CODEN:LUTFD2/(TFRT-5303)/1-064/(1983)

EN SJÄLVINSTÄLLANDE PREDIKTOR MED OPERATÖRSKOMMUNIKATION SKRIVEN I OMSI-PASCAL.

STEFAN KNUTSSON

INSTITUTIONEN FÖR REGLERTEKNIK
LUNDS TEKNISKA HÖGSKOLA

OKTOBER 1983

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

Document name
Master thesis

Date of issue
October 1983

Document number
CODEN:LUTFD2/(TFRT-5303)/1-064/(1983)

Author(s)
Stefan Knutsson

Supervisor
Jan Sternby

Sponsoring organization

Title and subtitle
En självinställande prediktor med operatörskommunikation skriven i Omsi-Pascal.
(A self-tuning predictor with operator communication written in Omsi-Pascal.)

Abstract

The thesis has been made in cooperation with Kockumation AB, Malmö. The task is to implement an adaptive predictor on a LSI-11.
The program is written such that all interesting parameters can be changed via the operator communication. The parameters of the model and variables are displayed on the terminal and are updated at each sampling interval. Further a curve can be displayed showing the prediction a number of sampling intervals ahead. The curve can be rescaled in order to give better resolution.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
Swedish

Number of pages
64

Recipient's notes

Security classification

EN SJÄLVINSTÄLLANDE PREDIKTOR MED OPERATÖRSKOMMUNIKATION SKRIVEN I OMSI-PASCAL.

EXAMENSARBETE UTFÖRT AV
STEFAN KNUTSSON

Lund i oktober 1982

Arbetet utfördes vid
Institutionen för Reglerteknik, LTH och
KOCKUMATION AB, Malmö

HANDLEDARE :
JAN STERNBY, KOCKUMATION AB
BJÖRN WITTENMARK, Institutionen för Reglerteknik, LTH

INNEHÅLL

REFERAT

Examensarbetet har utförts på uppdrag av Kockumation AB, Malmö och går ut på att implementera en adaptiv prediktor i en LSI-11.

Programmet har skrivits så att man via operatörskommunikationen kan ändra alla intressanta parametrar. Modellens parametrar och variabler kan visas på terminalen och dessa uppdateras då efter varje samplingstillfälle. Vidare kan en kurva ritas upp som visar prediktionen ett valfritt antal samplingsintervall framåt i tiden. Kurvan kan genom omskalning förstoras.

INTRODUKTION

I många industriella processer finns en tidsfördröjning inbyggd. Denna medför att man vid ändringar i insignalerna inte omedelbart får en ändring i utsignalen. Tidsfördröjningen behöver inte vara inbyggd utan kan även vara påtvingad t.ex. på grund av mätanordningarnas placering. Genom att prediktera framtida värden på processens utsignal kommer man ifrån de olägenheter som tidsfördröjningar medför. För att kunna prediktera krävs att processen beskrivs av en matematisk modell. Modellen är dynamisk, tidsdiskret och linjär i parametrarna med en utsignal y, minst en insignal u och en brussignal e.

$$A(q^{-1})y(t) = B(q^{-1})u(t) + C(q^{-1})e(t) + K$$

där A, B och C är polynom, av valfritt gradtal, i bakåtskiftoperatorn och K är en konstant. Om det finns flera insignaler, eller mätbara störningar, kommer dessa in på samma sätt som u.

Parametrarna i modellen uppdateras kontinuerligt med minsta kvadratmetoden, vilket innebär att man ändrar parametrarna så att summan av kvadraten på enstegsprediktionsfelet minimeras. Efter varje uppdatering stabilitetstestas C-polynomet. Detta är nödvändigt eftersom prediktorn bara blir stabil om C-polynomet är stabilt.

Den bästa prediktorn beräknas ur

$$\hat{Cy}(t+m|t) = Gy(t) + BFu(t+m) + q^{m}FK$$

där m är prediktionshorisonten och polynomen $G(q^{-1})$ och $F(q^{-1})$ bestäms ur identiteten

$$C = AF + q^{-m}G \ .$$

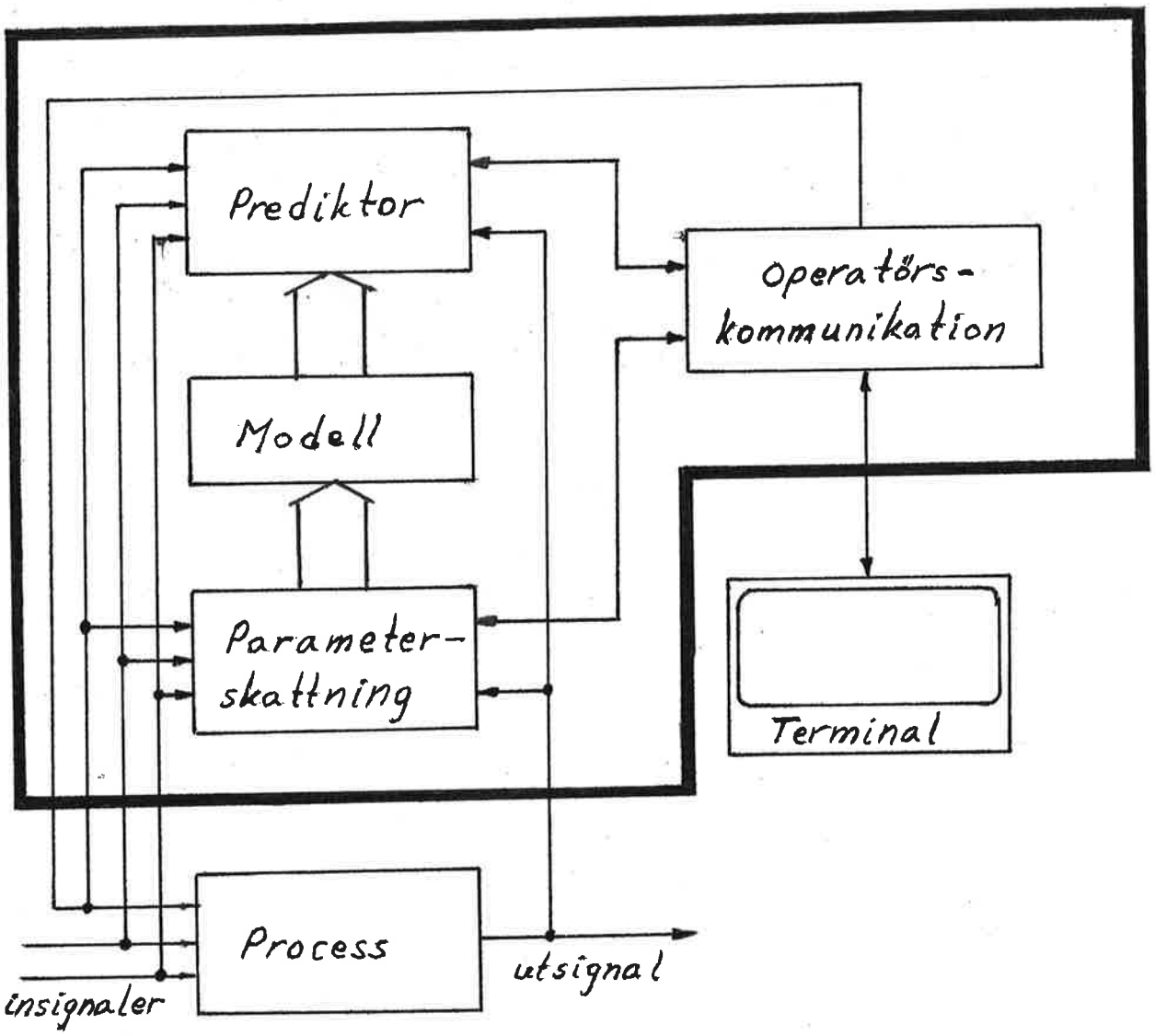
För härledning av parameterskattaren och prediktorn se bilaga A.

Introduktion

Operatören har 6 kommandon till sitt förfogande för att styra och följa händelseutvecklingen.

ON	: samplingen startas.
OFF	: samplingen stängs av.
PAR	: ändring av parametrar.
DISP	: parametrar och prediktion visas på terminal och uppdateras efter varje samplingstillfälle.
SAVE	: alla parametervärden lagras på sekundärminne.
UNSAVE	: alla parametervärden läses från sekundärminne.

Den pricipiella uppbyggnaden av programmet visas i figur 1. Möjlighet att ställa ut ett referensvärde på DA-omvandlare finns.



Figur 1. Programmets principiella uppbyggnad.

OPERATÖRSMANUAL

Uppkoppling och start av programmet.

Processens mätgivare kopplas samman med AD-omvandlaren. Utsignal till kanal 0, insignal 1 till kanal 1, insignal 2 till kanal 2 osv. Signalen in till AD-omvandlaren måste vara mellan -10V och +10V. Internt i datorn representeras signalen av ett tal i området [-1.0 , 1.0]. Om referensvärdessignal önskas fås denna från DA-omvandlarens kanal 0. En linjeskrivare kan kopplas till DA-omvandlarens kanal 1, där de predikterade värdena ställs ut. Från DA-omvandlaren fås en signal i området [-10V , +10V].

Programmet startas med kommandot R MAIN, varvid skärmen suddas. Cursorn stannar på kommandoraden. Börja med att trycka ner RETURN-knappen en gång. Datorn är nu redo att ta emot kommandon.

Kommandon.

Om kommandoraden inte är tom suddas den genom att RETURN trycks ner två gånger. Det finns sex kommandon till förfogande (ev startvärden på parametrar anges inom parantes) :

ON
Samplingen skall startas. Det finns två samplingstider som skall sättas. Dels hfast (50 tick = 1 sekund) som är den korta samplingstiden. Den signal som samplas kan här filtreras genom ett första ordningens lågpassfilter vars filterkonstant kan ändras med ett PAR-kommando. Den längre samplingstiden hslow (0) anger hur ofta man skall läsa av den signal som redan samplats med hfast. Ett exempel : hslow = 5 medför att när vi samplat snabbt 5 ggr skall vi läsa av den filtrerade signalen.

OFF
Samplingen stängs av, hslow sätts lika med 0.

PAR parameternamn
Anger att parametern med namnet parameternamn skall ändras. Först visas nuvarande värde och "newvalue:=" skrivs ut. Om ingen ändring önskas måste man skriva dit det gamla värdet. Om parameternamn är en vektor visas innehållet i hela vektorn. Ovanför varje värde visas i vilken position det ligger. Därefter skrivs "position:=" på kommandoraden. Den position som skall ändras skrivs varefter man får "newvalue:=" igen. En förteckning över parameternamn finns nedan.

DISP namn
Anger att hela bildskärmen tas i bruk för visning av dels parametrar och variabler och dels kurvor. Detta kommando medför att de värden som visas på bildskärmen uppdateras efter varje samplingstillfälle. Datorn kan ej ta emot andra kommando medan DISP är i funktion. Disptillståndet lämnas genom att trycka ner RETURN. Två olika "namn" finns tillgängliga, all och curve. All medför visning av modellpolynomens gradtal, skalningsvectorn, parametervektorn, variabelvektorn (är multiplicerad med

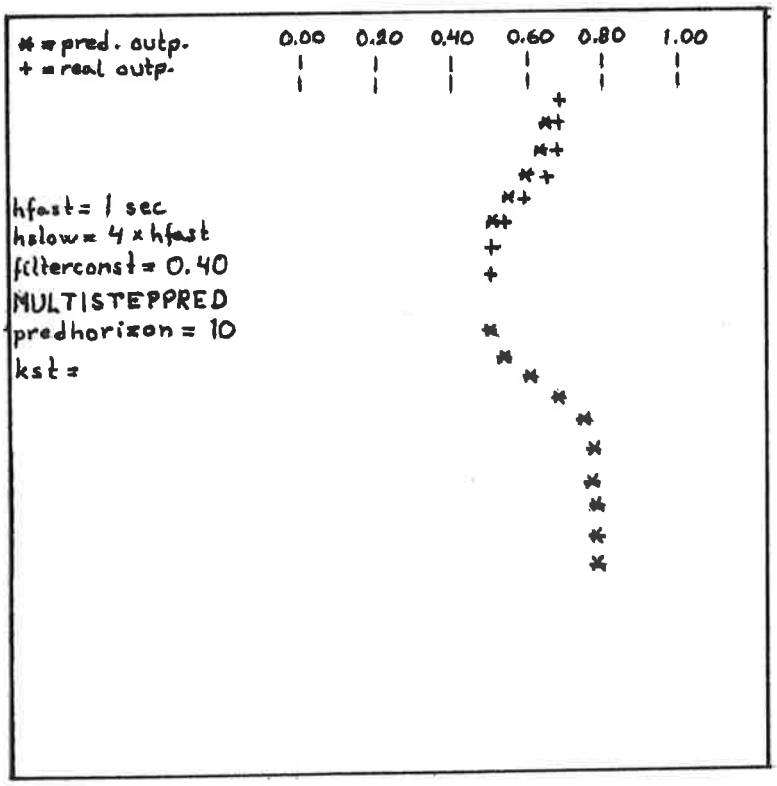
skalningsvektorn), K-vektorn samt spåret av P-matrisen. Curve visar samplingstiderna, filterkonstant, vilken typ av prediktor som används, prediktionshorisonten, kst (se nedan för förklaring) samt en kurva som visar dels en jämförelse mellan gamla prediktioner och motsvarande utsignal och dels den senaste prediktionen. Om prediktion och utsignal stämmer överens syns endast ett "+" på den raden. Se figur 2.

SAVE

Används då man vill spara samtliga parametrar och variabler. "Filename:" skrivs ut och ett valfritt namn anges. På DXO finns nu en fil namn.DAT. Filen kan dock inte läsas av kommandon typ "TYPE namn" eftersom det inte är ASCII-tecken som lagrats.

UNSAVE

Läser in samtliga parametrar och variabler från sekundärminne. Funktion som SAVE.



Figur 2. Kommandot CURVE.

