CODEN: LUTFD2/(TFRT-5439)/1–127/(1991)

Rekursiv identifiering av dynamiska egenskaper hos kokarvattenreaktor för stabilitetsmonitering

Per Abrahamsson
Peter Hallgren

Institutionen för Reglerteknik
Lunds Tekniska Högskola
Juni 1991

Department of Automatic Control
Lund Institute of Technology
P.O. Box 118
S-221 00 Lund Sweden

Document name
Master Thesis

Date of issue
June 1991

Document Number
CODEN:LUTFD2/(TFRT-5439)/1-127/(1991)

Author(s)
Per Abrahamsson, Peter Hallgren

Supervisor
Rolf Johansson, Stig Andersson

Sponsoring organisation
ABB Atom AB

Title and subtitle
Rekursiv identifiering av dynamiska egenskaper hos kokarvattenreaktor för stabilitetsmonitering

Abstract

Identification of process dynamics is used for stability monitoring in nuclear reactors (Boiling Water Reactor). This report treats the problem of estimating a damping factor and a resonance frequency from the neutron flux as measured in the reactor.

A new parametric online method for identification is derived and presented, and is shown to meet the requirements of stability monitoring. The technique for estimating the process parameters is based on a recursive lattice filter algorithm.

The problem of time varying parameters and offset, as well as offline experiments and signal processing are treated. All parts are implemented in a real-time program, using the language C.

In comparison with earlier identifications, the new way of estimating the damping factor is shown to work well. Estimates of both the damping factor and the resonance frequency show a stable and realible behavior.

Future developments and improvements are also indicated.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
Swedish

Number of pages
127

Recipient's notes

Security classification

SAMMANFATTNING

I denna rapport undersöks en delvis ny metod att identifiera dämpkvot och resonansfrekvens ur neutronflödesbrus i en kärnreaktor. Sådan identifiering av dynamiska systemegenskaper tillämpas vid sk. stabilitetsmonitering i kokvattenreaktorer.

En parametrisk realtidsmetod för identifiering i PC-miljö har utvecklats och presenteras. Den bygger på en rekursiv latticefilterbaserad algoritm för skattning av systemets parametrar. Problem med tidsvarierande parametrar och offsetnivå i mätsignalen behandlas, liksom förberedande signalbehandling och off-line identifiering.

Samtliga moment är implementerade i realtid och ett komplett C-program för detta presenteras. Funktionerna finns även utvecklade i MATLAB.

Jämförelse mellan befintliga reaktordata och tidigare utförda identifieringar tyder på ett gott resultat. Den skattade dämpkvoten och resonansfrekvensen visar sig stabil, och pålitlig vad gäller noggrannhet.

Framtida utvecklingsmöjligheter och begränsningar med den nya metoden presenteras.

ABSTRACT

Identification of process dynamics is used for stability monitoring in nuclear reactors (Boiling Water Reactor). This report treats the problem of estimating a damping factor and a resonance frequency from the neutron flux as measured in the reactor.

A new parametric online method for identification is derived and presented, and is shown to meet the requirements of stability monitoring. The technique for estimating the process parameters is based on a recursive lattice filter algorithm.

The problem of time varying parameters and offset, as well as offline experiments and signal processing are treated. All parts are implemented in a realtime program, using the language C.

In comparison with earlier identifications, the new way of estimating the damping factor is shown to work well. Estimates of both the damping factor and the resonance frequency show a stable and reliable behavior.

Future developments and improvements are also indicated.

FÖRORD

Denna rapport är ett examensarbete på institutionen för reglerteknik vid Lunds Tekniska Högskola. Handledare på institutionen har varit Rolf Johansson.

Arbetet som presenteras är utfört hos ABB Atom i Västerås under perioden november till april (90 -91). Rapporten är färdigställd i Lund under våren -91. Handledare i Västerås har varit Stig Andersson och Camilla Larsson i metodfrågor samt Caj Svensson i Datorfrågor.

Vi vill härmed rikta ett stort tack till samtliga handledare.

Författarna

Lund 910630

INNEHÅLL

1. INLEDNING

Identifiering av dynamiska systemegenskaper online tillämpas vid sk. stabilitetsmonitering i kokvattenreaktorer, en teknik som vuxit fram först de senaste åren. Kraven på snabb respons vid ändring av systemegenskaperna samt presentation on-line ställer hårda krav på algoritmerna.

ABB Atom har utvecklat ett sådant system, kallat COSMOS, som bygger på en "icke-parametrisk" metod.

Målet för vårt examensarbete har varit att utveckla och implementera en parametrisk metod för on-line identifiering i PC-miljö, bestämma prestanda samt verifiera metoden mot registrerade, verkliga reaktordata. I arbetet har ingått metodarbete med att ta fram en algoritm som passar de krav som ställs, implementering i programspråket C, uttestning i realtidsmiljö samt jämförelse av resultat med reaktordata mot andra tillgängliga utvärderingar.

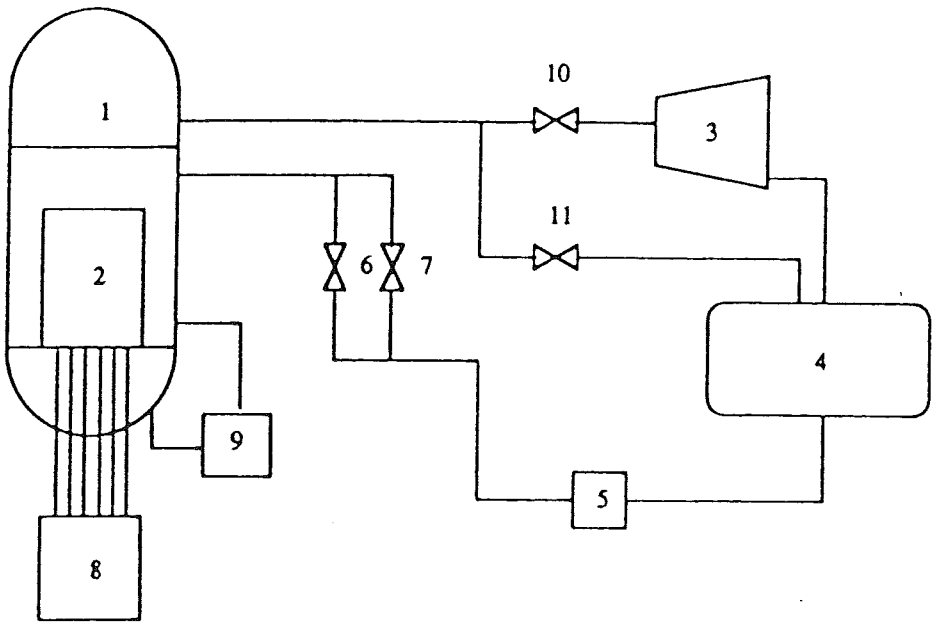
Identifieringen kan delas upp i ett antal delproblem. Dessa moment behandlas i kapitlen enligt följande. En bakgrund till problemet ges i kapitel 2 och 3, där även inledande försök behandlas. I kapitel 4 behandlas signalkonditionering och i kapitel 5 själva modell-skattningen med tyngdpunkt på latticefilter. Den kompletteras i appendix 5 och 6. I kapitel 6 redovisar vi genomförda försök med figurer och kommenterar resultatet. I kapitel 7 behandlas implementeringen. Resultat och framtida utvecklingsmöjligheter diskuteras i kapitel 8.

2 PROCESSEN

För att kunna förstå uppgiften och problematiken krävs en viss bakgrunds-kunskap om själva processen, dvs en kokvattenreaktor. Detta innefattar ren "reaktorkunskap" (2.1), lite allmänt om stabilitet (2.2) samt en problemformulering (2.3).

2.1 Bakgrund

För att belysa problematiken med stabilitet i en kärnreaktor, gör vi här en mycket kortfattad genomgång av en kokarreaktors uppbyggnad och dess funktion med avseende på stabilitets-egenskaper[16]. Se figur 2.1



Figur 2.1 Principschema för kokvattenreaktor (BWR).

I reaktortanken (1) finns reaktorns härd i form av bränsleelement av uran (2). Värmeutvecklingen i bränslet kan regleras med styrstavar (8) och huvudcirkulations-pumpar(9). Bränslet kyls med vatten som strömmar förbi bränsleelementen. På grund av den stora värmeutvecklingen förångas kylmediet (vattnet) och ångan som bildas leds ut genom ledningar i reaktortankens övre del .

Flödet av den heta ångan (ca 280 grader) når turbinanläggningen (3) och avger sin energi till turbinerna. Dessa driver elgeneratorer som avger elenergi med en spänning på ca 2000 Volt. När ångan har passerat turbinen, strömmar den in i kondensorn (4), där den kyls och övergår i vatten.

Vattnet kallas i detta skede matarvatten och en pump (5) pumpar in det i reaktorn igen. Innan vattnet når reaktorn passerar det två ventiler, avstängningsventilen (6) och reglerventilen (7). För att bibehålla en konstant nivå vatten/ånga, är det viktigt att den mängd ånga som lämnar reaktorn ersätts med lika mycket vatten.

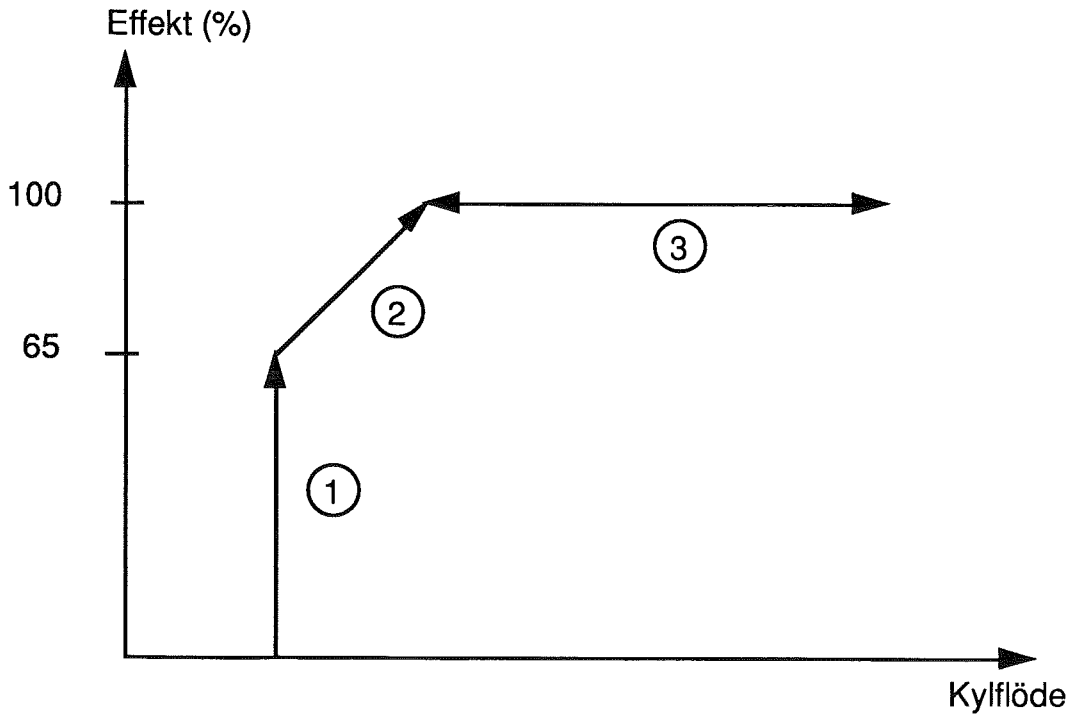
Trycket i reaktorn hålls konstant genom att ångflödet ändras med pådragsventiler (10) och dumpventiler (11). Pådragsventilerna styr flödet till turbinerna och dumpventilerna gör det möjligt att dumpa ånga till kondensorn, utan att ångan passerar turbinen.

Förändring av effektutvecklingen sker genom reglering av neutron-flödet i härden. Det kan regleras dels med hjälp av styrstavar och dels med hjälp av vattenflödet genom reaktorn. När styrstavarna (8) förs in i härden ökar absorberingen av neutroner och effekten minskar. När reaktorn laddas med nytt bränsle, skjuts styrstavarna in ca 60% före uppstarten. De dras sedan kontinuerligt ut i takt med att bränslet förbränns, för att hålla mängden aktivt bränsle konstant. Detta, tillsammans med nödstopp, då stavarna "skjuts" hydrauliskt in i härden, är styrstavarnas huvudsakliga funktion.

Vid normal drift regleras neutronflödet främst genom ändring av vattenflödet i reaktorn i samband med effektuttagsförändring. Med huvud-cirkulationspumparna (9) cirkuleras en blandning av vattnet i reaktorn och inkommande matarvatten. På så sätt ändras ånghalten i härden. När effekten minskar, minskas cirkulationen, vilket medför att ånghalten ökar och neutronernas hastighet ökar. Det är främst termiska neutroner som ger fission i denna typ av reaktor, dvs ökat kylflöde ger ökad effekt. Genom ändring av pumparnas varvtal regleras ca 40% av effekten, medan resterande

60% regleras med styrstavarna. Mätning av neutronflödet görs med ett antal givare, utplacerade på olika ställen i härden.

Med denna kunskap kan det vara intressant att titta på ett diagram över effekt som funktion av kylvattenflödet. Ett sådant är figur 2.2. Linje1 är förloppet vid uppstart av reaktorn, där man alltså successivt höjer effekten upp till ca 65%. Därefter höjs effekten genom kylflödes-ökning till full effekt (linje 2). Området 3 är där man rör sig under normal drift. Erfarenheter visar att under en effektuppgång vid lågt kylflöde, är stabilitetsmarginalerna snävast när man för första gången nått 65% effekt. Därefter tenderar stabiliteten i den uppnådda driftpunkten att förbättras successivt, allteftersom Xenon-koncentrationen i härden närmar sig jämvikt. Vid detta tillfälle är det önskvärt med en noggrann stabilitets-monitering.

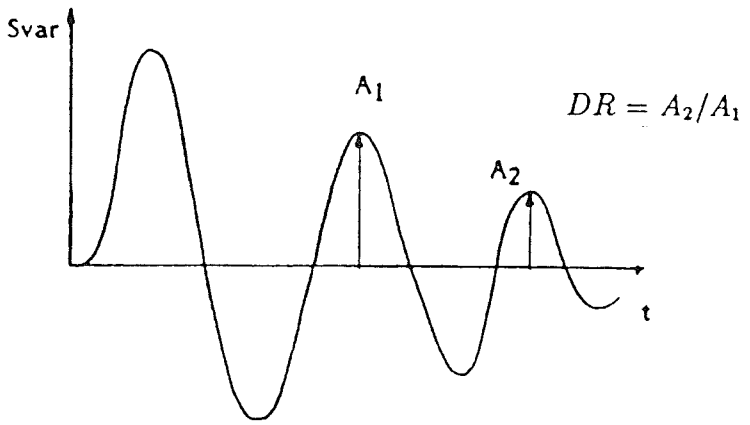


Figur 2.2 Effekt/kylvattenflödes-diagram

2.2 Stabilitet

Med stabilitet för ett dynamiskt system menas att, utgående från ett stationärt driftläge, störningar skall dämpas ut och processen ska efter en viss tid återgå till samma driftläge. En kärnreaktor blir instabil då en reaktivitetsökning från ett stationärt tillstånd medför en ökning av neutronflödet (och effekten) utan att reaktorn når ett nytt stationärt tillstånd. Vid dynamisk instabilitet ligger orsaken i dynamiken (tidsfördröjningar etc.) hos systemet och kan inte förklaras med rent statiska samband.

Praktiska stabilitetskriterier, dvs mått på hur långt ett system har till instabilitet är tex. amplitudmarginal och fasmarginal. Ytterligare ett sätt är dämpkvot (Decay ratio) eller dämpfaktor. Den definieras som amplitud-förhållandet mellan konsekutiva amplitudmaxima i ett oscillerande svar på en kortvarig störning.

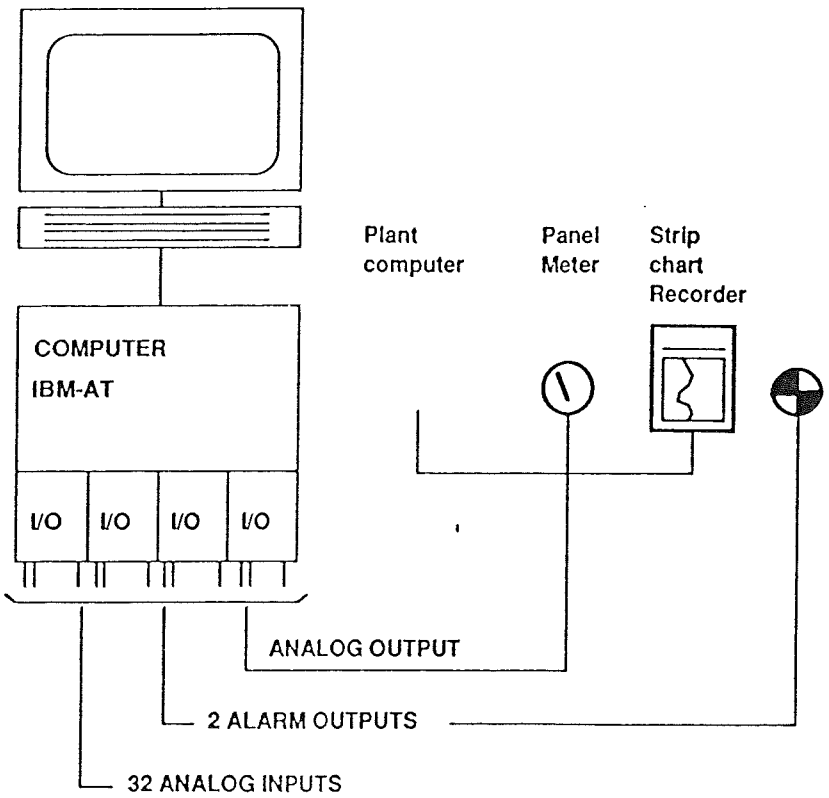


Figur 2.3 Definition av dämpkvot ur impulssvar

Inom kärnkraftsindustrin är man av tradition intresserad av dämpkvoten som stabilitetsmått. Efter en viss tid efter en pålagd störning dominerar systemets minst dämpade egenfrekvens, så att svaret blir nära exponentiellt avklingande. Dämpkvoten är lätt att

utvärdera ur tex. en impulsstörning. DR = 1 svarar mot ett odämpat system och DR < 1 mot ett dämpat system. DR < 0.25 ("quarter damping") används ofta som mått på ett "väl" dämpat system [2]. Det är emellertid omöjligt att lägga en impulsstörning på reaktorn och titta på dess impulssvar (av uppenbar anledning) utan man måste tillgripa andra metoder.

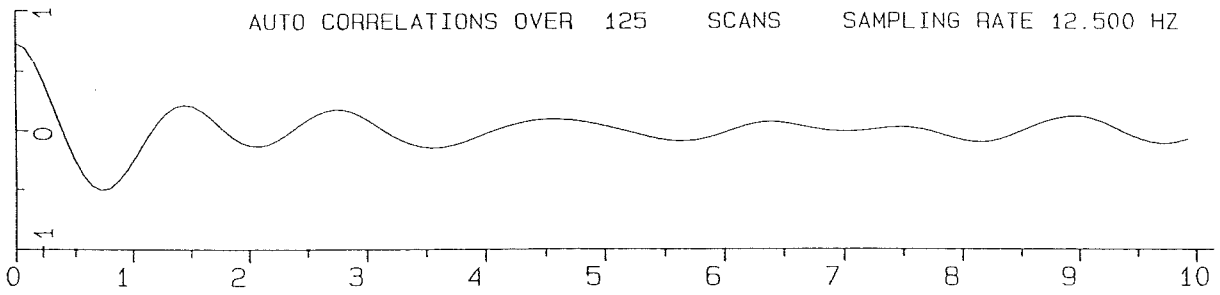
On-line identifiering av dynamiska systemegenskaper tillämpas vid s.k stabilitetsmonitering i kokarreaktorer. ABB Atom har utvecklat ett sådant system, COSMOS (Core Stability MOnitoring System) [17]. Det är en utrustning med snabb respons, som kontinuerligt övervakar härdstabiliteten (se fig 2.4). Systemet samlar in mätdata från upp till 32 kanaler, analyserar dessa och bestämmer dämpfaktor, brusnivå och resonansfrekvenser.



Figur 2.4 Principskiss över moniteringssystemet COSMOS

2.3 Problemformulering

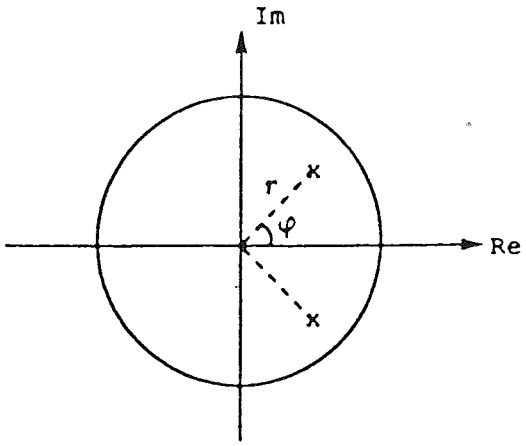
Hur mäter man då stabiliteten och dämpkvoten utan att lägga på en störning? Neutronflödet beskriver aktiviteten i reaktorn och ur denna signal kan mått på stabiliteten härledas (se kapitel 2.2). Mätsignalen är en "brussignal" med en resonanstopp för reaktor-dynamiken (se kapitel 2.3). Ett sätt att mäta stabiliteten är att ur reaktorbrusets autokorrelation, bestämma dämpkvot och resonans-frekvens. Autokorrelationsfunktionen ger visserligen samma information som impulssvaret, men är behäftad med statistiska egenskaper som gör att noggrannheten snabbt blir dålig. Transienter och överlagrade (stör-) frekvenser försvårar även identifieringen. Figur 2.5 visar exempel på en autokorrelations-funktion där närvaron av (högfrekventa) störningar är tydlig.



Figur 2.5 Exempel på autokorrelationsfunktion för uppmätt neutronflöde

En annan metod för att bestämma dämpkvoten, är att ur systemets poler beräkna dämpkvoten analytiskt. Detta sätt kräver en parametrisk modell. Uppgiften i detta examensarbete är att bestämma dämpkvot och resonansfrekvens på detta sätt, dvs att identifiera dynamiken bakom neutronflödet i reaktorn och ur modellen, analytiskt beräkna dämpkvot och resonansfrekvens. Man kan, för ett andra ordningens system, visa att dämpkvoten beräknas som:

$$DR = e^{2\pi \ln(r/\varphi)} \qquad (2.1)$$



Figur 2.6 Tidsdiskreta poler för andra ordningens system

I en parametrisk modell definierar man dämpkvoten ur det dominerande polparet (enligt formel 2.1), eftersom systemets minst dämpade egenfrekvens dominerar ett tag efter pålagd störning. En mer fullständig härledning av uttrycket för dämpkvoten finns i appendix A2.

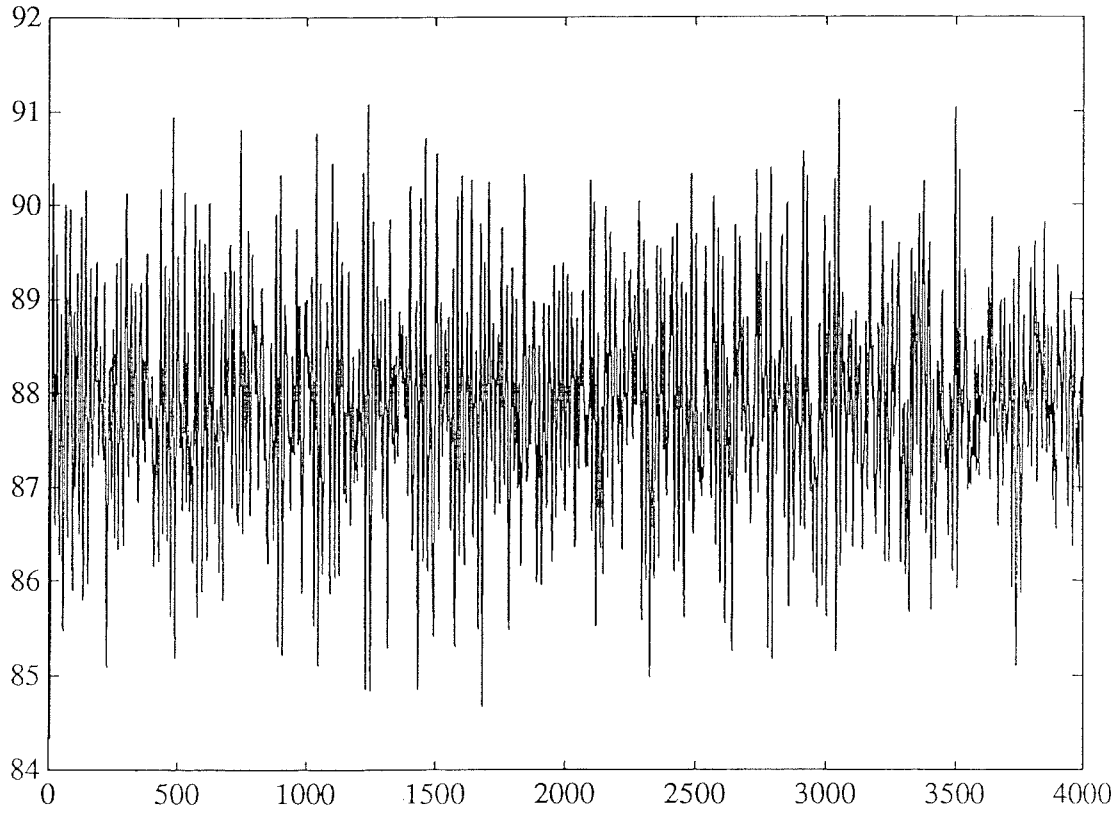
Sammantaget innebär detta att dämpkvoterna skiljer sig åt rent definitionsmässigt. Den icke-parametriskt framtagna via auto-korrelationen, blir en "momentan" som endast tar hänsyn till tidiga toppar i impulssvaret. Den analytisk beräknade dämpkvoten ur en parametrisk modell, blir "asymptotisk" då vi tar hänsyn till "hela" impulssvarets information (med hjälp av dominerande polparet) om hur snabbt en pålagd störning dämpas ut (se även definition av dämpning i kap 2.1).

3. INLEDANDE FÖRSÖK

I en identifieringsprocedur bör man börja med en serie inledande försök för att skaffa sig så mycket förkunskap om processens egenskaper som möjligt. Detta görs för att snabbare "komma rätt" och underlätta den egentliga identifieringen. Detta moment innefattar analys av mätsignalen och dess uppkomst (3.1) samt de off-line identifieringar som har genomförts (3.2). Off-line försök får i detta fall betraktas som inledande försök, i syfte att lära känna processen inför realtidsimplementeringen.

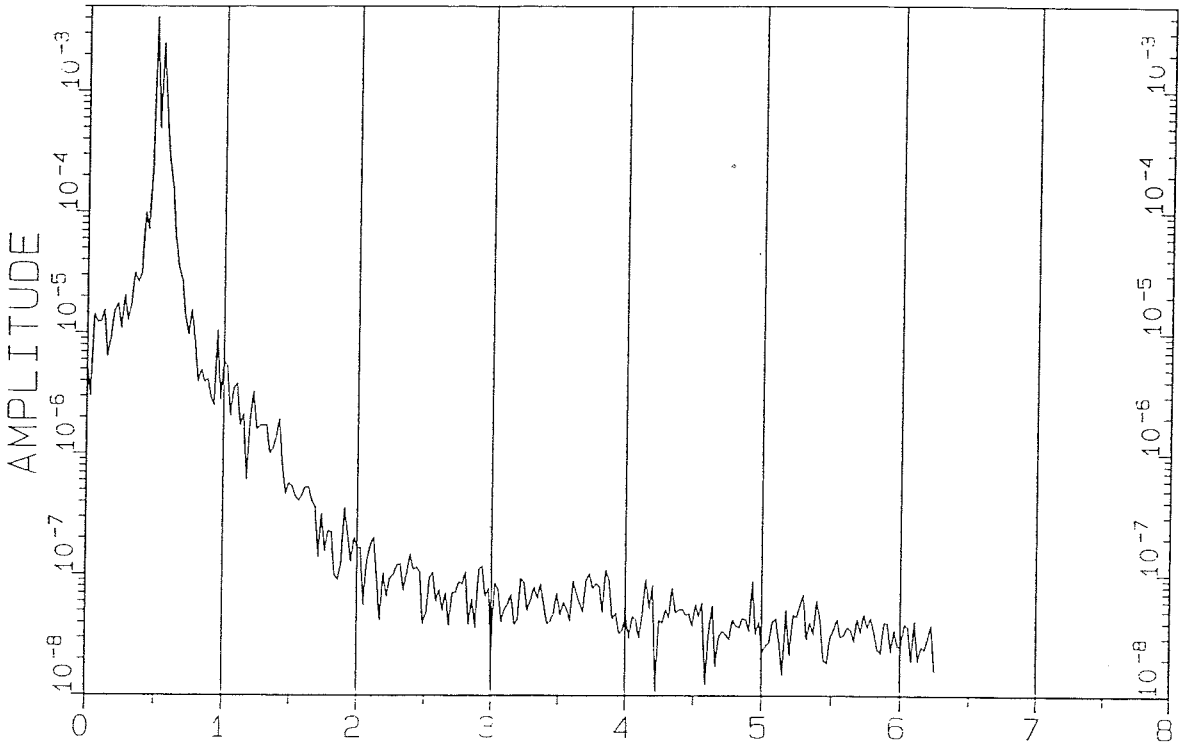
3.1 Analys av Mätsignalen

Mätsignalen är en brussignal och ser typiskt ut enligt figur nedan.



