LUND INSTITUTE OF TECHNOLOGY
DEPARTMENT OF AUTOMATIC CONTROL
Box 725
S 220 07 Lund 7 Sweden

Document name
Report

Date of issue
September 1982

Document number
CODEN:LUTFD2/(TFRT-5285)/1-065/(1982)

Author(s)
Per-Erik Svensson

Supervisor
Björn Wittenmark

Sponsoring organization

Title and subtitle
Experiment med en linjär-kvadratisk självinställare
(Experiments with a linear-quadratic selftuner)

Abstract
This report describes a program for a linear-quadratic selftuner. It also includes results from experiments with the regulator. In the experiments different processes has been simulated on an analog computer. An important result from the experiments is that if an antialiasfilter is used when regulating a process, the dynamics of the the filter must be included in the processmodel. It is also necessary to take into account the time-delay in the regulator when using short samlingperiods.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
Swedish

Number of pages
65

Recipient's notes

Security classification

Distribution: The report may be ordered from the Department of Automatic Control or borrowed through the University Library 2, Box 1010, S-221 03 Lund, Sweden, Telex: 33248 lubbis lund.

FÖRORD

Detta examensarbete har utförts vid institutionen för reglerteknik på Lunds Tekniska Högskola. Examensarbetet behandlar en linjär- kvadratisk självinställare.
Författaren vill rikta ett stort tack till Björn Wittenmark och övriga på institutionen som ställt sitt kunnande till mitt förfogande.

INNEHÅLL

1. INLEDNING

Detta examensarbete avser att visa några experiment med en linjär-kvadratisk självinställare. Algoritmen för regulatorn har konstruerats av Z.Y Zhou och K.J Åström.
I de inledande kapitlen beskrivs regulatorn kortfattat och efterföljande kapitel innehåller simuleringar med regulatorn.
I kapitel 2 och 3 beskrivs algoritmer för identifiering och beräkning av regulatorparametrar. I kapitel 4 beskrivs hur programmet används. En lista över parametrar och kommandon finns i detta kapitel. Kapitel 5 innehåller en förteckning över modifieringar som utförts i programmet innan simulering med regulatorn. I kapitel 6 presenteras några experiment som utförts och resultat av dessa. I ett appendix visas programmet för regulatorn i sin helhet.

2. BERÄKNING AV REGULATORPARAMETRAR

Betrakta ett system, med en insignal och en utsignal, som beskrivs av ekvationen

$$A(z^{-1})y(t) = B(z^{-1})u(t) + C(z^{-1})e(t) \qquad (2.1)$$

där A,B och C är polynom i bakåtskiftoperatorn. Kravet på regulatorn är att minimera förlustfunktionen

$$E\left[\lim_{t\to\infty}\frac{1}{t}\sum_{k=1}^{t}\left[y^2(k) + \rho u^2(k)\right]\right] \qquad (2.2)$$

Den återkoppling som minimerar ekvation (2.2) ges av

$$R(z^{-1})u(t) = C(z^{-1})y_{ref}(t) - S(z^{-1})y(t) \qquad (2.3)$$

Polynomen R och S beräknas genom följande procedur:

a) Karakteristiska polynomet $P(z^{-1})$ för det slutna systemet ges som lösning till spektralfaktoreringen

$$P(z)P(z^{-1}) = \rho\, A(z)A(z^{-1}) + B(z)B(z^{-1}) \qquad (2.4)$$

b) Polynomen R och S ges som lösningen till polynomekvationen

$$A(z^{-1})R(z^{-1}) + B(z^{-1})S(z^{-1}) = P(z^{-1})C(z^{-1}) \qquad (2.5)$$

alt b) Om integralverkan önskas hos regulatorn ges R och S som lösningen till polynomekvationen

$$A(z^{-1})R_1(z^{-1})\nabla + B(z^{-1})S(z^{-1}) = P(z^{-1})C(z^{-1}) \qquad (2.5')$$

där

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

$$R(z^{-1}) = \nabla\, R_1(z^{-1})$$

Ekvationerna (2.5) och (2.5') har entydig lösning om

$$\deg P \leq \deg A + \deg B \qquad (2.8)$$

Gradtalen för R och S ges då av

$$\left.\begin{aligned}\deg S &= \deg A - 1\\ \deg R &= \deg B - 1\end{aligned}\right\} \qquad (2.9)$$

Om villkoret (2.8) ej är uppfyllt finns två lösningar

$$\left.\begin{aligned}\deg R &= \deg P - \deg A\\ \deg S &= \deg A - 1\end{aligned}\right\} \qquad (2.10)$$

motsvarar en lösning med minimalt gradtal för $S(z^{-1})$ och

$$\left.\begin{aligned}\deg R &= \deg P - \deg B\\ \deg S &= \deg B - 1\end{aligned}\right\} \qquad (2.11)$$

motsvarar en lösning med minimalt gradtal för $R(z^{-1})$.

Det slutna systemet beskrivs av ekvationen

$$P(z^{-1})y(t) = B(z^{-1})y_{ref}(t) \qquad (2.12)$$

Regulatorn kan också användas vid polplacering. Istället för att beräkna $P(z^{-1})$ enligt ekvation (2.4) bestäms detta av operatören.

Algoritmen för spektralfaktorering beskrivs närmare i ref[1]. Lösningen av ekvation (2.5) och (2.5') behandlas utförligt i ref[2].

3. PROCESSIDENTIFIERING

Koefficienterna för polynomen A,B och C i ekvation (2.1) uppskattas av regulatorn genom rekursiv utvidgad minstakvadrat identifiering (ELS). För att beskriva denna metod införs vektorerna

$$\Theta = (a_1,.....,a_n,b_1,.....,b_m,c_1,.....,c_1)^T$$

$$\phi(t) = [-y(t-1),...,-y(t-n),u(t-1),...,u(t-n),$$

$$,\varepsilon(t-1),...,\varepsilon(t-1)]^T$$

där

$$\varepsilon(t+1) = y(t+1) - \phi(t+1)^T\Theta(t)$$

Algoritmen för minstakvadratidentifieringen ges då av

$$\left.\begin{aligned} \Theta(t+1) &= \Theta(t) + K(t+1)\varepsilon(t+1) \\ K(t+1) &= \frac{P(t)\phi(t+1)}{1 + \phi(t+1)^T P(t)\phi(t+1)} \\ P(t+1) &= \frac{1}{\lambda}\left\{ P(t) - \frac{P(t)\phi(t+1)\phi(t+1)^T P(t)}{1 + \phi(t+1)^T P(t)\ \phi(t+1)} \right\} \end{aligned}\right\} \qquad (3.1)$$

En utförlig beskrivning av metoden ges i ref[3].

4 PROGRAMBESKRIVNING

Programmet för regulatorn består av två delar:

Förgrundsprogrammet (foreground) utför systemidentifiering, spektralfaktorering och beräkning av regulatorparametrar.
Bakgrundsprogrammet (background) initialiserar regulatorn. Denna del innehåller också möjligheter för operatören att kommunicera med programmet under pågående exekvering.
Detta görs med hjälp av följande kommandon:

HELP: Listar möjliga kommandon, parametrar och in-ut-kanaler.

DISP: Listar aktuella parametrar (se nedan).

PAR: Ändrar parametrar.

RUN: Startar regulatorn.

STOP: Stoppar regulatorn

EXIT: Stoppar regulatorn och avbryter exekveringen.

STORE: Skapar en fil med styr- och mätsignaler.

RESULTU: Skapar en fil med regulatorparametrar.

RESULTP: Skapar en fil med processparametrar.

Följande tabell visar vilka parametrar som listas med kommandot DISP, och som kan ändras av operatören under pågående exekvering med kommandot PAR.

TSAMP Samplingsperioden.

LO ϱ i förlustfunktionen (2.2).

NS Koefficienterna i polynomen A,B,P,R och S listas under pågående exekvering. Hur ofta detta sker bestäms av 'ns' uttryckt i samplingsintervall.

NI Anger antalet iterationer som önskas vid spektralfaktoreringen.

ND Vid upptagning av simuleringsresultat på linjeskrivare krävs att styrsignalen presenteras med en viss tidsfördröjning Denna tidsfördröjning bestäms av 'nd' uttryck i samplingsintervallets längd

LAMBDA λ i ekvation (3.1).

PO Begynnelsevärden för matrisen P(t) i ekvation (3.1).

PY Parameter som användes vid digital filtrering av

mätsignalen.

EPS Skalfaktor vid beräkning av normen för ett polynom.

HILIM Maxvärde för styrsignal.

LOLIM Minvärde för styrsignal.

NTA,NTB Under exekvering inkrementeras en variabel 'k' vid varje samplingstillfälle. 'nta' och 'ntb' anger för vilka värden på denna variabel som koefficienter i A,B,C,R och S skall sparas.

YRCHAN Anger kanalnummer för referenssignal.

YCHÁN Anger kanalnummer för mätsignal.

UCHAN Anger kanalnummer för fördröjd styrsignal.

OUTCHAN Anger kanalnummer för styrsignal.

LOG Logisk variabel som anger om koefficienterna i polynomen A,B,C,R och S ska sparas.

MINR Logisk variabel som anger vilken av de två möjliga lösningarna (2.10) och (2.11) som ska väljas då villkoret (2.8) ej är uppfyllt.

INITA, INITP INITP Logiska variabler som anger om polynomen A,B och P skall initieras enligt operatörens önskemål.

NA,NB, NP,NT Gradtalen för polynomen A,B,P och C.

AA-AD Av operatören önskade startvärden för polynomet A.

BA-BD Av operatören önskade startvärden för polynomet B.

PZ-PE Av operatören önskade startvärden för polynomet P.

5 MODIFIERINGAR

För att få regulatorn att fungera enligt specifikationer måste följande ändringar utföras i det ursprungliga programmet.

a) I det ursprungliga programmet initialiserades polynomet $P(z^{-1})$ före varje spektralfaktorering. För att då få god överensstämmelse i ekvation (2.4) krävdes ett stort antal iterationer. Om istället polynomet från föregående samplingstillfälle används som startpolynom behövs ett mindre antal iterationer. Detta medför också kortare exekveringstid och mindre tidsfördröjning i regulatorn.

b) I proceduren REGDESIGN(A,B,R,S) beräknas polynomen R och S enligt ekvation (2.5). I denna procedure kontrolleras om villkoret (2.8) är uppfyllt:

```
if (p.d< a.d + b.d) and .......
```

där p.d,a.d och b.d är gradtalen för polynomen P,A och B. Eftersom polynomen A och B modifieras i proceduren TRANSFORMATION är denna konstruktion felaktig.
När A och B innehåller en gemensam faktor skall denna faktor förkortas bort innan ekvation (2.5) löses. Polynomen som erhålls efter eventuell förkortning har i programmet beteckningen A_1 och B_1. Villkoret (2.8) skall alltså kontrolleras på följande sätt:

```
if (p.d < a1.d + b1.d) and .......
```

c) När integralverkan önskas i regulatorn beräknas polynomen R och S enligt ekvation (2.5'). Detta erhålls i programmet genom att polynomet A multipliceras med $(1 - z^{-1})$ innan anrop av REGDESIGN. Men detta medför att gradtalet för A ökas. Variabeln 'nm' ,som anger det maximala gradtalet hos polynomen A och B, måste därmed ändras inuti proceduren REGDESIGN samtidigt som detta ej får ske i programmet utanför REGDESIGN. Detta problem elimineras genom att införa en för REGDESIGN lokal variabel 'nmloc'.

6. EXPERIMENT

I följande beskrivna experiment simuleras olika processer och filter på en analogimaskin. En utförlig beskrivning av dess funktion finns i ref[4].
Eftersom det finns flera sätt på vilket processer kan realiseras på en analogimaskin presenteras de kopplingar som använts i detta arbete i appendix A.
I samtliga experiment används regulator med integralverkan.

6.1 Inverkan av samplingsintevall

I detta experiment studeras hur samplingsintervallet inverkar på regulatorns förmåga att identifiera den process som regleras. Det slutna systemets stegsvar jämförs med det av regulatorn beräknade karakteristiska polynomet. Den process som regleras har överföringsfunktionen

$$G(s) = \frac{\omega_0^2}{s^2 + 2\omega_0\xi_0 s + \omega_0^2} \qquad (6.1)$$

I samplad form med samplingsintevallet h beskrivs processen av av pulsöverföringsfunktionen

$$H(z^{-1}) = \frac{B(z^{-1})}{A(z^{-1})} = \frac{b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \qquad (6.2)$$

där koefficienterna i polynomen ges av

$$\left.\begin{aligned} b_1 &= 1 - \alpha\left[\beta + \frac{\xi_0\omega_0}{\omega}\gamma\right] & \omega &= \sqrt{1-\xi_0^2}\,\omega_0 \\ b_2 &= \alpha^2 + \alpha\left[\frac{\xi_0\omega_0}{\omega}\gamma - \beta\right] & \alpha &= e^{-\xi_0\omega_0 h} \\ a_1 &= -2\alpha\beta & \beta &= \cos(\omega h) \\ a_2 &= \alpha^2 & \gamma &= \sin(\omega h) \end{aligned}\right\} \qquad (6.3)$$

Det slutna systemet önskas två gånger snabbare än det öppna systemet. För att veta när detta önskemål är uppfyllt genomförs simuleringen för olika värden på parametern ϱ i förlustfunktionen (2.1). Utifrån det av regulatorn beräknade polynomet $P(z^{-1})$ beräknas det slutna systemets karakteristiska polynom i kontinuerlig form p(s).

$$P(z^{-1}) = p_0 + p_1 z^{-1} + p_2 z^{-2} = p_1 [\, 1 + a_{1c} z^{-1} + a_{2c} z^{-2}] \quad (6.4)$$

$$p(s) = s^2 + 2\xi_c \omega_c s + \omega_c^2 \quad (6.5)$$

Koefficienterna i polynomet p(s) ges av

$$\left.\begin{aligned} \omega_c &= \frac{\left[\left[\arccos\left[\frac{a_{1c}}{-2\sqrt{a_{2c}}}\right]\right]^2 + \left[\ln \sqrt{a_{2c}}\right]^2\right]^{1/2}}{h} \\ \xi_c &= -\frac{\ln \sqrt{a_{2c}}}{\omega_c h} \end{aligned}\right\} \quad (6.6)$$

Experimentet utförs med två processer. En långsam med $\omega_0 = 0.1$ och en snabbare med $\omega_0 = 1.0$. I båda fallen är relativa dämpningen $\xi_0 = 0.2$. Figur (6.1) och (6.2) visar resultaten av experimenten med na = nb = 2, nc = 0 och $\lambda = 0.98$. I tabell (6.1) visas resultaten av regulatorns identifiering av processerna och det slutna systemets karakteristiska polynom $P(z^{-1})$. Motsvarande värden på parametrarna ω_c och ξ_c, beräknade med ekvation (6.6), redovisas också. Som framgår av tabellen har regulatorn svårigheter vid identifiering av den snabba processen. För korta samplinsintervall klarar den inte att uppskatta koefficienterna i polynomen $A(z^{-1})$ eller $B(z^{-1})$. För längre samplinsintervall uppskattar den polynomet $A(z^{-1})$ tillfredställande men har fortfarande svårigheter att uppskatta polynomet $B(z^{-1})$ riktigt. För korta samplingsintervall överensstämmer dessutom inte det slutna systemets stegsvar med karakteristiska polynomet p(s). En förklaring till resultatet är att tidsfördröjnongen i regulatorn ej kan försummas vid korta samplingsintervall. För längre samplinsintervall, vid simulering av den långsammare processen, har tidsfördröjningen inte någon större betydelse. Som

framgår av tabellen erhålls det önskade uppförandet hos det slutna systemet då parametern $\varrho = 0.05$. En subjektiv bedömning av stegsvaret för det långsamma systemet ger också en fingervisning om att samplingsintervallet kan väljas enligt tumregeln $\omega_c h = 0.5 \ldots 1.0$.