PAR-kommandot accepterar följande parametrar:

HF : den korta samplingstiden (50). Anges i tick, där 1 tick=20 ms. Om HF=0 utförs ingen sampling.

HS : den långa samplingstiden (0). Anges i antal korta samplingsintervall. Ex om HF=50 (=1sec) och HS=2 så utförs samplingen varannan sekund. OBS om HF ändras och samplingsintervallet önskas konstant så måste HS justeras.

NOOFU : antalet insignaler, som mäts, till processen (1).

CHAN : anger vilken kanal på AD-omvandlaren som mätsignalen kopplas till. CHAN är en vektor där position 1 avser processens utsignal, position 2 ... avser insignalerna (pos1=0, pos2=1 ...).

FILTERC : filterkonstanten (0.25) i det exponentiell utjämningsfilter som finns efter den snabba samplingen. Om FILTERC är nära 1 tas liten hänsyn till mätvärdet, dvs stor utjämning erhålles. Om FILTERC är nära 0 blir känsligheten för variationer i mätsignalen stor.

PARV : modellens parametrar (samtliga har startvärde 0).

SCALE : skalningsvektor (samtliga startvärden = 1) som används för att skala variablerna så att diagonalelementen i P-matrisen blir av samma storleksordning.

GRAD : en vektor som innehåller modellpolynomens gradtal (samtliga startvärden=3). Pos1=A-polynomets gradtal, pos2=B:s gradtal, pos3=C:s gradtal . Om flera insignaler används placeras de mellan B- och C-polynomet.

FORGET : glömskefaktorn (0.98). Möjliggör förändringar i processens dynamik. Innebär att man vid parameterskattningen tar mer hänsyn till de senaste mätvärdena. Om FORGET=1 medges inga förändringar i processdynamiken.

MAXTRP : anger maximum av spåret av P-matrisen (100).

TDELV : en vektor vars innehåll anger de olika insignalernas tidsfördröjning (samtliga startvärden = 0).

EST : anger typ av parameterskattare. Endast elsO är implementerad.

PRED : anger typ av prediktor (multistep). Man kan välja mellan enstegsprediktor, k-stegsprediktor och multistegsprediktor.

PREDHOR : prediktionshorisonten (10).

KST : vid multistegsprediktion kan man välja vilken prediktion som skall jämföras med verklig utsignal (1). Används endast vid display av kurvor.

OUTP : referensvärde på AD-omvandlarens kanal 0 (0).

POS : anger mellan vilka positioner på skärmen som kurvan skall ritas (25 - 75).

LIM : skalgradering (0.0 - 1.0).

Startvärden

De parametrar som kan ändras från terminal har följande startvärden:

```
CHAN = [  0  1  2  3  4  ]
EST = els0
FILTERC = 0.25
FORGET = 0.98
GRAD = [  3  3  3  0  0  0  ]
HF = 50
HIGHLIM = 1
HIGHPOS = 75
HS = 0
KST = 1
LOWLIM = 0
LOWPOS = 25
MAXTRP = 100
NOOFU = 1
OUTP = 0
PRED = multistep
PREDHOR = 10
PARV, SCALE = samtliga positioner 0
TDELV = [  0  0  0  0  ]
```

övriga parametrars startvärden är:

```
GRADPARVECT = 10
POINTVECT = [  1  4  7  10  0  0  0  ]
STOPFLAG = false
TRP = 10
YPREDVECT, OLDYV,
OLDYPREDV, TDELSAVE,
SAVEMAT, INVECT = samtliga positioner 0
K, VARVECT = samtliga positioner 0
P = diagonalelementen 1, övriga positioner 0
```

REFERENSER

B. Wittenmark, "A self-tuning predictor", IEEE Trans. Automat. Contr., Vol. AC-19, No. 6, pp. 848-851, Dec 1974.

K.J. Åström och B. Wittenmark," On self-tuning regulators", Automatica, Vol. 9, pp. 185-199, 1973.

J. Holst, "Adaptive prediction and recursive estimation", Inst. för Reglerteknik, LTH, Lund 1977.

H. Elmqvist, S.E. Mattsson och G. Olsson, "Datorer i reglersystem : Realtids-programmering", Inst. för Reglerteknik, LTH, Lund 1981.

Bilaga A

HÄRLEDNINGAR

Modellen.

Modellen är en dynamisk, tidsdiskret, MISO med linjära parametrar :

$$A(q^{-1})y(k) = B(q^{-1})u(k) + C(q^{-1})e(k) + K \qquad (1)$$

där
- $y(k)$ = utsignal från processen
- $u(k)$ = insignal till processen
- $e(k)$ = ej mätbart brus
- K = konstantterm

$$A(q^{-1}) = 1 + a_1 q^{-1} + \ldots + a_{na} q^{-na}$$

$$B(q^{-1}) = b_1 q^{-1} + \ldots + b_{nb} q^{-nb}$$

$$C(q^{-1}) = 1 + c_1 q^{-1} + \ldots + c_{nc} q^{-nc} .$$

Om det finns flera insignaler, eller mätbara störningar, kommer dessa in på samma sätt som u.

Prediktorn.

Definiera

$$F(q^{-1}) = 1 + f_1 q^{-1} + \ldots + f_{m-1} q^{-m+1} \quad \text{och}$$

$$G(q^{-1}) = g_0 + g_1 q^{-1} + \ldots + g_{n-1} q^{-n+1} .$$

Modellen kan skrivas om som (argumentet q^{-1} utelämnas hädanefter) :

$$Ay(k+m) = q^m Bu(k) + Ce(k+m) + q^m K$$

Bilaga A

Skriv C/A som

$$C/A\ e(k+m) = e(k+m) + f_1 e(k+m-1) + \ldots + f_{m-1} e(k+1) +$$

$$+ f_m e(k) + f_{m+1} e(k-1) + \ldots \qquad (2)$$

$e(k+m) + \ldots + f_{m-1} e(k+1)$ predikteras med sitt medelvärde, dvs 0. Om serieutvecklingen i (3) avbryts efter $f_{m-1} e(k+1)$ kan resten skrivas om som $q^{-m}G/A$ dvs

$$C = AF + q^{-m}G \ .$$

M-stegsprediktionen blir alltså

$$\hat{Ay}(k+m|k) = q^m Bu(k) + Ge(k) + q^m K \qquad (3)$$

e(k) kan beräknas tack vare kännedom om gamla värden på y och u. Från (1) får vi

$$Ce(k) = Ay(k) - Bu(k) - K$$

För att prediktorn skall bli stabil krävs det att C är stabilt. Sätt in detta i (3), och vi får

$$\hat{Cy}(k+m|k) = q^m BFu(k) + Gy(k) + q^m FK$$

K-stegsprediktorn använder denna formel direkt, medan multistegsprediktorn använder formeln rekursivt med m = 1 enligt

Bilaga A

$$\hat{y}(k+m|k) + c_1\hat{y}(k+m-1|k) + \ldots + c_{m-1}\hat{y}(k+1|k) +$$

$$+ c_m\hat{y}(k|k-1) + \ldots + c_n\hat{y}(k+m-n|k+m-n-1) =$$

$$= g_0\hat{y}(k+m-1|k) + \ldots + g_{m-2}\hat{y}(k+1|k) +$$

$$+ g_{m-2}y(t) + \ldots + g_{n-1}y(k+m-n) +$$

$$+ BFu(k+m) + q^m FK \quad \text{för } k <= n \text{ och}$$

$$\hat{y}(k+m|k) + \ldots + c_n\hat{y}(k+m-n|k) =$$

$$= g_0\hat{y}(k+m-1|k) + \ldots + g_{n-1}\hat{y}(k+m-n|k) +$$

$$+ BFu(k+m) + q^m FK \quad \text{för } k > n.$$

<u>Parameterskattaren</u>

Modellen kan skrivas om som

$$y(k) = (1-A)y(k) + Bu(k) + (C-1)e(k) + e(k) + K_b$$

där

$$1-A = -a_1 q^{-1} - \ldots - a_{na} q^{-na}$$

$$C-1 = c_1 q^{-1} + \ldots + c_{nc} q^{-nc}$$

K_b = konstant.

Definiera

$$\hat{\theta}(k-1) = [\ \hat{a}_1 \ \ldots\ \hat{a}_{na}\ \hat{b}_1 \ \ldots\ \hat{b}_{nb}\ \hat{c}_1 \ \ldots\ \hat{c}_{nc}\ \hat{K}_b\ ]^T$$

$$\varphi(k-1) = [\ -y(k-1) \ \ldots\ -y(k-na)\ u(k-1) \ \ldots\ u(k-nb)$$
$$\hat{e}(k-1|k-1) \ldots\ e(k-nc\,|k-nc)\ 1\ ]^T$$

Bruset e(k) ... skattas som en sekvens av enstegsprediktionsfel $\hat{e}(k|k)$ ... .

$$\hat{y}(k|k-1) = \hat{\theta}^T(k-1)\varphi(k)$$

$$\hat{e}(k|k) = y(k) - \hat{\theta}^T(k)\varphi(k)$$

Bilaga A

Arbetsgång:

1) Beräkna $\hat{y}(k|k-1)$.

2) Uppdatera K enligt

$$K(k) = P(k-1)\varphi(k)[\ 1 + \varphi(k)^T P(k-1)\varphi(k)\ ]^{-1} .$$

3) Uppdatera θ enligt

$$\hat{\theta}(k) = \hat{\theta}(k-1) + K(k)[\ y(k) - \hat{y}(k|k-1)\ ] .$$

4) Uppdatera P enligt

$$P(k)=P(k-1)-P(k-1)\varphi(k)[1+\varphi(k)^T P(k-1)\varphi(k)]^{-1}\varphi(k)^T P(k-1) .$$

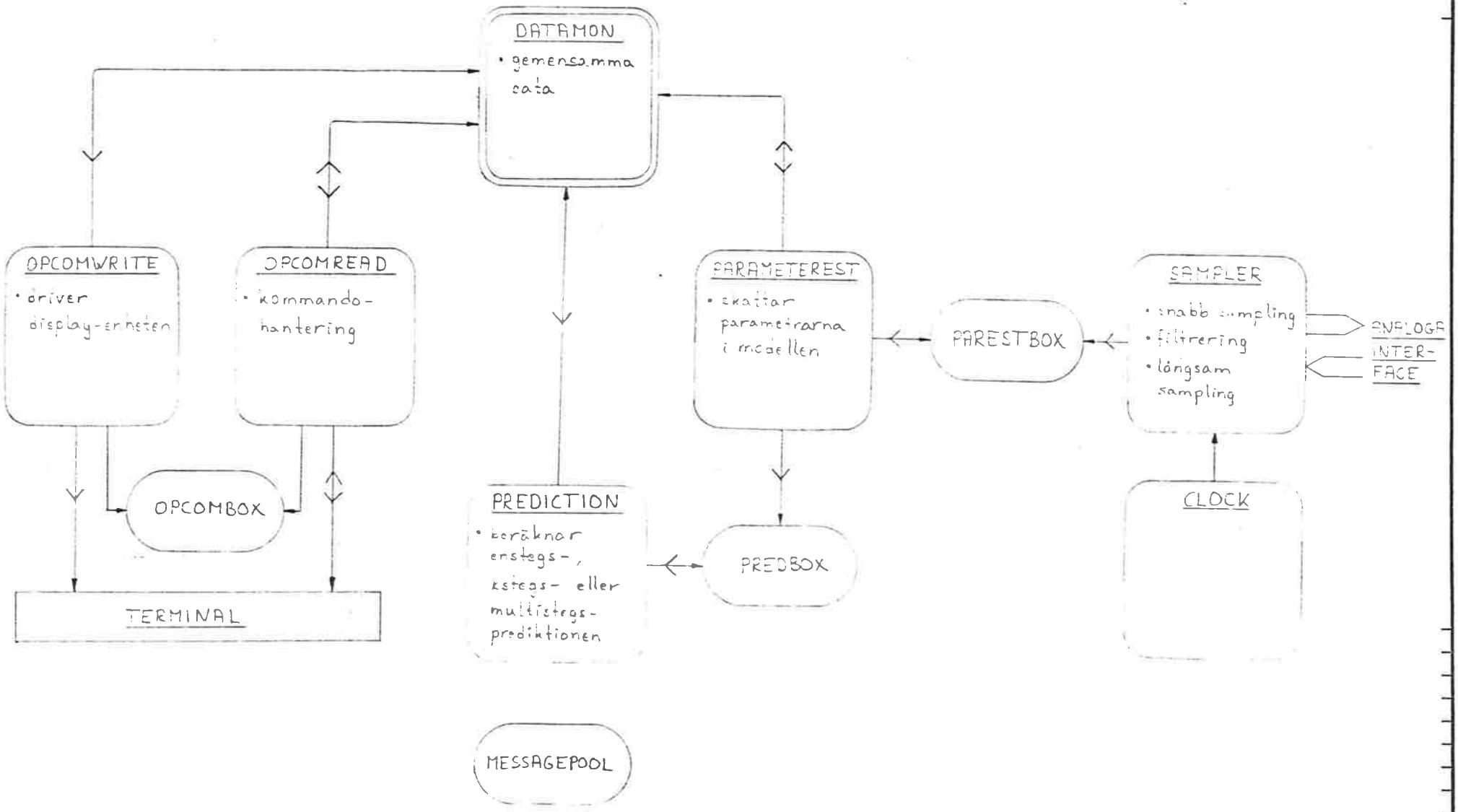
5) Beräkna $\hat{e}(k|k)$.

6) Uppdatera φ med $y(k)$, $u(k)$ och $\hat{e}(k|k)$.

K är en förstärkningsvektor som anger hur mycket parametrarna i θ-polynomet skall uppdateras. P är kovariansmatrisen. I $\hat{\theta}$-vektorn finns nu skattningarna av koefficienterna i A, B och C-polynomen som kan användas för prediktion och reglering.