Figur 3.1 Neutronflödet i reaktorhärden

Ett av de första momenten är att analysera mätsignalen. Dess ursprung är förklarat i kapitel 2.1 och vi var nu intresserade av signalens egenskaper. När man undersöker spektrum för signalen uppträder en signifikant resonanstopp vid ca 0.5 Hz. Det är denna resonanstopp som beskriver dynamiken för neutronflödet.

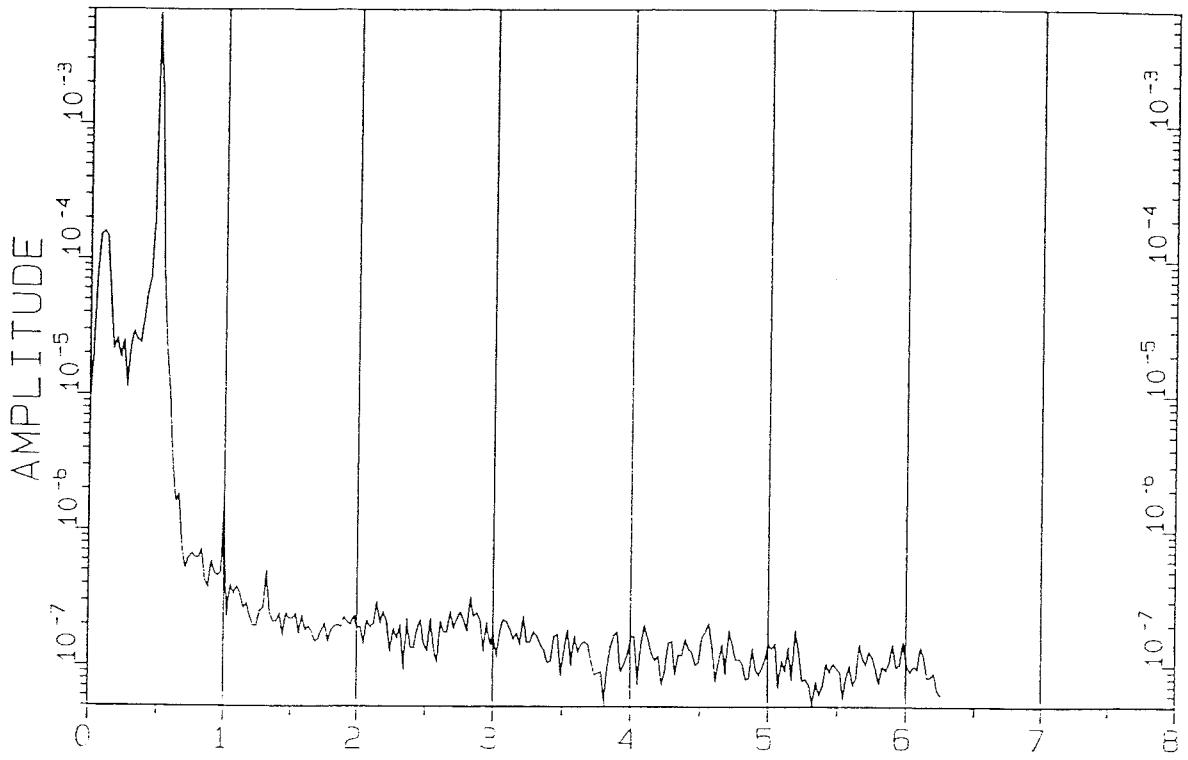


Figur 3.2 Exempel på effektspektrum med en tydlig resonanstopp. (Frekvens i x-led).

Frekvenser över ca 1 Hz har mycket litet informationsinnehåll och det är naturligt att sampla ned signalen. Det görs för att identifieringen ska bli lättare. Det är dessutom onödigt att försöka modellera dessa onödiga frekvenser, med högre ordningar som följd. Mer om nedsamplingen framgår i kapitel 4.1. De mätsignaler som vi haft tillgång till har varit samplade med 12.5 Hz och dessa har vi samplat ned tre gånger för att förenkla identifieringen. Det innebär en bandbredd på ca 2.1 Hz

Resonansen i neutronflödessignalen har sitt ursprung i dynamiska kopplingar mellan bla kylvattenflödet och reaktorhärden. Störningar i kylvattenflödet (som ständigt finns) ger upphov till täthetsvågor i tvåfassystemet (vatten och ånga). Neutronflödet svarar på dessa störningar via återkopplingar, på ett sätt som påminner om hur ett dynamiskt system av andra ordningen (massa, fjäder, dämpverkan) reagerar på utifrån kommande störningar. Detta är intressant med hänsyn till möjligheterna att beskriva en reaktorhärds stabilitets-egenskaper via en karakteristisk frekvens och en dämpfaktor [2]

Det finns emellertid fler resonanser i systemet. Dessa är både högre och lägre i frekvens. De lågfrekventa resonanserna (se figur 3.3) härrör sig bla. från det effektreglersystem till pumpar och liknande, som reglerar kylvattenflödet genom härden.



Figur 3.3 Effektspektrum med två resonanstoppar

Naturligtvis inverkar dessa resonanser på skattningen av dämpningen och speciellt när resonanstopparna har ungefär lika hög

amplitud och ligger nära varandra i effektspektra. Det är en av anledningarna till att modellordningen varierar vid olika drift-situationer (dämpningar).

3.2 Off-line skattningar

För att skaffa oss så mycket kunskap om processen som möjligt inledde vi med en serie off-line skattningar med hjälp av MATLAB:s funktioner AR och ARMAX. Vad vi ville veta om processens uppförande var främst förväntad modellordning, eventuella resonanser och dess frekvenser och framförallt dämpkvotens beteende för olika modellstrukturer och ordningar. Hur känslig dämpkvoten var för variationer i parametrar var också en fråga vi ställde oss. I appendix A2 finns en härledning av dämpkvot. Av sambandet ser man ett exponentiellt förhållande mellan dämpkvot och parametrar, vilket antyder att känsligheten kan vara stor.

Eftersom vi har ett system utan insignal var de modellstrukturer vi försökte anpassa AR och ARMA. Ett antagande var att systemet drevs av vitt brus, dvs en AR-modell skulle vara tillräcklig. Så är dock knappast fallet i verkligheten och frågan var hur giltig denna approximation var.

Teoretiskt kan man visa att man alltid inför ett fel när man anpassar en AR-modell till ett brusstört system, då bruset inte är vitt:

$$y_n = x_n + w_n$$

$$P_y(z) = \frac{s^2 \Delta t}{A(z)\, A^*(1/z^*)} + s_w^2 \Delta t$$

$$= \frac{[s^2 + s_w^2\, A(z)\, A^*(1/z^*)]\, \Delta t}{A(z)\, A^*(1/z^*)}$$

där

$$X_n = \frac{1}{A(z^{-1})} v_n$$

$$E\left\{v_n^2\right\} = s_v^2$$

$$E\left\{v_n\right\} = 0$$

$A^*(1/z^*)$ är reciproka A-polynomet och $z = e^{i\omega h}$ är operatorn.

P består av både poler och nollställen dvs y är en ARMA(p,p) process. Inkonsistensen för en AR-modell för brusstörd AR-process kan leda till felskattningar. Ett sätt att komma runt detta är att använda AR-modell av högre ordning än den "sanna" AR-modellen. Mer om detta står att läsa om i [8]. Frågan är hur mycket bruset avviker från vitt och hur mycket "fel" man gör.

I vårt fall har det visat sig att det inte är en dålig approximation att anta en AR-modell. Modellordningen kan förväntas bli något högre för en AR-modell än motsvarande ARMA-modell, vilket framgår ur tabell 3.1 och 3.2.

I spektrum för signalerna ser man ibland fler resonanstoppar (se kap 3.1) och dessa gav mer eller mindre besvär. För att modellera ett system med flera dominerande resonansfrekvenser, krävs i allmänhet en modell av högre ordning. I vårt fall ligger dessa frekvenser mycket nära varandra. Problem uppstod när skillnaden i amplitud mellan resonanstopparna i effektspektrum minskade. Oftast finns dock en dominerande frekvens med mycket kraftigare amplitud, och då uppstår inga problem i identifieringen

Nedan presenteras ett exempel på offline-identifiering. Mätsignal är det neutronflöde som visas i figur 3.1 dvs vid normal, stationär drift.

En AR-modell beskriver processen som:

$$A(q)\ y(t) = e(t)$$

där e(t) är vitt brus. Resultatet av ett antal identifieringar med olika modellordningar visar att det går att anpassa en AR-modell offline till processen, men att den är beroende av modellordning. Tabell 3.1 visar exempel på AR-modeller av olika ordningar. Genomgående för alla de mätdata vi tittat på, är att dessa modeller är mer varierande i kvalitet, vad gäller dämpkvot och frekvenser. Vi noterar också den minimala information som Akaike's FPE gav i dessa försök.

Kommandot AR i MATLAB skattar modellen med en minsta-kvadrat metod. En sådan är känslig för "outliers" eller abnorma mätvärden. Detta faktum tillsammans med antagandet om vitt brus ger en känslig skattning.

MODELLORDNING	AKAIKES FPE	DOMINERANDE FREKVENS	DÄMPKVOT
1	0,9597	-	-
2	0,3218	0,67	0,53
3	0,2951	0,72	0,54
4	0,2957	0,72	0,45
5	0,2960	0,71	0,59
6	0,2948	0,73	0,54
7	0,2903	0,77	0,32
8	0,2907	0,69	0,41
9	0,2924	0,69	0,43

Tabell 3.1 AR-identifiering

Nästa steg var att undersöka om en ARMA-modell gav ett bättre resultat än ovanstående AR-skattning. En ARMA modellerar bruset med ett C-polynom:

$$A(q)\ y(t) = C(q)\ e(t)$$

där e(t) är vitt brus. På grund av det färgade bruset kan man inte använda sig av MK-metoder utan att få systematiska fel i parametrarna (se [5], kap4.2). I MATLAB används en robustifierad PEM (Prediction Error Method), en iterativ skattningsmetod med möjlighet att påverka robustifiering. Vi identifierade samma data som för AR och upptäckte en del skillnader.

MODELLORDNING nA	nC	AKAIKES FPE	DOMINERANDE FREKVENS	DÄMPKVOT
2	2	0,2958	0,70	0,44
3	2	0,2946	0,72	0,45
4	2	0,2911	0,71	0,50
5	2	0,2902	0,72	0,51
6	2	0,2915	0,73	0,47
7	2	0,2918	0,71	0,41
8	2	0,2895	0,70	0,38
2	3	0,5070	0,72	0,51
3	3	0,4863	0,72	0,49
4	3	0,4902	0,73	0,49
5	3	0,4713	0,73	0,47
6	3	0,5201	0,72	0,52

Tabell 3.2 ARMA-identifiering

3.3 Slutsatser av inledande försök

Man kan ur tabellerna 3.1 och 3.2 utläsa att modellordningarna blir ungefär desamma för AR och ARMA. Tabell 2 visar att ARMA-modellerna blir något bättre både vad gäller parametrarnas

varianser och dämpkvotens beroende av modellordning. Skillnaderna var emellertid inte så stora som väntat och man bör komma ihåg att förutsättningarna inte var de samma. Dessutom minskade skillnaderna med ökad mätlängd. Detta pekar åt att en AR-modell kan vara en fullgod modell. Skillnaden ligger i den högre bruskänsligheten hos AR-strukturen vilket medför de ibland alltför varierande dämpkvoterna.

Detta sammantaget tyder på att en ARMA-modell antagligen hade givit något bättre resultat i realtidsskattningen, men inte så mycket bättre att det överväger de andra fördelarna med den AR-algoritm vi valt att använda, Se kap 5.1. Dessutom existerar ingen sådan "lattice"-algoritm för ARMA-modeller och det var avgörande för vårt beslut. När det gäller bestämning av modellordning gav Akaikes´ AIC och FPE mycket lite information i de ordinära skattningara. Även sådana tester som residualtest gav inga entydiga resultat angående rätt modellordning.

Det bör påpekas, att dessa försök gjordes på ett tidigt stadium och att de valideringsmetoder som vi använde då, inte är desamma som använts för slutalgoritmen. Dessutom hade vi ingen kunskap om uppförandet av systemet i detta tidiga skede och därför visade sig slutsatserna om modellordningar något förhastade. Vidare ska man inte glömma bort att det handlar om off-line dvs ett medelvärde över hela mätdataserier. Vi vet nu att dämpkvoten ingalunda är konstant under några längre intervall. AR-resultaten blev dessutom betydligt stabilare med ökad mätlängd. Vi hade alltför svag datorkraft i detta skede, vilket var orsaken till lite förvillande besked då vi inte kunde identifiera över tillräckligt långa mätserier.

Erfarenheterna från off-line testerna visar att känsligheten var ganska stor för val av ordningstal men att den minskade med ökad mätlängd. Ordningstalen höll sig dock inom ett ganska begränsad område, runt 5-7, beroende på modell-struktur. ARMA-modellerna något lägre, medan en AR-modell krävde något högre ordningstal.

4. SIGNALKONDITIONERING

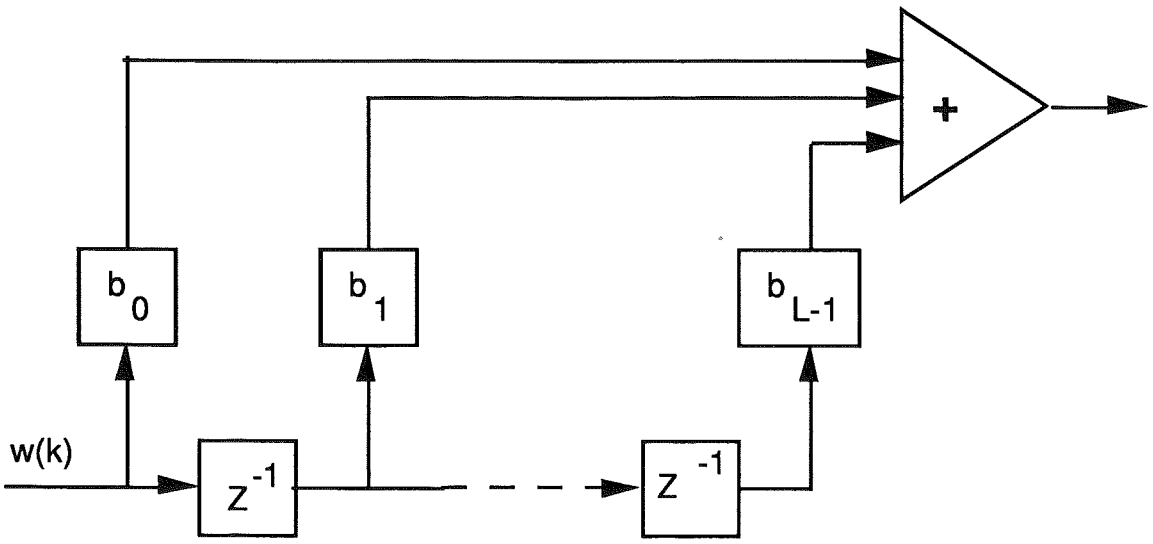
Den samplade råsignalen besitter egenskaper, som måste åtgärdas innan en identifiering kan anses ge fullgott resultat. En av dessa egenskaper är frekvensinnehållet och samplingsfrekvensens förhållande, vilka leder till att signalen bör samplas ned (4.1). En annan egenskap är förekomsten av offset i signalen (4.2) och de praktiska lösningarna på detta behandlas i kapitel 4.3.

4.1 Nedsampling och filtrering

Ur spektrum av signalen ser man att de intressanta frekvenserna (resonanstopp) ligger runt 0.5 Hz (se kap 3.1). Det nuvarande systemet samplar med 8 Hz, vilket ger en nyquistfrekvens på 4 Hz. Det innebär, att man vid en parametrisk identifiering, försöker modellera ett onödigt stort frekvensband om signalen identifieras direkt. Av den anledningen samplar vi ned signalen 2 ggr. Den nya Nyquistfrekvensen blir då 2 Hz. För att detta ska kunna göras utan att införa vikningsfenomen (aliasing), måste signalen lågpassfiltreras före nedsamplingen. Vi har gjort detta med ett 30:e ordningens FIR-filter (Finite Impulse Response). Ett FIR-filter innehåller enbart nollställen, vilket är en attraktiv egenskap att utnyttja då vi inte vill föra in ny dynamik i systemet. Det kan naturligtvis inte undvikas men det minimeras. Vidare är FIR-filter lätta att implementera och principen framgår av figur 4.1.

Förutom ovan nämnda fördelar är FIR-filter alltid stabila, på grund av det ändliga pulssvaret och det förhållande att de kan fås faslinjära [10]. Det finns naturligtvis även nackdelar. Den höga ordningen behövs för att få en någorlunda smal övergångs-zon mellan pass- och spärrband, vilket medför lång fördröjning för selektiva filter. Dessutom saknas enkla syntesmetoder för koefficienterna, men flera approximativa finns. Se [10]. I MATLAB finns en sådan syntes implementerad under kommandot FIR1, och från den hämtas färdiga filterkoefficienter till C-programmet (se kap 7.1). Frågan är hur mycket mätsignalen påverkas av filtreringen. Vi har jämfört ett antal kurvor före och efter nedsampling/filtrering samt tittat på frekvensinnehåll. Resultaten tyder på att

informationen är intakt efter filtret och framförallt inte påverkar den skattade dämpkvotens beteende.



Figur 4.1 Stukturen hos FIR-filter. b_o - b_{L-1} är filterkoefficienter

4.2 Offset på mätsignal

Ett vanligt problem i identifiering är förekomsten av offset i mätsignalen. Detta orsakar ett fel i skattningarna om det ignoreras och måste därför tas om hand. Det finns i huvudsak två olika sätt att lösa problemet

- Skatta medelvärdet explicit
- Använda modeller med differentierade data

Användningen av differentierade data innebär i princip en högpass-filtrering, och är enkelt att använda, men resultatet är inte så bra som man kan önska. Det dåliga resultatet beror på att nya brus-korrelationer införs, vilka kan störa en MK-skattning mer än den

ursprungliga offseten. Vi har därför valt att skatta medelvärdet explicit och koncentrerar oss därför på denna metod. Mer om andra metoder finns att läsa om i tex [9].

Skattning av medelvärdet explicit

Vid skattning av medelvärde kan man välja mellan att skatta det separat eller tillsammans med parametrarna, dvs utöka sin modell med en offset :

$$\varphi(t) = (\; y(t-1) \; y(t-2) \; ..... \; y(t-n) \; \; 1 \;)$$

$$\theta(t) = (\; a_1 \; a_2 \; ..... \; a_n \; \; m \;)$$

Då vi använder oss av en sk. latticealgoritm (Se kap 5), är det svårt att utöka modellen direkt. Vi har därför valt att skatta medelvärdet separat och sedan subtrahera det från signalen.

Vid skattning av ett medelvärde separat ställs man inför problemet att medelvärdet kan variera. Då modellen av ett medelvärde är en första ordningens modell, är detta analogt med att man måste detektera förändringar i parametrar, och sådana förändringar kan vara både snabba och långsamma. Detta är ett känt problem [4],[5] och man kan dela upp det i olika delar

- skatta ett "bra" medelvärde (dvs modell av ordning1)
- detektera snabba hopp
- detektera långsamma förändringar
- modifiera algoritmen för dessa fall

För att skatta ett tidsvarierande system, kan man välja mellan RLS (Recursive Least Square) med glömskefaktor eller en Kalmanansats. Vi har valt att använda RLS eftersom det är en enkel metod att handskas med. Minsta-kvadratmetoden minimerar kriteriet (förlust-funktionen):

$$V_N(\hat{\theta}) = \sum_{i=1}^{t} [\; y(t) - \hat{\theta}^T(t)\varphi(t) \;]^2$$

Fördelen med minsta-kvadratmetoden är att den är enkel och snabb att använda. Nackdelarna är att den kräver vitt brus för att ge konsistenta skattningar samt att känsligheten för störningar är stor. För att skatta medelvärde duger RLS-metoden dock utmärkt. Med glömskefaktor blir det möjligt att skatta tidsvarierande system [6], se figur 4.2. Förlustfunktion med glömskefaktor blir:

$$V_N(\hat{\theta}) = \sum_{i=1}^{t} \lambda^{t-i} [\ y(t) - \hat{\theta}^T(t)\varphi(t)\ ]^2 \qquad 0 < \lambda \leq 1$$

och algoritmen blir :

$$\hat{\theta}_k = \hat{\theta}_{k-1} + P_k \phi_k \varepsilon_k$$

$$\varepsilon_k = y_k - \phi_k^T \hat{\theta}_{k-1}$$

$$P_k = \frac{1}{\lambda} \left(P_{k-1} - \frac{P_{k-1}\phi_k \phi_k^T P_{k-1}}{\lambda + \phi_k^T P_{k-1} \phi_k} \right)$$

Då glömskefaktorn speglar hur många värden bak i tiden man tar hänsyn till vid uppdateringen, blir den avgörande för hur snabba förändringar man kan detektera. Antal sampel som en RLS-algoritm "kommer ihåg" är ungefär proportionellt mot:

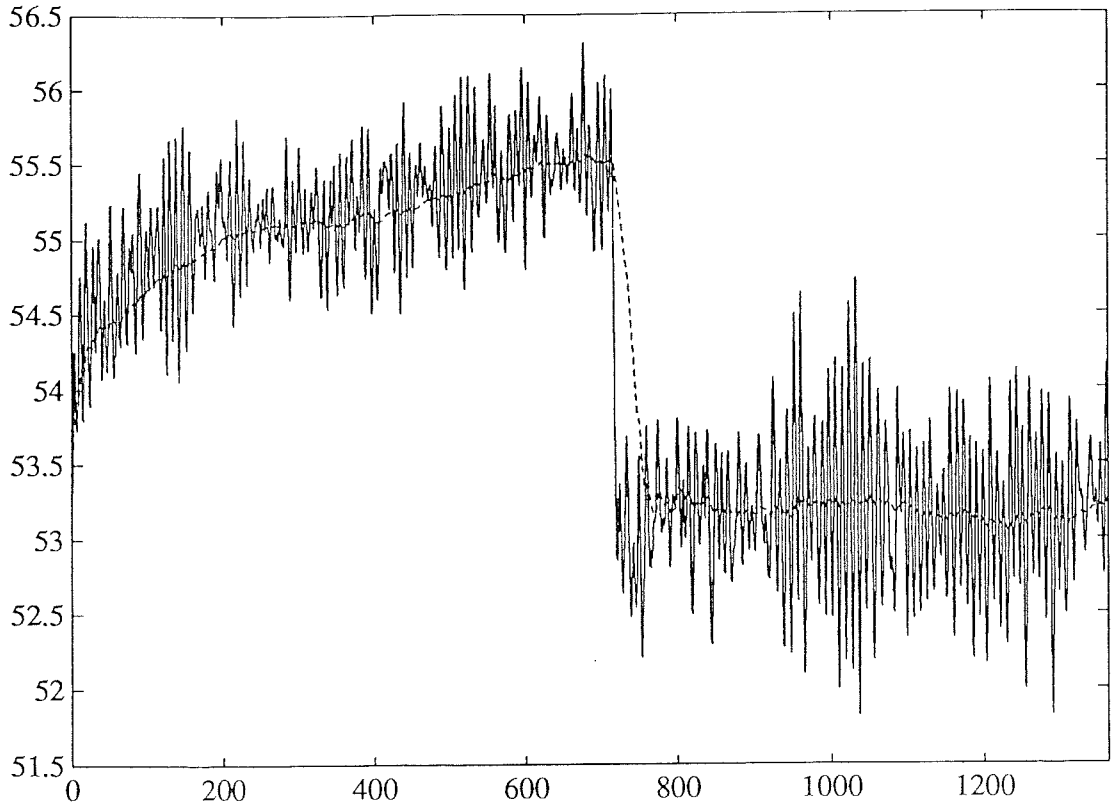
$$\frac{1}{1 - \lambda} \tag{4.1}$$

där λ är glömskefaktorn. Användningen av en glömskefaktor motsvarar exponentiell viktning av gamla data, då en mätning n sampels tillbaks har en vikt proportionell mot

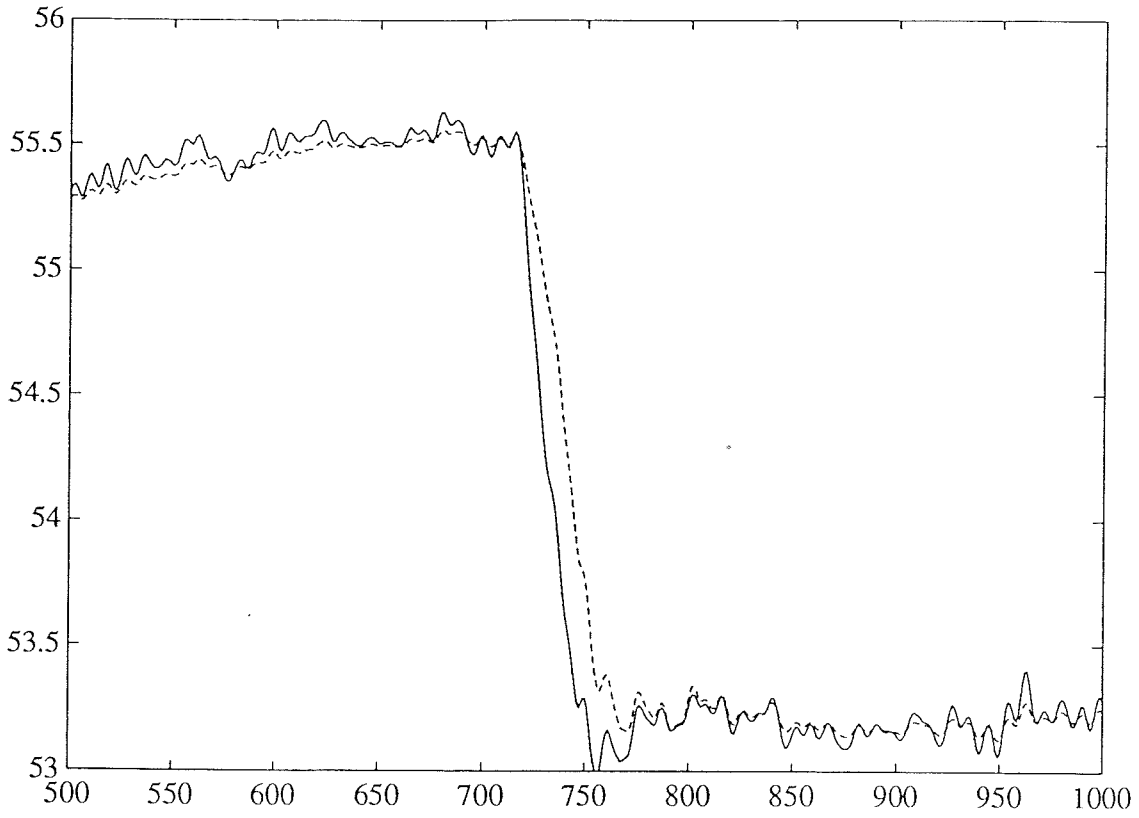
$$\lambda^n = e^{n \cdot \ln(\lambda)} \qquad \lambda < 1 \tag{4.2}$$

där äldre data inverkar mindre för $0 < \lambda < 1$. Valet av lambda blir en kompromiss mellan snabbhet och ett bra medelvärde. För litet λ ger mer en skattning av signalen än av dess medelvärde och ett för stort hindrar oss från att följa snabba förändringar i medelvärdet. I figur 4.3 visas skattningar av medelvärdet med olika λ. Ett kompromissvärde på glömskefaktorn som visat sig ge bra skattningar är λ= 0.96-0.98, men det är inte något kritiskt val inom detta intervall (se figur 4.3).

Problemet med långsamma medelvärdesförändringarna löses automatiskt på grund av det låga λ i medelvärdesskattningen. Med λ=0.96 viktas endast ca 25 gamla värden till skattningen och långsammare trender följs utan problem.