För att enkelt bestämma storleksordningen hos tidsfördröjningen utförs ett experiment med dubbelintegratorn. Denna process väljs då det är enkelt att härleda dess pulsöverföringsfunktion. Pulsöverföringsfunktionen för dubbelintegratorn med samplingsintervall h och tidsfördröjningen τ är

$$H(z) = \frac{b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3}}{1 - 2z^{-1} + z^{-2}} \qquad (6.7)$$

där

$$\left.\begin{aligned} b_1 &= \frac{(h-\tau)^2}{2} \\ b_2 &= \frac{h^2}{2} + h\tau - \tau^2 \\ b_3 &= \frac{\tau^2}{2} \end{aligned}\right\} \qquad (6.8)$$

Genom att bestämma koefficienterna b_1, b_2 och b_3 beräknas tidsfördröjningen τ med (6.8). Erhållna koefficienter vid simulering med h = 1, NI = 3 och NI = 1 samt beräknad tidsfördröjning τ visas i tabellen nedan.

NI	b_1	b_2	b_3	τ (s)
3	0.16	0.73	0.09	0.41
1	0.24	0.70	0.04	0.29

Tabellen visar att tidsfördröjningen är betydande vid användning av korta samplingsintervall.

ϱ	a_1	a_2	b_1	b_2	p_0	p_1	p_2	ω_c	ξ_c
h=1.0									
	-0.91	0.67	0.41	0.35	(teoretiska värden)				
1.0	-0.96	0.70	0.27	0.46	1.32	-0.80	0.52	1.16	0.40
0.1					0.71	-0.05	0.10	1.77	0.55
0.05					0.63	0.04	0.06	2.05	0.57
h=0.7									
	-1.35	0.76	0.21	0.20					
1.0	-1.39	0.70	0.09	0.29	1.22	-1.34	0.65	1.12	0.40
0.1	-1.42	0.82			0.58	-0.33	0.14	1.70	0.60
0.05	-1.43	0.84			0.48	-0.19	0.09	1.97	0.61
h=0.5									
	-1.60	0.82	0.11	0.11					
1.0	-1.68	0.90	0.04	0.16	1.18	-1.65	0.77	1.13	0.38
0.1	-1.74	0.99	0.03	0.17	0.51	-0.49	0.21	1.70	0.52
0.05					0.40	-0.32	0.13	1.94	0.58

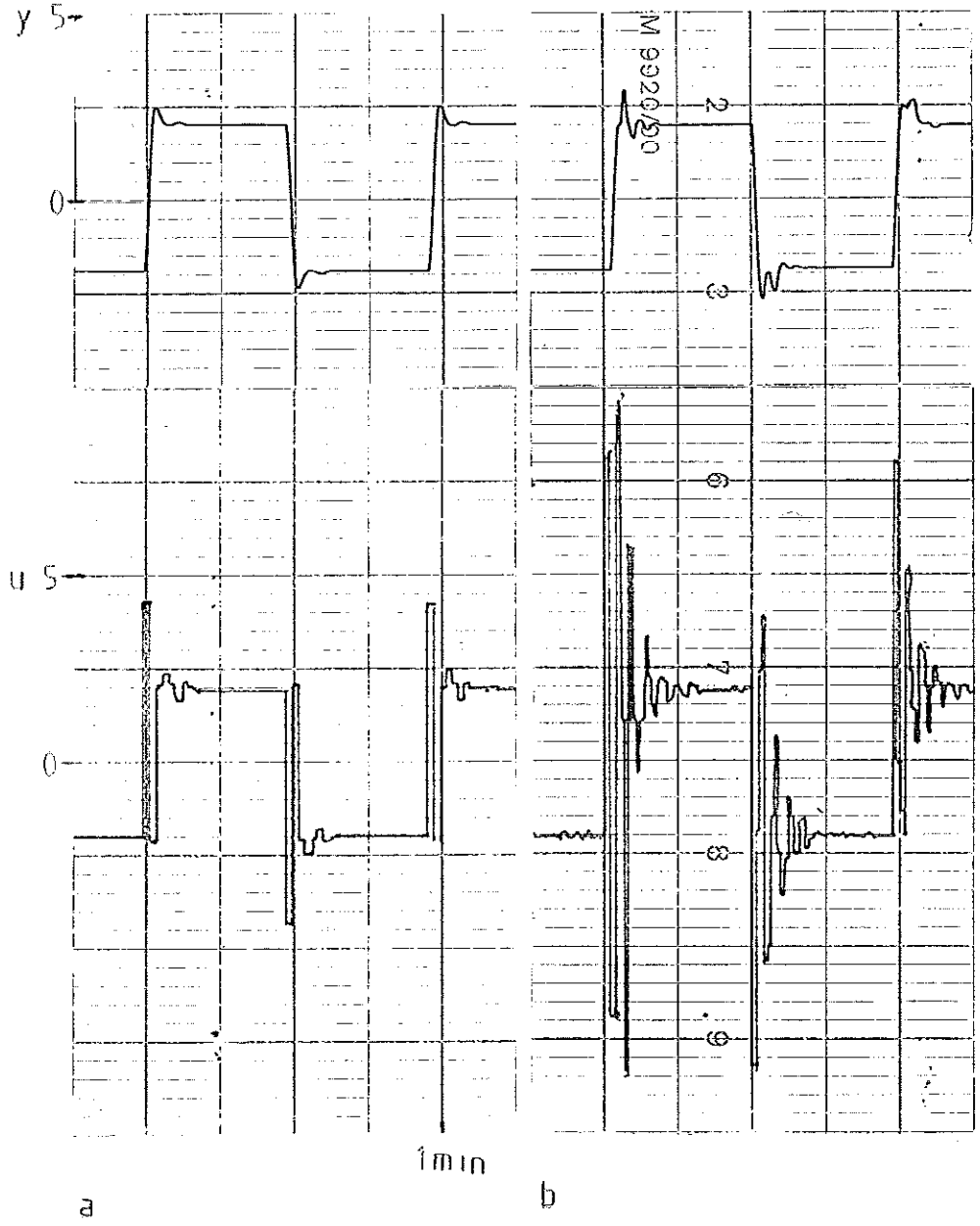
Tabell(6.1a). Teoretiska och uppskattade koefficienter för polynomen $A(z^{-1})$ och $B(z^{-1})$, beräknade koefficienter för polynomen $P(z^{-1})$ och p(s). ω_0=1.0, ξ_0=0.2 .

ϱ	a_1	a_2	b_1	b_2	p_0	p_1	p_2	ω_c	ξ_c
h=8.0									
	-1.21	0.73	0.27	0.25					
1.0	-1.20	0.73	0.26	0.26	1.25	-1.10	0.58	0.12	0.40
0.1					0.62	-0.20	0.12	0.18	0.57
0.05					0.52	-0.06	0.07	0.22	0.58
h=5.0									
	-1.60	0.82	0.11	0.11					
1.0	-1.60	0.82	0.10	0.11	1.16	-1.55	0.71	0.12	0.41
0.1					0.49	-0.42	0.17	0.18	0.57
0.05					0.39	-0.27	0.11	0.21	0.59
h=4.0									
	-1.71	0.85	0.07	0.07					
1.0	-1.71	0.86	0.07	0.08	1.12	-1.68	0.76	0.12	0.41
0.1					0.45	-0.49	0.19	0.18	0.60
0.05					0.35	-0.32	0.13	0.22	0.57

Tabell(6.1b). Teoretiska och uppskattade koefficienter för polynomen $A(z^{-1})$ och $B(z^{-1})$, beräknade koefficienter för polynomen $P(z^{-1})$ och p(s). $\omega_0=0.1$, $\xi_0=0.2$.

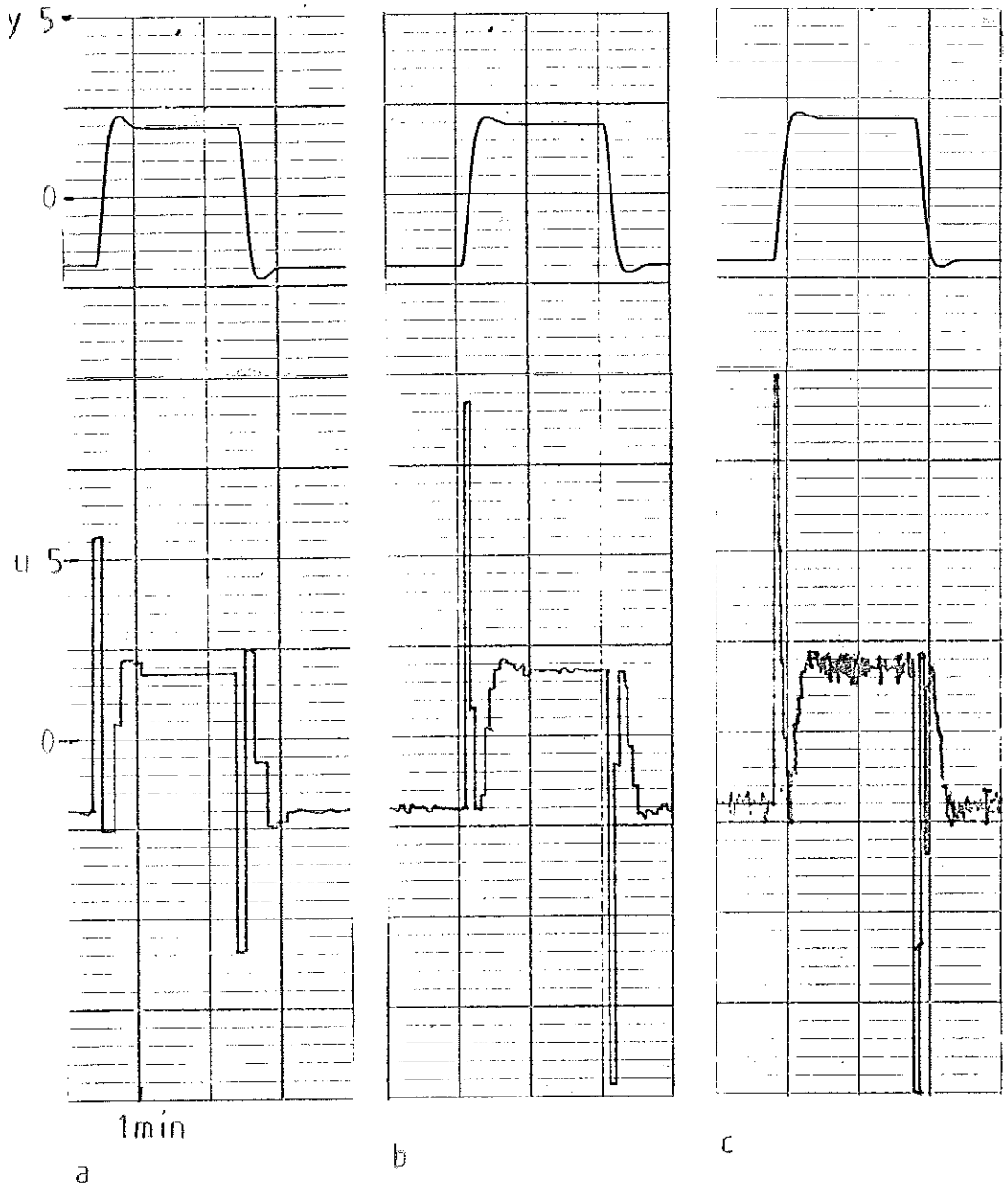
ϱ	a_1	a_2	b_1	b_2	p_0	p_1	p_2	ω_c	ξ_c
h=3.0									
	-1.80	0.89	0.04	0.04					
1.0	-1.80	0.88	0.04	0.05	1.09	-1.78	0.81	0.12	0.41
0.1					0.41	-0.54	0.22	0.18	0.57
0.05					0.31	-0.37	0.14	0.21	0.64
h=1.5									
	-1.92	0.94	0.01	0.01					
1.0	-1.91	0.93	0.01	0.01	1.05	-1.91	0.89	0.12	0.47
0.1					0.36	-0.60	0.26	0.17	0.64
0.05					0.27	-0.42	0.18	0.25	0.55

Tabell(6.1c). Teoretiska och uppskattade koefficienter för polynomen $A(z^{-1})$ och $B(z^{-1})$, beräknade koefficienter för polynomen $P(z^{-1})$ och p(s). $\omega_0=0.1$, $\xi_0=0.2$.



Figur(6.1) Stegsvar för process (6.1) med ω_0=1.0, ξ_0=0.2,

ϱ=0.05.
a. h=1.0 s
b. h=0.5 s



Figur(6.2) Stegsvar för process (6.1) med ω_0=0.1, ξ_0=0.2,

ϱ=0.05.
a. h=8 s
b. h=4 s
c. h=1.5 s

6.2 Inverkan av församplingsfilter

I detta experiment studeras hur ett församplingsfilter inverkar på det slutna systemet process-regulator. Processen som regleras beskrivs av (6.1) med $\omega_0 = 0.1$ och $\xi_0 = 0.2$. Det församplingsfilter som används är ett 2:a ordningens Butterworthfilter. Detta har överföringsfunktionen

$$G_f(s) = \frac{\omega_f^2}{s^2 + 2\xi_f \omega_f + \omega_f^2} \qquad (6.9)$$

där $\xi_f = 0.71$ och ω_f är brytfrekvensen.