Orienteringsruta	Ändringen omfattar Alteration description	Ändr v Date	Sign	sid. 17

BILAGA 6



Pos	Antal Quantity	Artikelnummer Item no	Benämning Description	Data	Material	Anm Note

Kockumation

SJÄLVINSTÄLLANDE PREDIKTOR

PROCESSGRAF

Skala	Upprättad av	Kontrollerad	Granskad av

Ort	År — vecka	Ev ordernr	Status	Blad	Forts	Dokumentnummer och typ	Rev
				1		—	

Dokumentnummer och typ

1

Bilaga C

SYNKRONISERING AV PROCESSERNA

Synkroniseringen mellan processerna sker mestadels med meddelanden, men även händelsevariabler och semaforer användes.

Process CLOCK räknar upp en semafor (samplersync) varje gång den har väntat ett samplingsintervall. Process SAMPLER väntar på att samplersync skall bli större än 0. När så sker räknas samplersync ner med 1 och SAMPLER exekveras. Sist i SAMPLER skickas ett meddelande till PARESTBOX. När process PARAMETEREST inte exekverar så väntar den på ett meddelande i PARESTBOX. PARAMETEREST skickar i sin tur ett meddelande till PREDBOX när skattningen av parametrarna är färdig. I PREDBOX väntar process PREDICTION. När PREDICTION har räknat ut prediktionen påverkas en händelsevariabel (dispchange) som talar om för OPCOMWRITE att displayen kan uppdateras.

Vid start av samplingen måste man om hfast=0 starta CLOCK. Det gör man med händelsevariabeln change som sätts i OPCOMREAD då man ändrar hfast till ett värde större än 0. Om hslow=0 skall den långsamma samplingen inte utföras. Det klaras genom att testa på en logisk variabel (stopflag). Om den är true utförs inte samplingen. Stopflag används även då vissa parametrar i programmet ändras via OPCOMREAD. Samplingen stoppas och därigenom stoppas även PARAMETEREST och PREDICTION varvid räknefel undviks. Samplingen startas automatiskt genom att stopflag sätts till false då parameterändringarna är utförda.

För information om realtidskärnan hänvisas till "Datorer i reglersystem : Realtids-programmering" av H. Elmqvist, S.E. Mattson och G. Olsson, Inst. för Reglerteknik, LTH, Lund 1981.

Bilaga D

VARIABELFÖRKLARING

Chanvect : kanalerna som signalerna kopplas till

[1 ... noofu+1 ... maxnoofu+1] [y u1 ... u4]

Parvect : modellens parametrar
Varvect : processens in- och utsignaler
K : viktningsvektor

[1 2 ... noofu+3 ... veclength] [y u1 ... u4 $\hat{e}$]

Pointvect : pekare på polynomen i parvect, varvect och K

[1 ... noofu+3 ... maxnoofu+3] [y u1 ... u4 $\hat{e}$ K]

Gradvect : polynomens gradtal

[1 ... noofu+2 ... maxnoofu+2] [y u1 ... u4 $\hat{e}$]

Tdelvect : u-signalernas tidsfördröjningar

[1 ... noofu ... maxnoofu] [u1 ... u4]

Tdelsave : innehåller de tidsfördröjda u-signalerna

$$\begin{bmatrix} 1 \ldots \text{tdelvect}[1] \ldots \text{maxtimedelay} \\ \vdots \\ \text{noofu} \ldots \\ \vdots \\ \text{maxnoofu} \ldots \text{tdelvect}[\text{maxnoofu}] \ldots \end{bmatrix} \begin{bmatrix} u1 \ldots \\ \vdots \\ u4 \ldots \end{bmatrix}$$

P : kovariansmatris

$$\begin{bmatrix} 1 \quad \ldots \quad \text{noofu+3} \quad \ldots \text{ veclength} \\ \vdots \\ \text{noofu+3} \ldots \\ \vdots \\ \text{veclength} \ldots \end{bmatrix}$$

Bilaga D

Ypredvect : innehåller de predikterade y-värdena

[1 ... predhorizon ... maxpredhorizon]

Typen divvector : diversevektor

[1 ... maxgradC]

Typen savematrix : innehåller de sex senaste värdena av y, u och $\hat{y}$. Ev. tidsfördröjningar ligger lagrade i tdelsave och skiftas in efter hand.

$$\begin{bmatrix} 1 & \dots & \text{maxgradC} \\ \vdots & & \vdots \\ \text{noofu+2} & \dots & \\ \vdots & & \\ \text{maxnoofu+2} & \dots & \end{bmatrix} \qquad \begin{bmatrix} y(k) & \dots & y(k-5) \\ u1(k) & \dots & u1(k-5) \\ \vdots & & \vdots \\ u4(k) & \dots & u4(k-5) \\ \hat{y}(k+1|k) & \dots & \hat{y}(k-4|k-5) \end{bmatrix}$$

Orienteringsruta	Ändringen omfattar / Alteration description	Ändr v Date	Sign	sid. 21

BILAGA E

SJÄLVINSTÄLLANDE PREDIKTOR

Adress	Fram:..... / Back:.....	Sampling	..... sek / enligt ingång :	Avbrott	Nivå: / enligt flagga : ST.....	Ref.nr.	PROGRAM NR..... / NAMN (ID): CLOCK

CHANGE

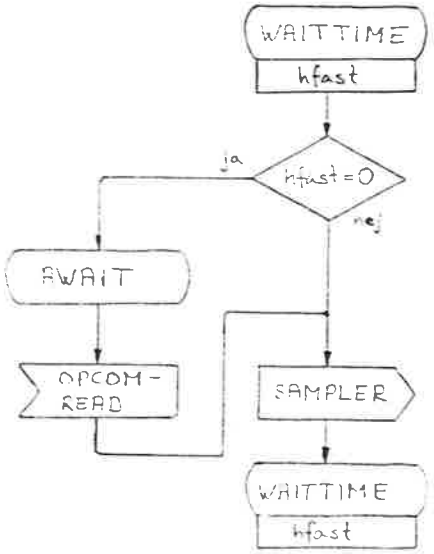
BOX INDATA

BOX INSTATUS

BOX UTDATA

SAMPLER-SYNC

BOX UTSTATUS



INDATA

INTERNDATA

STATUS

UT DATA

INTERNDATA

RA-STATUS

Parametrar

HFAST

Prog. variabler

Dokumentnummer och typ

Pos	Antal Quantity	Artikelnummer Item no	Benämning Description	Data	Material	Anm Note

Kockumation

SJÄLVINSTÄLLANDE PREDIKTOR

PROCESS CLOCK

Skala	Upprättad av	Kostn.ställe 630	Granskad av

Ort	År — vecka	Ev ordernr	Status	Blad	Forts	Dokumentnummer och typ	Rev
MALMÖ	8224			1		252.01. — FB	

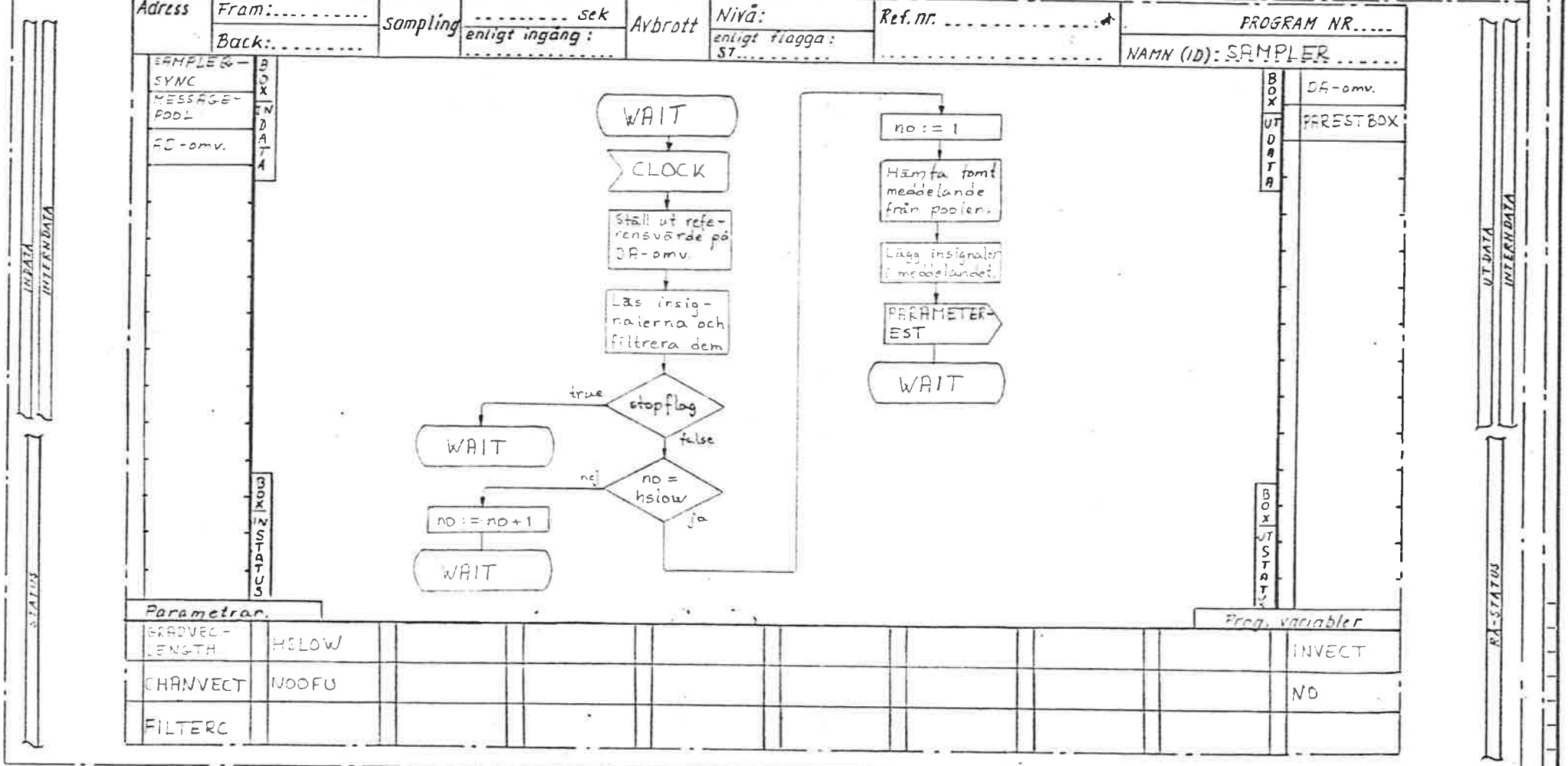
Ändringsruta	Ändringen omfattar / Alteration description	Ändr v / Date	Sign	Sid.

BILAGA F

SÄLVINSTÄLLANDE PREDIKTOR

DATA INSAMLINGSENHET

Adress | Fram:..... | Back:..... | Sampling ..... sek enligt ingång : ..... | Avbrott | Nivå: enligt flagga : S7..... | Ref.nr. ..... | PROGRAM NR..... | NAMN (ID): SAMPLER .....



Parametrar.		Prog. variabler
GRADVEC-LENGTH	HSLOW	INVECT
CHANVECT	NOOFU	NO
FILTERC		

Pos	Antal Quantity	Artikelnummer Item no	Benämning Description	Data	Material	Anm Note

Kockumation

SÄLVINSTÄLLANDE PREDIKTOR

PROCESS SAMPLER

Satt	Upprättad av	Kostn ställe	Granskad av
		630	

Ort	Ar — vecka	Ev ordernr	Status	Blad / Forts	Dokumentnummer och typ	Rev
MALMÖ	8224			1 | 1	252.01. — FB	

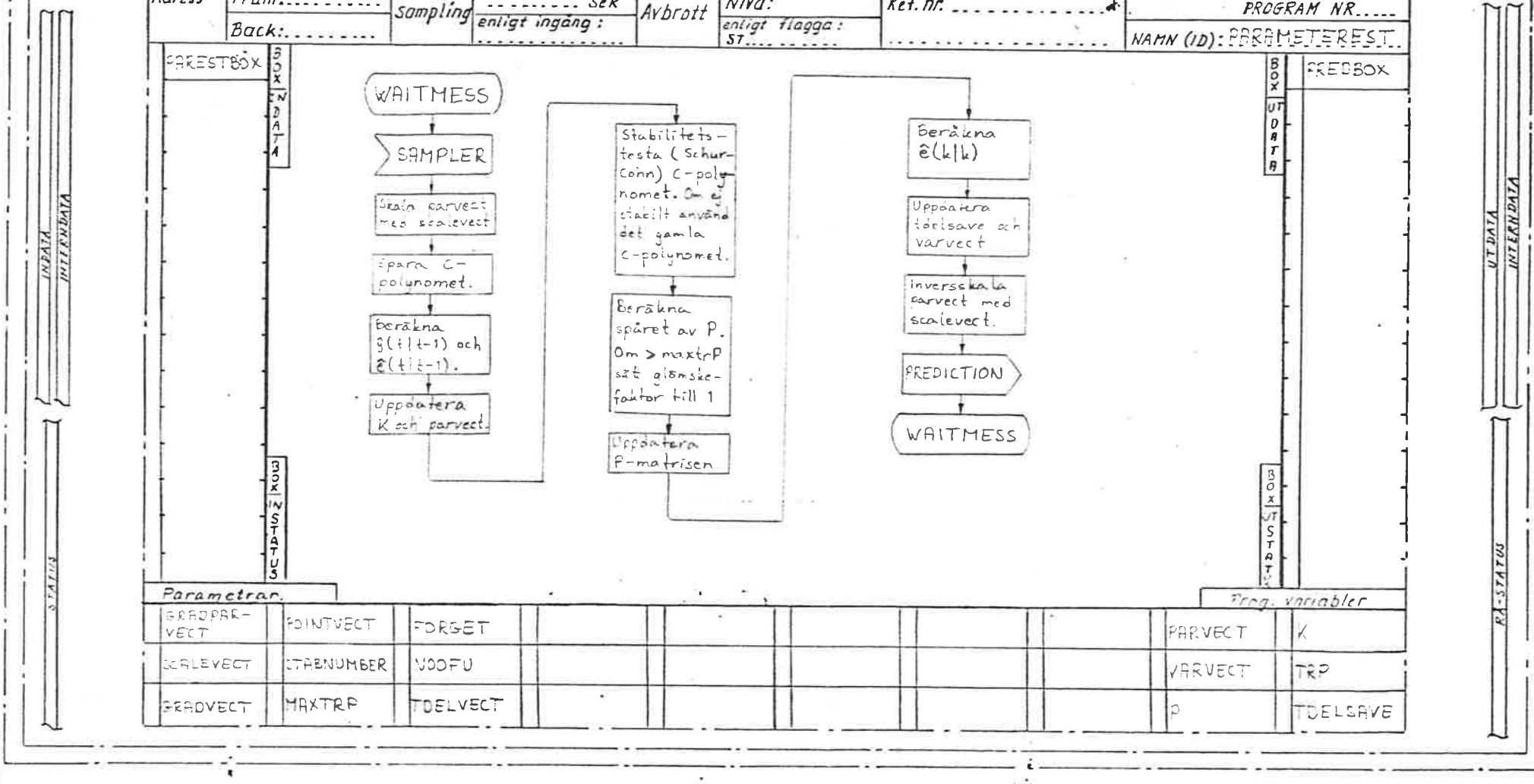
Orienteringsruta	Ändringen omfattar / Alteration description	Ändr v / Date	Sign	sid. 13

BILAGA G

SJÄLVINSTÄLLANDE PREDIKTOR

DATA BEARBETNINGSENHET

Adress Fram:..... Back:..... | Sampling ..... sek enligt ingång : ..... | Avbrott | Nivå: enligt flagga : ST..... | Ref. nr. ..... | PROGRAM NR..... NAMN (ID): PARAMETEREST.



