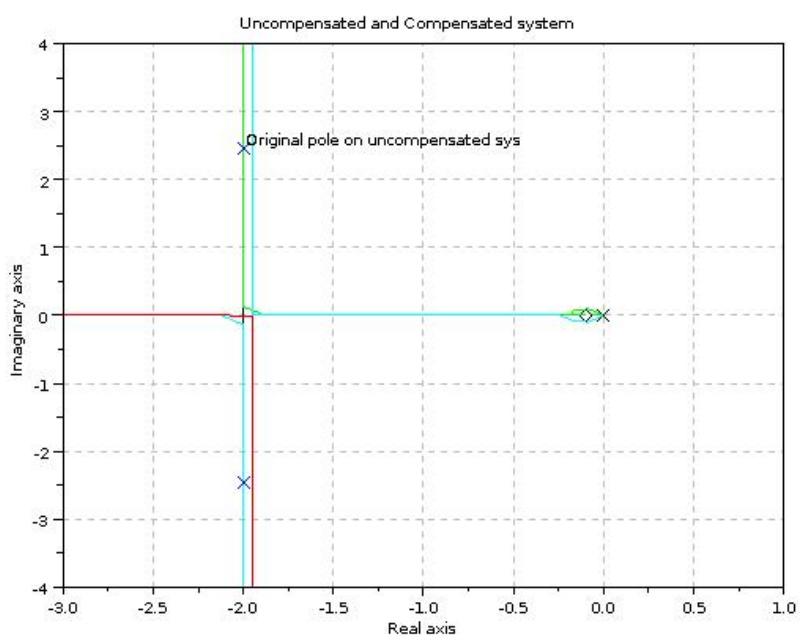


Figure 6.34: Design of lag compensator

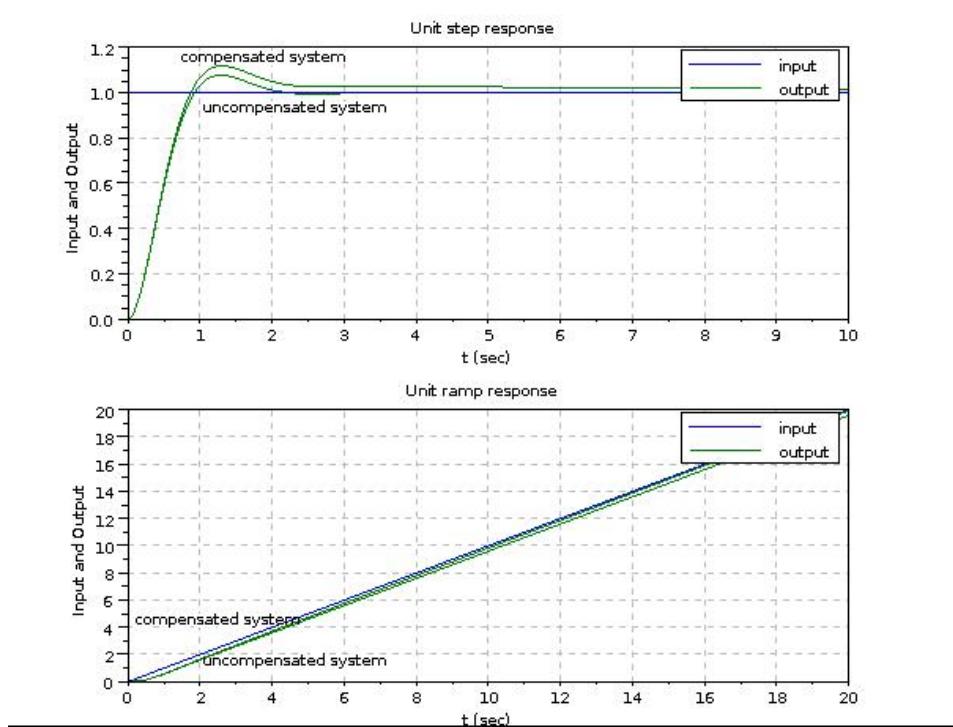


Figure 6.35: Design of lag compensator

check Appendix AP 2 for dependency:

`plotresp.sci`

check Appendix AP 7 for dependency:

`rootl.sci`

Scilab code Exa 6.16 Design of lag lead compensator

```
1 // Example A-6-16
2 // Design of lag lead compensator
```

```

3
4 clear; clc;
5 xdel(winsid()); // close all windows
6 mode(0);
7
8 // please edit the path
9 // cd "/<your code directory>/";
10 // exec("rootl.sci");
11 // exec("plotresp.sci");
12
13 s = %s;
14 G = syslin('c', 10, s * (s + 2) * (s + 8));
15
16 Kv = 80; // desired velocity constant
17 R = [-2 -2];
18 I = [2*sqrt(3) -2*sqrt(3)];
19 dp = R(1) + %i*I(1)
20
21 disp(horner(s*G, 0), 'Kv (uncompensated system) = ');
22
23 // designing lead part
24 Kc = Kv / abs(horner(s*G, 0))
25 angdef = 180 - phasemag(horner(G, dp))
26 z1 = 3.7 // z1 and p1 determined graphically
27 p1 = 53.35
28 T1 = 1 / z1
29 _beta = T1 * p1; disp(_beta, 'beta = ');
30
31 Gc1 = Kc * (s + z1)/(s + p1); disp(Gc1, 'Lead
    compensator Gc1 = ');
32
33 // Lag compensator design
34 p2 = 0.01 // say
35 z2 = p2 * _beta
36 Gc2 = (s + z2)/(s + p2);
37 disp(Gc2, 'Lag compensator Gc2 = ');
38 disp(abs(horner(Gc2, dp)), 'magnitude contribution of
    lag part = ');

```

```

39 disp(phasemag(horner(Gc2,dp)), 'angle contribution of
    lag part =');
40 // these are acceptable
41
42 Gc = Gc1 * Gc2
43 H = G * Gc;           // compensated system
44 H = syslin('c',numer(H),denom(H));
45
46 subplot(1,2,1);
47 rootl(G,[-10 -10; 10 10], 'Uncompensated system');
48 plot(R,I,'x');
49 xgrid(color('gray'));
50 subplot(1,2,2);
51 rootl(H,[-10 -10; 10 10], 'Compensated system');
52 plot(R,I,'x');
53 xgrid(color('gray'));
54 xstring(R(1),I(1), 'Desired closed loop poles');
55
56 G1 = syslin('c',G /. 1);
57 C = syslin('c',H /. 1);      // final closed loop
    system
58 disp(C, 'closed loop system =');
59 disp(roots(C.den), 'closed loop poles = ');
60 disp(horner(s*H,0), 'velocity error constant Kv =')
61
62 scf();
63 subplot(2,1,1);
64 t = 0:0.05:10;
65 u = ones(1,length(t));
66 plotresp(u,t,G1, '');
67 plotresp(u,t,C, 'Unit step response');
68 xstring(1,0.5, 'uncompensated system');
69 xstring(0.7,1.12, 'compensated system');
70
71 subplot(2,1,2);
72 plotresp(t,t,G1, '');
73 plotresp(t,t,C, 'Unit ramp response');
74 xstring(2,0.9, 'uncompensated system');

```

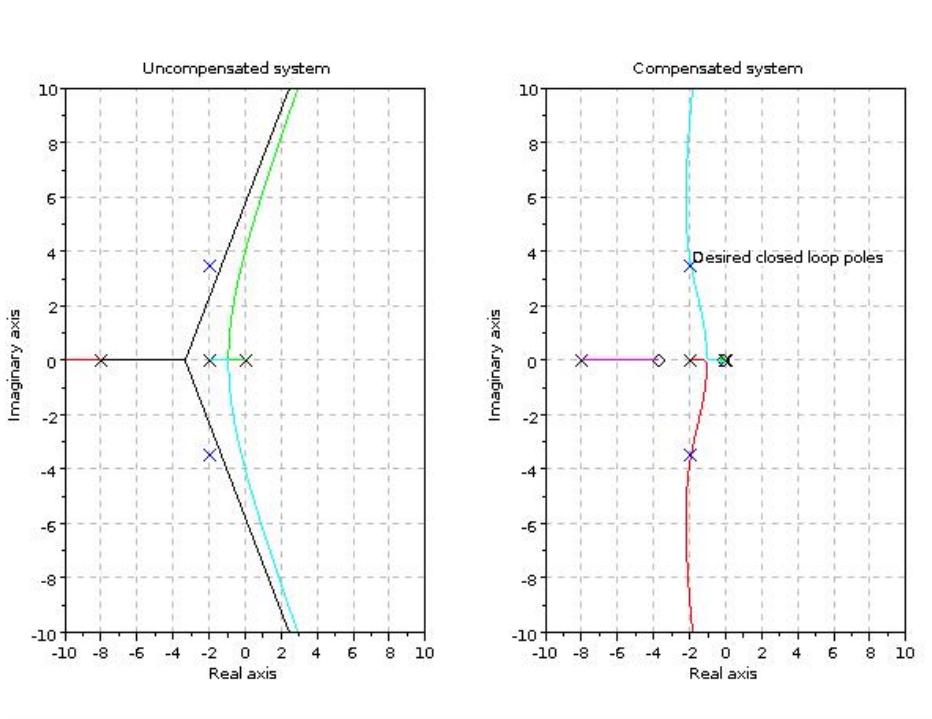


Figure 6.36: Design of lag lead compensator

```
75 xstring(0.5,2,'compensated system');
```

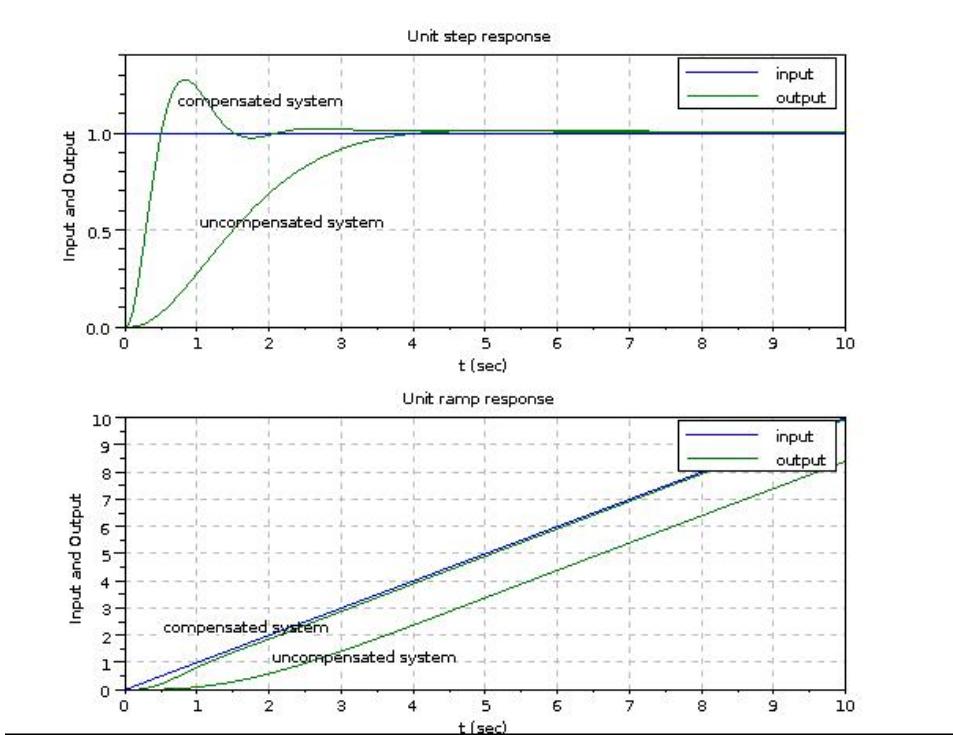


Figure 6.37: Design of lag lead compensator

Chapter 7

Control Systems Analysis and Design by Frequency Response Method

Scilab code Exa 7.a.1 Bode plot

```
1 // Example A-7-1
2 // Bode plot
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 s = %s /2 /%pi; // frequencies in rad/s
8 G = syslin('c', 10*(s + 1), (s + 2)*(s + 5));
9 bode(G,0.1,100);
10 xtitle('Bode plot of G(s) = 10*(s + 1)/[(s + 2)*(s +
5)]', 'rad/s');
11 a = gcf(); set(a.children(1).x_label,'text','rad/s');
```

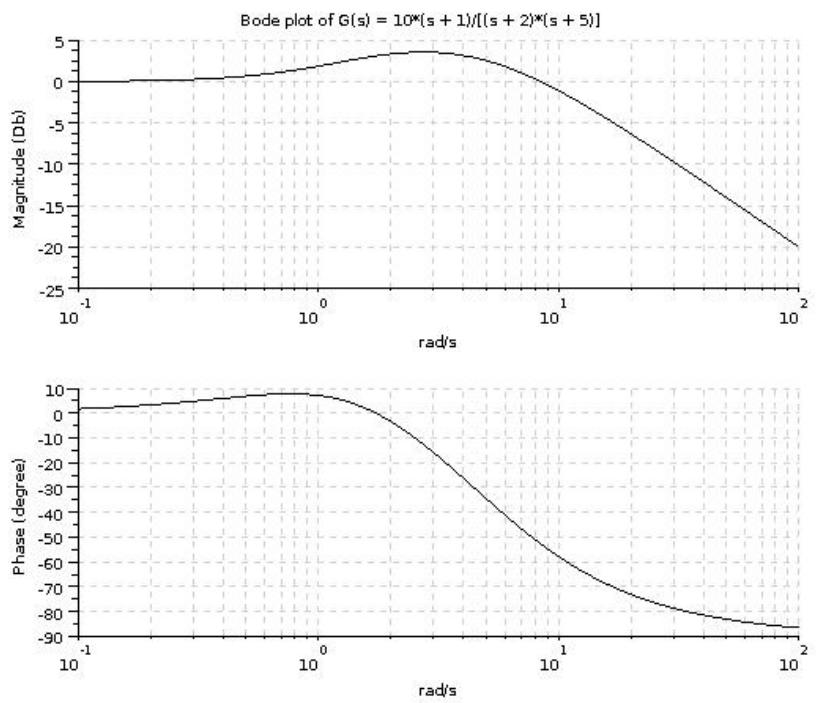


Figure 7.1: Bode plot

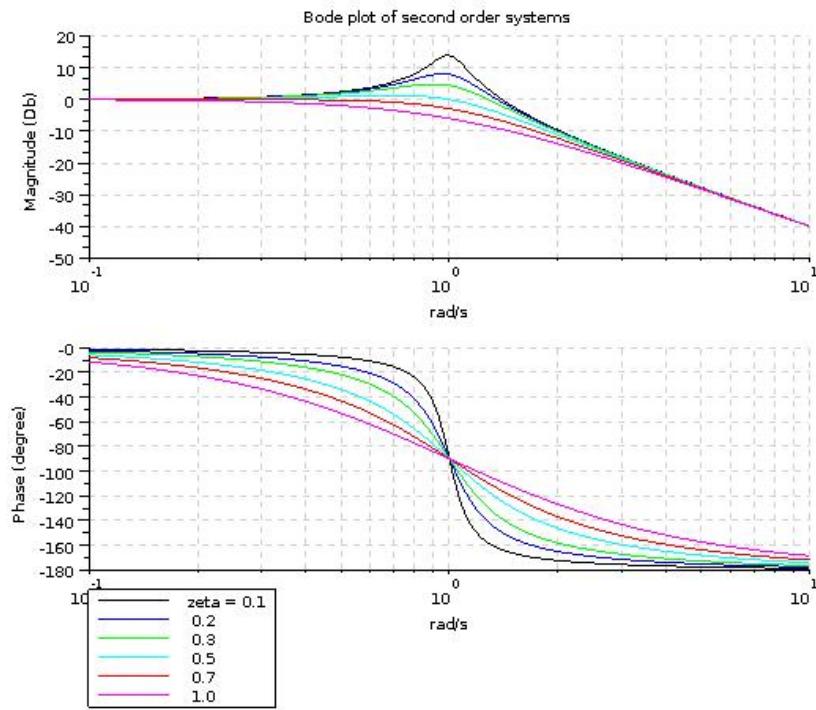


Figure 7.2: Bode plot for 2nd order systems with varying zeta

Scilab code Exa 7.i.1 Bode plot for 2nd order systems with varying zeta

```
1 // Illustration 7-1
2 // Bode plot of second order systems with varying
damping (zeta)
3
4 // With refernce to section 7.2 (Figure 7.9)
5
6 clear; clc;
7 xdel(winsid()); // close all windows
8
9 s = %s;
10 // Taking wn = 1 in all cases
11 zeta = [0.1 0.2 0.3 0.5 0.7 1.0];
12
13
14 N = ones(6,1);
15 D = zeros(6,1);
16 for i = 1:6
17     D(i) = s^2 + 2*zeta(i)*s + 1;
18 end
19 H = syslin('c',N,D);
20
21 omega = logspace(-1,1,100);
22 f = omega / 2 / %pi;
23 repf = repfreq(H,f); // Frequency response
24
25 bode(omega,repf, ['zeta = 0.1', ' 0.2', ' 0.3', ' 0.5',
' 0.7', ' 1.0']);
26 xtitle('Bode plot of second order systems', 'rad/s');
27 a = gcf(); set(a.children(1).x_label, 'text', 'rad/s');
```

Scilab code Exa 7.a.3 Bode plot for system in state space

```
1 // Example A-7-3
2 // Bode plot for system in state space
3
4 clear; clc;
5 xdel(winsid()); //close all windows
6
7 // please edit the path
8 // cd "/<your code directory >/";
9 // exec("transferf.sci");
10
11 A = [0 1; -25 -4];
12 B = [1 1; 0 1];
13 C = [1 0; 0 1];
14 D = zeros(2,2);
15 G = transferf(A,B,C,D); disp(G," transfer function = "
);
16
17 subplot(2,2,1);
18 bode(G(1,1));
19 subplot(2,2,2);
20 bode(G(1,2));
21 subplot(2,2,3);
22 bode(G(2,1));
23 subplot(2,2,4);
24 bode(G(2,2));
```

check Appendix AP 4 for dependency:

transferf.sci

Scilab code Exa 7.a.4 Bode plot for different gain K

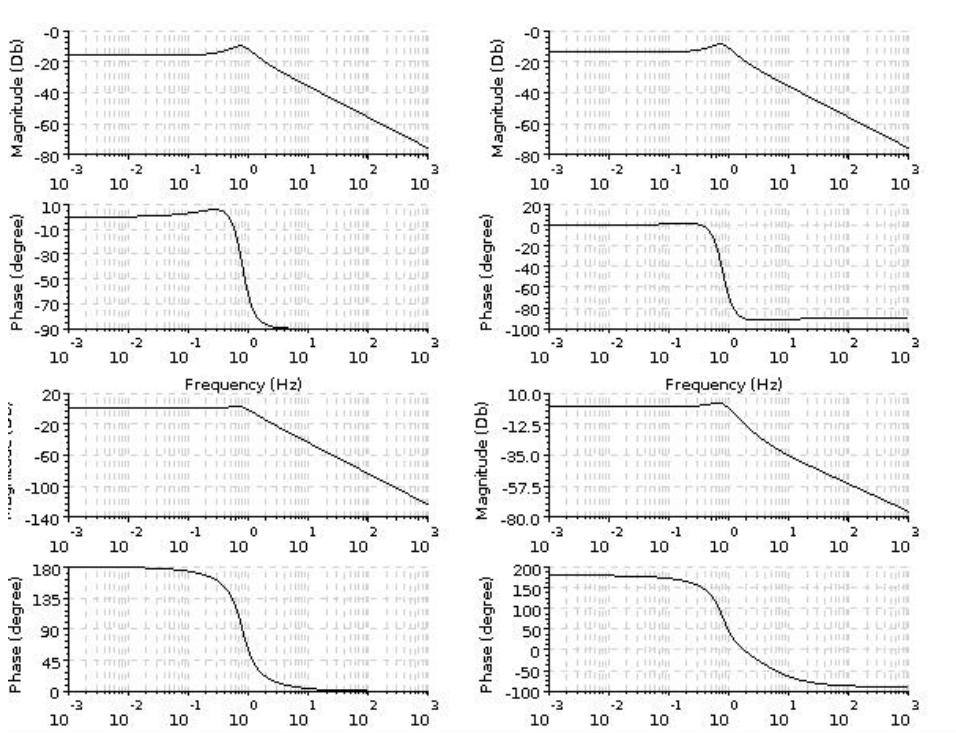


Figure 7.3: Bode plot for system in state space

```

1 // Example A-7-4
2 // Bode plot for different gain K
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 s = %s /2/%pi ;
8 P = s*(s+1)*(s+5);
9 num = [1,10,20];
10 den = [P+1 , P+10 , P+20];
11 Gtf = num ./ den;
12 G = syslin('c',Gtf);
13
14 bode([G(1,1); G(1,2); G(1,3)],0.1,100,['K = 1'; 'K =
    10'; 'K = 20']); )
15 xtitle(' ', 'rad/s');
16 a = gcf(); set(a.children(1).x_label,'text','rad/s');

```

Scilab code Exa 7.a.8 Stability check

```

1 // Example A-7-8
2 // Stability check
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 s = %s;
8 K = 2;
9 P = s*(s+1)*(2*s+1) + K;
10 routh_t(P)
11 // unstable since two roots are in RHP

```

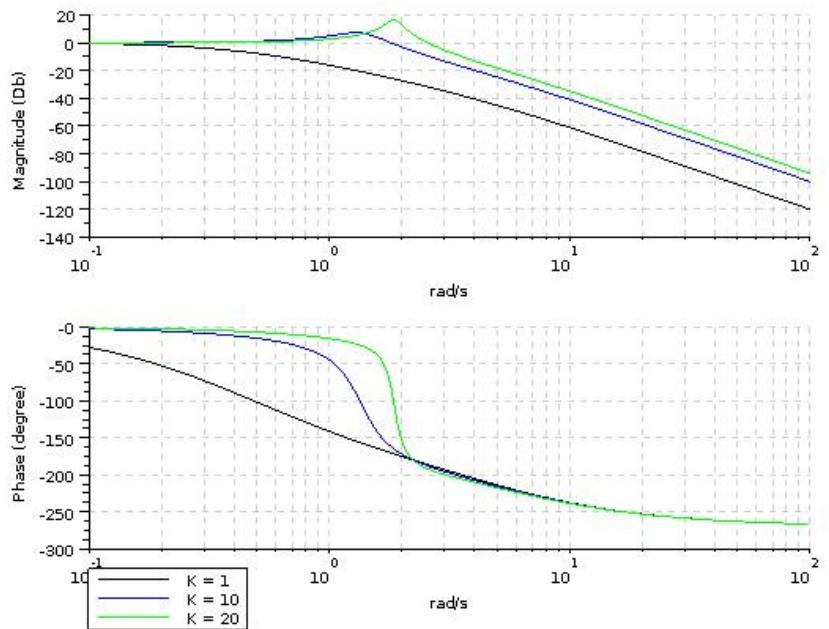


Figure 7.4: Bode plot for different gain K

Scilab code Exa 7.a.10 Nyquist Plot with transport lag

```
1 // Example A-7-10
2 // Nyquist Plot with transport lag
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7
8 omega = logspace(-2,2,100);
9 f = omega ./ (2*pi);
10 repf = 2.65 * exp(%i*omega*-0.8) ./ (ones(1,length(
    omega)) + %i*omega);
11
12 nyquist(f,repf);
13 plot(-1,0,'.');
14 xstring(-0.9,0,'passes -1',0,1);
```

Scilab code Exa 7.a.11 Nyquist Plot

```
1 // Example A-7-11
2 // Nyquist Plot
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 s = %s /2 /%pi;
8 num = 20 * ( s^2 + s + 0.5);
9 den = s * (s + 1) * (s + 10);
10 G = syslin('c',num,den);
```

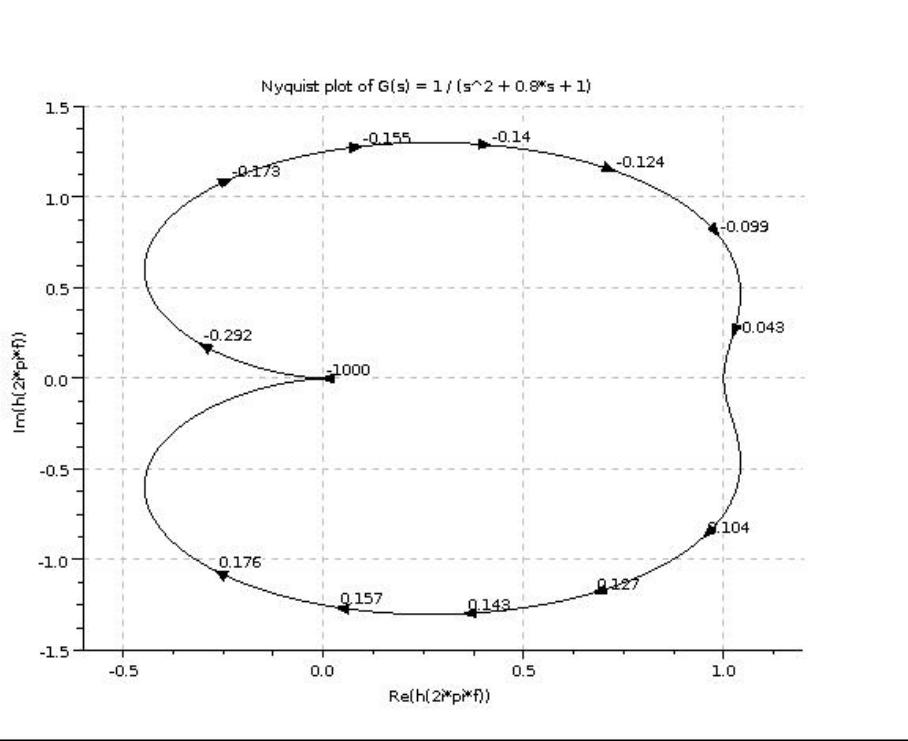


Figure 7.5: Nyquist Plot with transport lag

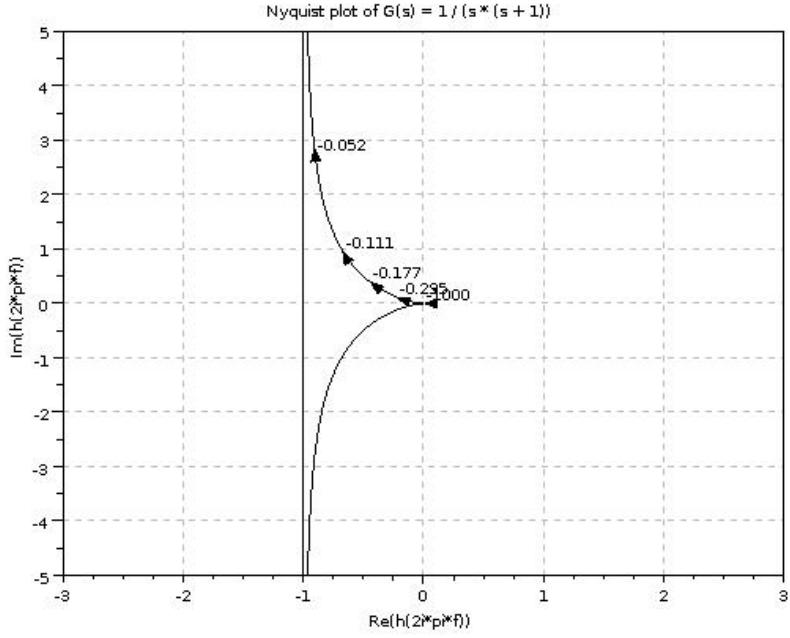


Figure 7.6: Nyquist Plot

```

11
12 a = gca();
13 a.clip_state = 'on';
14 nyquist(G,-1000,1000);
15 xgrid(color('gray'));
16 a.data_bounds = [-2 -3 ; 3 3];
17 a.box = 'on';

```

Scilab code Exa 7.a.12 Nyquist plot for positive omega

1 // Example A-7-12

```

2 // Nyquist plot for positive omega
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 s = %s /2 /%pi;
8 num = 20 * ( s^2 + s + 0.5);
9 den = s * (s + 1) * (s + 10);
10 G = syslin('c',num,den);
11
12 a = gca();
13 a.clip_state = 'on';
14 nyquist(G,0.01,1000);
15 xgrid(color('gray'));
16 a.data_bounds = [-3 -5 ; 3 1];
17 a.box = 'on';

```

Scilab code Exa 7.a.13 Nyquist plot with points at selected frequencies

```

1 // Example A-7-13
2 // Nyquist plot with points plotted at selected
   frequencies
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 s = %s /2 /%pi;
8 num = 20 * ( s^2 + s + 0.5);
9 den = s * (s + 1) * (s + 10);
10 G = syslin('c',num,den);
11
12 a = gca();
13 a.clip_state = 'on';

```

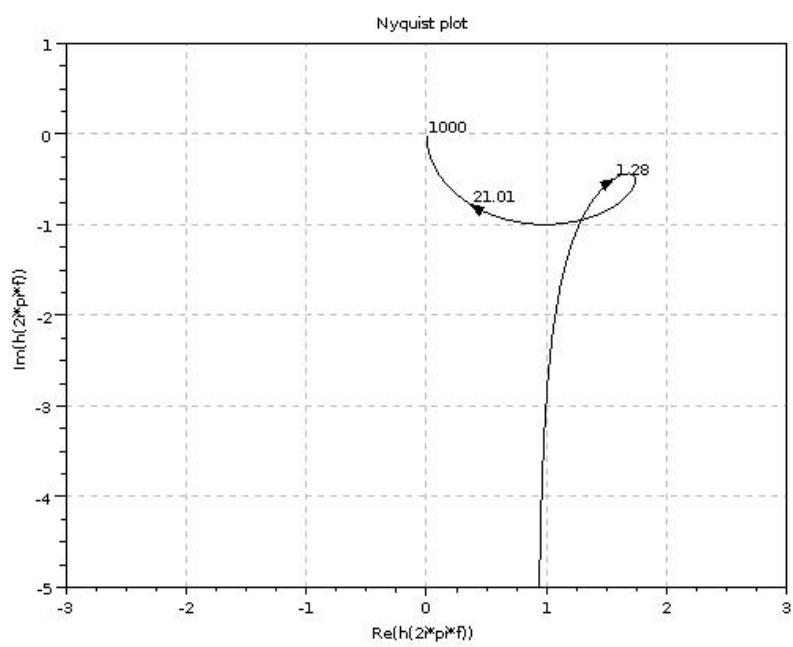


Figure 7.7: Nyquist plot for positive omega

```

14 nyquist(G,0.01,1000);
15 xtitle('Nyquist Diagram');
16 a.data_bounds = [-2 -5 ; 3 0];
17 a.box = 'on';
18
19 omega = [0.2 0.3 0.5 1 2 6 10 20];
20 z = repfreq(G,omega);
21 plot(real(z), imag(z), '.k');
22
23 x = [1 1.1 1.2 1.3 1.8 1.5 0.8 0.25];
24 y = [-4.7 -3.3 -1.7 -0.51 -0.4 -1 -1.3 -1];
25 text = ['w = 0.2' '0.3' '0.5' '1.0' '2.0' '6.0' '10'
          '20'];
26 xstring(x,y,text,0,1);
27
28 [phi db] =phasemag(z);
29 mag = abs(z);
30 disp(['omega' ' mag' ' phi'] , '[w mag phi] = ');

```

Scilab code Exa 7.a.14 Nyquist plot for positive and negative feedback

```

1 // Example A-7-14
2 // Nyquist plot for positive and negative feedback
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 s = %s;
8 num = s^2 + 4*s + 6;
9 den = s^2 + 5*s + 4;
10 G = syslin('c',num,den);
11 H = syslin('c',-1 * num,den);
12