Koefficienterna för polynomen $A(z^{-1})$ och $B(z^{-1})$ i processen beräknas enligt ekvation (6.3). Med h = 4 s blir dessa

$$A(z^{-1}) = 1 - 0.706\ z^{-1} + 0.852\ z^{-2}$$

$$B(z^{-1}) = 0.075\ z^{-1} + 0.071\ z^{-2}$$

Genom modifiering av programmet kopplas estimatorn bort, och regulatorn initieras med de beräknade koefficienterna. Experimentet utförs med två olika värden på parametern ϱ. I första fallet är ϱ=1.0, vilket betyder att för det slutna systemet är $\omega_c = 0.12$ och $\xi_c = 0.41$. I andra fallet är ϱ= 0.05, och det slutna systemet är ungefär 2 gånger snabbare än det öppna har en relativ dämpning $\xi_c = 0.57$. Samplingsintervallet är i båda fallen h = 4 s, vilket ger Nyqvistfrekvensen $\omega_N = 0.785$ rad/s. Brytfrekvensen hos församplingsfiltret varieras från $\omega_b = \infty$, dvs inget filter, och närmar sig Nyqvistfrekvensen. Resultatet visas i figur (6.3) och (6.4). I figur (6.3) är $\varrho = 1.0$ och i (6.4) är $\varrho = 0.05$.
Då brytfrekvensen är stor har filtret ingen inverkan på det slutna systemet. En lägre brytfrekvens hos filtret betyder en större fasförskjutning hos den filtrerade signalen. Detta kan tolkas som en tidsfördröjning i filtret. Figurerna visar att då denna tidsfördröjning

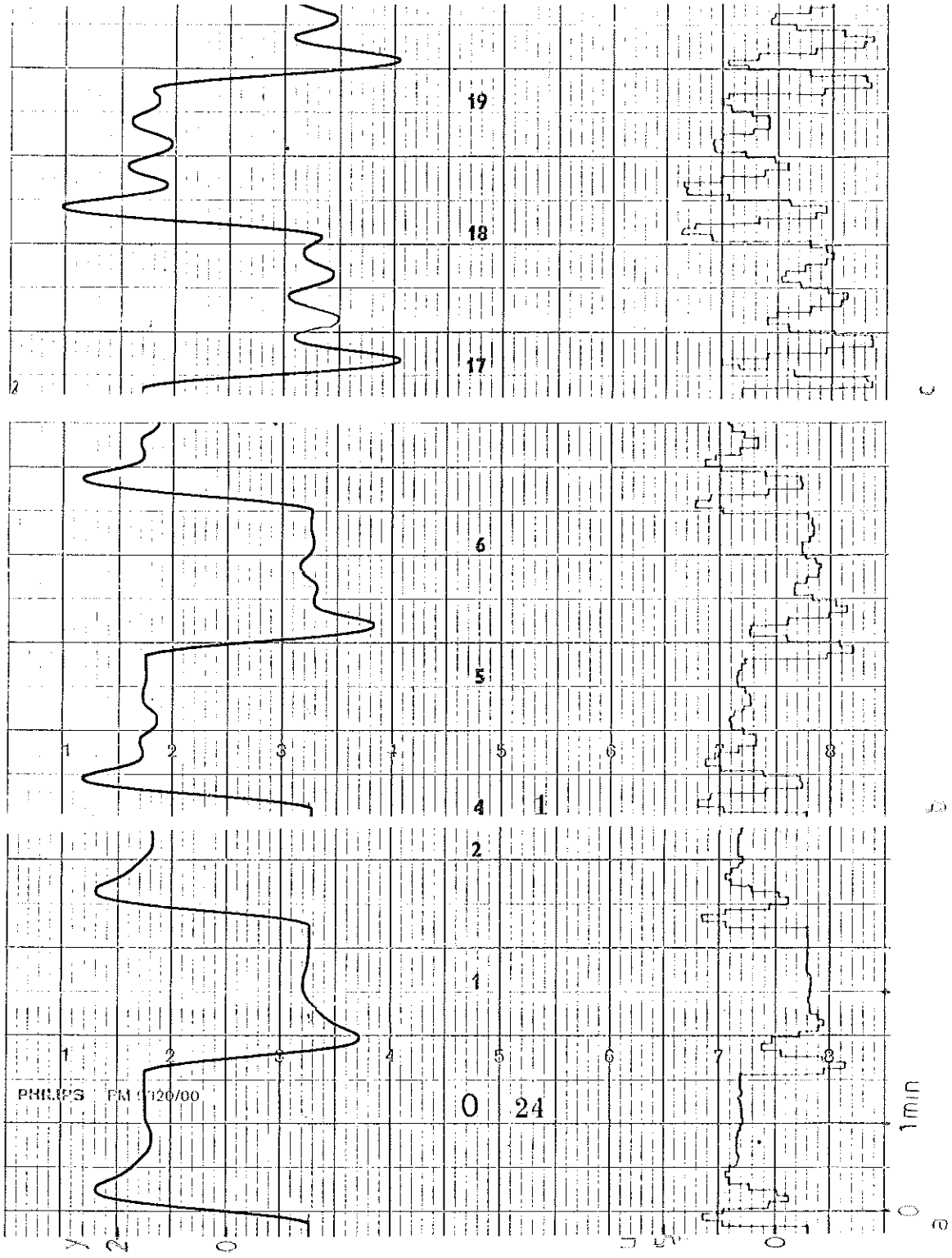
blir alltför stor kan regulatorn ej reglera processen tillfredsställande. Effekten är större då ϱ = 0.05 och för detta fall är systemet klart instabilt för ω_b = 1.0 .

Enligt tumregeln gäller att man kan välja samplingsintervallet så att $\omega_c h$ = 0.5...1.0.

Nyqvistfrekvensen ω_N = π/h. Detta medför att ω_c =(0.15...0.30)ω_N. Men för att slippa vikningseffekten måste $\omega_f \leq \omega_N$. Approximativt bör alltså gälla att

$$\omega_c \leq 0.2\omega_f \quad (6.10)$$

vilket stämmer bra med experimenten ovan.
Problemet vid ett snabbt slutet system kvarstår dock. Detta elimineras genom att vid konstuktion av regulatorn även ta hänsyn till filtrets dynamik, vilket görs genom att öka gradtalen för polynomen $A(z^{-1})$ och $B(z^{-1})$. Detta visas också tydligt i figur (6.5) där regulatorn används som självinställare. I figur (6.5a) är na = nb = 3 och i figur (6.5b) är na = nb = 4 .
Detta experiment visar att hänsyn bör tas till församplingsfiltret dynamik vid bestämning av regulatorparametrar, åtminstone då villkoret (6.10) ej är uppfyllt.

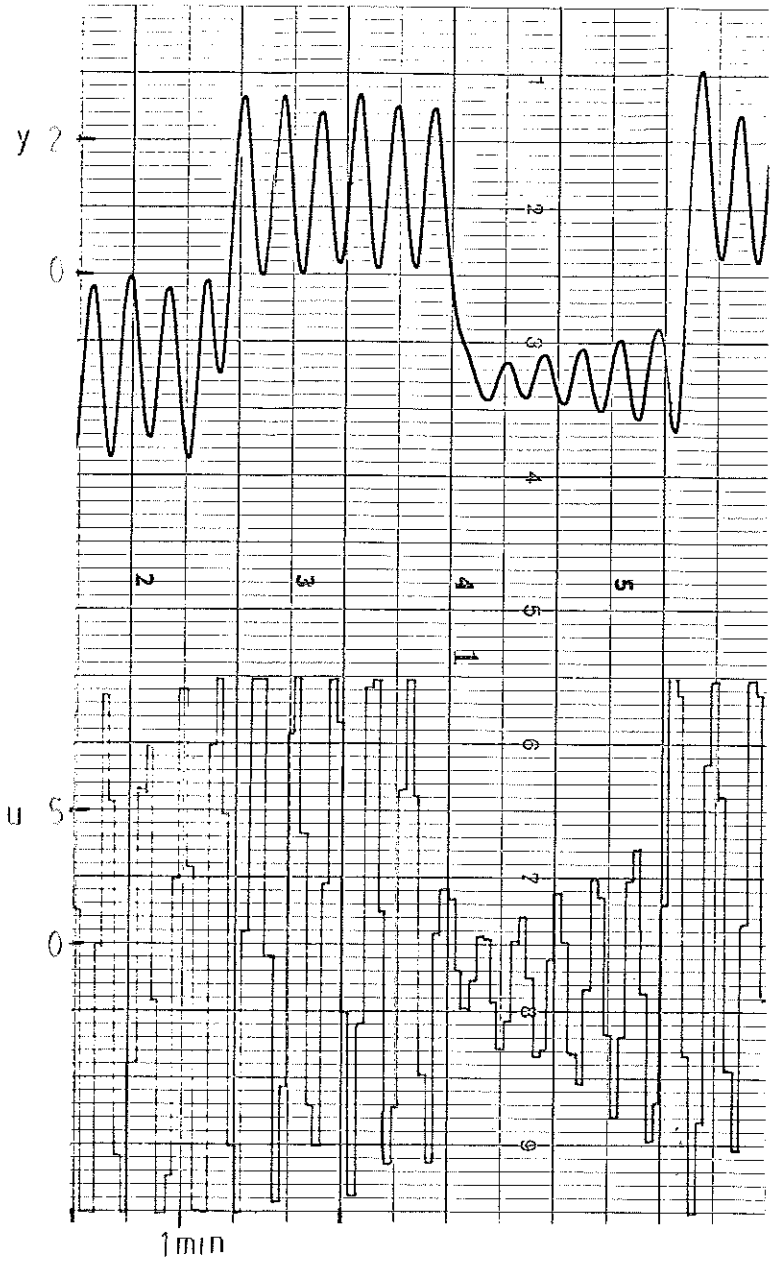


Figur(6.3) Stegsvar för process (6.1) med ω_0=0.1,ξ_0=0.2, ϱ=1.0. A och B fixerade. Församplingsfilter med brytfrekvens ω_f. ω_N = 0.785.

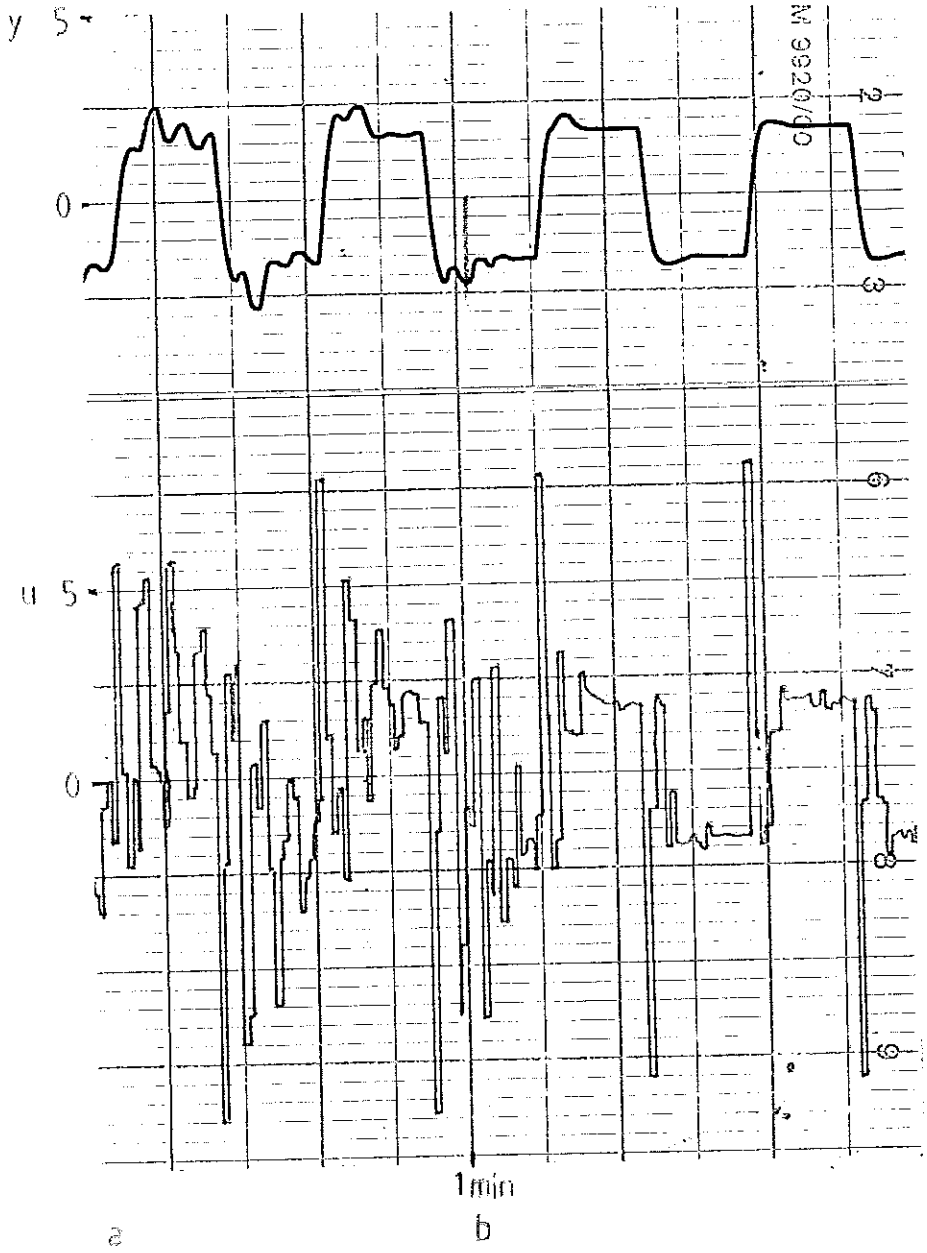
a. ω_f = 0.8

b. ω_f = 0.6

c. ω_f = 0.5



Figur(6.4) Stegsvar för process (6.1) med ω_0=0.1,ξ_0=0.2,
,ϱ=0.05. A och B fixerade. Församplingsfilter
med brytfrekvens ω_f = 1.0. ω_N = 0.785.

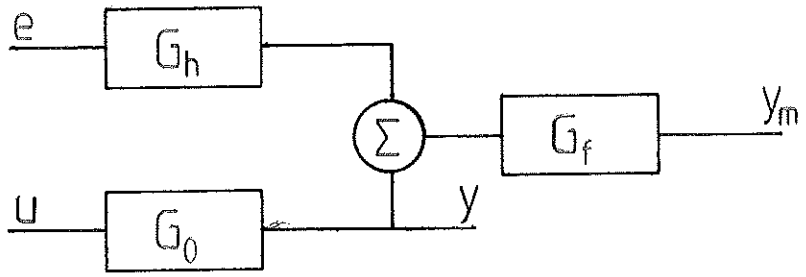


Figur(6.5) Stegsvar för process (6.1) med ω_0=0.1,ξ_0=0.2,
,ϱ=0.05. Församplingsfilter med brytfrekvens
ω_f=0.785. ω_N = 0.785.

a. na=nb=3
b. na=nb=4

6.3 Inverkan av brus hos mätsignal

I detta experiment undersöks hur regulatorn fungerar då mätsignalen störs av brus. Mätbruset består av vitt brus som filtrerats i ett högpassfilter av 1:a ordningen. Den brusiga mätsignalen filtreras i ett församplingsfilter. Både 1:a och 2:a ordningens filter används. Processen som regleras är ett 2:a ordningens system. Figur (6.6) visar en schematisk bild av systemet.



Figur(6.6) Modell för systemet i exp. 6.3.

Överföringsfunktionerna för högpassfiltret är

$$G_h(s) = \frac{s}{s + b}$$

Överföringsfunktionen för församplingsfiltret av första ordningen är

$$G_1(s) = \frac{\omega_f}{s + \omega_f}$$

Församplingsfiltret av 2:a ordningen är ett Butterworthfilter (se (6.9)).