Parametrar.

GRADPAR-VECT	POINTVECT	FORGET
SCALEVECT	STABNUMBER	VOOFU
GRADVECT	MAXTRP	TDELVECT

Prog. variabler

PARVECT	K
VARVECT	TRP
P	TDELSAVE

Dokumentnummer och typ

1

Pos	Antal Quantity	Artikelnummer Item no	Benämning Description	Data	Material	Anm Note

Kockumation			SJÄLVINSTÄLLANDE PREDIKTOR	
Skala	Upprättad av	Ritnr skala 630	Granskad av	PROCESS PARAMETEREST
Ort MALMÖ	År — vecka 8224	Ev ordernr	Status / Blad 1 / Forts	Dokumentnummer och typ 252.01. —FB / Rev

BILAGA H

SJÄLVINSTÄLLANDE PREDIKTOR

DATA BEARBETNINGSENHET

Adress | Fram:..... | Back:..... | Sampling | ..... sek enligt ingång : ..... | Avbrott | Nivå: enligt flagga : ST..... | Ref.nr. ..... | PROGRAM NR..... | NAMN (ID): PREDICTION

PREDBOX

BOX INDATA

WAITMESS

PARAMETER-EST

Aktuella insignaler sparas i savemat

Processens senaste utsignal läses in i olduv.

Förberedd oldypredv för inskrift av nästa prediktion

onestep | k-step | multistep

Beräkna G- och F-polynomen

Beräkna B&F

Beräkna enstegs-prediktionen och lägg den i ypredvect

Spara prediktionen i oldypredvect

Ställ ut prediktionen på DA-omv.

Beräkna G- och F-polynomen. Beräkna B&F.

Skifta in ingsignalerna i mät. Beräkna k-stegs-prediktionen och lägg den i ypredvect.

Spara prediktionen i oldypredvect samt ställ ut värdet på DA-omv.

Beräkna G- och F-polynomen. Beräkna B&F.

Skifta in insignalerna i m. Beräkna multistegs-prediktionen och lägg dem i ypredvect.

Spara prediktion nr. k+1 och lagra den i oldypredv.

Ställ ut prediktion nr k+1 på DA-omv

OPCOM-WRITE

WAITMESS

BOX UTDATA

DISPCHANGE

BOX INSTATUS

BOX UTSTATUS

INDATA | INTERNDATA | STATUS

UTDATA | INTERNDATA | RA-STATUS

Parametrar

GRAD-VECLENGTH	MEXPRED-HORIZON	GRADVECT	TOELVECT					SAVEMAT	OLDYPREDV
MAXGRADC	PRED	POINTVECT	TOELSAVE						OLDYV
NDOFU	PREDHORIZON	PARVECT							YPREDVECT

Prog. variabler

Dokumentnummer och typ

Pos	Antal Quantity	Artikelnummer Item no	Benämning Description	Data	Material	Anm Note

Kockumation

SJÄLVINSTÄLLANDE PREDIKTOR

PROCESS PREDICTION

Skala	Upprättad av	Kostn.ställe 630	Granskad av
Or MALMÖ	År — vecka 8224	Ev.ordernr	Status / Blad 1 / Forts

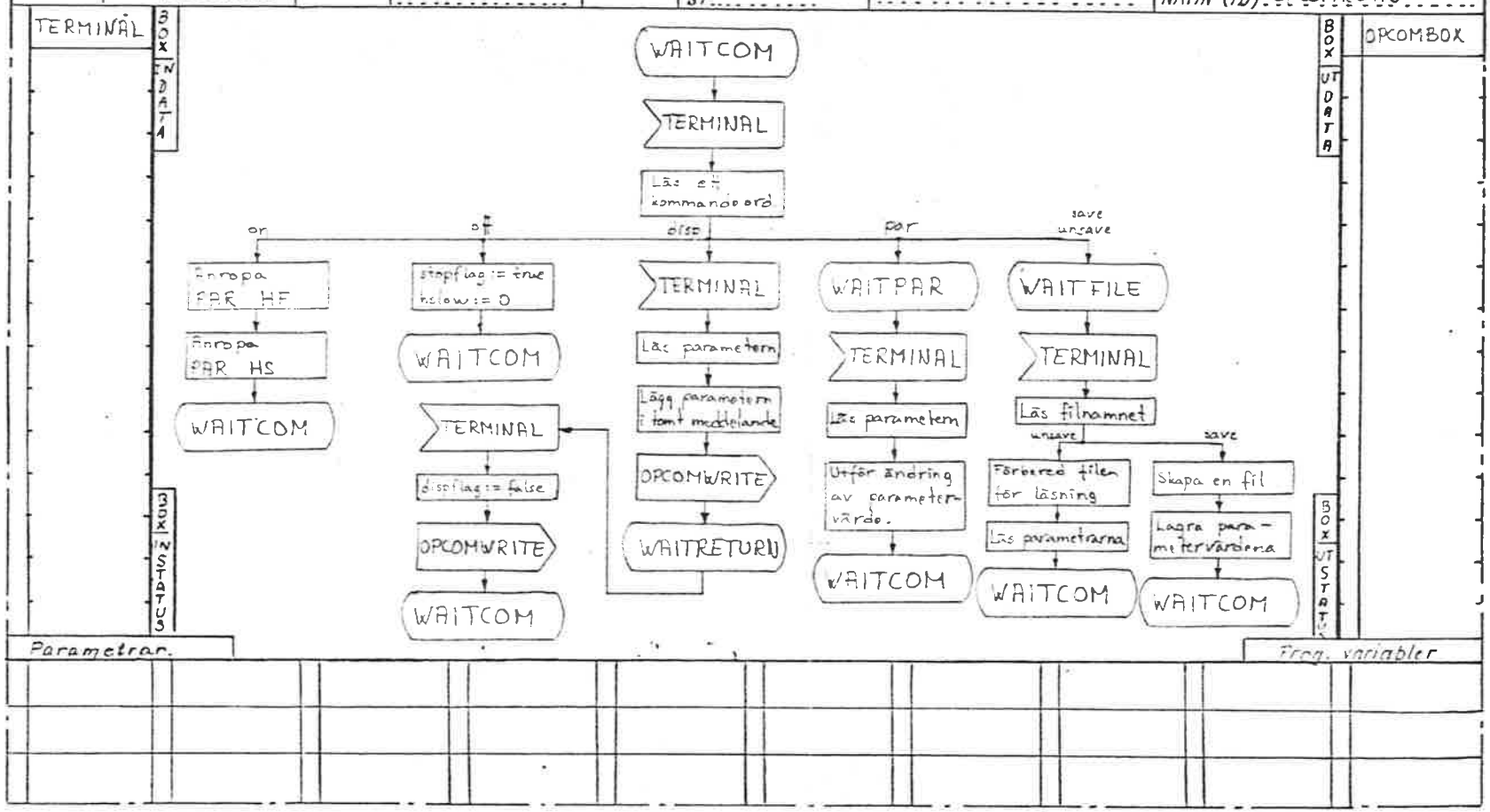
Dokumentnummer och typ: 252.01. —FB

Rev

SJÄLVINSTÄLLANDE PREDIKTOR

OPERATÖRSKOMMUNIKATION

Adress: Fram:..... Back:..... | Sampling: ..... sek, enligt ingång: ..... | Avbrott | Nivå: enligt flagga: ST..... | Ref.nr. ..... | PROGRAM NR..... NAMN (ID): OPCOMREAD.....



Pos	Antal Quantity	Artikelnummer Item no	Benämning Description	Data	Material	Anm Note

Kockumation

SJÄLVINSTÄLLANDE PREDIKTOR

PROCESS OPCOMREAD

Skala	Upprättad av	Kostn ställe	Granskad av
		630	

Ort	År – vecka	Ev ändring	Status	Blad	Forts	Dokumentnummer och typ	Rev
MALMÖ	8224					252.01. –FB	

Dokumentnummer och typ

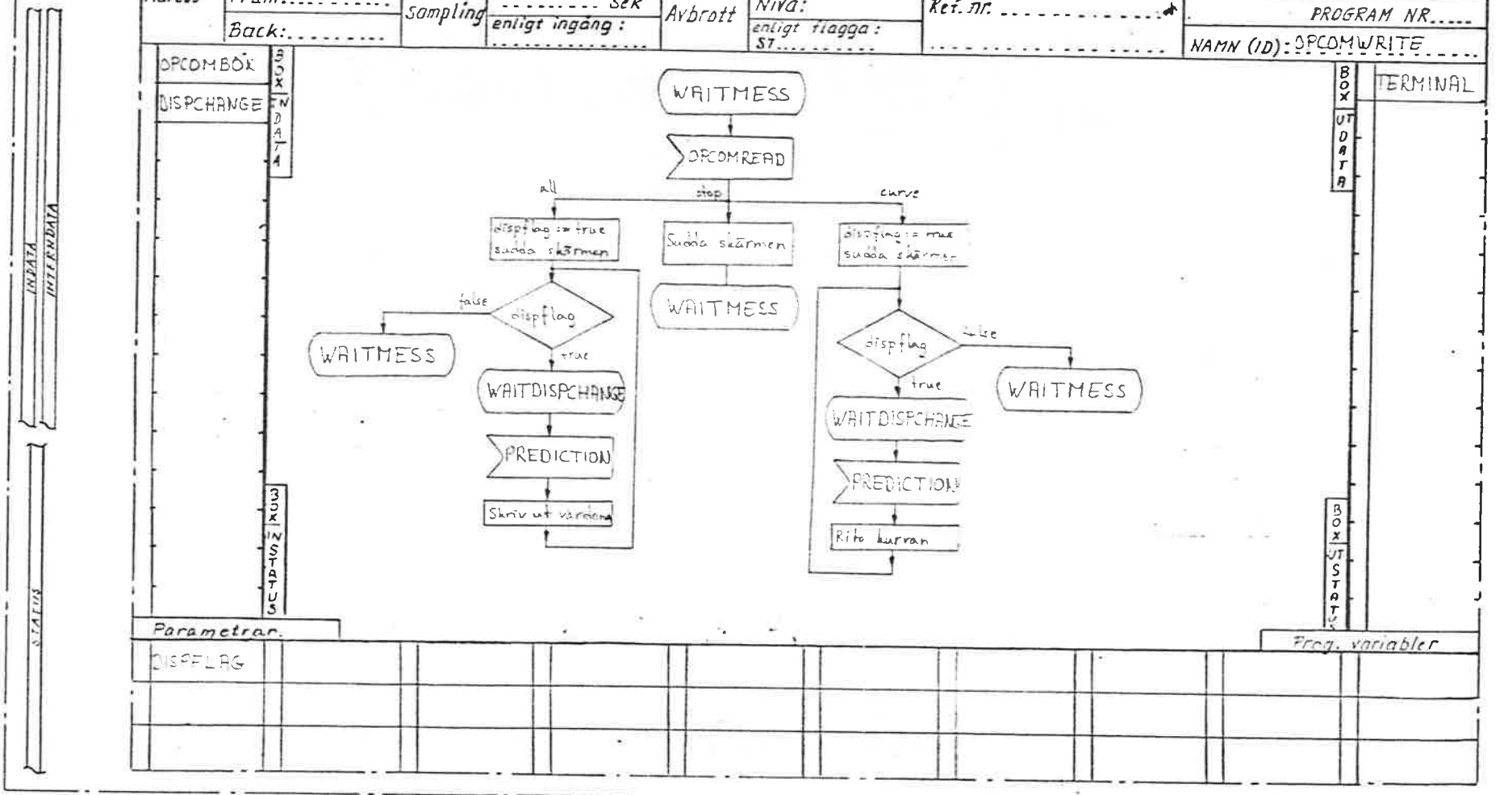
Ort	Ändringens omfattar / Alteration description	Ändr v / Date	Sign	sid. 26

BILAGA 3

SJÄLVINSTÄLLANDE PREDIKTOR

OPERATÖRSKOMMUNIKATION

Adress: Fram:..... Back:.....
Sampling: ..... sek enligt ingång: .....
Avbrott
Nivå: enligt flagga: S7.....
Ref. nr: .....
PROGRAM NR.....
NAMN (ID): OPCOMWRITE.....