```

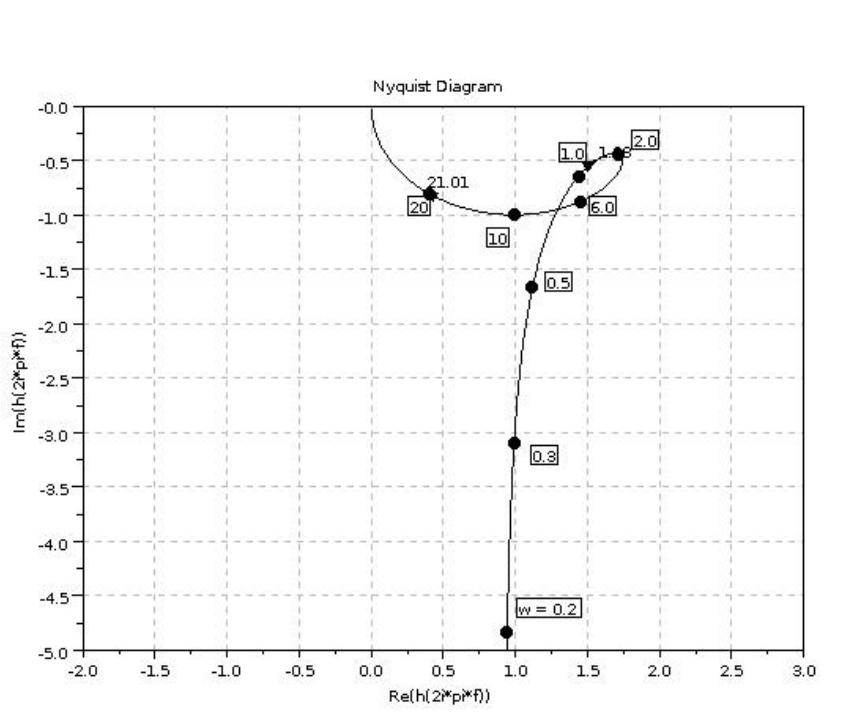


Figure 7.8: Nyquist plot with points at selected frequencies

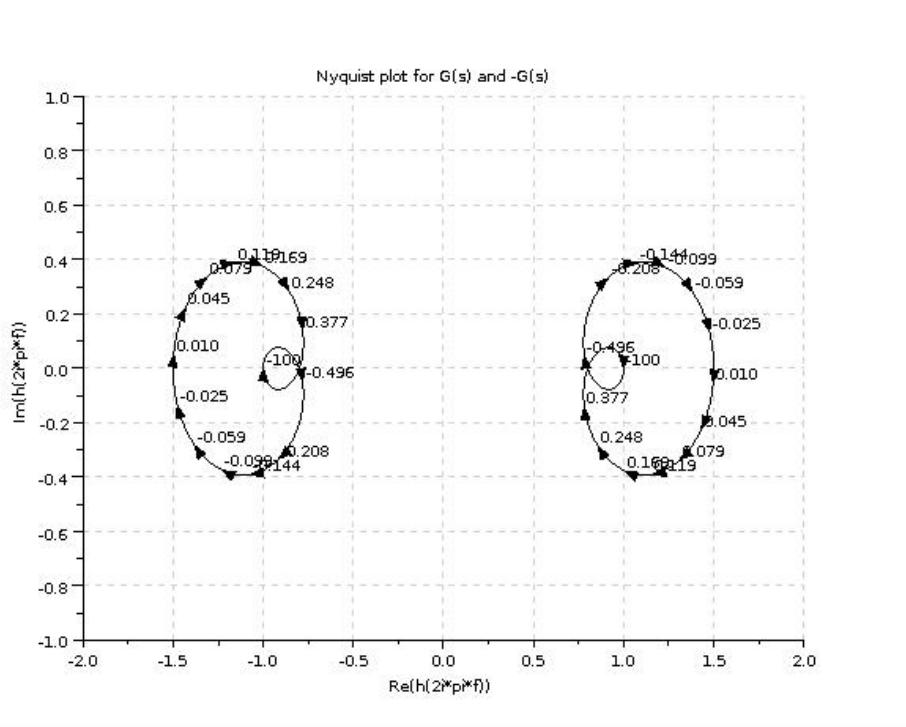


Figure 7.9: Nyquist plot for positive and negative feedback

```

13 nyquist(G, -100, 100);
14 nyquist(H, -100, 100);
15 xtitle('Nyquist plot for G(s) and -G(s)');
16 a = gca(); a.data_bounds = [-2 -1; 2 1];

```

Scilab code Exa 7.a.18 Verifying experimentally derived Transfer function

```

1 // Example A-7-18
2 // Verifying experimentally derived Transfer
function

```

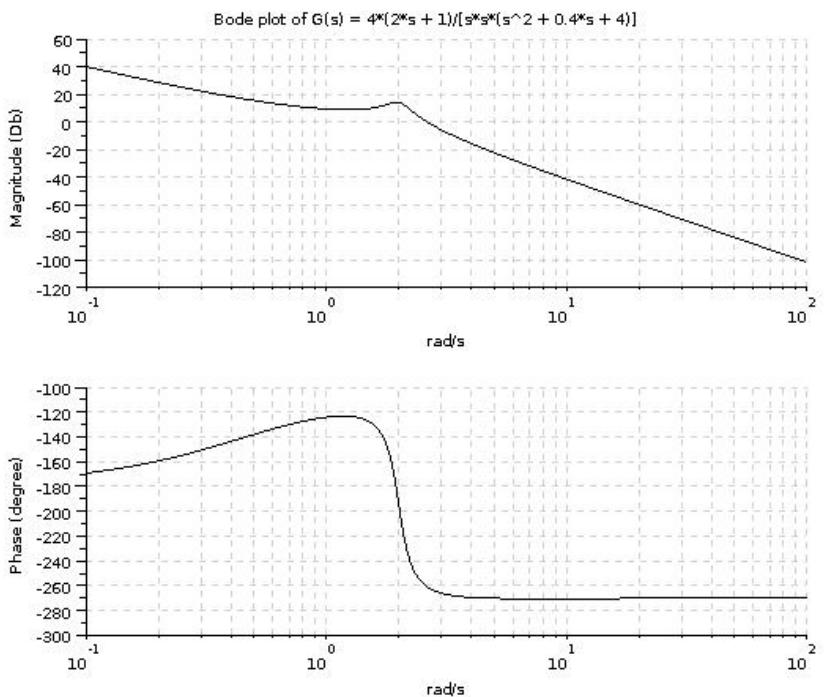


Figure 7.10: Verifying experimentally derived Transfer function

```

3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 s = %s /2 /%pi; // frequencies in rad/s
8 G = syslin('c', 4*(2*s + 1), s*s*(s^2 + 0.4*s + 4) )
    ;
9 bode(G,0.1,100);
10 xtitle('Bode plot of G(s) = 4*(2*s + 1) / [ s*s*( s^2 +
    0.4*s + 4) ] ', 'rad/s');
11 a = gcf(); set(a.children(1).x_label,'text','rad/s');

```

Scilab code Exa 7.a.23 Nichols plot

```
1 // Example A-7-23
2 // Nichols plot
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 s = %s;
8 G = syslin('c', 9 , s*(s+0.5)*(s^2 + 0.6*s + 10) );
9 black(G);
10 chart([8 -4],[],list(1,0));
11 xgrid(color('gray'));
```

check Appendix AP 2 for dependency:

plotresp.sci

Scilab code Exa 7.1 Steady state sinusoidal output

```
1 // Example 7-1
2 // Steady state sinusoidal output
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 // please set the path
8 // cd "/<your code directory>/"
9 // exec("plotresp.sci")
10
11 s = %s;
12 w = 1;
13 K = 5;
14 T = 0.1;
```

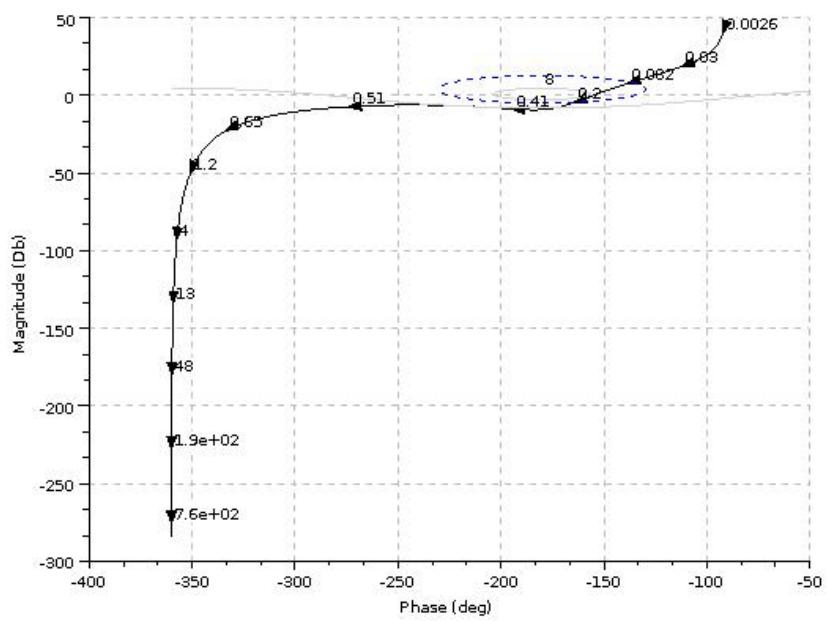


Figure 7.11: Nichols plot

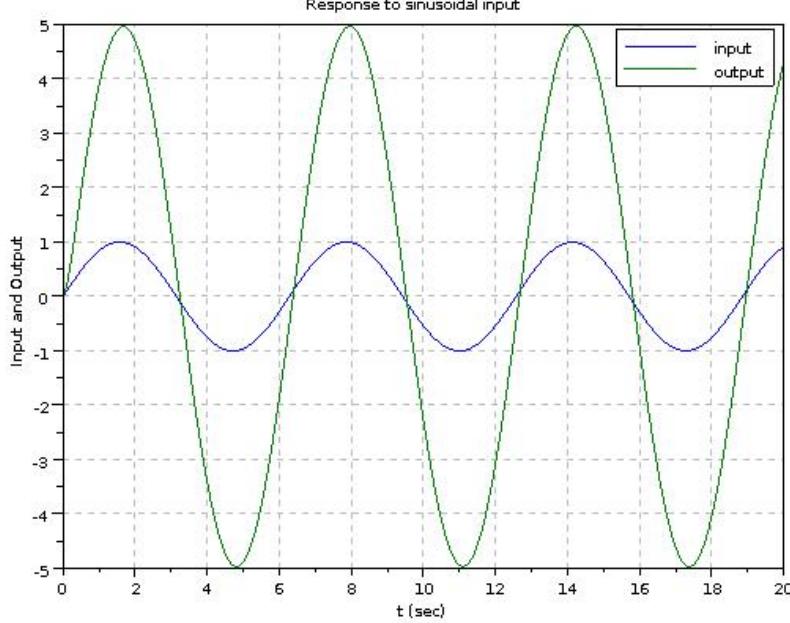


Figure 7.12: Steady state sinusoidal output

```

15
16 G = syslin('c',K,T*s + 1);
17 t = 0:0.1:20;
18 u = sin(w*t);
19 plotresp(u,t,G,'Response to sinusoidal input');
20 // as T*w is small amplitude of output is ~ K (5)

```

check Appendix AP 2 for dependency:

`plotresp.sci`

Scilab code Exa 7.2 Steady state sinusoidal output lag and lead

```

1 // Example 7-2
2 // Steady state sinusoidal output lag and lead
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 // please set the path
8 // cd "/<your code directory >/"
9 // exec("plotresp.sci")
10
11 s = %s;
12 T1 = 1;
13 T2 = 5;
14 a = s + 1/T1;
15 b = s + 1/T2;
16 w = 1;
17
18 G1 = syslin('c',a,b);
19 G2 = syslin('c',b,a);
20 t = 0:0.1:50;
21 u = sin(w*t);
22 plotresp(u,t,G1,'Response to sinusoidal input');
23 plotresp(u,t,G2,'Response to sinusoidal input');
24 xstring(17,1.4,'Lead network T1 > T2 : lead network'
    );
25 xstring(17,-0.8,'Lag network T1 > T2 : lead network'
    );

```

Scilab code Exa 7.3 Bode Plot in Hz

```

1 // Example 7-3
2 // Bode Plot in Hz
3

```

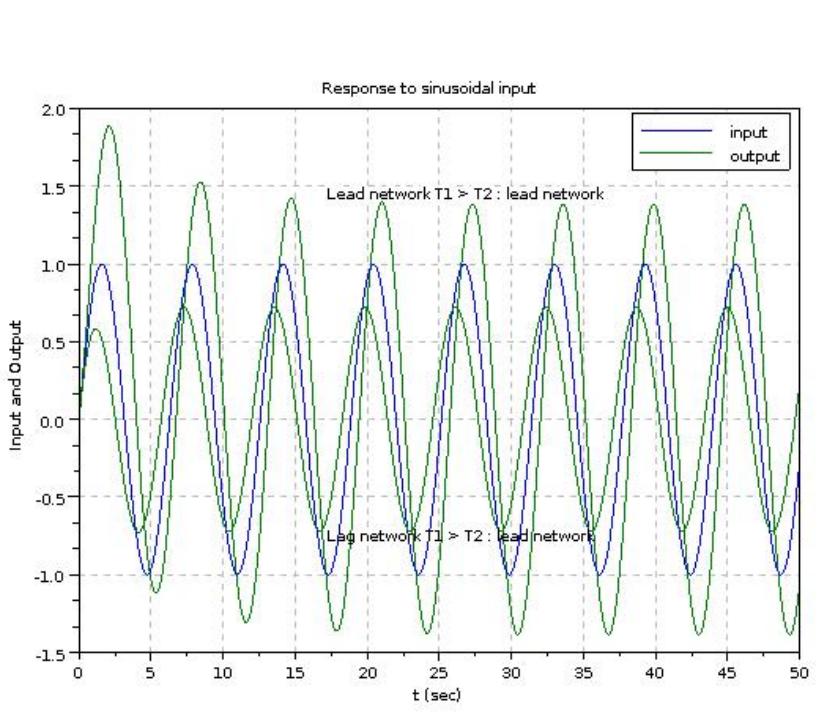


Figure 7.13: Steady state sinusoidal output lag and lead

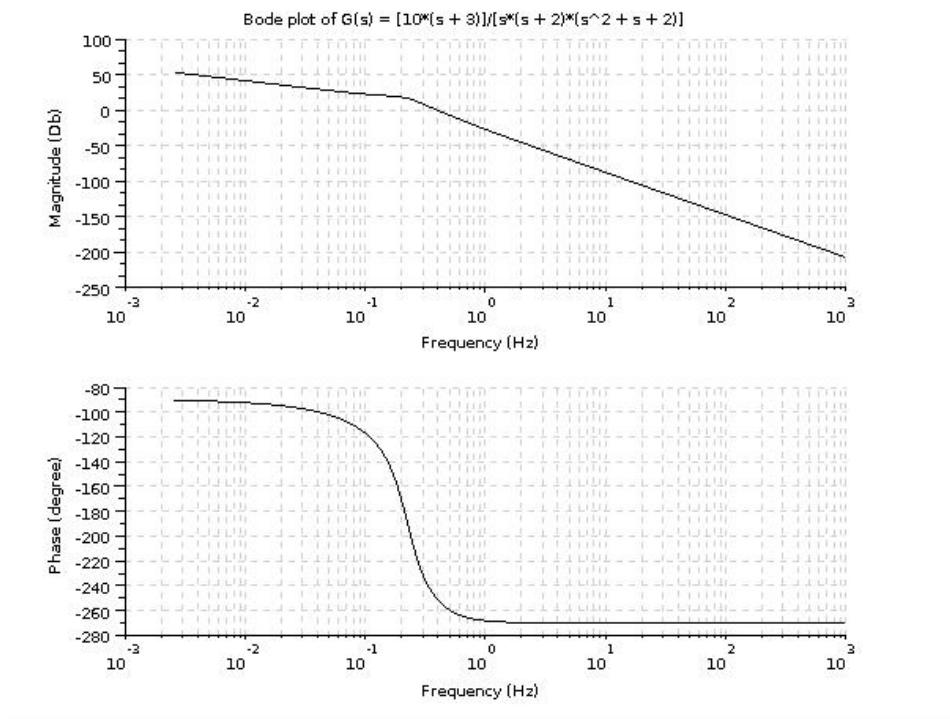


Figure 7.14: Bode Plot in Hz

```

4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 s = %s;
8 num = 10*(s + 3);
9 den = s * (s + 2) * (s^2 + s + 2);
10 G = syslin('c',num,den);
11
12 bode(G);
13 xtitle('Bode plot of G(s) = [10*(s + 3)]/[s*(s + 2)
    *(s^2 + s + 2)]');

```

Scilab code Exa 7.4 Bode Plot with transport lag

```
1 // Example 7-4
2 // Bode Plot with transport lag
3
4 clear; clc;
5 xdel(winsid()); //close all windows
6
7 omega = logspace(-1,1,100);
8 repf = exp(%i*omega*-0.5) ./ (ones(1,length(omega))
+ %i*omega);
9
10 bode(omega,repf);
11 xtitle('Bode plot of G(s) = exp(-0.5jw) / [1 + jw]', 'rad/s');
12 a = gcf(); set(a.children(1).x_label, 'text', 'rad/s');
```

Scilab code Exa 7.5 Bode Plot in rad per s

```
1 // Example 7-5
2 // Bode Plot in rad/s
3
4 clear; clc;
5 xdel(winsid()); //close all windows
6
7 s = %s;
8 num = 25;
9 den = s^2 + 4*s + 25;
10 G = syslin('c',num,den);
11
12 bode(G);
13 xtitle('Bode plot of G(s) = 25 / s^2 + 4*s + 25');
```

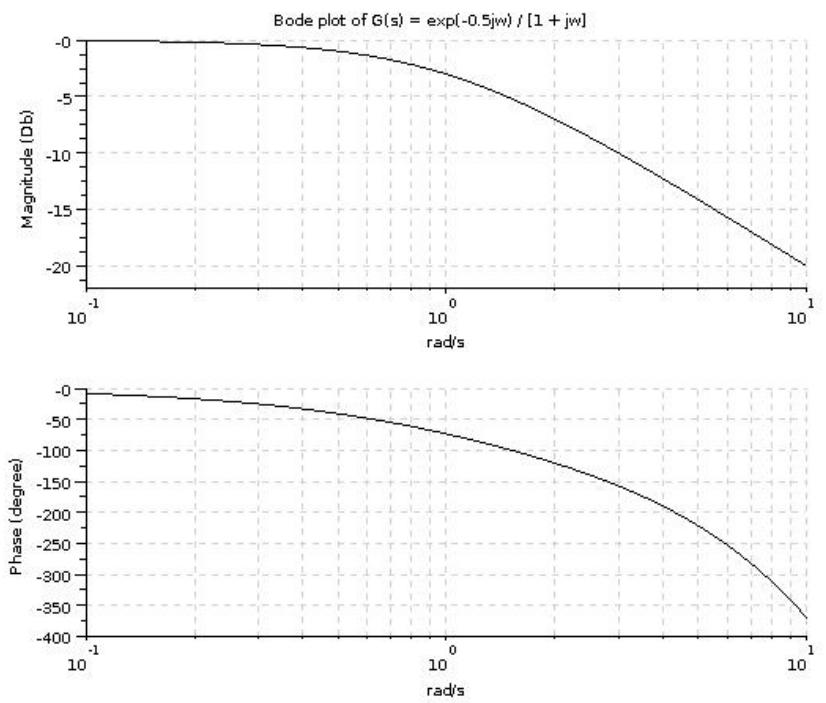


Figure 7.15: Bode Plot with transport lag

```

15 // Note , bode plots in Sci-Lab use the frequency in
16 // Hz and not in
17 // rad/s . If we wish to get the plot with rad/s we
18 // can ...
19
20 omega = logspace(-2,2,50);
21 f = omega / 2 / %pi;
22 repf = repfreq(G,f);      // calculate the frequency
23                         // response
24                         // repf is a vector of
25                         // complex numbers
26 scf();
27 bode(omega,repf);
28 xtitle('Bode plot of G(s) = 25 / s^2 + 4*s + 25', '
29        'rad/s');
30 a = gcf(); set(a.children(1).x_label,'text','rad/s');

```

Scilab code Exa 7.6 Bode plot in rad per s

```

1 // Example 7-6
2 // Bode Plot in rad/s
3 // Plots made with angular frequency – rad/s on the
4 // x-axis
5 clear; clc;
6 xdel(winsid()); //close all windows
7
8 s = %s / 2 / %pi; //correction to get frequency
9 axis in rad/s
10 num = 9 * (s^2 + 0.2*s + 1);
11 den = s * (s^2 + 1.2*s + 9);

```

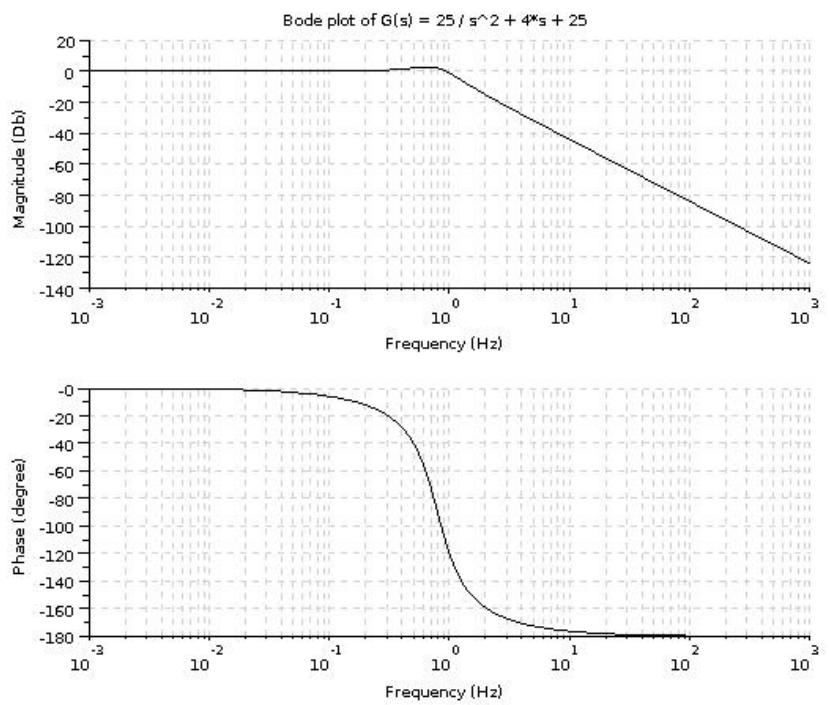


Figure 7.16: Bode Plot in rad per s

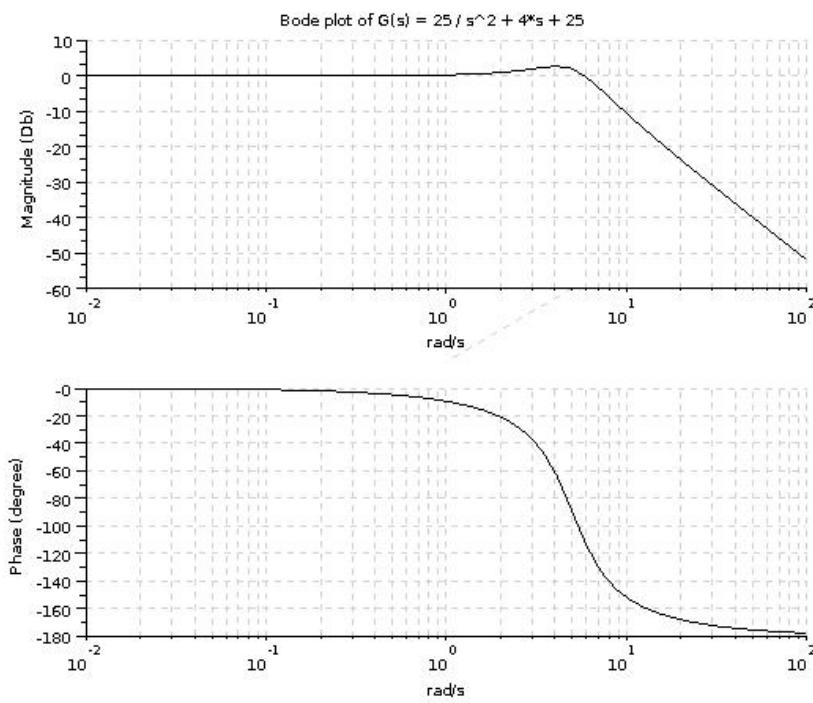


Figure 7.17: Bode Plot in rad per s

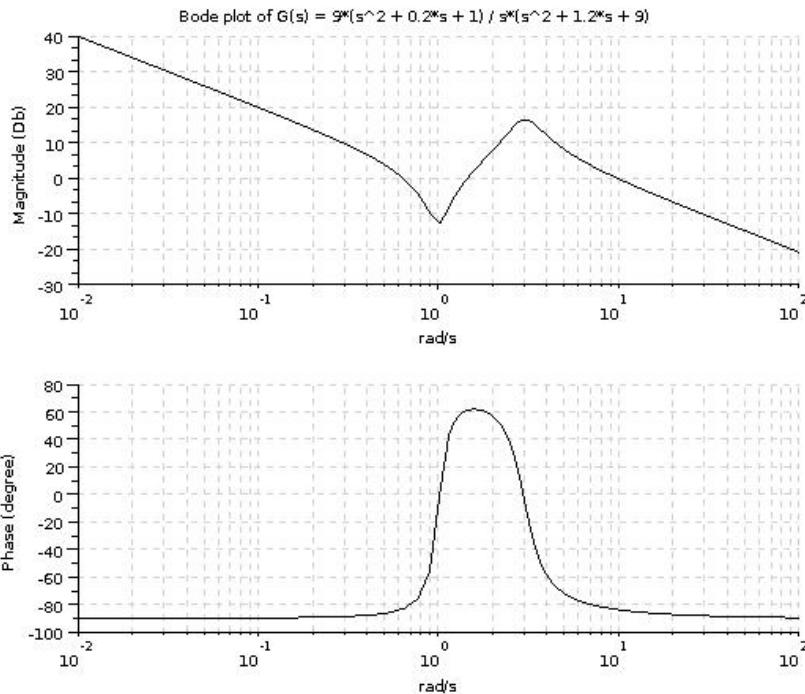


Figure 7.18: Bode plot in rad per s

```

11 G = syslin('c',num,den);
12
13 bode(G,0.01,100);
14 xtitle('Bode plot of G(s) = 9*(s^2 + 0.2*s + 1) / s
           *(s^2 + 1.2*s + 9)', 'rad/s');
15 a = gcf(); set(a.children(1).x_label,'text','rad/s');

```

Scilab code Exa 7.7 Bode Plot for a system in State Space

1 // Example 7-7

```

2 // Bode Plot for a system in State Space
   representation
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 A = [0 1; -25 -4];
8 B = [0 ; 25];
9 C = [1 0];
10 D = [0];
11 G = syslin('c',A,B,C,D);
12
13 omega = logspace(-1,2,100);
14 f = omega / 2 / %pi;
15 repf = repfreq(G,f); // Frequency response
16
17 bode(omega,repf);
18 xtitle('Bode Diagram','rad/s');
19 a = gcf(); set(a.children(1).x_label,'text','rad/s');

```

check Appendix AP 9 for dependency:

`spolarplot.sci`

Scilab code Exa 7.8 Polar Plot of a linear system

```

1 // Example 7-8
2 // Polar Plot of a linear system
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 // please edit the path
8 // ed "/<your code directory>/";

```

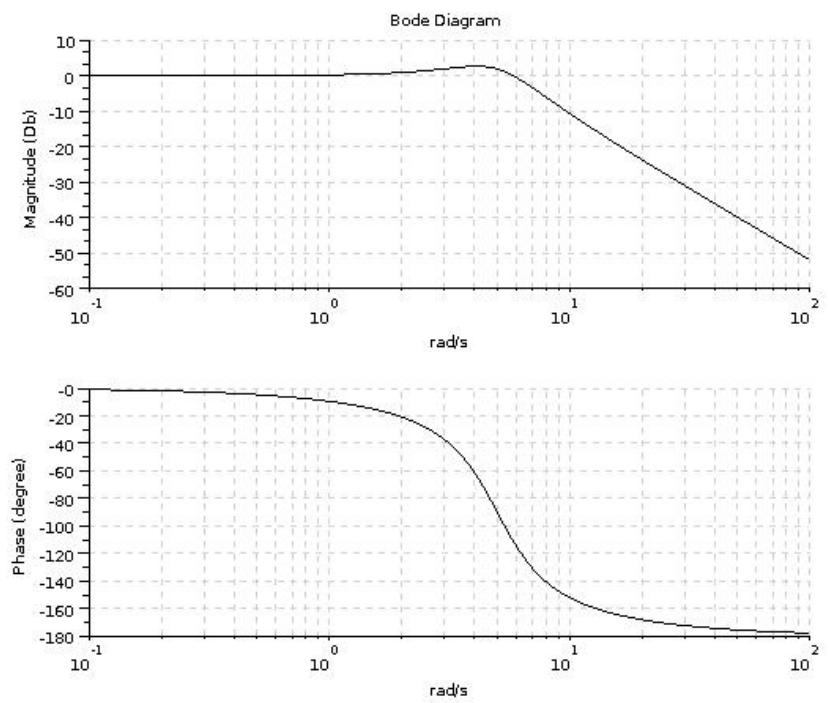


Figure 7.19: Bode Plot for a system in State Space

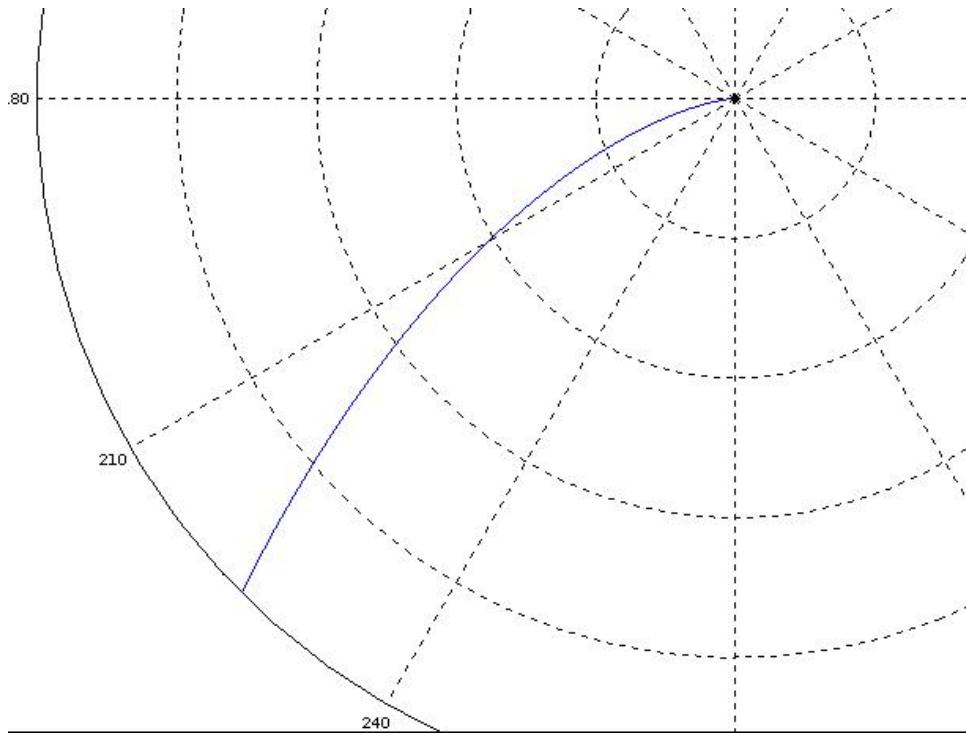


Figure 7.20: Polar Plot of a linear system

```

9 // exec("spolarplot.sci");
10
11 T = 10; s = %s;
12 omega = logspace(-1,3,1000);
13 G = syslin('c',1,s*(T*s + 1));
14 spolarplot();

```

Scilab code Exa 7.9 Polar Plot with transport lag

```

1 // Example 7-9
2 // Polar Plot with transport lag

```

```

3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 T = 10;
8 L = 100;
9 omega = logspace(-1,2,1000);
10 s = %i * omega;
11 den = s .* (T*s + 1);
12 num = exp(-1*s*L);
13 repf = num ./ den;
14 rad = abs(repfn);
15 theta = atan(imag(repfn),real(repfn));
16
17 polarplot(theta,rad,style = 2);

```

Scilab code Exa 7.10 Nyquist Plot

```

1 // Example 7-10
2 // Nyquist Plot
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 s = %s;
8 num = 1;
9 den = s^2 + 0.8*s + 1;
10 G = syslin('c',num,den);
11
12 nyquist(G,-1000,1000);
13 xgrid(color('gray'));
14 xtitle('Nyquist plot of G(s) = 1 / (s^2 + 0.8*s + 1)');

```

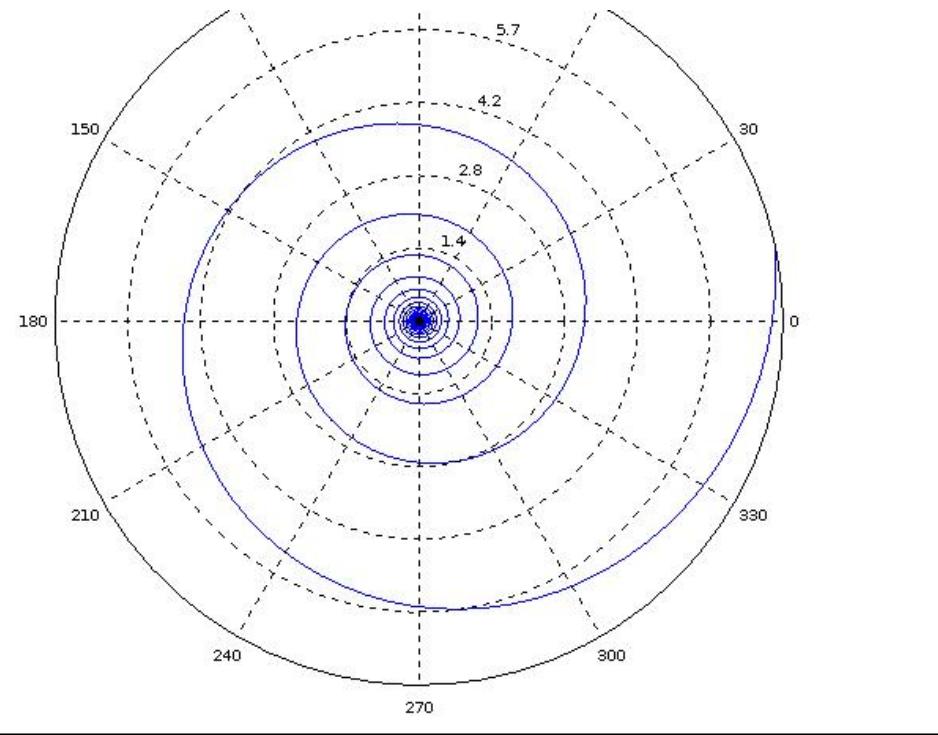


Figure 7.21: Polar Plot with transport lag

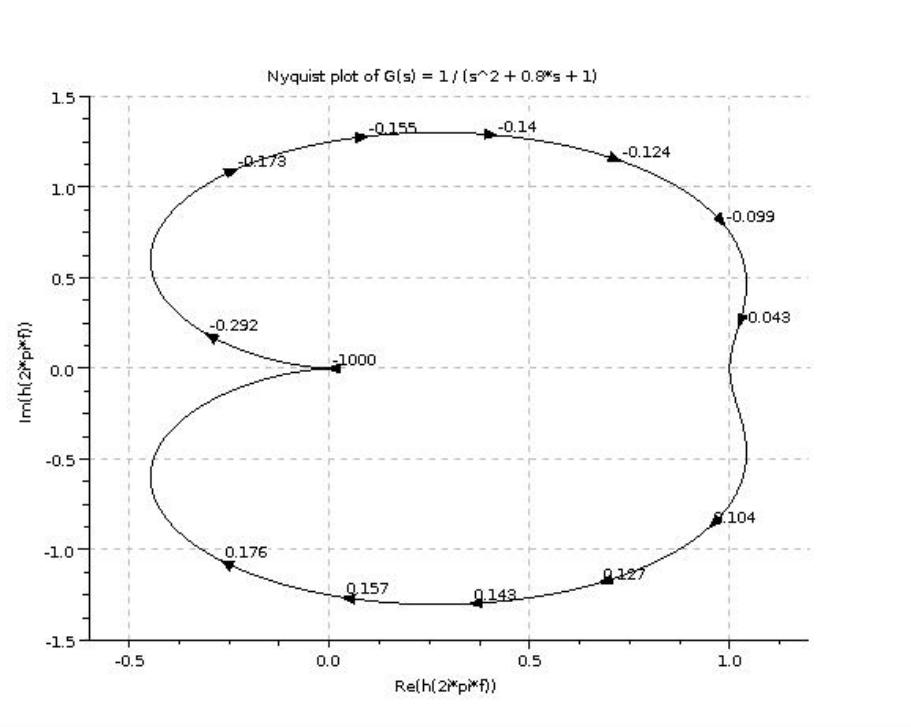


Figure 7.22: Nyquist Plot

```

15
16 // Note: nyquist function plots frequencies -1000
      and 1000 in Hz and not in rad/s