Processen som regleras beskrivs av (6.1) med $\omega_0 = 0.1$ och $\xi_0 = 0.2$.

I experimentet varieras både högpassfiltrets och församplingsfiltrets brytfrekvenser, och därmed brusets inverkan på den filtrerade mätsignalen. Figur (6.7) och (6.8) visar resultaten av experiment med np = 2, ϱ = 0.05 och h = 4 s. Nyqvistfrekvensen är därmed $\omega_N = 0.785$ rad/s. I figur (6.7) är filtret av 1:a ordningen och i figur (6.8) är det av 2:a ordningen. I figur (6.7) är na = nb = 3 och i figur (6.8) är na = nb = 4. Som framgår av figurerna ökar brusets påverkan på utsignalen då högpassfiltrets brytfrekvens närmar sig antialiasfiltrets. Regleringen blir också sämre då brus med frekvens över Nyqvistfrekvensen ej filtreras bort från mätsignalen, vilket beror på vikningseffekten. I vissa av figurerna uppträder plötsliga förändringar i utsignalens amplitud. Detta beror på att de estimerade polynomen $A(z^{-1})$ och $B(z^{-1})$ innehåller faktorer som är nästan lika. Vad som då inträffar vid beräkning av regulatorparametrarna visas i följande exempel från simuleringen av figur (6.8 d).

Exempel på parametrar innan 'spiken'

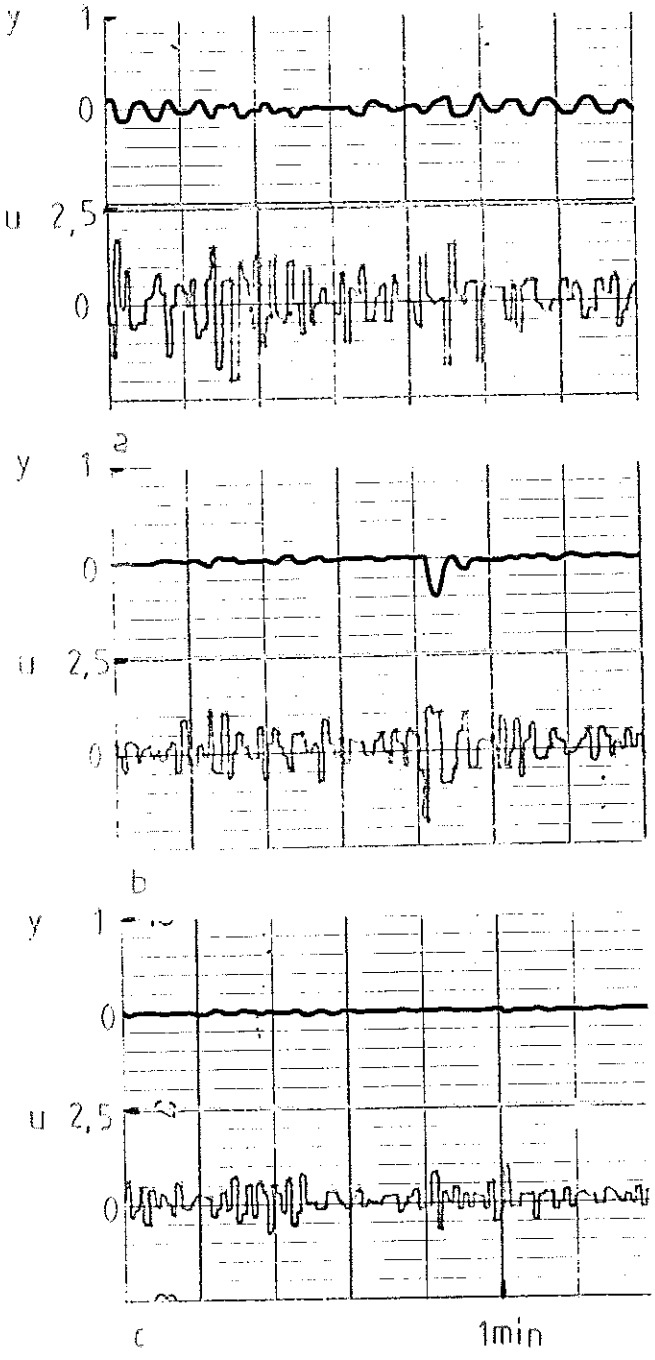
	z^0	z^{-1}	z^{-2}	z^{-3}
$A(z^{-1})$	1	-0.6979	-0.4489	0.6361
$B(z^{-1})$	0	0.0522	0.1264	0.0808
$P(z^{-1})$	0.3268	-0.0236	-0.1155	
$R(z^{-1})$	0.3248	0.0929	-0.1578	-0.2599
$S(z^{-1})$	1.9944	0.0755	-3.4038	2.0464

Exempel på parametrar i 'spiken'

$A(z^{-1})$	1	-0.4726	-0.3775	0.6396
$B(z^{-1})$	0	0.0462	0.1171	0.0653
$P(z^{-1})$	0.2950	-0.0149	-0.0991	
$R(z^{-1})$	0.2840	-0.3654	-0.4551	0.5365
$S(z^{-1})$	10.9736	-21.9137	16.9830	-5.2546

Små förändringar i processparametrar ger här upphov till mycket stora förändringar av regulatorparametrar. Detta visar sig också i styrsignalen som blir väldigt stor och ibland bottnar.

När $A(z^{-1})$ och $B(z^{-1})$ innehåller en gemensam faktor skall denna förkortas bort innan beräkning av $R(z^{-1})$ och $S(z^{-1})$. I algoritmen som beräknar regulatorparametrarna finns en konstant ε_1 som bestämmer hur 'nära' faktorer i $A(z^{-1})$ och $B(z^{-1})$ får vara innan förkortning sker. I dessa simuleringar är denna konstant $\varepsilon_1 = 0.0001$. I figur (6.9) visas en simulering med $\varepsilon_1 = 0.01$ men detta tycks inte eliminera problemet.



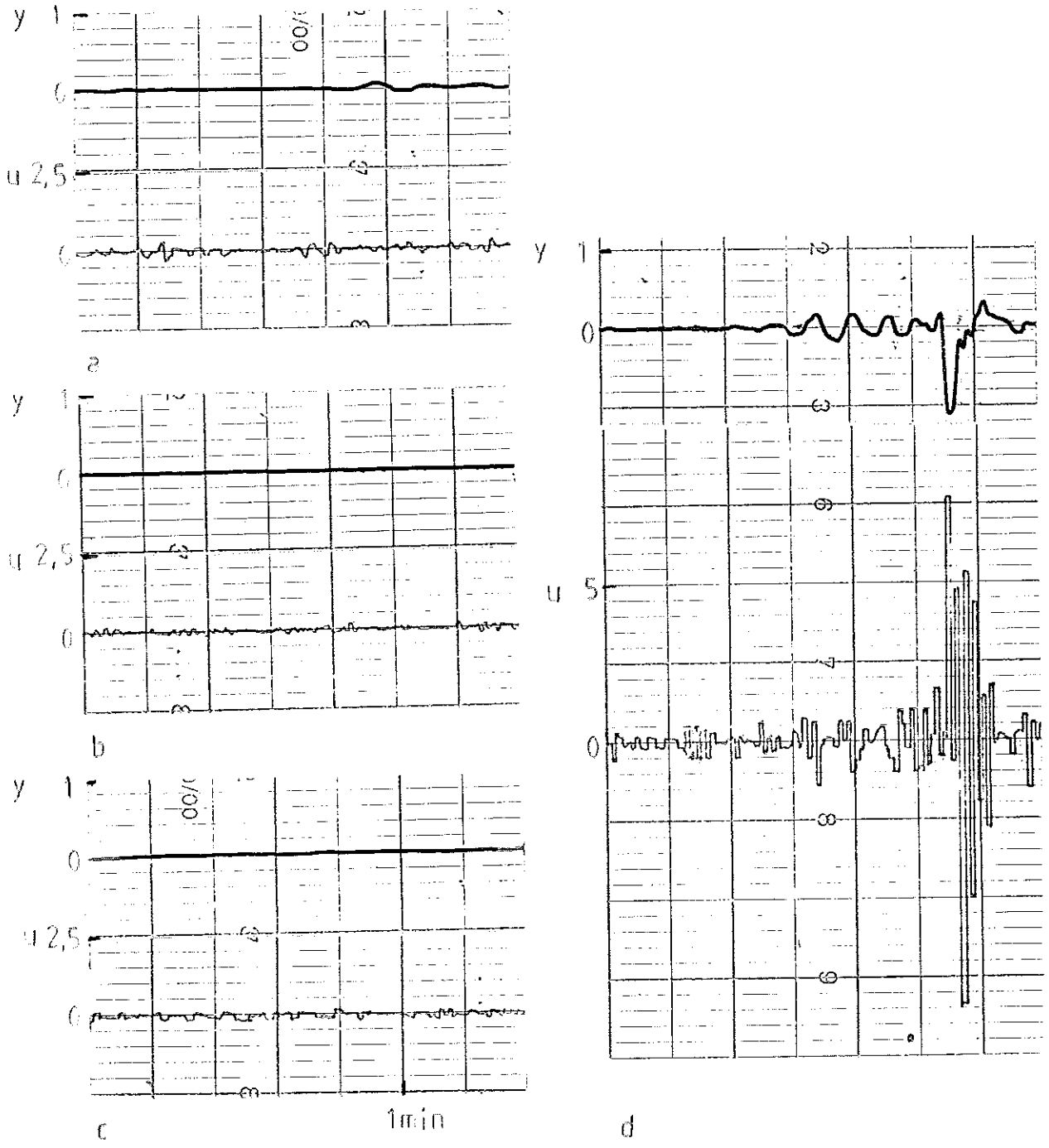
Figur(6.7) Experiment med mätbrus. i:a ordningens filter med brytfrekvens ω_f. Högpassfilter med brytfrekvens b.

ω_N=0.785. ε_i=0.0001.

a. ω_f=1.57 b=5

b. ω_f=0.785 b=5

c. ω_f=0.393 b=5



Figur(6.8) Experiment med mätbrus. 2:a ordningens filter med brytfrekvens ω_f. Högpassfilter med brytfrekvens

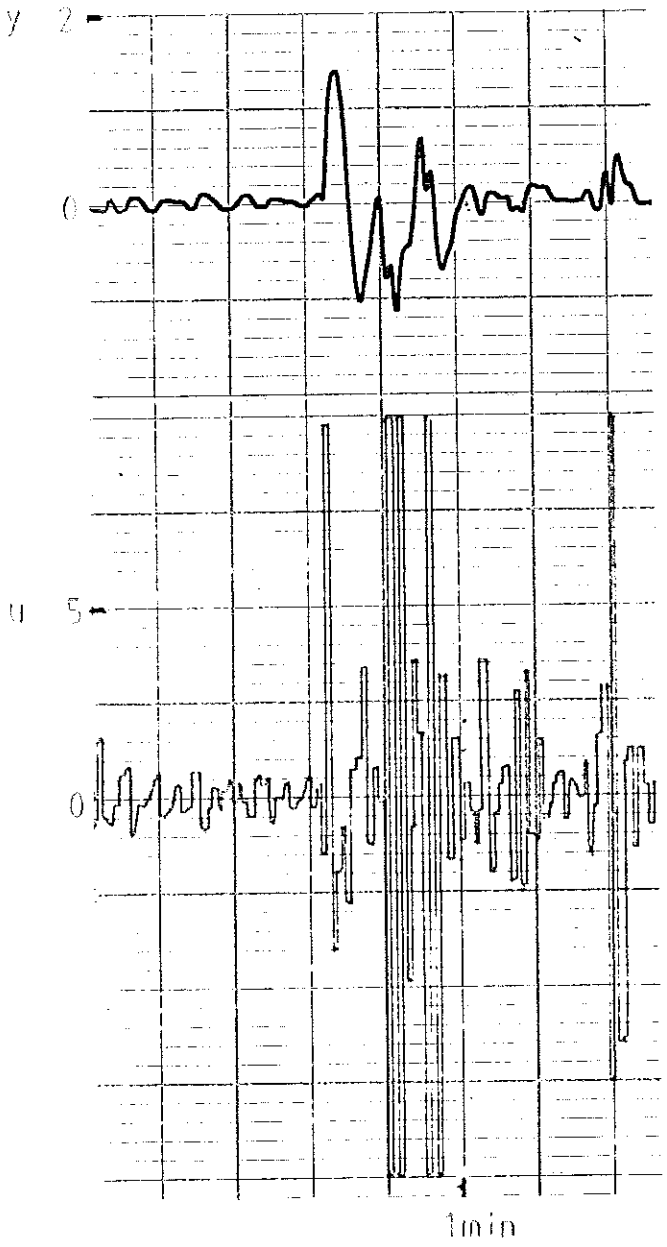
b. $\omega_N = 0.785 . \varepsilon_1 = 0.0001$.

a. $\omega_f = 1.57$ b=5

b. $\omega_f = 0.785$ b=5

c. $\omega_f = 0.393$ b=5

d. $\omega_f = 0.785$ b=0.2



Figur(6.9) Experiment med mätbrus. 2:a ordningens filter med brytfrekvens ω_f=1.57. Högpassfilter med brytfrekvens b=1.0. ε_1 = 0.01.