Pos	Antal / Quantity	Artikelnummer / Item no	Benämning / Description	Data	Material	Anm / Note

Kockumation

SJÄLVINSTÄLLANDE PREDIKTOR
PROCESS OPCOMWRITE

Skala	Ritad av	Kodn ställe: 630	Granskad av

Ort	År — vecka	Ev ordernr	Status	Blad	Forts	Dokumentnummer och typ	Rev
MALMÖ	8224			1		252.01. —FB	

Dokumentnummer och typ

Bilaga K

KOMPILERING OCH LÄNKNING.

För att underlätta ändringar är programmet uppdelat i block som ligger i separata filer. Dessa är

```
DEKL   .PAS : deklarationer
INIT   .PAS : initialiseringar
PRED   .PAS : prediktorn
PAREST.PAS : parameterskattaren
SAMP   .PAS : samplaren och klockan
OPCR   .PAS : kommandohanteringen (rubout, error,
              skipblanks, checkend, getcom)
OPCPAR.PAS : kommandohanteringen (par)
OPCM   .PAS : kommandohanteringen (on, off,
              filehandler, disp, main OPCOMREAD)
OPCW   .PAS : displayenheten (eraseterm, all,
              curve)
MAIN   .PAS : huvudprogrammet.
```

Programmet är så stort att allt inte får plats samtidigt i primärminnet. Därför används overlay-teknik, vilket innebär att minnet delas in i, i detta fallet, två areor. Programmet delas upp i tre delar. En av delarna, roten, består av huvudprogrammet. Roten läggs i en av areorna i minnet. I roten anropas initialiseringsbiten som laddas i den återstående arean. När initialiseringarna är avklarade behövs inte denna del av programmet mer, varför roten kan anropa resten av programmet som laddas ovanpå initaliseringsbiten. Roten finns i MAIN.PAS, initialiseringsbiten i INIT.PAS och övriga filer är de som laddas sist. De tre programdelarna måste kompileras var för sig. Tre kommandofiler har skapats för att underlätta kompileringen. KI.COM, KM.COM, KLR.COM.

```
KI.COM :
   R PASCAL
   INIT,TT:/N=DX1:PREOPC,DX1:DEKL,DX1:INIT
   MACRO INIT
```

Bilaga K

```
KM.COM :
   R PASCAL
   MAIN,TT:/N=DX1:PREOPC,DX1:DEKL,DX1:MAIN
   MACRO MAIN

KLR.COM :
   COPY/CONC DX1:(PRED.PAS,PAREST.PAS,SAMP.PAS,
                  OPCR.PAS) DX1:STP1.PAS
   COPY/CONC DX1:(STP1.PAS,OPCPAR.PAS,OPCM.PAS,
                  OPCW.PAS) DX1:STP.PAS
   R PASCAL
   STP[-1],TT:/N=DX1:PREOPC,DX1:DEKL,DX1:STP
   MACRO STP
```

Länkningskommando finns också på en kommandofil.

```
LINK.COM :
   R LINK
   MAIN=MAIN,DX1:TERM2,DX1:KERNEL,DX1:PASLIB/C
   INIT/O:1/C
   MAIN/O:1
```

PREOPC innehåller externdeklarationer till realtidskärnan och terminalhanteraren. TERM2 är terminalhanteraren och KERNEL innehåller realtidskärnan.

En kommandofil exekveras genom att trycka "@DX1:NAMN". Programmet exekveras med kommandot R MAIN.

TEST

TEST_MOT_SALTPROCESSEN_OCH_ANALOGIMASKINEN

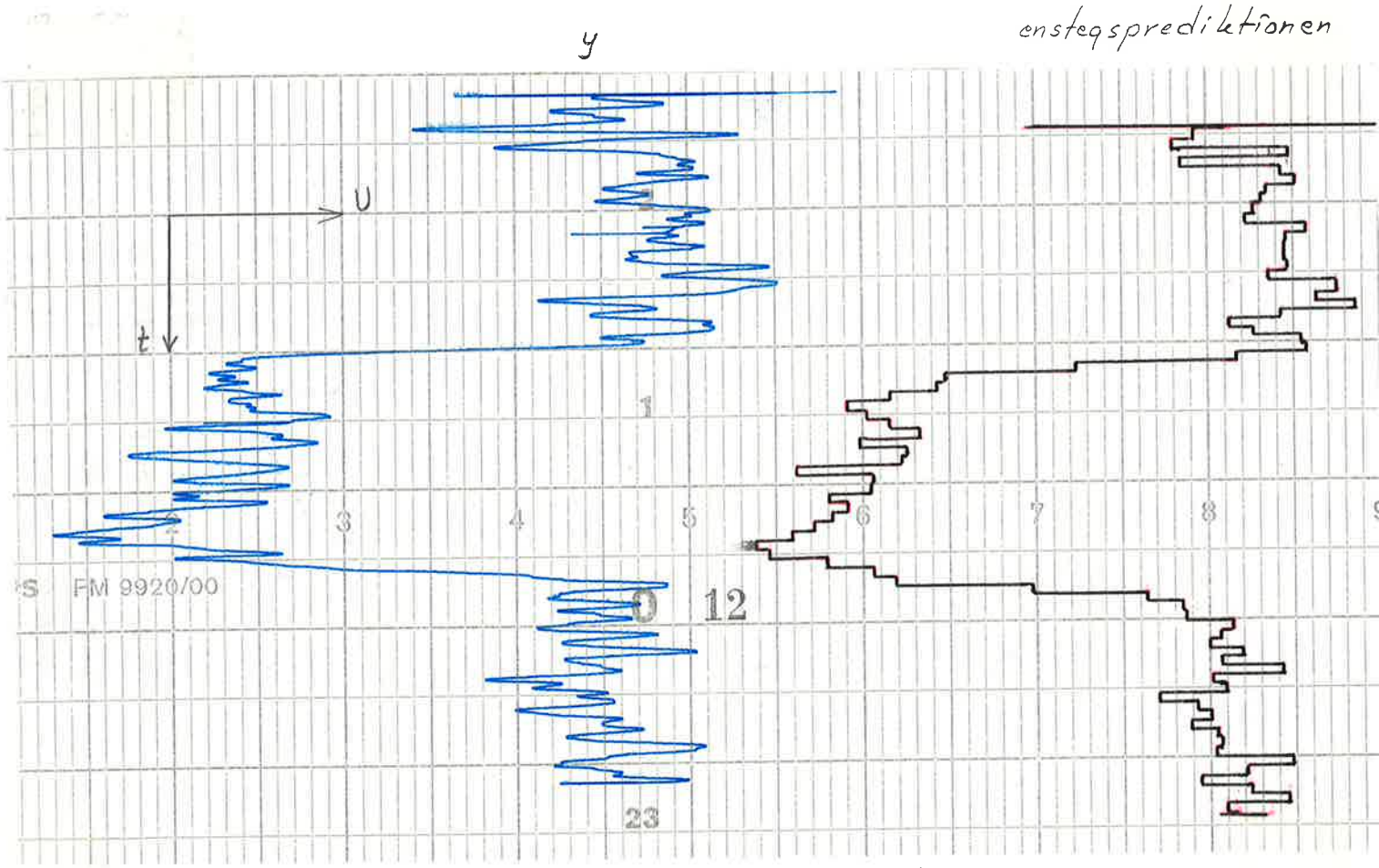
Saltblandningsprocessen

gradA = gradB = gradC = 3

parvect =[-0.551 (a_1) -0.243 (a_2) 0.046 (a_3)

0.257 (b_1) 0.509 (b_2) 0.242 (b_3)

0.128 (c_1) -0.062 (c_2) -0.002 (c_3) 0.006 (K_b)]



ensteg spredikt ionen

0.5 V/ruta 2cm/min flöde = 0.5 l/min

ONESTEPPRED.

TEST

Analogimaskinen

Uppkopplat : $H(s) = 1 / (s + 1)$

$$H(q^{-1}) = (1 - e^{-4})q^{-1} / (1 - e^{-4} q^{-1}) =$$

$$= 0.98 q^{-1} / (1 - 0.018 q^{-1})$$

Körning 1 :

gradA = gradB = gradC = 2 h = 4 sec

parvect = w = [$\underset{a_1}{-0.302}$ $\underset{a_2}{-0.072}$ $\underset{b_1}{0.486}$ $\underset{b_2}{0.057}$

$\underset{c_1}{0.255}$ $\underset{c_2}{0.069}$ $\underset{K_b}{0.026}$]

Körning 2 :

gradA = gradB = gradC = 1 h = 4 sec

parvect = [$\underset{a_1}{-0.399}$ $\underset{b_1}{0.463}$ $\underset{c_1}{0.151}$ $\underset{K_b}{0.043}$] =>

$$(1 - 0.399)y(k) = 0.463u(k-1) + 0.151e(k) + 0.043$$

DEKLARATIONER

```
const   comlength          = 8;
        veclength          = 20;
        maxnoofu           = 4;
        maxgradC           = 6;
        chanveclength      = 5;                (* = maxnoofu + 1 *)
        gradveclength      = 6;                (* = maxnoofu + 2 *)
        pointveclength     = 7;                (* = maxnoofu + 3 *)
        maxtimedelay       = 20;
        maxpredhorizon     = 18;
        maxBFlength        = 25;
        maxsavey           = 20;
        stabnumber         = 3;


type    chanvector       = array[1..chanveclength] of integer;
        inputvector      = array[1..gradveclength] of real;
        gradvector       = array[1..gradveclength] of integer;
        pointvector      = array[1..pointveclength] of integer;
        timedelayvector  = array[1..maxnoofu] of integer;
        vector           = array[1..veclength] of real;
        matrix           = array[1..veclength,1..veclength] of real;
        timedelaysave    = array[1..maxnoofu,1..maxtimedelay] of real;
        savematrix       = array[1..gradveclength,1..maxgradC] of real;
        predvector       = array[1..maxpredhorizon] of real;
        divvector        = array[1..maxgradC] of real;

        estimator         = ( els0 , els2 , rml );
        predictor         = ( onestep , kstep , multistep );

        datamonitor = record
           mutexdata     : semaphore;
           dispchange    : event;      { Syncronises the displayunit. }
           parvect ,                  { Used in the estimation of the }
           varvect ,                   { a, b and c-polynomials       }
           k             : vector;
           p             : matrix;
           tdelsave     : timedelaysave;      {Contains the u-inputs }
                                              { during the timedelay }
        end;

        command = ( onx,offx,parx,dispx,savex,unsavex,lastcomx );

        parameters = ( hfp,hsp,noofup,chanp,filterconstp,parvectp,
                       scalep, gradp, tdelvectp, forgetp, tracep,
                       estp, predp, predhorizonp, kstepp, outpp,
                       posp, limitp, wronghelp, lastparp );

        display    = ( alld , stopd , curved , lastpard );

        messagetype  = ( parest , regul , opcom );
        message      = record
           nextmess  : messageref;
           case messtype : messagetype of
              parest : ( messvect : inputvector );
              opcom  : ( messdisp : display );
        end;
```

```
        identtype       = array[ 1..comlength ] of char;
        commnametype    = array[ command ] of identtype;
        parnametype     = array[ parameters ] of identtype;
        dispnametype    = array[ display ] of identtype;
var     datamon         : datamonitor;
        samplersync     : semaphore;      { Syncronises the sampling-   }
                                          { process to the clock.       }
        mutex           : semaphore;      { Used in process Clock.      }
        change          : event;          {          --  --            }
        messagepool ,
        parestbox ,
        predbox ,
        opcombox ,
        regulbox        : mailbox;
        hfast ,                           { Sampleperiod.               }
        hslow           : integer;        { In no of sampleperiods.     }
        filterc         : real;
        noofu           : integer;        { Number of u-inputs.         }
        chanvect        : chanvector;     { Inputchannelnumbers.        }
        pointvect       : pointvector;    { Pointers on polynomials     }
                                          { in parvect and varvect.     }
        gradvect        : gradvector;     { Degree of polynomials.      }
        gradparvect     : integer;        { Total degree of polynomials.}
        scalevect       : vector;
        tdelvect        : timedelayvector;    { in no of sampleperiods.}
        forget ,
        maxtrP ,
        trP             : real;               { Trace of P-matrix.      }
        ypredvect ,                       { Contains the predicted      }
                                          { y-values.                   }
        oldypredv ,                       { Old k-step predictions.     }
        oldyv           : predvector;     { Old y-values.               }
        est             : estimator;      { Estimationmethod.           }
        pred            : predictor;      { Type of predictor.          }
        predhorizon     : integer;        { How far to predict.         }
        lowpos ,                          { Colposition for the lowest  }
        highpos         : integer;        { and highest curvevalues.    }
        lowlim ,                          { Array bounds for the curve- }
        highlim         : real;           { values.                     }
        kst             : integer;        { The k-stepprediction to be  }
                                          { displayed.                  }
        outp            : real;           { Referencevalue.             }
        dispflag        : boolean;        { Syncronises the display-    }
                                          { unit to commands given from }
                                          { the terminal.               }
        stopflag        : boolean;        { Stops process parameterest  }
                                          { while parameters are changed}
        commname        : commnametype;
        parname         : parnametype;
        dispname        : dispnametype;
function ADin( chan : integer ): real; external;
procedure DAout( chan : integer ; value : real ); external;
```