Figur 4.2 Offsetskattning (streckad) med RLS-algoritm, λ = 0.98



Figur 4.3 Offsetskattning med λ = 0.95 och 0.98 (streckad)

Det återstår nu att modifiera algoritmen, när hopp i medelvärdet uppstår. Eftersom medelvärdesskattningen ändå är "fel", kan man minska λ för att snabbare komma rätt igen. Samtidigt måste parameterskattningen modifieras, då signalen nu har en offset. I RLS-fallet kan man då tex. uppdatera linjärt istället för kvadratiskt , dvs man tecknar förlust-funktionen utan kvadrat för att ta mindre hänsyn till de felaktiga värdena.

$$J(\hat{\theta}) = \sum_{i=1}^{t} [\ y(t) - \hat{\theta}^T(t)\varphi(t)\ ] \qquad (4.3)$$

Det innebär, att man tar för lite hänsyn till gamla värden, om man detekterar ett falskt hopp. Detta problem har mer eller mindre sofistikerade lösningar (se [1],[4],) som alla bygger på iden att skatta

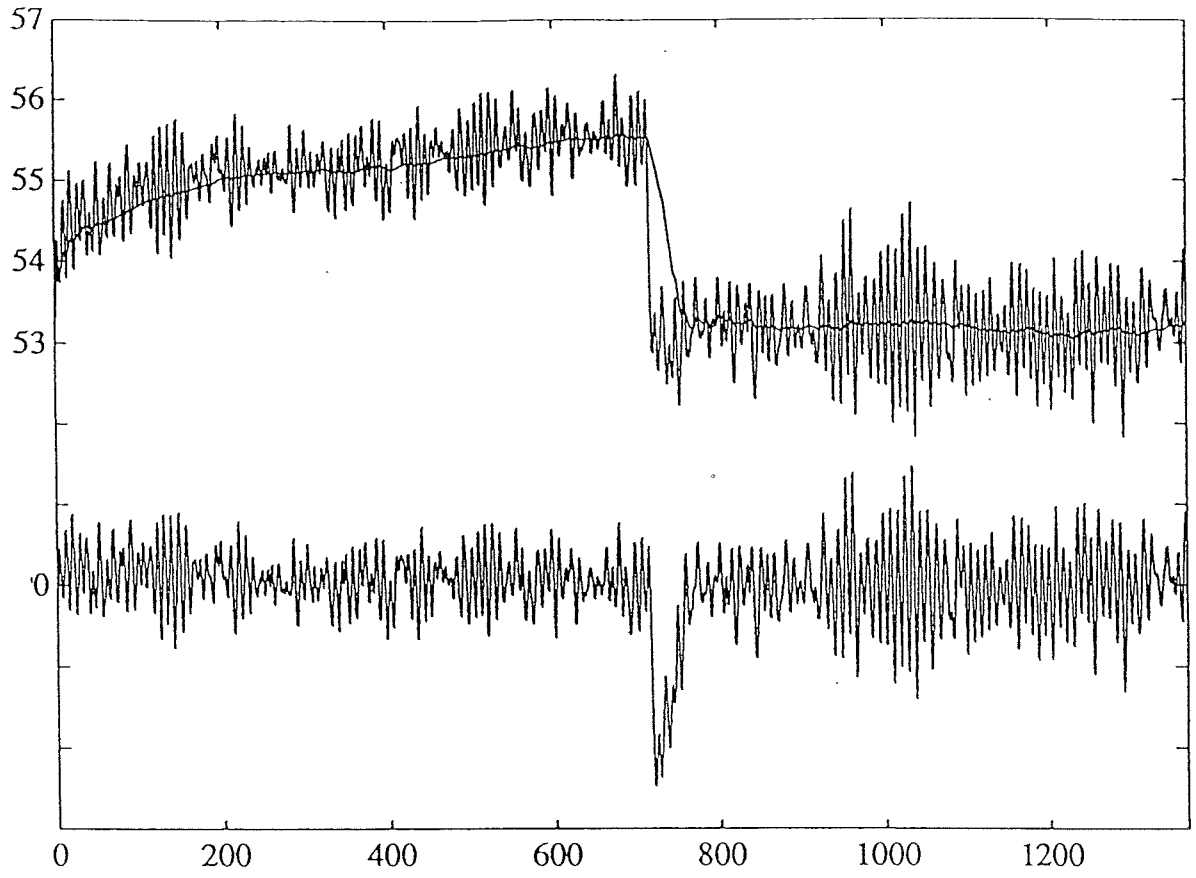
flera modeller parallellt. Om en modell detekterar ett falskt hopp och förstör sina gamla mätvärden, finns det andra som är intakta. Genomgående är dessa metoder komplexa att programmera, tar mer datorkraft och kräver mycket förkunskap om signalens uppförande. Bland annat ska man i de flesta fall ange sannolikheter för hopp och liknande parametrar, vilket är omöjligt i vårt fall. Metoderna är dessutom mycket känsliga för val av dessa parametrar. På grund av dessa svårigheter har vi valt att använda en enklare och "robustare" metod.

4.3 Feldetektering

När ett hopp i medelvärdet kommer, uppstår i signalen motsvarande avvikelse från noll-nivån i form av en mer eller mindre bred "spik" (se fig 4.4). Problemet att detektera hoppet (skattade medelvärdets avvikelse från det sanna) kvarstår fortfarande och ett sätt är att försöka detektera denna "spik".

Det vanligaste sättet är att använda sig av residualen ε, dvs skillnaden mellan sanna utsignalen och den skattade utsignalen från RLS-algoritmen [5]. Man kan då tänka sig att ange en nivå, som den signalen ej får överskrida.

Denna nivå måste naturligtvis variera med signalamplituden och en sådan nivå kan vara brusvariansen för prediktionsfelet i medelvärdesskattningen. Denna brusvarians skattar vi fram med en MK-metod, se nedan. En nackdel är att endast fel som har tillräckligt stor amplitud kan detekteras, dvs små hopp i medelvärdet kan ge avvikelser från nollnivån som är tillräckligt stora för att störa skattningen, men för små för att detekteras. Den skattade brusnivån multipliceras därför med en faktor γ och i exemplet i figur 4.5 är γ = 3.3. Valet av γ blir en kompromiss mellan att kunna detektera små medelvärdeshopp utan att detektera amplitudvariationer som fel.



figur 4.4 Skattad offset (övre) och nettosignal efter bortdragen offset, $\lambda = 0.98$

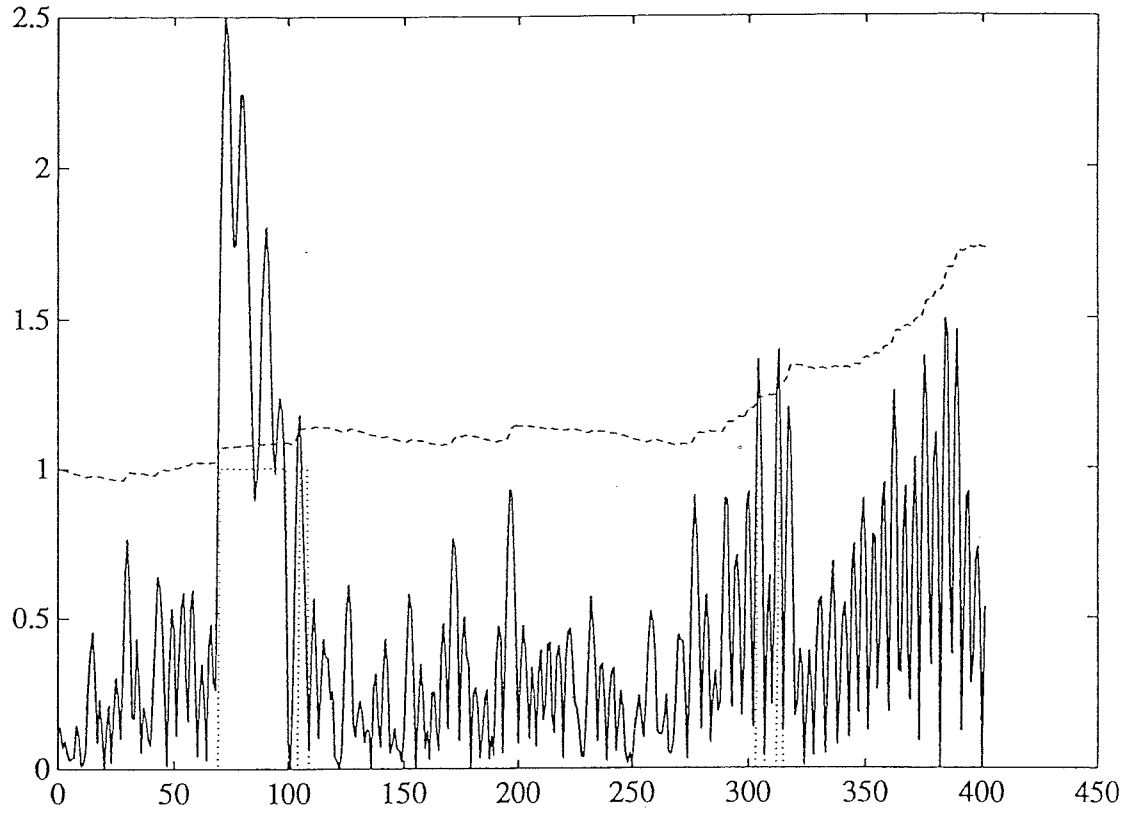
Skattning av brusvariansen

När skattningen av $\hat{\theta}$ är nära det sanna värdet θ (t), är prediktionsfelet

$$\varepsilon(t) = y(t) - \varphi^T(t)\,\hat{\theta}$$

nästan vitt brus och

$$\frac{1}{N}\sum_{k=1}^{t}\varepsilon^2(k) \approx E\,[e^2(t)]$$



figur 4.5 Absolutvärdet av signalen i figur 4.4 (heldragen) med skattad brusvarians multiplicerad med γ (streckad). Prickad signal är flagga till parameterskattningen, se mer i kapitel 4.3

En bra skattning i detta fallet borde vara, se [1]:

$$\hat{R}_2 = \sum_{k=1}^{t} (y(k) - \hat{\theta}(t)\varphi(k))^2 \qquad (4.4)$$

där $\hat{R}_2(t)$ är skattat R_2 vid tiden t. För att glömma bort gamla och möjligen felaktiga värden av $\varepsilon(t)$, dvs dåliga approximationer av e(t), men framförallt för att kunna följa förändringar i R_2, kan även här en glömskefaktor användas (μ) [1]:

$$\hat{R}_2 = (1 - \mu) \sum_{k=1}^{t} (\mu^{t-k})(y(k) - \hat{\theta}\varphi(k)))^2 \qquad (4.5)$$

eller rekursivt [1]

$$\hat{R}_2(t) = (1- \mu) (1- \varphi^T(t) P_{i,max}(t) \varphi(t)) \ \varepsilon^2(t) + \mu \hat{R}_2(t-1) \qquad (4.6)$$

där

$$i_{max} = \arg \max \alpha_i(t)$$

Här bör nämnas att ekvation 4.6 är en exakt rekursiv version av 4.5 endast om glömskefaktorn μ överensstämmer med den som används vid skattning av θ. I praktiken är dock 4.6 en fullgod approximation [1]. Vad man ska komma ihåg är att när skattningen av θ är långt från det sanna värdet, kan detta vara en dålig skattning av brusvariansen. Nu är inte exaktheten i skattningen av R2 så kritisk eftersom vi eftersträvar endast ett mått, som någorlunda följer amplitudnivån. För detta ändamål är skattningen fullt tillräcklig.

4.4 Praktiska lösningar

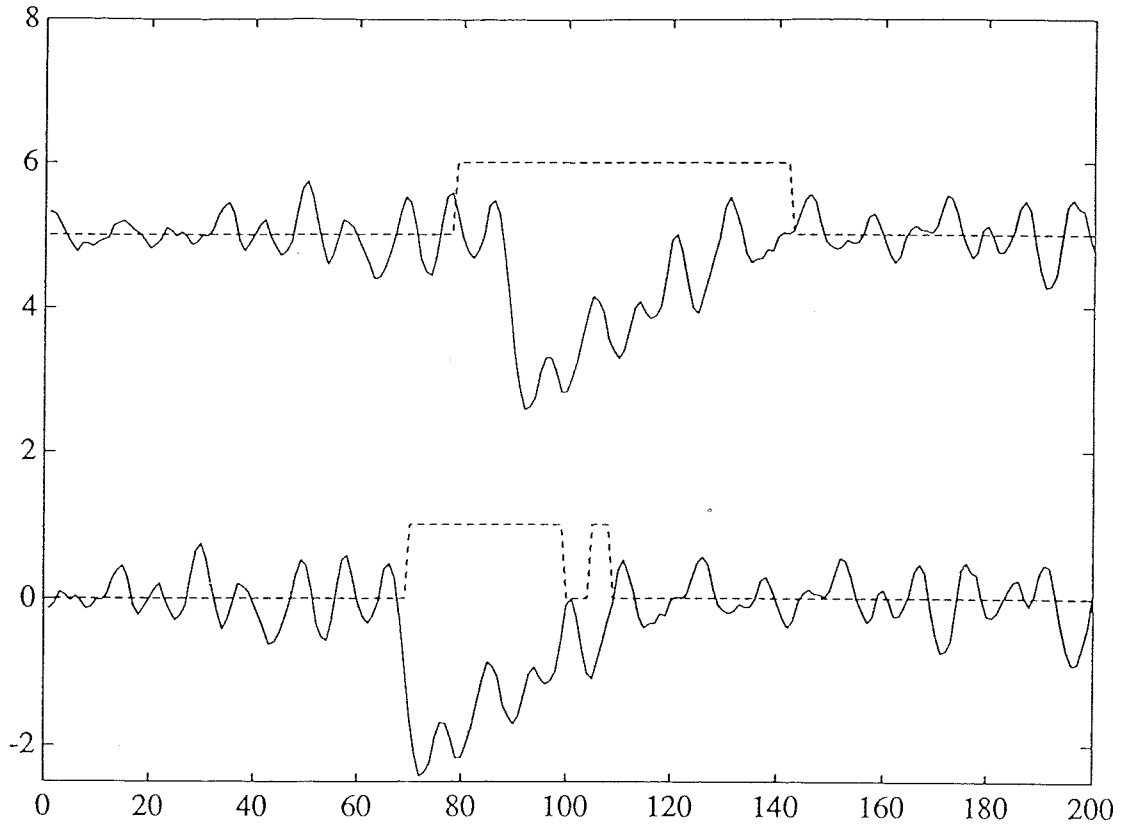
Kvar att behandla är hur man anpassar "lattice-algoritmen" när ett fel är detekterat. Då vi har en lattice-algoritm utan direkt förlustfunktion (se kapitel 5) har vi valt att helt stänga av skattningen av de sk. reflektionskoefficienterna (som motsvara modellens parametrar) då ett hopp detekteras. Detta är en ganska "brutal" metod, men har visat sig förhållandevis robust. Problem kan uppstå, när algoritmen detekterar ett falskt hopp, dvs stänger av skattningen trots att inget hopp i medel-värde har skett. Det visade sig att dessa feldetekteringar inte är så allvarliga. Man får lite fasvridningsfenomen i och med den diskontinuerliga skarvningen vid avstängningar, men det stör inte skattningen av dämpkvoten.

Vi erinrar oss om, att ett för litet λ ger mer en skattning av signalen än av dess medelvärde och ett för stort λ hindrar oss från att följa snabba förändringar i medelvärdet (se figur 4.3). Ett sätt att

förbättra detekteringen av små hopp kan vara att öka λ . Då får man visserligen ett långsammare beteende vid hopp, men med det resultatet att avvikelsen från nollnivå i signalen blir större för små hopp, vilket ökar känsligheten. Nackdelen är att avvikelsen (felet) blir bredare, vilket gör att "lattice"-rutinen är avstängd under längre tidsintervall med förlorad information som följd. Även här blir alltså valet av λ en kompromiss, nämligen mellan hur lång avstängning man kan acceptera och hur små hopp man måste kunna detektera. λ = 0.98 är en kompromiss vi har gjort. Valet är emellertid inte speciellt kritiskt i intervallet 0.96-0.98. Eventuellt kan detta behöva ökas men med de data vi hade tillgång till fungerade det tillfredsställande

Avstängning av parameterskattning sker när $\varepsilon > 3.3\sqrt{R_2}$. Att gränsen är just denna, är empiriskt utprovat på de data vi har haft tillgång till. I figur 4.5 är den prickade signalen indikation på att medelvärdesskattningen är fel. Den uppträder vid sampel 75 och vid ca 300. Under de sampel när parameterskattningen är avstängd, ökas glömskefaktorn μ för att det skattade R_2 ska få samma värde före som efter felet. Även detta illustreras av figur 4.5.

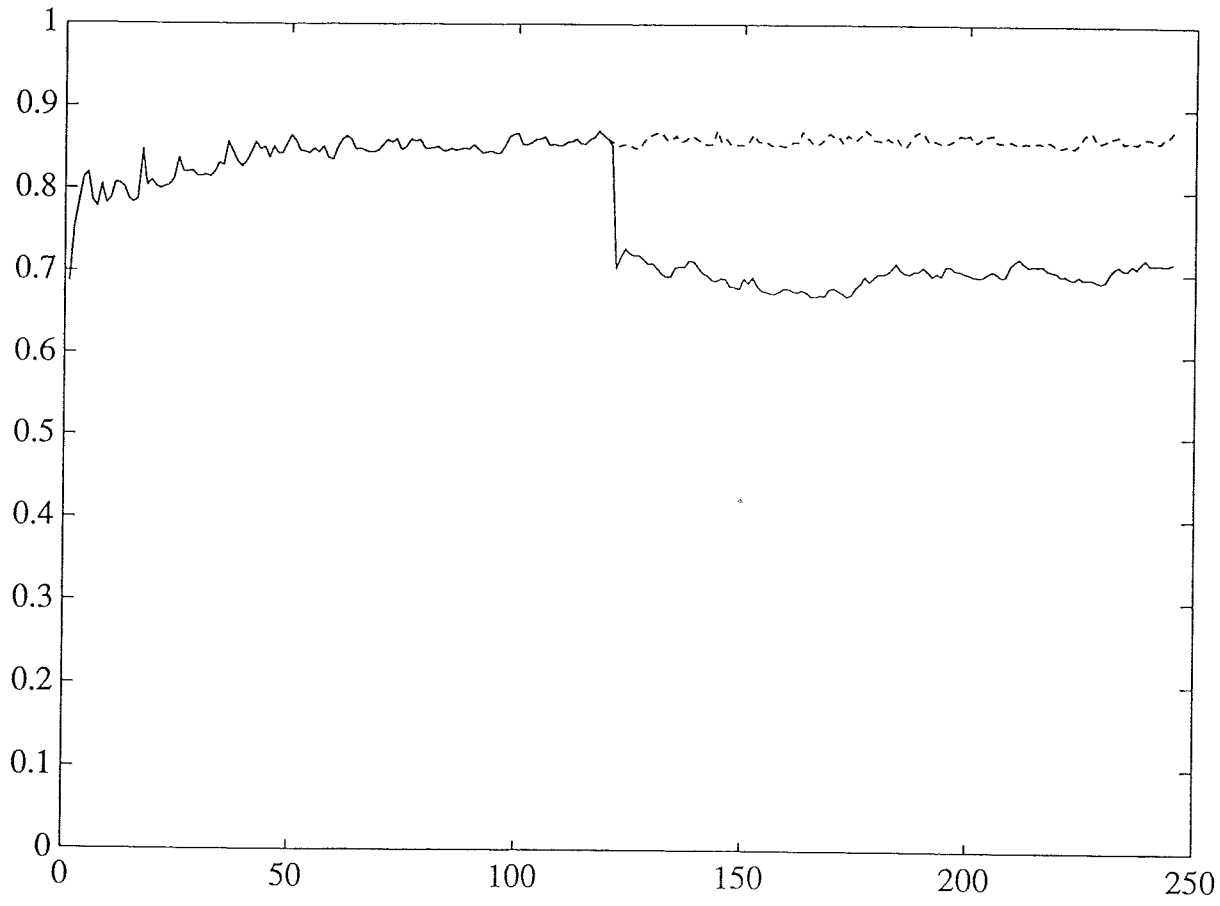
För att förhindra att enstaka sampel i början och slutet av en feldetektering används, stängs parameterskattningen av några sampel före felet uppstår samt fördröjs ett antal sampel efter. Detta sker genom en fördröjning av mätsignalen. I figur 4.6 illustreras detta där undre figuren är "verkliga" felindikationen och den övre är mätsignal och "styrsignal", såsom den används av parameter-skattningen, dvs efter fördröjningar. Med "styrsignal" menas i detta sammanhang en flagga som indikerar feldetektering och som läses av parameterskattningsalgoritmen. En bild säger mer än tusen ord:



Figur 4.6 Förstoring av samma signal som i figur 4.5 och 4.6, men med "flaggans" (streckad) egenskaper förtydligade.

Denna metod har visat vara ett utmärkt sätt att ta hand om förändringar i medelvärdet, men är känslig för inställningen av vissa parametrar. Talet $\gamma = 3.3$ som används för att definiera nivån på R_2 är en sådan och även glömskefaktorn i skattningen av R_2 påverkar detekteringen ganska kraftigt.

Med de mätdata vi har haft tillgång till, fungerar metoden tillfredsställande, men det kan naturligtvis uppstå situationer när den ger feldetekteringar. En feldetektering stör skattningen lite men dämpkvoten svänger snabbt in sig till "rätt" värde. Ignoreras hoppet störs skattningen kraftigt och det tar mycket lång tid för dämpkvoten att svänga in sig mot rätt värde. I figur 4.7 jämförs skattad dämpkvot med och utan hänsyn till hopp i medelvärdet. Inverkan av hoppet är tydlig när vi inte tar hänsyn till det. Hoppet i medelvärde uppträder ungefär vid sampel 120.



Figur 4.7 Skattad dämpkvot på mätdata med hopp i medelvärdet. (λ = 0.98). Streckad kurva är skattning med hänsyn tagen till hoppet och i heldragna kurvan ignoreras förekomsten av hopp i medelvärdet.

5. PARAMETERSKATTNING

Vi har vid skattningen av systemets parametrar använt oss av en RLS-baserad skattningsmetod som bygger på okorrelerade koefficienter. På grund av signalflödets utseende kallas algoritmen för "lattice" eller "ladder" algoritm. En sådan algoritm är mer komplex att implementera än en ordinär RLS, men har många fördelar (5.1). En kortfattad bakgrund till teorin återfinns i kap 5.2 och en mer fullständig (men begränsad) härledning finns i appendix A6. Algoritmen finns uppställd i appendix A5.

5.1 Latticefilter som skattningsmetod

Ett av de stora problemen är, som tidigare nämnts, systemets egenskap att ändra sin sanna ordning vid olika driftsituationer. Detta gör att det färdiga programmet enkelt måste kunna ändra sin modellordning efter systemets variationer. Det innebär att algoritmen, efter eventuell detektering av systemets sanna modellordning (se kap8), enkelt ska kunna skatta en ny modellordning.

Det finns ett antal sätt att tolka denna stuktur (se nedan) och det finns ett par stora (och några mindre) fördelar med en lattice-algoritm. I vår tillämpning är en fördel en minskad beräkningsbörda, $O(m^2)$ att jämföra med $O(m^3)$ för en ordinär RLS, vilket sparar värdefull tid [11]. Den stora fördelen är dock att metoden bygger på okorrelerade koefficienter, vilket möjliggör ändring av modellordning i realtid.

Lattice-implementation av ordningen N är egentligen en implementation av alla prediktorer upp till ordning N dvs i praktiken skattas N st ordningar "parallellt", vilket gör att en ändring av AR-modellens ordning under drift blir trivial. Man väljer helt enkelt bara hur många s.k. reflektionskoefficienter man vill använda sig av i parameteruppdateringen och slipper räkna fram en helt ny uppsättning för varje lägre ordning. Om man tex. väljer N=7 på lattice-strukturen och vill ha en AR-modell av ordning 5, använder man de fem första reflektionskoefficienterna och transformerar till AR-parametrar. Det gör att man kan skatta en konstant hög ordning av

"latticemodellen" och sedan vid behov uppdatera önskad AR-modell. Detta passar vår tillämpning ypperligt då dämpkvoten, dvs AR-modellen, bara behöver uppdateras tex. var 40:e sampel.

Dessutom är en Lattice-algoritm, till skillnad mot en ordinär RLS numeriskt stabil. Det ska inte föraktas i en on-line tillämpning av det här slaget, där numeriska fel lätt kan kan misstolkas, med tämligen dyra och onödiga driftstopp som värsta resultat. Vid valet av skattningsmetod har de ovan nämnda fördelarna övervägt, trots den mer komplexa strukturen hos algoritmen och därmed implementeringen. Angående de i kapitel 3 nämnda offline-resultaten kan nämnas att det ännu inte existerar någon lattice-algoritm för en ARMA-modell. Det närmaste man kommer är en ARX-algoritm, se referensen [7]

5.2 Teoretisk bakgrund

Det finns många sätt att härleda teorin för en latticestruktur. Det kan ses som en instrumentvariabelmetod [6] eller som en tolkning av Yule-Walker ekvationerna och lösningen av dem med hjälp av Levinson-Durbins algoritm [3],[6],[8]. Om man kan anse att en AR-modell är tillräcklig, se kap 3.3, kan man prediktera sina parametrar med hjälp av autokorrelationsfunktionen. Sambandet mellan AR-parametrar och autokorrelationen ges av Yule-Walker ekvationerna. Dessa löses effektivast av Levinsons algoritm. Alla metoder utnyttjar att Yule-Walker ekvationerna bildar s.k. symmetriska Toeplitz-matriser, vars matriselement är lika längs matrisens diagonaler. Det finns en naturlig uppdelning där Yule-Walker ekvationerna ger de s.k. reflektions-koefficienterna, se tex [6].

Man kan även närma sig det som ett basbyte där problemet med att öka en given modellordning är utgångspunkten[9]. Det finns mycket skrivet i litteraturen om "latticefilter" och en enkel härledning finns i apendix A6. Iden är som nämnts ovan att på något sätt få parametrarna okorrelerade. Det sker genom ett basbyte, där den "nya" modellen besitter just den egenskapen.

En AR-modell kan skrivas som

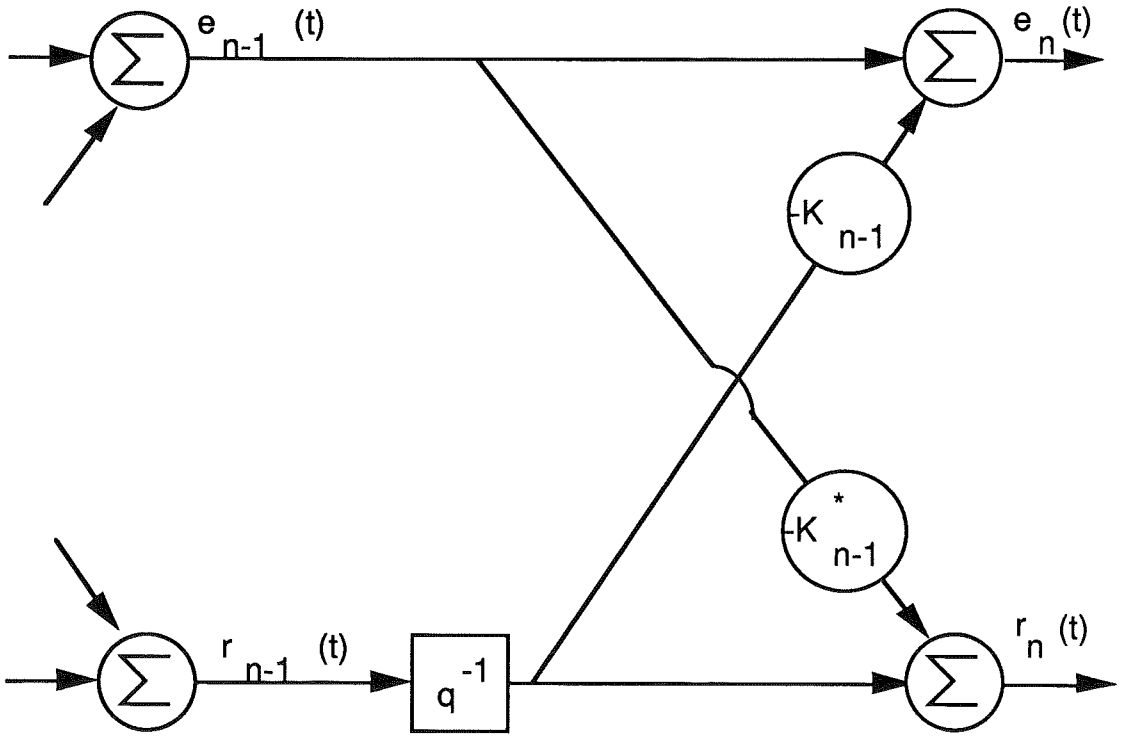
$$y = \theta^T \varphi(t) = y(t) + a_1 y(t-1) + \ldots\ldots + a_n y(t-n) = e(t) \qquad (4.1)$$

och i den "nya" modellen kallas parametrarna för reflektions-koefficienter och den nya regressorn betecknas .r_i:

$$\hat{y} = \tilde{\theta}^T \tilde{\varphi}(t) = K_o r_o(t-1) + \ldots + K_{n-1} r_{n-1}(t-1) \qquad (4.2)$$

där K_i är reflektionsparametrarna. Dessa har den sökta egenskapen att de är okorrelerade med varandra

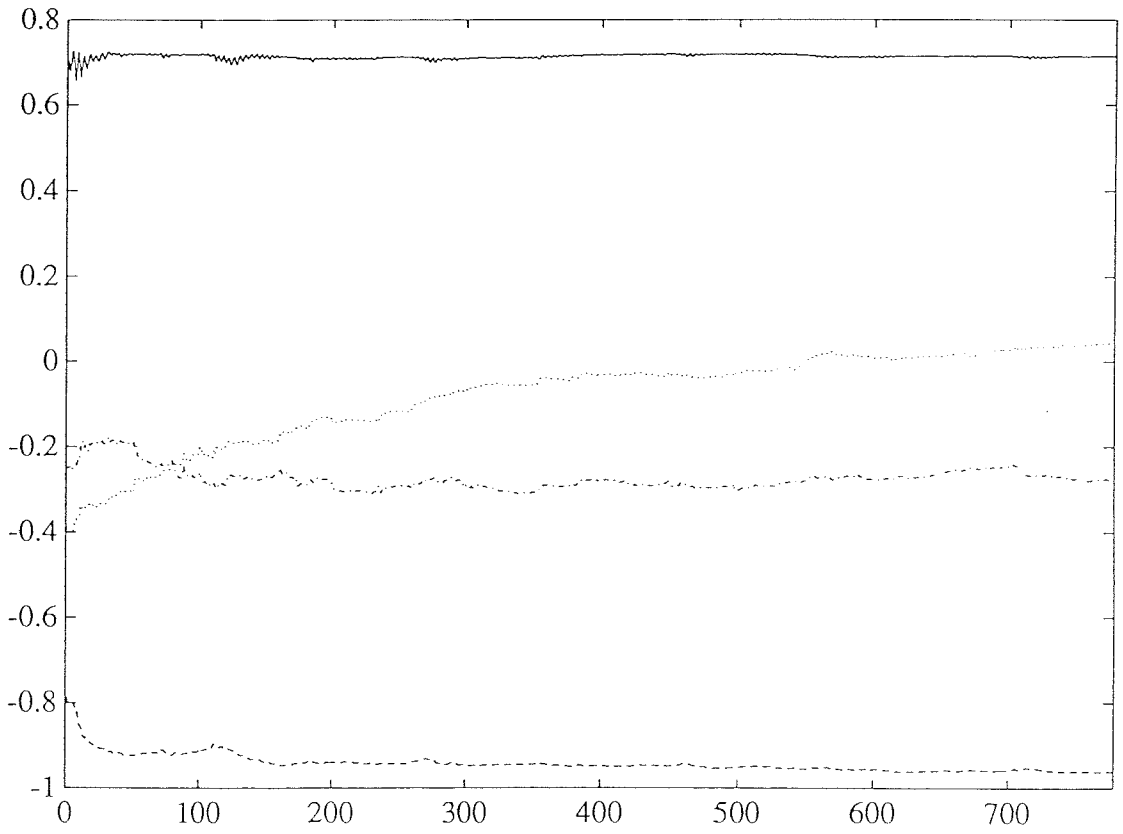
Vad som krävs nu är dels ett sätt att skatta $r_i(t)$ dels en översättning till de ursprungliga AR-parametrarna. Modellen (4.2) tillsammans med en algoritm att skatta $r_i(t)$ ur y(t), kallas "lattice" eller "ladder" struktur. Det användes från början för implementering av digitala filter och i andra signalbehandlings sammanhang. Namnet kommer från utseendet på den bild av signalflödet, som representerar en sådan struktur (se figur 5.1).