6.4 Omodellerad dynamikstörning

I detta experiment studeras hur regulatorn förmår att reglera en ganska enkel process då dynamikstörning komplicerar processens uppförande. Den enkla process som ska regleras är av första ordningen och beskrivs av överföringsfunktionen

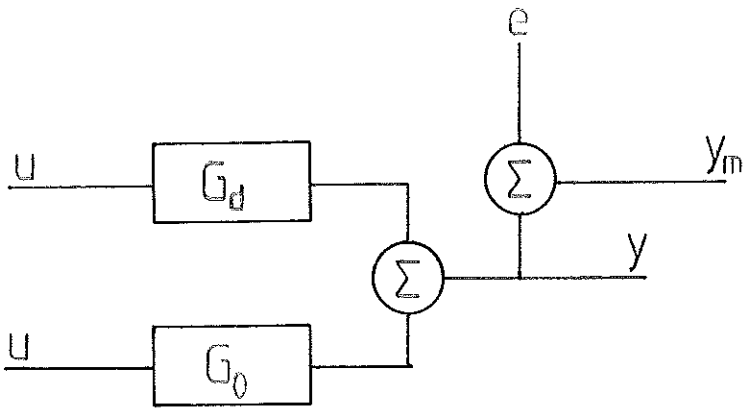
$$G_0(s) = \frac{3}{s+3}$$

Dynamikstörningen beskrivs av överföringsfunktionen

$$G_d(s) = \frac{101}{s^2 + 20\,s + 101}$$

Det önskade slutna systemet beskrivs av

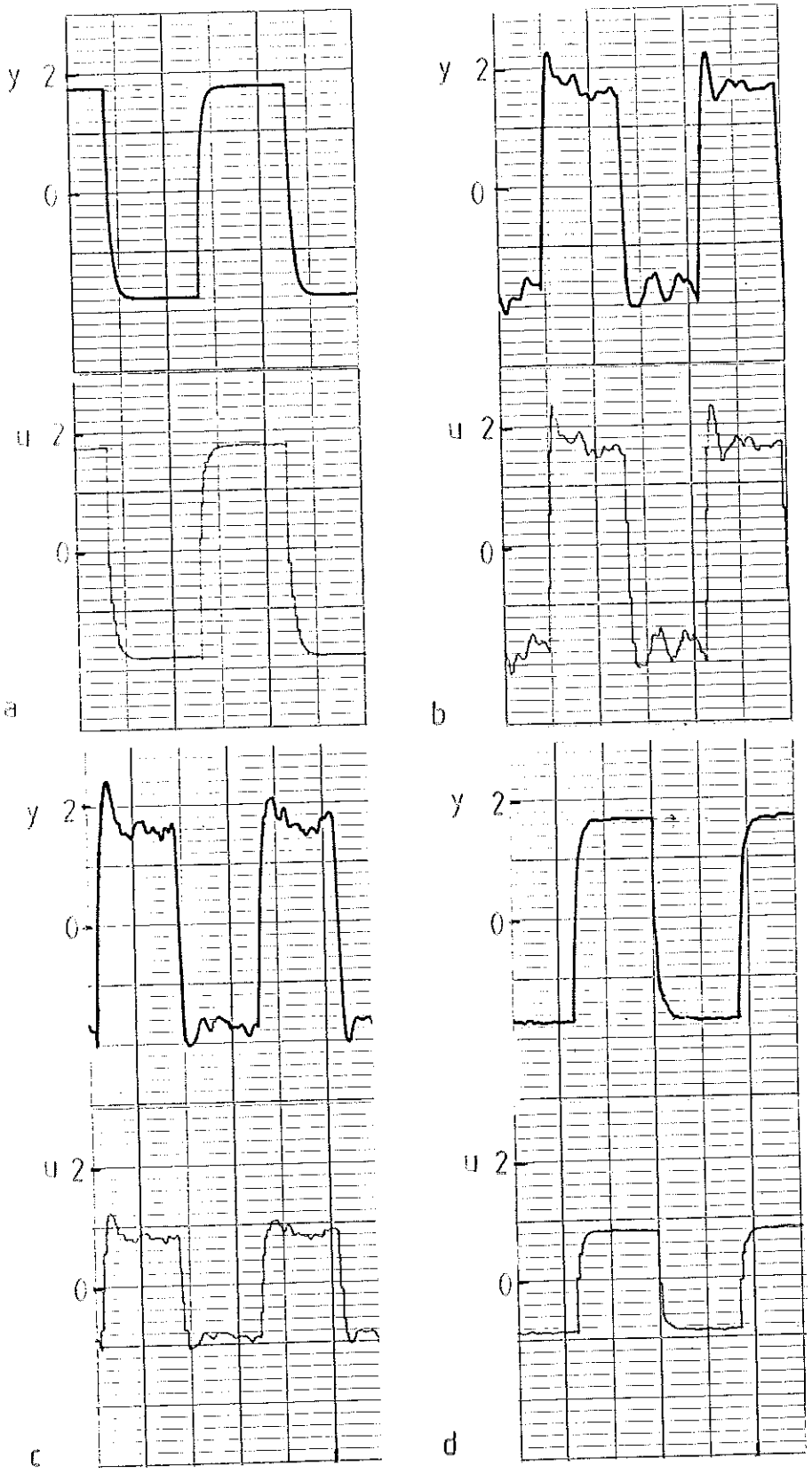
$$G_c(s) = \frac{1}{s+1}$$



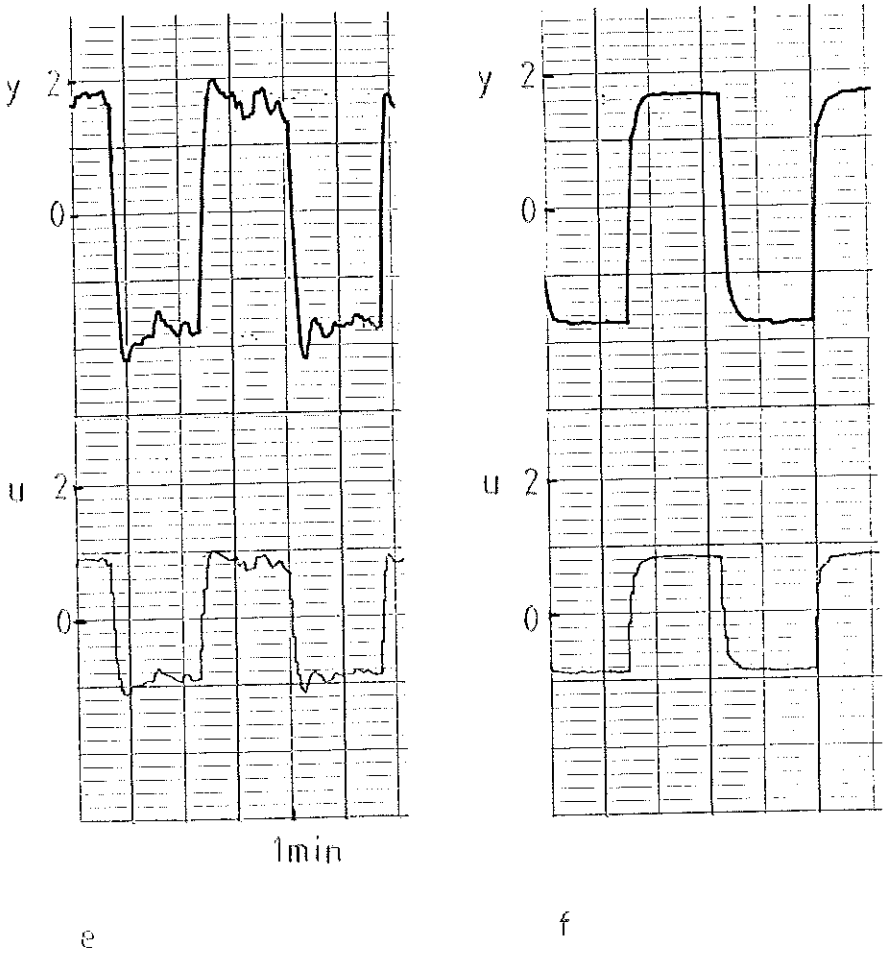
Figur(6.10) Modell för system i exp. 6.4.

Figur (6.10) visar en graf över hur dynamikstörningen påverkar processen. Det slutna systemets karakteristiska polynom, i samplad form, är $P(z^{-1}) = 1 - 0.607\,z^{-1}$. Det

slutna systemet erhålls genom att i regulatorn använda polplacering. Genom att mätbrus tillförs systemet ska dynamikstörningen fås att exciteras och därmed störa det slutna systemet. I experimentet, som utförs med vitt brus som mätbrus, är na = nb = 1 då inget församplingsfilter används och na = nb = 2 när mätsignalen filtreras. Detta i överensstämmelse med resultatet i kapitel (6.2). Filtrets,som är av 1:a ordningen, har en brytfrekvens $\omega_f = \pi$, d v s halva Nyqvistfrekvensen. Övriga regulatorparametrar är $\lambda = 0.98$,h = 0.5 s. Resultaten visas i figur (6.11). Denna visar att regulatorn klarar att reglera processen tillfredsställande då mätsignalen filtreras genom ett antialiasfilter av 1:a ordningen. Även när dynamikstörningens relativa dämpning halveras förmår regulatorn reglera processen bra. Försök görs också med process och dynamikstörning i serie (multiplikativt) men detta ger ingen försämring vid regleringen.



Figur(6.12a) Experiment med omodellerad dynamikstörning.
a. utan störning
b. mätbrus
c. mätbrus + dynamikstörning
d. mätbrus + dynamikstörning + filter



Figur(6.12b) Experiment med omodellerad dynamikstörning.
e. mätbrus + dynamikstörning(halv dämpn.)
f. mätbrus + dynamikstörning(halv dämpn.)+filter

7. SAMMANFATTNING

De viktigaste resultaten av experimenten i denna rapport är att om regulatorn används med församplingsfilter bör filtrets dynamik finnas med i den modell som används för att beskriva den reglerade processen. Dessutom bör hänsyn tagas till den tidsfördröjning som finns i regulatorn, speciellt vid användning av korta samplingsintervall.
En modifiering av programmet bör göras så att denna tidsfördröjning minskas om regulatorn ska användas till annat än simulering. Detta är ej gjort i den version som presenteras i denna rapport.
Ett annat problem som förekommer i detta arbete är vad som inträffar när det finns en gemensam faktor i A och B. Problemet som tycks vara ett numeriskt problem bör vara intressant att studera och försöka finna en lösning på.
En stor fördel med algoritmen i denna rapport är att den är lätt att starta och få att fungera med bra resultat, vilket gör den bekväm att använda vid simuleringar.

REFERENSER

[1]. Zhou Z.Y, Aström K.J: A microcomputer implementation of LQG-selftuner Report. CODEN: LUTFD2/(TFRT-7226)/1-052/(1981)

[2]. Zhou Z.Y, Aström K.J: Regulator synthesis based on polynom manipulation Report. CODEN: LUTFD2/(TFRT-7225)/1-044/(1981)

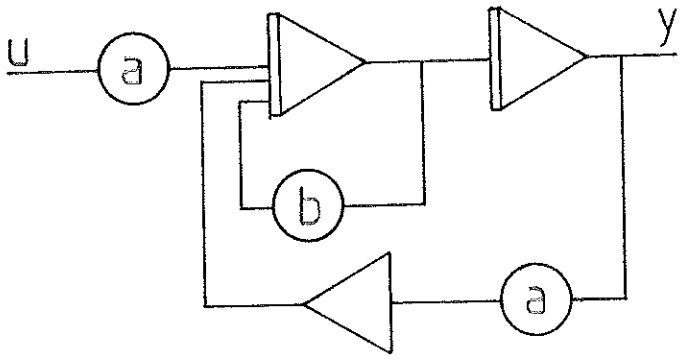
[3]. Aström K.J, Wittenmark B: Computer Control Theory

[4]. Reglerteknik AK: Studiematerial

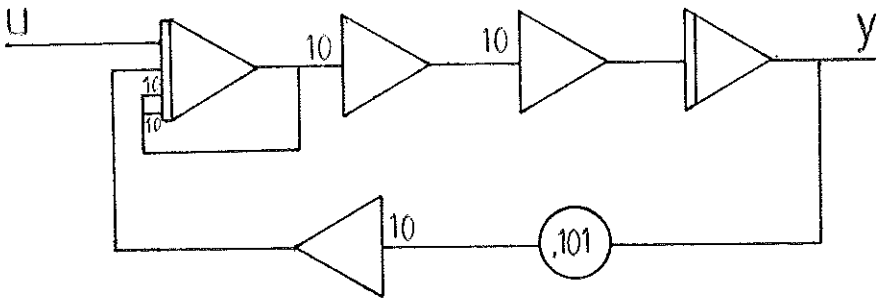
Appendix A

I detta appendix visas hur processer i denna rapport realiserats på analogimaskinen.

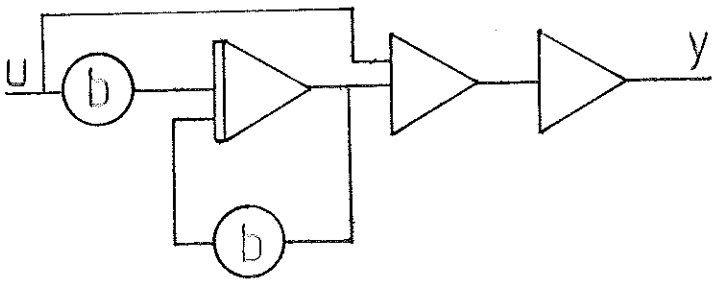
$$G(s) = \frac{\omega_0^2}{s^2 + 2\omega_0\xi_0 s + \omega_0^2} \quad , a = \omega_0^2 \quad , b = 2\omega_0\xi_0$$



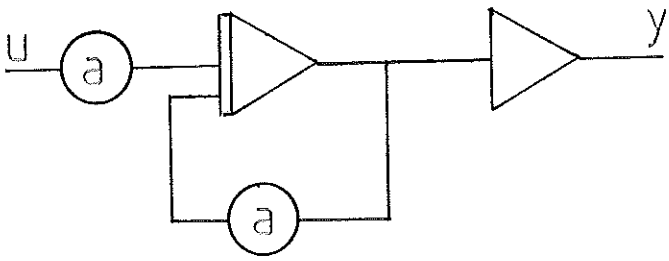
$$G(s) = \frac{101}{s^2 + 20s + 101} \approx \frac{100}{s^2 + 20s + 101}$$