INIT

```
(*$E+*)
procedure init;

procedure initvars;
var  i    : integer;
begin
   initsem( mutex , 1 );
   initevent( change , mutex );
   initsem( samplersync , 0 );
   hfast := 50;
   hslow := 0;
   filterc := 0.25;
   noofu := 1;
   for i := 1 to chanveclength do
      chanvect[ i ] := i - 1;
   for i := 1 to noofu+2 do
   begin
      gradvect[ i ] := 3;
      pointvect[ i ] := 1 + (i-1)*3;
   end;
   pointvect[ noofu+3 ] := pointvect[ noofu+2 ] + 3;
   for i := noofu+3 to gradveclength do
      gradvect[ i ] := 0;
   gradparvect := 10;
   for i := noofu+4 to pointveclength do
      pointvect[ i ] := 0;
   for i := 1 to veclength do
      scalevect[ i ] := 1;
   for i := 1 to maxpredhorizon do
   begin
      ypredvect[ i ] := 0;
      oldyv[ i ] := 0;
      oldypredv[ i ] := 0;
   end;
   for i := 1 to maxnoofu do
      tdelvect[ i ] := 0;
   maxtrp := 100;
   trp := 10;
   forget := 0.98;
   est := els0;
   pred := multistep;
   predhorizon := 10;
   stopflag := true;
   lowpos := 25;
   highpos := 75;
   lowlim := 0;
   highlim := 1;
   kst := 1;
   outp := 0;
end;

procedure initdatamon;
var  i , j  : integer;
begin
   with datamon do
```

```
    begin
       initsem( mutexdata , 1 );
       initevent( dispchange , mutexdata );
       for i := 1 to veclength do
          for j := 1 to veclength do
             p[ i , j ] := 0;
       for i := 1 to veclength do
       begin
          parvect[ i ] := 0;
          varvect[ i ] := 0;
          k[ i ] := 0;
          p[ i , i ] := 1;
       end;
       varvect[ pointvect[ noofu+3 ] ] := 1;
       for i := 1 to maxnoofu do
          for j := 1 to maxtimedelay do
             tdelsave[ i , j ] := 0;
    end;
end;

procedure initnames;
begin
   comname[ onx ] :=           'ON      ';
   comname[ offx ] :=          'OFF     ';
   comname[ parx ] :=          'PAR     ';
   comname[ dispx ] :=         'DISP    ';
   comname[ savex ] :=         'SAVE    ';
   comname[ unsavex ] :=       'UNSAVE  ';
   parname[ hfp ] :=           'HF      ';
   parname[ hsp ] :=           'HS      ';
   parname[ noofup ] :=        'NOOFU   ';
   parname[ chanp ] :=         'CHAN    ';
   parname[ filterconstp ] :=  'FILTERC ';
   parname[ parvectp ] :=      'PARV    ';
   parname[ scalep ] :=        'SCALE   ';
   parname[ gradp ] :=         'GRAD    ';
   parname[ forgetp ] :=       'FORGET  ';
   parname[ tracep ] :=        'MAXTRP  ';
   parname[ tdelvectp ] :=     'TDELV   ';
   parname[ estp ] :=          'EST     ';
   parname[ predp ] :=         'PRED    ';
   parname[ predhorizonp ] :=  'PREDHOR ';
   parname[ kstepp ] :=        'KST     ';
   parname[ outpp ] :=         'OUTP    ';
   parname[ posp ] :=          'POS     ';
   parname[ limitp ] :=        'LIM     ';
   dispname[ alld ] :=         'ALL     ';
   dispname[ curved ] :=       'CURVE   ';
end;

begin
   initvars;
   initdatamon;
   initnames;
end;
```

```
PREDICTION
dummy, makemessage, invfilt, countGF


(*$E+*)

procedure dummy;
begin
end;

procedure makemessage( box : mailbox ; no : integer );
var   mess    : messageref;
      i       : integer;
begin
   for i := 1 to no do
   begin
      new( mess );
      sendmessage( box , mess );
   end;
end;     (* makemessage *)

{-------------------------------------------------------------------}
                                                        { PREDICTION }

function invfilt( val , oldval : real ) : real;
begin
   invfilt := ( val - filterc * oldval ) / ( 1 - filterc );
end;

procedure countGF( var G:divvector ; var F:predvector ; ph:integer );
var   i , j , gr : integer;          { Identifies the G- and F-polynom }
begin
   with datamon do
   begin
      for i := 1 to maxgradC do                  { Put zeros in G and F }
         G[ i ] := 0;
      for i := 1 to maxpredhorizon do
         F[ i ] := 0;
      G[ 1 ] := 1;                                        { Initialize G }
      for i := 1 to gradvect[ noofu+2 ] do
         G[ i+1 ] := parvect[ pointvect[ noofu+2 ]+i-1 ];
      for i := 1 to ph do
                                                          { Count G and F }
      begin
         F[ i ] := G[ 1 ];
         for j := 1 to maxgradC-1 do
            if j <= gradvect[ 1 ] then
               G[ j ] := G[ j+1 ] - F[ i ] * parvect[ j ]
            else
               G[ j ] := G[ j+1 ];
      end;
   end;
end;     (*   countGF   *)
```

PREDICTION
countpred

```
function countpred( G : divvector ; F : predvector ;
                    mat : savematrix ; ph : integer ) : real;

type  BFvector  = array [ 1..maxBFlength ] of real;
var   r         : real;
      gr , i ,
      j , l , n : integer;
      BF        : BFvector;

begin                                        { Counts one predictionvalue }
   for i := 1 to maxBFlength do
      BF[ i ] := 0;
   r := 0;
   gr := gradvect[ noofu+2 ] - ph + 1;
   if gr < gradvect[ 1 ] then
      gr := gradvect[ 1 ];
   for i := 1 to gr do
      r := r + G[ i ] * mat[ 1 , i ];
   with datamon do
   begin
      for i := 1 to noofu do     { For the number of inputsignals do }
      begin
         for j := 1 to ph do                                  { Count BF }
            for l := 1 to gradvect[ i+1 ] do
               BF[ j+l-1 ] := BF[ j+l-1 ] +
                              parvect[ pointvect[ i+1 ]+l-1 ] * F[ j ];
        for j:=gradvect[i+1]+ph-1 downto tdelvect[i]+gradvect[i+1] do
         begin
            if tdelvect[ i ] > 0 then          { If time-delay then }
               r := r + BF[ j ] * tdelsave[ i , 1 ]
            else                                                { else }
               r := r + BF[ j ] * mat[ i+1 , 1 ];
         end;
         for j:=tdelvect[i]+gradvect[i+1]-1 downto gradvect[i+1] do
         begin
            if tdelvect[ i ] > 0 then          { If time-delay then }
               r := r + BF[j] * tdelsave[ i , j-gradvect[ i+1 ]+1 ]
            else                                                { else }
               r := r + BF[ j ] * mat[ i+1 , j-gradvect[ i+1 ]+1 ];
         end;
         for j := gradvect[ i+1 ]-1 downto 1 do
               r := r + BF[ j ] * mat[ i+1 , gradvect[ i+1 ]-j ];
      end;
      for i := 1 to gradvect[ noofu+2 ] do
         r := r - parvect[ pointvect[noofu+2]+i-1 ] * mat[noofu+2,i];
      for i := 1 to ph do                                   { F * Konst }
         r := r + F[ i ] * parvect[ pointvect[ noofu+3 ] ];
      countpred := r;                          { The predicted value }
   end;
end;    (*   countpred   *)
```

PREDICTION
multisteppred

```
procedure multisteppred( var mat : savematrix ; ph : integer );

var   G           : divvector;                    { Counts the multistep }
      F           : predvector;                      { predictionvalues }
      m           : savematrix;
      i , j , l   : integer;

begin
   countGF( G , F , 1 );
   with datamon do
   begin
      wait( mutexdata );                          { Count ŷ( k+1 I k ) }
      mat[ noofu+2 , 1 ] := countpred( G , F , mat , 1 );
      ypredvect[ 1 ] := mat[ noofu+2 , 1 ];
      m := mat;
      for i := 2 to ph do          { Count the number of predictions }
      begin
         for j := 1 to noofu+2 do                    { Uppdate m-matrix }
         begin
            for l := maxgradC downto 2 do
               m[ j , l ] := m[ j , l-1 ];
            if j = 1 then
               m[ 1 , 1 ] := ypredvect[ i-1 ]
            else if j < ( noofu+2 ) then
               if tdelvect[ j-1 ] >= ( i-1 ) then
                  m[ j , 1 ] := tdelsave[ j-1 , tdelvect[j-1]-i+2 ]
               else
               begin
                  if tdelvect[ j-1 ] > 0 then
                     m[ j , 1 ] := tdelsave[ j-1 , 1 ];
               end
            else
               m[ j , 1 ] := countpred( G , F , m , 1 );
         end;
         ypredvect[ i ] := m[ noofu+2 , 1 ];  { The predicted value }
      end;
      signal( mutexdata );
   end;
end;    (*   multisteppred   *)
```

```
PREDICTION
ksteppred, process prediction


function ksteppred( mat : savematrix ; ph : integer ) : real;

var   G          : divvector;          { Counts the K-stepprediction }
      F          : predvector;
      i , j , l  : integer;

begin
   countGF( G , F , ph );
   with datamon do
   begin
      wait( mutexdata );
      for i := 1 to noofu do
         for j := 1 to ph do                          { Uppdate m-matrix }
         begin
            for l := maxgradC downto 2 do
               mat[ i+1 , l ] := mat[ i+1 , l-1 ];
            if tdelvect[ i ] >= j then
               mat[ i+1 , 1 ] := tdelsave[ i , tdelvect[ i ]-j+1 ];
         end;
      ksteppred := countpred( G , F , mat , ph );
      signal( mutexdata );
   end;
end;    (*   ksteppred   *)


   (*************************
   *   PROCESS  PREDICTION   *
   *************************)


(* process *) procedure prediction;

var   mess       : messageref;
      savemat    : savematrix; { Contains the six latest values of }
      i , j      : integer;    { y, u and ypred.                   }

begin
   setpriority( 10 );
   for i := 1 to gradveclength do
      for j := 1 to maxgradC do
         savemat[ i , j ] := 0;
```

```
PREDICTION
process prediction


   while true do
   begin
      receivemessage( predbox , mess );  { Wait for sync. message }
      for i := 1 to noofu+2 do                    { Uppdate savemat }
      begin
         for j := maxgradC downto 2 do
            savemat[ i , j ] := savemat[ i , j-1 ];
         if i ( ( noofu+2 ) then
            savemat[ i , 1 ] := mess.messvect[ i ];
      end;
      for i := maxpredhorizon downto 2 do      { Uppdate oldypredv }
      begin                                              { and oldyv }
         oldypredv[ i ] := oldypredv[ i-1 ];
         oldyv[ i ] := oldyv[ i-1 ];
      end;
      oldyv[ 1 ] := mess.messvect[ 1 ];
      case pred of
         onestep     : begin
                          multisteppred( savemat , 1 );
                          oldypredv[ 1 ] := ypredvect[ 1 ];
                          DAout( 1 , ypredvect[ 1 ] );
                       end;
         kstep       : begin
                          for i := maxpredhorizon downto 2 do
                             ypredvect[ i ] := ypredvect[ i-1 ];
                          ypredvect[1]:=ksteppred(savemat,predhorizon);
                          oldypredv[ 1 ] := ypredvect[ predhorizon ];
                          DAout( 1 , ypredvect[ 1 ] );
                       end;
         multistep   : begin
                          multisteppred( savemat , predhorizon );
                          oldypredv[ 1 ] := ypredvect[ kst ];
                          DAout( 1 , ypredvect[ kst ] );
                       end;
      end;
      with datamon do
      begin
         wait( mutexdata );
         cause( dispchange );
         signal( mutexdata );
      end;
      sendmessage( messagepool , mess );
   end;
end;   (*   prediction   *)
```

```
PARAMETEREST
vectmult, matrixmult, PmatrixOK


procedure extendedIs( mess : messageref );

var  yest ,
     eest ,
     forg ,
     sum          : real;
     i , j ,
     gradC ,
     point        : integer;
     oldc         : divvector;
     vec          : vector;

function vectmult( var x , z : vector ; length : integer ) : real;
var  i      : integer;
     m      : real;
begin   (*   vectmult   *)
   m := 0;
   for i := 1 to length do
      m := m + x[ i ] * z[ i ];
   vectmult := m;
end;   (*  vectmult  *)

procedure matrixmult( var x,y:vector;var z:matrix;length:integer );
var   i , j : integer;
      sum   : real;
begin
   for i := 1 to length do
   begin
      sum := 0;
      for j := i to length do
         sum := sum + z[ i , j ] * y[ j ];
      x[ i ] := sum;
   end;
end;   (*   matrixmult   *)

function PmatrixOK : boolean;
begin
   with datamon do
   begin
      trp := 0;
      for i := 1 to gradparvect do              { Count trace of P. }
         trp := trp + p[ i , i ];
      if trp <= maxtrp then
         PmatrixOK := true
      else
         PmatrixOK := false;
   end;
end;   (*   PmatrixOK   *)
```

