ISSN 0280–5316
ISRN LUTFD2/TFRT--5483--SE

Simulering av adaptiva regulatorer i Matlab

Anders Ringdahl

Institutionen för Reglerteknik
Lunds Tekniska Högskola
September 1993

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

Document name
Master Thesis

Date of issue
September 1993

Document Number
ISRN LUTFD2/TFRT--5483--SE

Author(s)
Anders Ringdahl

Supervisor
Bo Bernhardsson

Sponsoring organisation

Title and subtitle
Simulering av adaptiva regulatorer i Matlab
(Simulation of adaptive controllers using Matlab)

Abstract

The objective with this master thesis has been to evaluate the possibilities of using Matlab as a simulation tool for simulation of adaptive controllers. The extent of the evaluation has been restricted to the self-tuning-regulator.

As a result of the evaluation, a toolbox called Adaptive toolbox, has been created. The toolbox contains special designed functions describing the functional blocks of a self-tuning-regulator. As a complement to the functions, rules for the implementation and data structure in Matlab has been defined.

The result clearly indicates several advantages using Matlab compared with more traditional simulation tools.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280–5316

ISBN

Language
Swedish

Number of pages
75

Recipient's notes

Security classification

Innehållsförteckning

1. Inledning

Ett av problemen vid simulering av system innehållande en adaptiv regulator, är den mängd matrisberäkningar som oftast måste utföras. Vid användandet av traditionella simuleringsverktyg, exempelvis Simnon, leder detta snabbt till problem då dessa oftast saknar effektiva hjälpmedel för matrisberäkningar. Även enkla system tenderar att snabbt växa i komplexitet så ett av de större problemen för användaren ej längre är att förstå och analysera simuleringsresultaten, utan att skapa en kod som fungererar. Simuleringen blir alltså inte bara ett analysverktyg för ökad förståelse, utan själva programmeringsarbetet blir ett problem i sig själv. Detta är självfallet ej en önskvärd situation då det är rimligt att anta att både tid och kraft spenderas på aktiviteter som borde vara av underordnad betydelse vilket får till följd att själva arbetet med att analysera och skapa förståelse för det simulerade systemet blir lidande.

Det faller sig alltså naturligt att söka sig till beräkningsverktyg som innehåller funktioner för matrisberäkningar och möjlighet att utföra rekursiva beräkningar. Det krävs ej mycket fantasi och kreativ tankeförmåga för att inse att Matlab borde vara möjligt att använda. Matlab står för *matrix laboratory* och utvecklades ursprungligen för att användas som ett beräkningsverktyg för komplexa matrisberäkningar. Utöver funktioner för matrisberäkningar finns i Matlab även möjlighet att skapa beräkningsloopar samt tillgång till funktioner för grafisk presentation.

Målsättningen med arbetet har varit att:

* Undersöka möjligheterna att skapa en miljö i Matlab som möjliggör simulering av adaptiva regulatorer
* Skapa en metod för simulering av adaptiva regulatorer i Matlab
* Skapa en toolbox i Matlab innehållande funktioner direkt anpassade för simulering av adaptiva regulatorer

Utgångspunkt för arbetet har varit den självinställande regulatorn, *Self Tuning Regulator*, i forstättningen benämnd som STR. Genom att endast utnyttja denna typ har det varit möjligt att begränsa arbetet till en hanterlig omfattning. Samtidigt som man bibehåller möjligheten att avgöra om Matlab är en lämplig miljö för simulering av adaptiva regulatorer.

För att skapa en metod för simulering i Matlab, måste kraven analyseras utifrån vilka ingående delar som finns i ett adaptivt system, hur dessa skall beskrivas, hur de ingående delarna skall kopplas ihop till ett system och slutligen hur simuleringen skall utföras. Kapitel 2 beskriver hur program- och datastruktur har valts utifrån ovannämnda krav samt de möjligheter och begränsningar som finns i Matlab.

Metoden som valts kan kortfattat beskrivas så som att en fil skapas innehållande den Matlabkod som beskriver system och utför nödvändiga beräkningar. Denna fil benämns i forstättningen som simuleringsfil. Utseendet på denna fil bestämms utifrån den i kapitel 2 valda struktur.