$$G(s) = \frac{s}{s + b}$$



$$G(s) = \frac{a}{s + a}$$



Appendix B

```
PROGRAM SELFTUNER;

   { Author   Zhou Z.Y.
     Data     May. 1981.
     References
       Zhou Z.Y.(1980):A Microcomputer Implementation of
         Datalogging and Recursive Least Squares Estimation.
       Zhou Z.Y.(1981):Recursive Formulae of Complex Integral
         I(l) Used in Polynomial Factorization Problem.
       Zhou Z.Y.(1981):Regulator Synthesis by Use of polynomial
         equation.
       Wittenmark B.,Hagander P.,and Gustavsson I.(1980):
         STUPID Implementation of a Self-tuning PID-controller.
         Regulator. Report LUTFD2/(TFRT-7154)/1-034/(1978).
       Mattson S.E.(1978):A Simple Real-time Schedurer.
         Report CODEN:LUTFD2/(TFRT-T156)/1-010/(1978).
       Kucera V.(1979): Discrete Linear Control. Academia
         Prague.
         Sequential Estimation. Academic Press. New York.

     The program  performs on-line  recursive least squares
     (RLS)  or  square  root  (SR)  estimation  , spectral
     factorization (SF) and regulator design (RD) for SISO
     system .     All  results  can  be   stored  in  files
     if respective logging command is  given.  The  control
     algorithm  based  on pole-placement or linear quadratic
     control.

     The program consists of these files:
     FG    -- foreground program.
     OPCOM -- operator communication and main program.
     RD    -- regulator design which contains
              UD - estimator,
              SF - spectral factorization, and
              RC - redulator calculation.                  }

{ GLOBAL DATA DECLARATIONS }

const n=12;
      m= 5;
      l=10;

type  specificationtype=record
        tsamp,lambda,po,py,eps,lo,lolim,hilim,aa,ab,ac,ad,
              ba,bb,bc,bd,pz,pa,pb,pc,pd,pe:real;
        na,nb,nd,ni,np,ns,nt,nta,ntb,yrchan,ychan,uchan,
              outchan,delayb:integer;
        log,minr,inita,initb,initp:boolean;
        end;
      polytype=record
        z:array[0..n] of real;
        d:integer;
        end;
var   specif,compar:specificationtype;
```

```
      a,b,c,p,r,s,t,xm,sold,rold:polytype;
      k,nm,nab,q,w,period,iyr,sumorder:integer;
      yr,y,u,fi:array[0..n] of real;
      loga,logb,loguy,logt,logr,logs:array[0..m,1..l] of real;
      pp:array[0..n,0..n] of real;
      yfold,ufold,ui,ub,ey,t0:real;
      newpar:boolean;
      delu:array[1..100] of real;

{ EXTERNAL PROCEDURE DECLARATION }

Function  adin(chan:integer):real;external;
Function  rform(r:real; size:integer):integer;external;
Procedure daout(chan:integer; value:real);external;
Procedure schedule(procedure FG; var period:integer);external;
Procedure clksave;external;
Procedure clkrestore;external;

{ END OF GLOBAL }
```

```
Procedure estimation(y:real; var a,b:polytype);external;
Procedure specfac(a,b:polytype; var p:polytype);external;
Procedure regdesign(a,b:polytype; var r,s:polytype);external;

{=========================================================FOREGROUND}
Procedure foreground;

   { The procedure FG is organized as follows:
     Adin     inputs yr(t) and y(t);
     Control  compute control signal u(t) from measurements
              and regulator parameters of given process;
     Daout    outputs u(t);
     Estimation estimates the parameters of the process;
     Regdesign  calculates the regulator parameters ;
     Display display parameters of estimation and reglator
              on screen;
     Logdata prepares data for store in files after running;
     delayu  delays the output of control signal;
     Updata  updates input and output for next step.          }

var d,ai:polytype;
    i,j:integer;
    yf,uf:real;
{-------------------------------------------------------------------mul}
Procedure Mul(za,zb:polytype; var zc:polytype);
 var i,j:integer;
 begin
   zc.d:=za.d+zb.d;
   for i:=0 to zc.d do zc.z[i]:=0.0;
   for i:=0 to za.d do
       for j:=0 to zb.d do
           zc.z[i+j]:=zc.z[i+j]+za.z[i]*zb.z[j];
 end; { of mul }

{-----------------------------------------------------------------control}
Procedure control;
   { computes control signals u(t) from measurements ( yr
     and y ), regulator parameters  T(z) , R(z)  and S(z). }
 var ps,bs,ts,rs:real;
     i:integer;
 begin
   with specif do begin
     ui:=0;
     bs:=0;   for i:=delayb to b.d do bs:=bs+b.z[i];
     ps:=0;   for i:=0 to p.d do ps:=ps+p.z[i];
     t0:=abs(ps/bs);
     for i:=0 to t.d do ui:=ui+t.z[i]*yr[i];
     ui:=ui*t0;
     mul(r,d,r);
     for i:=0 to s.d do ui:=ui-s.z[i]*y[i];
     for i:=1 to r.d do ui:=ui-r.z[i]*u[i];
     ui:=ui/r.z[0];
     u[0]:=ui;
     if ui<lolim then u[0]:=lolim;
     if ui>hilim then u[0]:=hilim;
```

```
   end;
 end; { of control }

{-------------------------------------------------------display}
Procedure display;
   { displays parameters of the process and the regulator
     on screen }
 var bb,bbs,pp1,pp2,pp3:real;
     i:integer;
 begin
   with specif do begin
     if q=ns then begin
        writeln('point',k);
        write(' A(z) ');
        for i:=1 to a.d do begin
            write('    ');   write(a.z[i]:8:4);
        end;
        writeln;
        write(' B(z) ');
        for i:=1 to b.d do begin
            write('    ');   write(b.z[i]:8:4);
        end;
        writeln;
        write(' P(z) ');
        for i:=0 to p.d do begin
            write('    ');   write(p.z[i]:8:4);
        end;
        bb:=p.z[1]*p.z[1]-4*p.z[0]*p.z[2];
        bbs:=sqrt(abs(bb));
        if bb)0.0 then begin
           pp1:=(-p.z[1]+bbs)/(2*p.z[0]);
           pp2:=(-p.z[1]-bbs)/(2*p.z[0]);
           pp3:=0.0;
        end else begin
                   pp1:=-p.z[1]/(2*p.z[0]);
                   pp2:=pp1;
                   pp3:=bbs/(2*p.z[0]);
        end;
        writeln('    ',pp1:8:4,'    ',pp2:8:4,'    ',pp3:8:4);
        write(' T(z) ');
        for i:=0 to t.d do begin
            write('    ');   write(t.z[i]:8:4);
        end;
        writeln;
        write(' R(z) ');
        for i:=0 to r.d do begin
            write('    ');       write(r.z[i]:8:4);
        end;
        writeln;
        write(' S(z) ');
        for i:=0 to s.d do begin
            write('    ');       write(s.z[i]:8:4);
        end;
        writeln;
        write('yr y ui ub');    write(yr[0]:7:4);
```

```
        write('    ');        write(y[0]:8:4);
        write('    ');        write(u1:8:4);
        write('    ');        writeln(ub:8:4);
        q:=0
      end;
      q:=q+1;
    end;
 end; { of display }

{-------------------------------------------------------logdata}
Procedure logdata;
   { prepares data and parameters from k=nta to k=ntb for
     store in files after running }
 var i:integer;
 begin
   with specif do begin
     for i:=1 to a.d do loga[i,w]:=a.z[i];
     for i:=1 to b.d do logb[i,w]:=b.z[i];
     loguy[0,w]:=yr[0];       loguy[1,w]:=y[0];
     loguy[2,w]:=u1;          loguy[3,w]:=u[0];
     for i:=0 to t.d do logt[i,w]:=t.z[i];
     for i:=0 to r.d do logr[i,w]:=r.z[i];
     for i:=0 to s.d do logs[i,w]:=s.z[i];
     w:=w+1
   end
 end; { of logdata }

{-------------------------------------------------------updata}
Procedure updata;
   { updates input and output preparing for next step }
 var b1,b2,bs,nxm:real;
     i:integer;
 begin
   with specif do begin
     for i:=t.d downto 1 do yr[i]:=yr[i-1];
     for i:=a.d downto 1 do y[i]:=y[i-1];
     for i:=b.d downto 1 do u[i]:=u[i-1];
     for i:=sumorder downto 2 do fi[i]:=fi[i-1];
     fi[1]:=yfold-yf;
     fi[a.d+1]:=uf-ufold;
     fi[nab+1]:=ey;
     yfold:=yf;
     ufold:=uf;
   end
 end; { of update }

{-------------------------------------------------------delayu}
Procedure delayu;
   { delays output u for record. }
  var i:integer;
  begin
    with specif do begin
      for i:=nd downto 2 do
          delu[i]:=delu[i-1];
      delu[1]:=u[0];
```

```
      daout(0,delu[nd]);
    end;
  end; { of delayu }

{-------------------------------------------------------------initabp}
Procedure initabp;
  var i:integer;
  begin
    with specif do begin
      if na>nb then nm:=na else nm:=nb;
      if nm<np then nm:=np;
      a.d:=na;
      if inita then begin
         a.z[1]:=aa;      a.z[2]:=ab;
         a.z[3]:=ac;      a.z[4]:=ad;
         for i:=na+1 to nm do a.z[i]:=0.0;
         inita:=false;
      end;
      b.d:=nb;
      if initb then begin
         for i:=0 to delayb-1 do b.z[i]:=0.0;
         b.z[delayb]:=ba;      b.z[delayb+1]:=bb;
         b.z[delayb+2]:=bc;    b.z[delayb+3]:=bd;
         for i:=nb+1 to nm do b.z[i]:=0.0;
          initb:=false;
      end;
      p.d:=np;
      if initp then begin
         p.z[0]:=pz;           p.z[1]:=pa;
         p.z[2]:=pb;           p.z[3]:=pc;
         p.z[4]:=pd;           p.z[5]:=pe;
         for i:=np+1 to nm do p.z[i]:=0.0;
         initp:=false;
     end;
      t.d:=nt;
      sumorder:=na+nb+nt;
      nab:=na+nb;
      q:=1;                    w:=1;
    end;
  end; { of initabp }

{-----------------------------------------------CODE OF FOREGROUND}
{ CODE FOREGROUND }

begin
   with specif do begin
     yr[0]:=adin(yrchan);
     y[0]:= adin(ychan);
     yf:=-py*yfold+y[0];
     estimation(yf-yfold,a,b);
     if initp=false then specfac(a,b,p);
     d.d:=1;   d.z[0]:=1;    d.z[1]:=-1;
     mul(a,d,ai);
     regdesign(ai,b,r,s);
     control;
```

```
      daout(outchan,u[0]);
      uf:=-py*ufold+u[0];
      display;
      if log then
         if (k)=nta) and (k(=ntb) then logdata;
      delayu;
      updata;
      k:=k+1;
      if newpar then begin
         specif:=compar;
         period:=trunc(50*tsamp);
         initabp;
         newpar:=false
      end
   end
end; { of foreground }
```

```
{=============================================================estimation}
Procedure estimation(defy:real; var a,b:polytype);
   { The procedure ERLS computes a Extended Recursive Least
    Squares estimation for different numbers of parameters. }
var rr:real;
    fi:array[1..n] of real;
    kgain:array[1..n] of real;
    i,j:integer;
begin
  with specif do begin
    rr:=1;
    ey:=defy;
    for i:=1 to sumorder do begin
        fi[i]:=0;
        for j:=1 to na do fi[i]:=fi[i]+pp[i,j]*fi[j];
        for j:=na+delayb to sumorder do fi[i]:=fi[i]+pp[i,j]*fi[j];
        rr:=rr+fi[i]*fi[i];
    end;
    for i:=1 to a.d do ey:=ey-fi[i]*a.z[i];
    for i:=delayb to b.d do ey:=ey-fi[a.d+i]*b.z[i];
    for i:=1 to t.d do ey:=ey-fi[nab+i]*t.z[i];
    for i:=1 to sumorder do
        kgain[i]:=fi[i]/rr;
    for i:=1 to a.d do
        a.z[i]:=a.z[i]+kgain[i]*ey;
    for i:=delayb to b.d do
        b.z[i]:=b.z[i]+kgain[a.d+i]*ey;
    for i:=1 to t.d do
        t.z[i]:=t.z[i]+kgain[nab+i]*ey;
    for i:=1 to sumorder do
        for j:=i to sumorder do begin
            pp[i,j]:=pp[i,j]-kgain[i]*fi[j];
            pp[j,i]:=pp[i,j];
            if i=j then pp[i,j]:=pp[i,j]/lambda;
        end;
  end;
end; { of estimation }
```

```
{=============================================================SF}
Procedure Specfac(a,b:polytype; var p:polytype);

{ The procedure uses Newton's iteration method and recursive
  fomulae of complex integral I (1) to perform the spectral
  factorization.
                -1                  -1              -1
     P(z)*P(z   ) = lo*A(z)*A(z   ) + B(z)*B(z   ).     }

label 10;
var ao,bo,c,cr,co,co1,x:polytype;
    qa,qb,int:array[0..n] of real;
    ra,rb:array[0..n,0..n] of real;
    fra,frb:array[0..n] of real;
    i,j,la,si,stab,result:integer;
    fa,fb,fc:real;
    it:integer;

{-----------------------------------------------------Polydivision}
Procedure Polydivision;
var i:integer;
begin
  with specif do begin
    qa[0]:=ao.z[0]/c.z[0];
    qb[0]:=bo.z[0]/c.z[0];
    for i:=1 to nm do begin
        ra[0,i]:=ao.z[i]-qa[0]*c.z[i];
        rb[0,i]:=bo.z[i]-qb[0]*c.z[i];
    end;
    int[0]:=0;
    if la>0 then
       for si:=1 to la do begin
           qa[si]:=ra[si-1,1]/c.z[0];
           qb[si]:=rb[si-1,1]/c.z[0];
           ra[si-1,nm+1]:=0;
           rb[si-1,nm+1]:=0;
           for i:=1 to nm do begin
               ra[si,i]:=ra[si-1,i+1]-qa[si]*c.z[i];
               rb[si,i]:=rb[si-1,i+1]-qb[si]*c.z[i];
           end;
           ra[si,0]:=0;
           rb[si,0]:=0;
         int[si]:=lo*qa[si]*ao.z[0]+qb[si]*bo.z[0];
       end;
  end;
end;{ of polydivision }

{-------------------------------------------------------Iteration}
Procedure Iteration;
var int1:real;
    i,j,laa:integer;
begin
  with specif do begin
    for i:=0 to nm do begin
        a.z[i]:=ao.z[i];
```

```
    b.z[i]:=bo.z[i];
end;
for j:=0 to nm-1 do begin
    int1:=lo*a.z[nm-j]*a.z[nm-j]+b.z[nm-j]*b.z[nm-j];
    int[0]:=int[0]+int1/c.z[0];
    if la>0 then for si:=1 to la do begin
       int1:=lo*a.z[nm-j]*ra[si,nm-j]+b.z[nm-j]*rb[si,nm-j];
       int[si]:=int[si]+int1/c.z[0];
    end;
    for i:=0 to nm-j do cr.z[i]:=c.z[nm-j-i];
    fc:=c.z[nm-j]/c.z[0];
    fa:=a.z[nm-j]/c.z[0];
    fb:=b.z[nm-j]/c.z[0];
    if la>0 then for si:=1 to la do begin
       fra[si]:=ra[si,nm-j]/c.z[0];
       frb[si]:=rb[si,nm-j]/c.z[0];
    end;
    for i:=0 to nm-j-1 do begin
        c.z[i]:=c.z[i]-fc*cr.z[i];
        a.z[i]:=a.z[i]-fa*cr.z[i];
        b.z[i]:=b.z[i]-fb*cr.z[i];
        if la>0 then for si:=1 to la do begin
           ra[si,i]:=ra[si,i]-fra[si]*cr.z[i];
           rb[si,i]:=rb[si,i]-frb[si]*cr.z[i];
        end;