```

Scilab code Exa 7.11 Nyquist Plot

```

1 // Example 7-11
2 // Nyquist Plot
3
4 clear; clc;
5 xdel(winsid()); // close all windows

```

```

6
7 s = %s;
8 num = 1;
9 den = s * (s + 1);
10 G = syslin('c',num,den);
11
12 scf();
13 a = gca();
14 a.clip_state = 'on'; // clip the extra nyquist plot
15 nyquist(G,-1000,1000);
16 xgrid(color('gray'));
17 xtitle('Nyquist plot of G(s) = 1 / (s * (s + 1))');
18 a.data_bounds = [-3 -5 ; 3 5];
19 a.box = 'on';

```

Scilab code Exa 7.12 Nyquist Plots of system in state space

```

1 // Example 7-11
2 // Nyquist Plot
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 s = %s;
8 num = 1;
9 den = s * (s + 1);
10 G = syslin('c',num,den);
11
12 scf();
13 a = gca();
14 a.clip_state = 'on'; // clip the extra nyquist plot
15 nyquist(G,-1000,1000);
16 xgrid(color('gray'));

```

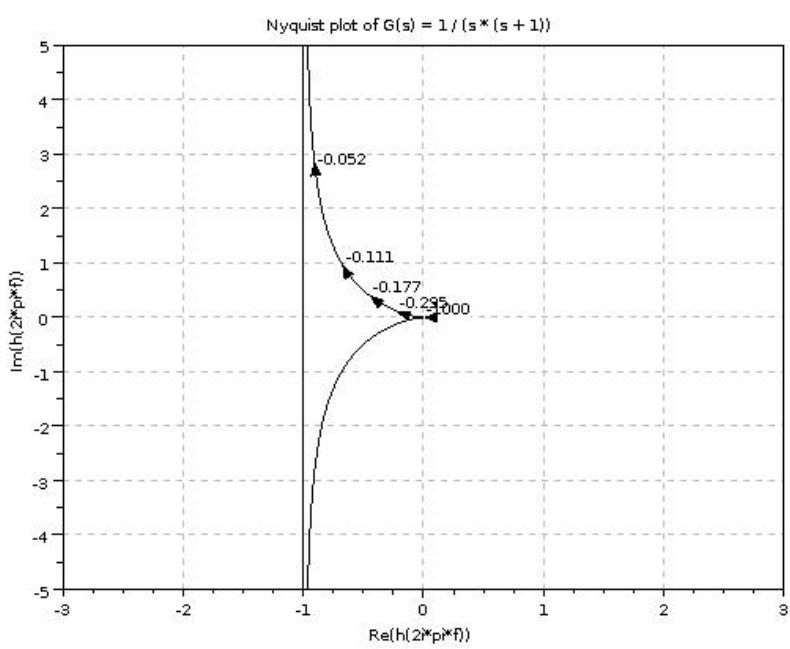


Figure 7.23: Nyquist Plot

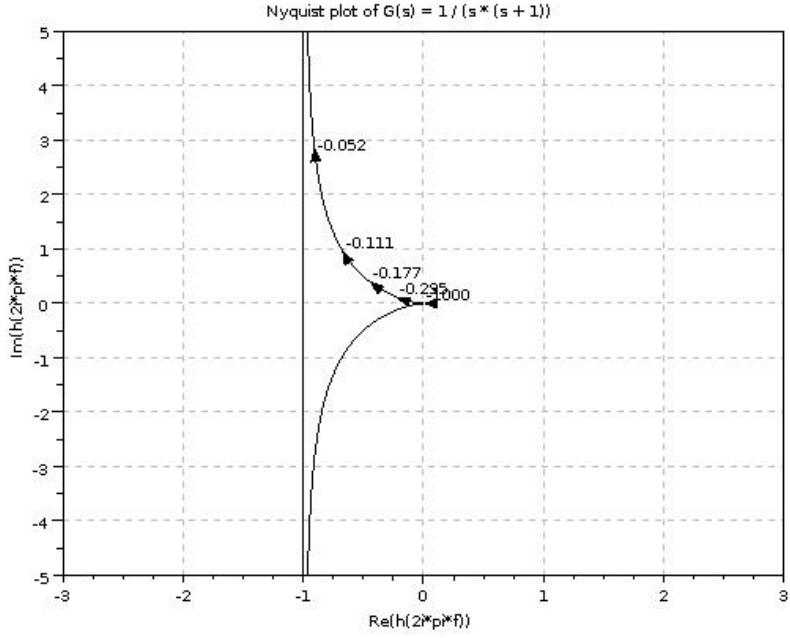


Figure 7.24: Nyquist Plots of system in state space

```

17 xtitle('Nyquist plot of G(s) = 1 / (s * (s + 1))');
18 a.data_bounds = [-3 -5 ; 3 5];
19 a.box = 'on';

```

Scilab code Exa 7.13 Nyquist Plot of MIMO system

```

1 // Example 7-13
2 // Nyquist Plot of MIMO system
3
4 clear; clc;
5 xdel(winsid()); // close all windows

```

```

6
7 A = [-1 -1 ; 6.5 0];
8 B = [1 1; 1 0];
9 C = [1 0; 0 1];
10 D = [0 0; 0 0];
11 G = syslin('c',A,B,C,D);
12 P = clean(ss2tf(G));
13
14 subplot(2,2,1);
15 nyquist(P(1,1),-100,100);
16 xgrid(color('gray'));
17 xtitle('Nyquist plot: From U1', 'Real Axis', 'To Y1');
18
19 subplot(2,2,2);
20 nyquist(P(2,1),-100,100);
21 xgrid(color('gray'));
22 xtitle('Nyquist plot: From U1', 'Real Axis', 'To Y2');
23
24 subplot(2,2,3);
25 nyquist(P(1,2),-100,100);
26 xgrid(color('gray'));
27 xtitle('Nyquist plot: From U2', 'Real Axis', 'To Y1');
28
29 subplot(2,2,4);
30 nyquist(P(2,2),-100,100);
31 xgrid(color('gray'));
32 xtitle('Nyquist plot From U2', 'Real Axis', 'To Y2');

```

Scilab code Exa 7.14 Nyquist Stability Check

```

1 // Example 7-14
2 // Nyquist Stability Check
3

```

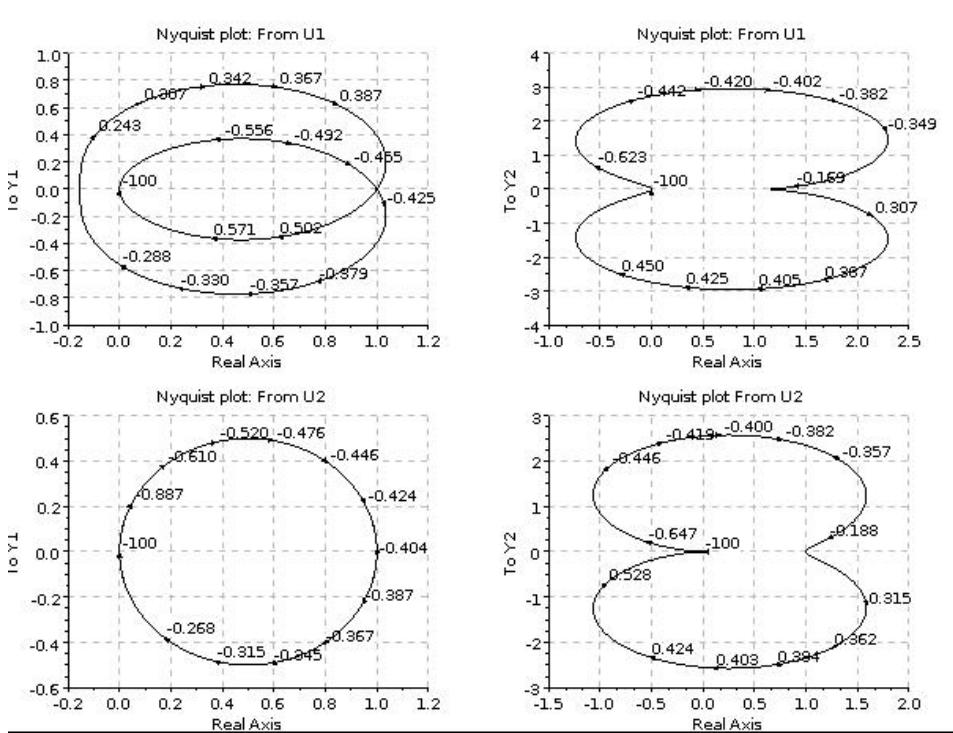


Figure 7.25: Nyquist Plot of MIMO system

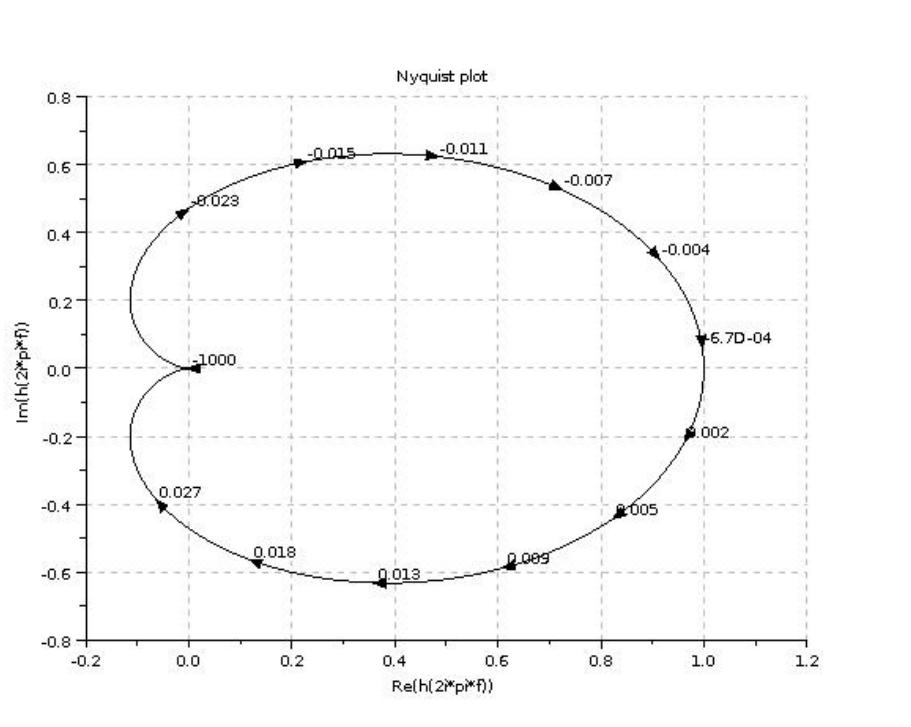


Figure 7.26: Nyquist Stability Check

```

4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 s = %s;
8 T1 = 5; T2 = 10;
9
10 K = 1;
11 den = (T1*s + 1)*(T2*s + 1);
12 GH = syslin('c',K,den);
13 nyquist(GH,-1000,1000);
14 xgrid(color('gray'));

```

Scilab code Exa 7.19 Nyquist plot stability check

```
1 // Example 7-19
2 // Nyquist plot stability check
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 s = %s;
8 num = s + 0.5;
9 den = s^3 + s^2 + 1;
10 disp(routh_t(den), 'routh table ='); // display the
    routh table
11 GbyK = syslin('c', num, den); // open loop system
12
13 nyquist(GbyK, -1000, 1000);
```

check Appendix AP 10 for dependency:

shmargins.sci

Scilab code Exa 7.20 Gain and phase margins for different K

```
1 // Example 7-20
2 // Gain and phase margins for different K
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 // please edit the path
8 // cd "/<your code directory>/";
9 // exec("shmargins.sci");
10
```

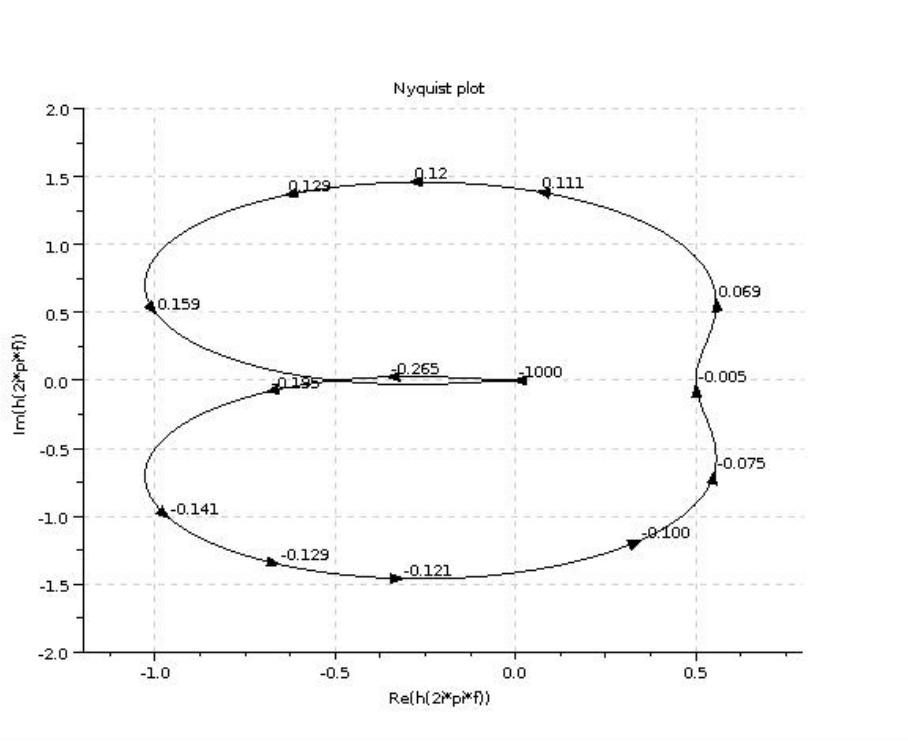


Figure 7.27: Nyquist plot stability check

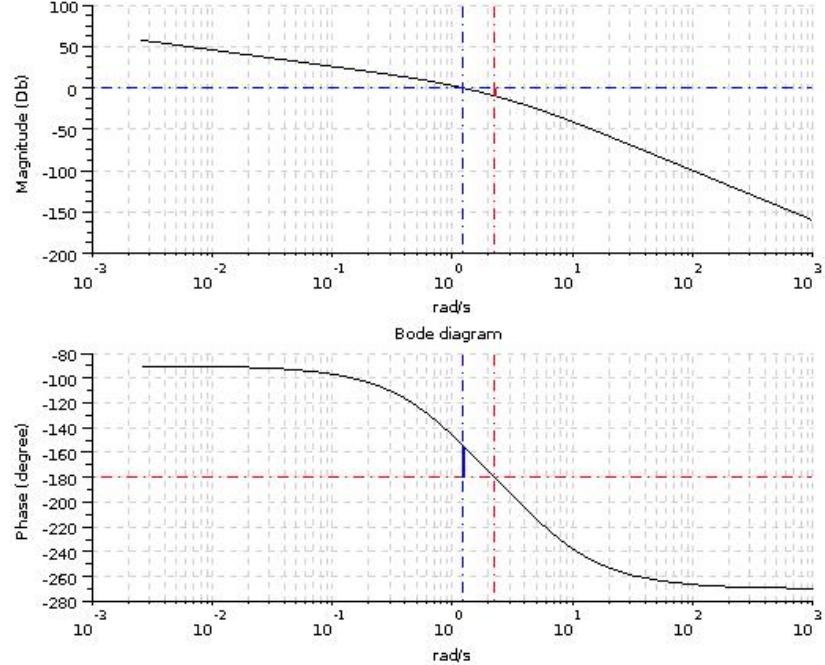


Figure 7.28: Gain and phase margins for different K

```

11 s = %s / 2 / %pi; // corrected for frequencies in
    rad/s
12 K = 10;
13 G = syslin('c', K, s*(s+1)*(s+5));
14 shmargins(G);
15 scf();
16 K = 100;
17 G = syslin('c', K, s*(s+1)*(s+5));
18 shmargins(G);

```

check Appendix AP 10 for dependency:

`shmargins.sci`

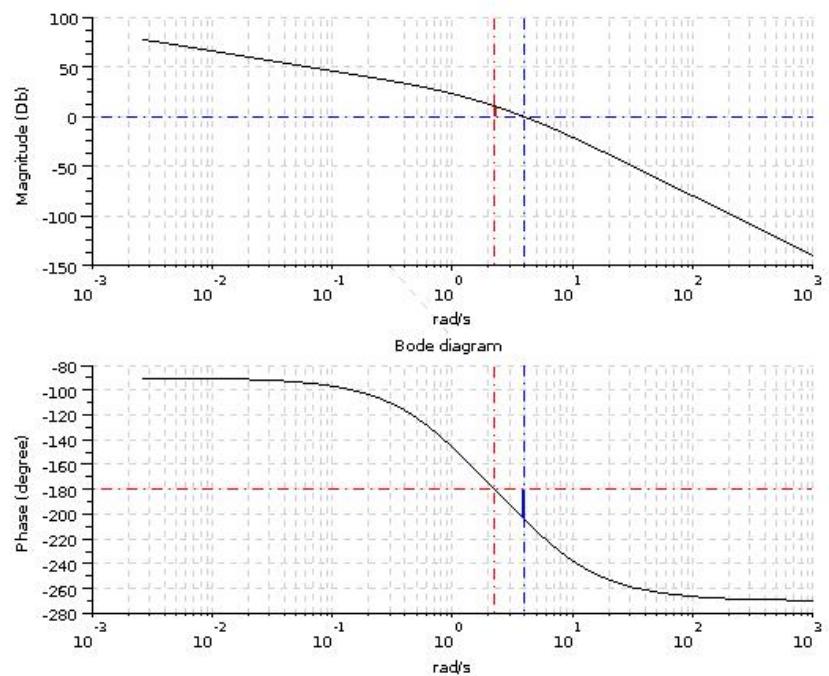


Figure 7.29: Gain and phase margins for different K

Scilab code Exa 7.21 Stability Margins

```
1 // Example 7-21
2 // Stability Margins
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 // please edit the path
8 // cd "/<your code directory>/";
9 // exec("shmargins.sci");
10
11 s = %s /2 / %pi; // corrected for frequencies in
12 num = 20*(s+1);
13 den =s * (s + 5) * (s^2 + 2*s + 10);
14 G = syslin('c',num,den);
15 shmargins(G);
```

check Appendix AP 2 for dependency:

plotresp.sci

Scilab code Exa 7.22 Correlating bandwidth and speed of response

```
1 // Example 7-22
2 // Correlating bandwidth and speed of response
3
4 clear; clc;
5 xdel(winsid()); // close all windows
```

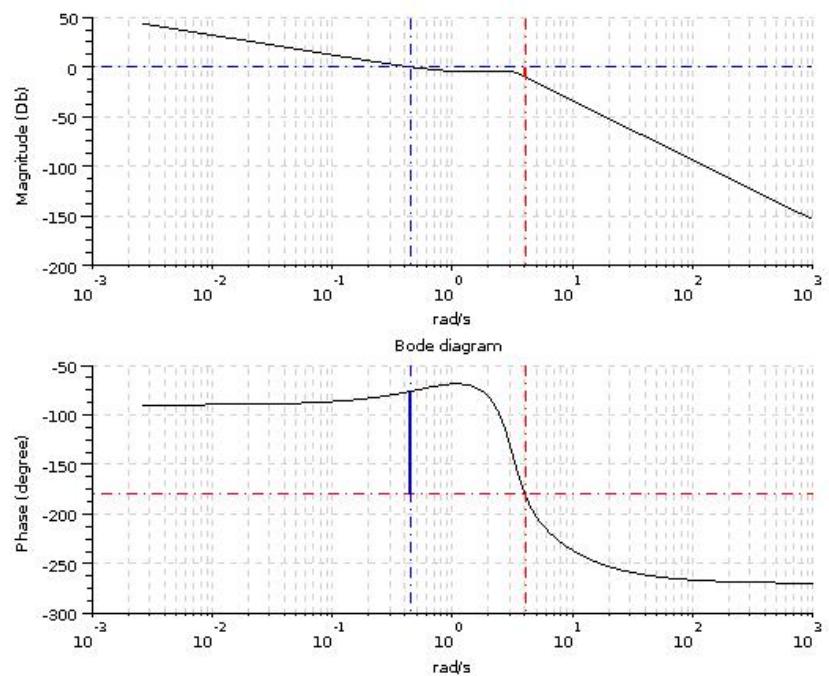


Figure 7.30: Stability Margins

```

6
7 // please edit the path
8 // cd "/<your code directory>";
9 // exec("plotresp.sci");
10
11 s = %s /2 /%pi; // frequencies in rad/s
12 G1 = syslin('c',1,s + 1);
13 G2 = syslin('c',1,3*s + 1);
14 subplot(2,1,1);
15 gainplot(G1,0.1,10);
16 xtitle('system 1 : 1 / (s + 1)', 'rad/s');
17 subplot(2,1,2);
18 gainplot(G2,0.1,10);
19 xtitle('system 2 : 1 / (3*s + 1)', 'rad/s');
20
21 scf();
22 t = 0:0.05:1;
23 u = ones(1,length(t));
24 subplot(2,1,1);
25 plotresp(u,t,G1,'');
26 plotresp(u,t,G2,'Step response of two systems with
different bandwidth');
27 xstring(0.1,0.75,'System 1');
28 xstring(0.35,0.4,'System 2');
29
30 subplot(2,1,2);
31 plotresp(t,t,G1,'');
32 plotresp(t,t,G2,'Ramp response of two systems with
different bandwidth');
33 xstring(0.45,0.35,'System 1');
34 xstring(0.8,0.45,'System 2');

```

check Appendix AP 11 for dependency:

freqch.sci

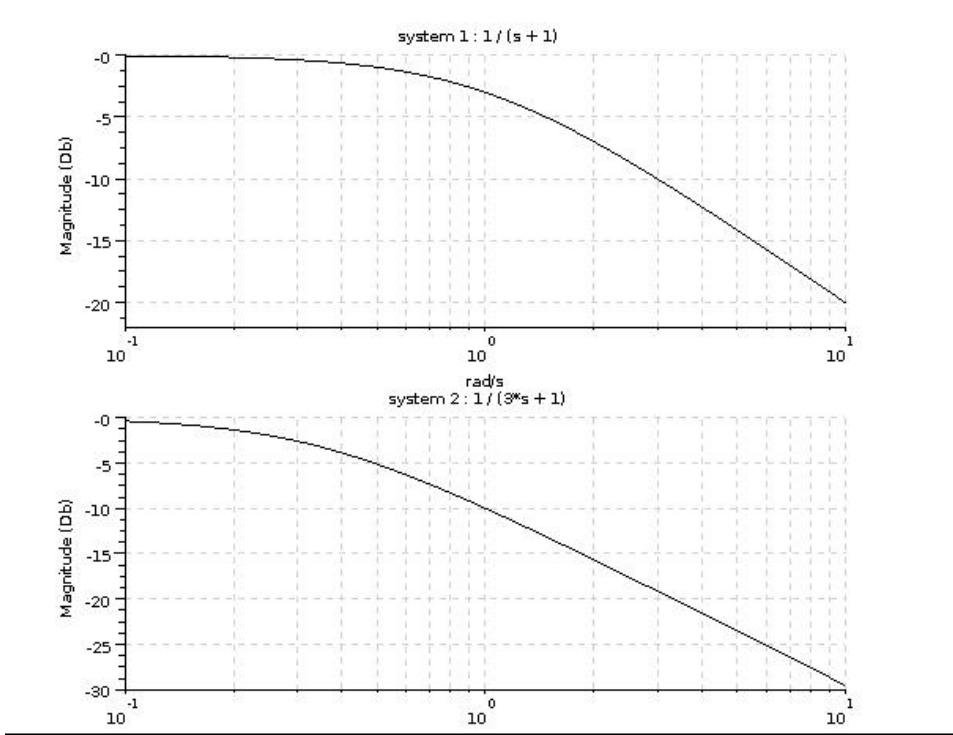


Figure 7.31: Correlating bandwidth and speed of response

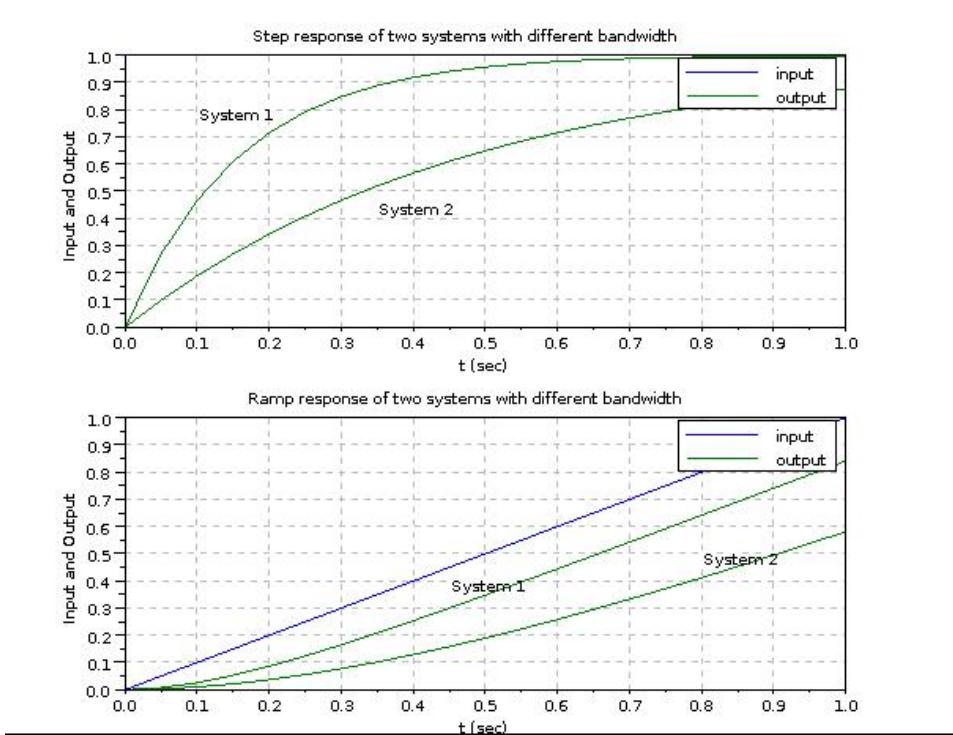


Figure 7.32: Correlating bandwidth and speed of response

Scilab code Exa 7.23 Frequency characteristics

```
1 // Example 7-23
2 // Frequency characteristics
3 clear; clc;
4 xdel(winsid()); // close all windows
5
6 // please edit the path
7 // cd "/<your code directory>/";
8 // exec("freqch.sci");
9
10 s = %s /2 /%pi; // frequencies in rad/s
11 G = 1 / (s * (0.5*s + 1) * (s + 1));
12 H = syslin('c', G /. 1);
13 omega = logspace(-1, 1, 200);
14
15 [Mr wr bw repf] = freqch(H, omega);
16 bode(omega, repf);
17 xtitle('Bode Diagram', 'rad/s');
18 a = gcf(); set(a.children(1).x_label, 'text', 'rad/s');
```

check Appendix AP 9 for dependency:

`s polarplot.sci`

Scilab code Exa 7.24 Polar and Nichols plot with M circles

```
1 // Example 7-24
2 // Polar and Nichols plot with M circles
3
```

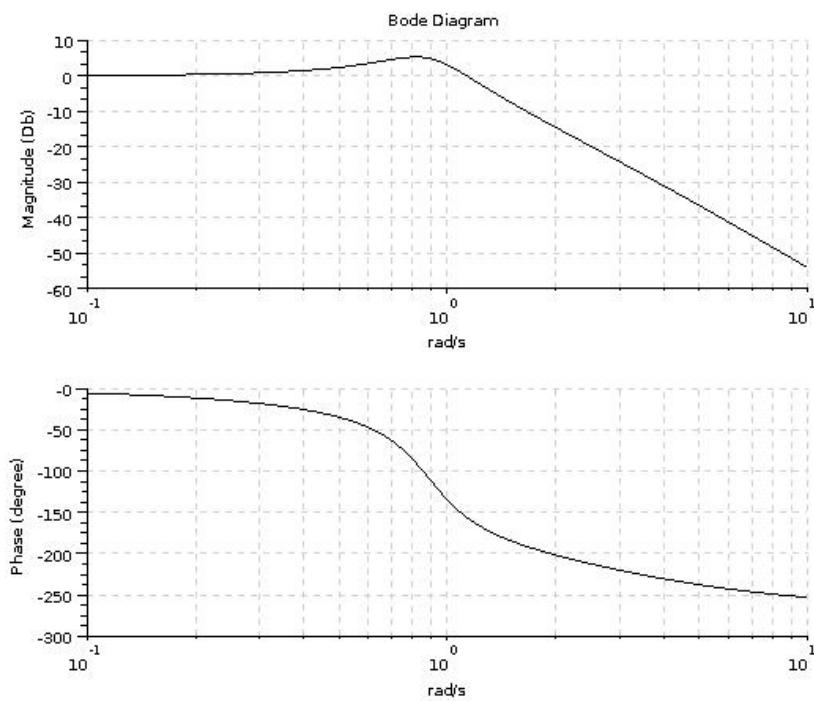


Figure 7.33: Frequency characteristics

```

4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 // please edit the path
8 // cd "/<your code directory >/";
9 // exec("spolarplot.sci");
10
11 s = %s;
12 G = syslin('c', 1, s*(s+1));
13 omega = logspace(-2, 2, 100);
14 repf = spolarplot(G, omega);
15
16 scf();
17 black(omega, repf);
18 chart([1.4], [], list(1, 0));
19 xgrid(color('gray'));
20 xstring(-150, 8, 'Mr = 1.4')

```

Scilab code Exa 7.25 Verifying experimentally derived Transfer function

```

1 // Example 7-25
2 // Verifying experimentally derived Transfer
   function
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 s = %s;
8 num = 320*(s + 2);
9 den = s * (s + 1) * (s^2 + 8*s + 64);
10 G = syslin('c', num, den);