Den toolbox som har skapats består av två delar

1: Funktioner

2: Simuleringsfiler för den indirekta STR:n respektive direkta STR:n

De funktioner som toolboxen innehåller finns kortfattat beskrivna i kapitel 3. I detta kapitel ges också regler för simuleringsfilens principiella uppbyggnad och datastruktur. Antalet nya funktioner som varit nödvändigt att skriva, har kunnat minimerats då strategin har varit att utnyttja så mycket av existerande Matlabfunktioner som möjligt.

Uppbyggnaden av simuleringsfilerna för den indirekta STR:n respektive direkta STR:n beskrivs i detalj i kapitel 4 respektive kapitel 5. För att exemplifiera och visa hur simulering utförs, finns en större simuleringsuppgift redovisad i dessa kapitel.

Avslutningsvis presenteras de slutsatser om Matlabs lämplighet som simuleringsmiljö för adapitva regulatorer i kapitel 6.

2. Program- och datastruktur

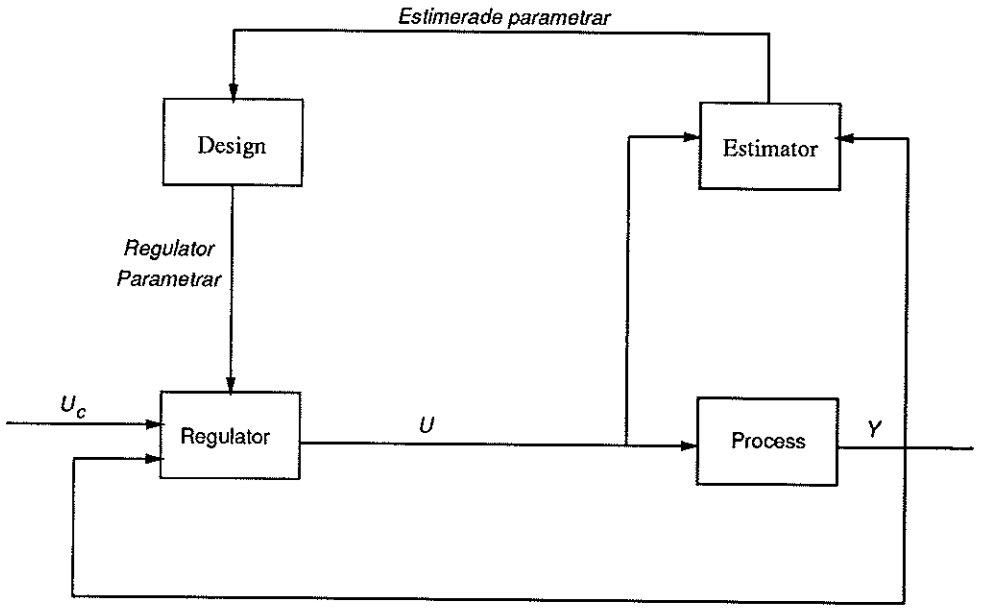
Målet med toolboxen är att skapa en miljö i Matlab för simulering av adaptiva regulatorer. Det här kapitlet diskuterar val av program- och datastruktur. Inför detta val har följande beaktats:

1. Vilka ingående delar finns i ett adaptivt system? Hur skall dessa beskrivas?
2. Hur skall de ingående delarna kopplas ihop till ett system?
3. Hur skall simuleringen av systemet utföras?

I avsnitt 2.1 diskuteras de ingående delarna, och hur dessa skall beskrivas. Beskrivningen av ingående delar och sammankopplingen av dessa till ett system, begränsas delvis av den datastruktur som finns tillgänglig. Val av datastruktur tas upp i avsnitt 2.2. I avsnitt 2.3 diskuteras uppbyggnad och simulering av systemet. Slutligen, i avsnitt 2.4, görs en jämförelse mellan Matlab och Simnon som simuleringsspråk.

2.1 Innehåll i toolboxen

Utgångspunkten för toolboxen har varit den självinställande regulatorn (*Self Tuning Regulator* , förkortas *STR*). Denna begränsning har gjorts för att begränsa arbetet, samtidigt som man bibehåller möjligheten att se hur generella funktioner kan skrivas. En självinställande regulator kan delas upp i följande block, se figur 2.1.



Figur 2.1 Blockschema för den självinställande regulatorn.