    end;

    if c.z[0]<0 then stab:=stab-1;
    if j=nm-1 then begin
       int1:=lo*a.z[nm-j-1]*a.z[nm-j-1]+b.z[nm-j-1]*b.z[nm-j-1];
       int[0]:=int[0]+int1/c.z[0];
       if la>0 then for si:=1 to la do begin
          int1:=lo*a.z[nm-j-1]*ra[si,nm-j-1]+b.z[nm-j-1]*rb[si,nm-j-1]
          int[si]:=int[si]+int1/c.z[0];
       end;
    end;
end;

if la<0 then laa:=0 else laa:=la;
for si:=0 to laa do begin
    int[si]:=int[si]/co.z[0];
    x.z[si]:=co.z[si]*int[0];
    if (la>0) and (si>0) then
       for i:=1 to si do x.z[si]:=x.z[si]+2*co.z[si-i]*int[i];
end;
x.z[nm]:=(2*(lo*ao.z[0]*ao.z[nm]+bo.z[0]*bo.z[nm])
         -co.z[nm]*x.z[0])/co.z[0];
if nm>1 then begin
   x.z[nm-1]:=2*(lo*(ao.z[0]*ao.z[nm-1]+ao.z[1]*ao.z[nm])
         +bo.z[0]*bo.z[nm-1]+bo.z[1]*bo.z[nm]);
   if nm>2 then
   x.z[nm-1]:=(x.z[nm-1]-co.z[nm-1]*x.z[0]-co.z[nm]*x.z[1]
         -co.z[1]*x.z[nm])/co.z[0]
   else x.z[1]:=(x.z[nm-1]-co.z[1]*x.z[0]
         -co.z[1]*x.z[nm])/(co.z[0]+co.z[2]);
```

```
    end;
  end;
end; { of iteration }

{-------------------------------------------------------------Code of sp}
begin
  with specif do begin
    la:=nm-2;
    stab:=0;
    it:=0;
    for i:=0 to nm do begin
        ao.z[i]:=a.z[i];
        bo.z[i]:=b.z[i];
        co.z[i]:=p.z[i];
        co1.z[i]:=p.z[i];
        c.z[i]:=p.z[i];
    end;
    repeat
      polydivision;
      iteration;
      for i:=0 to nm do c.z[i]:=(co.z[i]+x.z[i])/2;
      if stab<0 then begin
         for i:=0 to nm do p.z[i]:=co1.z[i];
         goto 10;
      end;
      for i:=0 to nm do begin
          co1.z[i]:=co.z[i];
          co.z[i]:=c.z[i];
      end;
      it:=it+1;
    until it=ni;
    for i:=0 to nm do p.z[i]:=c.z[i];
  end;
10:
end; { of specfac }
```

```
{=========================================================Regdesign}
Procedure Regdesign(a,b:polytype; var r,s:polytype);

{ The procedure solves the diophantine equation
      AR + BS = C
The general solution is given by
      R = PE + GV
      S = PF + HV
where V is an arbitrary polynomial.
If    deg p <= deg A + deg B -1
then R and S are found uniquely , else there are one
minimum degree solution for R and one for S. }

const eps1=0.1;
var   com,e,f,g,h,a0,b0,a1,b1,p1:polytype;
      r1,r2,s1,s2,v1,v2,ep,fp:polytype;
      i,j,ki,rd,nmloc:integer;

{-----------------------------------------------------------Inipoiy}
Procedure Inipoly;
{ Set the initial values for polynomials. }
var i,j:integer;
begin
  a0.d:=a.d;
  b0.d:=b.d;
  a1.d:=a.d;
  b1.d:=b.d;
  for i:=0 to a.d do begin
      a0.z[i]:=a.z[i];
      a1.z[i]:=a.z[i];
  end;
  for i:=0 to b.d do begin
      b0.z[i]:=b.z[i];
      b1.z[i]:=b.z[i];
  end;
  if a.d>b.d then nmloc:=a.d else nmloc:=b.d;
  e.d:=b.d;
  g.d:=b.d;
  f.d:=a.d;
  h.d:=a.d;
  for i:=0 to n do begin
      e.z[i]:=0;
      f.z[i]:=0;
      g.z[i]:=0;
      h.z[i]:=0;
  end;
  e.z[0]:=1;
  h.z[0]:=1;
end; { of initpoly }

{-----------------------------------------------------------Exchange}
Procedure exchange(hi,hj:polytype; var hk,hl:polytype);
{ Exchange polynomials Hi and Hj. }
var i:integer;
begin
```

```
  hl.d:=hi.d;
  hk.d:=hj.d;
  for i:=0 to hi.d do hl.z[i]:=hi.z[i];
  for i:=0 to hj.d do hk.z[i]:=hj.z[i];
end; { of exchange }

{-------------------------------------------------------------Norm}
Procedure norm(hi:polytype);
{ Calculate the Euclidean norm of a polynomial Hi. }
var di:real;
    i:integer;
begin
  if hi.d>=0 then begin
     di:=0.0;
     for i:=0 to hi.d do if abs(hi.z[i])>1000 then di:=100000.0
     else for i:=0 to hi.d do di:=di+hi.z[i]*hi.z[i];
     with specif do di:=eps*sqrt(di);
     rd:=round(di);
  end;
end; { of norm }

{-------------------------------------------------------------Reduce}
Procedure reduce(var hi:polytype);
{ Reduce Ni of Hi(z) if its leading coefficient is smaller
  in modulus than an external variable. }
label 2,4;
begin
  if hi.d<0 then goto 4;
2:if abs(hi.z[hi.d])<=rd then begin
     hi.d:=hi.d-1;
     if hi.d<0 then goto 4;
     goto 2;
  end;
4:
end; { of reduce }

{--------------------------------------------------------------Sign}
Procedure Sign(pn:real; var pna:integer);
begin
  if pn>0 then pna:=1;
  if pn<0 then pna:=-1;
end; { of sign }

{-----------------------------------------------------Transformation}
Procedure Transformation;
{ performs the Euclidean transformation to find a general
  solution for the diophantine eguation. }

label 10,20,30,40;
var qab:real;
    i,bb:integer;
begin
    if a.d<b.d then goto 20 else goto 30;
10: while a.d>=b.d do begin
      ki:=a.d-b.d;
```

```
      if abs(b.z[b.d])<eps1 then begin
         sign(b.z[b.d],bb);
         qab:=a.z[a.d]/(eps1*bb);
      end
      else qab:=a.z[a.d]/b.z[b.d];
      a.d:=a.d-1;
      if ki<=a.d then
         for i:=ki to a.d do a.z[i]:=a.z[i]-qab*b.z[i-ki];
      for i:=ki to nmloc do begin
          e.z[i]:=e.z[i]-qab*g.z[i-ki];
          f.z[i]:=f.z[i]-qab*h.z[i-ki];
      end;
      reduce(a);
      if a.d<b.d then goto 20;
    end;
20:
    exchange(a,b,a,b);
    exchange(e,g,e,g);
    exchange(f,h,f,h);
    if (b.d=0) and (abs(b.z[0])<eps1) then goto 40;
30: if b.d>=0 then begin
       norm(b);
       goto 10;
    end;
40: if abs(a.z[0])<eps1 then qab:=1 else qab:=a.z[0];
    for i:=0 to nmloc do begin
        e.z[i]:=e.z[i]/qab;
        f.z[i]:=f.z[i]/qab;
        a.z[i]:=a.z[i]/qab;
    end;
    norm(e);
    reduce(e);
    norm(f);
    reduce(f);
    g.d:=g.d-a.d;
    h.d:=h.d-a.d;
end; { of transformation }

{-------------------------------------------------------------Polymul}
Procedure Polymul(p,ef:polytype; var efp:polytype);
{ Polynomial multiplication efp=p*ef }
var i,j:integer;
begin
  efp.d:=p.d+ef.d;
  for i:=0 to efp.d do efp.z[i]:=0;
  for i:=0 to p.d do
      for j:=0 to ef.d do
          efp.z[i+j]:=efp.z[i+j]+p.z[i]*ef.z[j];
end; { of polymul }

{-------------------------------------------------------------Polydiv}
Procedure Polydiv(efp,gh:polytype; var rs,vv:polytype);
{ Polynomial division rs=efp mod gh.
                      vv=-(efp div gh). }
var efp1,gh1,rs1,vv1:array[0..n] of real;
```

```
    i,ghdeg,gg:integer;
begin
  repeat
    ghdeg:=gh.d;
    norm(gh);
    reduce(gh)
  until ghdeg=gh.d;
  vv.d:=efp.d-gh.d;
  rs.d:=gh.d-1;
  for i:=0 to efp.d do efp1[i]:=efp.z[efp.d-i];
  for i:=0 to gh.d do gh1[i]:=gh.z[gh.d-i];
  for i:=gh.d+1 to efp.d do gh1[i]:=0;
  vv1[0]:=efp1[0]/gh1[0];
  for i:=1 to efp.d do
      rs1[i]:=efp1[i]-vv1[0]*gh1[i];
  for i:=1 to vv.d do begin
      vv1[i]:=rs1[1]/gh1[0];
      rs1[efp.d+1]:=0;
      for j:=i to efp.d do
          rs1[j]:=rs1[j+1]-vv1[i]*gh1[j];
  end;
  for i:=1 to rs.d+1 do rs.z[i-1]:=rs1[rs.d+2-i];
  for i:=0 to vv.d do vv.z[i]:=-vv1[vv.d-i];
end; { of polydiv }

{-------------------------------------------------------Polyadd}
Procedure Polyadd(efp,ghv:polytype; var rs:polytype);
{ Polynomial addition rs=efp+ghv. }
var i,diff:integer;
begin
  diff:=efp.d-ghv.d;
  if diff>0 then begin
     rs.d:=efp.d;
     for i:=0 to efp.d do rs.z[i]:=efp.z[i];
     for i:=0 to ghv.d do
         rs.z[i]:=rs.z[i]+ghv.z[i];
     end
     else begin
       rs.d:=ghv.d;
       for i:=0 to ghv.d do rs.z[i]:=ghv.z[i];
       for i:=0 to efp.d do
           rs.z[i]:=rs.z[i]+efp.z[i];
     end;
end; { of polyadd }

{-------------------------------------------------------Regulator}
Procedure Regulator;
{ computes coefficients of V1,V2,R1,R2,S1 and R2.}
var rs1:polytype;

begin
  with specif do begin
    polymul(p,t,p1);
    polymul(p1,e,ep);
    polymul(p1,f,fp);
```

```
    if minr then begin
       polydiv(ep,g,r1,v1);
       polymul(v1,h,rs1);
       polyadd(fp,rs1,s1);
       norm(s1);
       reduce(s1);
    end
    else begin
           polydiv(fp,h,s2,v2);
           polymul(v2,g,rs1);
           polyadd(ep,rs1,r2);
           norm(r2);
           reduce(r2);
         end;
  end;
end; { of regulator }

{-------------------------------------------------------------Division}
Procedure Division;
{ Cancell the commom factor from A(z) and B(z). }
var i,j:integer;
begin
  a.d:=a0.d-com.d;
  b.d:=b0.d-com.d;
  for i:=0 to a0.d do a.z[i]:=a0.z[i];
  for i:=0 to b0.d do b.z[i]:=b0.z[i];
  for j:=0 to a.d do begin
      a.z[j]:=a.z[j]/com.z[0];
      for i:=j+1 to com.d+j do
          a.z[i]:=a.z[i]-com.z[i-j]*a.z[j];
  end;
  for j:=0 to b.d do begin
      b.z[j]:=b.z[j]/com.z[0];
      for i:=j+1 to com.d+j do
          b.z[i]:=b.z[i]-com.z[i-j]*b.z[j];
  end;
  a1.d:=a.d;
  b1.d:=b.d;
  for i:=0 to a.d do a1.z[i]:=a.z[i];
  for i:=0 to b.d do b1.z[i]:=b.z[i];
end; { of division }

{=============================================================Code of RC}
begin
  inipoly;
  transformation;
  if a.d>0 then begin
     com.d:=a.d;
     for i:=0 to a.d do com.z[i]:=a.z[i];
     division;
  end;
  regulator;
  with specif do
    if minr then begin
       r.d:=r1.d;  for i:=0 to r.d do r.z[i]:=r1.z[i];
```

```
        s.d:=s1.d;  for i:=0 to s.d do s.z[i]:=s1.z[i];
        if (p.d<a1.d+b1.d) and (s.d)a1.d-1) then s.d:=a1.d-1;
    end
    else begin
           r.d:=r2.d;  for i:=0 to r.d do r.z[i]:=r2.z[i];
           s.d:=s2.d;  for i:=0 to s.d do s.z[i]:=s2.z[i];
           if (p.d<a1.d+b1.d) and (r.d)b1.d-1) then r.d:=b1.d-1;
         end;
end; { og regdesign }
```

```
{=============================================================OPCOM}
Procedure Opcom;

   { The procedure OPCOM is organized as follows:
     Initialize    recognizes identifiers and initializes:
        initestpar initializes estimated parameters;
        initregpar initializes regulator parameters;
        inituy     initializes input and output;
        initspecif gives specifications;
     Help          lists available commands.
     Par           modifies parameters:
        getiden    reads the identifiers;
        get        reads parameter if it is a real;
        take       reads parameter if it is a integer;
        askboolean reads command if it is a boolean;
        initabp;
        error      gives a sensible orror message if in error.
     Run           runs the selftuner.
     Stop          stops the selftuner.
     Disp          display the current parameters.
     Store         logs inputs and output.
        error
     Resultp       logs estimated parameters.
        error
     Resultu       logs reglator parameters.
        error
     Exit          stops the whole program and exits it into
                   computer.}

{ DECLARATIONS OF OPCOM }

label 999;
const idlength=8;
      blank=' ';
type  identifiertype=array[1..idlength] of char;
      opindex=(xhelp,xpar,xrun,xstop,xdisp,xdispp,xstore,
               xresultp,xresultu,xexit,xlastop);
      asindex=(stsamp,slo,slambda,spo,spy,seps,slolim,shilim,
               snd,sni,sns,snta,sntb,slog,sminr,sinita,sinitb,
               sinitp,snp,spz,spa,spb,spc,spd,spe,sna,saa,sab,
               sac,sad,snb,sba,sbb,sbc,sbd,snt,sdelayb,syrchan,
               sychan,suchan,soutchan,slastas);
      errors=(fewarg,manyarg,noname,prierr,illfile,nolog);
var   identifier:identifiertype;
      operation: array[opindex] of identifiertype;
      assignment:array[asindex] of identifiertype;
      xop:opindex;
      sas:asindex;
      ch: char;
      logdata,logtheta,logreg:boolean;

{ PROCEDURE FOR OPCOM }

{-------------------------------------------------------------help}
Procedure help;
```

```
  { lists available commands and information on screen }
 var i:integer;
 begin
   for i:=1 to 5 do writeln;
   writeln('          SISO Self-tuner');
   writeln;
   writeln('The available commands are as follows:');
   writeln;
   write(' ':4,'DISP');          writeln(' ':12,'STORE');
   write(' ':4,'PAR');           writeln(' ':13,'RESULTP');
   write(' ':4,'RUN');           writeln(' ':13,'RESULTU');
   write(' ':4,'STOP');          writeln(' ':12,'EXIT');
   writeln;
   writeln('The parameters can be changed:');
   writeln;
   writeln(' ':4,'TSAMP,LO,LAMBDA,PO,PY,EPS,ND,NI,NS,NT,
           NTA,NTB,LOLIM,HILIM');
   writeln;
   writeln('The initial values can be assigned:');
   writeln;
   writeln(' ':4,'NA,AA,AB,AC,AD,NB,DELAYB,BA,BB,BC,BD,NP,
           PZ,PA,PB,PC,PD,PE');
   writeln;
   writeln('The boolean variables are:');
   writeln;
   writeln(' ':4,'LOG,MINR,INITA,INITB,INITP');
   writeln;
   writeln('The channels can be chosen:');
   writeln;
   writeln(' ':4,'YRCHAN,YCHAN,UCHAN,OUTCHAN'); writeln
 end; { of help }

{-------------------------------------------------------------error}
Procedure error(err:errors);
   { gives a sensible error message }
 begin
   case err of
     fewarg: writeln('too few arguments');
     manyarg:writeln('too many arguments');
     prierr: writeln(identifier,':illegal value');
     noname: writeln(identifier,':illegal argument');
     nolog:  writeln(identifier,':log do not be reguired');
     illfile:writeln(identifier,':ill file')
   end;
   goto 999
 end; { of error }

{-----------------------------------------------------initspecif}
Procedure initspecif;
   { assigns specification }
 begin
   with compar do begin
     tsamp:=15;       lo:=5;
     na:=2;           nb:=2;
     delayb:=1;       np:=2;
```

```
      nt:=0;              ni:=3;
      ns:=5;              nd:=5;
      lambda:=0.98;       po:=100;
      eps:=0.0000001;     py:=0;
      lolim:=-1;          hilim:=1;
      nta:=1;             ntb:=10;