```

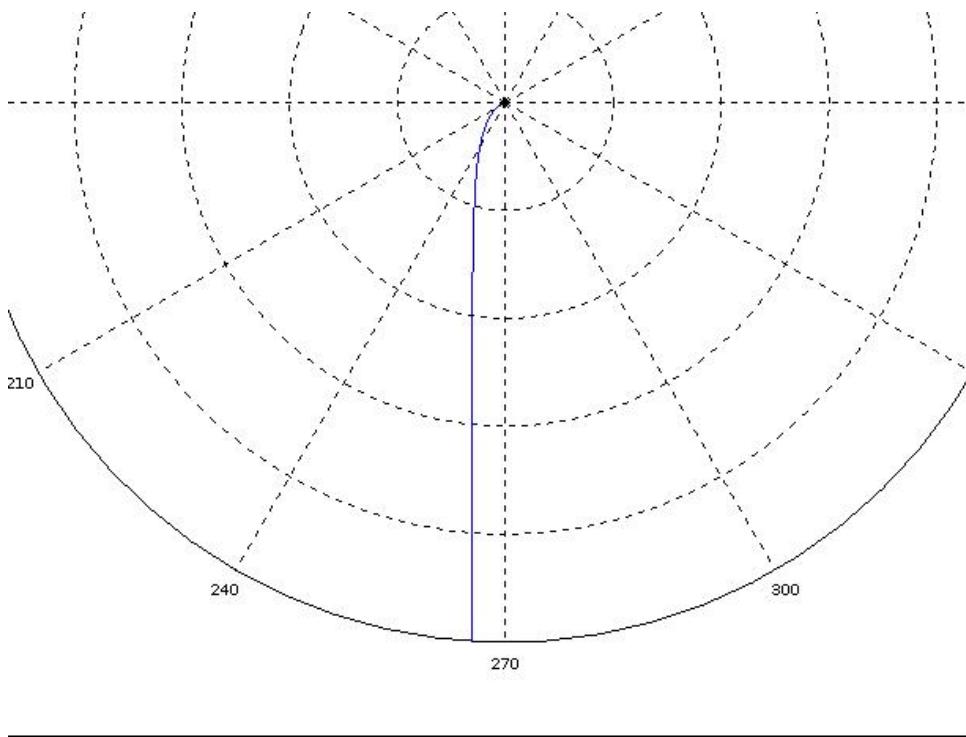


Figure 7.34: Polar and Nichols plot with M circles

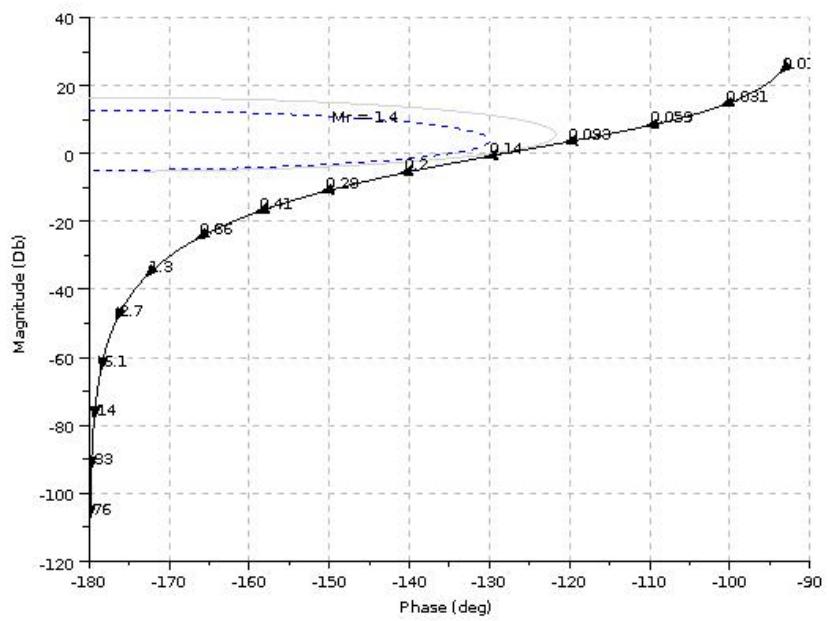


Figure 7.35: Polar and Nichols plot with M circles

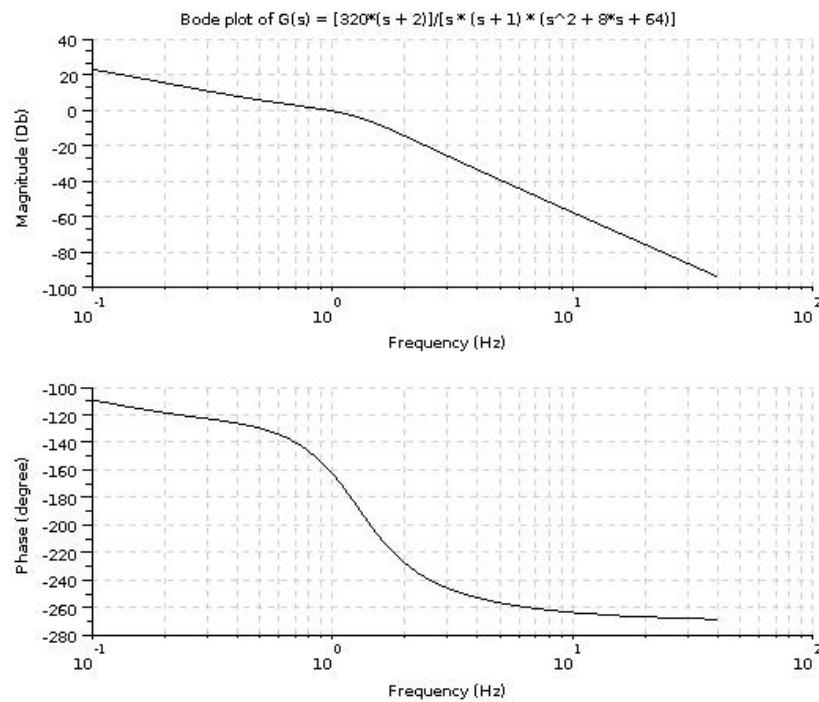


Figure 7.36: Verifying experimentally derived Transfer function

```

11
12 bode(G,0.1,40);
13 xtitle('Bode plot of G(s) = [320*(s + 2)]/[ s * (s +
1) * (s^2 + 8*s + 64)]');

```

Scilab code Exa 7.26.1 Design of Lead compensator with Bode plots

```

1 // Example 7-26-1
2 // Design of Lead compensator with Bode plots
3
4 clear; clc;

```

```

5 xdel(winsid()); // close all windows
6 mode(0);
7
8 // please edit the path
9 // cd "/<your code directory >/";
10 // exec("shmargins.sci");
11
12 s = %s/2/%pi;
13 G = 4 / (s * (s + 2));
14 Kv = 20;
15 K = Kv / horner(s * G, 0)
16
17 GK = syslin('c', K * G);
18
19 [gm, gcrw, pm, pcrw] = shmargins(GK);
20 // required specification is pm = 50 degrees
21 phi = 50 - pm + 6 // 6 deg compensation
22 sn = sind(phi);
23 alpha = (1 - sn)/(1 + sn)
24
25 wc = 9; // new gain crossover freq.
26 z = wc * sqrt(alpha) // z = 1 / T
27 p = wc / sqrt(alpha) // p = 1 / (alpha*T)
28 Kc = K / alpha
29 disp(Kc * (%s + z)/(%s + p), 'Gc = ');
30 Gc = Kc * (s + z)/(s + p);
31 GGc = syslin('c', Gc * G);
32 scf();
33 shmargins(GGc);

```

check Appendix AP 10 for dependency:

`shmargins.sci`

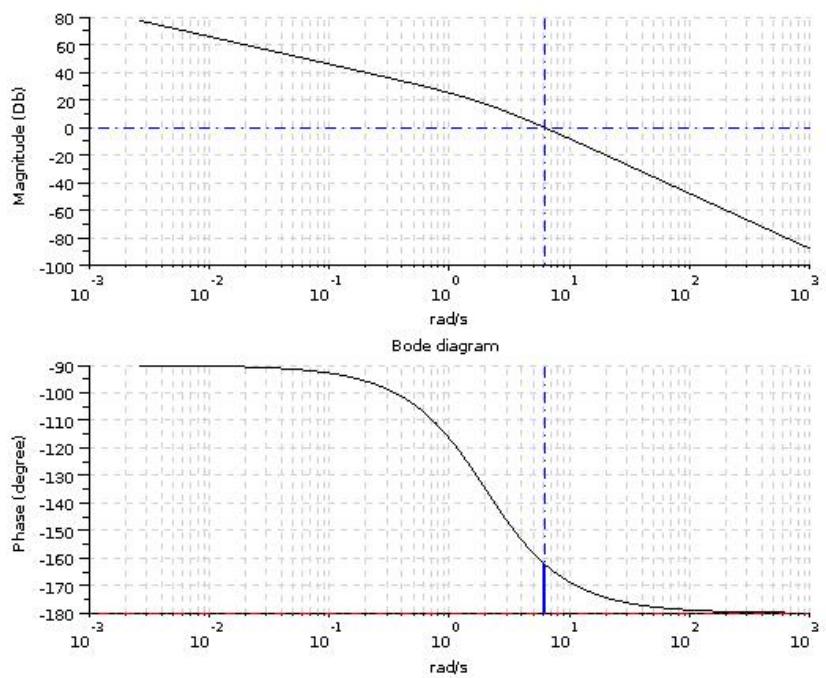


Figure 7.37: Design of Lead compensator with Bode plots

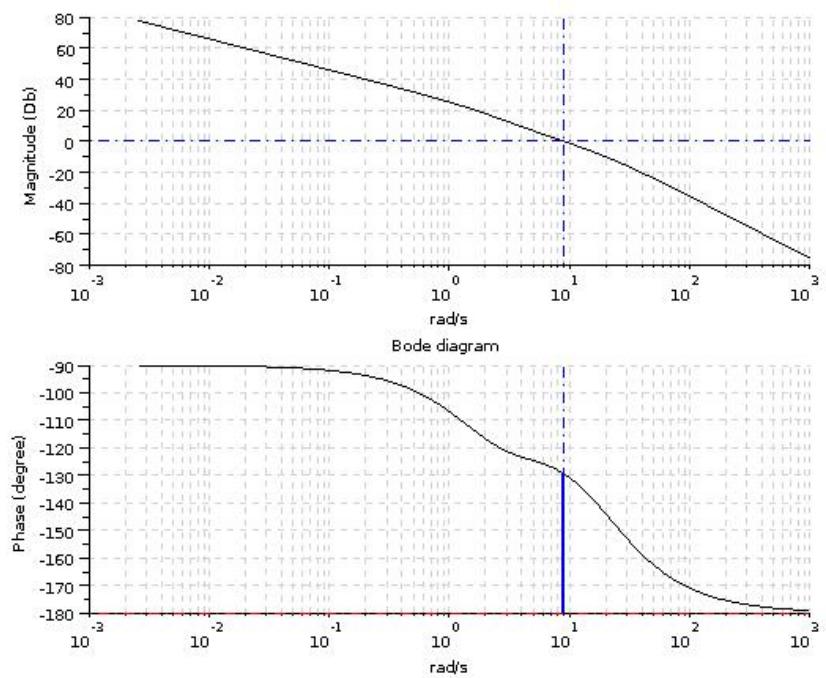


Figure 7.38: Design of Lead compensator with Bode plots

Scilab code Exa 7.26.2 Evaluating Lead compensated system

```
1 // Example 7-26-2
2 // Evaluating Lead compensated system
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 // please edit the path
8 // cd "/<your code directory>/";
9 // exec("plotresp.sci");
10
11 s = %s;
12 G = 4 / (s * (s + 2));
13
14 Kc = 42.104125;
15 z = 4.3861167;
16 p = 18.467361;
17 Gc = Kc * (s + z)/(s + p);
18 GGc = G*Gc;
19
20 H = syslin('c',G /. 1);
21 Hc = syslin('c',GGc /. 1);
22
23 t = 0:0.05:5;
24 u1 = ones(1,length(t)); //step response
25 u2 = t; //ramp response
26
27 subplot(2,1,1);plotresp(u1,t,H,'');
28 plotresp(u1,t,Hc,'Unit step response');
29 xstring(0.65,0.55,'uncompensated system');
30 xstring(0.1,1.2,'compensated system');
31 subplot(2,1,2);plotresp(u2,t,H,'');
32 plotresp(u2,t,Hc,'Unit ramp response');
```

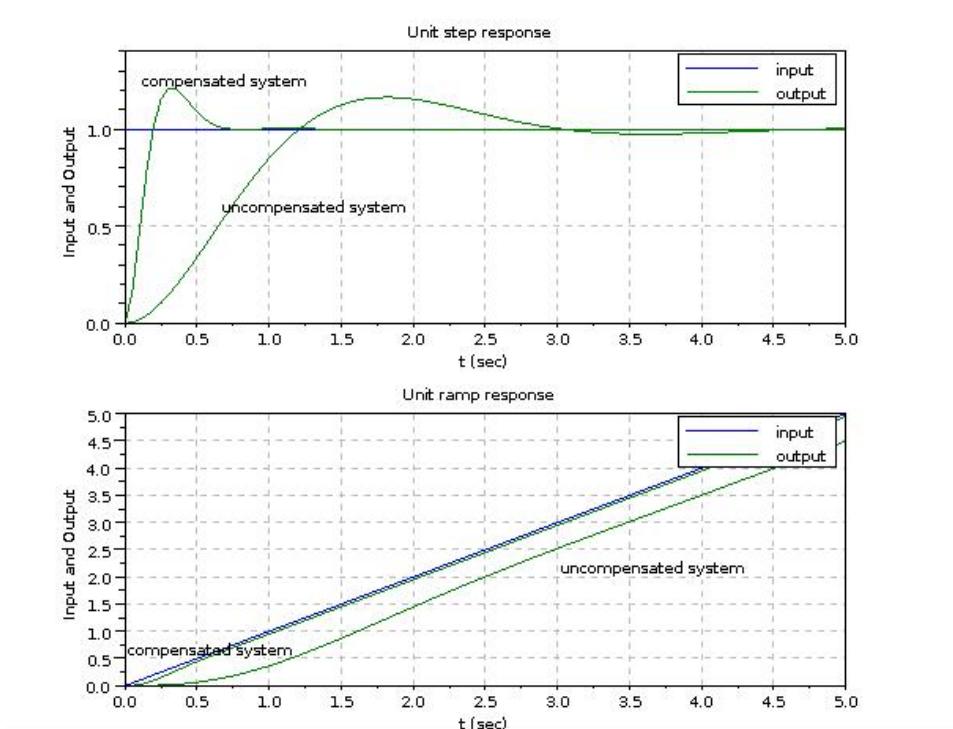


Figure 7.39: Evaluating Lead compensated system

```
33 xstring(3.0,2.0,'uncompensated system');
34 xstring(0,0.5,'compensated system');
```

check Appendix AP 2 for dependency:

`plotresp.sci`

Scilab code Exa 7.27.1 Design of Lag compensator with Bode plots

```
1 // Example 7-27-1
2 // Design of Lag compensator with Bode plots
3
```

```

4 clear; clc;
5 xdel(winsid()); // close all windows
6 mode(0);
7
8 // please edit the path
9 // cd "/<your code directory>/";
10 // exec("shmargins.sci");
11
12 s = %s/2/%pi;
13 G = 1 / (s * (s + 1) * (0.5*s + 1));
14 Kv = 5;
15 K = Kv / horner(s * G, 0)
16
17 GK = syslin('c', K * G);
18
19 [gm, gcrw, pm, pcrw] = shmargins(GK);
20 // required specification is pm = 40 degrees
21
22 wc = 0.5; // new gain crossover freq.
23 beta = 10
24 z = 0.1 // z = 1 / T is chosen one octave less
25 p = z / beta
26 Kc = K / beta
27 disp(Kc * (%s + z)/(%s + p), 'Gc = ');
28 Gc = Kc * (s + z)/(s + p);
29 GGc = syslin('c', Gc * G);
30 scf();
31 shmargins(GGc);

```

check Appendix AP 10 for dependency:

shmargins.sci

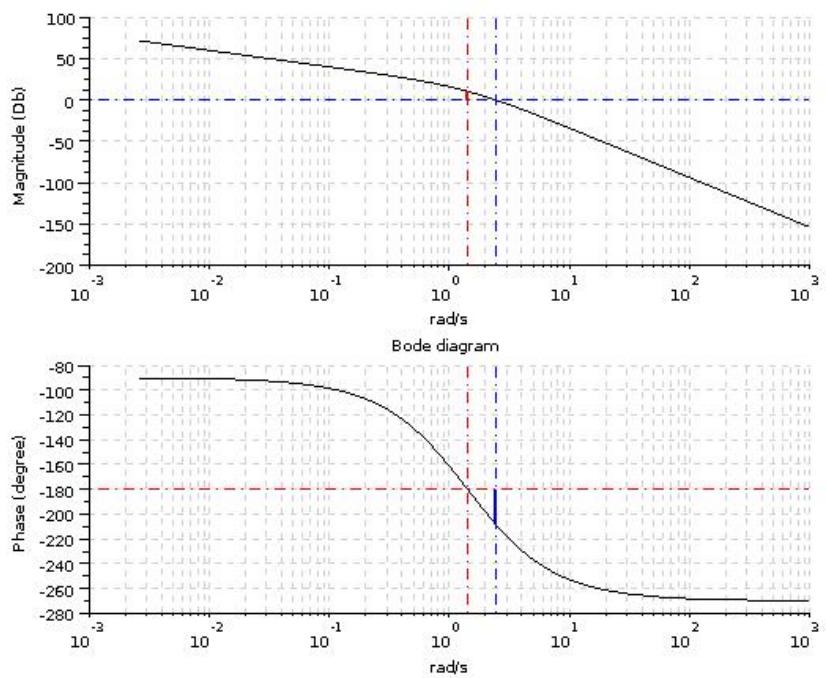


Figure 7.40: Design of Lag compensator with Bode plots

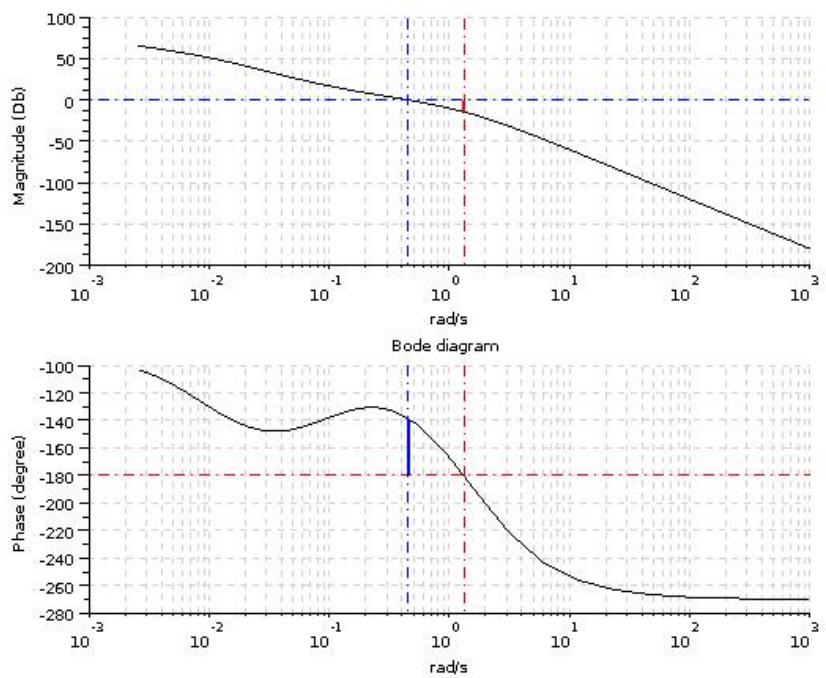


Figure 7.41: Design of Lag compensator with Bode plots

Scilab code Exa 7.27.2 Evaluating Lag compensated system

```
1 // Example 7-27-2
2 // Evaluating Lag compensated system
3
4 clear; clc;
5 xdel(winsid()); //close all windows
6
7 // please edit the path
8 // cd "/<your code directory >/";
9 // exec("plotresp.sci");
10
11 s = %s;
12 G = 1 / (s * (s + 1) * (0.5*s + 1));
13
14 Kc = 0.5;
15 z = 0.1;
16 p = 0.01;
17 Gc = Kc * (s + z)/(s + p);
18 GGc = G*Gc;
19
20 H = syslin('c',G /. 1);
21 Hc = syslin('c',GGc /. 1);
22
23 t = 0:0.5:40;
24 u1 = ones(1,length(t)); //step response
25
26 subplot(2,1,1);plotresp(u1,t,H,'');
27 plotresp(u1,t,Hc,'Unit step response');
28 xstring(2.5,0.55,'uncompensated system');
29 xstring(0.1,1.3,'compensated system');
30
31 t = 0:0.5:30;
32 u2 = t; //ramp response
33 subplot(2,1,2);plotresp(u2,t,H,'');
34 plotresp(u2,t,Hc,'Unit ramp response');
35 xstring(15,13,'uncompensated system');
36 xstring(14,20,'compensated system');
```

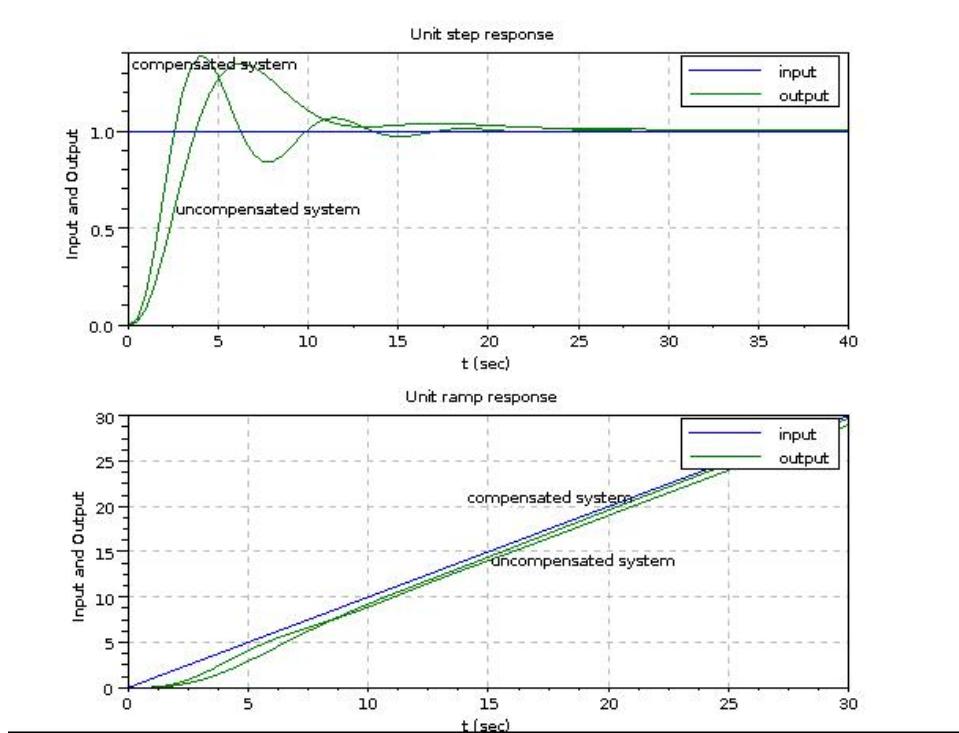


Figure 7.42: Evaluating Lag compensated system

check Appendix AP 2 for dependency:

`plotresp.sci`

Scilab code Exa 7.28.1 Design of Lag lead compensation with Bode plots

```

1 // Example 7-28-1
2 // Design of Lag - lead compensation with Bode plots
3
4 clear; clc;
5 xdel(winsid()); // close all windows

```

```

6 mode(0);
7
8 // please edit the path
9 // cd "<your code directory>/";
10 // exec("shmargins.sci");
11
12 s = %s /2 /%pi ;
13 G = 1 / (s * (s + 1) * (s + 2));
14 Kv = 10;
15 K = Kv / horner(s * G,0)
16 GK = syslin('c',K * G);
17
18 [gm, gcrw, pm, pcrw] = shmargins(GK);
19 wc = 1.5; // new gain crossover freq.
20
21 // required specification is pm = 50 degrees
22 phi = 55 // 6 deg compensation
23 sn = sind(phi);
24 beta = (1 + sn)/(1 - sn)
25
26 z2 = wc /10; // z2 = 1 / T2 :1 decade below our new
     gain cross freq.
27 p2 = z2 / beta;
28
29 disp((%s + z2)/(%s + p2), 'Gclead = ');
30 Gclead = (s + z2)/(s + p2);
31
32 z1 = 0.7 ; //corner frequencies are around w = 7 <->
     -20db
33 p1 = 7;
34 disp((%s + z1)/(%s + p1), 'Gclag = ');
35 Gclag = (s + z1)/(s + p1);
36
37 Gc = K * Gclag * Gclead;
38 GGc = syslin('c',Gc * G);
39 scf();
40 shmargins(GGc);

```

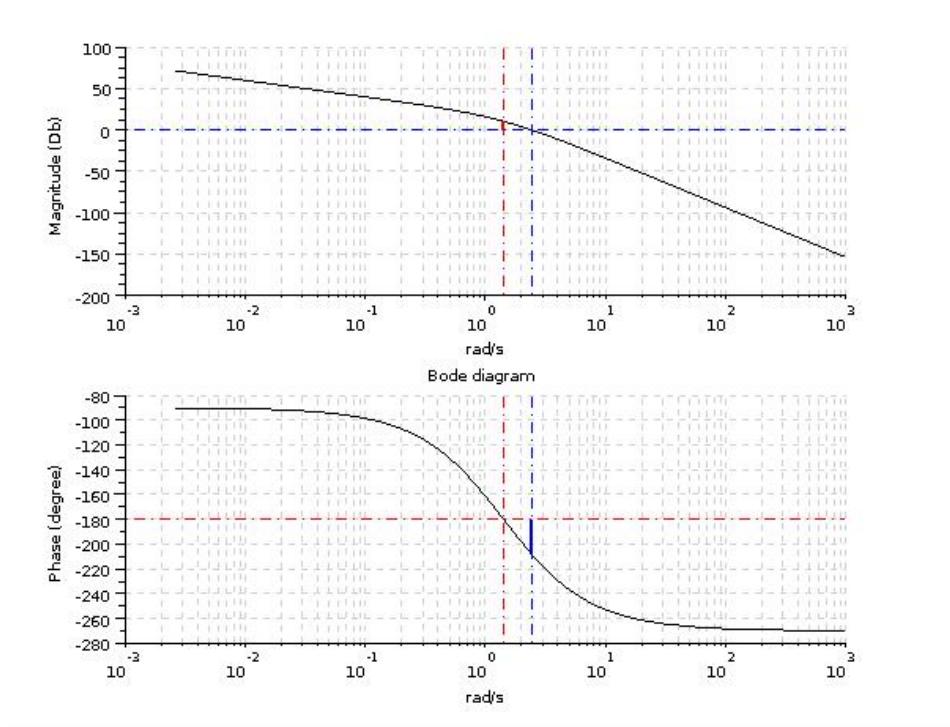


Figure 7.43: Design of Lag lead compensation with Bode plots

check Appendix AP 10 for dependency:

`shmargins.sci`

Scilab code Exa 7.28.2 Evaluating Lag Lead compensated system

```

1 // Example 7-26-2
2 // Evaluating Lag Lead compensated system
3

```

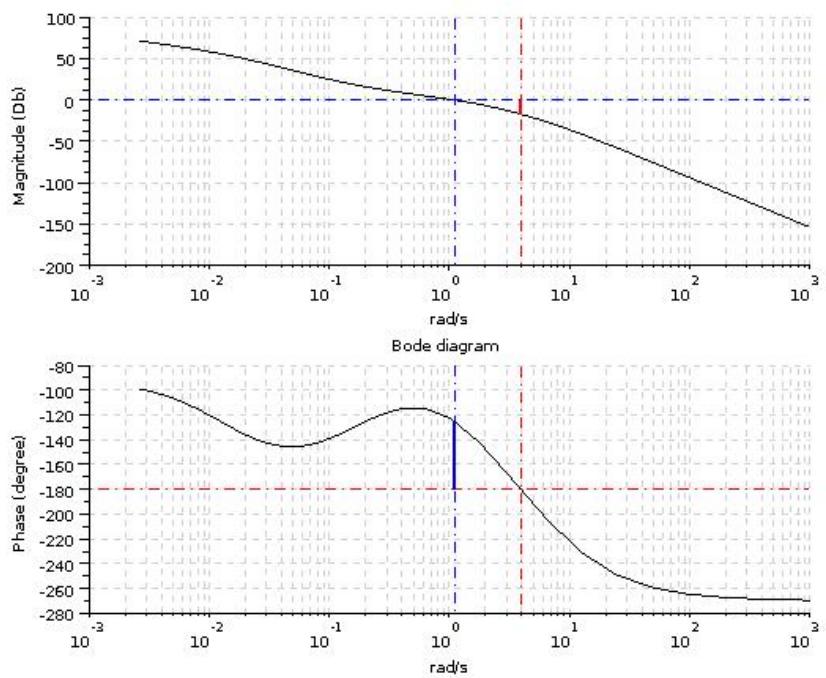


Figure 7.44: Design of Lag lead compensation with Bode plots

```

4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 // please edit the path
8 // cd "/<your code directory>/";
9 // exec("plotresp.sci");
10
11 s = %s;
12 G = 1 / (s * (s + 1) * (s + 2));
13
14 Gc = 20 * (s + 0.7) * (s + 0.15) / (s + 7) / (s +
    0.015);
15 GGc = G*Gc;
16
17 H = syslin('c',G /. 1);
18 Hc = syslin('c',GGc /. 1);
19
20 t = 0:0.1:30;
21 u1 = ones(1,length(t)); //step response
22 u2 = t; //ramp response
23
24 subplot(2,1,1);plotresp(u1,t,H,'');
25 plotresp(u1,t,Hc,'Unit step response');
26 xstring(3,0.8,'uncompensated system');
27 xstring(0.7,0.6,'compensated system');
28 subplot(2,1,2);plotresp(u2,t,H,'');
29 plotresp(u2,t,Hc,'Unit ramp response');
30 xstring(10,7,'uncompensated system');
31 xstring(2,0.5,'compensated system');

```

check Appendix AP 2 for dependency:

`plotresp.sci`

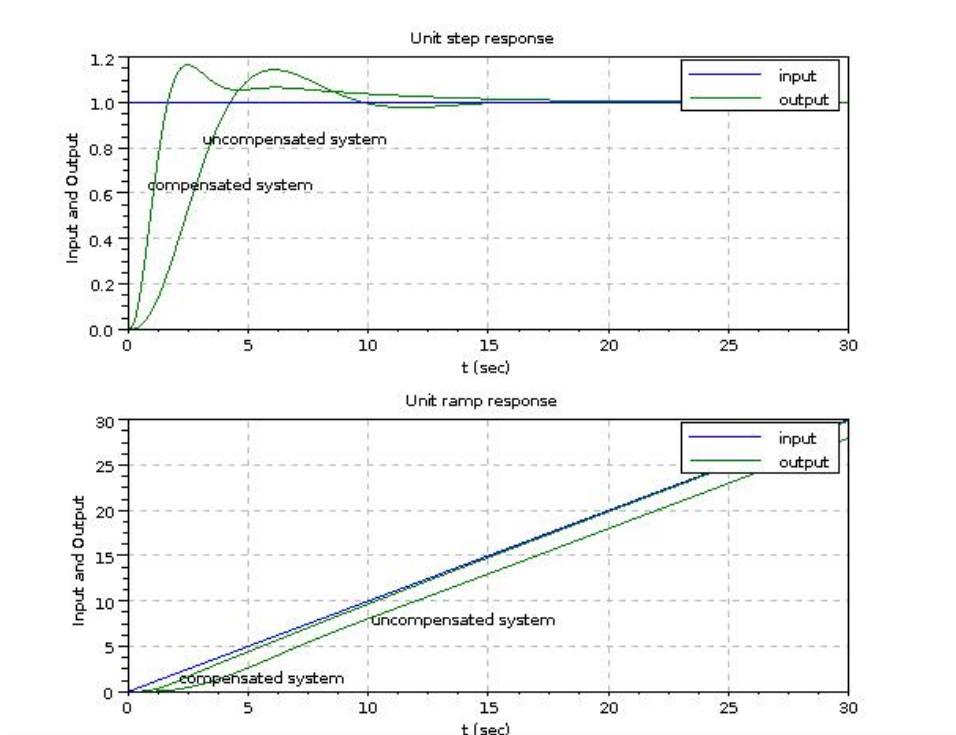


Figure 7.45: Evaluating Lag Lead compensated system

Chapter 8

PID Controllers and Modified PID Controllers

Scilab code Exa 8.i.1 PID Design with Frequency Response

```
1 // Illustration 8.1
2 // PID Design with Frequency Response
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6 mode(0);
7 // please edit the path
8 // cd "<your code directory>";
9 // exec("plotresp.sci");
10
11 s = %s;
12 G = syslin('c', 1, s^2 + 1);
13 Kv = 4;
14 K = Kv / abs(horner(G, 0))
15
```