PARAMETEREST
stabtest

```
procedure stabtest;
var   i , j ,
      number     : integer;
      fi         : real;
      vec1 ,                                        { Contains C-polynom. }
      vec2 ,                                      { Converted C-polynom. }
      vec3       : divvector;              { Contains current C-polynom. }
      stable     : boolean;

begin   (*   stabtest   *)
   with datamon do
   begin
      stable := false;
      number := 0;
      for i := 1 to gradC do
      begin
         vec1[ i ] := parvect[ point+i-1 ];
         vec2[ gradC-i+1 ] := vec1[ i ];                    { Reverse C. }
      end;
      while not stable do
      begin
         stable := true;
         vec3 := vec1;                          { Save current C-polynom. }
         i := 1;
         while ( i <= gradC ) and stable do
         begin
            if vec1[ 1 ] <= 0 then                    { Tests stability. }
               stable := false
            else
            begin
               fi := vec2[ i ] / vec1[ 1 ];
               for j := 1 to gradC - i do
               begin
                  vec1[ j ] := vec1[ j ] - vec2[ j ] * fi;
                  vec2[ gradC-i-j+1 ] := vec1[ j ];
               end;
            end;
            i := i + 1;
         end;
         if not stable then
            if number = stabnumber then
            begin
               vec3 := oldc;
               stable := true;
            end
            else
            begin
               number := number + 1;
               eest := eest / 2;
               for i := 1 to gradC do
               begin
                  vec1[ i ] := vec3[ i ] + k[ point+i-1 ] * eest;
                  vec2[ gradC-i+1 ] := vec1[ i ];
               end;
            end;
```

```
PARAMETEREST
stabtest, uppdate


      end;   (*   while not stable do   *)
      for i := 1 to gradC do
         parvect[ point+i-1 ] := vec3[ i ];     { Put back stable C. }
   end;
end;   (*   stabtest   *)



procedure uppdate( mess : messageref );          { Uppdates tdelsave }
                                                 { and varvect       }

var   i , j    : integer;
      inp      : real;

begin   (*   uppdate   *)
   with datamon , mess do
   begin
      for i := 1 to noofu do                     { Shift in new inputs }
         if tdelvect[ i ] > 0 then               { in tdelsave.        }
         begin
            inp := tdelsave[ i , tdelvect[ i ] ];
            for j := tdelvect[ i ] downto 2 do
               tdelsave[ i , j ] := tdelsave[ i , j-1 ];
            tdelsave[ i , 1 ] := messvect[ i+1 ];
            messvect[ i+1 ] := inp;              { Put timedelayd input }
         end;                                    { in messvect.        }
      messvect[ noofu+2 ] := eest;
      i := noofu + 2;
      for j := gradparvect-1 downto 1 do         { Shift in messvect }
         if j = pointvect[ i ] then              { in varvect.       }
         begin
            if i = 1 then
               varvect[ j ] := -messvect[ 1 ] * scalevect[ 1 ]
            else
               varvect[j] := messvect[i] * scalevect[pointvect[i]];
            i := i - 1;
         end
         else
            varvect[ j ] := varvect[ j-1 ];
   end;
end;   (*   uppdate   *)
```

```
PARAMETEREST
main extendedls


begin   (***     extendedls     ***)
   with datamon , mess do
   begin
      wait( mutexdata );
      for i := 1 to gradparvect do
         parvect[ i ] := parvect[ i ] * scalevect[ i ];
      gradC := gradvect[ noofu+2 ];
      point := pointvect[ noofu+2 ];
      for i := 1 to gradC do                          { Save old C-pol. }
         oldc[ i ] := parvect[ point+i-1 ];
      yest := vectmult( parvect , varvect , gradparvect );
      eest := messvect[ 1 ] - yest;
      matrixmult(vec,varvect,p,gradparvect);  { vec = P * varvect. }
      sum := 1 + vectmult(varvect,vec,gradparvect); { sum = 1 +    }
      for i := 1 to gradparvect do                  { varvect * P  }
                                                    { * varvect.   }
      begin
         k[ i ] := vec[ i ] / sum;
         parvect[i] := parvect[i] + k[i] * eest;  { parvect ready. }
      end;
      stabtest;                          { Checks stability of C and   }
                                         { returns a stable C-polynom. }
      if pmatrixOK then
         forg := forget
      else
         forg := 1;
      for i := 1 to gradparvect do                          { Count p. }
         for j := 1 to gradparvect do
            p[ i , j ] := ( p[ i , j ] - vec[ i ] * k[ j ] ) / forg;
      for i := 1 to gradparvect do                          { Clean P. }
         for j := i to gradparvect do
         begin
            p[ i , j ] := ( p[ i , j ] + p[ j , i ] ) / 2;
            p[ j , i ] := p[ i , j ];
         end;
      yest := vectmult( parvect , varvect , gradparvect );
      eest := messvect[ 1 ] - yest;
      uppdate( mess );             { Uppdates tdelsave and varvect. }
      for i := 1 to gradparvect do
         parvect[ i ] := parvect[ i ] / scalevect[ i ];
      signal( mutexdata );
   end;   (* with datamon , mess do *)
end;   (***     extendedls     ***)
```

PARAMETEREST

```
process parameterest


    (*****************************
    *   PROCESS  PARAMETEREST   *
    *****************************)


(* process *) procedure parameterest;

var  mess    : messageref;

begin   (* main parameterest *)
   setpriority( 8 );
   while true do
   begin
      receivemessage( parestbox , mess );
      case est of
         els0    : extendedls( mess );
         els2    : ;
         rml     : ;
      end;
      sendmessage( predbox , mess );
   end     (*  while  *);
end;   (*  main parameterest  *)
```

SAMPLER

```
   (************************
   *   PROCESS  SAMPLER   *
   ************************)


(* process *) procedure sampler;

var  mess      : messageref;
     i , no    : integer;
     invect    : inputvector;

function filter( val , f : real ) : real;
begin
   filter := filterc * f + ( 1-filterc ) * val;
end;   (*  filter  *)

begin   (*  main sampler  *)
   setpriority( 4 );
   for i := 1 to gradveclength do
      invect[ i ] := 0;
   no := 1;
   while true do
   begin
      wait( samplersync );
      DAout( 0 , outp );
      for i := 1 to noofu+1 do          { Put filtered input in invect }
         invect[i] := filter( ADin( chanvect[i] ) , invect[i] );
      if not stopflag then
         if no = hslow then
         begin
            no := 1;
            receivemessage( messagepool , mess );{Get empty message}
            with mess do
            begin
               messtype := parest;
               for i := 1 to noofu+1 do
                  messvect[ i ] := invect[ i ];     { Put inputs in }
                                                   { message }
            end;
            sendmessage( parestbox , mess );      { Send message to }
         end                                      { process parameterest }
         else
            no := no + 1;
   end;   (*  while true do  *)
end;   (*  main sampler  *)
```

```
CLOCK
process clock, rubout


    (**********************
    *   PROCESS  CLOCK    *
    **********************)


(* process *) procedure clock;

begin    (*  main clock  *)
   setpriority( 2 );
   while true do
   begin
      wait( mutex );
      while hfast = 0 do await( change );
      signal ( mutex );
      signal( samplersync );                { Start process SAMPLER }
      waittime( hfast );
   end;
end;   (*  main clock  *)



procedure rubout( row , col : integer );          { Rubout one row }
begin
   setcursor( row , col );
   erase;
end;
```

```
OPCOMREAD
error, skipblanks


   (*************************
   *   PROCESS  OPCOMREAD   *
   *************************)

(* process *) procedure opcomread;  { Reads commands from terminal }

label   999;
const  blanks  = '          ';
type   errors  = ( toomanyarg,illcom,illpar,fewarg,chanused,
                   tdelwrong,nosamp,nofile,outofrange );
var    commandx     : command;
       comm         : identtype;
       ch           : char;
       i , k , l ,
       wrong ,
       row , col    : integer;
       r            : real;

procedure error( err : errors );
  { Writes errormessages on the commandline.
    The errormessage is removed by return }
begin   (*  error  *)
   setcursor( 23 , 50 );
   case err of
      toomanyarg   : write(' TOO MANY ARGUMENTS');
      illcom       : write(' ILLEGAL COMMAND');
      illpar       : write(' ILLEGAL PARAMETER');
      fewarg       : write(' MISSING ARGUMENTS');
      chanused     : write(' CHANNEL OCCUPIED');
      tdelwrong    : write(' 0 (= TDEL (= ', maxtimedelay:2 );
      nosamp       : write(' SAMPLING IS OFF ');
      nofile       : write(' FILE NOT FOUND ');
      outofrange   : write(' VALUE OUT OF RANGE ');
   end;
   setcursor( 23 , 49 );
   readln;
   read( ch );
end;   (*  error  *)


procedure skipblanks;
begin   (*  skipblanks  *)
   while ( not eoln ) and ( ch = ' ' ) do
      read( ch );
end;   (*  skipblanks  *)
```

```
OPCOMREAD
checkend, getcom


procedure checkend;
  { Checks the lineend }
begin   (*  checkend  *)
   skipblanks;
   if ch <> ' ' then
   begin
      error( toomanyarg );
      goto 999;
   end;
end;    (*  checkend  *)


procedure getcom;
  { Reads one command from the commandline }
var  n : integer;

function upper( ch : char ) : char;
 (* converts to big letter *)
begin   (*  upper  *)
   if ( ch >= 'a' ) and ( ch <= 'z' ) then
      upper := chr( ord( ch ) - 32 )
   else
      upper := ch;
end;    (*  upper  *)

begin   (***  getcom  ***)
   comm := blanks;
   skipblanks;
   n := 1;
   ch := upper( ch );
   while ( ( ch >= 'A' ) and ( ch <= 'Z' ) ) or
         ( ( ch >= '0' ) and ( ch <= '9' ) ) do
   begin
      if n > comlength then
      begin
         error( illpar );
         goto 999;
      end;
      comm[ n ] := ch;
      n := n + 1;
      if eoln then
         ch := ' '
      else
      begin
         read( ch );
         ch := upper( ch );
      end;
   end;    (*  while  *)
end;    (***   getcom   ***)
```

OPCOMREAD
par

```
procedure par;
     { Handles the PAR command :    PAR  parameter }

label  900;
var    param    : parameters;
       i,j,l    : integer;
       r        : real;
       flag     : boolean;

   procedure show;     { Shows the parametervalue on the commandline }
   begin
      rubout( 23 , 2 );
      write( 'PAR   ', parname[ param ] ,'=' );
      case param of
         hfp             : write( hfast:4 );
         hsp             : write( hslow:4 );
         noofup          : write( noofu:1 );
         filterconstp    : write( filterc:5:2 );
         forgetp         : write( forget:4:3 );
         tracep          : write( maxtrp:6:2 );
         predhorizonp    : write( predhorizon:2 );
         kstepp          : write( kst:2 );
         outpp           : write( outp:4:2 );
      end;
      write('     newvalue:=');
   end;     (*   show   *)

   procedure showvect( limit : integer );{Shows the parametervector}
   begin
      rubout( 23 , 2 );
      write('PAR  ', parname[ param ] );
      for i := 1 to limit do
      begin
         setcursor( 19 , i*7 );
         write( i:2 );
         setcursor( 20 , i*7 );
         case param of
            chanp        : write( chanvect[ i ]:2 );
            parvectp     : write( datamon.parvect[ i ]:4:2 );
            scalep       : write( scalevect[ i ]:4:1 );
            gradp        : write( gradvect[ i ]:2 );
            tdelvectp    : write( tdelvect[ i ]:2 );
         end;
      end;
      setcursor( 23 , 18 );
      write('Position:=');
   end;     (*   showvect   *)
```

```
OPCOMREAD
par

    procedure read1( low , high , t : integer );
    begin                                    { t=1 => integer, t=2 => real }
       if t = 1 then                         { Reads new parametersettings }
       begin
          read( i );
          if ( i < low ) or ( i > high ) then
          begin
             error( outofrange );
             param := wronghelp;
             goto 900;
          end;
       end
       else
       begin
          read( r );
          if ( r < low ) or ( r > high ) then
          begin
             error( outofrange );
             param := wronghelp;
             goto 900;
          end;
       end;
    end;     (*    read1    *)

begin    (*    par    *)
    if commandx <> onx then
    begin
       ch := ' ';
       getcom;
       checkend;
    end;
    param := hfp;
    while ( parname[ param ] <> comm ) and ( param <> lastparp ) do
       param := succ( param );
900:  case param of                                  {     Label 900     }
          hfp           : begin
                             show;
                             read1( 0 , maxint , 1 );
                             hfast := i;
                             if hfast > 0 then
                             begin
                                wait( mutex );
                                cause( change );
                                signal( mutex );
                             end;
                          end;
          hsp           : begin
                             show;
                             read1( 0 , maxint , 1 );
                             hslow := i;
                             if hslow > 0 then
                                stopflag := false
                             else
                                stopflag := true;
                          end;
```

```
OPCOMREAD
par

          noofup        : begin
                             show;
                             read1( 0 , maxnoofu , 1 );
                             noofu := i;
                             param := gradp;
                             goto 900;
                          end;
          chanp         : begin
                             showvect( noofu+1 );
                             read1( 1 , noofu+1 , 1 );
                             setcursor( 23 , 35 );
                             write('Newvalue:=');
                             read( j );
                             flag := false;
                             for l := 1 to noofu+1 do
                                if (j=chanvect[l]) and (i<>l) then
                                   flag := true;
                             if flag = true then error( chanused )
                             else
                                chanvect[ i ] := j;
                          end;
          filterconstp  : begin
                             show;
                             read1( 0 , 1 , 2 );
                             filterc := r;
                          end;
          parvectp      : begin
                             showvect( gradparvect );
                             read1( 1 , gradparvect , 1 );
                             setcursor( 23 , 35 );
                             write('Newvalue:=');
                             read( r );
                             with datamon do
                             begin
                                wait( mutexdata );
                                parvect[ i ] := r;
                                signal( mutexdata );
                             end;
                          end;
          scalep        : begin
                             showvect( gradparvect );
                             read1( 1 , gradparvect , 1 );
                             setcursor( 23 , 35 );
                             write('Newvalue:=');
                             read( r );
                             scalevect[ i ] := r;
                          end;
```