      yrchan:=0;          ychan:=1;
      uchan:=2;           outchan:=1;
      log:=false;         minr:=true;
      inita:=false;       initb:=false;
      initp:=false;       aa:=-0.5;
      ab:=0.001;          ac:=0.001;
      ad:=0.001;          ba:=0.1;
      bb:=0.5;            bc:=0.001;
      bd:=0.001;          pz:=1;
      pa:=0.001;          pb:=0.001;
      pc:=0.001;          pd:=0.001;
      pe:=0.001;
    end;
 end; { of initspecif }

{-------------------------------------------------------initestpar}
Procedure initestpar;
   { initializes th, fi and p }
 var i,j:integer;
 begin
   with compar do begin
     for i:=0 to n do begin
         a.z[i]:=0.001;
         b.z[i]:=0.001;
         p.z[i]:=0.001;
         xm.z[i]:=0.001;
         fi[i]:=0.0;
         for j:=1 to n do
             if i=j then pp[i,j]:=po else pp[i,j]:=0.0;
     end;
     a.d:=na;                a.z[0]:=1;
     b.d:=nb;
     p.d:=np;                p.z[0]:=1;
     xm.d:=np;
     if na>nb then nm:=na else nm:=nb;
     sumorder:=4;            nab:=4;
     ub:=0;
     k:=1;                   q:=1;
     w:=1;
   end;
 end; { of initestpar }

{-----------------------------------------------------------inituy}
Procedure inituy;
   { initializes input and output }
 var i:integer;
 begin
   with compar do begin
     for i:=0 to n do begin
```

```
        yr[i]:=0.0;
        y[i]:=0.0;      yfold:=0.0;
        u[i]:=0.0;      ufold:=0.0;
    end;
    for i:=1 to nd do delu[i]:=0.0;
    daout(0,0.0);  daout(1,0.0)
  end;
end; { of inituy }

{-------------------------------------------------------------initregpar}
Procedure initregpar;
  { initializes reglator parameters }
var i:integer;
begin
  for i:=1 to n do begin
      t.z[i]:=0.0;
      r.z[i]:=0.0;
      s.z[i]:=0.0;
  end;
  t.d:=0;     t.z[0]:=1;
  r.d:=0;     r.z[0]:=1;
  s.d:=0;     s.z[0]:=0;
end; { of initregpar }

{-----------------------------------------------------------initialize}
Procedure initialize;
  { recognizes identifiers and initializes }
begin
  operation[xhelp]:=     'HELP     ';
  operation[xdisp]:=     'DISP     ';
  operation[xdispp]:=    'DISPP    ';
  operation[xpar]:=      'PAR      ';
  operation[xrun]:=      'RUN      ';
  operation[xstop]:=     'STOP     ';
  operation[xexit]:=     'EXIT     ';
  operation[xstore]:=    'STORE    ';
  operation[xresultp]:= 'RESULTP  ';
  operation[xresultu]:= 'RESULTU  ';
  assignment[stsamp]:=   'TSAMP    ';
  assignment[slo]:=      'LO       ';
  assignment[slambda]:= 'LAMBDA   ';
  assignment[spo]:=      'PO       ';
  assignment[spy]:=      'PY       ';
  assignment[seps]:=     'EPS      ';
  assignment[slolim]:=   'LOLIM    ';
  assignment[shilim]:=   'HILIM    ';
  assignment[snd]:=      'ND       ';
  assignment[sni]:=      'NI       ';
  assignment[sns]:=      'NS       ';
  assignment[snt]:=      'NT       ';
  assignment[snta]:=     'NTA      ';
  assignment[sntb]:=     'NTB      ';
  assignment[slog]:=     'LOG      ';
  assignment[sminr]:=    'MINR     ';
  assignment[sinita]:=   'INITA    ';
```

```
    assignment[sinitb]:=  'INITB   ';
    assignment[sinitp]:=  'INITP   ';
    assignment[syrchan]:= 'YRCHAN  ';
    assignment[sychan]:=  'YCHAN   ';
    assignment[suchan]:=  'UCHAN   ';
    assignment[soutchan]:='OUTCHAN ';
    assignment[sna]:=     'NA      ';
    assignment[saa]:=     'AA      ';
    assignment[sab]:=     'AB      ';
    assignment[sac]:=     'AC      ';
    assignment[sad]:=     'AD      ';
    assignment[snb]:=     'NB      ';
    assignment[sdelayb]:= 'DELAYB  ';
    assignment[sba]:=     'BA      ';
    assignment[sbb]:=     'BB      ';
    assignment[sbc]:=     'BC      ';
    assignment[sbd]:=     'BD      ';
    assignment[snp]:=     'NP      ';
    assignment[spz]:=     'PZ      ';
    assignment[spa]:=     'PA      ';
    assignment[spb]:=     'PB      ';
    assignment[spc]:=     'PC      ';
    assignment[spd]:=     'PD      ';
    assignment[spe]:=     'PE      ';
    with compar do begin
      initspecif;
      initestpar;
      initregpar;
      inituy;
      specif:=compar;
      newpar:=true
    end;
  end; { of initialize }

{-----------------------------------------------------------skipblank}
Procedure skipblank;
  begin
    repeat read(ch) until (ch()blank) or eoln
  end; { of skipblank }

{-----------------------------------------------------------getident}
Procedure getident;
    { reads a identifier from the terminal }
  var i:integer;
  begin
    for i:=1 to idlength do identifier[i]:=blank;
    skipblank;
    i:=1;
    while (ch)='A') and (ch(='Z') and (i(9) and not eoln do
    begin identifier[i]:=ch;
          i:=i+1;
          read(ch)
    end;
    if (ch)='A') and (ch(='Z') then identifier[i]:=ch
  end; { of getident }
```

```
{-------------------------------------------------------------take}
Procedure take(var rn:real);
   { reads a number if it is a real }
 begin
   if ch=blank then read(rn);
   if (rn<-1000) or (rn>1000) then error(prierr) else read(ch);
   if ch=',' then read(ch)
 end; { of take }

{-------------------------------------------------------------get}
Procedure get(var inn:integer);
   { reads a number if it is a integer }
 begin
   if ch=blank then read(inn);
   if inn<0 then error(prierr) else read(ch);
   if ch=',' then read(ch)
 end; { of get }

{-----------------------------------------------------askboolean}
Procedure askboolean(var command:boolean);
   { reads a command if it is a boolean }
 var comname:array[1..idlength] of char;
 begin
     comname:=identifier;
     if ch=blank then getident;
     if identifier='TRUE      ' then command:=true
     else command:=false;
     identifier:=comname
 end; { of askboolean }

{-----------------------------------------------------------par}
Procedure par;
   { updates parameters }
 var pc:specificationtype;
 begin
   if eoln then error(fewarg);
   pc:=compar;
   repeat getident;
     assignment[slastas]:=identifier;
     sas:=stsamp;
     while assignment[sas]<>identifier do sas:=succ(sas);
     case sas of
       stsamp: take(pc.tsamp);
       slambda: take(pc.lambda);
       spo:     take(pc.po);
       spy:     take(pc.py);
       seps:    take(pc.eps);
       slo:     take(pc.lo);
       slolim:  take(pc.lolim);
       shilim:  take(pc.hilim);
       saa:     take(pc.aa);
       sab:     take(pc.ab);
       sac:     take(pc.ac);
       sad:     take(pc.ad);
```

```
        sba:      take(pc.ba);
        sbb:      take(pc.bb);
        sbc:      take(pc.bc);
        sbd:      take(pc.bd);
        spz:      take(pc.pz);
        spa:      take(pc.pa);
        spb:      take(pc.pb);
        spc:      take(pc.pc);
        spd:      take(pc.pd);
        spe:      take(pc.pe);
        sna:      get(pc.na);
        snb:      get(pc.nb);
        sdelayb:  get(pc.delayb);
        snp:      get(pc.np);
        snd:      get(pc.nd);
        sni:      get(pc.ni);
        sns:      get(pc.ns);
        snt:      get(pc.nt);
        snta:     get(pc.nta);
        sntb:     get(pc.ntb);
        syrchan:  get(pc.yrchan);
        sychan:   get(pc.ychan);
        suchan:   get(pc.uchan);
        soutchan:get(pc.outchan);
        slog:     askboolean(pc.log);
        sminr:    askboolean(pc.minr);
        sinita:   askboolean(pc.inita);
        sinitb:   askboolean(pc.initb);
        sinitp:   askboolean(pc.initp);
        slastas:  error(noname)
      end;
      writeln(identifier,'has been read')
    until eoln;
    newpar:=false;
    compar:=pc;
    if period=0 then specif:=compar;
    newpar:=true;
end;  { of par }

{-------------------------------------------------------------------disp}
Procedure disp;
   { displays current parameters }
 var rr:real;

 Procedure dataform(rr:real);
    { formats a real number }
  var w:integer;
  begin
    w:=rform(rr,6);  write(rr:10:w)
  end; { of dataform }

 begin { disp }
   with compar do begin
     writeln('   The current parameters are:');
     write('tsamp=  ');  dataform(tsamp);     write('   ');
```

```
      write('lo=      ');  dataform(lo);          writeln;
      write('ns=      ');  write(ns:6);           write(' ':7);
      write('ni=      ');  writeln(ni:6);
      write('nd=      ');  write(nd:6);           write(' ':7);
      write('lambda=  ');  dataform(lambda);      writeln;
      write('po=      ');  dataform(po);          write('    ');
      write('py=      ');  dataform(py);          writeln;
      write('eps=     ');  dataform(eps);         write('    ');
      write('hilim=   ');  dataform(hilim);       writeln;
      write('lolim=   ');  dataform(lolim);       write('    ');
      write('nta=     ');  writeln(nta:6);
      write('ntb=     ');  write(ntb:6);          write(' ':7);
      write('yrchan=  ');  writeln(yrchan:6);
      write('ychan=   ');  write(ychan:6);        write(' ':7);
      write('uchan=   ');  writeln(uchan:6);
      write('outchan=');   write(outchan:6);      write(' ':7);
      write('log=     ');  writeln(' ':4,log);
      write('minr=    ');  write(' ':4,minr);     write(' ':3);
      write('inita=   ');  writeln(' ':4,inita);
      write('initb=   ');  write(' ':4,initb);    write(' ':3);
      write('initp=   ');  writeln(' ':4,initp);
      write('na=      ');  write(na:6);           write(' ':7);
      write('aa=      ');  dataform(aa);          writeln;
      write('ab=      ');  dataform(ab);          write('    ');
      write('ac=      ');  dataform(ac);          writeln;
      write('ad=      ');  dataform(ad);          write('    ');
      write('nb=      ');  write(nb:6);           writeln;
      write('delayb=  ');  write(delayb:6);       write(' ':7);
      write('ba=      ');  dataform(ba);          writeln;
      write('bb=      ');  dataform(bb);          write('    ');
      write('bc=      ');  dataform(bc);          writeln;
      write('bd=      ');  dataform(bd);          write('    ');
      write('np=      ');  write(np:6);           writeln;
      write('pz=      ');  dataform(pz);          write('    ');
      write('pa=      ');  dataform(pa);          writeln;
      write('pb=      ');  dataform(pb);          write('    ');
      write('pc=      ');  dataform(pc);          writeln;
      write('pd=      ');  dataform(pd);          write('    ');
      write('pe=      ');  dataform(pe);          writeln;
      write('nt=      ');  writeln(nt:6);
    end
  end; { of disp }

{-------------------------------------------------------------dispp}
Procedure dispp;
    { write P matrix on sceeen }
  var i,j:integer;
  begin
    with specif do begin
      for i:=1 to sumorder do begin
          for j:=1 to sumorder do write(pp[i,j]:8:4);
          writeln;
      end;
    end;
  end;
```

```
{-------------------------------------------------------buildfile}
Procedure buildfile;
   { stores data of input-output, estimated parameters and
     regulator parameters in files respectively }
 var outfile:file of char;
     filename:array[1..14] of char;
     i,len:integer;
 begin
   with specif do
     if log=false then error(nolog) else begin
     read(filename);   len:=-1;
     rewrite(outfile,filename,'DAT',len);
     if len=-1 then error(illfile);
     for w:=1 to (ntb-nta) do begin
         if logdata then  begin
            for i:=0 to 2 do write(outfile,loguy[i,w]:8:4);
            writeln(outfile)
         end;
         if logtheta then begin
            for i:=1 to a.d do write(outfile,loga[i,w]:8:4);
            for i:=1 to b.d do write(outfile,logb[i,w]:8:4);
            writeln(outfile)
         end;
         if logreg then begin
            for i:=0 to t.d do write(outfile,logt[i,w]:8:4);
            for i:=0 to r.d do write(outfile,logr[i,w]:8:4);
            for i:=0 to s.d do write(outfile,logs[i,w]:8:4);
            writeln(outfile)
         end;
       end;
       close(outfile)
     end
 end; { of buildfile }

{-----------------------------------------------------------stop}
Procedure stop;
 begin
   period:=0; writeln(k:6,' data points have been collected.')
 end; { of stop }

{-------------------------------------------------CODE OF OPCOM}
{ CODE OPCOM }

begin
   initialize;
   repeat
     write(')');
     getident;
     operation[xlastop]:=identifier;
     xop:=xhelp;
     while operation[xop]<>identifier do xop:=succ(xop);
     case xop of
          xhelp:   help;
          xdisp:   disp;
```

```
          xdispp:  dispp;
          xpar:    par;
          xrun:    with specif do
                     period:=trunc(50*tsamp);
          xstop;
          xexit:   stop;
          xstore:  begin logdata:=true;
                     logtheta:=false;    logreg:=false;
                     buildfile
                   end;
          xresultp:begin logtheta:=true;
                     logdata:=false;     logreg:=false;
                     buildfile
                   end;
          xresultu:begin logreg:=true;
                     logdata:=false;     logtheta:=false;
                     buildfile
                   end;
          xlastop: error(noname)
     end;
999: readln
   until xop=xexit
end; { of opcom }

{=============================================CODE OF MAIN PROGRAM}
{ CODE MAIN PROGRAM }
Procedure foreground;external;

begin
   period:=0;
   clksave;
   schedule(foreground,period);
   opcom;
   clkrestore;
end. { of main program }
```