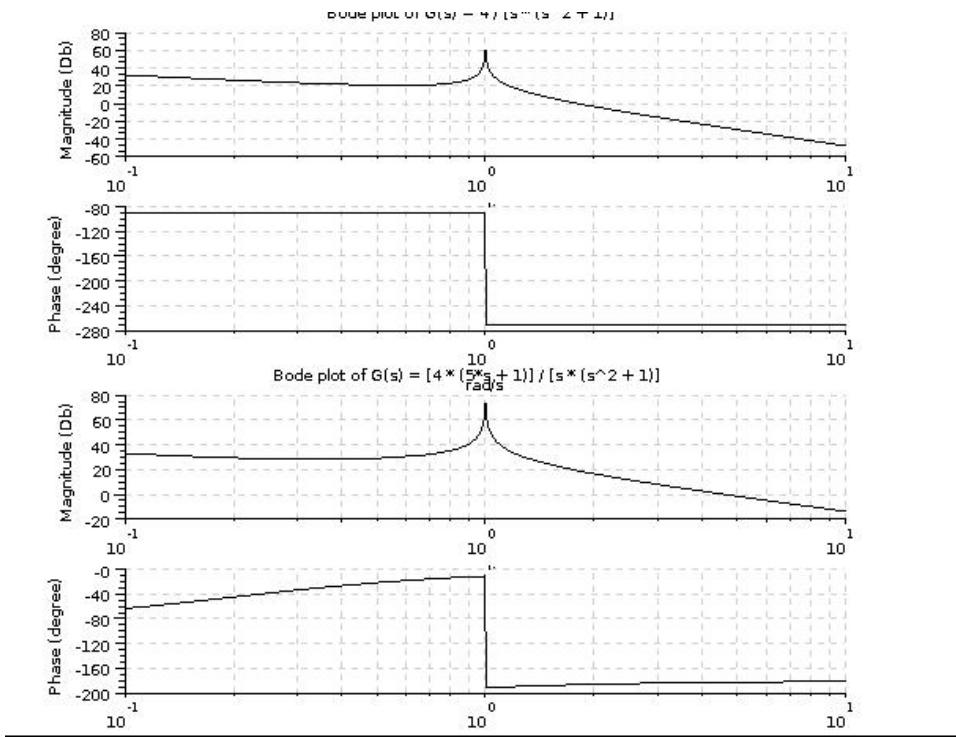


Figure 8.1: PID Design with Frequency Response

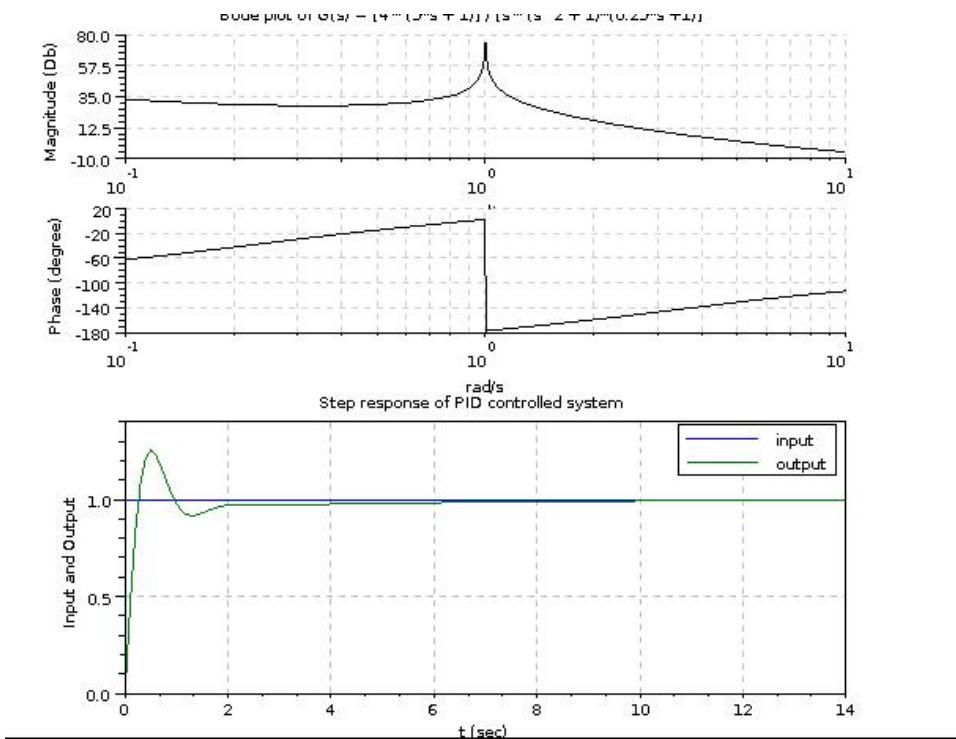


Figure 8.2: PID Design with Frequency Response

```

16 // Step 1 : Gain adjust
17 G2 = G * K / s
18 G2w = syslin('c', horner(G2, %s/2/%pi) ); // correction for frequencies in rad/s
19
20 omega = calfrq(G2w, 0.1, 10); // discretises such
    that the peak is // well
    represented
21 [db phi] = dbphi(repfreq(G2w, omega));
22 phi( 53:99 ) = -270;
23 subplot(2,1,1); bode(omega, db, phi);
24 xtitle('Bode plot of G(s) = 4 / [s * (s^2 + 1)] , , rad/s');
25 a = gcf(); set(a.children(1).x_label, 'text', 'rad/s');
26 disp(p_margin(G2w), 'Phase margin of G2 =');
27
28 // Step 2:
29 a = 5 // a is chosen to be 5;
30 G3 = G2 * (a*s + 1)
31 G3w = syslin('c', horner(G3, %s/2/%pi) );
32 subplot(2,1,2); bode(G3w, 0.1, 10);
33 xtitle('Bode plot of G(s) = [4 * (5*s + 1)] / [s * (s^2 + 1)] , , rad/s');
34 a = gcf(); set(a.children(1).x_label, 'text', 'rad/s');
35 disp(p_margin(G3w), 'Phase margin of G3 =');
36
37 // Step 3
38 scf();
39 b = 0.25
40 G4 = G3 * (b*s + 1)
41 G4w = syslin('c', horner(G4, %s/2/%pi) );
42 subplot(2,1,1); bode(G4w, 0.1, 10);
43 xtitle('Bode plot of G(s) = [4 * (5*s + 1)] / [s * (s^2 + 1)*(0.25*s + 1)] , , rad/s');
44 a = gcf(); set(a.children(1).x_label, 'text', 'rad/s');
45 disp(p_margin(G4w), 'Phase margin of G4 =');
46

```

```

47 C = syslin('c',G4 /. 1)
48 disp(roots(C.den), 'closed loop poles =');
49 t = 0:0.1:14;
50 u = ones(1,length(t));
51 subplot(2,1,2); plotresp(u,t,C, 'Step response of PID
controlled system');

```

check Appendix AP 2 for dependency:

`plotresp.sci`

Scilab code Exa 8.a.5 PID design

```

1 // Example A-8-5
2 // PID design
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6 mode(0);
7 // please edit the path
8 // cd "";
9 // exec("plotresp.sci");
10 // exec("stepch.sci");
11
12 s = %s;
13 zeta = 0.5 // dominant pole characteristics
14 wn = 4
15 sigma = zeta*wn;
16 ts = 4 /(zeta*wn);
17 disp(ts, 'settling time approximate (ts) =' );
18
19 D = (s + 10) * (s^2 + 2*zeta*wn*s + wn^2);
20 cf = coeff(D);
21
22 K = cf(1)
23 a_plus_b = (cf(2) - 9) / K

```

```

24 ab = (cf(3) - 3.6) / K
25
26 Gc = K * (ab * s^2 + a_plus_b *s+ 1) / s
27 CbyD = syslin('c',s,D)
28
29 CbyR = syslin('c',numer(Gc),D)
30
31 t = 0:0.05:5;
32 u = ones(1,length(t));
33 plotresp(u,t,CbyD,'Response to step disturbance
    input');
34 a = gca(); a.data_bounds = [0 , -4D-3; 5 , 14D-3];
35 scf();
36 [Mp ,tp ,tr ,ts] = stepch(CbyR,0,5,0.05,0.02);
37 disp(Mp,'Max overshoot =');
38 disp(ts,'settling time actual (ts) =');

```

check Appendix [AP 2](#) for dependency:

`plotresp.sci`

check Appendix [AP 8](#) for dependency:

`stepch.sci`

Scilab code Exa 8.a.6 PID design

```

1 // Example A-8-6
2 // PID Design
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6 mode(0);

```

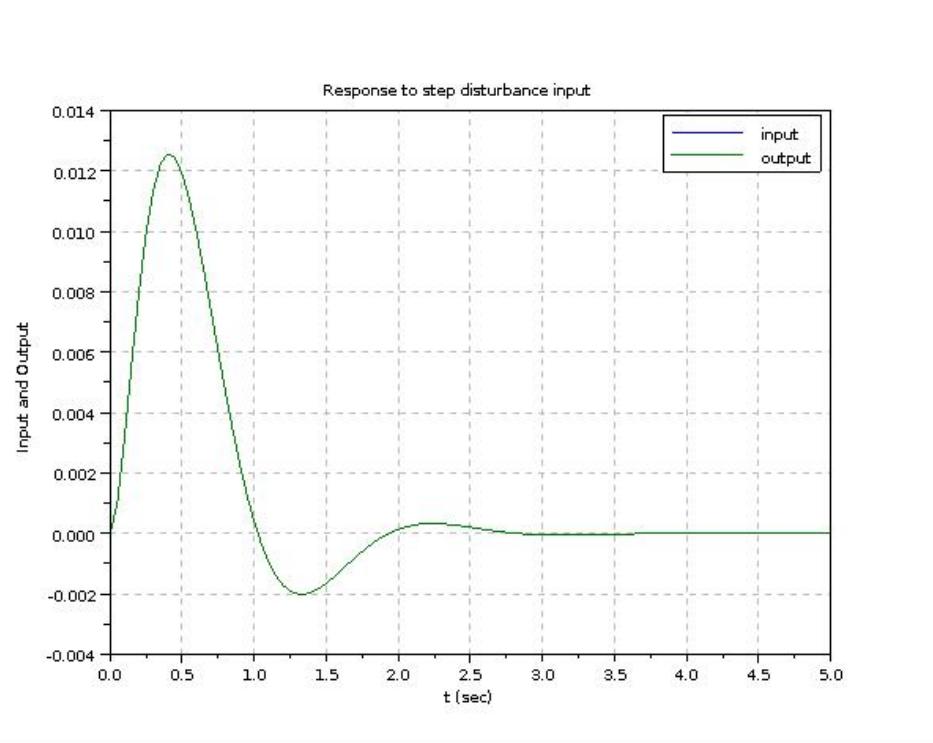


Figure 8.3: PID design

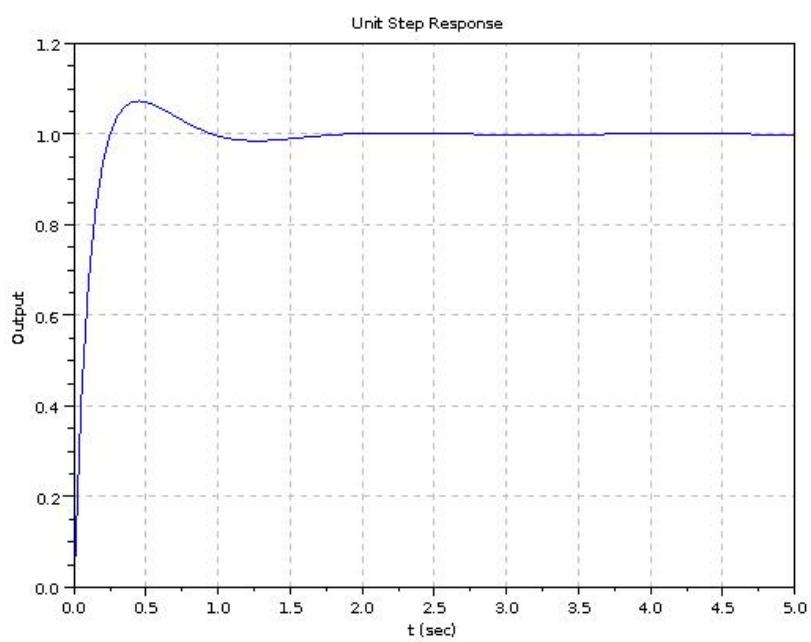


Figure 8.4: PID design

```

7
8 // please edit the path
9 // cd "<your code directory>";
10 // exec("plotresp.sci");
11 // exec("rootl.sci")
12
13 s = %s;
14 G = syslin('c', 1, s^2 + 1);
15 dp = -1 + sqrt(3)*%i;
16
17 angdef = 180 - phasemag(horner(G*(s+1)/s, dp))
18 // Determining b
19 b = 1 + sqrt(3)*cotd(angdef)
20 Gc1 = (s + 1) * (s + b) / s;
21 K = 1/ abs(horner(G*Gc1, dp))
22 Gc = K * Gc1
23
24 evans(G*Gc1, 50);
25 xgrid();
26 a = gca();
27 a.data_bounds = [-5 -3; 1 3];
28 a.children(1).visible = 'off';
29 xtitle('Root locus plot of open loop system');
30
31
32 C = syslin('c', G*Gc /. 1);
33 disp(C, 'closed loop system =');
34 scf();
35 t = 0:0.05:12;
36 u = ones(1, length(t));
37 plotresp(u, t, C, 'Unit step response of compensated
    system');

```

check Appendix AP 2 for dependency:

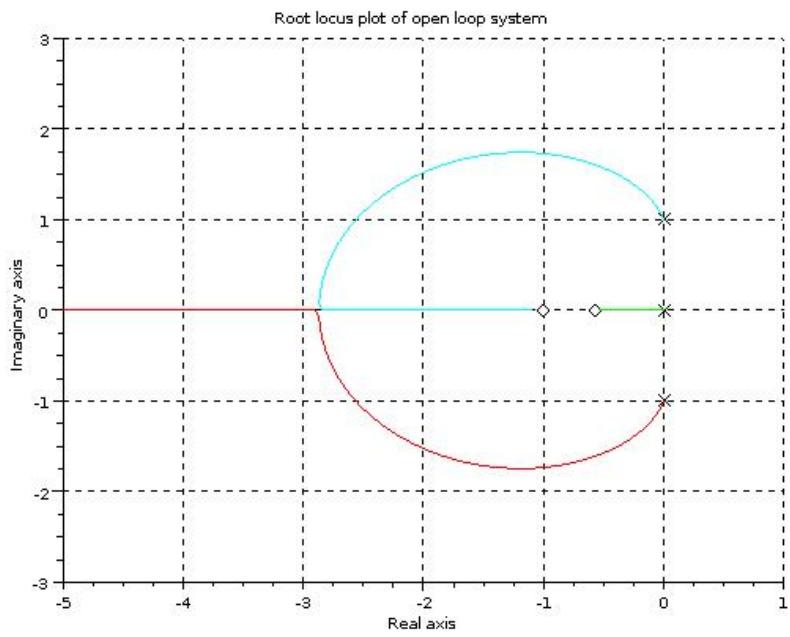


Figure 8.5: PID design

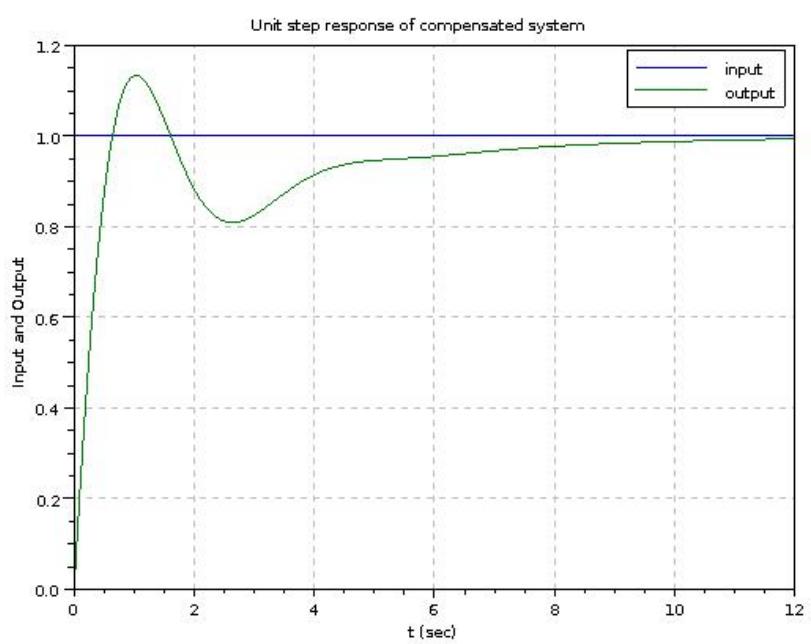


Figure 8.6: PID design

plotresp.sci

check Appendix AP 7 for dependency:

rootl.sci

Scilab code Exa 8.a.7.1 PID Design with Frequency Response

```
1 // Example A-8-7-1
2 // PID Design with Frequency Response
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6 mode(0);
7
8 // please edit the path
9 // cd "<your code directory>";
10 // exec("plotresp.sci");
11
12 s = %s;
13 Gp = syslin('c', s + 0.1, s^2 + 1);
14 Kv = 4;
15 K = Kv / abs(horner(Gp, 0))
16
17 // Step 1 : Gain adjust
18 G1 = Gp * K / s
19 G1w = syslin('c', horner(G1, %s/2/%pi)); // correction for frequencies in rad/s
20
21
22 subplot(2,1,1); bode(G1w);
23 xtitle('Bode plot of G(s) = 40*(s + 0.1)/[s*(s^2 + 1)]', 'rad/s');
24 a = gcf(); set(a.children(1).x_label, 'text', 'rad/s');
25 disp(p_margin(G1w), 'Phase margin of G =');
```

```

27 // Step 2:
28 a = 0.1526;
29 GGc = G1 * (a*s + 1)
30 GGcw = syslin('c', horner(GGc, %s/2/%pi) );
31 subplot(2,1,2); bode(GGcw,0.1,10);
32 xtitle('Bode plot of G*Gc = [4 *(0.1526*s + 1)*(s +
    0.1)]/[s*(s^2 + 1)]', 'rad/s');
33 a = gcf(); set(a.children(1).x_label,'text','rad/s');
34 disp(p_margin(GGcw), 'Phase margin of G*Gc =');
35 disp(g_margin(GGcw), 'Gain margin of G*Gc =');
36
37 scf();
38 C = syslin('c',GGc /. 1)
39 disp(roots(C.den), 'closed loop poles =');
40 t = 0:0.05:10;
41 u = ones(1,length(t));
42 subplot(2,1,1); plotresp(u,t,C, 'Step response of PID
    controlled system');
43 subplot(2,1,2); plotresp(t,t,C, 'Ramp response of PID
    controlled system');

```

check Appendix [AP 2](#) for dependency:

`plotresp.sci`

Scilab code Exa 8.a.12 Computing optimal solution

```

1 // Example A-8-12
2 // Computing optimal solution
3
4 clear; clc;
5 xdel(winsid()); // close all windows

```

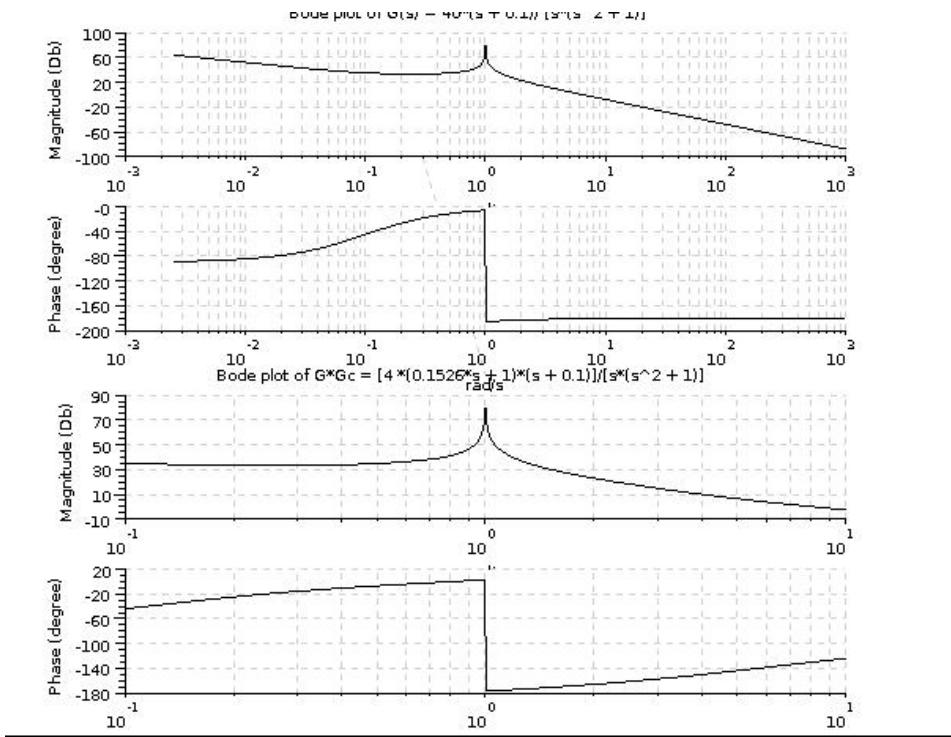


Figure 8.7: PID Design with Frequency Response

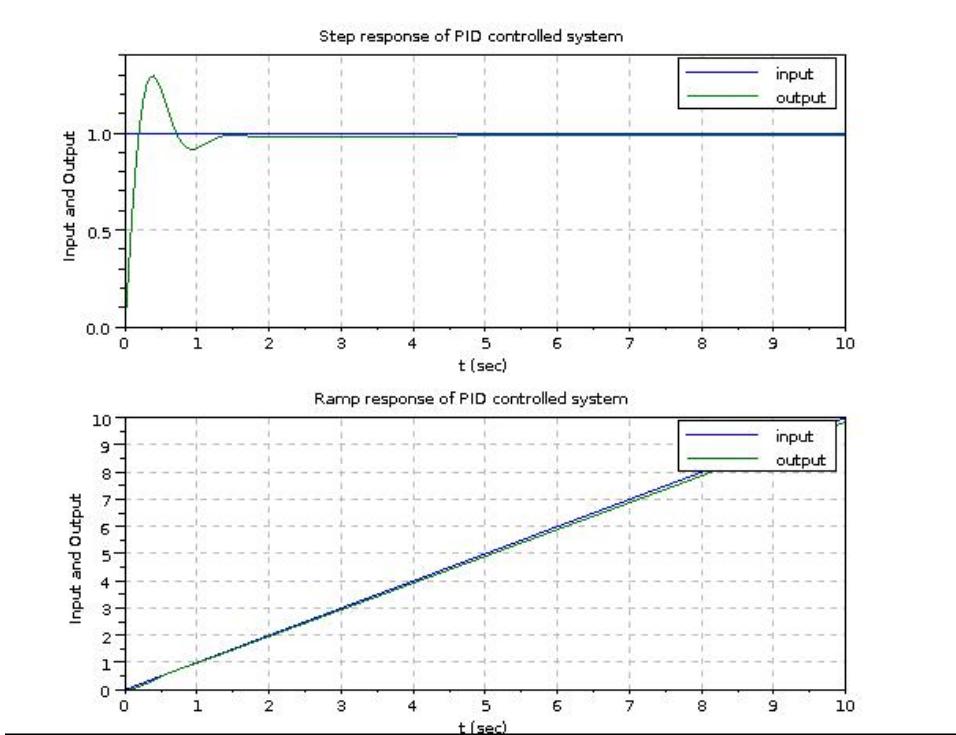


Figure 8.8: PID Design with Frequency Response

```

6
7 s = %s;
8 t = 0:0.1:5; u = ones(1,length(t));
9 t1 = 0:0.01:5;N =length(t1); u1 = ones(1,N);
10
11 k = 0;
12 mprintf('Processing ... \n');
13 for K = 50:-1:2
14     for a = 2:-0.05:0.05
15         num = K * ((s + a)^2) ;
16         den = s * s * (s^2 + 6*s + 5);
17         G = syslin('c',num,num + den);
18         y = csim(u,t,G);
19         m = max(y);
20         if m < 1.1 & m > 1.00 then;
21             y = csim(u1,t1,G);
22             if m < 1.1 & m > 1.02 then;
23                 l = N;
24                 while y(l) > 0.98 & y(l) < 1.02 ; l = l-1;
25                     end
26                     ts = (l-1)*0.01;
27                     if ts < 3.0;
28                         k= k + 1;
29                         solution(k,:) = [K a m ts];
30                     end
31     end
32
33 end
34 mprintf('completed %d%%\n',(50 - K)/48*100);
35 end
36 disp(solution,'solution = ');
37
38 // sort the solution set
39 [x 0] = gsort(solution(:,3), 'r', 'i');
40 for i = 1:k
41     sortsolution(i,:) = solution(0(i),:);
42 end

```

```

43 disp(sortsolution,'sortsolution = ');
44
45 x = sortsolution(7,:); K = x(1); a = x(2)
46 num = K * ((s + a)^2) ;
47 den = s * s * (s^2 + 6*s + 5);
48 G = syslin('c',num,num + den);
49 y1 = csim('step',t1,G);
50
51 x = sortsolution(2,:); K = x(1); a = x(2)
52 num = K * ((s + a)^2) ;
53 den = s * s * (s^2 + 6*s + 5);
54 G = syslin('c',num,num + den);
55 y2 = csim('step',t1,G);
56 plot(t1,y1,t1,y2);
57 xgrid();
58 xtitle('Unit Step response curves','t (sec)', 'output');
59 legend('K = 29 , a = 0.25', 'K = 27 , a = 0.2');

```

Scilab code Exa 8.a.13 Design of system with two degrees of freedom

```

1 // Example A-8-13
2 // Design of system with two degrees of freedom
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6 mode(0);
7 // please edit the path
8 // cd "<path to dependencies";
9 // exec("plotresp.sci");
10
11 s = %s;
12 Gp = 100 / (s*(s + 1))

```

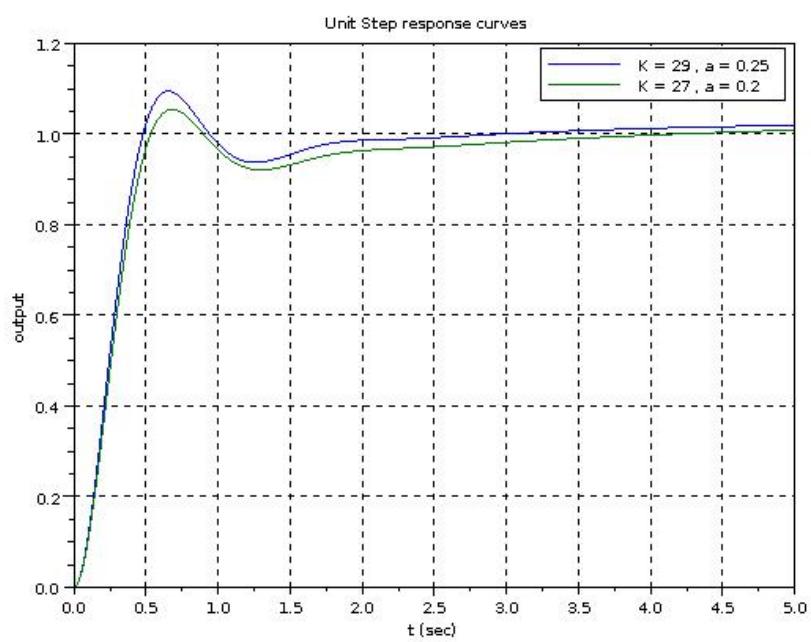


Figure 8.9: Computing optimal solution

```

13 dp = -5 + %i*5;
14
15 // Step 1: Design of Gc1 using root locus approach
16 angdef = 180 - phasemag(horner(Gp/s,dp))
17 angdef2 = angdef /2;
18 disp(angdef2, 'each pole must contribute an angle of '
      );
19
20 a = 5 + 5*cotd(angdef2)
21 Gcx = (s + a)^2 / s;
22 K = 1/ abs(horner(Gcx*Gp, dp) )
23 Gc1 = K * (s + a)^2 / s
24
25 // determining Kp, Ti and Td
26 cf = coeff(numer(Gc1));
27 Kp = cf(2)
28 Ti = Kp / cf(1)
29 Td = cf(3) / Kp
30
31 t = 0:0.01:4;
32 u = ones(1,length(t));
33 subplot(2,1,1);
34 YbyD = syslin('c',Gp / (1 + Gp * Gc1))
35 plotresp(u,t,YbyD,'Response to step disturbance
      input');
36 ax = gca();
37 ax.data_bounds = [0 0; 3 2];
38
39 //Step 2: Design of Gc
40 Gc = (YbyD.den - s^3) / 100 / s
41
42 YbyR = syslin('c',1 - s^3 / YbyD.den )
43 subplot(2,1,2);
44 t = 0:0.01:3;
45 u = ones(1,length(t));
46 plotresp(u,t,YbyR,'Response to step reference input'
      );
47 scf();

```

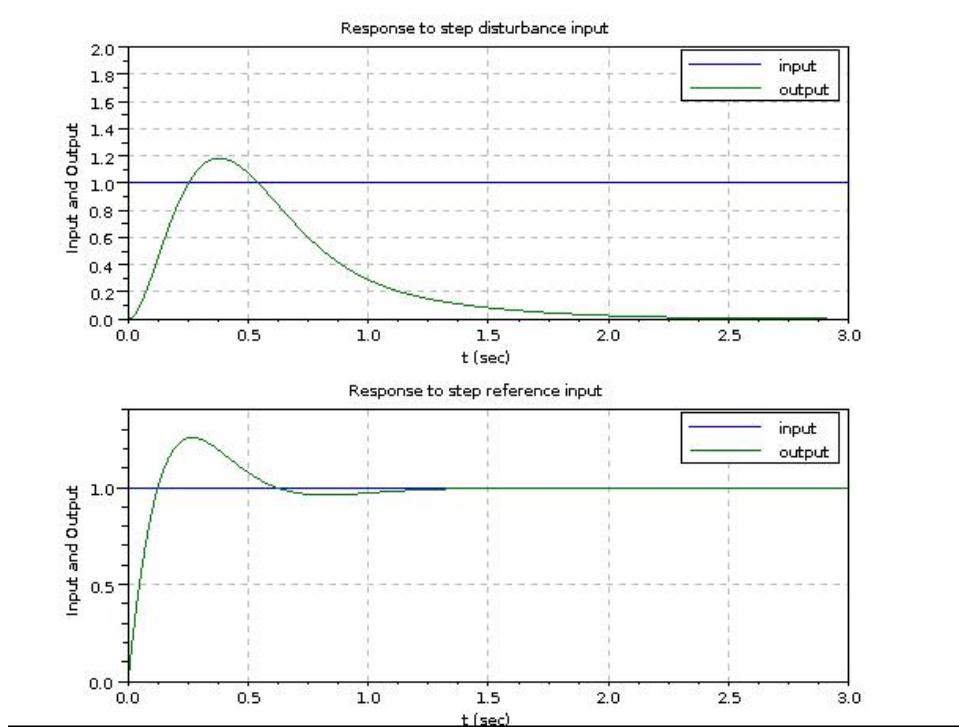


Figure 8.10: Design of system with two degrees of freedom

```

48 subplot(2,1,1);
49 plotresp(t,t,YbyR,'Response to ramp reference input',
    );
50 subplot(2,1,2);
51 t = 0:0.01:2;
52 u = 1/2 * t.^2;
53 plotresp(u,t,YbyR,'Response to acceleration
    reference input');

```

check Appendix AP 2 for dependency:

`plotresp.sci`

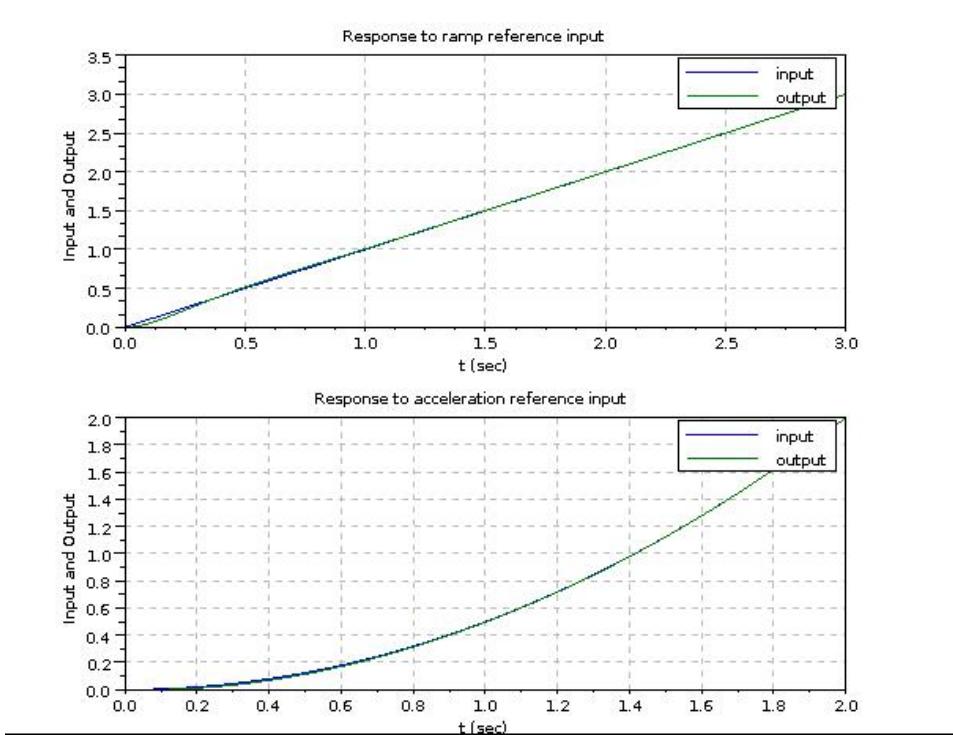


Figure 8.11: Design of system with two degrees of freedom

check Appendix AP 2 for dependency:

`plotresp.sci`

check Appendix AP 7 for dependency:

`rootl.sci`

Scilab code Exa 8.1 Tuning a PID controller using Nichols Second Rule

```
1 // Example 8-1
2 // Tuning a PID controller using Nichols Second Rule
3 clear; clc;
4 xdel(winsid()); // close all windows
5 mode(0);
6
7 // please edit the path
8 // cd <your code directory>
9 // exec("plotresp.sci");
10 // exec("rootl.sci");
11
12 s = %s;
13 G = 1 / ( s * (s + 1) * (s + 5) )
14
15 // finding Kcr and wcr (omega cr)
16 w = poly(0, 'w');
17 D = horner(denom(G), %i * w);
18 x = roots(imag(D));
19 wcr = abs(x(2)) // the non zero root
20 Kcr = -1*clean(horner(D, wcr))
21 Pcr = 2*pi / wcr
22
23 Kp = 0.6 * Kcr
24 Ti = 0.5*Pcr
25 Td = 0.125*Pcr
26 Gc = Kp * ( s + 1/Ti + s^2*Td ) / s
```

```

27 GGc = syslin('c',G*Gc);
28 H = syslin('c',GGc /. 1);
29 disp(H,'closed loop system =');
30
31 rootl(GGc,0,'Root locus of open loop system');
32 sgrid([0.3],[]);
33 a = gca(); a.data_bounds = [-7 -4; 2 4];
34 xstring(-1,1,'zeta = 0.3');
35
36 scf();
37 t = 0:0.1:14;
38 u = ones(1,length(t));
39 plotresp(u,t,H,'');
40 // unacceptably large maximum overshoot
41
42 // new system
43 Kp2 = 39.42
44 Ti2 = 3.077
45 Td2 = 0.7692
46 Gc2 = Kp2 * ( s + 1/Ti2 + s^2*Td2 ) / s
47 GGc2 = syslin('c',G*Gc2);
48 H2 = syslin('c',GGc2 /. 1);
49 disp(H2,'closed loop system2 =');
50 plotresp(u,t,H2,'Step Response to a PID controlled
    system');
51 xstring(1.5,1.65,'System 1');
52 xstring(0.5,1.3,'System 2');