1. Processen.
2. Parameterskattare.
3. Regulatordesign.
4. Beräkning av styrsignalen.

Ovanstående punkter gäller för den indirekta regulatorn. Den direkta *STR:n* innehåller ingen regulatordesign, utan dessa parametrar skattas direkt.

Det är önskvärt att som användare ha tillgång till en rad olika varianter av ovanstående block. Man skall sedan på ett enkelt sätt kunna kombinera ihop dessa block till önskat system.

I Matlab finns möjligheten att skriva egna funktionerna, då Matlab kan läsa "kommando" från en extern fil. Denna fil kan vara av två typer, *script file* eller *function file* . Skillnaden mellan dessa är att *script filen* jobbar på det globala arbetsminnet, medan *funktions filen* arbetar på de parametrar som är argument till funktionen, se [1], kap 11.

För att skapa en miljö för simulering av system, krävs det mer än att bara kunna beskriva de ingående delarna. En användare måste också ha möjlighet att:

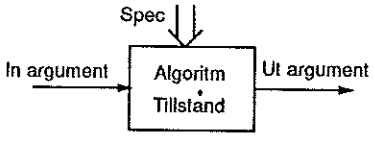
- Påverka systemet med en referenssignal.
- Påverka systemet med laststörningar.
- Påverka systemet med brus.
- Kunna att ange initialtillstånd.
- Filtrera signaler.
- Presentera resultat.
- Spara resultat på filer.

Hur dessa funktioner skall se ut i detalj, beror på vilken datastruktur som väljs och hur systemet kopplas ihop. Detta beskrivs i avsnitt 2.2 och 2.3 . Innehållet i funktionerna diskuteras mer allmänt i följande avsnitt.

Funktioner för beskrivning av ingående delar

Det är praktiskt att se de ingående blocken som funktioner. En sådan funktion kan då rent allmänt ses som i figur 2.2 . Funktionen har alltså ett inargument och ett utargument. Dessutom kan det vara nödvändigt att tilldela algoritmen vissa specifikationer. De flesta algoritmer behöver också någon form av tillstånd. Notera att in- och utargumenten, specifikationerna eller tillstånden kan vara av typ skalär, vektor eller matris, i Matlab.

Som nämnts tidigare finns möjligheten att skriva egna funktioner i Matlab, med valfritt antal in- och utargument. Funktionerna jobbar dock bara med de



Figur 2.2 Principutseende för en funktion

definierade in- och utargumenten, och kan ej hämta värden ifrån det globala arbetsminnet. Med undantaget om variablerna är definierade som *globala* . Att arbeta med denna typ av variabler är emellertid ej att rekommendera. Det är inte svårt att föreställa sig de problem som kan uppstå, om man av misstag använder en variabel som är global, se [1]. För funktionerna innebär det att de specifikationerna som algoritmen behöver, också skall ses som inargument. En närmare diskussion om hur funktionerna skall tilldelas tillstånden sker i avsnitt 2.2 .

Toolboxen bör göras kompatibel med andra, nu existerande hjälpmedel i Matlab. Ett intressant hjälpmedel är Lennart Ljungs *System Identification Toolbox.* Den har under vinter 1990-91 blivit utökad med rekursiva skattningsmetoder.

Ett viktigt krav är att funktionerna skall vara skrivna på ett så generellt sätt, att oavsett ordningen på systemet, antal skattade parametrar o.s.v skall samma funktion gå att använda. Vidare bör de vara skrivna på ett pedagogiskt sätt. Detta för att en användare skall kunna utgå ifrån de färdigskrivna funktionerna, och på så sätt vidareutveckla egna funktioner på ett enkelt sätt. Toolboxen blir då mer generell och lättarbetad.

Signal generering

En grundläggande förutsättning för att simulera ett dynamsikt system, är möjligheten att påverka systemet med referenssignaler, laststörningar och brus. Matlab erbjuder stora möjligheter att på ett enkelt sätt skapa vektorer, innehållande en godtycklig tidsserie. På detta sätt är det lätt för en användare att skapa önskade signaler. Dock kan det ändå vara önskvärt med färdigskrivna funktioner speciella signaler såsom fyrkantsvåg, vitt brus med en viss varians o.s.v . Hur dessa vektorer skall se ut, beror självklart på den datastruktur som väljs för toolboxen.