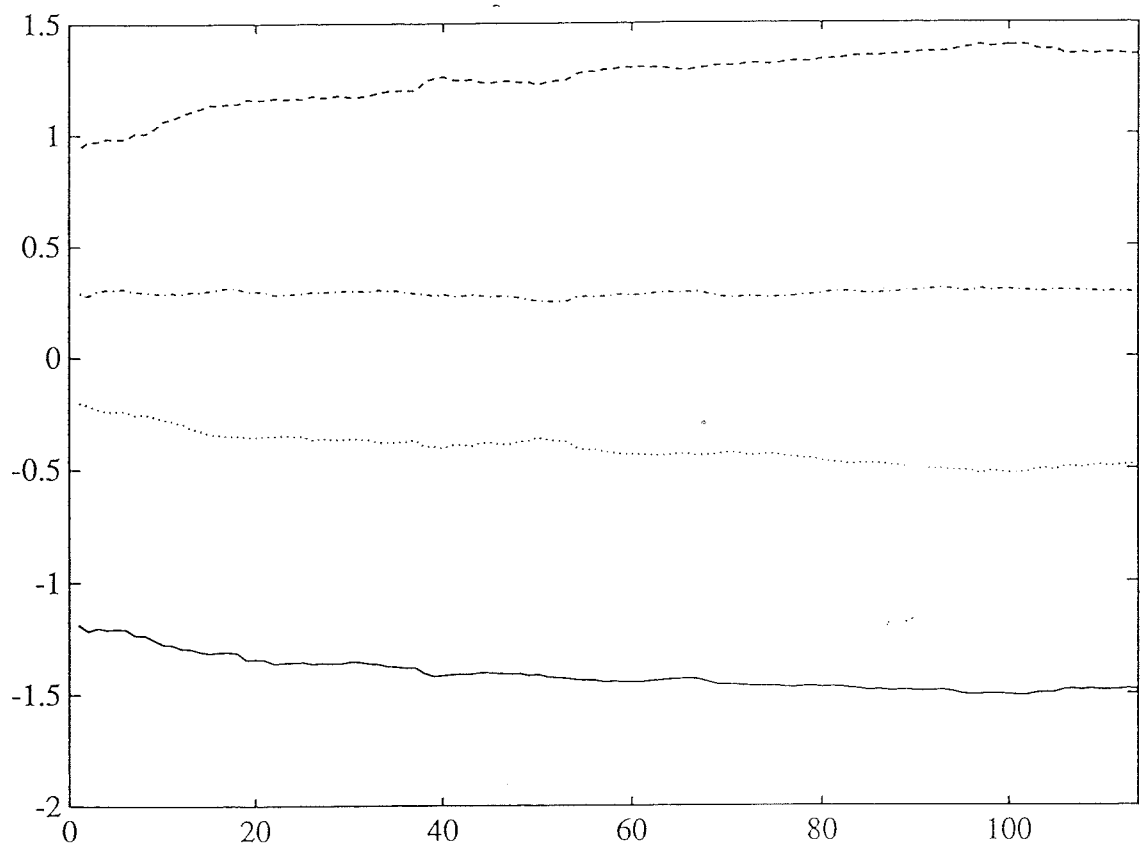
Figur 5.1 Signalflödet i en lattice-struktur

Algoritmen för skattning av $r_i(t)$, tillsammans med uppdatering av AR-parametrar och en tolkning av de ingående parametrarnas, återfinns i appendix A4. De variabler i algoritmen som ingår i figur 5.1 kan tolkas direkt. De är e, som är framåtprediktionsfel, r som är nya regressorn eller bakåtprediktionsfelet (se A8), och K som är reflektionsparametrar (en "bakåt" (K*) och en "framåt" (K)). Övriga variabler som ingår i algoritmen är varianser för framåt- och bakåtprediktionsfel samt korskovariansen mellan dessa. En utförligare beskrivning finns i appendix A4.

De skattade reflektionskoefficienterna uppvisar ett snabbt insvängningsförlopp och följer väl förändringar i systemet. I figur 5.2 visas exempel på reflektionsparametrar med tillhörande AR-parametrar.



Figur 5.2 Exempel på reflektionskoefficienter, ordning 4



Figur 5.3 Tillhörande AR-parametrar, ordning 4

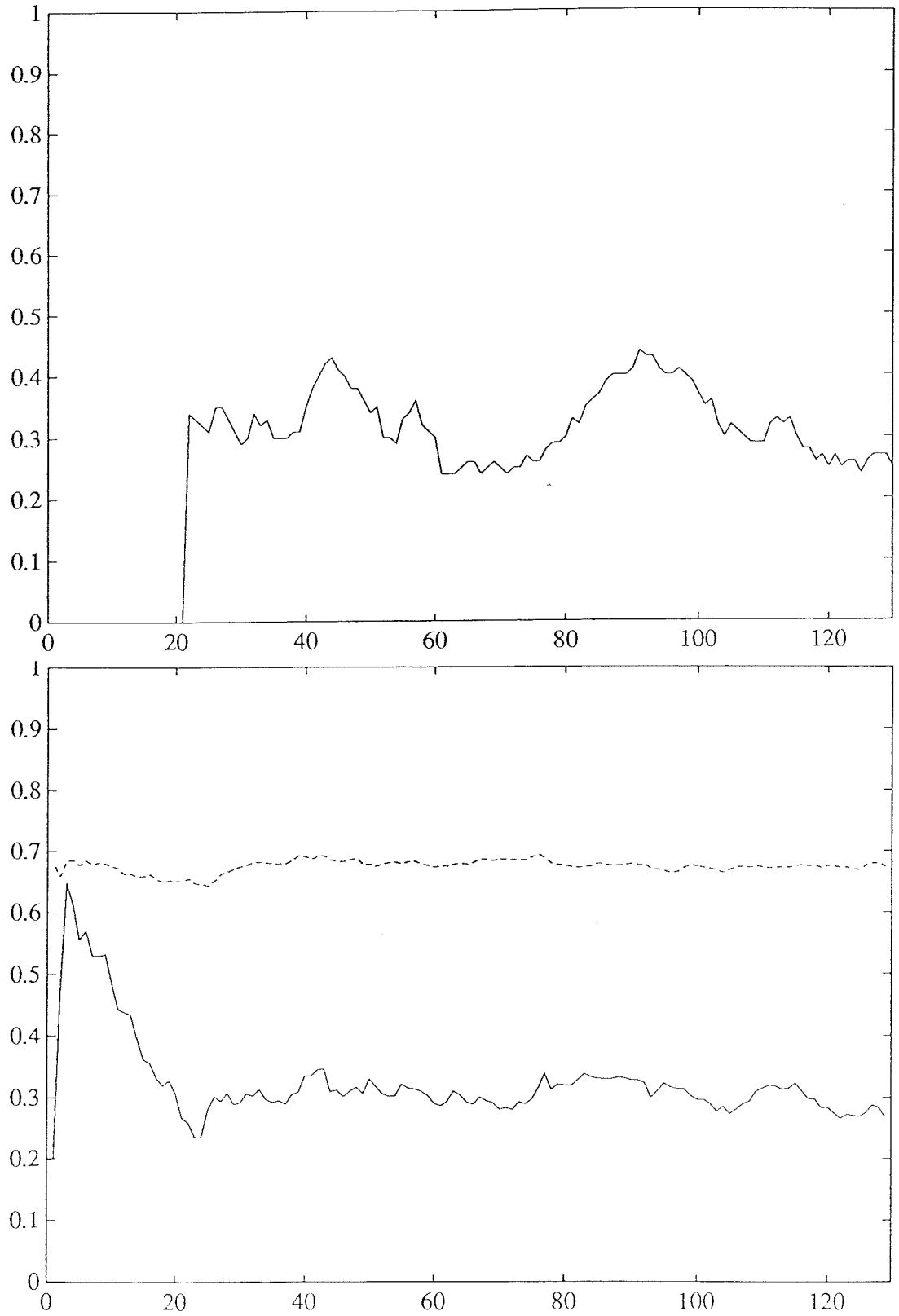
6. IDENTIFIERING

Avsikten med identifieringen av neutronflödet är att bestämma reaktorns stabilitet dvs dämpkvoten. En jämförelse mellan tidigare uppmätta dämpningar och de vi fått fram i våra försök, är den validering av modellen som vi anser betydelsefull (se kap. 6.2). I kap. 6.1 redovisas resultat av genomförda experiment med kommentarer om resultaten. Fler plottar över dämpkvoter återfinns i appendix A1. I de flesta fallen tyder jämförelserna med nuvarande system, på att modellen med stor sannolikhet beskriver dynamiken bakom neutronflödet i härden.

6.1 Experiment

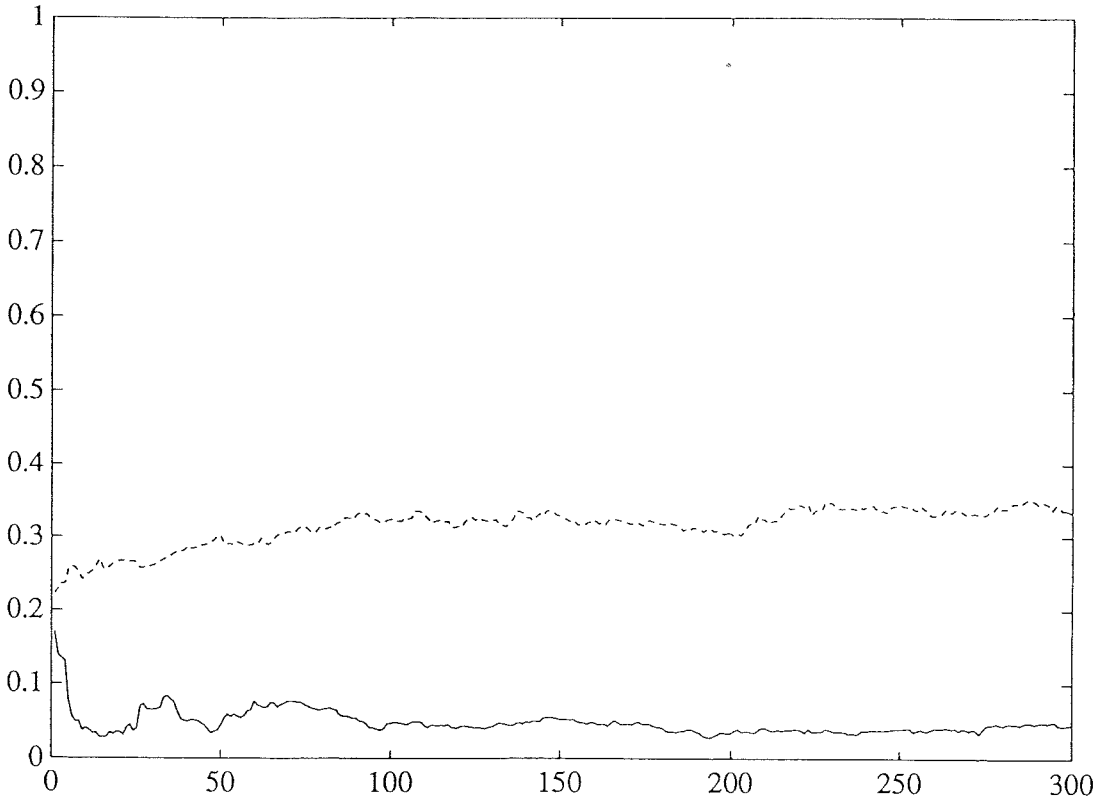
I figurerna som följer visas resultaten av dämpkvoten. De heldragna linjerna är dämpkvot och de streckade är resonansfrekvensen. Som jämförelse finns plottar över dämpkvoten, utvärderad med COSMOS (se kapitel 2) på exakt samma mätdata. Där finns inga frekvenser framtagna, men genomgående är frekvensen betydligt lättare att skatta och den stämmer mycket väl överrens med våra. De driftfall som används för identifieringen, är samtliga vid stationär drift. Speciella situationer som kan uppstå beskrivs i kapitel 8. Fler plottar finns i appendix A1 utan närmare kommentarer. De slutsatser som dras för exemplet nedan gäller även för övriga mätdata/simuleringar

Det är ett antal parametrar i det färdiga programmet som är empiriskt utprovade och redan angivna som konstanter i koden. Ett par parametrar måste anges vid start av programmet, nämligen maximal modellordning för latticealgoritmen samt ursprunglig sampelfrekvens. Olika insvängningsförlopp innebär att en jämförelse inte är möjlig förrän efter ca. sampel 20 i plottarna. För att få jämförbara resultat har vi i programmet (latticealgoritmen) en glömskefaktor $\lambda = 0.99875$. I figurerna motsvaras 20 sampel av 100 sekunders mättid.



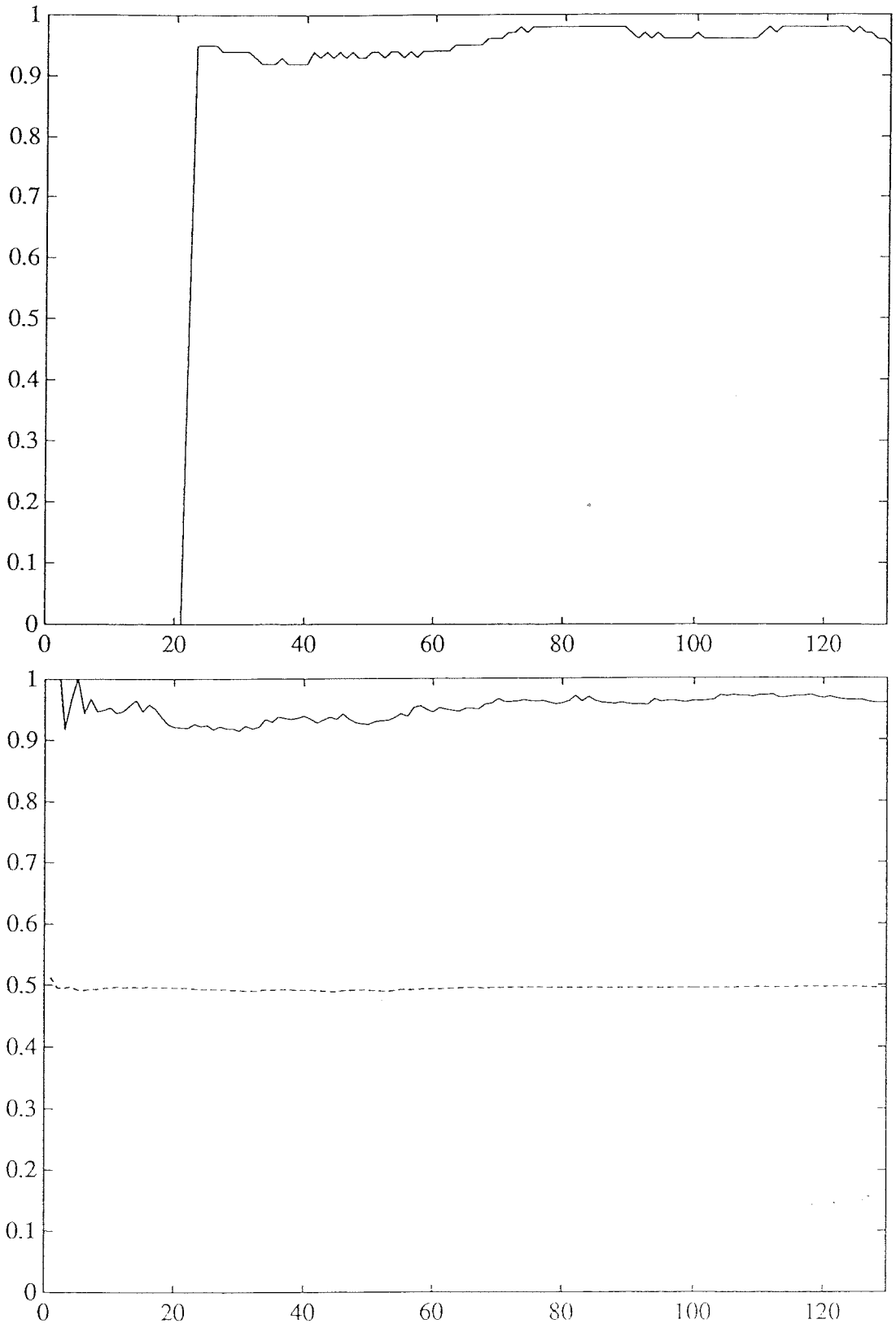
Figur 6.1 Skattad dämpkvot, ur COSMOS (övre) och med lattice-algoritmen (undre). Streckad kurva är frekvens och skalningen i x-led är antal sampel

Figur 6.1 på föregående sida är exempel på ett tämligen väl dämpat system. I den övre figuren är dämpkvoten utvärderad i COSMOS, medan den undre är skattad med "lattice"algoritmen. Det kan noteras att dämpkvoten uppvisar ett betydligt stabilare beteende när den skattas parametriskt. Detta kan dock stå i motsatsförhållande till snabbheten (responstiden) hos systemet vid processändringar, se kapitel 8. Även frekvensskattningen är mycket stabil.



Figur 6.2 Exempel på låg dämpkvot, skattad med latticealgoritmen. n=5

Figur 6.2 är exempel på mycket låg dämpkvot. Detta är dock normal dämpning under stationär drift vid full effekt. En stor fördel med en parametriska metod, är att den bevarar noggrannheten (liten varians) även för så här låga dämpkvoter.



Figur 6.3 Skattad dämpkvot, ur COSMOS (övre) och med lattice-algoritmen (undre). Streckad kurva är frekvens och skalningen i x-led är antal sampel

I figur 6.3 på föregående sida visas ett exempel på mycket hög dämpkvot. Denna situation inträffar under uppstart (se kapitel 2), men betyder panik under normal drift. Vår skattade dämpkvot är mycket stabil, och likheten med COSMOS-skattningen är stor. Även frekvensen är mycket stabil. Det bör dock påpekas, att så här höga dämpkvoter är lättare att skatta, då resonanstoppen är kraftig.

6.2 Validering

Syftet med modellvalidering är att på något sätt verifiera att modellen uppfyller de krav man kan ställa med avseende på avsikten med identifieringen. Avsikten med identifieringen av neutronflödet, är att bestämma reaktorns stabilitetsegenskaper, dvs dämpkvoten. Jämförelse mellan tidigare uppmätta dämpningar och de vi fått fram är den validering av modellen som vi anser meningsfull. I kapitel 6.1 och appendix A1 finns plottar som jämför tidigare uppmätta dämpkvoter med våra, både vad gäller utseende och storlek. I de flesta fallen tyder jämförelserna på att modellen med stor sannolikhet beskriver dynamiken bakom neutronflödet i härden och medföljande dämpkvot i jämförelse med nuvarande system.

En ytterligare validering som enligt vår mening bör göras, är av modellordningen. En av anledningarna till latticestrukturen var att kunna ändra modellordningen i realtid, vid uträkningen av dämpkvoten var tex. 40:e sampel. I latticealgoritmen skattar man i princip alla ordningar lägre än en maximal ordning parallellt och kan fritt välja hur många reflektionskoefficienter som ska användas vid parameteruppdateringen. Problemet blir nu att avgöra vad som är "rätt" modellordning. I vårt system är den varierande och får avgöras från fall till fall. De medel som står till buds är tex Akaike´s AIC och FPE, residualtest, F-test samt att titta på parametrarnas storlek och deras varians. Den enda praktisk användbara i vårt fall, pga AR-modell och lattice-strukturen, är att titta på parametrarna samt att jämföra dämpkvoterna. I allmänhet kan man misstänka för hög ordning om de sista parametrarna är små och har hög varians. Intuitivt kan man misstänka samma sak, om sista parametern är mycket mindre än övriga. Detta kan användas som en sorts validering av modellordning.

En annan möjlig metod att validera sin modellordning är balanserad realisering och modellreduktion. Balanserad realisering innebär kortfattat att man gör en transformation $z_k=Tx_k$, dvs [6]

$$S: \begin{cases} z_{k+1} = T\Phi T^{-1} z_k + T\Gamma u_k \\ y_k = CT^{-1} z_k \end{cases}$$

De s.k gramianerna för detta system är då:

$$P_z = TPT^T$$
$$Q_z = T^T Q T^{-1}$$

Man väljer transformationen T sådan att

$$P_z = Q_z = \Sigma_i \quad \text{med} \quad \sigma_i = \sqrt{\lambda_i(PQ)}$$

där $\lambda_i(PQ)$ är egenvärden av matrisen PQ och Σ är en diagonalmatris med element σ_i . De beskriver relativ vikt av respektive tillstånd och man kan stryka rad j och kolumn j i en tillståndsform med små element σ_j i Gramianen. Se mer om detta i [6]. Det innebär att om modellen har en ordning som är större än den sanna ska det synas i gramianens diagonalelement och det blir ett sätt att validera modell-ordning. Man får dessutom reda på vilka tillstånd som kan strykas. I tabell 6.1 visas gramianernas storlek för några mätdataserier. Genom-gående för dessa gramianer, och för övriga kurvor, är att det är svårt att entydigt avgöra vilka tillstånd som kan strykas, dvs vilken modellordning som är den rätta. Det gör att man tillsammans med parametrarnas storlek (som inte heller ger entydiga besked) och dämpkvotens utseende får avgöra den modell-ordning som är lämplig för varje enskilt fall. Dessa försök är offline-försök och inte använd-bara i onlinetillämpningen. De ger dock resultatet (som tidigare misstänkts) att den "sanna" modellordningens variationer är för-hållandevis små i de testade fallen.

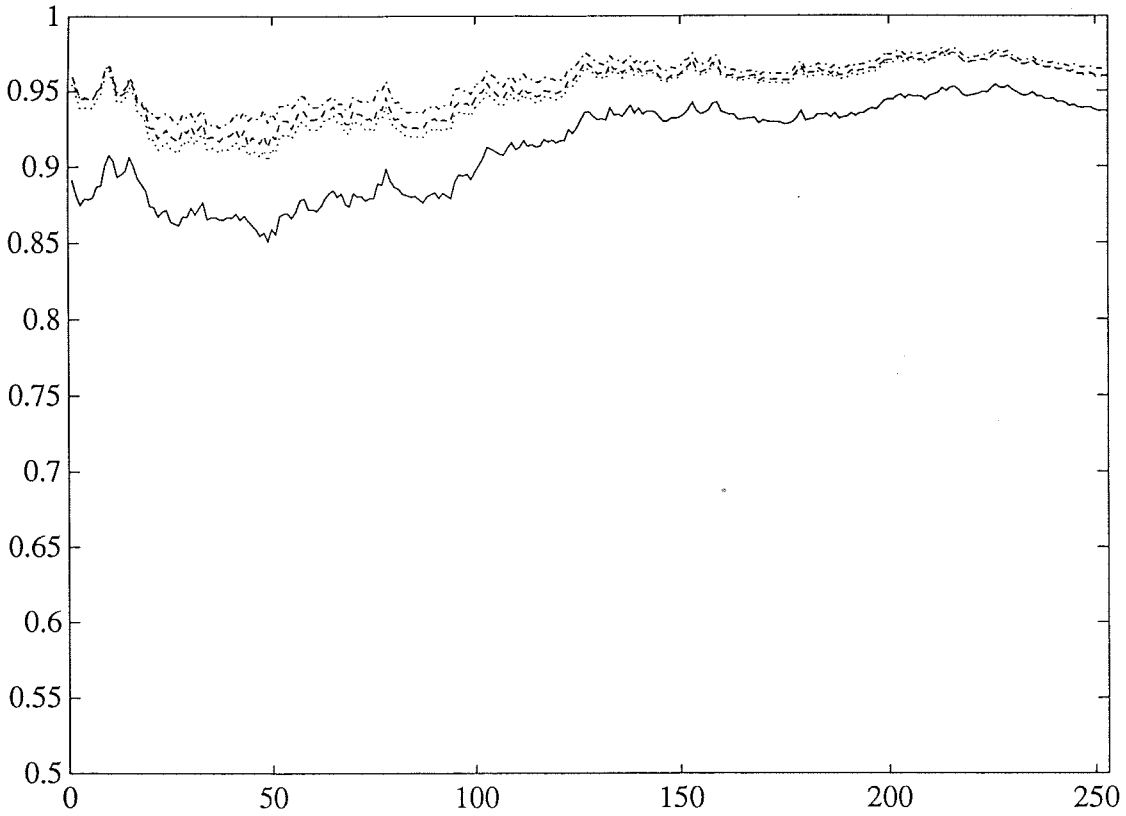
ordning	serie 1	serie 2	serie 3	serie 4
1	1.2955	25.348	18.380	23.807
2	0.7115	24.556	17.696	23.189
3	0.2029	2.9885	0.1634	0.0982
4	0.0297	0.1247	0.1247	0.0568
5	0.0239	0.1034	0.0427	0.0181
6	0.0008	0.0416	0.0265	0.0019
7	0.0002	0.0026	0.0000	0.0000

Tabell 6.1. Gramianernas storlek för några olika kurvor.

I tabell 6.2 ser man att för mätserie1 kan ordning 5 förväntas, för mätserie 2, 5 eller 6, för mätserie 3 ordning 6 och för mätserie 4 antagligen ordning 5. Sammantaget följer all data denna trend, nämligen ganska liten skillnad i modellordning och inte helt entydiga besked. Detta är dock den "validerings-metod" som ger mest information.

Sammanfattningsvis kan man säga att validering av modellen är ganska svårt eftersom vi valt en AR-modell, dvs ingen insignal finns att tillgå. Olika dataserier har givit olika ordningar, även där dämpningen är ungefär densamma, och det är som sagt dämpkvotens utseende som från fall till fall fått avgöra, vilken ordning vi ansett vara bäst i det enskilda fallet.

I ett framtida system vill man ha någon form av automatisk ordningsbestämning, och de tester vi gjort på modellordning är inte användbara i en realtidstillämpning. Ett viktigt resultat är att dämpkvoten trots allt inte varierar så mycket för olika val av modell-ordning (ca 5-8) i de fall vi tittat på (se figur 6.4). Ett visst systematiskt fel insmyger sig dock, men detta anser vi borde gå att få bort med hjälp av någon skattning av modellordningen. Detta bör enligt oss utvecklas mera och i kapitel 8 framlägger vi en del ideer om detta.



figur 6.4 Skattning av dämpkvot med olika modellordningar (4-7). Ordning 4 heldragen, 5-7 streckade

7. IMPLEMENTERING

Algoritmerna är implementerade dels i matlab (7.1) dels i C (7.2). Utvecklingen är mestadels gjord i matlab och sedan översatt till C. C-koden arbetar i realtid (on-line) medan matlabskoden är "styckvis" i realtid (se 7.1).

7.1 Matlab-funktioner

De funktioner som är skrivna i MATLAB listas nedan. Rutinerna är styckvis realtidsanpassade, dvs de olika momenten (nedsampling, offsetskattning, parametersakttning etc.) arbetar i realtid var för sig. Man får med andra ord köra dessa program var sig för hela mätdata-serier och flytta resultaten mellan de olika momenten.

RLS	Skattar insignalens offset med Minsta Kvadrat algoritm och subtraherar detta från insignalen. Sätter även styrsignal till LATTICE 2 för av-stängning vid hopp i offset
LATTICE 1	Parameterskattning med latticefilter. Arbetar utan styrsignalen från RLS.
LATTICE 2	Parameterskattning med latticefilter. Version med avstängnig vid hopp i medelvärde. Kräver styrsignal från RLS.
ARUPDATE	Uppdatering av AR-parametrar. Intervallet bestäms av variabeln "every" i LATTICE 1 eller LATTICE 2.
DRUPDATE	Löser rötterna till det skattade A-polynomet. Tar fram det dominerande polparet och beräknar dämpkvoten. Anropas direkt efter ARUPDATE.
DR	Beräknar dämpkvoten ur det dominerande pol-paret. Används av DRUPDATE

7.2 C-program

C-koden är utvecklad, och arbetar, under operativsystemet VENIX, som är en realtidsversion av UNIX. För att kunna integrera koden i nuvarande COSMOS-systemet är alla rutiner implementerade i denna miljö. En beskrivning av de funktioner som ingår presenteras nedan. I appendix A6 finns en utförligare beskrivning av koden tillsammans med en programlistning

Följande tre huvudprogram finns implementerade:

DRRATIO1	Filtrerar insignalen genom 30:e ordningens FIR-filter. Samplar ner 3 ggr. Identifiering genom latticefilter. Uppdatering av AR-parametrar samt beräkning och presentation av dämpkvot.
DRRATIO2	Filtrerar insignalen genom 30:e ordningens FIR-filter. Samplar ner 2 ggr. Identifiering genom latticefilter. Uppdatering av AR-parametrar samt beräkning och presentation av dämpkvot.
DRRATIO3	Filtrerar insignalen genom 30:e ordningens FIR-filter. Samplar ner 3 ggr. Identifiering genom latticefilter med "varierande glömske-faktor" (se kapitel 8.2). Uppdatering av AR-parametrar samt beräkning och presentation av dämpkvot.