```

check Appendix AP 2 for dependency:

`plotresp.sci`

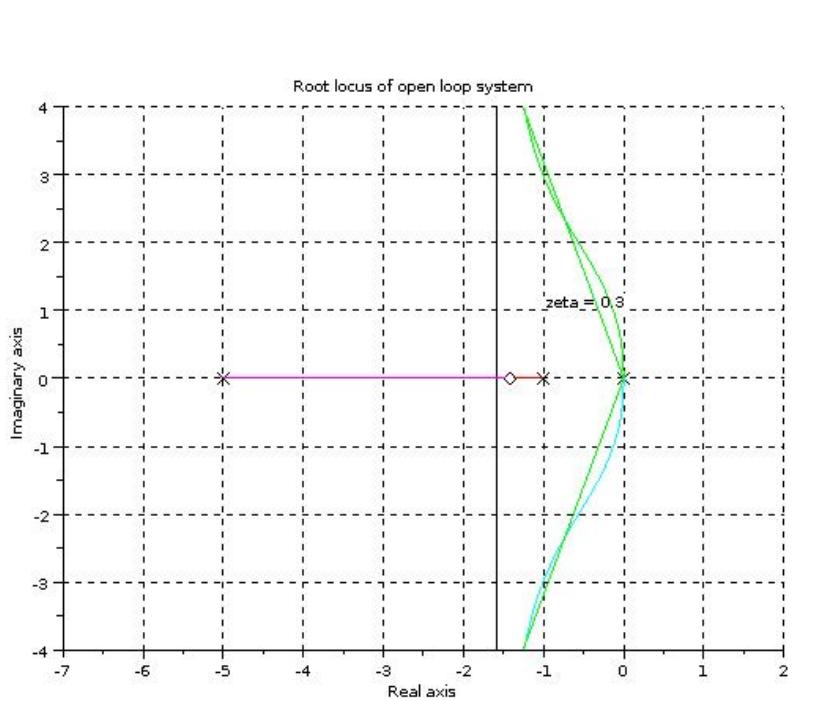


Figure 8.12: Tuning a PID controller using Nichols Second Rule

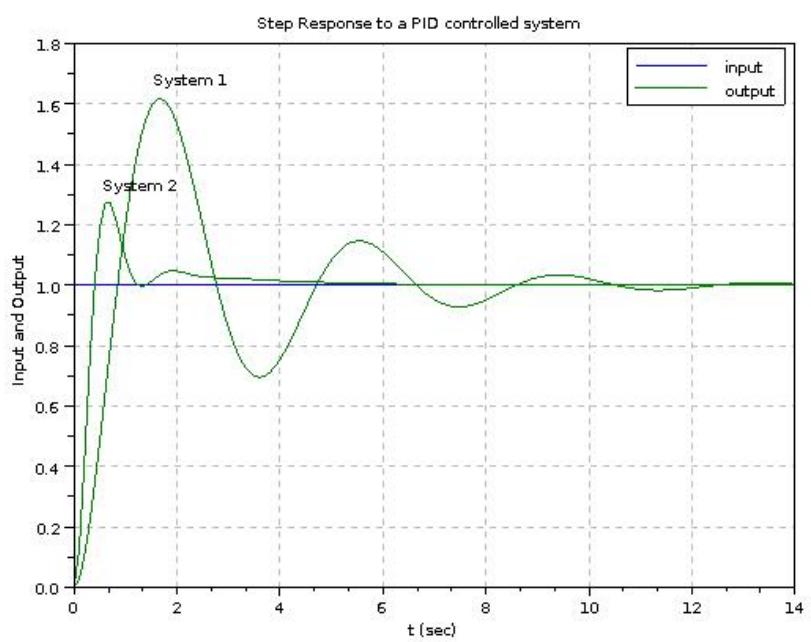


Figure 8.13: Tuning a PID controller using Nichols Second Rule

Scilab code Exa 8.2 Computation of Optimal solution 1

```
1 // Example 8-2
2 // Computation of Optimal solution 1
3
4 clear; clc;
5 xdel(winsid()); //close all windows
6
7 // please edit the path
8 // cd "";
9 // exec("plotresp.sci");
10
11 s = %s;
12 G = 1.2 / ( 0.36*s^3+ 1.86*s^2 + 2.5*s + 1);
13
14 K = 2.0 : 0.2 : 3.0;
15 a = 0.5 : 0.2 : 1.5;
16
17 t = 0:0.1:5; u = ones(1,length(t));
18 // lesser points for a rough check
19 t1 = 0:0.01:5; u1 = ones(1,length(t1));
20 // more points for a rigorous check
21
22 k = 0;
23 for i = 1:6
24   for j = 1:6
25     Gc = K(i) * (s + a(j))^2 / s;
26     H = G * Gc;
27     H = syslin('c', H /. 1);
28     y = csim(u,t,H);
29     m = max(y);
30     if m < 1.1 then
31       y = csim(u1,t1,H);
32       m = max(y);
33       if m < 1.1 then
34         k = k + 1;
35         solution(k,: ) = [K(i) a(j) m];
36     end
```

```

37     end
38 end
39 end
40 disp(solution,'solution [K a m] = ');
41 // to sort the matrix
42 [x 0] = gsort(solution(:,3), 'r', 'i');
43 // re order the matrix
44 for i = 1:k
45 sortsolution(i,:) = solution( 0(i) , :);
46 end
47 disp(sortsolution,'sortsolution [K a m] = ');
48
49 // Response with largest overshoot above 10%
50 x = sortsolution(k,:);
51 K = x(1); a = x(2);
52 Gc = K * (s + a)^2 / s;
53 H = G * Gc;
54 H = syslin('c', H /. 1);
55 plotresp(u1,t1,H,'Step Response with 10% overshoot')
;
56 disp(Gc,'Gc = ');
57 disp(H,'H = ');

```

check Appendix AP 2 for dependency:

`plotresp.sci`

Scilab code Exa 8.3 Computation of Optimal solution 2

```

1 // Example 8-3
2 // Computation of Optimal solution 2
3
4 clear; clc;
5 xdel(winsid()); // close all windows

```

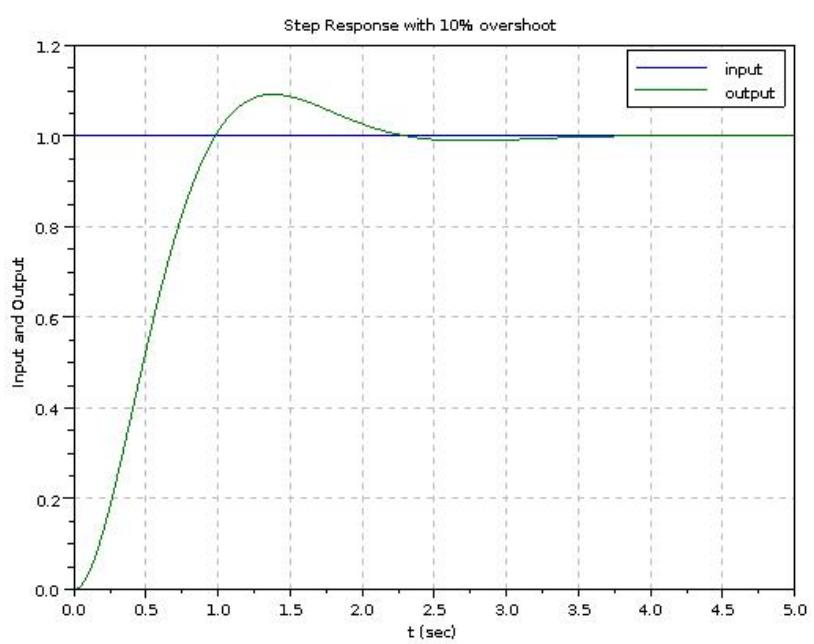


Figure 8.14: Computation of Optimal solution 1

```

6
7 // please edit the path
8 // cd "";
9 // exec("plotresp.sci");
10
11 s = %s;
12 G = 4 / ( s^3 + 6*s^2 + 8*s + 4 );
13
14 t = 0:0.1:8; u = ones(1,length(t));
15 // lesser points for a rough check
16 t1 = 0:0.01:8; u1 = ones(1,length(t1));
17 // more points for a rigorous check
18
19 k = 0;
20 mprintf('Processing ... \n');
21
22 for K = 3:0.2:6
23   for a = 0.1:0.1:3
24     Gc = K * (s + a)^2 / s;
25     H = G * Gc;
26     H = syslin('c', H /. 1);
27     y = csim(u,t,H);
28     m = max(y);
29     if m < 1.15 & m > 1.08 then
30       // give a margin of 0.02 for the rough check
31       - 1.08
32       y = csim(u1,t1,H);
33       m = max(y);
34       if m < 1.15 & m > 1.10 then
35         // check for settling time
36         l = length(t1);
37         while y(l) > 0.98 & y(l) < 1.02 ; l = l-1;
38         end
39         ts = (l-1) * 0.01;
40         if ts < 3.00 then
41           k = k + 1;
42           solution(k,:) = [K a m ts];
43         end

```

```

42      end
43    end
44
45  end
46 if modulo(K*10,2) == 0 then mprintf(' completed
%d%%\n', (K - 3)/3*100)
47 end
48 end
49
50 disp(solution,'solution [K a m ts] = ');
51
52 [x_0] = gsort(solution(:,3), 'r', 'i');
53 for i = 1:k
54   sortsolution(i,:) = solution(0(i), :);
55 end
56 disp(sortsolution,'sortsolution [K a m ts] = ');
57
58 // Response with smallest overshoot
59 x = sortsolution(1,:);
60 K = x(1); a = x(2);
61 Gc = K * (s + a)^2 / s;
62 H = G * Gc;
63 H = syslin('c', H /. 1);
64 plotresp(u,t,H,'Step Response with smallest
overshoot');
65 disp(Gc,'Gc = ');
66 disp(H,'H = ');

```

check Appendix [AP 2](#) for dependency:

`plotresp.sci`

Scilab code Exa 8.4 Design of system with two degrees of freedom

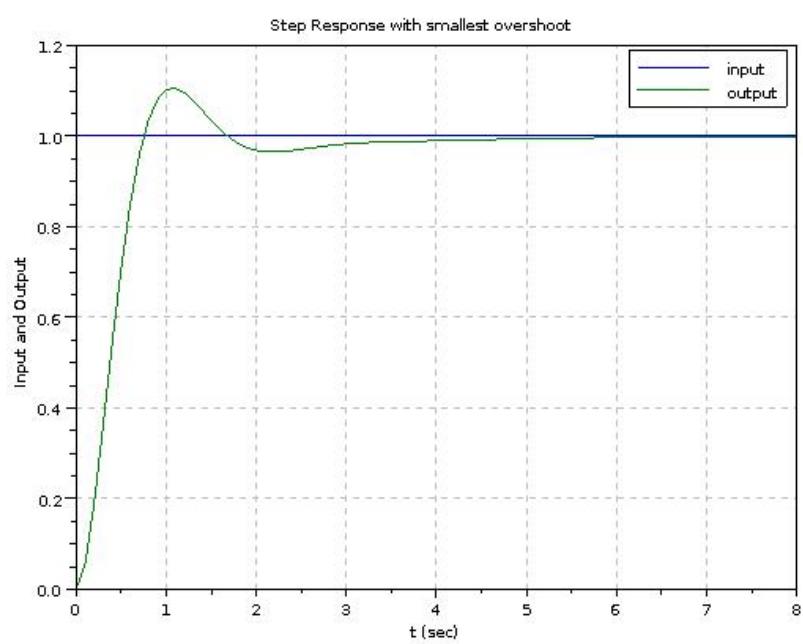


Figure 8.15: Computation of Optimal solution 2

```

1 // Example 8-4
2 // Design of system with two degrees of freedom
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6 mode(0);
7 // please edit the path
8 // cd "";
9 // exec("plotresp.sci");
10
11 s = %s;
12 // Design Step 1: choosing a, b and c.
13
14 t = 0:0.1:4;
15 u = ones(1,length(t));
16
17 t1 = 0:0.01:4;
18 N = length(t1);
19 u1 = ones(1,N);
20 // N = N - 3
21
22 k = 0;
23 mprintf('Processing ... \n');
24
25 for i = 1:21
26     a = 6.2 - 0.2*i;
27     for j = 1:21
28         b = 6.2 - 0.2*j;
29         for h = 1:21
30             c = 12.2 - 0.2*h;
31             num = (2*a + c)*s^2 + (a*a + b*b + 2*a*c)*s +
32                     (a*a + b*b)*c;
33             den = s^3 + num;
34             G = syslin('c',num,den);
35             y = csim(u,t,G);
36             m = max(y);
37             if m < 1.19 & m > 1.00 then
38                 y = csim(u1,t1,G);

```

```

38     m = max(y);
39     if m < 1.19 & m > 1.02 then
40         l = N;
41         while y(l) > 0.98 & y(l) < 1.02 ; l = l-1;
42             end
43             ts = (l-1) * 0.01;
44             if ts < 1.0 then
45                 k = k + 1;
46                 solution(k,:) = [a b c m ts];
47                 end
48             end
49
50         end
51     end
52     mprintf( ' completed %d%%\n' , (6 - a)/4*100);
53 end
54
55 disp(solution,'solution = ');
56
57 K = solution(1,:);
58 a = K(1); b = K(2); c = K(3);
59 num = (2*a + c)*s^2 + (a*a + b*b + 2*a*c)*s + (a*a +
    b*b)*c;
60 den = s^3 + num;
61 YbyR = syslin('c',num,den); disp(YbyR,'Y(s)/R(s) = ');
62 subplot(2,1,1);
63 plotresp(u1,t1,YbyR,'Step response for a = 4.2 ,b =
    2 ,c =12');
64
65 cf = coeff(den);
66 K = (cf(3) - 1) / 10
67 alpha_plus_beta = cf(2) / K /10
68 alphabeta = cf(1) / K / 10
69 Gc = K * (s^2 + alpha_plus_beta*s + alphabeta) / s
70 YbyD = syslin('c',10*s,den);
71 disp(YbyD,'Y(s)/D(s) = ');
72 subplot(2,1,2);

```

```

73 plotresp(u1,t1,YbyD,'Response to step disturbance
    input for a = 4.2 ,b = 2 ,c =12');
74 a = gca(); a.data_bounds = [0 -0.01; 4 0.07];
75
76 // Design Step 2
77 scf();
78 Gc1 = (YbyR.num / 10) / s
79 Gc2 = Gc - Gc1
80
81 // response to reference inputs
82 y1 = csim(t,t,YbyR); u = 1/2 * t.^2;
83 y2 = csim(u,t,YbyR);
84
85 subplot(2,1,1);
86 plotresp(t,t,YbyR,'Response to unit ramp input');
87 subplot(2,1,2);
88 plotresp(u,t,YbyR,'Response to unit acceleration
    input');

```

check Appendix AP 2 for dependency:

`plotresp.sci`

Scilab code Exa 8.5 Design of system with two degrees of freedom 2

```

1 // Example 8-5
2 // Design of system with two degrees of freedom 2
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6 mode(0);
7 // please edit the path

```

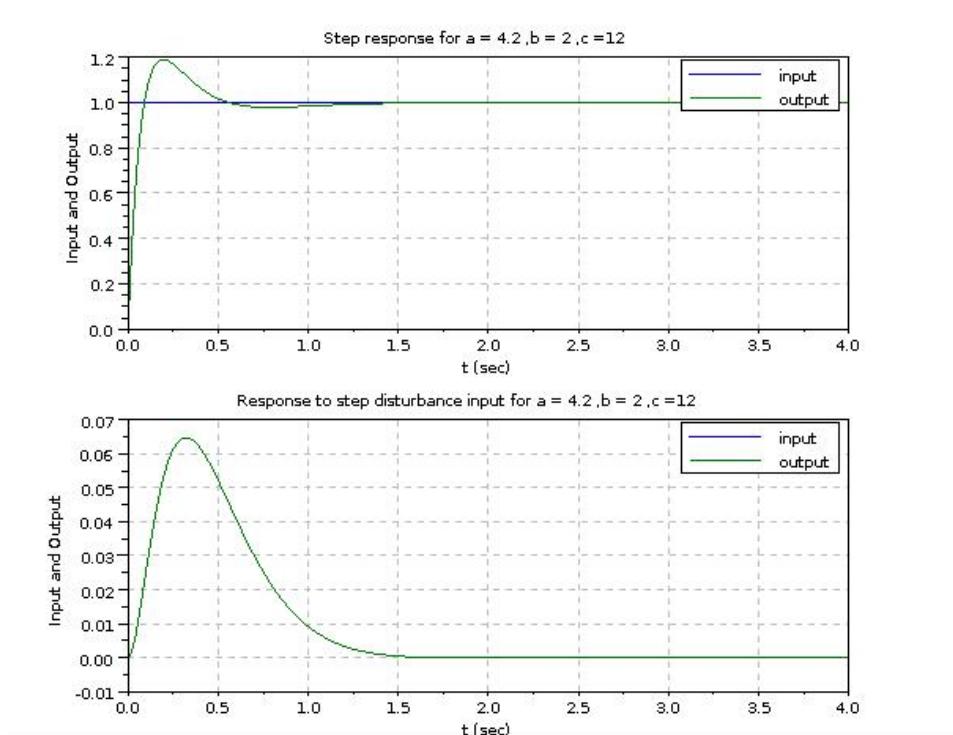


Figure 8.16: Design of system with two degrees of freedom

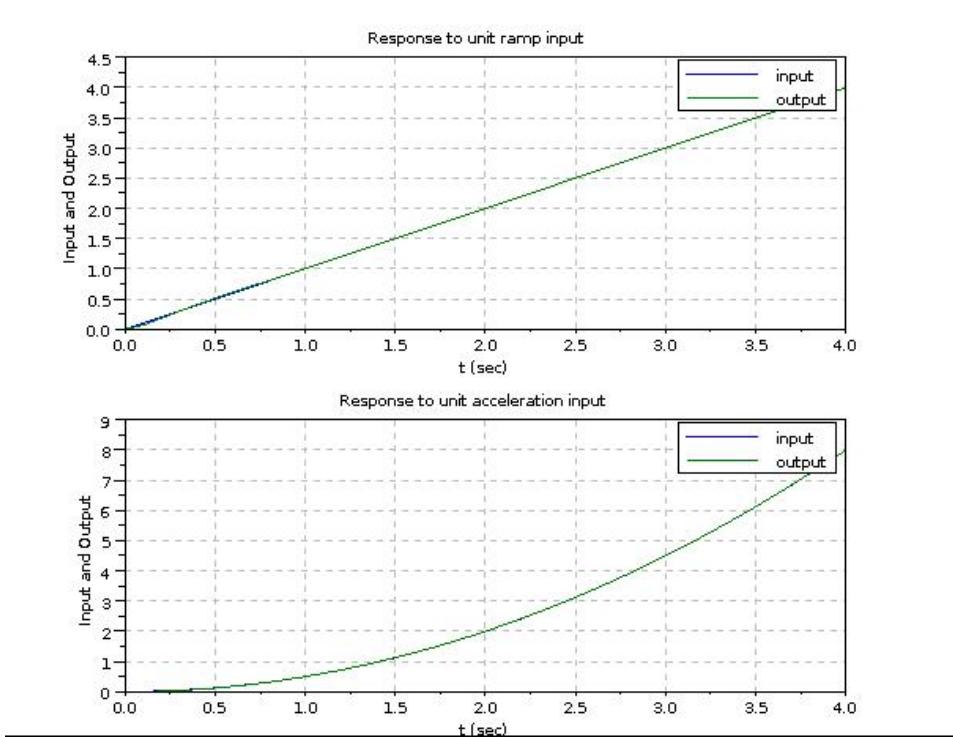


Figure 8.17: Design of system with two degrees of freedom

```

8 // cd "";
9 // exec("plotresp.sci");
10
11 s = %s;
12 Gp = 5 / (s+1) / (s+5)
13 t = 0:0.01:3;
14 u = ones(1,length(t));
15
16 // Step 1: Design of Gc1
17 a = sqrt(13)
18 K = 4 / (5*a - 15)
19 Gc1 = K * (s + a)^2 / s
20
21 // determining Kp, Ti and Td
22 cf = coeff(numer(Gc1));
23 Kp = cf(2)
24 Ti = Kp / cf(1)
25 Td = cf(3) / Kp
26
27 subplot(2,1,1);
28 YbyD = syslin('c',Gp / (1 + Gp * Gc1))
29 plotresp(u,t,YbyD,'Response to step disturbance
    input');
30 a = gca();
31 a.data_bounds = [0 0; 3 0.1];
32
33 //Step 2: Design of Gc2
34 cf = coeff(YbyD.den);
35 Kp2 = (cf(2) - 47.63) / 5
36 Td2 = (cf(3) - 6.6051) / 5 / Kp2
37
38 Gc2 = Kp2 * (1 + Td2*s)
39
40 YbyR = syslin('c',1 - s^3 / YbyD.den )
41 subplot(2,1,2);
42 t = 0:0.05:2;
43 u = ones(1,length(t));

```

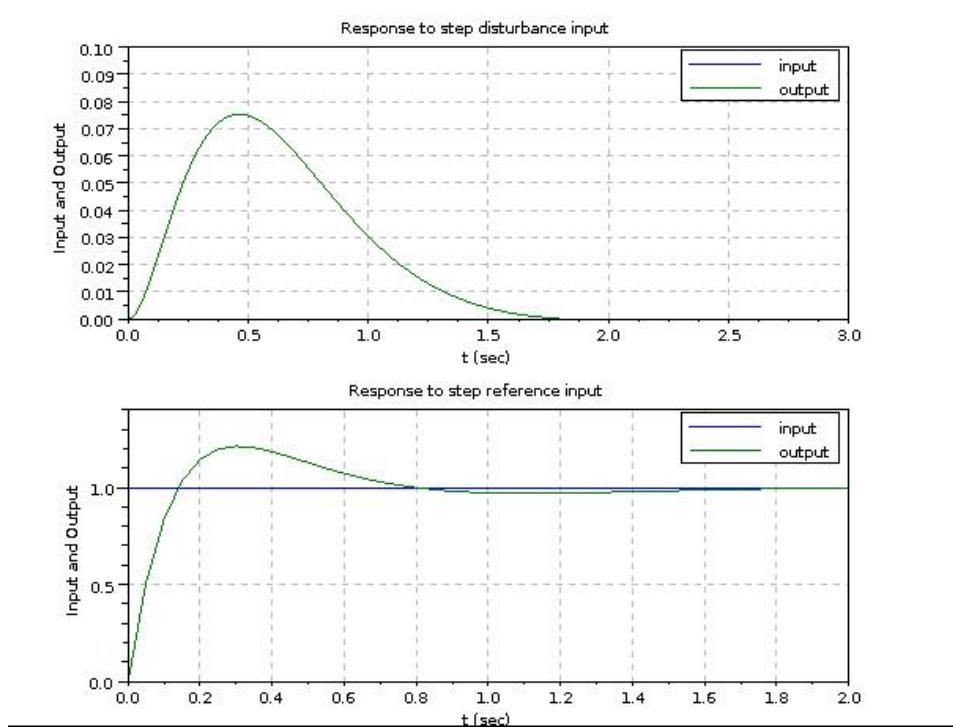


Figure 8.18: Design of system with two degrees of freedom 2

```

44 plotresp(u,t,YbyR,'Response to step reference input');
45 scf();
46 subplot(2,1,1);
47 plotresp(t,t,YbyR,'Response to ramp reference input');
48 subplot(2,1,2);
49 u = 1/2 * t.^2;
50 plotresp(u,t,YbyR,'Response to acceleration
reference input');

```

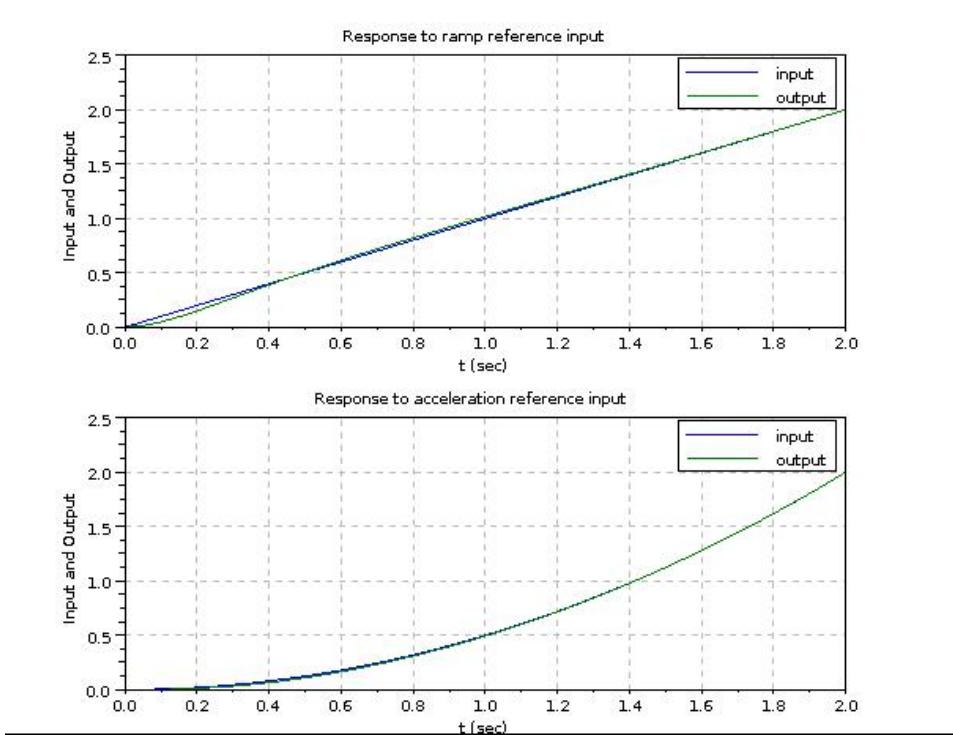


Figure 8.19: Design of system with two degrees of freedom 2

Chapter 9

Control Systems Analysis in State Space

Scilab code Exa 9.b.3 Obtaining canonical form

```
1 // Exercise B-9-3
2 // Obtaining canonical form
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 // please edit the path
8 // cd "<path to dependencies>";
9 // exec("transferf.sci");
10
11 A = [1 2; -4 -3];
12 B = [1; 2];
13 C = [1 1];
14 D = 0;
15
16 [Ac Bc U ind] = canon(A,B);
17 U = -1*U; // a correction
18 Cc = C*U;
19 disp(clean(Ac), 'Ac = ');
```

```

20 disp(clean(Bc), 'Bc = ');
21 disp(clean(Cc), 'Cc = ');
22 disp(U, 'transformation matrix U = ');
23 // Ac=inv(U)*A*U, Bc=inv(U)*B
24
25 // check
26 Htf1 = transferf(A, B, C, D);
27 Htf2 = transferf(Ac, Bc, Cc, D);
28 disp(Htf1, 'Htf1 = ');
29 disp(Htf2, 'Htf2 = ');

```

check Appendix AP 4 for dependency:

`transferf.sci`

Scilab code Exa 9.a.5 Conversion from transfer function model to state space model

```

1 // Example A-9-5
2 // Conversion from transfer function model to state
   space model
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 s = %s;
8 num = 25.04*s + 5.008;
9 den = poly([5.008 25.1026 5.03247 1], 's', 'c');
10
11 Hss = cont_frm(num, den);
12 disp(Hss, 'Hss = ');

```

Scilab code Exa 9.a.16 Controllability and pole zero cancellation

```

1 // Example A-9-16
2 // Controllability and pole zero cancellation
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 // please edit the path
8 // cd "<path to dependencies>";
9 // exec(" transferf.sci");
10
11
12 A = [-3 1; -2 1.5];
13 B = [1; 4];
14 C = [1 0];
15 D = 0;
16 Cc = cont_mat(A,B); disp(Cc, 'state controllability
matrix =');
17 disp(det(Cc), 'det(Cc) = ');
18
19 Htf = transferf(A,B,C,D); disp(Htf, 'Reduced transfer
function =');
20 e = spec(A); disp(e, 'Eigen values = ');
21 D = poly(e,'s'); disp(D, 'actual denominator (
characteristic poly) =');

```

check Appendix AP 4 for dependency:

`transferf.sci`

Scilab code Exa 9.a.17 Controllability observability and pole zero cancellation

```

1 // Example A-9-17
2 // Controllability observability and pole zero
cancellation
3

```

```

4 clear; clc;
5 xdel(winsid()); // close all windows
6
7
8 A = [0 1; -0.4 -1.3];
9 B = [0; 1];
10 C = [0.8 1];
11 D = [0];
12 G1 = syslin('c',A,B,C,D); ssprint(G1);
13
14 G2 = syslin('c',A', C',B',D); ssprint(G2);
15
16 Cc1 = cont_mat(A,B); disp(Cc1, 'state controllability
matrix 1 =');
17 disp(det(Cc1), 'det(Cc1) = ');
18 Ob1 = obsv_mat(A,C); disp(Ob1, 'observability matrix
1 =');
19 disp(det(Ob1), 'det(Ob1)');
20
21 Cc2 = cont_mat(A',C'); disp(Cc2, 'state
controllability matrix 2 =');
22 disp(det(Cc2), 'det(Cc2) = ');
23 Ob2 = obsv_mat(A',B'); disp(Ob2, 'observability
matrix 2 =');
24 disp(det(Ob2), 'det(Ob2)');
25
26 Htf = ss2tf(G1); disp(Htf, 'Reduced transfer function
= ');
27 e = spec(A); disp(e, 'Eigen values = ');
28 D = poly(e,'s'); disp(D, 'actual denominator (
characteristic poly) = ');

```

check Appendix AP 6 for dependency:

`pf_residu.sci`

Scilab code Exa 9.1 Transfer function to controllable observable and jordan canonical forms

```
1 // Example 9-1
2 // Transfer function to controllable , observable and
   jordon canonical forms
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 // please edit the path
8 // cd "<path for the dependencies>";
9 // exec(" pf_residu.sci");
10
11 s = %s;
12 N = s + 3;
13 D = s^2 + 3*s + 2;
14
15 Hc = cont_frm(N,D);
16 disp('controllable form ='); ssprint(Hc);
17
18 Ho =syslin('c', (Hc.A)', (Hc.C)', (Hc.B)', Hc.D);
19 disp('observable form ='); ssprint(Ho);
20
21 A = diag(roots(D));
22 B = [1;1];
23 C = pf_residu(N,D)';
24 D = Hc.D; // in this case : b0 = 0
25 Hj = syslin('c',A,B,C,D);
26 disp('jordon canonical form ='); ssprint(Hj);
27
28 // This example will work for any proper transfer
   function
29 // with all distinct poles or eigen values
```

Scilab code Exa 9.2 Transformations in state space

```
1 // Example 9-2
2 // Transformations in state space
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6 mode(0);
7
8 A = [0 1 0; 0 0 1; -6 -11 -6];
9 B = [0; 0; 0];
10 C = [1 0 0];
11 D = [0];
12 H = syslin('c',A,B,C,D);
13 disp('non standard form ='); ssprint(H);
14
15 e = spec(A)', // eigen values
16 P = [ones(1,3); e; e.^2] // P is the transformation
   matrix
17 A1 = diag(e);
18 B1 = inv(P)* B;
19 C1 = C * P;
20 D1 = D;
21 H1 = syslin('c',A1,B1,C1,D1);
22 disp('standard form ='); ssprint(H1);
```

check Appendix AP 4 for dependency:

`transferf.sci`

Scilab code Exa 9.3 Conversion from state space to transfer function model

```
1 // Example 9-3
2 // Conversion from state space to transfer function
   model
3
```

```

4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 // please edit the path
8 // cd "<path to your dependencies>";
9 // exec("transferf.sci");
10
11 A = [0 1 0; 0 0 1; -5.008 -25.1026 -5.03247];
12 B = [0; 25.04; -121.005];
13 C = [1 0 0];
14 D = [0];
15
16 H = transferf(A,B,C,D);
17 disp(H, 'H =');

```

check Appendix [AP 4](#) for dependency:

`transferf.sci`

Scilab code Exa 9.4 Conversion from state space to transfer function model

```

1 // Example 9-4
2 // Conversion from state space to transfer function
model
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 // please edit the path
8 // cd "<path to your dependencies>";
9 // exec("transferf.sci");
10
11 A = [0 1; -25 -4];
12 B = [1 1; 0 1];
13 C = [1 0; 0 1];
14 D = [0 0; 0 0];

```

```

15
16 H = transferf(A,B,C,D);
17 disp(H, 'H =');
18
19 // Htf is the tranfer function matrix with four
   transfer functions
20 // Htf(y1,u1),Htf(y1,u2)
21 // Htf(y2,u1),Htf(y2,u2)

```

check Appendix AP 5 for dependency:

`ilaplace.sci`

check Appendix AP 6 for dependency:

`pf_residu.sci`

Scilab code Exa 9.5 State transition matrix

```

1 // Example 9-5
2 // State transition matrix
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 // please edit the path
8 // cd "<path for the dependencies>";
9 // exec(" pf_residu.sci");
10 // exec(" ilaplace.sci");
11
12 s = %s;
13 A = [0 1; -2 -3];
14 L = inv(s*eye(2,2) - A);
15 disp(L, 'inv(sI - A) =');
16
17 // Find the Inverse Laplace transform
18 for i = 1:2

```

```

19   for j = 1:2
20     phi(i,j) = ilaplace(L(i,j));
21   end;
22 end;
23
24 disp(phi, 'state transition matrix =');
25 // ilaplace may not work for systems with repeated
   poles