Presentera och spara resultat

I grundversionen av Matlab finns utmärkta möjligheter att visa resultat grafiskt. Men då toolboxen bara behandlar det tidsdiskreta fallet, kan det vara önskvärt att ha tillgång till plotrutiner som kan hantera sådana signaler. Att spara resultat på externa filer, är ej heller några problem. I Matlab har man möjligheten att spara allt från hela arbetsminnet, till enstaka variabler.

Sammanfattning

I detta avsnitt har diskuterats vad ett simuleringsverktyg måste innehålla. I nedanstående punkter sammanfattas vad Matlab-toolboxen måste innehålla.

- Funktioner för att beskriva ingående delar i systemet.
- Funktioner för att skapa signaler som påverkar systemet.
- Regler för hur ett system skall kopplas ihop och simuleras.
- Funktioner för att grafiskt presentera resultat.

2.2 Datastruktur

Den enda datastruktur Matlab erbjuder är typen matris. Dock kan storleken på matrisen väljas fritt, vilket ger möjlighet att också utnyttja typerna vektor och skalär. Denna datatyp passar utmärkt att lagra data från simulering av ett tidsdiskret system. Det naturliga sättet att spara en variabel som innehåller data från en sådan simulering, är att låta den vara av typen matris eller vektor.

$$\theta = \begin{pmatrix} \theta_1(1) & \theta_2(1) & \dots & \theta_k(1) \\ \vdots & \vdots & & \vdots \\ \theta_1(n) & \theta_2(n) & \dots & \theta_k(n) \end{pmatrix} \tag{2.1}$$

I Matlab kan man indexera enskilda elementen i vektorer och matriser, genom att ange radindex och kolonindex. Elementen $\theta(1:100, k)$ betyder i detta fall värdet av θ_k under de 100 första samplen.

Matlabs begränsade datastruktur leder dock till svårigheter vid rekursiva beräkningar. En rekursiv funktion med $u(k)$ som insignal och $x(k)$ som utsignal, kan skrivas som

$$\begin{aligned} x(k) &= f(u(k), state(k)) \\ state(k) &= [x(k-1), \dots, x(k-n), u(k-1), \dots, u(k-m)] \end{aligned} \tag{2.2}$$

Problemet är att Matlab ej tillåter negativa index. Detta kommer att leda till problem när man skall starta upp de rekursiva algoritmerna. Utan att utöka den tillgängliga datastrukturen, kan man tänka sig ett par olika lösningar.

1. Utöka vektorn eller matrisen så att de första element representerar initialskedet.
2. Inför speciella tillståndsvariabler som innehåller alla gamla värden. Dessa tilldelas initialvärden innan simuleringen börjar.

Utökade variabler

Variabler som representerar signaler,kommer med detta lösningsförslag att få följande form

$$X^T = \left(\underbrace{x(-n)\ x(-n+1)\ \dots\ x(-1)\ x(0)}_{neg\ tid}\ \ \underbrace{x(1)\ \dots\ x(k)}_{sim\ tid} \right) \tag{2.3}$$

Om man adderar till antalet element som representerar initalskedet, till det index som anger sampel. Har man på ett enkelt sätt utökat datastrukturen, så att uppdateringen av tillståndsvektorn kan ske enligt (2.2).

Men med denna struktur är det ej längre nödvändigt att ha en speciell tillståndsvariabel. Eftersom en variabeln av formen (2.3), innehåller alla gamla värden. Ett funktionsanrop för att utföra en rekursiv beräkning, kan då se ut som

$$X = f(X, U, par, k) \tag{2.4}$$

Där k är det index som anger sampel och *par* är en variabel som specificerar vilka element i vektorerna X och U som skall användas.

Men det är inte bara sampelindex som ger problem med denna lösning. För att detta index skall kunna användas av alla funktioner, måste alla vektorer och matriser vara av samma längd. Det vill säga att antalet element som anger initialskedet måste vara lika många.