Huvudprogrammen använder följande uppsättning underprogram:

FIRINIT1	Används av DRRATIO1 och DRRATIO3 för att initiera 30:e ordningens FIR-filter med gränsfrekvensen = ursprungliga sampelfrekvensen/6.
FIRINIT2	Används av DRRATIO2 för att initiera 30:e ordningens FIR-filter med gränsfrekvensen = ursprungliga sampelfrekvensen/4.

FIRFILT	Filtrerar signalen genom FIR-filter (ordning 30) före nedsampling.
RLSINIT	Initiering av offsetskattningen. RLS Skattar insignalens offset med rekursiv minstakvadrat (RLS) samt subtraherar denna från insignalen.
LATTINIT	Initiering av latticealgoritmen.
LATTICE	Uppdatering av latticeparametrarna. Detta sker varje sampel av den nedsamplade signalen.
ARUPDATE	Uppdatering av AR-parametrar. Intervallet mellan uppdateringarna kan väljas med hjälp av huvudprogrammets variabel "every" (default 10), se A5
DRUPDATE	Löser rötterna till det skattade A-polynomet. Tar fram det dominerande polparet och beräknar dämpkvoten. Anropas direkt efter ARUPDATE.
NEWLAMBDA	Används endast i DRRATIO3. Kontrollerar om glömskefaktorn i latticealgoritmen skall ändras eller ej, se vidare kapitel 7.2

Underprogrammet DRUPDATE använder följande uppsättning underprogram:

RPZERO	Rotlösning av polynom. Möjligheter finns att snabba upp rotlösningen genom att skicka in föregående rötter som initialvärden (se kapitel 7.2).
DR	Beräknar dämpkvoten ur det dominerande polparet.

8. SLUTSATSER

8.1 Slutsatser

Syftet med examensarbetet var att undersöka om en parametrisk metod kunde användas för identifiering av dämpkvot. De resultat vi erhållit vid experiment tyder på att det är fullt möjligt, men att nya typer av problem uppstår, som tex. val av modellordning. Lattice-algoritmen ger stabila modellparametrar och har ett snabbt insvängningsförlopp. Förekomsten av offset, och snabba förändringar i densamma, på mätdata tas effektivt omhand av algoritmerna och ger inga försämringar av modellskattningen. Prestanda med avseende på tidskrav verkar också vara tillräckliga. Med tillräcklig prestanda menas i detta fall att alla beräkningar går snabbare än 5 sekunder i realtid. Här får man dock komma ihåg att vårt program behandlar en signal. Ett färdigt system ska mäta och beräkna dämpkvoter på upp till 32 kanaler parallellt och denna prestanda är inte utprovad i realtid ännu.

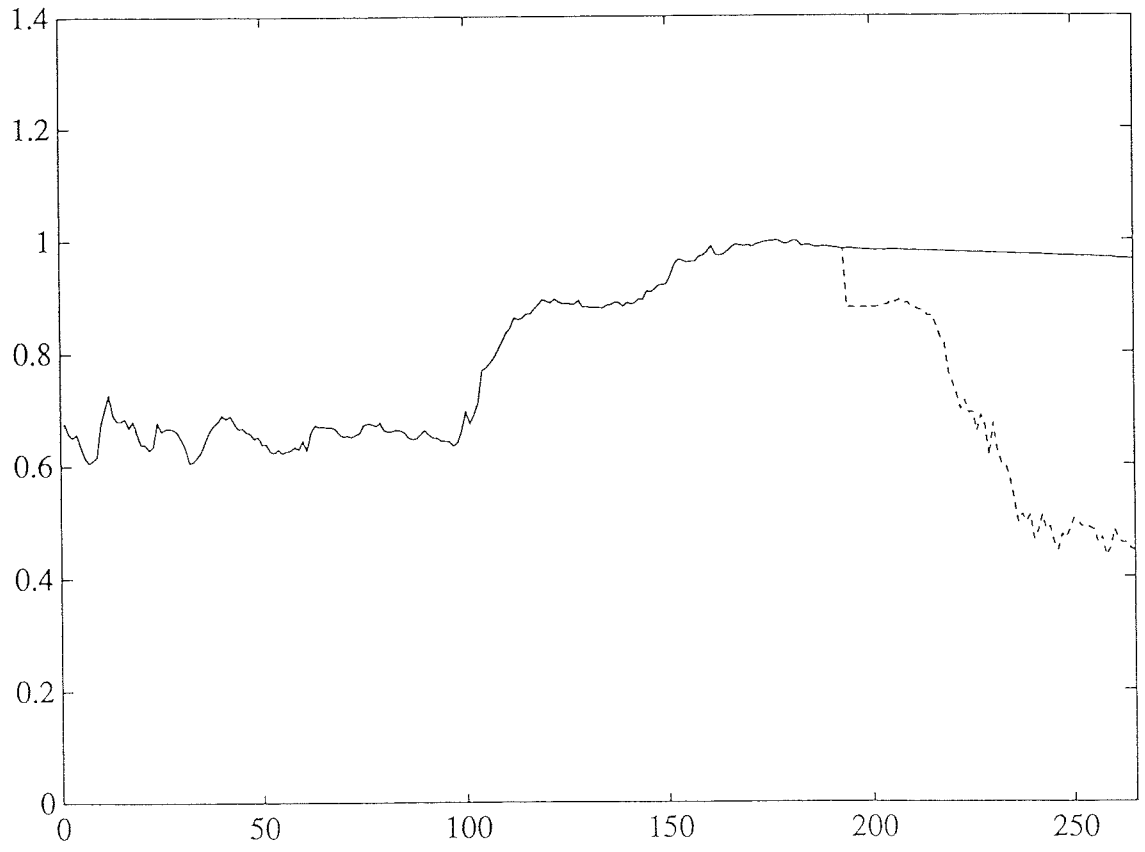
Studerar man plottar över vår skattade dämpkvot jämfört med dämpkvoter från COSMOS (se kap 6.1 samt apendix A1) är skillnaderna mycket små. Det kan noteras att de dämpkvoter vi har erhållit har ett något stabilare uppförande, framför allt vid låga värden (DR < 0.5). Att svårigheterna ökar vid skattning av låga dämpkvoter är känt och kvarstår i viss mån, om man jämför de resultat vi fått inbördes med varandra. Jämfört med nuvarande metod är dock dessa låga dämp-kvoter stabilare (vilket är positivt).

8.2 Förslag till förbättring och utveckling

Det finns naturligtvis en del kvar att förbättra, och framförallt att vidareutveckla i vårt arbete. Det ryms emellertid inte inom ramen för detta examensarbete. När det gäller skattning av dämpkvoten finns i huvudsak två problem kvar att behandla. Det ena är det faktum att metoden är långsam vid en övergång från hög till låg dämpkvot när skillnaden i dämpning är stor. Det andra problemet att behandla är bestämning av modellordning i realtid. Vid beräkning av modellens poler, dvs A-polynomets nollställen, används en

rotlösningsalgoritm. Algoritmen som används är förhållandevis tidskrävande och bör optimeras. Ett enkelt förslag till sådan åtgärd finns nedan.

En övergång från låg till hög dämpning innebär i spektrum att den "nya" resonanstoppen har betydligt lägre amplitud. Detta faktum gör att så länge algoritmen "kommer ihåg" data från den högre dämpningen, modelleras den "gamla", kraftigare resonansen. Med andra ord identifierar man på "fel" data. Figur 8.1 är ett, ur verkligheten taget, exempel på detta där dämpkvoten går från 0.99 till ca 0.50. Tydligt är att skattningen är korrekt tom. sampel 200 där värdet på dämp-kvoten blir fel. Det ska påpekas att fenomenet uppträder först vid relativt stora ändringar.



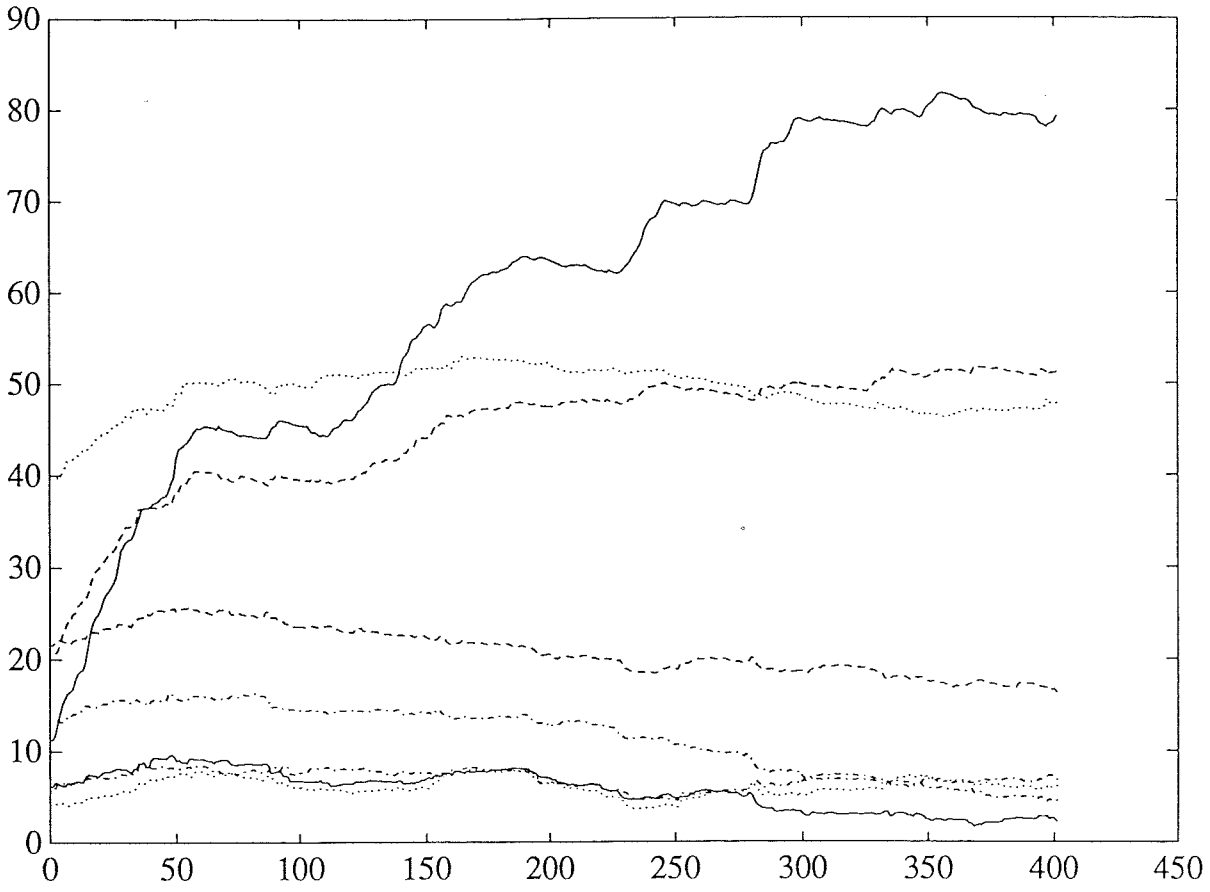
Figur 8.1 Exempel på felskattning vid övergång från hög till låg dämpkvot Heldragen kurva är "konstant glömskefaktor" och streckad är "varierande glömskefaktor" (enligt kapitel 8.2.). N=5

En lösning på problemet är att minska glömskefaktorn vid en sådan situation. Vi noterade att en variabel i lattice-algoritmen minskade med minskad dämpkvot. Genom att använda lutningen på denna signal (prediktionsfelet framåt, se mer kap 5), kan man eventuellt detektera en sådan situation. Rent praktiskt sker det genom att derivera variabeln för att på så sätt få fram lutningen. Denna ide finns implementerad i programmet DRRATIO 3 och är testad på en dataserie, se figur 8.1. Den stora nackdelen med metoden är att variabelns lutning inte entydigt följer värdet på dämpkvoten. Tyvärr ger detta en minskning av glömskefaktorn när det inte ska ske och det stör skattningen ordentligt. För att metoden ska kunna fungera måste man göra ytterligare analyser på fler mätdata för att fastställa om sambandet var tillfälligt för dessa data.

Kvar att utveckla är också en metod att bestämma modellordningen i realtid. Det är ett omfattande ämnesområde och värt ett examensarbete i sig. Skattningen av modellordning bör helst ske med de latticeparametrar som finns, då de uppdateras varje sampel. Det är konstaterat att prediktionsfelen minskar även om man passerar den sanna ordningen, men enligt [8] bör någon av varianserna anta ett minimum. Den parameter vi eventuellt tror på är korskovariansen mellan framåt och bakåtprediktionen (se mer i kapitel 5). Vi har studerat denna signal och den antyder att ett minimum faktiskt existerar och dessutom byter ordning vid driftändring (läs; ändring av dämpkvot). Det är dock en hypotes och bör enligt vår mening utredas mycket mer ingående innan några slutsatser kan dras. Figur 8.2 visar denna signal där dämpkvoten ändras vid sampel 275 och man ser att en annan ordnings korskovarians antar minimum.

Den första förändring som bör göras av programmet gäller rotlösningsalgoritmen. I den kan man välja mellan att ange startvärden eller om den själv ska söka upp dessa. Att använda föregående rötter som startvärde vid ny rotlösning är mycket tidsbesparande. Eftersom tidskraven är stora på detta system är det viktigt att åtgärda.

Kvar att tillägga är att man kanske skulle önska någon form av varians för den beräknade dämpkvoten. Ett resultat är ju i allmänhet ganska ointressant, om man inte kan precisera noggrannheten i det. Att beräkna varianser för dämpkvoten är dock knappast möjligt och de jämförelser som gjorts i kap 6 får därför tjäna som ett noggrannhetsmått.



Figur 8.2 Exempel på korskovarians mellan prediktion för olika ordningar (n = 3-10)

REFERENSER

[1] Andersson Peter (1985): "Adaptive forgetting in recursive identification through multiple models", *IEEE Trans. on automatic control,* 15:10

[2] Blomstrand J. (1990): "Oskarshamn2. Härdstabiltetsmätningarna i september 1989", Rapport UR 90-143, ABB Atom, Västerås

[3] Friedlander B. (1982): "Lattice Filters for Adaptive Processing", *proc. of the IEEE*, vol 70, no8, pp 829-863

[4] Gustavsson F. (1990): "Optimal Segmentation of linear regression parameter", Report LIU-tek-lic-1990:46, Institutionen för systemteknik, Linköpings Tekniska Högskola

[5] Hägglund Tore (1983), "New estimation techniques for adaptive control", Report LUTFD2/TFRT-1025, Institutionen för reglerteknik, Lunds Tekniska Högskola.

[6] Johansson R.(1990): *Processidentifiering*, Kurslitteratur, Institutionen för reglerteknik, Lunds Tekniska Högskola.

[7] Karlsson Erlendur (1987), "ARMA Modeling of Linear Time-Varying Systems: Lattice Filter Structures and Fast RLS Algoritm", *IEEE Trans. on acoustic, speech and signal processing*. Vol.ASSP-35, no 7, pp 994-1014.

[8] Kay S.M och Marple S.L (1981): "Spectrum Analysis - A modern perspective", *Proceedings of the IEEE,* Vol 69, No 11, pp1380-1414

[9] Ljung L och Söderström T (1983): *Theory and Practice Recursive of identification,* MIT press, Cambridge MA

[10] Salomonsson G. m.f (1987): *Tidsdiskreta Kretsar och signaler,* Institutionen för Teletransmissionsteori, Lunds Tekniska Högskola.

[11] Stoica P. och Söderström T.(1989): *System Identification,* Prentice Hall, New York.

[12] Zhang H., Duhamel P.och Tressens S. (1990): "An improved Burg-type recursive lattice method for autoregressive spectral analysis", *IEEE on acoustics, speech and signal processing*, Vol 38, No 8, pp1437-1445.

[13] Åström K.J och Eykhoff .P (1971): "System Identification-A survey", *Automatica*, Vol.7, pp123-162

[14] Åström K.J. och Wittenmark B.(1989): *Computer Controlled systems*, Prentice Hall, New York

[15] Reaktorteknisk OrienteringsKurs (ROK), ABB ATOM, kap. 5.

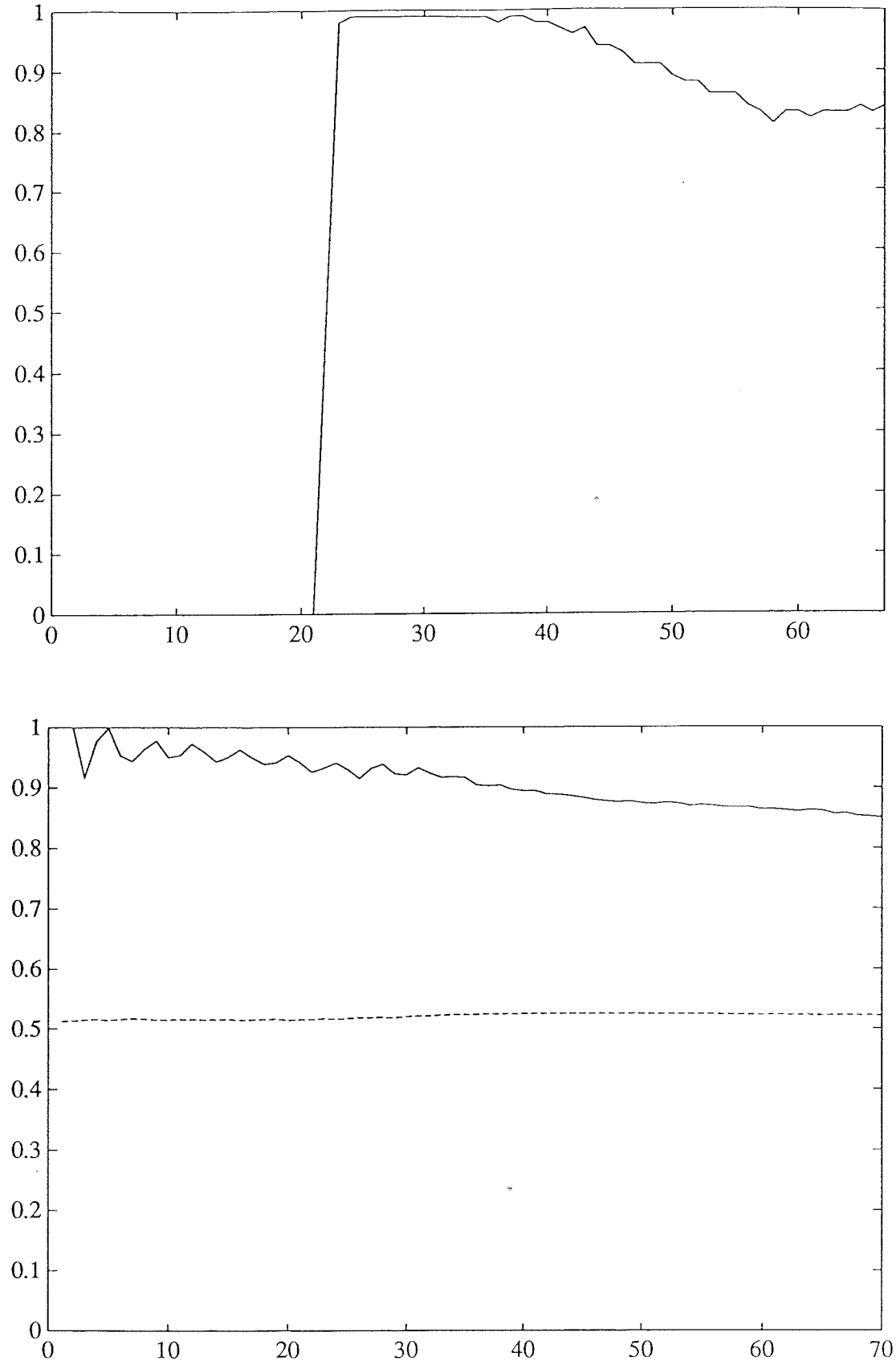
[16] Kärnkraftsäkerhet och utbildning, Nyköping 1988, " Så här fungerar en kokvattenreaktor"

[17] "Online core stability monitoring system", ENC'90. *ENS/ANS - Foratom Conference Transactions,* Vol 3, pp 1316-1318

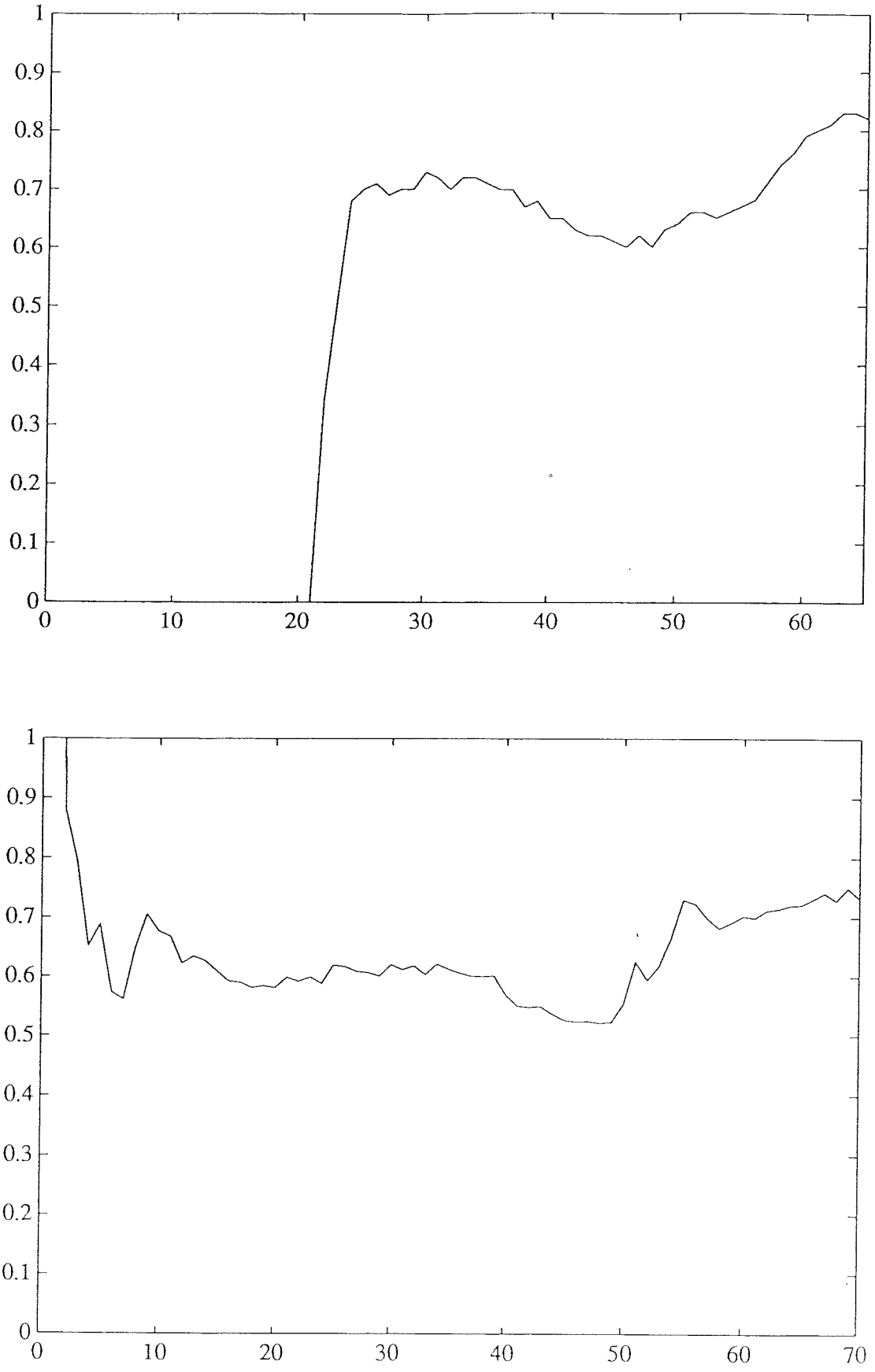
A1. IDENTIFIERINGAR AV DÄMPKVOT

Nedan följer ett antal plottar som innefattar skattade dämpkvoter och frekvenser för ett antal olika driftfall och reaktorer. Vi har inte givit några utförligare kommentarer till dessa (se kapitel 6). Som jämförelse visas även tillhörande dämpkvot, skattad i COSMOS till de flesta kurvorna. Dessa kurvor är resultatet av vårt examensarbete och vid jämförelse med tidigare skattningar framgår tydligt de slutsatser om resultatet vi dragit. Dessa redovisas i kapitel 6 och 7.

Figur A1 är ytterligare ett exempel på problemet med en dämpkvot som går från hög till låg, som disskuterats i kapitel 8.

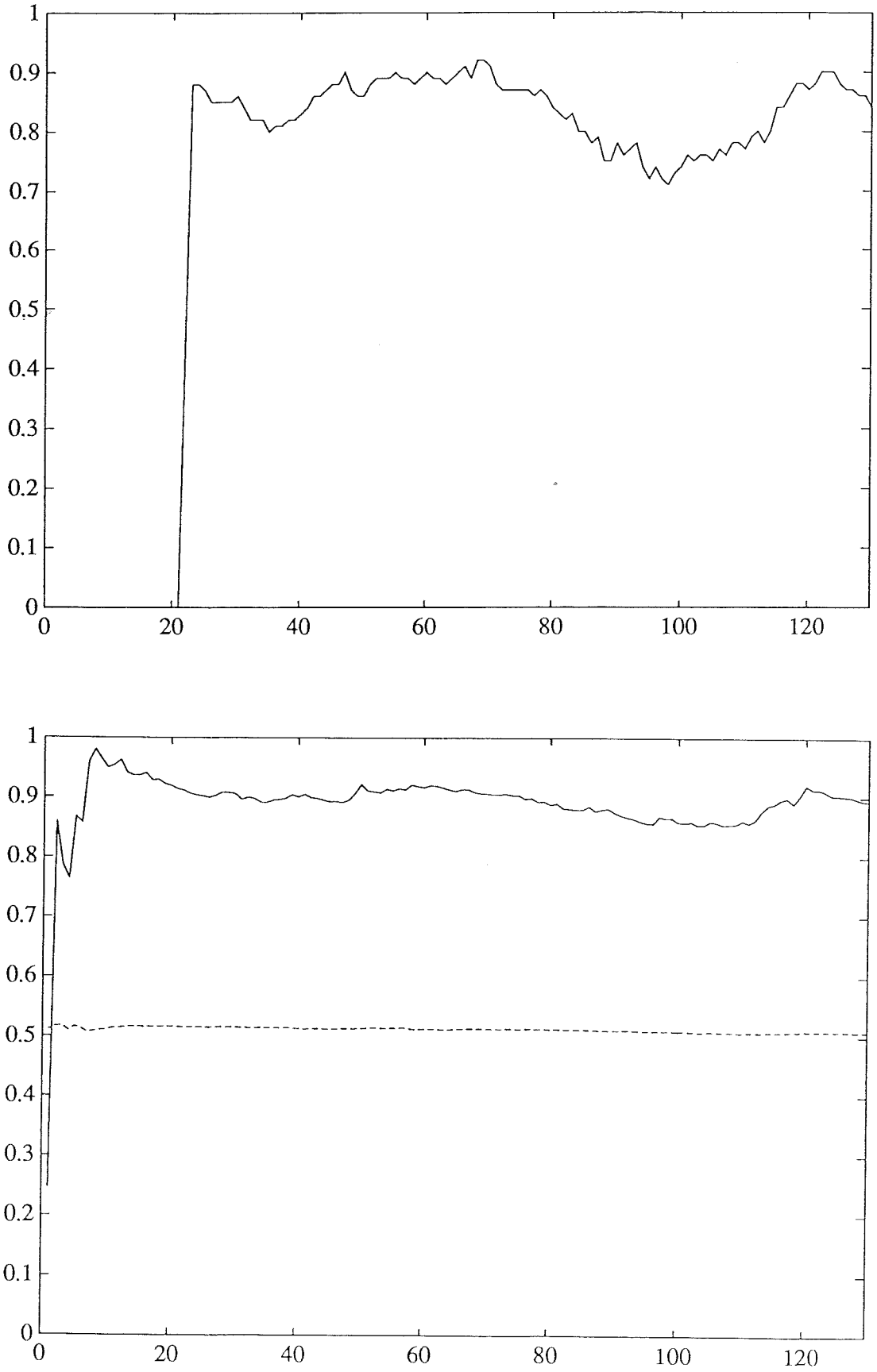


Figur A1. Exempel på felskattning vid ändrad dämpkvot, som nämns i kap 8.2. Övre är dämpkvot ur COSMOS och undre är skattning med lattice-algoritmen (dämpkvot hel, frekvens streckad)