```

check Appendix AP 5 for dependency:

`ilaplace.sci`

check Appendix AP 6 for dependency:

`pf_residu.sci`

Scilab code Exa 9.7 Finding e to the power At using laplace transforms

```

1 // Example 9-7
2 // Finding e to the power At using laplace
   transforms
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 // please edit the path
8 // cd "<path for the dependencies>";
9 // exec("pf_residu.sci");
10 // exec("ilaplace.sci");
11
12 s = %s;
13 A = [0 1; 0 -2];
14 L = inv(s*eye(2,2) - A);
15 disp(L, 'inv(sI - A) =');
16
17 // Find the Inverse Laplace transform

```

```

18 for i = 1:2
19   for j = 1:2
20     phi(i,j) = ilaplace(L(i,j));
21   end;
22 end;
23 disp(phi, 'e^At =');

```

Scilab code Exa 9.9 Linear dependence of vectors

```

1 // Example 9-9
2 // Linear dependence of vectors
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6 mode(0)
7
8 x1 = [1; 2; 3]
9 x2 = [1; 0; 1]
10 x3 = [2; 2; 4]
11 A = [x1 x2 x3];
12 disp(A, '[x1:x2:x3] =');
13 disp(clean(det(A)), 'det([x1:x2:x3]) ='); // singular
14
15 x3 = [2;2;2]
16 A = [x1 x2 x3];
17 disp(A, '[x1:x2:x3] =');
18 disp(det(A), 'det([x1:x2:x3]) ='); // non singular

```

Scilab code Exa 9.14 State and ouput controllability and observability

```

1 // Example 9-14
2 // State and ouput controllability and observability
3

```

```

4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 A = [1 1; -2 -1];
8 B = [0; 1];
9 C = [1 0];
10 D = [0];
11 G = syslin('c', A, B, C, D); ssprint(G);
12
13 Cc = cont_mat(A, B); disp(Cc, 'state controllability
    matrix =');
14 c = [C*B C*A*B]; disp(c, 'output controllability
    matrix =');
15 Ob = obsv_mat(A, C); disp(Ob, 'observability matrix ='
    );

```

Scilab code Exa 9.15 Observability

```

1 // Example 9-15
2 // Observability
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 A = [0 1 0; 0 0 1; -6 -11 -6];
8 B = [0; 0; 1];
9 C = [4 5 1];
10
11 Ob = obsv_mat(A, C);
12 disp(Ob, 'observability matrix =');
13 disp(clean(det(Ob)), 'det(Ob) =');
14 // system is not completely observable

```

Chapter 10

Control Systems Design in State Space

Scilab code Exa 10.i.1 Designing a regulator using a minimum order observer

```
1 // Illustration 10.1
2 // Designing a regulator using a minimum order
   observer
3
4 // Section 10-6 of the book
5
6 clear; clc;
7 xdel(winsid());    // close all windows
8 mode(0);
9
10 // please edit the path
11 // cd "<path to dependencies>";
12 // exec("minorder.sci");
13
```

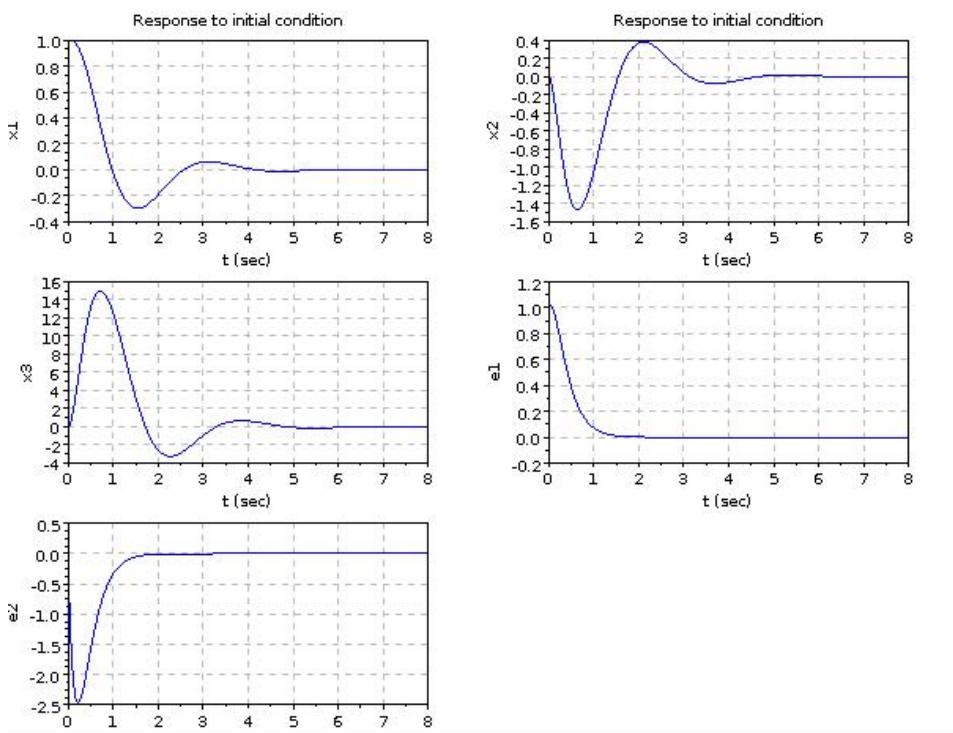


Figure 10.1: Designing a regulator using a minimum order observer

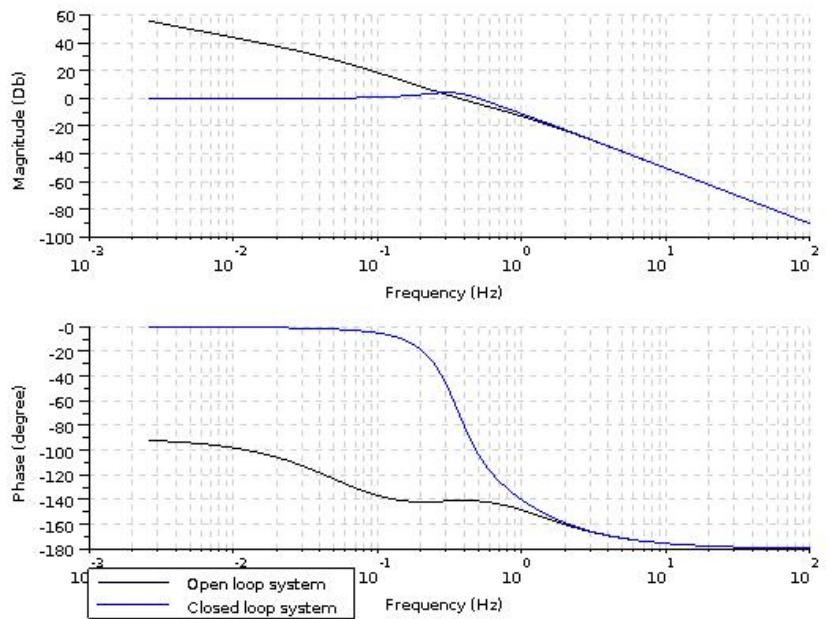


Figure 10.2: Designing a regulator using a minimum order observer

```

14 function smallplot(i)
15   subplot(3,2,i);xgrid(color('gray'));
16   plot(t,x(i,:));
17 endfunction
18
19
20 A = [0 1 0; 0 0 1; 0 -24 -10];
21 B = [0; 10; -80];
22 C = [1 0 0];
23 D = [0];
24 Gp = syslin('c',A,B,C,D);
25
26 // Trial 1
27 disp('trial 1')
28 P = [-1 + %i*2, -1 - %i*2, -5 ]
29 Q = [-10 -10] // observer poles
30
31 // Determining gains K and Ke
32 // Determining observer controller transfer function
33 [K Ke Go ch] = minorder(A,B,P,Q);
34 K
35 Ke
36 disp(Go,'observer controller transfer function =');
37 disp(ch,'overall system characteristic equation =');
38 disp(roots(Go.den),'observer controller has unstable
root! ');
39
40 disp('trial 2'); // Trial 2;
41 P
42 Q = [-4.5 -4.5]; // change Q
43 [K Ke Go ch AA] = minorder(A,B,P,Q);
44 K
45 Ke
46 disp(Go,'observer controller transfer function =');
47 disp(ch,'overall system characteristic equation =');
48 disp(roots(Go.den),'observer controller has all
stable roots! ');
49

```

```

50 // system response to initial conditions
51 x0 = [1; 0; 0; 1; 0];
52 G = syslin('c',AA,[1 ;0 ;0 ;0],[1 0 0 0 0],[0],x0
    );
53
54 t = 0:0.01:8;
55 u = zeros(1,length(t));
56 [y x] = csim(u,t,G);
57
58 smallplot(1);
59 xtitle('Response to initial condition','t (sec)','x1
    ');
60 smallplot(2);
61 xtitle('Response to initial condition','t (sec)','x2
    ');
62 smallplot(3);
63 xtitle('','t (sec)','x3');
64 smallplot(4);
65 xtitle('','t (sec)','e1');
66 smallplot(5);
67 xtitle('','t (sec)','e2');
68
69 scf();
70 // Bode diagram
71 O = Go*Gp; C = O /. 1;
72 bode([O;C],0.001,100,['Open loop system'; 'Closed
    loop system']);
73 disp(p_margin(O),'Phase margin');

```

check Appendix AP 1 for dependency:

`minorder.sci`

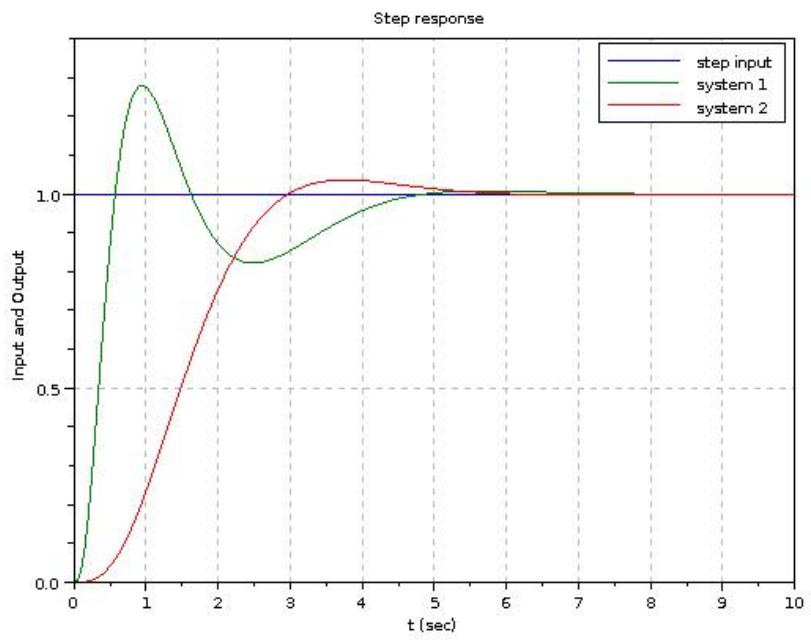


Figure 10.3: Designing a control system with a minimum order observer

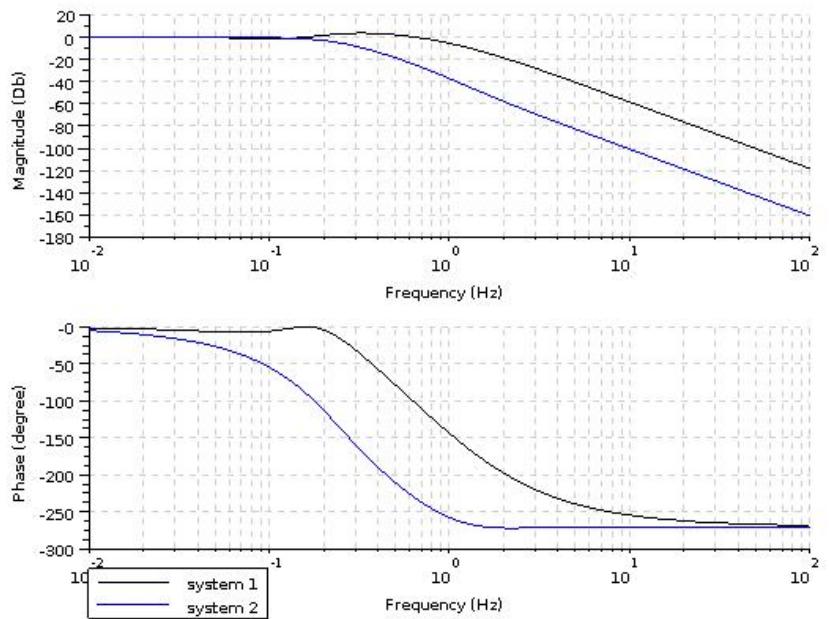


Figure 10.4: Designing a control system with a minimum order observer

Scilab code Exa 10.i.2 Designing a control system with a minimum order observer

```
1 // Illustration 10.2
2 // Designing a control system with a minimum order
   observer
3
4 // Section 10-7 of the book
5
6 clear; clc;
7 xdel(winsid()); // close all windows
8 mode(0);
9
10 // please edit the path
11 // cd "<path to dependencies>";
12 // exec("minorder.sci");
13 // exec("plotresp.sci");
14
15 s = %s;
16 t = 0:0.05:10;
17 u = ones(1,length(t));
18 Gp = syslin('c',1,s*(s^2 + 1));
19 Gs = cont_frm(1,s*(s^2 + 1));
20 A = Gs.A;
21 B = Gs.B;
22 C = Gs.C;
23 D = Gs.D;
24
25 // designing the observer controller
26 P = [-1 + %i, -1 - %i, -8]
27 Q = [-4 -4] // observer poles
28 [K Ke Go] = minorder(A,B,P,Q);
29 K
30 Ke
31 disp(Go, 'observer controller transfer function =');
32
33 // First configuration
34 C1 = Go*Gp /. 1;
```

```

35 disp(C1,'closed loop system of first configuration =
');
36 plotresp(u,t,C1,'Step response');
37
38 // Secoond Configuration
39 C = Gp /. Go;
40 N = 1 / horner(C,0)
41 C2 = syslin('c',N*C);
42 y = csim(u,t,C2);
43 disp(C2,'closed loop system of second configuration
=');
44 plot(t,y,'r');
45 legend('step input','system 1','system 2');
46
47 // Bode diagram
48 scf();
49 bode([C1;C2],0.01,100,['system 1';'system 2']);
50 // frequency in Hz

```

check Appendix AP 1 for dependency:

`minorder.sci`

check Appendix AP 2 for dependency:

`plotresp.sci`

Scilab code Exa 10.a.5 Feedback gain for moving eigen values

```

1 // Example A-10-5
2 // Feedback gain for moving eigen values
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6 mode(0);
7
8 s = %s;

```

```

9 A = [0 1;-2 -3];
10 B = [0; 2];
11 C = [1 0];
12 E = [-3 -5]; // new eigen values
13
14 ch = det(s*eye(2,2) - A)
15 cf = coeff(ch);
16 a = cf(1: $-1)
17
18 chd = poly(E, 's');
19 cf2 = coeff(chd);
20 alpha = cf2(1: $-1)
21
22 M = cont_mat(A,B)
23 W = [cf(2:$) ; 1 0]
24 T = M*W
25
26 Ti = inv(T); disp(Ti, 'inv(T)');
27 K = (alpha - a) * Ti

```

Scilab code Exa 10.a.6 Gain matrix determination

```

1 // Example A-10-6
2 // Gain matrix determination
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7
8 A = [0 1 0; 0 0 1;-6 -11 -6];
9 B = [0; 0; 10];
10
11 P = [-2 + %i*2*sqrt(3) , -2 - %i*2*sqrt(3), -10];
12 K = ppol(A,B,P); disp(K, 'K = ');

```

Scilab code Exa 10.a.9 Transforming to canonical form

```
1 // Example A-10-9
2 // Transforming to canonical form
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6 mode(0);
7
8 s = %s;
9 A = [1 1; -4 -3];
10 B = [0; 2];
11 C = [1 1];
12
13 ch = det(s*eye(2,2) - A)
14 cf = coeff(ch);
15 a = cf(1: $-1)
16
17
18 N = obsv_mat(A, C)'
19 W = [cf(2:$) ; 1 0]
20 Qi = W*N'
21 Q = inv(Qi)
22
23 A1 = Qi*A*Q
24 B1 = Qi*B
```

Scilab code Exa 10.a.13 Designing a regulator using a minimum order observer

```
1 // Example A-10-13
```

```

2 // Designing a regulator using a minimum order
   observer
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6 mode(0);
7
8 function smallplot(i)
9   subplot(3,2,i);xgrid(color('gray'));
10  plot(t,x(i,:));
11 endfunction
12
13
14 A = [0 0 1 0; 0 0 0 1; -36 36 -0.6 0.6; 18 -18 0.3
      -0.3];
15 B = [0; 0; 1; 0];
16 C = [1 0 0 0; 0 1 0 0];
17 D = [0;0];
18 Gp = syslin('c',A,B,C,D);
19
20 Aab = A(1:2,3:$);
21 Abb = A(3:$,3:$);
22
23 P = [-2 + %i*2*sqrt(3), -2 - %i*2*sqrt(3), -10, -10 ]
24 Q = [-15 -16] // observer poles
25
26 K = ppol(A,B,P)
27 Ke = ppol(Abb',Aab',Q)'
28 Kb = K(3:$);
29
30 AA = [A - B*K , B*Kb; zeros(2,4) , Abb - Ke*Aab]
31
32 // system response to initial conditions
33 x0 = [0.1; 0; 0; 0; 0.1; 0.05];
34 G = syslin('c',AA,zeros(6,1),zeros(1,6),[0],x0);
35
36 t = 0:0.01:4;
37 u = zeros(1,length(t));

```

```

38 [y x] = csim(u,t,G);
39
40 smallplot(1);
41 xtitle('Response to initial condition','t (sec)','x1');
42 smallplot(2);
43 xtitle('Response to initial condition','t (sec)','x2');
44 smallplot(3);
45 xtitle(' ','t (sec)','x3');
46 smallplot(4);
47 xtitle(' ','t (sec)','x4');
48 smallplot(5);
49 xtitle(' ','t (sec)','e1');
50 smallplot(6);
51 xtitle(' ','t (sec)','e2');

```

Scilab code Exa 10.a.14 Designing a regulator using a minimum and full order observer

```

1 // Example A-10-14
2 // Designing a regulator using a minimum and full
   order observer
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6 mode(0);
7
8 // please edit the path
9 // cd "<path to dependencies>";
10 // exec("minorder.sci");
11
12 function smallplot(i)

```

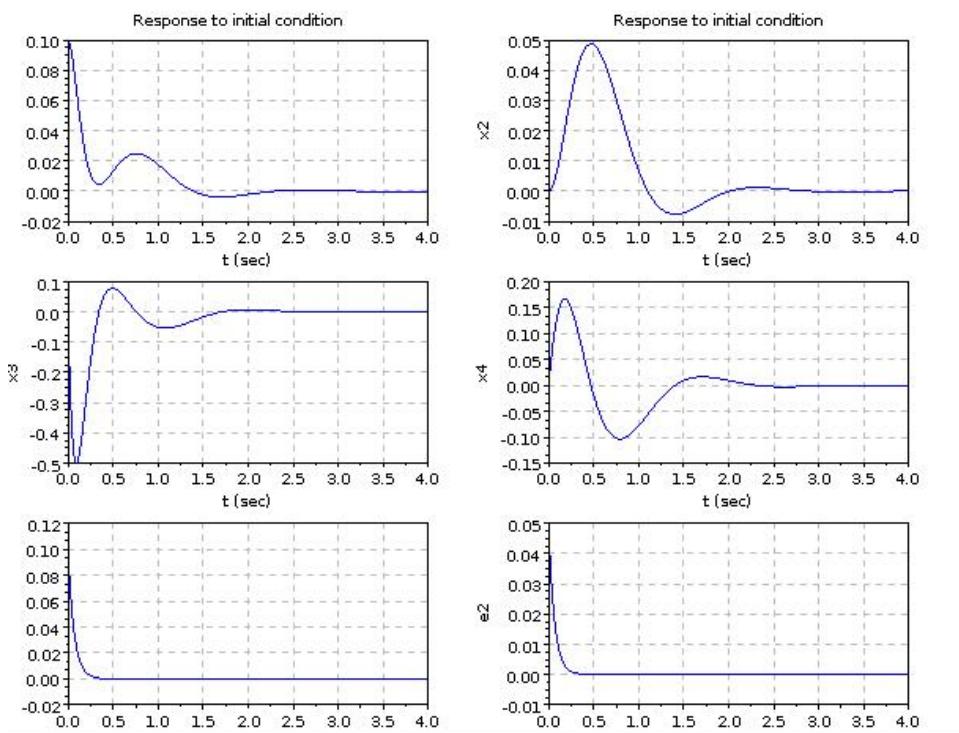


Figure 10.5: Designing a regulator using a minimum order observer

```

13 subplot(2,2,i);xgrid(color('gray'));
14 plot(t,x(i,:));
15 endfunction
16
17 A = [0 1; 0 -2];
18 B = [0; 4];
19 C = [1 0];
20 D = [0];
21 Gp = syslin('c',A,B,C,D);
22
23 P = [-2 + %i*2*sqrt(3), -2 - %i*2*sqrt(3)]
24 Q1 = [-8 -8]
25 Q2 = [-8];
26
27 disp('full order obssrver -');
28 K1 = ppol(A,B,P)
29 Ke1 = ppol(A',C',Q1)'
30
31 Go1 = transferf(A-B*K1-Ke1*C,Ke1,K1,[0]);
32 disp(Go1,'full order observer controller transfer
    function =');
33
34 // system response to initial conditions
35 AA1 = [A - B*K1, B*K1; zeros(2,2), A - Ke1*C];
36 x0 = [1; 0; 1; 0];
37 G = syslin('c',AA1,zeros(4,1),zeros(1,4),[0],x0);
38
39 t = 0:0.05:8;
40 u = zeros(1,length(t));
41 [y x] = csim(u,t,G);
42 smallplot(1);
43 xtitle('Response to initial condition (Full order)',
    't (sec)', 'x1');
44 smallplot(2);
45 xtitle('Response to initial condition (Full order)',
    't (sec)', 'x2');
46 smallplot(3);
47 xtitle(' ', 't (sec)', 'e1');

```

```

48 smallplot(4);
49 xtitle(' ', 't (sec)', 'e2');
50
51 disp('minimal order observer -');
52 P
53 Q2
54 [K2 Ke2 Go2 ch AA2] = minorder(A,B,P,Q2);
55 K2
56 Ke2
57 disp(Go2, 'minimal order observer controller transfer
      function =');
58
59 x0 = [1; 0; 1];
60 G = syslin('c', AA2, zeros(3,1), zeros(1,3), [0], x0);
61
62 t = 0:0.05:8;
63 u = zeros(1,length(t));
64 [y x] = csim(u,t,G);
65 scf();
66 smallplot(1);
67 xtitle('Response to initial condition (minimal order
      )', 't (sec)', 'x1');
68 smallplot(2);
69 xtitle('Response to initial condition (minimal order
      )', 't (sec)', 'x2');
70 smallplot(3);
71 xtitle(' ', 't (sec)', 'e');
72
73 scf();
74 // Bode diagram
75 C1 = Go1*Gp /. 1;
76 C2 = Go2*Gp /. 1;
77 bode([C1 ; C2], 0.1, 100, ['System 1'; 'System 2']);

```

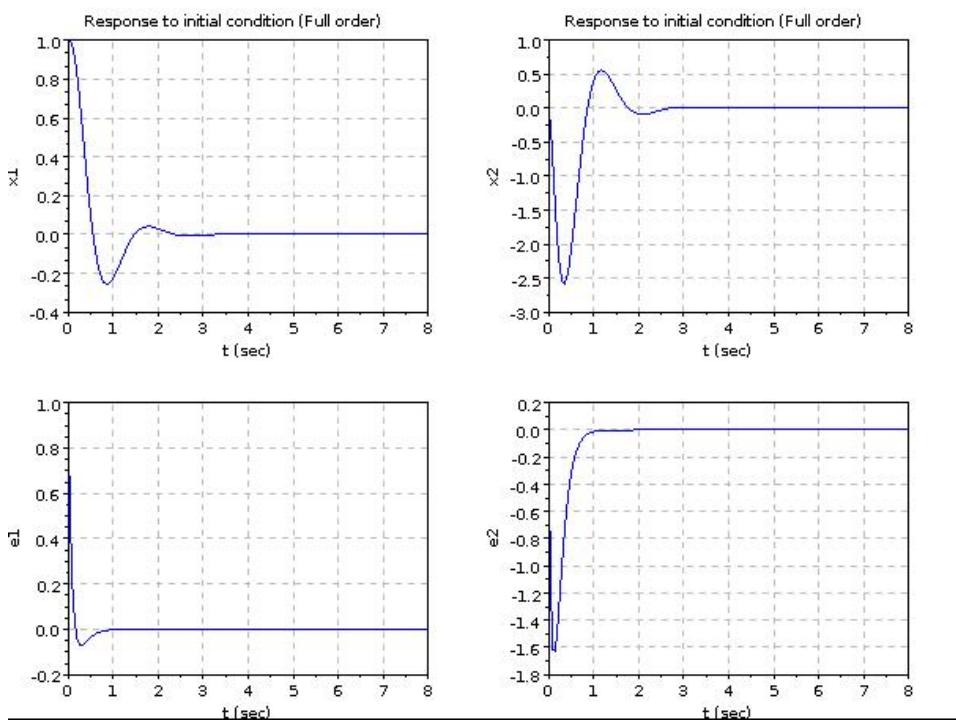


Figure 10.6: Designing a regulator using a minimum and full order observer

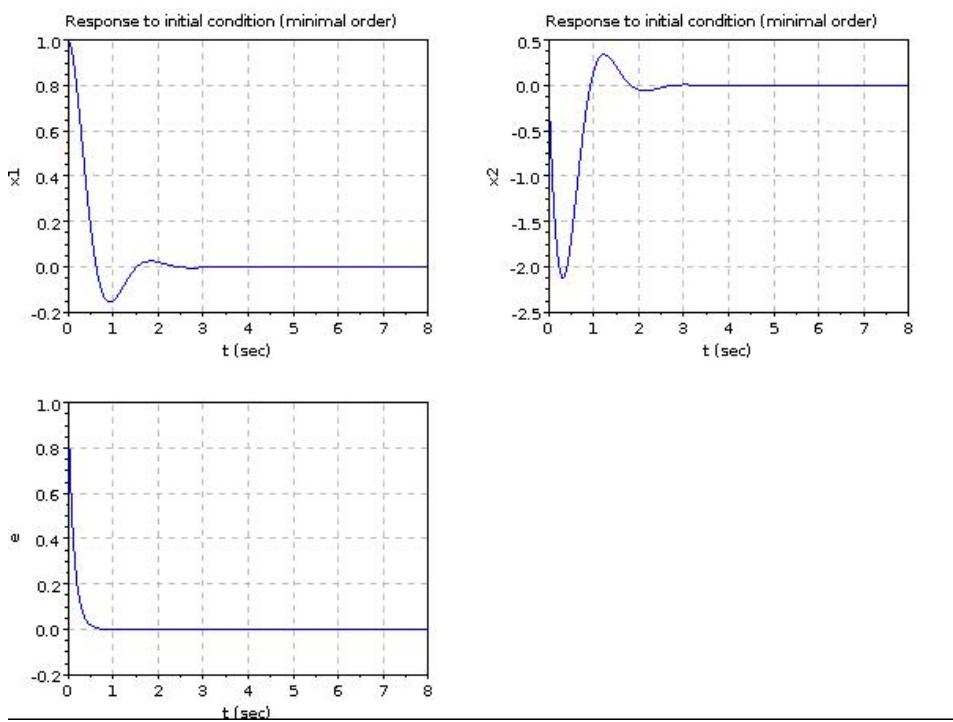


Figure 10.7: Designing a regulator using a minimum and full order observer

check Appendix AP 1 for dependency:

`minorder.sci`

Scilab code Exa 10.a.17 Design of quadratic optimal regulator system and finding the response

```
1 // Example A-10-17
2 // Design of quadratic optimal regulator system and
   finding the response
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6 mode(0);
7
8 function smallplot(i)
9     subplot(3,2,i); xgrid(color('gray'));
10    plot(t,x(i,:));
11 endfunction
12
13 A = [0 1 0 0; 20.601 0 0 0; 0 0 0 1; -0.4905 0 0 0];
14 B = [0; -1; 0; 0.5];
15 C = [0 0 1 0];
16
17 Ahat = [A zeros(4,1); -C 0]
18 Bhat = [B ; 0]
19
20 Q = eye(5,5); Q(1,1) = 100
21 R = [0.01]
22
23 // solve the riccati equation
24 P = riccati(Ahat, Bhat*inv(R)*Bhat', Q, 'c');
25 K = inv(R)*Bhat'*P
26 k1 = -K($);
27
28 AA = Ahat - Bhat*K
```

```

29 G = syslin('c',AA,[zeros(4,1); 1] , [C 0], [0]);
30 t = 0:0.05:10;
31 u = ones(1,length(t));
32 [y,x] = csim(u,t,G);smallplot(1);
33
34 xtitle('x1','t (sec)', '');
35 smallplot(2);
36 xtitle('x2','t (sec)', 'x2');
37 smallplot(3);
38 xtitle('','t (sec)', 'x3');
39 smallplot(4);
40 xtitle('','t (sec)', 'x4');
41 smallplot(5);
42 xtitle('','t (sec)', 'x5');

```

check Appendix AP 3 for dependency:

`ackermann.sci`

Scilab code Exa 10.1 Gain matrix using characteristic eq and Ackermannns formula

```

1 // Example 10-1
2 // Gain matrix using characteristic eq and
   Ackermannns formula
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6 mode(0);
7
8 // please edit the path
9 // cd "<path to dependencies>";
10 // exec(" ackermann.sci");
11

```

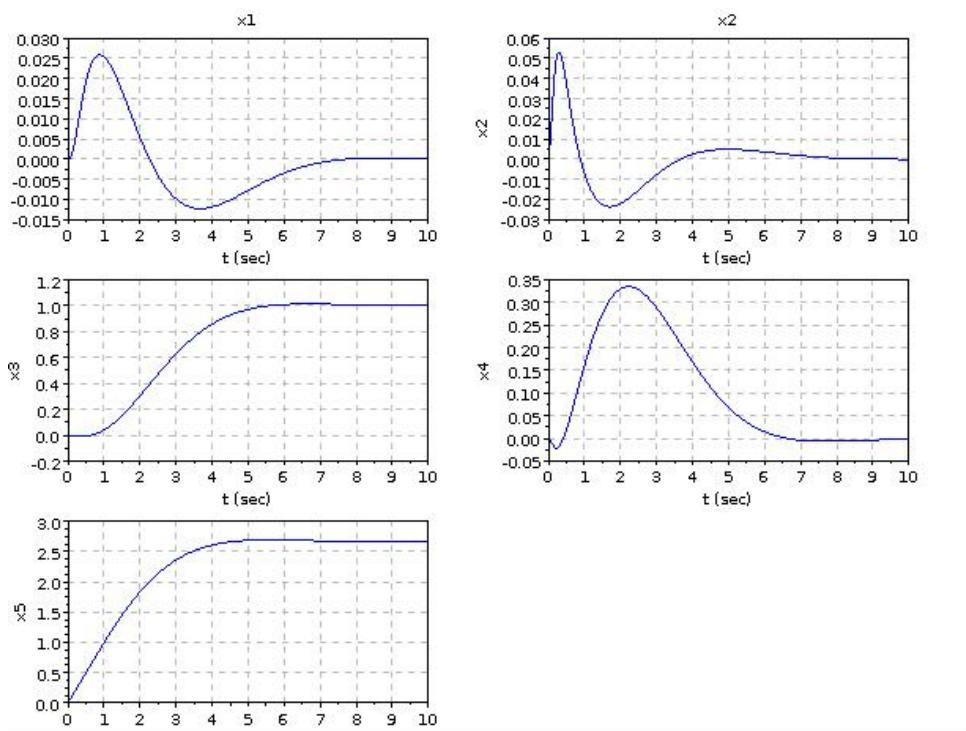


Figure 10.8: Design of quadratic optimal regulator system and finding the response

```

12 A = [0 1 0; 0 0 1;-1 -5 -6];
13 B = [0; 0; 1];
14 P = [-2 + %i*4 , -2 - %i*4, -10];
15
16 // Method 1
17 phi = poly(spec(A), 's');
18 disp(phi, 'Given systems characteristic eq = ');
19 cf = coeff(phi);
20 a = cf(1:$-1)
21
22 phid = poly(P, 's');
23 disp(phid, 'Desired characteristic eq = ');
24 cf = coeff(phid);
25 alpha = cf(1:$-1)
26
27 T = eye(3,3) // in this case
28 K = (alpha - a) * inv(T)
29
30 // Method 2
31 [K, phiA] = ackermann(A,B,P);
32 disp(cont_mat(A,B), ' controllability matrix = ');
33 disp(phiA, 'phi(A) = ');
34 disp(K, 'using ackermannns formula K = ');

```

check Appendix AP 3 for dependency:

`ackermann.sci`

Scilab code Exa 10.2 Gain matrix using ppol and Ackermannns formula

```

1 // Example 10-2
2 // Gain matrix using ppol and Ackermannns formula
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6

```

```

7 // please edit the path
8 // cd "<path to dependencies>";
9 // exec("ackermann.sci");
10
11 A = [0 1 0; 0 0 1;-1 -5 -6];
12 B = [0; 0; 1];
13 P = [-2 + %i*4 , -2 - %i*4, -10];
14 K = ackermann(A,B,P); disp(K, 'using ackermannns
    formula K = ');
15 K = ppol(A,B,P);   disp(K, 'using ppol function K = '
    )
16
17 // ackermann's formula is computationally tedious
18 // and hence avoided