OPCOMREAD
par

```
        gradp       : begin
                        showvect( noofu+2 );
                        read1( 0 , noofu+2 , 1 );
                        setcursor( 23 , 35 );
                        write('Newvalue:=');
                        read( gradvect[ i ] );
                        for l := noofu+3 to gradveclength do
                        begin
                           gradvect[ l ] := 0;
                           pointvect[ l ] := 0;
                        end;
                        pointvect[ pointveclength ] := 0;
                        gradparvect := 1;
                        for  l := 1 to noofu+2 do
                        begin
                           gradparvect:=gradparvect+gradvect[l];
                           pointvect[ l+1 ] := pointvect[ l ] +
                                               gradvect[ l ];
                        end;
                        pointvect[noofu+3]:=pointvect[noofu+2] +
                                            gradvect[noofu+2];
                        with datamon do
                        begin
                           wait( mutexdata );
                           varvect[ pointvect[noofu+3] ] := 1;
                           signal( mutexdata );
                        end;
                      end;
        forgetp     : begin
                        show;
                        read1( 0 , 1 , 2 );
                        forget := r;
                      end;
        tracep      : begin
                        show;
                        read1( 0 , maxint , 2 );
                        maxtrp := r;
                      end;
        tdelvectp   : begin
                        showvect( noofu );
                        read1( 1 , noofu , 1 );
                        setcursor( 23 , 35 );
                        write('Newvalue:=');
                        read( j );
                        if ( j<0 ) or ( j>maxtimedelay ) then
                           error( tdelwrong )
                        else
                           tdelvect[ i ] := j;
                      end;
```

```
OPCOMREAD
par

          estp          : begin
                             setcursor( 19 , 1 );
                             write('1=els0    2=els2    3=rml');
                             rubout( 23 , 2 );
                             write('PAR    est=');
                             case est of
                                els0  : write('1');
                                els2  : write('2');
                                rml   : write('3');
                             end;
                             write('   newvalue:=');
                             read1( 1 , 3 , 1 );
                             case i of
                                1  : est := els0;
                                2  : est := els2;
                                3  : est := rml;
                             end;
                          end;
          predp         : begin
                             setcursor( 19 , 1 );
                             write('1=onestp  2=kstep  3=multistep');
                             rubout( 23 , 2 );
                             write('PAR   pred=');
                             case pred of
                                onestep     : write('1');
                                kstep       : write('2');
                                multistep   : write('3');
                             end;
                             write('    newvalue:=');
                             for i := 1 to maxpredhorizon do
                             begin
                                ypredvect[ i ] := 0;
                                oldypredv[ i ] := 0;
                             end;
                             read1( 1 , 3 , 1 );
                             case i of
                                1  : begin
                                        pred := onestep;
                                        predhorizon := 1;
                                     end;
                                2  : begin
                                        pred := kstep;
                                        param := predhorizonp;
                                        goto 900;
                                     end;
                                3  : begin
                                        pred := multistep;
                                        param := predhorizonp;
                                        goto 900;
                                     end;
                             end;
                          end;
```

```
OPCOMREAD
par

          predhorizonp : begin
                           show;
                           read1( 0 , maxpredhorizon , 1 );
                           predhorizon := i;
                         end;
          kstepp       : begin
                           show;
                           read1( 0 , maxpredhorizon , 1 );
                           kst := i;
                         end;
          outpp        : begin
                           show;
                           read1( 0 , 1 , 2 );
                           outp := r;
                         end;
          posp         : begin
                           rubout( 23 , 2 );
                           write('PAR   POS   lowpos=',lowpos:3,
                                 '     newvalue:=');
                           read1( 25 , 130 , 1 );
                           lowpos := i;
                           rubout( 23 , 2 );
                           write('PAR  POS   highpos=',highpos:3,
                                 '     newvalue:=');
                           read1( 25 , 130 , 1 );
                           highpos := i;
                         end;
          limitp       : begin
                           rubout( 23 , 2 );
                           write('PAR  LIM   lowlim=',lowlim:4:2,
                                 '    newvalue:=');
                           read1( -maxint , maxint , 2 );
                           lowlim := r;
                           rubout( 23 , 2 );
                           write('PAR  LIM   highlim=',highlim:4:2,
                                 '     newvalue:=');
                           read1(maxint , maxint , 2 );
                           highlim := r;
                         end;
          wronghelp    : ;
          lastparp     : error( illpar );
        end;
        for i := 17 to 21 do
          rubout( i , 1 );
end;   (*   par   *)
```

```
OPCOMREAD
on; off; filehandler


procedure on;                                   { Handles the ON-command }
begin
   comm := 'HF      ';
   par;
   comm := 'HS      ';
   par;
end;


procedure off;                                  { Handles the OFF-command }
begin
   stopflag := true;
   hslow := 0;
end;


procedure filehandler;      { Save or read a file with parameters }
type  ft    = record
                 int : integer;
                 re  : real;
              end;
      fil   = file of ft;
var   i , j , l  : integer;
      rea        : real;
      c          : command;
      ext        : identtype;
      f          : fil;

   procedure rw( var i : integer ; var r : real );
   begin
      case c of
         savex     : with f do
                     begin
                        int := i;
                        re := r;
                        put( f );
                     end;
         unsavex   : with f do
                     begin
                        get( f );
                        i := int;
                        r := re;
                     end;
      end;
   end;

begin    (*   filehandler   *)
   rubout( 23 , 2 );
   c := commandx;
   write( comname[ c ] ,'    filename:' );
   ch := ' ';
   readln;
   getcom;
   checkend;
   case c of
```

OPCOMREAD
filehandler

```
        savex      : begin
                        i := -i;
                        rewrite( f , comm , 'DAT' , i );
                     end;
        unsavex    : begin
                        reset( f , comm , 'DAT' , i );
                        if i = -1 then
                        begin
                           error( nofile );
                           goto 999;
                        end;
                     end;
    end;
    rw( hfast , rea );
    if hfast ) 0 then
    begin
       wait( mutex );
       cause( change );
       signal( mutex );
    end;
    stopflag := true;
    rw( hslow , rea );
    if hslow ) 0 then stopflag := false;
    rw( j , filterc );
    rw( noofu , rea );
    for i := 1 to chanveclength do
       rw( chanvect[ i ] , rea );
    for i := 1 to pointveclength do
       rw( pointvect[ i ] , rea );
    for i := 1 to gradveclength do
       rw( gradvect[ i ] , rea );
    rw( gradparvect , rea );
    for i := 1 to maxnoofu do
       rw( tdelvect[ i ] , rea );
    with datamon do
    begin
       wait( mutexdata );
       for i := 1 to veclength do
       begin
          rw( j , parvect[ i ] );
          rw( j , scalevect[ i ] );
          rw( j , k[ i ] );
       end;
       for i := 1 to veclength do
          for l := 1 to veclength do
             rw( j , p[ i , l ] );
       signal( mutexdata );
    end;
    rw( j , forget );
    rw( j , maxtrp );
    rw( j , trp );
    rw( j , lowlim );
    rw( j , highlim );
    close( f );
end;
```

OPCOMREAD
disp

```
procedure disp;
     { handles the display command :  DISP parameter }

var    disppar : display;
       mess    : messageref;

begin   (*   disp   *)
   ch := ' ';
   getcom;
   checkend;
   disppar := alld;
   while ( dispname[disppar]()comm ) and ( disppar()lastpard ) do
      disppar := succ( disppar );
   if disppar () lastpard then
      dispflag := false;
   receivemessage( messagepool , mess );
   with mess do
   begin
      messtype := opcom;
      case disppar of
         alld       : messdisp := alld;
         stopd      : messdisp := stopd;
         curved     : messdisp := curved;
         lastpard   : error( illpar );
      end;
   end;
   if disppar = lastpard then
      sendmessage( messagepool , mess )
   else
   begin
      sendmessage( opcombox , mess );
      readln;
      rubout( 23 , 2 );
      read( ch );
      dispflag := false;
      receivemessage( messagepool , mess );
      mess.messtype := opcom;
      mess.messdisp := stopd;
      sendmessage( opcombox , mess );
   end;
end;   (*   disp   *)
```

```
OPCOMREAD
main opcomread


(*****      main OPCOMREAD      *****)


begin
   setpriority( 12 );
   for i := 1 to 23 do
      rubout( i , 1 );
   setcursor( 22 , 1 );
   write('OFF');
   while true do
   begin
999:  setcursor( 22 , 1 );
      if ( hfast = 0 ) or ( hslow = 0 ) then
         write('OFF')
      else
         write(' ON');
      rubout( 23 , 1 );
      write(')');
      readln;
      setcursor( 23 , 2 );
      ch := ' ';
      getcom;
      commandx := onx;
      while ( comname[ commandx ] <> comm ) and
            ( commandx <> lastcomx ) do
         commandx := succ( commandx );
      case commandx of
         onx         : on;
         offx        : off;
         parx        : par;
         dispx       : begin
                          if ( hfast=0 ) or ( hslow=0 ) then
                             error( nosamp )
                          else
                             disp;
                       end;
         savex ,
         unsavex     : filehandler;
         lastcomx    : error( illcom );
      end;
   end;    (*  while true do  *)
end;    (******  main opcomread  ******)
```

```
OPCOMWRITE
eraseterm, all


   (**************************
   *   PROCESS  OPCOMWRITE   *
   **************************)


(* process *) procedure opcomwrite;

var mess    : messageref;
    row ,
    col ,
    c ,
    i       : integer;
    r       : real;


procedure eraseterm;
begin
   getcursor( row , col );
   for i := 1 to 21 do
      rubout( i , 1 );
   rubout( 24 , 1 );
   setcursor( row , col );
end;   (*   eraseterm   *)


procedure all; { Displays gradvect,parvect,varvect,k and traceP.
                 Uppdating is done every sampleperiod }
begin
   dispflag := true;
   eraseterm;
   getcursor( row , col );
   setcursor( 1 , 1 );
   write('VECTPOS');
   for i := 1 to 10 do
      write( i:7 );
   setcursor( 2 , 1 );
   write('GRADVEC');
   for i := 1 to gradveclength do
      write( ' ', gradvect[ i ]:6 );
   setcursor( 3 , 1 );
   write('POINTV ');
   for i:= 1 to pointveclength do
      write(' ', pointvect[ i ]:6 );
   setcursor( 5 , 1 );
   write('SCALEV');
   setcursor( 8 , 1 );
   write('PARVECT');
   setcursor( 11 , 1 );
   write('VARVECT');
   setcursor( 14 , 1 );
   write('K       ');
   setcursor( 17 , 1 );
   write('traceP ');
   setcursor( row , col );
   while dispflag do
```

OPCOMWRITE
all

```
      with datamon do
      begin
         wait( mutexdata );
         await( dispchange );
         getcursor( row , col );
         for i := 1 to gradparvect do
            if i <= 10 then
            begin
               c := 8 + (i-1)*7;
               setcursor( 5 , c );
               write(' ', scalevect[ i ]:5:2 );
               setcursor( 8 , c );
               write(' ', parvect[ i ]:6:3 );
               setcursor( 11 , c );
               write(' ', varvect[ i ]:6:3 );
               setcursor( 14 , c );
               write(' ', k[ i ]:6:3 );
            end
            else
            begin
               c := 8 + (i-11)*7;
               setcursor( 6 , c );
               write(' ', scalevect[ i ]:5:2 );
               setcursor( 9 , c );
               write(' ', parvect[ i ]:6:3 );
               setcursor( 12 , c );
               write(' ', varvect[ i ]:6:3 );
               setcursor( 15 , c );
               write(' ', k[ i ]:6:3 );
            end;
         setcursor( 17 , 8 );
         write( trp:8:3 );
         setcursor( row , col );
         signal( mutexdata );
      end;   (* with datamon do *)
end;
```

OPCOMWRITE
curve

```
procedure curve;                    { Displays a curve of predictions }

var   j ,
      poswidth      : integer;
      y , hf ,
      step ,
      limwidth      : real;

begin
   dispflag := true;
   poswidth := highpos - lowpos;
   limwidth := highlim - lowlim;
   step := limwidth / poswidth;
   eraseterm;
   getcursor( row , col );
   for i := 0 to 10 do
   begin
      j := i * poswidth div 10;
      setcursor( 1 , ( lowpos + j )-2 );
      r := lowlim + step * j;
      write( r:3:2 );
      setcursor( 2 , lowpos + j );
      write('!');
   end;
   setcursor( 2 , 1 );
   writeln('*=predicted output');
   write('+=real output');
   setcursor( 10 , 1 );
   hf := hfast * 0.02;
   writeln('hfast =', hf:5:3 ,'sec');
   writeln('hslow =', hslow:3 ,' * hfast');
   writeln('filterconst=', filterc:4:3 );
   case pred of
      onestep     : writeln('ONESTEPPRED');
      kstep       : writeln('KSTEPPRED');
      multistep   : writeln('MULTISTEPPRED');
   end;
   writeln('predhorizon=', predhorizon:2 );
   writeln('kst=', kst:2 );
   setcursor( row , col );
   while dispflag do
   begin
      with datamon do
      begin
         wait( mutexdata );
         await( dispchange );
         signal( mutexdata );
      end;
      getcursor( row , col );
      j := maxpredhorizon - predhorizon;
      for i := 1 to j do
      begin
         rubout( i+2 , lowpos );
         r := oldypredv[ j+1+kst-i ] - lowlim;
         setcursor( i+2 , lowpos + round( r/step ) );
```

```
OPCOMWRITE
curve; main opcomwrite


            write('*');
            r := oldyv[ j+1-i ] - lowlim;
            setcursor( i+2 , lowpos + round( r/step ) );
            write('+');
         end;
         if ( pred=kstep ) or ( pred=onestep ) then
         begin
            rubout( j+4 , lowpos );
            r := ypredvect[ 1 ] - lowlim;
            setcursor( j+4 , lowpos + round( r/step ) );
            write('*');
         end
         else
            for i := j+1 to maxpredhorizon do
            begin
               rubout( i+3 , lowpos );
               r := ypredvect[ i-j ] - lowlim;
               setcursor( i+3 , lowpos + round( r/step ) );
               write('*');
            end;
         setcursor( row , col );
      end;
   end;



(****     main opcomwrite     ****)

   begin
      setpriority( 14 );
      while true do
      begin
         receivemessage( opcombox , mess );
         with mess do
         case messdisp of
            alld          : all;
            stopd         : eraseterm;
            curved        : curve;
         end;
         sendmessage( messagepool , mess );
      end;
   end;    (***    main opcomwrite    ***)
```

```
MAIN STP
MAIN


procedure init; external;
procedure dummy; external;
procedure makemessage( box : mailbox ; no : integer ); external;
procedure prediction; external;
procedure parameterest; external;
procedure sampler; external;
procedure clock; external;
procedure opcomread; external;
procedure opcomwrite; external;


   (************
   *   MAIN   *
   ************)


begin     (*****    main STP    *****)
   init;
   dummy;
   initkernel( 2250 );
   initio;
   initmailbox( messagepool );
   initmailbox( parestbox );
   initmailbox( predbox );
   initmailbox( regulbox );
   initmailbox( opcombox );
   makemessage( messagepool , 10 );
   createprocess( prediction , 920 );
   createprocess( parameterest , 2650 );
   createprocess( sampler , 110 );
   createprocess( clock ,  50 );
   createprocess( opcomread , 725 );
   createprocess( opcomwrite , 200 );
   setpriority( maxpriority );
end.
```