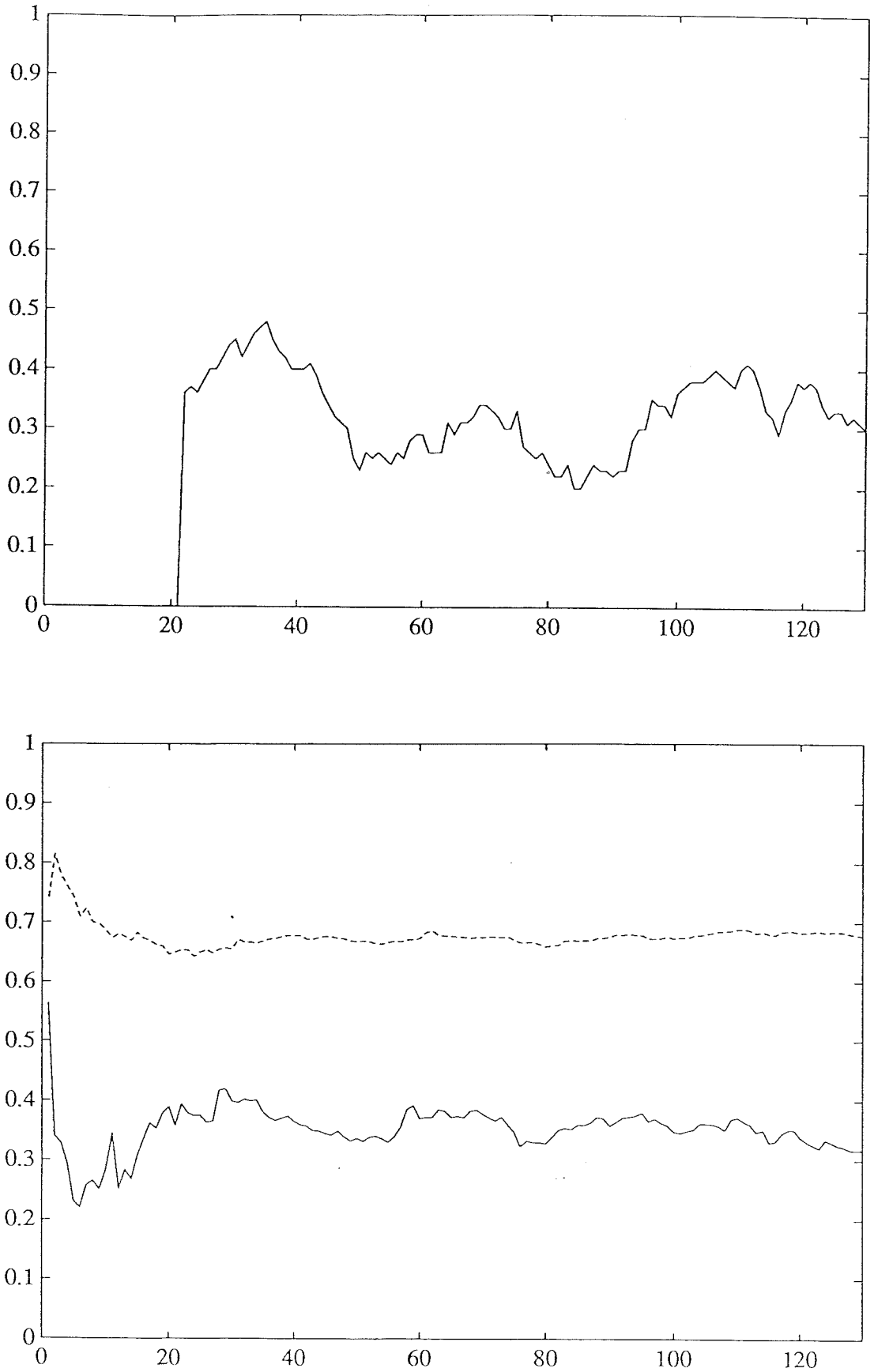


Figur A2. Skattad dämpkvot, ur COSMOS (övre) och med lattice-algoritmen (undre). Skalningen i x-led är antal sampel

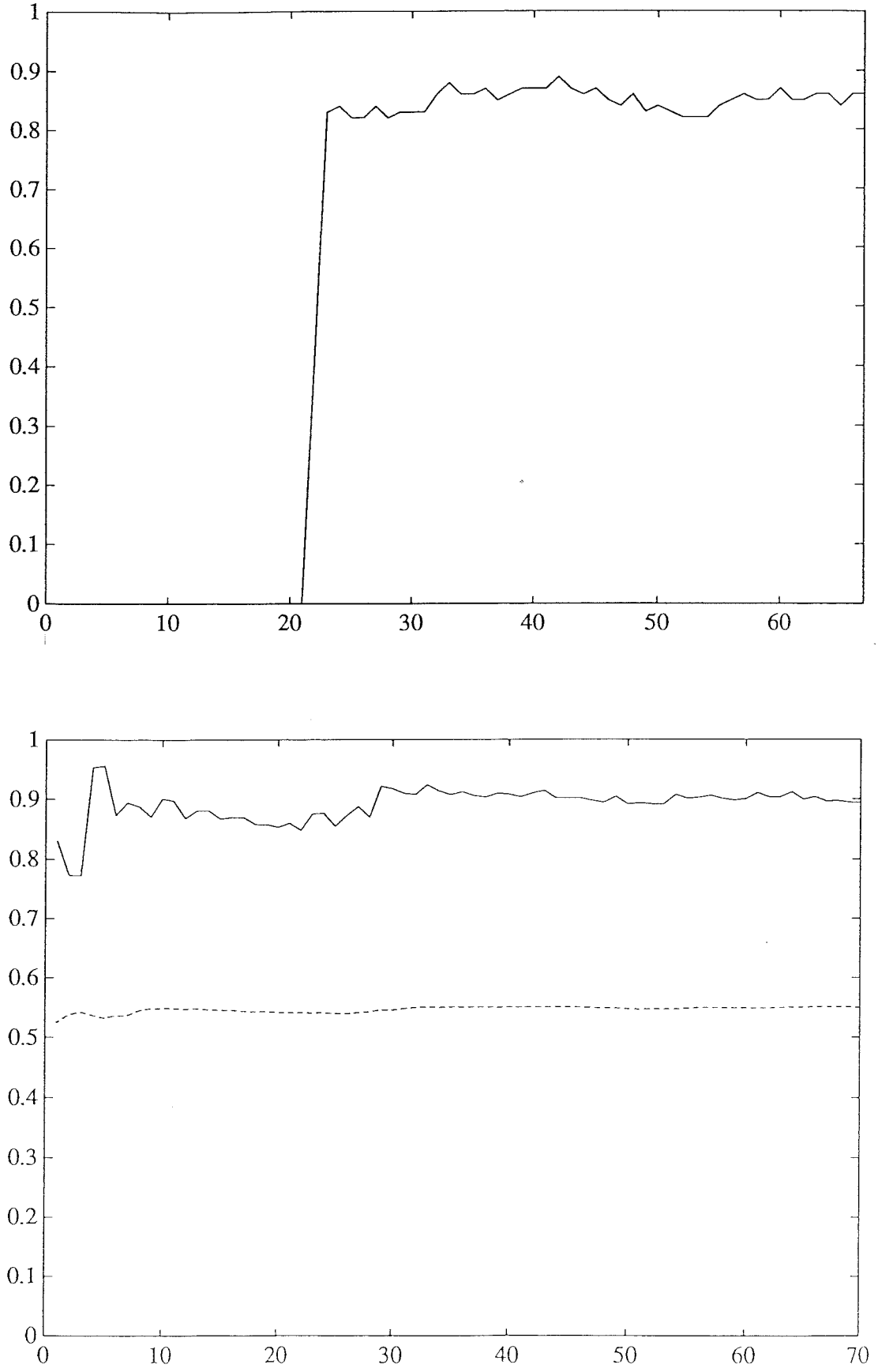
A1 Identifiering av dämpkvot



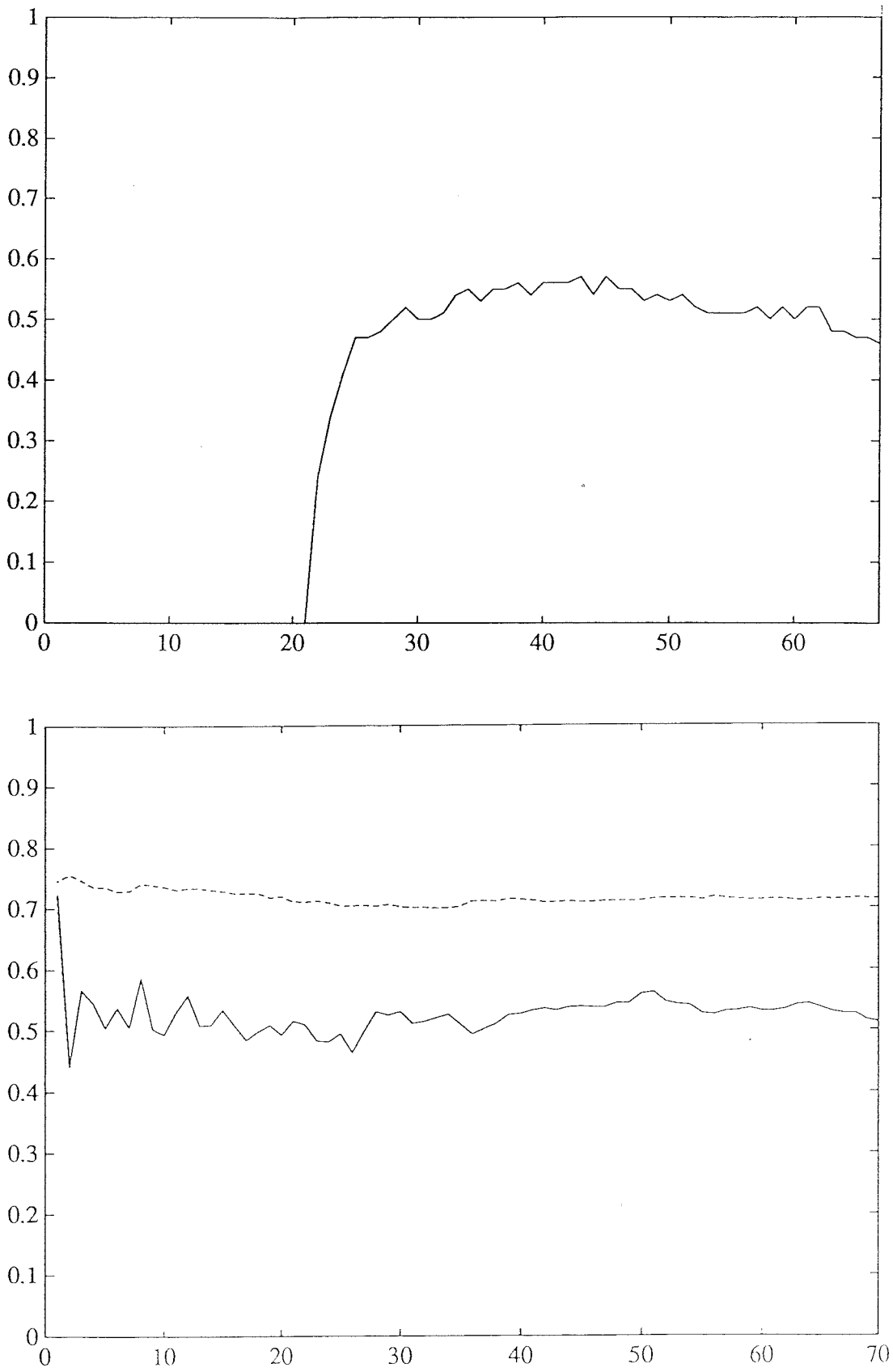
Figur A3. Skattad dämpkvot, ur COSMOS (övre) och med lattice-algoritmen (undre). Streckad kurva är frekvens och skalningen i x-led är antal sampel



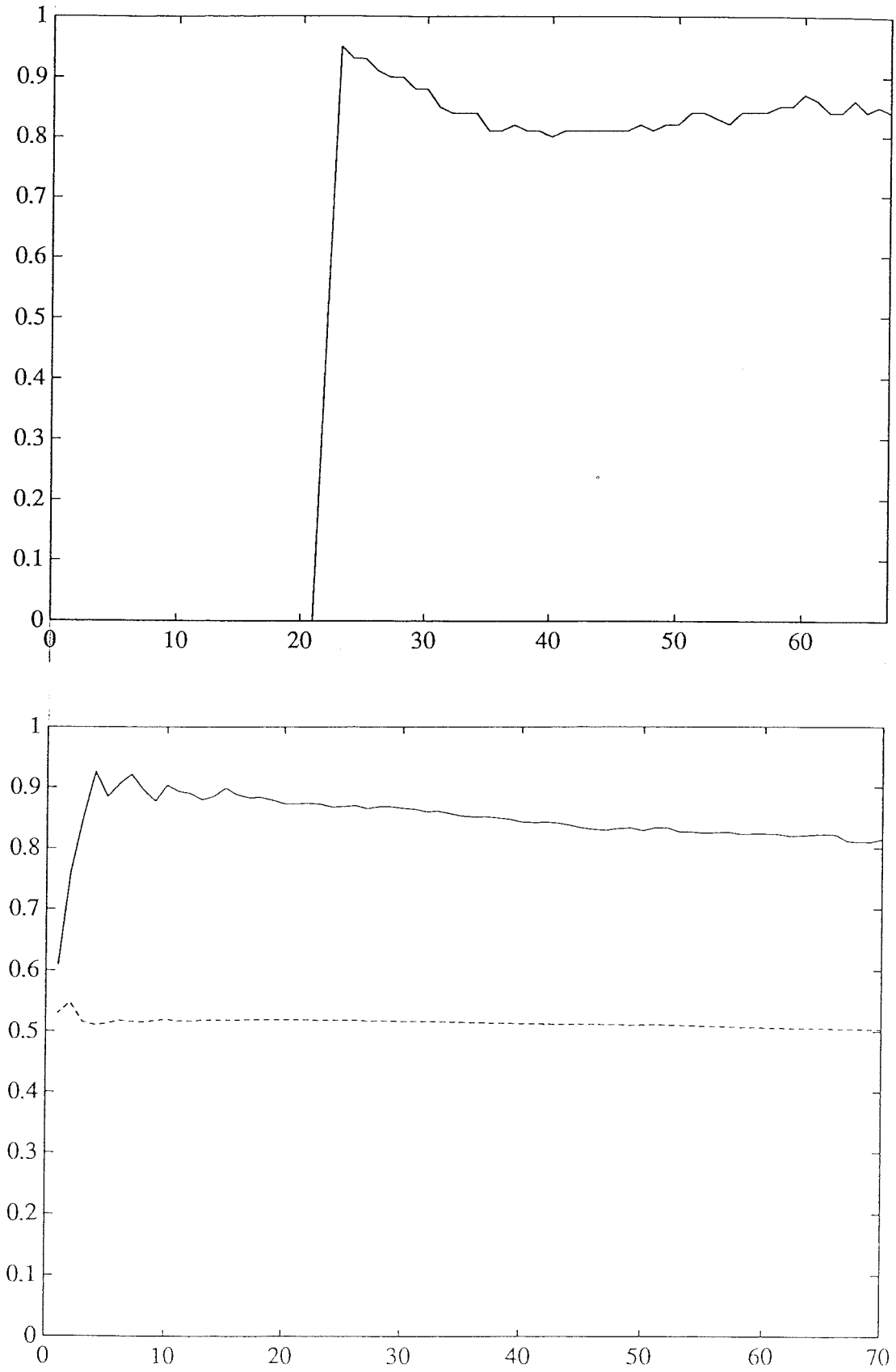
Figur A4. Skattad dämpkvot, ur COSMOS (övre) och med lattice-algoritmen (undre). Streckad kurva är frekvens och skalningen i x-led är antal sampel



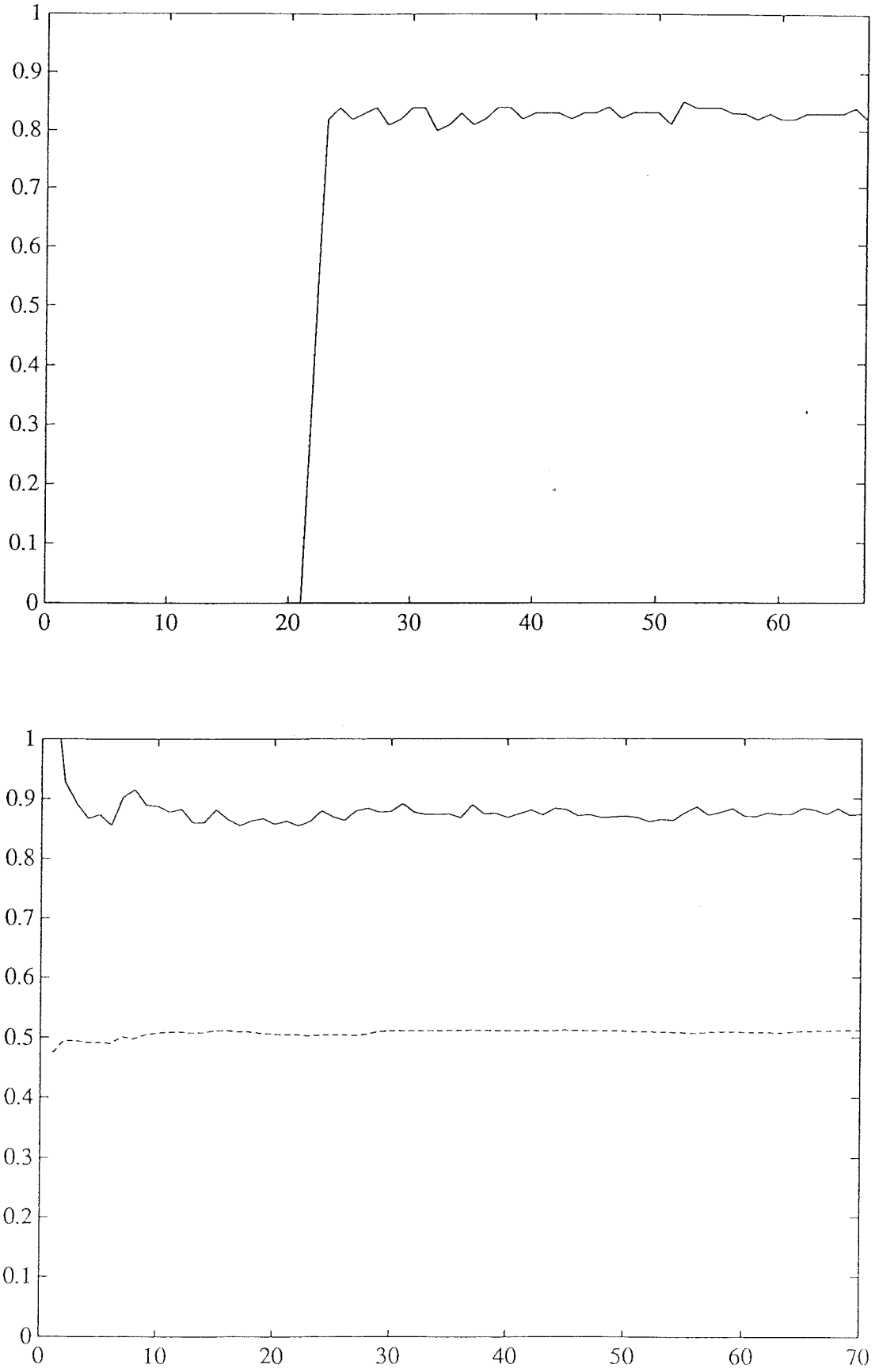
Figur A5. Skattad dämpkvot, ur COSMOS (övre) och med lattice-algoritmen (undre). Streckad kurva är frekvens och skalningen i x-led är antal sampel



Figur A6. Skattad dämpkvot, ur COSMOS (övre) och med lattice-algoritmen (undre). Streckad kurva är frekvens och skalningen i x-led är antal sampel



Figur A7. Skattad dämpkvot, ur COSMOS (övre) och med lattice-algoritmen (undre). Streckad kurva är frekvens och skalningen i x-led är antal sampel



Figur A8. Skattad dämpkvot, ur COSMOS (övre) och med lattice-algoritmen (undre). Streckad kurva är frekvens och skalningen i x-led är antal sampel

A2. Härledning av dämpkvot för andra ordningens system

Vi utgår från ett andra ordningens system:

$$G(s) = \frac{\omega_o^2}{s^2 + 2\zeta\omega_o s + \omega_o^2} \qquad (a.1)$$

Stegsvaret blir för $\zeta < 1$

$$Y(t) = 1 - \frac{e^{-\zeta\omega_o t}}{\sqrt{1-\zeta^2}} \sin(\omega_d t + \phi) \qquad (a.2)$$

där $\omega_d = \omega_o\sqrt{1-\zeta^2}$ och ϕ uppfyller $\cos\phi = \zeta$.

Deriveras ekvationen (a.2) med avseende på tiden, får man impulssvaret för systemet:

$$g(t) = \frac{\zeta\omega_o\, e^{-\zeta\omega_o t}}{\sqrt{1-\zeta^2}} \sin(\omega_d t + \phi) - \frac{\omega_d\, e^{-\zeta\omega_o t}}{\sqrt{1-\zeta^2}} \cos(\omega_d t + \phi) \qquad (a.3)$$

Trigonometrisk hjälpformel:

$$A \sin\alpha + B \cos\alpha = R \sin(\alpha + \theta) \qquad (a.4)$$

där

$$R = \sqrt{A^2 + B^2}$$

$$\theta = \arctan B/A$$

Kombineras (a.3) med (a.4) får man:

$$R = \sqrt{\frac{\zeta^2 \omega_o^2 e^{-2\zeta\omega_o}}{1 - \zeta^2} + \frac{\omega_d^2 e^{-2\zeta\omega}}{1 - \zeta^2}}$$

$$\theta = \arctan \frac{-\omega_d}{\zeta \omega_o} \qquad (a.5)$$

Där R är amplituden för impulssvaret och $(\phi + \theta)$ fasvinkeln för detta. Insättning av amplituden och fasvinkeln i ekvation (a.3) ger:

$$g(t) = \frac{\omega_o e^{-\zeta\omega_o t}}{\sqrt{1 - \zeta^2}} \sin\left(\omega_d + \phi - \arctan \frac{\sqrt{1-\zeta^2}}{\zeta}\right) \qquad (a.6)$$

För att beräkna dämpkvoten, är det toppar i impulssvaret som är intressant. Man vet att fasvinkeln för toppar i en sinusfunktion är multiplar av $\pi/2$, dvs

$$\omega_d + \phi - \arctan \frac{\sqrt{1 - \zeta^2}}{\zeta} = \pi/2 + 2n\pi \qquad (a.7)$$

Två godtyckliga på varandra följande toppar kan man beräkna då argumentet är $(\pi/2 + 2n\pi)$ resp. $(\pi/2 + 2(n+1)\pi)$ för motsvarande

toppar vid tiderna t_1 och t_2. Beräkning av amplituderna A_1 och A_2 kräver först att ur motsvarande argument beräkna t_1 och t_2.

$$\omega_d t_1 + \phi - \arctan \frac{\sqrt{1-\zeta^2}}{\zeta} = \pi/2 + 2n\pi \qquad \text{(a.8)}$$

$$\omega_d t_2 + \phi - \arctan \frac{\sqrt{1-\zeta^2}}{\zeta} = \pi/2 + 2(n+1)\pi \qquad \text{(a.9)}$$

Nu kan man beräkna dämpkvoten enligt följande:

$$DR = \frac{A_2}{A_1} = \frac{h(t_2)}{h(t_2)} = \frac{e^{-\zeta\omega_o t_2}}{e^{-\zeta\omega_o t_1}} = e^{\zeta\omega_o(t_1-t_2)} \qquad \text{(a.10)}$$

Där (t_1 - t_2) ges av (a.8) och (a.9) till

$$t_1 - t_2 = - 2\pi / \omega_d \qquad \text{(a.11)}$$

Insättning av $\omega_d = \omega_o\sqrt{1-\zeta^2}$ i (a.11) ger

$$t_1 - t_2 = \frac{-2\pi}{\omega_o\sqrt{1-\zeta^2}} \qquad \text{(a.12)}$$

Detta samband och ekvationen (a.10) ger uttrycket för dämpkvoten i det tidskontinuerliga fallet. Alltså är dämpkvoten:

$$DR = e^{-2\pi\zeta/\sqrt{1-\zeta^2}} \qquad (a.13)$$

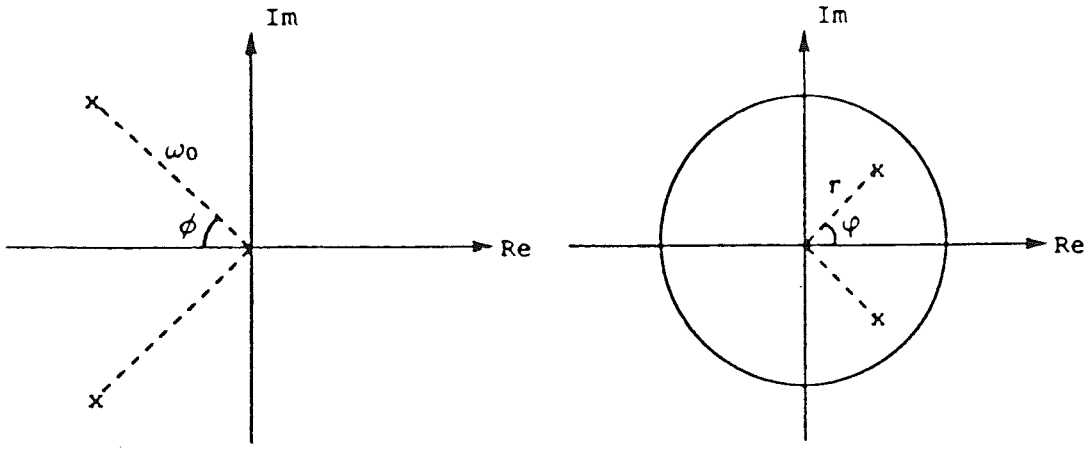
Detta uttryck är oberoende av n, dvs för ett andra ordningens system gäller det för vilka, på varandra följande, toppar som helst.

Sambandet (a.13) gäller ej för högre ordning än två. För ordningar högre än två studeras det dominerande polparet enligt kap 2.2. Med hjälp av ovanstående formel beräknas ett approximativt värde på dämpkvoten som vi kallar "asymptotisk dämpkvot".

Samband med tidsdiskret modell

Nedan följer sambandet mellan poler och dämpkvot i tidsdiskreta fallet. Polerna för ett andra ordningens system med överförings-funktion enligt (a.1) är:

$$s = \omega_o(-\zeta \pm i\sqrt{1-\zeta^2}) \qquad (a.14)$$



Figur A.1 Kontinuerliga och diskreta poler för 2:a ordningens system

Detta är polerna i det kontinuerliga fallet, se figur A.1. Vi har ett samband mellan kontinuerliga och diskreta poler enligt:

$$z_i = e^{s_i T} \qquad i = 1,2,3, \ldots, n \tag{a.15}$$

Använder vi ekvation (a.15) och sätter in den av de tidskontinuerliga polerna med positiv imaginärdel enligt (a.14), får vi:

$$z = e^{\omega_o(-\zeta+i\sqrt{1-\zeta^2})T} \tag{a.16}$$

Detta är motsvarande tidsdiskreta pol, som kan förenklas enligt följande:

$$z = e^{-\omega_o \zeta T} \cdot e^{i\omega_o \sqrt{1-\zeta^2}}$$

där, se figur A1.

$$\begin{cases} r = e^{-\omega_o \zeta T} \\ \varphi = \omega_o \sqrt{1-\zeta^2 T} \end{cases}$$

Utifrån de två ovanstående ekvationerna kan vi enkelt räkna fram ζ och $\sqrt{1-\zeta^2}$,

$$\begin{cases} \zeta = -\ln r/\omega_o T \\ \sqrt{1-\zeta^2} = \varphi/\omega_o T \end{cases}$$

Division av de två ovanstående ekvationerna ger oss

$$\frac{\zeta}{\sqrt{1-\zeta^2}} = \frac{-\ln r}{\varphi} \tag{a.17}$$

Insättning av (a.17) i sambandet (a.13) ger

$$DR = e^{2\pi \ln r/\varphi} \tag{a.18}$$

Detta är uttrycket för för dämpkvoten i tidsdiskreta fallet som vi använder oss av för bestämning av aktuell dämpning i identifieringen.

A3. HÄRLEDNING AV LATTICEFILTER

Som nämnts i kapitel 5 är avsikten med denna struktur att få okorrelerade parametrar. Det finns ett antal sätt att närma sig detta på och vi har valt ett som vi anser lämpligt. Det är en ganska kort genomgång och mer finns att läsa i bla. [6], [8], [9], [11] och [13]. Vi har valt en härledning där problemet med att öka en given ordning är utgångspunkten. Man kan visa att en ökning av AR-modellens ordning, förbi den sanna, ger systematiska fel i skattningarna (se sid 348-349 i [9]).

Anta en Autoregression:

$$y(t) + a_1 y(t-1) + \ldots\ldots + a_n y(t-n) = e(t) \qquad (4.1)$$

Att använda y(t-k) som regressor är naturligt, då det är verkliga mät-data. Detta val har emellertid nackdelen att MK-skattningen av a_i beror på ordningen n. Anta att man har skattat en n:te ordningens AR-modell:

$$\hat{y}_n(t) = \theta_n^T \varphi_n(t) \qquad (4.2)$$

Man ökar nu ordningen till n+1 och adderar till regressorn y(t-n-1):

$$\hat{y}_{n+1}(t) = \theta_n^T \varphi_n(t) - a_{n+1} y(t-n-1) \qquad (4.3)$$

Denna regressor är typiskt korrelerad med $\varphi(t)$, dvs med tidigare värden av y(t-i), $1 \leq i \leq n$, men innehåller också ny information som inte finns i $\varphi(t)$.