```

Scilab code Exa 10.3 Response to initial condition

```

1 // Example 10-3
2 // Response to initial condition
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6
7 A = [0 1 0; 0 0 1;-1 -5 -6];
8 B = [0; 0; 1];
9 C = [0 0 0];
10 D = 0;
11 K = [199 55 8];
12 x0 = [1; 0; 0]; // initial state
13
14 G = syslin('c',(A - B*K),C',C,D,x0);
15 t = 0:0.01:4;
16 u = zeros(1,length(t)); // zero input response
17 [y x] = csim(u,t,G);
18

```

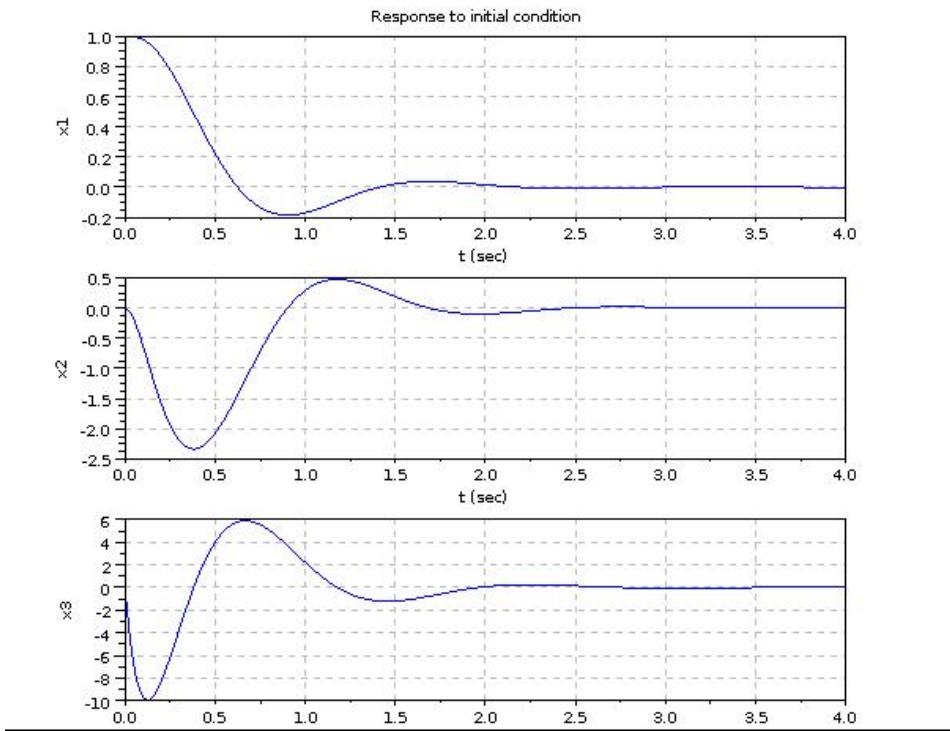


Figure 10.9: Response to initial condition

```

19 xtitle('Response to initial condition','t (sec)','x1');
20 subplot(3,1,1);xgrid(color('gray'));
21 plot(t,x(1,:));
22 subplot(3,1,2);xgrid(color('gray'));
23 xtitle(' ','t (sec)','x2');
24 plot(t,x(2,:));
25 subplot(3,1,3);xgrid(color('gray'));
26 xtitle(' ','t (sec)','x3');
27 plot(t,x(3,:));

```

check Appendix AP 2 for dependency:

`plotresp.sci`

Scilab code Exa 10.4 Design of servo system with integrator in the plant

```
1 // Example 10-4
2 // Design of servo system with integrator in the
   plant
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6 mode(0)
7
8 // please edit the path
9 // cd "<path to dependencies>";
10 // exec("plotresp.sci");
11
12 s = %s;
13 Gp = cont_frm( 1, s*(s+1)*(s+2));
14 A = Gp.A
15 B = Gp.B
16 J = [-2 + %i*2*sqrt(3) , -2 - %i*2*sqrt(3) , -10];
17 K = ppol(A,B,J)
18
19 A1 = A - B*K;
20 B1 = [0; 0; 160];
21 C1 = [1 0 0];
22 D1 = [0];
23
24 G = syslin('c',A1,B1,C1,D1); ssprint(G);
25
26 t = 0:0.01:5;
27 u = ones(1,length(t));
28 plotresp(u,t,G,'Unit-Step Response of servo system')
;
```

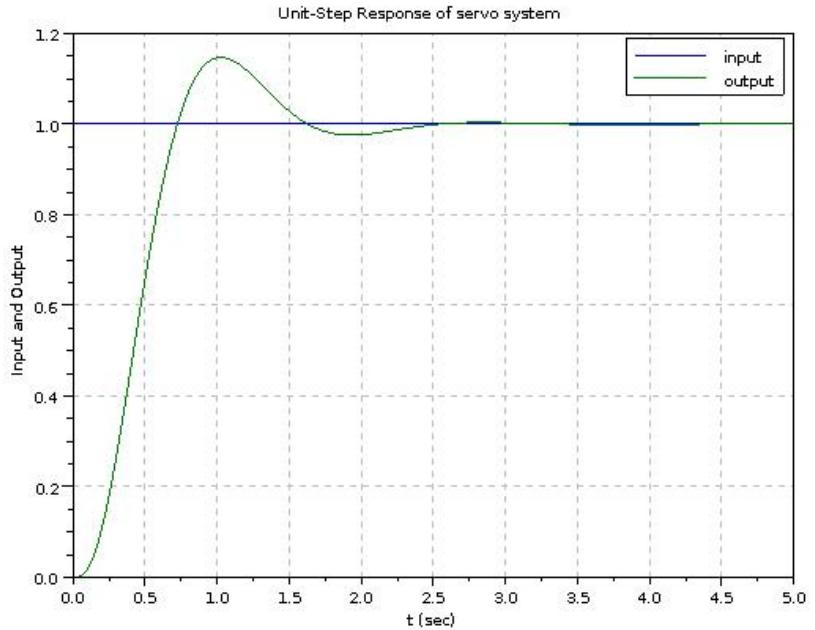


Figure 10.10: Design of servo system with integrator in the plant

Scilab code Exa 10.5 Design of servo system without integrator in the plant

```

1 // Example 10-5
2 // Design of servo system without integrator in the
   plant
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6 mode(0);

```

```

7
8 function smallplot(i)
9   subplot(3,2,i);xgrid(color('gray'));
10  plot(t,x(i,:));
11 endfunction
12
13 // Plant
14 A = [0 1 0 0; 20.601 0 0 0; 0 0 0 1; -0.4905 0 0 0];
15 B = [0; -1; 0; 0.5];
16 C = [0 0 1 0];
17 J = [-1 + %i*sqrt(3) , -1 - %i*sqrt(3), -5, -5, -5];
18
19
20 // Error dynamics with the error as a state variable
21
22 Ahat = [A zeros(4,1); -C 0];
23 Bhat = [B ; 0];
24 Khat = ppol(Ahats,Bhat,J)
25 K = Khat(1: $-1)
26 k1 = -Khat($)
27
28 // Over all system with the error as a state
29 // variable
30 A1 = Ahat - Bhat*Khat;
31 B1 = [zeros(4,1); 1];
32 C1 = [C , 0];
33 D1 = [0];
34 G = syslin('c',A1,B1,C1,D1);
35
36 t = 0:0.02:6;
37 u = ones(1,length(t));
38 [y ,x] = csim(u,t,G);
39
40 smallplot(1);
41 xtitle('x1','t (sec)', '');
42 smallplot(2);
43 xtitle('x2','t (sec)', 'x2');
44 smallplot(3);

```

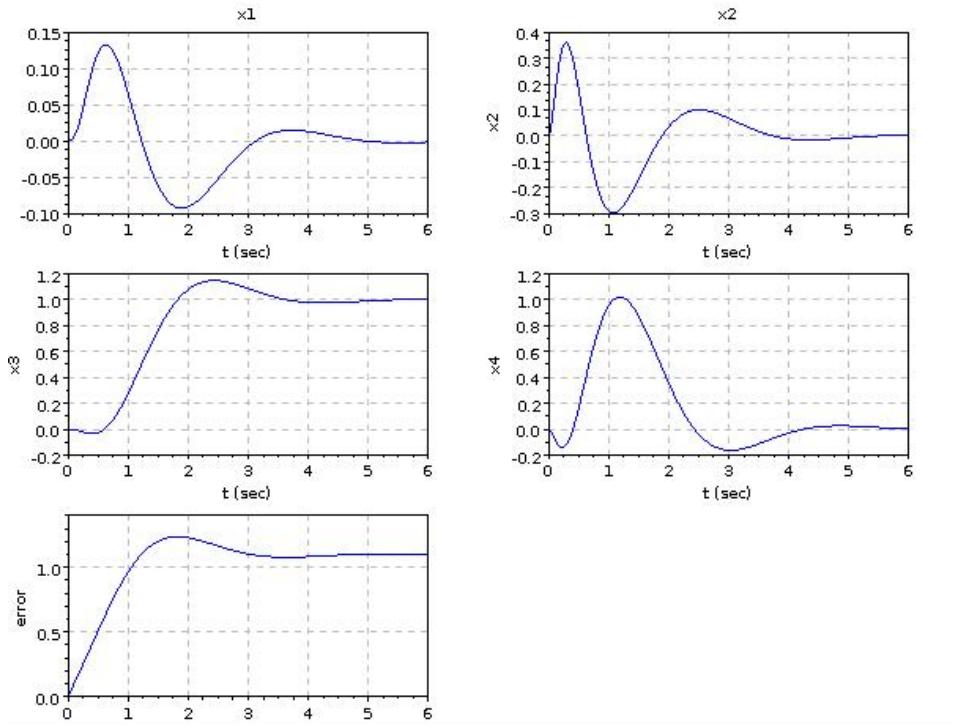


Figure 10.11: Design of servo system without integrator in the plant

```

44 xtitle( '' , 't ( sec )' , 'x3' );
45 smallplot(4);
46 xtitle( '' , 't ( sec )' , 'x4' );
47 smallplot(5);
48 xtitle( '' , 't ( sec )' , 'error' );

```

check Appendix [AP 3](#) for dependency:

`ackermann.sci`

Scilab code Exa 10.6 Observer Gain matrix using ch eq and Ackermann's formula

```

1 // Example 10-6
2 // Observer Gain matrix using ch eq and Ackermannns
   formula
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6 mode(0);
7
8 // please edit the path
9 // cd "<path to dependencies>";
10 // exec("ackermann.sci");
11
12 A = [0 20.6; 1 0];
13 C = [0 1];
14 P = [-10 -10];
15
16 // Method 1
17 phi = poly(spec(A), 's');
18 disp(phi, 'Given systems characteristic eq = ');
19 cf = coeff(phi);
20 a = cf(1:$-1)';
21
22 phid = poly(P, 's');
23 disp(phid, 'Desired characteristic eq = ');
24 cf = coeff(phid);
25 alpha = cf(1:$-1)'
26
27 T = eye(2,2) // in this case
28 Ke = inv(T) * (alpha - a)
29
30 // Method 2
31 [Ke, phiA] = ackermann(A', C', P);
32 disp(obsv_mat(A, C), 'observability matrix = ');
33 disp(phiA', 'phi(A) = ');
34 disp(Ke', 'using ackermannns formula Ke = ');

```

Scilab code Exa 10.7 Designing a controller using a full order observer

```
1 // Example 10-7
2 // Designing a controller using a full order
   observer
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6 mode(0);
7
8 function smallplot(i)
9   subplot(2,2,i); xgrid(color('gray'));
10  plot(t,x(i,:));
11 endfunction
12
13 s = %s;
14 A = [0 1; 20.6 0];
15 B = [0; 1];
16 C = [1 0];
17 D = [0];
18 P = [-1.8 + %i*2.4, -1.8 - %i*2.4];
19 Q = [-8 -8]; // observer poles
20
21 K = ppol(A,B,P)
22 Ke = ppol(A',C',Q)'
23
24 // The transfer function of observer controller
25 A1 = A - B*K - Ke*C
26 M = s*eye(A1) - A1
27 UbyE = K * inv(M) * Ke;
28 disp(UbyE, 'U(s) / E(s) =');
29
30 // Plant dynamics
31 Gp = syslin('c',A,B,C,D);
```

```

32 disp('plant dynamics'); ssprint(Gp);
33 YbyU = ss2tf(Gp)
34
35 // Observer controller dynamics
36 disp('observer controller dynamics (x = xbar) ,(u =
y), (y = u)');
37 Goc = syslin('c',A1,Ke,-K,[0]);
38 ssprint(Goc);
39
40 // Overall System transfer funtion
41
42 GsFullsystem = UbyE * YbyU /. 1
43
44 // Overall System
45 x0 = [1; 0; 0.5; 0]; // initial state
46 As = [A-B*K, B*K ; zeros(2,2) , A-Ke*C];
47 Gss = syslin('c',As,[1;0;0;0] , [1 0 0 0] , [0],x0);
48
49 // Unit step response
50 t = 0:0.01:4;
51 u = zeros(1,length(t));
52 [y x] = csim(u,t,Gss);
53
54 smallplot(1);
55 xtitle('Response to initial condition ','t (sec)', 'x1');
56 smallplot(2);
57 xtitle('Response to initial condition ','t (sec)', 'x2');
58 smallplot(3);
59 xtitle(' ', 't (sec)', 'e1');
60 smallplot(4);
61 xtitle(' ', 't (sec)', 'e2');

```

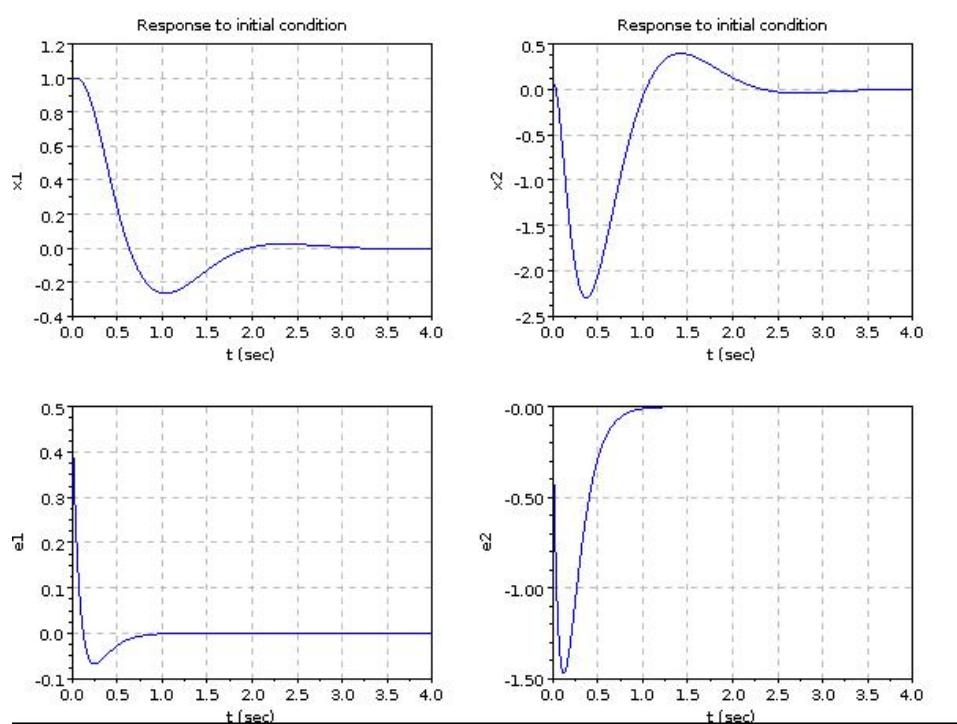


Figure 10.12: Designing a controller using a full order observer

Scilab code Exa 10.8 Designing a controller using a minimum order observer

```
1 // Example 10-8
2 // Designing a controller using a minimum order
   observer
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6 mode(0);
7
8 A = [0 1 0; 0 0 1; -6 -11 -6];
9 B = [0; 0; 1];
10 C = [1 0 0];
11 D = [0];
12 P = [-2 + %i*2*sqrt(3), -2 - %i*2*sqrt(3), -6];
13 Q = [-10 -10]; // observer poles
14
15 K = ppol(A, B, P)
16
17 // Observer design
18 Aaa = A(1,1)
19 Aab = A(1,2:$)
20 Aba = A(2:$,1)
21 Abb = A(2:$,2:$)
22
23 Ke = ppol(Abb', Aab', Q)'
24
25 Ba = B(1,1)
26 Bb = B(2:$,1)
27
28 Ahat = Abb - Ke*Aab;
29 disp(Ahat, 'Ahat = Abb - Ke*Aab =');
30 Bh = Aba - Ke*Aaa;
31 disp(Bh, 'Aba - Ke*Aaa =');
32 Chat = [zeros(1,2); eye(2,2)]
33 Dhat = [1; Ke]
34 Fhat = Bb - Ke*Ba;
```

```
35 disp(Fhat, 'Fhat = Bb - Ke*Ba =');
```

Scilab code Exa 10.9 Design of quadratic optimal regulator system

```
1 // Example 10.9
2 // Design of quadratic optimal regulator system
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6 mode(0);
7
8 A = [0 1;0 0];
9 B = [0;1];
10 Q = [1 0; 0 1];
11 R = [1];
12
13 // solve the riccati equation
14 P = riccati(A, B*inv(R)*B', Q, 'c')
15 K = inv(R)*B'*P
16 E = spec(A - B*K) // eigen values
```

Scilab code Exa 10.10 Design of quadratic optimal regulator system

```
1 // Example 10-10
2 // Design of quadratic optimal regulator system
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6 mode(0);
7
8 A = [-1 1;0 2];
9 B = [1;0];
10 Q = [1 0; 0 1];
```

```

11 R = [1];
12
13 // solve the riccati equation
14 P = riccati(A, B*inv(R)*B', Q, 'c')
15 K = inv(R)*B'*P
16 E = spec(A - B*K) // eigen values
17 // when a solution does not exist
18 // a different method is used - least square
    solution

```

Scilab code Exa 10.11 Design of quadratic optimal regulator system

```

1 // Example 10-11
2 // Design of quadratic optimal regulator system
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6 mode(0);
7
8 A = [0 1;0 -1];
9 B = [0;1];
10 Q = [1 0; 0 1];
11 R = [1];
12
13 // solve the riccati equation
14 P = riccati(A, B*inv(R)*B', Q, 'c')
15 K = inv(R)*B'*P
16 E = spec(A - B*K) // eigen values

```

Scilab code Exa 10.12 Design of quadratic optimal regulator system and finding the response

```
1 // Example 10-12
```

```

2 // Design of quadratic optimal regulator system and
   finding the response
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6 mode(0);
7
8 A = [0 1 0; 0 0 1; -35 -27 -9];
9 B = [0; 0; 1];
10 Q = [1 0 0; 0 1 0; 0 0 1];
11 R = [1];
12
13 // solve the riccati equation
14 P = riccati(A, B*inv(R)*B', Q, 'c')
15 K = inv(R)*B'*P
16 E = spec(A - B*K) // eigen values
17
18 x0 = [1; 0; 0]; // initial state
19
20 G = syslin('c',(A - B*K),[0;0;0],[0 0 0],[0], x0);
21 t = 0:0.01:8;
22 u = zeros(1,length(t));
23 [y x] = csim(u,t,G);
24
25 xtitle('Response to initial condition','t (sec)', 'x1');
26 subplot(3,1,1);xgrid(color('gray'));
27 plot(t,x(1,:));
28
29 subplot(3,1,2);xgrid(color('gray'));
30 xtitle(' ', 't (sec)', 'x2');
31 plot(t,x(2,:));
32
33 subplot(3,1,3);xgrid(color('gray'));
34 xtitle(' ', 't (sec)', 'x3');
35 plot(t,x(3,:));

```

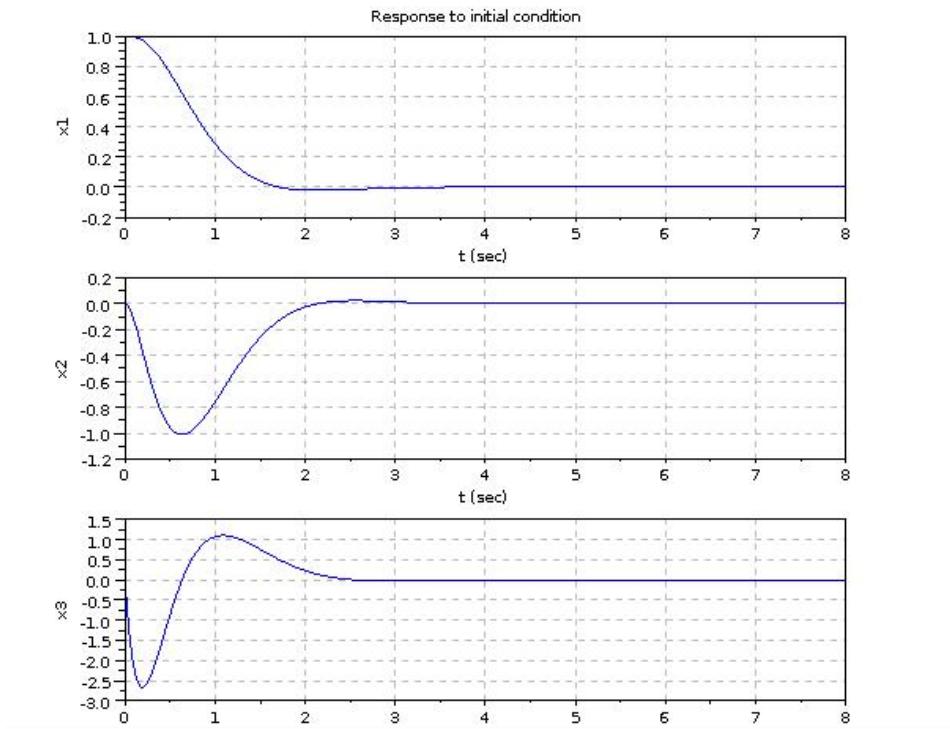


Figure 10.13: Design of quadratic optimal regulator system and finding the response

Scilab code Exa 10.13 Design of quadratic optimal regulator system and finding the response

```

1 // Example 10-13
2 // Design of quadratic optimal regulator system
3
4 clear; clc;
5 xdel(winsid()); // close all windows
6

```

```

7 A = [0 1 0; 0 0 1; 0 -2 -3];
8 B = [0; 0; 1];
9 C = [1 0 0];
10 Q = [100 0 0; 0 1 0; 0 0 1];
11 R = [0.01];
12
13 // solve the riccati equation
14 P = riccati(A, B*inv(R)*B', Q, 'c');
15 K = inv(R)*B'*P;
16 disp(K, 'K = ');
17 k1 = K(1);
18
19 G = syslin('c', A - B*K, B*k1, C, [0]);
20 t = 0:0.01:8;
21 u = ones(1, length(t));
22 [y, x] = csim(u, t, G);
23 plot(t, x);
24 xgrid(color('gray'));
25 xtitle('Step-Response', 't (sec)', 'state variables');
26 legend('x1 (= y)', 'x2', 'x3');

```

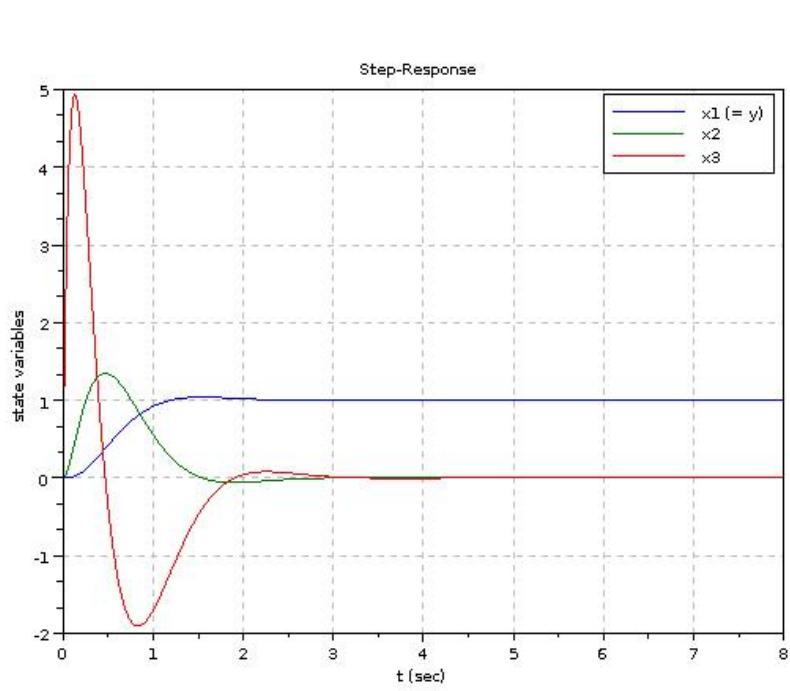


Figure 10.14: Design of quadratic optimal regulator system and finding the response

Appendix

Scilab code AP 1 Determine Gains and transfer function for minimal order observer

```
1
2 // Determine Gains and transfer function for minimal
   order observer
3
4 function G = transferf(A,B,C,D)
5   H = syslin('c',A,B,C,D);
6   G = clean(ss2tf(H));
7 endfunction
8
9 function [K,Ke,Go,ch,AA,Ahat,Bhat,Chat,Dhat,Fhat] =
   minorder(A,B,P,Q)
10    s = %s;
11    K = ppol(A,B,P);
12    Ka = K(1);
13    Kb = K(2:$);
14
15    Aaa = A(1,1);
16    Aab = A(1,2:$);
17    Aba = A(2:$,1);
18    Abb = A(2:$,2:$);
19    Ba = B(1,1);
20    Bb = B(2:$,1);
21
22    Ke = ppol(Abb',Aab',Q)'
23
```

```

24     n = length(Kb);
25     Ahat = Abb - Ke*Aab;
26     Bhat = Ahat*Ke + Aba - Ke*Aaa;
27     Chat = [zeros(1,n); eye(n,n)];
28     Dhat = [1; Ke];
29     Fhat = Bb - Ke*Ba;
30     Atld = Ahat - Fhat*Kb;
31     Btld = Bhat - Fhat*(Ka + Kb*Ke);
32     Ctld = -Kb;
33     Dtld = -(Ka + Kb*Ke);
34
35     Go = transferf(Atld,Btld,-Ctld,-Dtld);
36     ch = det(s*eye(n+1,n+1) - A + B*K) * det(s*eye(n,n)
37           - Abb + Ke*Aab);
38     AA = [A - B*K , B*Kb; zeros(n,n+1) , Abb - Ke*Aab];
39 endfunction

```

Scilab code AP 2 Plot System Response

```

1
2 // Plot System Response
3 // Computes the response and plots the input and
4 // response together
5 function y = plotresp(u,t,G,text)
6     y = csim(u,t,G);
7     plot(t,u,t,y);
8     xtitle(text,'t (sec)', 'Input and Output');
9     xgrid(color('gray'));
10    legend('input','output');
11 endfunction

```

Scilab code AP 3 Compute the feedback gain matrix using ackermannns formula

```

1 // Compute the feedback gain matrix using ackermannns
// formula

```

```

2
3 function [K ,phiA] = ackermann(A,B,P)
4   // construct characteristic equation
5   phi = poly(P, 'x');
6   c = coeff(phi);
7   phiA = eye(A)*c(1);
8   powA = eye(A);
9   for i=2:length(c)
10    powA = powA * A;
11    phiA = phiA + powA * c(i);
12  end
13  K = [zeros(1,length(B)- 1), 1] * inv(cont_mat(A,B)
14    ) * phiA;
14 endfunction

```

Scilab code AP 4 Transfer function of A,B,C,D.

```

1
2 function G = transferf(A,B,C,D)
3   H = syslin('c',A,B,C,D);
4   G = clean(ss2tf(H));
5 endfunction

```

Scilab code AP 5 Inverse Laplace transform of a rational polynomial in s

```

1 // Inverse Laplace transform of a rational
2 // polynomial in s
3 // depends on pf_residu
4
5 function s = ilaplace(H)
6   if(H ~= 0) then
7     [r z p] = pf_residu(H.num,H.den);
8     n = length(r);
9     s = '';
10    for i = 1:(n-1) ;
11      s = s + string(r(i)) + '*e^' + string(p(i)) +
12        't  + ';
11 end

```

```

12     s = s + string(r(n)) + '*e^' + string(p(n)) + 't
13     ';
14 else
15     s = '0';
16 end
17 endfunction

```

Scilab code AP 6 Partial Fraction Residue

```

1
2 // Partial Fraction Residue
3 // Gives the coefficients of partial fraction
   expansion for the given polynomial
4
5 function [r,z,p] = pf_residu(N,D)
6 z = roots(N) //Zeros
7 p = roots(D) //Poles
8
9 q = round(p);
10 m = 1; // to keep a count of the root's
   multiplicity
11
12 for i = 1:length(p)
13   if(i < length(p) & q(i + 1) == q(i))
14     m = m + 1;
15   else
16     P1 = N / pdiv(D,( s - p(i)) ^ m );
17     r(i) = horner(P1 ,p(i));
18     for j = 1:(m-1)
19       P1 = derivat(P1);
20       r(i - j) = horner(P1 / gamma(j + 1) ,p(i));
21     end // gamma(j + 1) = j! ( factorial
           )
22     m = 1;
23   end
24 end
25 endfunction
26

```

```
27 // for details on this method please refer  
28 // http://en.wikipedia.org/wiki/Partial_fraction
```

Scilab code AP 7 Plot the root locus in a box

```
1 // Plot the root locus in a box  
2 // rootl(G,box,text)  
3 // G : linear system  
4 // box: so ordinates of axis bounds  
5 // text: title of plot window  
6  
7 function rootl(G,box,text)  
8   evans(G);  
9   xgrid();  
10  a = gca();  
11  if box ~= 0 then  
12    a.box = "on";  
13    a.data_bounds = box;  
14  end  
15  a.children(1).visible = 'off'; //remove the legend  
     block  
16  xtitle(text);  
17 endfunction
```

Scilab code AP 8 Step response characteristics

```
1 // Step response characteristics  
2 // Plots the step response and computes Maximum  
   Overshoot  
3 // Peak Time, Rise Time and Settling Time  
4  
5 function [Mp,tp,tr,ts] = stepch(G,from,to,step,  
      settling_margin)  
6  
7 t = from:step:to;  
8 u = ones(1,length(t));  
9 y = csim(u,t,G);  
10 plot(t,y);
```

```

11  xtitle('Unit Step Response', 't ( sec )', 'Output');
12  xgrid(color('gray'));
13
14  [m t1] = max(y);
15  tp = (t1 - 1) * step;
16  Mp = m - 1;
17
18  i = 1;
19  if tp == to then
20      tr = %nan;
21  else
22      while(y(i) < 0.1) i = i + 1; end;
23      r1 = i;
24      while(y(i) < 0.9) i = i + 1; end;
25      tr = (i-r1) * step;
26  end
27
28  l = 1 - settling_margin;
29  h = 1 + settling_margin;
30  for i = length(t):-1:1
31      if( y(i) < l | y(i) > h) break; end;
32  end
33  ts = (i - 1) * step;
34 endfunction

```

Scilab code AP 9 Polar plot of a linear system

```

1 // polar plot of a linear system
2 // repf = spolarplot(G,omega)
3 // G: linear sytem and omega:is frequency in rad/s
4 // repf: is the complex frequency response
5
6 function repf = spolarplot(G,omega)
7     f = omega /2/%pi;
8     repf = repfreq(G,f);
9     r = abs(repf);
10    theta = atan(imag(repf),real(repf));
11    polarplot(theta,r,style = 2);

```

```
12 endfunction
```

Scilab code AP 10 Display gain and phase margins

```
1 // Display gain and phase margins on a bode plot
2
3 function [gm,gcrf,pm,pcrf] = shmargins(G)
4
5 show_margins(G, 'bode');
6 xtitle('Bode diagram', 'rad/s');
7 a = gcf(); set(a.children(2).x_label, 'text', 'rad/s',
8 );
9 [gm pcrf] = g_margin(G);
10 [pm gcrf] = p_margin(G);
11 disp(gcrf, 'Gain crossover frequency = ', pm, 'Phase
12 margin (degrees)= ');
13 disp(pcrf, 'Phase crossover frequency = ', gm, 'Gain
margin (dB) = ');
14 endfunction
```

Scilab code AP 11 Frequency response characteristics

```
1 // Frequency response characteristics
2 function [Mr,wr,bw,repf] = freqch(G,omega)
3
4 repf = repfreq(G,omega); // frequency response
5 (complex numbers)
6 [mag phi] = dbphi(repf); // mag in db
7 [Mr k] = max(mag); // resonant peak
8 wr = omega(k); // resonant freq.
9 mag = abs(mag + 3); // mag = abs( mag - (- 3
10 dB) )
11 [M j] = min(mag); // j : is the point
12 where mag == -3db
13 bw = omega(j);
```

```
13     disp(wr, 'resonant frequency = ');
14     disp(Mr, 'resonant peak (dB)= ');
15     disp(bw, 'bandwidth = ');
16 endfunction
```

Scilab code AP 12 Gain at a point on a root locus

```
1 // Gain at a point on a root locus
2
3 function [K,p] = gainat(G)
4 z = locate(1,1);
5 x = z(1);y = z(2);
6 p = x + %i*y;
7 disp( p , 'p = ');
8 K = 1 / abs(horner(G,p))
9 disp( K , 'K = ');
10 plot(x,y,'.');
11 xstring(x,y,'K = ' + string(K));
12 endfunction
```