Detta kan skrivas som:

$$y(t-n-1) = - B^T\varphi(t) + r_n(t-1) \qquad (4.4)$$

där $r_n(t-1)$ är den nya informationen i $y(t-n-1)$ som inte finns i $\varphi(t)$, dvs $r_n(t-1)$ och $\varphi(t)$ är okorrelerade med varandra. Med (4.4) insatt i (4.3) fås:

$$\hat{y}_{n+1}(t) = (\theta_n + a_{n+1}B)^T\varphi(t) - a_{n+1}r_n(t-1) \qquad (4.5)$$

När nu parametern a_{n+1} justeras för att ta hänsyn till den nya informationen i $r_n(t-1)$, måste även $\hat{\theta}_n$ justeras i förhållande till (4.2) på grund av termen $a_{n+1}B$ i (4.45). Med andra ord, om $\hat{\theta}_n^{(n)}$ avser skattningen av θ_n i (4.2) och $\hat{\theta}_n^{(n+1)}$ avser skattningen av θ_n i (4.3), ger det

$$\hat{\theta}_n^{(n+1)} + \hat{a}_{n+1} B = \hat{\theta}_n^{(n)} \qquad (4.6)$$

Med vetskapen om att en skattning av parametrar i en autoregression beror på modellordning är det nu enkelt att konstruera en metod, som inte har denna egenskap. Om man, när man utökar sin modell till att gälla för $y(t-n-1)$, adderar endast den nya informationen $r_n(t-1)$ istället för $y(t-n-1)$, blir skattningen av θ (de "gamla" parametrarna) oförändrad. Tanken är att konsekvent använda $r_n(t-1)$, $k=0, \ldots, n$, som regressor istället för $y(t-k-1)$. Ur (4.4) ser man att $r_n(t-1)$ kan tolkas som bakåtprediktionsfel, till prediktionen av $y(t-n-1)$ ur $\varphi^T(t) = (-y(t-1) \ldots . -y(t-n))$. Man ser också att $r_n(t-1)$ är en linjärkombination av $y(t-k-1)$. Därav:

$$\tilde{\varphi}(t) = \begin{bmatrix} r_n(t-1) \\ \vdots \\ r_n(t-1) \end{bmatrix} = T\varphi(t) \qquad (4.7)$$

Dvs transformationen från originalregressorn y(t-k) till bakåt-prediktionsfelen $r_n(t-1)$, är ett linjärt basbyte. Motsvarande byte av koefficienter är

$$\begin{bmatrix} K_0 \\ \vdots \\ K_{n-1} \end{bmatrix} = \tilde{\theta} = T^{-T}\theta \qquad (4.8)$$

där koefficienterna K_i kallas för reflektions-koefficienter. Modellen blir nu

$$\hat{y} = \tilde{\theta}^T \tilde{\varphi}_t = K_o r_o(t-1) + \ldots + K_{n-1} r_{n-1}(t-1) \qquad (4.9)$$

Vad som behövs nu är ett sätt att skatta $r_i(t)$. Modellen (4.9) tillsammans med en algoritm att skatta $r_i(t-1)$ ur y(t), kallas lattice eller ladder struktur. Se figur 4.1. Algoritmen för skattning av $r_i(t)$, tillsammans med uppdatering (transformationen) till AR-parametrar finns beskriven i appendix A4.

A4. LATTICEALGORITM

Nedan följer den latticealgoritm som är implementerad i programmet. Det är en RLS-baserad algoritm, egentligen algebraiskt identisk, och kallas för " Fast Least square lattice". Se [11] och [3] för utförligare beskrivningar och härledningar. "Fast" står för det faktum att beräkningsbördan är mindre än för en ordinär realtidsmässig RLS-algoritm (se sid 361-2 i [11]). Detta gäller emellertid enbart uppdateringen av reflektionsparametrarna.

Metoden går ut på att konstruera vektorn $r_n(t-1)$ som bakåt-prediktionsfelet av sekvensen y(t), baserat på n framtida värden, dvs använda en regressor (se A3):

$$y(t - n) = -B_n^T \varphi_n(t+1) + r_n(t)$$

A7.1 LSL, "Least square Lattice" - algoritm som används i skattningen av reflektionsparametrarna:

In-parametrar:

N = maximala ordningen av lattice

y_T = data sekvens vid tiden T

λ = exponentiell viktfaktor

Variabler:

$R^{\varepsilon}_{p,T}$, $R^{r}_{p,T-1}$ = variansen för framåt / bakåt prediktionerna

$\Delta_{p,T}$ = korrelationskoefficient

$\gamma^{c}_{p,T} = 1 - \gamma_{p,T}$ = Likelihood-variabel

$\varepsilon_{p,T}$, $r_{p,T-1}$ = framåt / bakåt prediktionsfel

$K^{\varepsilon}_{p,T}$, $K^{r}_{p,T-1}$ = framåt / bakåt reflektions-koefficienterna

där index p är ordning och T tid.

Nedanstående rekursion sker för varje tidpunkt T (T=0,. . ,Tmax). dvs För varje tid, T, gör:

Initiera:

$\varepsilon_{0,T} = r_{0,T} = y_T$

$R^{\varepsilon}_{0,T} = R^{r}_{0,T} = \lambda R^{\varepsilon}_{0,T-1} + y_T y'_T$

$\gamma^{c}_{-1,T} = 1$

för p = 0 till min[N,T] - 1 ,dvs för varje ordning, uppdatera:

$\Delta_{p+1,T} = \lambda \Delta_{p+1,T-1} + \varepsilon_{p,T} r'_{p,T-1} / \gamma^{c}_{p-1,T-1}$

$\gamma^{c}_{p,T} = \gamma^{c}_{p-1,T} - r'_{p,T-1} \cdot R^{-r}_{p,T} \cdot r_{p,T}$

$K^{r}_{p+1,T} = \Delta_{p+1,T} \cdot R^{-r}_{p,T-1}$

$\varepsilon_{p+1,T} = \varepsilon_{p,T} - K^{r}_{p+1,T} r_{p,T-1}$

$$R^{\varepsilon}_{p+1,T} = R^{\varepsilon}_{p,T} - K^{r}_{p+1,T}\,\Delta'_{p+1,T}$$

$$K^{\varepsilon}_{p+1,T} = R^{-\varepsilon}_{p,T}\,\Delta_{p+1,T}$$

$$r_{p+1,T} = r_{p,T-1} - K^{\varepsilon}_{p+1,T}\,\varepsilon_{p,T}$$

$$R^{r}_{p+1,T} = R^{r}_{p,T-1} - \Delta'_{p+1,T}\,K^{\varepsilon}_{p+1,T}$$

Att notera är att endast Δ, R^r, R^ε, γ^c, r behöver sparas mellan varje tidsuppdatering. De sätts alla till 0 initialt (vid tiden t = 0). R^r,γ^c, r behöver sparas dubbelt, för att förhindra "överskrivning".

Likelihood-variabeln:

Variabeln $\gamma_{N,T}$ som uppträder i algoritmen, har en viktig tolkning som en approximativ log-likelihood variabel till processen y_T. Man kan visa att en exakt log.likelihood funktion enkelt kan beräknas ur lattice-parametrar [3]. Variabeln $\gamma_{N,T}$ kommer att vara liten så länge indata, [y_{T-N}, . . .,y_T] är Gaussiskt fördelat. Om data kommer från någon annan fördelning, kommer $\gamma_{N,T}$ att gå mot ett. Detta medför att förstärkninsfaktorn $1/(1 - \gamma_{N,T})$ som uppträder i tidsuppdateringen kommer att bli stor, vilket får lattice-parametrarna Δ, R^r, R^ε att ändra sig snabbt. Med andra ord fungerar $1/(1 - \gamma_{N,T})$ som en adaptiv förstärkning, som medger snabb spårning av ändringar i indatas fördelning [3].

A7.2 Beräkning av AR-parametrar ur Reflektionskoefficienterna:

Initiering: $B^{*}_{p,-1} = 0$ för $p = 0, \ldots, N-1$

$C_{0,0} = 0$

för varje $i = 0, \ldots, N$

$A_{o,i} = B_{o,i} = 1 \qquad i = 0$

$A_{o,i} = B_{o,i} = 0 \qquad i > 0$

för varje $p = 0, \ldots, N-1$

$$B^{*}_{p,i} = B_{p,i} - r_p \, C_{p,i} / \gamma^{c}_{p-1}$$

$$C_{p+1,i} = C_{p,i} - r'_p \, R^{-r}_p \, B_{p,i}$$

$$A_{p+1,i} = A_{p,i} = - K^{r}_{p+1} \, B^{*}_{p,i-1}$$

$$B_{p+1,i} = B^{*}_{p,i-1} - K^{\varepsilon}_{p+1} \, A_{p,i}$$

Ur denna rekursion får vi alltså de sökta AR-parmetrarna av ordning p och som biprodukt även alla lägre ordningar ur reflektions-koefficienterna. Detta var den främsta anledningen (tillsammans med numerisk stabilitet) till att vi valde att använda oss av en "lattice"-algoritm. AR-parametrarna uppdateras enbart vid vissa tidpunkter, i vårt fall var 40:e sampel.

A5 C-PROGRAM

Uppbyggnaden av hela C-programmet finns i kapitel 8.2. Nedan listas de ingående delarnas C-kod. Varje underprogram finns presenterat var för sig med inledande kommentarer om funktionen samt kortare kommentarer inne i koden. Som nämnts tidigare är allt implementerat i VENIX, men själva C-koden är standardkod utan några speciella realtidsfunktioner.

```
/*****************************************************************
*                                                                *
*       ****************                                         *
*       *   Arupdate   *                                         *
*       ****************                                         *
*                                                                *
*       Computing AR-coefficients from the lattice parameters    *
*                                                                *
*       Written:                1991-03-28                       *
*       Authors:                Per Abrahamsson                  *
*                               Peter Hallgren                   *
*                                                                *
*****************************************************************/

#include"drratio.h"

extern struct lattbuff lt;

arupdate(M,ke,kr,a_vekt)

int *M;
float ke[], kr[], a_vekt[];

{
int     p, i;
float   bs[MAXORD][MAXORD], b[MAXORD][MAXORD];
float   a[MAXORD][MAXORD], c[MAXORD][MAXORD];

    for( i=0; i<=*M; i++ ) {
        c[0][i] = 0;
        if( i==0 ){
            a[0][i] = 1;
            b[0][i] = 1;
        }
        else{
            a[0][i] = 0;
            b[0][i] = 0;
        }
        for( p=0; p<*M; p++ ){
            if( p==0 )
                bs[p][i] = b[p][i];
            else
                bs[p][i] = b[p][i]-lt.beta[p]*c[p][i]/lt.gam[p-1];
            c[p+1][i] = c[p][i]-lt.beta[p]*b[p][i]/lt.sigma[p];
            if( i==0 ){
                a[p+1][i] = a[p][i];
                b[p+1][i] = -ke[p+1]*a[p][i];
            }
            else{
                a[p+1][i] = a[p][i] -kr[p+1]*bs[p][i-1];
                b[p+1][i] = bs[p][i-1]-ke[p+1]*a[p][i];
            }
        }
    }
    for(i=0; i<=*M; i++)
        a_vekt[i]=a[*M][i];
}
```

```
/*************************************************************
*                                                            *
*                                                            *
*          ****************                                  *
*          *   complex.c  *                                  *
*          ****************                                  *
*                                                            *
*                                                            *
*          Functions for complex calculation                 *
*                                                            *
*                                                            *
*                                                            *
*          Written:  1991-03-28                              *
*                                                            *
*          Authors:  Per Abrahamsson                         *
*                    Peter Hallgren                          *
*                                                            *
*************************************************************/

 #include "complex.h"

/*
 *   Complex multiplication, one complex and one float
 */

complex cfmul(c, f)

complex c;
float f;

{
     c.r *= f;
     c.i *= f;
     return (c);
}



/*
 *  Complex addition
 */

complex cadd(a, b)

complex a, b;
{
     a.r += b.r;
     a.i += b.i;
     return (a);
}



/*
 *  Complex subtraction
 */
```

```
complex csub(a, b)

complex a, b;

{
     complex temp;
     temp = a;
     temp.r -= b.r;
     temp.i -= b.i;
     return (temp);
}



/*
 *  Complex multiplication, two complex
 */

complex cmul(a, b)

complex a, b;

{
     complex temp;
     temp.r = a.r * b.r - a.i * b.i;
     temp.i = a.r * b.i + a.i * b.r;
     return (temp);
}



/*
 *  Complex division, two complex
 */

complex cdiv(a, b)

complex a, b;

{
     complex t1, t2;
     float f;

     t1 = cmplx(b.r, -b.i);
     t2 = cmul(a, t1);
     f = (b.r * b.r) + ( b.i * b.i );
     t2.r /= f;
     t2.i /= f;
     return (t2);
}



/*
 *  Absolute
 */
```

```
float cabs(a)

complex a;

{
     float temp;

     temp = a.r * a.r;
     temp += a.i * a.i;
     return (sqrt(temp));
}



/*
 *  Real part of complex number
 */

float creal(a)

complex a;

{
     return (a.r);
}



/*
 *  Imaginary part of complex number
 */

float cimag(a)

complex a;

{
     return (a.i);
}



/*
 *  Making complex number
 */

complex cmplx(r, i)

float r, i;

{
     complex temp;

     temp.r = r;
     temp.i = i;
     return (temp);
}
```

```
#define MAXORD 15

struct lattbuff{
      float ra[MAXORD], delta[MAXORD], sigma[MAXORD];
      float beta[MAXORD], gam[MAXORD], lambda, sum, med, vekt[20];
      int init, t, temp;
};

struct firbuff {
      float b[31];
      float w[31];
};

struct rlsbuff {
      float delay[20];
      float p, th, hyst, lam, r2, eps, mu;
      int intjump, high, low, lowclear, extend;
} ;
```

```
/*************************************************************
*          *************                                     *
*             drratio1                                       *
*          *************                                     *
*                                                            *
*     This is the main function. It decimates the input data *
*     three times and predict the offset and subtract it.    *
*     System identification thrue a lattice-algoritm gives the *
*     AR-parameters which are used for decay ratio calculation. *
*                                                            *
*     Start commando:                                        *
*               drratio1  arg1  arg2  arg3  arg4  arg5       *
*                                                            *
*     Input:                                                 *
*               arg1:     inputdata                        *
*               arg2:     model order                      *
*               arg3:     sample frequency                 *
*     Output:                                                *
*               arg4:     decay ratio                      *
*               arg5:     frequency                        *
*                                                            *
*     Written:  1991-03-28                                   *
*     Authors:  Per Abrahamsson                              *
*               Peter Hallgren                               *
*                                                            *
*************************************************************/


# include<stdio.h>
# include<math.h>
# include"drratio.h"

float farr[16300], drarr[1000], frarr[1000];

struct rlsbuff sc;
struct firbuff fir;
struct lattbuff lt;

float ke_par[MAXORD], kr_par[MAXORD], a_par[MAXORD];

main(argc,argv)

int argc;
char *argv[];

{
     FILE *fp, *fopen();
     int x, l, t,jump, w, order;
     int  every, antjump, p, pp;
     float sm, data, initarr[30];
     float  dr_ratio, frq, sfrq;
     float firfilt();
```

```
if ((fp = fopen(argv[1], "r")) == NULL) {
     printf( "Can't open file for reading \n" );
     exit(1);
}
l = 0;
while ( fscanf(fp,"%f\n",&farr[l] ) != EOF ) {
     l++;
}
printf( "Number of elements in file: %d\n",l );

sscanf(argv[2], "%d", &order);
sscanf(argv[3], "%f", &sfrq);

sfrq = sfrq/3;
every = 10;                    /*   Variable which decide the
                                    interval between
                                    decay ratio updating */

/*
 *   Initializing of the antialiasing filter
 *   FIR-filter of order 30
 */

firinit1();
for ( t = 0; t < 30; t++){
     data = farr[t];
     data = firfilt(data);
}
w = 0;
x = 2;

/*
 *   Initializing the offset identification
 */


for (t = 30; t < 90; t++ ) {
     x++;
     data = farr[t];
     data = firfilt(data);
     if ( x == 3 ) {
          initarr[w] = data;
          x = 0;
          w++;
     }
}
rlsinit(initarr);

/*
 *   Delaying the decimated and detrended data 20 samples
 */

for ( t = 90; t < 150; t++ ) {
     x++;
```

```
        data = farr[t];
        data = firfilt(data);
        if ( x == 3 ) {
             rls( &data, &sm, &jump );
             x = 0;
        }
   }
   pp = 0;
   antjump = 0;

   /*
    *   Main loop begins
    */

   for ( t = 150; t < l; t++ ) {
        x++;
        data = farr[t];
        data = firfilt(data);
        if ( x == 3 ) {                          /* every third filtered
                                                    value */
             rls( &data, &sm, &jump );
             x = 0;
             if ( t == 150 )
                  lattinit(sm,order);
             else {
                  if ( jump == 0 ) {
                       lattice( &sm, &jump, ke_par, kr_par, &order
                            );
                       if ( t >= 450 ) {

                            /*
                             *   Choosen interval (every) for
                             *   parameter updating and
                             *   calculation of decay ratio.
                             */
                            p++;
                            if ( p == every ) {
                                 arupdate(&order, ke_par, kr_par,
                                      a_par);
                                 drupdate( order, a_par, sfrq,
                                      &dr_ratio, &frq );
                                 printf("%c[2J%c[H",27,27);
                                 printf("\n\n\n");
                                 printf("  TIME:     %d    (%d)\n",
                                      t, l);
                                 printf("\n\n\n\n");
                                 printf("       DECAY RATIO
                                      %4.2f \n",dr_ratio);
                                 printf("\n\n\n");
                                 printf("       FREQUENCY
                                      %4.2f \n",frq);
                                 drarr[pp] = dr_ratio;
                                 frarr[pp] = frq;
                                 pp++;
```

```
                                        p = 0;
                                    }
                                }
                            }
                            else{
                                antjump++;
                            }
                        }
                    }
                }
                printf("%c[2J%c[H",27,27);
                printf("\n\n\n\n\n\n");
                printf("      READY\n\n");
                printf("      Detected change in offset: %d   samples \n\n\n\n",
                        antjump);

                if (( fp = fopen(argv[4],"w" )) == NULL )
                    printf( "Can't open output file (arg 4) \n");
                for ( t = 0; t < pp; t++)
                    fprintf( fp, "%f\n", drarr[t] );

                if (( fp = fopen(argv[5],"w" )) == NULL )
                    printf( "Can't open output file (arg 5) \n" );
                for ( t = 0; t < pp; t++)
                    fprintf( fp, "%f\n", frarr[t] );

}
```

```
/*****************************************************************
*                                                                *
*                                                                *
*                                                                *
*         *****************                                      *
*         *    drratio2   *                                      *
*         *****************                                      *
*                                                                *
*    This is the main function. It decimates the input data      *
*    three times and predict the offset and subtract it.         *
*    Systm identification thrue a lattice-algoritm gives the     *
*    AR-parameters which are used for decay ratio calculation    *
*                                                                *
*    Start commando:                                             *
*              drratio2  arg1  arg2  arg3  arg4  arg5            *
*                                                                *
*    Input                                                       *
*              arg1:     inputdata                               *
*              arg2:     model order                             *
*              arg3:     sample frequency                        *
*    Output                                                      *
*              arg4:     decay ratio                             *
*              arg5:     frequency                               *
*                                                                *
*    Written:  1991-03-28                                        *
*    Authors:  Per Abrahamsson                                   *
*                        Peter Hallgren                          *
*                                                                *
*****************************************************************/


# include<stdio.h>
# include<math.h>
# include"drratio.h"

float farr[16300], drarr[1000], frarr[1000];

struct rlsbuff sc;
struct firbuff fir;
struct lattbuff lt;

float ke_par[MAXORD], kr_par[MAXORD], a_par[MAXORD];

main(argc,argv)

int argc;
char *argv[];

{
     FILE *fp, *fopen();
     int x, l, t, jump, w, order;
     int every, antjump, p, pp;
     float sm, data, initarr[30];
     float  dr_ratio, frq, sfrq;
```

```
float firfilt();


if ((fp = fopen(argv[1], "r")) == NULL) {
     printf( "Kan inte îppna filen fîr lÑsning \n" );
     exit(1);
}
l = 0;
while ( fscanf(fp,"%f\n",&farr[l] ) != EOF ) {
     l++;
}
printf( "Number of elements in file: %d\n",l );

sscanf(argv[2], "%d", &order);
sscanf(argv[3], "%f", &sfrq);

sfrq = sfrq/2;
every = 10;                              /*   Variable which decide
                                              the interval between
                                              decay ratio updating */

/*
 *   Initializing of the antialiasing filter
 *   FIR-filter of order 30
 */
firinit2();

for ( t = 0; t < 30; t++){
     data = farr[t];
     data = firfilt(data);
}
w = 0;
x = 1;


/*
 *   Initializing the offset identification
 */

for (t = 30; t < 70; t++ ) {
     x++;
     data = farr[t];
     data = firfilt(data);
     if ( x == 2 ) {
          initarr[w] = data;
          x = 0;
          w++;
     }
}
rlsinit(initarr);

/*
 *   Delaying the decimated and detrended data 20 samples
 */
```

```
for ( t = 70; t < 110; t++ ) {
     x++;
     data = farr[t];
     data = firfilt(data);
     if ( x == 2 ) {
          rls( &data, &sm, &jump );
          x = 0;
     }
}
pp = 0;
antjump = 0;

/*
 *   Main loop begins
 */

for ( t = 110; t < l; t++ ) {
     x++;
     data = farr[t];
     data = firfilt(data);
     if ( x == 2 ) {                    /* every second filtered
                                            value */
          rls( &data, &sm, &jump );
          x = 0;
          if ( t >= 130){
               if ( t == 130 )
                    lattinit(sm,order);
               else {
                    if ( jump == 0 ) {
                         lattice( &sm, &jump, ke_par, kr_par,
                                  &order );
                         if ( t >= 310 ) {

                              /*
                               *   Choosen interval (every) for
                               *   parameter updating and
                               *   calculation of decay ratio.
                               */
                              p++;
                              if ( p == every ) {
                                   arupdate(&order, ke_par,
                                            kr_par, a_par);
                                   drupdate( order, a_par, sfrq,
                                            &dr_ratio, &frq );
                                   printf("%c[2J%c[H",27,27);
                                   printf("\n\n\n");
                                   printf("  TIME:      %d
                                            (%d)\n", t, l);
                                   printf("\n\n\n\n");
                                   printf("        DECAY RATIO
                                        %4.2f \n",dr_ratio);
                                   printf("\n\n\n");
```

```
                                        printf("        FREQUENCY
                                                %4.2f \n",frq);
                                        drarr[pp] = dr_ratio;
                                        frarr[pp] = frq;
                                        pp++;
                                        p = 0;
                                    }
                                }
                            }
                            else {
                                antjump++;
                            }
                        }
                    }
                }
            }
            printf("%c[2J%c[H",27,27);
            printf("\n\n\n\n\n\n");
            printf("      READY\n\n");
            printf("      Detected change in offset: %d   samples \n\n\n\n",
                    antjump);

            if (( fp = fopen(argv[4],"w" )) == NULL )
                printf( "Can't open output file (arg 4) \n");
            for ( t = 0; t < pp; t++)
                fprintf( fp, "%f\n", drarr[t] );

            if (( fp = fopen(argv[5],"w" )) == NULL )
                printf( "Can't open output file (arg 5) \n" );
            for ( t = 0; t < pp; t++)
                fprintf( fp, "%f\n", frarr[t] );

}
```

```
/*************************************************************
*                                                            *
*                                                            *
*     ****************                                       *
*     *    drratio3  *                                       *
*     ****************                                       *
*                                                            *
*     This is the main function. It decimates the input data *
*     three times and predict the offset and subtract it.    *
*     System identification thrue a lattice-algoritm gives the *
*     AR-parameters which are used for decay ratio calculation. *
*     A function NEWLAMBDA is used to adjused the lattice lambda *
*     in case of decreasing decay ratio.                     *
*                                                            *
*     Start commando:                                        *
*               drratio3  arg1  arg2  arg3  arg4  arg5       *
*                                                            *
*     Input                                                  *
*               arg1:     inputdata                        *
*               arg2:     model order                      *
*               arg3:     sample frequency                 *
*     Output                                                 *
*               arg4:     decay ratio                      *
*               arg5:     frequency                        *
*                                                            *
*     Written:  1991-03-28                                   *
*     Authors:  Per Abrahamsson                              *
*               Peter Hallgren                               *
*                                                            *
*************************************************************/


# include<stdio.h>
# include<math.h>
# include"drratio.h"

float farr[16300], drarr[1000], frarr[1000];

struct rlsbuff sc;
struct    firbuff fir;
struct lattbuff lt;

float ke_par[MAXORD], kr_par[MAXORD], a_par[MAXORD];

main(argc,argv)

int argc;
char *argv[];

{
     FILE *fp, *fopen();
     int x, l, t,jump, w, order;
     int  every, antjump, p, pp;
     float sm, data, initarr[30];
```

```
float  dr_ratio, frq, sfrq;
float firfilt();


if ((fp = fopen(argv[1], "r")) == NULL) {
     printf( "Can't open file for reading \n" );
     exit(1);
}
l = 0;
while ( fscanf(fp,"%f\n",&farr[l] ) != EOF ) {
     l++;
}
printf( "Number of elements in file: %d\n",l );

sscanf(argv[2], "%d", &order);
sscanf(argv[3], "%f", &sfrq);

sfrq = sfrq/3;
every = 10;                    /*   Variable which decide the
                                    interval between
                                    decay ratio updating */

/*
 *   Initializing of the antialiasing filter
 *   FIR-filter of order 30
 */

firinit1();
for ( t = 0; t < 30; t++){
     data = farr[t];
     data = firfilt(data);
}
w = 0;
x = 2;

/*
 *   Initializing the offset identification
 */

for (t = 30; t < 90; t++ ) {
     x++;
     data = farr[t];
     data = firfilt(data);
     if ( x == 3 ) {
          initarr[w] = data;
          x = 0;
          w++;
     }
}
rlsinit(initarr);

/*
 *   Delaying the decimated and detrended data 20 samples
 */
```

```
for ( t = 90; t < 150; t++ ) {
     x++;
     data = farr[t];
     data = firfilt(data);
     if ( x == 3 ) {
          rls( &data, &sm, &jump );
          x = 0;
     }
}
pp = 0;
antjump = 0;

/*
 *   Main loop begins
 */

for ( t = 150; t < l; t++ ) {
     x++;
     data = farr[t];
     data = firfilt(data);
     if ( x == 3 ) {                    /* every third filtered
                                            value */
          rls( &data, &sm, &jump );
          x = 0;
          if ( t == 150 )
               lattinit(sm,order);
          else {
               if ( jump == 0 ) {
                    lattice( &sm, &jump, ke_par, kr_par, &order
                         );
                    if ( t >= 450 ) {

                         /*
                          *   Choosen interval (every) for
                          *   parameter updating and
                          *   calculation of decay ratio.
                          */
                         p++;
                         if ( p == every ) {
                              newlambda(every);
                              arupdate(&order, ke_par, kr_par,
                                        a_par);
                              drupdate( order, a_par, sfrq,
                                        &dr_ratio, &frq );
                              printf("%c[2J%c[H",27,27);
                              printf("\n\n\n");
                              printf("  TIME:      %d     (%d)\n",
                                        t, l);
                              printf("\n\n\n\n");
                              printf("        DECAY RATIO
                                        %4.2f \n",dr_ratio);
                              printf("\n\n\n");
```

```
                                        printf("        FREQUENCY
                                                %4.2f \n",frq);
                                        drarr[pp] = dr_ratio;
                                        frarr[pp] = frq;
                                        pp++;
                                        p = 0;
                                    }
                                }
                            }
                            else
                                antjump++;
                        }
                    }
                }
                printf("%c[2J%c[H",27,27);
                printf("\n\n\n\n\n\n");
                printf("      READY\n\n");
                printf("      Detected change in offset: %d   samples \n\n\n\n",
                        antjump);

                if (( fp = fopen(argv[4],"w" )) == NULL )
                    printf( "Can't open output file (arg 4) \n");
                for ( t = 0; t < pp; t++)
                    fprintf( fp, "%f\n", drarr[t] );

                if (( fp = fopen(argv[5],"w" )) == NULL )
                    printf( "Can't open output file (arg 5) \n" );
                for ( t = 0; t < pp; t++)
                    fprintf( fp, "%f\n", frarr[t] );

}
```

```
/*****************************************************************
*                                                                *
*                                                                *
*      *****************                                         *
*      *    drupdate   *                                         *
*      *****************                                         *
*                                                                *
*   Finds the zeros of the a-polynomial. Selects the dominating  *
*   pole pair and computes decay ratio and frequency             *
*                                                                *
*                                                                *
*     Written:       1991-03-28                                 *
*                                                                *
*     Authors:       PerAbrahamsson                             *
*                    Peter Hallgren                             *
*                                                                *
*                                                                *
*****************************************************************/

#include "lattice.h"
#include "complex.h"

drupdate(n,a,ws,ratio,freq)

float *ratio,*freq,ws,a[];
int n;
{

    int     i, plats, nr;
    complex r[MAXORD], temp[MAXORD];
    float   nyq, atemp[MAXORD];

    nyq = ws/2;      /*  Nyquist frequency  */

    /*
     *  Shifting the a-polynomial one step up in a_vector
     */

    for (i=1; i<=n+1; i++)
        atemp[i] = a[i-1];
    for ( i = 1; i <=n+1; i++)
        a[i]=atemp[i];

    /*
     *  Find the zeros of the a-polynomial
     */

    rpzero(n, a, r, 0);
    nr = 1;

    /*
     *  Find the complex conjugated solutions
     */

    for (i=1; i<=n+1; i++)
        if ((r[i].i > 0.001) && (r[i].r > 0.001)){
            temp[nr] = r[i];
```

```
            nr++;
        }

    /*
     *  Find the dominating poles
     */

    plats = 1;
    for (i=2; i<nr; i++)
        if (cabs(temp[i]) > cabs(temp[i-1]))
            plats = i;

    dr(temp[plats].r, temp[plats].i, nyq, ratio, freq);
}



/*
 *  Function which compute decay ratio and frequency out of
 *  the dominating pole pair
 */


dr(r, i, nyq, ratio, freq)

float r, i, nyq, *ratio, *freq;
{
    float temp, fi;

    temp = sqrt(r * r + i * i);
    if (r < 0)
        fi = M_PI - atan(i / fabs(r));
    else
        fi = atan(i / r);
    *ratio = exp(2 * M_PI * log(temp) / fi);
    *freq = nyq * fi / M_PI;
}
```

```
/*****************************************************************
*                                                                *
*       ***************                                          *
*       *   firfilt   *                                          *
*       ***************                                          *
*                                                                *
*       Filtering the input data thrue a 30:th order low-pass    *
*       FIR-filter to avoid aliasing when decimating             *
*                                                                *
*       Written:    1991-03-28                                   *
*       Authors:    Per Abrahamsson                              *
*                   Peter Hallgren                               *
*                                                                *
*                                                                *
*                                                                *
*****************************************************************/

#include"drratio.h"

extern struct firbuff fir;

float firfilt( in )

    float in;

{
    float y;
    int i;

        y =0;

        for ( i = 30; i > 0; i-- )
            fir.w[i] = fir.w[i-1];
        fir.w[0] = in;

        for ( i =0; i < 31; i++ )
            y = y + fir.b[i] * fir.w[i];

        return (y);
}
```

```
/*****************************************************************
*                                                                *
*       **************                                           *
*       *  FIRINIT1  *                                           *
*       **************                                           *
*                                                                *
*        Assignment of the FIR-filter coefficients               *
*                                                                *
*        Written:     1991-03-28                                 *
*        Authors:     Per Abrahamsson                            *
*                     Peter Hallgren                             *
*                                                                *
*                                                                *
*                                                                *
*****************************************************************/

#include"drratio.h"

extern struct firbuff fir;

firinit1()

{


    fir.b[0] = 0.0;
    fir.b[1] = 1.7693000e-003;
    fir.b[2] = 2.5341776e-003;
    fir.b[3] = 0.0;
    fir.b[4] = -5.8063744e-003;
    fir.b[5] = -8.5270213e-003;
    fir.b[6] = 0.0;
    fir.b[7] = 1.6913653e-002;
    fir.b[8] = 2.3108742e-002;
    fir.b[9] = 0.0;
    fir.b[10] = -4.2360041e-002;
    fir.b[11] = -5.8300075e-002;
    fir.b[12] = 0.0;
    fir.b[13] = 1.3206306e-001;
    fir.b[14] = 2.7230022e-001;
    fir.b[15] = 3.3260873e-001;
    fir.b[16] = 2.7230022e-001;
    fir.b[17] = 1.3206306e-001;
    fir.b[18] = 0.0;
    fir.b[19] = -5.8300075e-002;
    fir.b[20] = -4.2360041e-002;
    fir.b[21] = 0.0;
    fir.b[22] = 2.3108742e-002;
    fir.b[23] = 1.6913653e-002;
    fir.b[24] = 0.0;
    fir.b[25] = -8.5270213e-003;
    fir.b[26] = -5.8063744e-003;
    fir.b[27] = 0.0;
    fir.b[28] = 2.5341776e-003;
    fir.b[29] = 1.7693000e-003;
    fir.b[30] = 0.0;
}
```

```
/*****************************************************************
*                                                                *
*        ****************                                        *
*            lattice                                             *
*        ****************                                        *
*                                                                *
*                                                                *
*                                                                *
*         Lattice algoritm for system identification             *
*                                                                *
*                                                                *
*                                                                *
*         Written:            1991-03-28                         *
*                                                                *
*         Authors:            Per Abrahamsson                    *
*                             Peter Hallgren                     *
*                                                                *
*                                                                *
*                                                                *
*                                                                *
 ****************************************************************/

#include <stdio.h>
#include "drratio.h"

extern struct lattbuff lt;

/*
 *  ke, kr          forward/backward reflection coefficients
 *  eps, beta       forward/backward prediction error
 *  ra, sigma       forward/backward prediction error covariance
 *  delta           cross correlation coefficient
 *  gam             likelihood variable
 */

lattice( sign, jmp, ke, kr, M )

float *sign, ke[], kr[];
int *jmp, *M;

{
    float oldsigma[MAXORD], oldbeta[MAXORD];
    float oldgam[MAXORD], eps[MAXORD];

    float y, diff, oldmed;
    int p, index;

    y = *sign;
    for (p = 0; p < *M+1; p++) {
        oldsigma[p] = lt.sigma[p];
        oldbeta[p] = lt.beta[p];
        oldgam[p] = lt.gam[p];
    }
    eps[0] = y;
    lt.beta[0] = y;
    lt.ra[0] = lt.lambda * lt.ra[0] + y * y;
    lt.sigma[0] = lt.ra[0];
```

```
    if (lt.init == 1 ) {
        lt.t++;
        if ( lt.t == *M ){
            lt.init = 0;
            index = *M;
        }
        else
            index = lt.t;
    }
    for (p = 0; p <= index-1; p++) {
        if (p == 0) {
            lt.delta[p+1] =
            lt.lambda*lt.delta[p+1]+eps[p]*oldbeta[p];
            lt.gam[p] = 1.0-lt.beta[p]*lt.beta[p]/lt.sigma[p];
        }
        else {
              lt.delta[p+1]=
                lt.lambda*lt.delta[p+1]+eps[p]*oldbeta[p]/oldgam[p-1];
            lt.gam[p] = lt.gam[p-1]-lt.beta[p]*lt.beta[p]/lt.sigma[p];
        }
        kr[p+1] = lt.delta[p+1]/oldsigma[p];
        eps[p+1] = eps[p]-kr[p+1]*oldbeta[p];
        lt.ra[p+1] = lt.ra[p]-kr[p+1]*lt.delta[p+1];
        ke[p+1] = lt.delta[p+1]/lt.ra[p];
        lt.beta[p+1] = oldbeta[p]-ke[p+1]*eps[p];
        lt.sigma[p+1] = oldsigma[p]-lt.delta[p+1]*ke[p+1];
    }
    if( lt.temp<20 ){
        lt.sum = lt.sum+lt.delta[1];
        lt.vekt[lt.temp] = lt.delta[1];
        lt.temp = lt.temp+1;
    }
    else{
        lt.sum =lt.sum-lt.vekt[0]+lt.delta[1];
        for(p=0; p<19; p++)
            lt.vekt[p] = lt.vekt[p+1];
        lt.vekt[19] = lt.delta[1];
    }
}
```

```
/*****************************************************************
*                                                                *
*       ****************                                         *
*       *   lattinit   *                                         *
*       ****************                                         *
*                                                                *
*       Initializing the lattice-filter                          *
*                                                                *
*       Written:    1991-03-28                                   *
*       Authors:    Per Abrahamsson                              *
*                   Peter Hallgren                               *
*                                                                *
*                                                                *
*                                                                *
*****************************************************************/
#include "drratio.h"

extern  struct lattbuff lt;

lattinit(y, M)

float y;
int M;

{
    int p;

    for( p=0; p < M+1; p++){
        lt.beta[p] = 0;
        lt.sigma[p] = 0;
        lt.ra[p] = 0;
        lt.delta[p] = 0;
        lt.gam[p] = 0;
    }

    lt.beta[0] = y;
    lt.sigma[0] = y*y;
    lt.ra[0] = y*y;
    lt.init = 1;
    lt.t = 0;
    lt.lambda = 0.99875;
    lt.temp = 0;
    lt.sum = 0;
    lt.med = 0;
}
```

```
/*****************************************************************
*                                                                *
*                                                                *
*       ****************                                         *
*       *      rls       *                                       *
*       ****************                                         *
*                                                                *
*       Predict the inputdata offset and subtract it. When a     *
*       large and abrupt change in the offset occurs, a varible  *
*       jump goes high.                                          *
*                                                                *
*       Written:     1991-03-28                                  *
*                                                                *
*       Authors:     Per Abrahamsson                             *
*                    Peter Hallgren                              *
*                                                                *
*                                                                *
*                                                                *
*****************************************************************/

# include"drratio.h"
# include<math.h>

extern struct rlsbuff sc;

rls( indata, extsm, extjump )

float *indata, *extsm;
int *extjump;

{
    float gamma, newsm;
    int i, intjump, hold, lowlim;

    gamma = 3.5;
    lowlim = 15;
    hold = 30;
    sc.p = ( sc.p - sc.p * sc.p / ( sc.lam + sc.p )) / sc.lam;
    sc.eps = *indata - sc.th;
    sc.th = sc.th + sc.p * sc.eps;
    if ( fabs(sc.eps) < sc.hyst * gamma * sqrt(sc.r2) )
        sc.r2 = ( 1-sc.mu )*( 1-sc.p )*sc.eps*sc.eps+sc.mu*sc.r2;
    else
        sc.r2 = ( 1-sc.mu )*( 1-sc.p
                )*sqrt(fabs(sc.eps))+sc.mu*sc.r2;
    if ( fabs(sc.eps) < sc.hyst*gamma*sqrt(sc.r2) ) {
        sc.mu = 0.995;
        intjump = 0;
        sc.hyst = 1.0;
    }
    else {
        sc.mu = 0.9999;
        intjump = 1;
        sc.hyst = 0.2;
    }
    if ( intjump == 1 ) {
        sc.high++;
```

```
        sc.lowclear = 0;
        if ( sc.high >= 15 )
            sc.extend = 15+hold-lowlim;
    }
    else {
        if ( sc.high < 15 )
            sc.high = 0;
        if ( sc.extend > 0 )
            sc.low++;
        if (( sc.low > lowlim ) || ( sc.lowclear == 1)) {
            sc.extend--;
            sc.high = 0;
            sc.low = 0;
            sc.lowclear = 1;
        }
    }
    if (( sc.high >= 15 ) || ( sc.extend > 0 )) {
        *extjump = 1;
        sc.lam = 0.91;
    }
    else {
        *extjump = 0;
        sc.lam = 0.98;
    }
    *extsm = sc.delay[19];
    for ( i = 19; i > 0; i-- )
        sc.delay[i] = sc.delay[i-1];
    newsm = *indata -sc.th;
    sc.delay[0] = newsm;
}
```

```
/*****************************************************************
*                                                                *
*                                                                *
*       ****************                                         *
*       *    rlsinit    *                                        *
*       ****************                                         *
*                                                                *
*       Initializing the offset identification                   *
*                                                                *
*       Written:        1991-03-28                               *
*       Authors:        Per Abrahamsson                          *
*                       Peter Hallgren                           *
*                                                                *
*                                                                *
*                                                                *
*****************************************************************/

# include"drratio.h"

extern struct rlsbuff sc;

rlsinit( y )

float y[];

{
    int i;
    float p, th, eps;

    sc.p = 100.0;
    sc.th = y[0];
    sc.hyst = 1.0;
    sc.lam = 0.98;
    sc.mu = 0.995;
    sc.high = 0;
    sc.low = 0;
    sc.lowclear = 0;
    sc.extend = 0;

    p = 100.0;
    th = y[0];
    for ( i = 0; i < 20; i++ ) {
        sc.delay[i] = 0.0;
            p = (p - p * p / ( 0.98 + p ))/0.98;
            eps = y[i] - th;
            th = th + p * eps;
    }
    sc.r2 = 4*eps*eps;
}
```

```
/*****************************************************************
*                                                                *
*                                                                *
*       ****************                                         *
*       *   rpzero     *                                         *
*       ****************                                         *
*                                                                *
*       Finds the zeros of the a-polynomial.                     *
*                                                                *
*                                                                *
*       Written:        1991-03-28                               *
*       Authors:        Per Abrahamsson                          *
*                       Peter Hallgren                           *
*                                                                *
*                                                                *
*                                                                *
*****************************************************************/

#include "lattice.h"
#include "complex.h"

rpzero(n, a, r, iflg)

int n, iflg;
float a[];
complex r[];

{
    int i;
    complex t[MAXORD];

    for (i = 1; i <= n + 1; i++)
        t[i] = cmplx(a[i], 0.0);

    cpzero(n, t, r, iflg);
}



/*
 *  Finds the zeros of the complex polynomial
 */

cpzero(in, a, r, iflg)

int in, iflg;
complex a[], r[];

{
    complex temp[2], t[MAXORD], pn[MAXORD];
    float x, u, v, sune;
    int n, imax, n1, i, nr, nmax, nit, j;

    n = in;
    n1 = n + 1;

    /*
     *   If iflg=0 no initial estimates of zeros are input
```

```
     *  and some have to be found
     */

    if (iflg == 0) {
        temp[1] = cdiv(a[2], cfmul(a[1], (float)-n));
        cpevl(n, n, a, temp[1], t, t, 0);
        imax = n + 2;
        t[n1] = cmplx(cabs(t[n1]), 0.0);
        for(i = 2; i <= n1; i++) {
            t[n+i] = cmplx(-cabs(t[n+2-i]), 0.0);
            if (creal(t[n+i]) < creal(t[imax]))
                imax = n + i;
        }
        x = pow(-creal(t[imax]) / creal(t[n1]), 1.0 / (imax - n1));
        do {
            x *= 2.0;
            cpevl(n, 0, &t[n1-1], cmplx(x, 0.0), pn, pn, 0);
        } while (creal(pn[1]) < 0.0);
        u = 0.5 * x;
        v = x;
        do {
            x = 0.5 * (u + v);
            cpevl(n, 0, &t[n1-1], cmplx(x, 0.0), pn, pn, 0);
            if(creal(pn[1]) > 0)
                v = x;
            else
                u = x;
        } while ((v - u) > 0.001 * (1 + v));
        for(i = 1; i <= n; i++) {
            u = (3.14159265 / n) * (0.5 + 2.0 * (i - 1));
            r[i] = cadd((cfmul(cmplx(cos(u), sin(u)), MAX(x, (0.001
            * cabs(temp[1])))))), temp[1]);
        }
    }

    /*
     *  Main iteration loop starts here
     */

    nr = 0;
    nmax = 25 * n;
    nit = 1;
    while ((nit <= nmax) && (nr != n)) {
        for(i = 1; i <= n; i++) {
            if ((nit == 1) || (cabs(t[i]) != 0)) {
                cpevl(n, 0, a, r[i], pn, temp, 1);
                if((fabs(creal(pn[1])) + fabs(cimag(pn[1]))) <=
                (creal(temp[1]) + cimag(temp[1]))) {
                    t[i] = cmplx(0.0, 0.0);
                    nr += 1;
                }
                else {
                temp[1] = a[1];
                for (j = 1; j <= n; j++)
                    if (j != i)
                        temp[1] = cmul(temp[1], csub(r[i], r[j]));
                        t[i] = cdiv(pn[1], temp[1]);
                }
```

```
            }
        }
        for (i = 1; i <= n; i++)
            r[i] = csub(r[i], t[i]);
        nit++;
    }
    if (nr != n) {
        iflg = 2;           /* The program failed to converge */
    }
    else {
        iflg = 0;           /* All is well */
    }
}


#define FABS(c) (c < 0 ? -c : c)


complex za(c)

complex c;

{
    c.r = FABS(c.r);
    c.i = FABS(c.i);
    return (c);
}



/*
 *  Evaluate a complex polynomial and its derivates
 */

cpevl(n, m, a, z, c, b, kbd)

int n, m, kbd;
complex a[], z, c[], b[];

{
    complex ci, cim1, bi, bim1, t;
    float d1, r, s;
    int np1, i, j, mini;

    d1 = pow(2.0, -23.0);
    np1 = n + 1;
    for (j = 1; j <= np1; j++) {
        ci = cmplx(0.0, 0.0);
        cim1 = a[j];
        bi = cmplx(0.0, 0.0);
        bim1 = cmplx(0.0, 0.0);
        mini = MIN(m+1, n+2-j);
        for (i = 1; i <= mini; i++) {
            if (j != 1)
                ci = c[i];
            if (i != 1)
                cim1 = c[i-1];
```

```
                c[i] = cadd(cim1, cmul(z, ci));
                if (kbd != 0) {
                    if (j != 1)
                        bi = b[i];
                    if (i != 1)
                        bim1 = b[i-1];
                    t = cadd(bi, cfmul(za(ci), 3*d1+4*d1*d1));
                    r = creal(cmul(za(z), cmplx(t.r, -t.i)));
                    s = cimag(cmul(za(z), t));
                        b[i] = cadd(bim1, cadd(cfmul(za(cim1), d1),
                                cmplx(r, s)));
                        b[i] = cfmul(b[i], 1.0 + 8 * d1);
                    if (j == 1)
                        b[i] = cmplx(0.0, 0.0);
                }
            }
        }
    }
```

A6. MATLAB-FUNKTIONER.

Nedan följer en listning av de funktioner som är skrivna i MATLAB. Samtliga finns beskrivna i kapitel 8.1. Varje funktion finns presenterad var för sig med en kort inledning om funktionen samt kortare kommentarer inne i programmen. Som nämnts i kapitel 8 är MATLAB-funktionerna inte helt realtidsanpassade. Därför får man köra de olika funktionerna var för sig, med hela mätdataserier som insignal. De enskilda underprogramen arbetar dock helt i realtid, dvs arbetar on-line med ett signalvärde i taget.

```
function [a]= arupdate (N,kr,ke,beta,gamma,sigma)
%
%   [a] = arupdate(order,kr,ke,beta,gamma,sigma)
%
%     Indata:
%        order  =  order of a-polynomial to update
%        kr, ke =  reflection coefficients
%        beta   =  backward prediction error
%        gamma  =  likelihood variable
%        sigma  =  backward prediction variance
%
%     Output:
%        a      =  a-polynomial
%
%

    for p=2:N+1
        bs(p,1)=0;
    end
    for i=2:N+2
        if i==2
            a(2,i)=1;
            b(2,i)=1;
            c(2,i)=0;
        else
            a(2,i)=0;
            b(2,i)=0;
            c(2,i)=0;
        end

        for p=2:N+1
            bs(p,i)=b(p,i)-beta(p)*c(p,i)/gamma(p-1);
            c(p+1,i)=c(p,i)-beta(p)*b(p,i)/sigma(p);
            a(p+1,i)=a(p,i)-kr(p+1)*bs(p,i-1);
            b(p+1,i)=bs(p,i-1)-ke(p+1)*a(p,i);
        end
    end
end
```

```
function [dr,frq] = dr(rel,im,wn)
%
%    [dr,fr]=dr(rel, im, nyq)
%
%        Input:
%           rel = real part of dominating pole
%           im  = imaginary part of dominating pole
%
%        Output:
%           dr  = decay ratio
%           fr  = frequency
%
%
    r=sqrt(rel^2+im^2);
    if (rel < 0)
        fi=pi-atan(im/abs(rel));
    else
        fi=atan(im/rel);
    end
    dr=exp(2*pi*log(r)/fi);
    frq=wn*fi/pi;
end
```

```
function l = length(x)
%
%     LENGTH(X)
%      returns the length of vector X.
%
     l = max(size(x));
end
```

```
function [dra,frq]=drupdate(modell,nyq);
%
%    [dr,fr]=dmpreal(model,nyq)
%
%     Indata:
%         model = a-polynomial
%         nyq   = Nyquist frequency
%     Output:
%         dr    = decay ratio
%         fr    = frequency
%
%
    a=[1 modell];
    kv=[];
    temp=[];
    dra=[];
    frq=[];
    pol=roots(a);
    for x=1:size(pol)
        if imag(pol(x))>0
           temp=[temp;pol(x)];
        end
    end

   [belopp,plats]=max(abs(temp));
   re=real(temp(plats));
   im=imag(temp(plats));
   [dra,frq]=dr(re,im,nyq);
end
```

```
function [dmp,frq]=lattice1(signal,init,every,sfrq,low,high)
%
%      [dr,fr]=lattice1(signal,init,every,sampfrek,min,max)
%
%        Input:
%           signal =  input signal
%           init   =  number of sampels without uppdate
%           every  =  number of sampels between updating decay ratio
%           sfrq   =  sample frequency
%           low    =  lowest order number
%           high   =  highest order number
%
%        Output:
%           dr     =  decay ratio
%           fr     =  frequency
%
%
%

    clc
    lambda=0.99875;
    w=0;
    dmp=[];
    frq=[];
    L=length(signal);

%
%   Initializing lattice algoritm
%
    for t=1:high+2
        beta(t)=0;
        sigma(t)=0;
        ra(t)=0;
        delta(t)=0;
        gamma(t)=0;
    end

%
%   Main loop
%
    for t=1:L
        y=signal(t);
        epsilon(2)=y;
        beta(2)=y;
        ra(2)=lambda*ra(2)+y^2;
        sigma(2)=ra(2);
        gamma(1)=1;

        %
        % Main lattice iteration
        %
        for p=2:min([high t-1])+1
            if (Ggamma(p-1) < 1E-20)
                delta(p+1)=lambda*delta(p+1);
```

```
        else
            delta(p+1)=
            lambda*delta(p+1)+epsilon(p)*Gbeta(p)/Ggamma(p-1);
        end;
        if (sigma(p) < 1E-20)
            gamma(p)=gamma(p-1);
        else
            gamma(p)=gamma(p-1)-beta(p)^2/sigma(p);
        end;
        if (Gsigma(p) < 1E-20)
            kr(p+1)=0;
        else
            kr(p+1)=delta(p+1)/Gsigma(p);
        end;
        epsilon(p+1)=epsilon(p)-kr(p+1)*Gbeta(p);
        ra(p+1)=ra(p)-kr(p+1)*delta(p+1);
        if (ra(p) < 1E-20)
            ke(p+1)=0;
        else
            ke(p+1)=delta(p+1)/ra(p);
        end;
        beta(p+1)=Gbeta(p)-ke(p+1)*epsilon(p);
        sigma(p+1)=Gsigma(p)-delta(p+1)*ke(p+1);
    end

    %
    % Time shift
    %
    Gsigma=sigma;
    Ggamma=gamma;
    Gbeta=beta;

    if t<=init
        home
        disp('initializing:');
        disp(t);
        if t==init
            clc
            w=every-1;
        end
    else
        w=w+1;
        if w==every
            w=0;
            home
            disp('sampel :');
            disp(t);
            dp=[];
            fq=[];

            %
            % Updating decay ratio for orders low to high
```

```
                %
                %

                for z=low:high

                    %
                    % Updating the A-polynomial
                    %
                    [a]=arupdate(z,kr,ke,beta,gamma,sigma);
                    a=a(z+2,3:z+2);

                    %
                    % Updating and display of decay ratio
                    %
                    [dptemp,fqtemp]=drupdate(a,sfrq/2);
                    if size(dptemp)==0
                        dptemp=0;
                        fqtemp=0;
                    end
                    dp=[dp dptemp];
                    fq=[fq fqtemp];
                end
                disp('DECAY RATIO :');
                if size(dp)==0
                    disp('');
                else
                    disp(dp);
                end
                disp('FREQUENCY :');
                if size(fq)==0
                    disp('');
                else
                    disp(fq);
                end
                dmp=[dmp;dp];
                frq=[frq;fq];
            end
        end
    end
end
```

```
function [dmp,frq]=lattice2(signal,jump,init,every,sfrq,low,high)
%
%      [dr,fr]=lattice1(signal,jump,init,every,sampfrek,min,max)
%
%        Input:
%           signal =  detrended indata
%           jump   =  vektor which indicate if
%           init   =  number of sampels without uppdat
%           every  =  number of sampels between updating decay ratio
%           sfrq   =  sample frequency
%           low    =  lowest order number
%           high   =  highest order number
%
%        Output:
%           dr     =  decay ratio
%           fr     =  frequency
%
%
%

    clc
    lambda=0.99875;
    w=0;
    dmp=[];
    frq=[];
    L=length(signal);

%
%   Initializing lattice algoritm
%
    for t=1:high+2
        beta(t)=0;
        sigma(t)=0;
        ra(t)=0;
        delta(t)=0;
        gamma(t)=0;
    end

%
%   Main loop
%
    for t=1:L

        %
        % If offset jump in indata
        %
        if (jump(t) == 0)
            y=signal(t);
            epsilon(2)=y;
            beta(2)=y;
            ra(2)=lambda*ra(2)+y^2;
            sigma(2)=ra(2);
            gamma(1)=1;
```

```
%
% Main lattice iteration
%
for p=2:min([high t-1])+1
    if (Ggamma(p-1) < 1E-20)
        delta(p+1)=lambda*delta(p+1);
    else
        delta(p+1)=
        lambda*delta(p+1)+epsilon(p)*Gbeta(p)/Ggamma(p-1);
    end;
    if (sigma(p) < 1E-20)
        gamma(p)=gamma(p-1);
    else
        gamma(p)=gamma(p-1)-beta(p)^2/sigma(p);
    end;
    if (Gsigma(p) < 1E-20)
        kr(p+1)=0;
    else
        kr(p+1)=delta(p+1)/Gsigma(p);
    end;
    epsilon(p+1)=epsilon(p)-kr(p+1)*Gbeta(p);
    ra(p+1)=ra(p)-kr(p+1)*delta(p+1);
    if (ra(p) < 1E-20)
        ke(p+1)=0;
    else
        ke(p+1)=delta(p+1)/ra(p);
    end;
    beta(p+1)=Gbeta(p)-ke(p+1)*epsilon(p);
    sigma(p+1)=Gsigma(p)-delta(p+1)*ke(p+1);
end
%
% Time shift
%
Gsigma=sigma;
Ggamma=gamma;
Gbeta=beta;

if t<=init
    home
    disp('initializing:');
    disp(t);
    if t==init
        clc
        w=every-1;
    end
else
    w=w+1;
    if w==every
        w=0;
        home
        disp('sampel :');
        disp(t);
        dp=[];
```

```
                    fq=[];

                    %
                    % Updating decay ratio from order number
                    % low to high
                    %
                    for z=low:high

                        %
                        % Updating a-polynomial
                        %
                        [a]=arupdate(z,kr,ke,beta,gamma,sigma);
                        a=a(z+2,3:z+2);

                        %
                        % Updating decay ratio
                        %
                        [dptemp,fqtemp]=drupdate(a,sfrq/2);
                        if size(dptemp)==0
                            dptemp=0;
                            fqtemp=0;
                        end
                        dp=[dp dptemp];
                        fq=[fq fqtemp];
                    end
                    disp('DECAY RATIO :');
                    if size(dp)==0
                        disp('');
                    else
                        disp(dp);
                    end
                    disp('FREQUENCY :');
                    if size(fq)==0
                        disp('');
                    else
                        disp(fq);
                    end
                    dmp=[dmp;dp];
                    frq=[frq;fq];
                end
            end
        end
    end
end
```

```
function [extsm,extjump]=rls(y,lam,gamma,hold,lowlimit)
%
%       Recursive Least Square identification of offset
%
%
%
%            [esm,ejump] = RLS (signal,lambda,gamma,hold,low)
%
%
%    Input:
%        lambda = forgetting factor (0.98)
%        gamma  = Constant for detection of jump in offset (3.3 - 3.5)
%        hold   = Constant that extends the turnoff-signal (30)
%        low    = Minimum length of turnoff-signal to be
%                 accepted as output (15)
%
%    Output:
%        esm    = input signal with subtracted offset
%        ejump  = Turnoff signal for the function "lattice1"
%
%
    clc
    low = 0;
    lowclear = 0;
    th=y(1);
    m=length(y);
    mu=0.995;
    p=100;
    high =0;
    ths=[];
    newsignal = [];
    r2s=[];
    extjump = [];
    intjump = [];
    extsm = [];
    epsil =[];
    hysteres=1;
    xx=0;
    lambda=lam;
%
% ****  Initializing the prediction of R2  *****
%
    for t=1:20
        extsm = [extsm;0];
        p = (p-p*p/(lambda+p))/lambda;
        k = p;
        epsilon = y(t) - th;
        th = th + k * epsilon;
    end
    r2 =epsilon^2;
    p = 100;
    th = y(1);
%
```

```
% ****  RLS - identification of the offset ****
%
% ****  Main loop ****
%
    for t=1:m
     xx=xx+1;
     if xx==20
           home;
         disp('');
         disp('SAMPEL :');
         disp('');
           disp(t);
           xx=0;
        end
        p=(p-p*p/(lambda+p))/lambda;
        k=p;
        epsilon=(y(t)-th);
        th=th+k*epsilon;
        if abs(epsilon) < hysteres*gamma*sqrt(r2)
            r2 = (1-mu)*(1-p)*epsilon*epsilon+mu*r2;
        else
            r2 = (1-mu)*(1-p)*sqrt(abs(epsilon))+mu*r2;
        end
%
%       ****  Detection of abrupt change in offset (jump)****
%
        if abs(epsilon) < hysteres * gamma * sqrt(r2)
             nysig = y(t) - th;
             newsignal = [newsignal; nysig];
             intjump = [intjump; 0];
             hysteres = 1;
             mu=0.995;
        else
             nysig = y(t) - th;
             intjump = [intjump ;1];
             newsignal=[newsignal;nysig];
             hysteres = 0.2 ;
             mu=0.9999;
        end
%
%       **** Check if change in offset is significant ****
%
        if intjump(t) == 1
                high = high + 1;
                lowclear = 0;
                if high >= 10
                        extend = 10 + hold - lowlimit ;
          end
        else
                if high < 15
                        high = 0;
                end
                if extend > 0
```

```
                        low = low + 1;
                end
                if (low > lowlimit) | (lowclear == 1)
                        extend = extend - 1;
                        high = 0;
                        low = 0;
                        highflag = 0;
                        lowclear = 1;
                end
        end
%
%       **** Setting the turnoff-signal if significant change
%            in offset
%
        if (high >= 10) | (extend > 0)
             extjump = [extjump; 1];
             lambda = 0.91;
        else
             extjump = [extjump; 0];
             lambda = lam;
        end
%
%       **** Delaying the output signal ****
%
        if t > 20
                extsm = [extsm ;newsignal(t-20) ];
        end
    end
```