EXEMPEL 2.1
Tag ett system som endast består av två filter med överföringsfunktionerna:

$$Y_1 = \frac{b_1 q^{-1}}{1 + a_1 q^{-1}} U_1$$

$$Y_2 = \frac{b_2 q^{-1}}{1 + a_2 q^{-1}} U_2$$

Y_1, Y_2, U_1 och U_2 skall alla ha ett utseende som

$$Y_1^T = \left(\underbrace{y_1(0)}_{neg\ tid} \quad \underbrace{y_1(1)\ y_1(2)\ldots}_{sim\ tid} \right)$$

Om man nu önskar att byta ut filter 2 mot

$$H_2(q^{-1}) = \frac{b_{21} q - 1 + b_{22} q^{-2}}{1 + a_{21} q^{-1} + a_{22} q^{-2}}$$

skall Y_2 och U_2 ha följande utseende.

$$Y_2^T = \left(\underbrace{y_2(-1)\ y_2(0)}_{neg\ tid} \quad \underbrace{y_2(1)\ y_2(2)\ldots}_{sim\ tid} \right)$$

För att samma index skall kunna användas för att beräkna utsignalen från filter 1 och filter 2, måste Y_1 och U_1 se ut som.

$$Y_1^T = \left(\underbrace{0\ y_1(0)}_{neg\ tid} \quad \underbrace{y_1(1)\ y_1(2)\ldots}_{sim\ tid} \right)$$

□

Det är lätt att inse att detta kan skapa problem då många signaler finns representerarde i ett system. Dock är det inte bara nackdelar med detta lösningsalternativet. Om det finns funktioner som automatiskt ordnar vektorernas längd, så ger denna lösning intressanta alternativ för att koppla ihop och simulera systemet. Mer om detta i avsnitt 2.3.

Tillståndvariabler

Det andra och enklaste alternativet är att införa en tillståndsvariabel, som innehåller alla gamla värden. Varje funktion måste ha en unik tillståndsvektor. En rekursiv funktion kan då ses som.

$$[x(k), state(k+1)] = f(u(k), state(k)) \tag{2.5}$$

där uppdateringen av statevektorn sker i funktionen. Detta är kanske det alternativ som förefaller naturligast att välja, men även här kommer man att få problem i initialskedet.

EXEMPEL 2.2
Betrakta en rekursiv skattningsalgoritm, där regressorn har följande utseende

$$\varphi^T(n) = [y(n-1)\ldots y(n-n_a)\ u(n-d)\ldots(u-d-n_b)]$$

För att uppdatera φ behövs alltså $y(n)$, och $u(n-d+1)$. Om $d > 1$ får man problem i initialskedet. För att uppfylla kravet att indexet skall vara större än ett, måste man alltså uppdatera med $y(n)$ och $u(n)$. I detta fallet kan man ju tänka sig att lägga till ett extra element i θ vektorn, vilket innebär att man måste skatta en extra parameter. En annan lösning är att införa en tillståndvektor som har följande innehåll.

$$state(n) = [y(n-1)\ldots\ u(n-1)\ldots]$$

Uppdateraringen av $state(n+1)$ sker nu med $y(n)$ och $u(n)$ och φ får bildas med element ur *state* vektorn. □

Generellt sätt innebär detta att tillståndsvariablerna måste innehålla alla gamla värden. Dessutom måste man, precis som i fallet med utökade variabler, kompletera inargumenten med en parameter som talar om vilka element som skall användas ur tillståndsvektorn, se (2.4).

En nackdel med att införa tillstånd, är att varje funktion måste ha en egen tillståndsvariabel definierad. Detta innebär att en viss signals initialvärden kan behöva definieras i flera tillståndsvariabler, om denna signal skall användas i olika funktioner. Detta problem undviker man om man väljer alternativet med utökade variabler, där initialvärdena på signalen ingår i den variabel som beskriver signalen.

Slutsatser

Den lösning som jag har kommit fram till fungerar bäst, är att införa tillståndvektorer. Funktionsanropen får då ett utseende som i (2.5), med en extra parameter som talar om vilka element ur statevektorn som används.

$$[x(k), state(k+1)] = f(u(k), state(k), par) \tag{2.6}$$

Men tillgång till en utökad datastruktur, som tillåter att binda initialvärden till en signal är önskvärd. Detta kan säkert lösas på många sätt, där den enklaste lösningen torde vara att utöka datastrukturen så att man tilläts att referera till element i matriser som hade negativa index. På detta sätt skulle man slippa alla problem vid uppdateringen av tillståndsvektorerna. Detta skulle medföra att algoritmerna skulle kunna skrivas på ett ännu mer pedagogiskt sätt.

2.3 Uppbyggnad och simulering av systemet

Hur de ingående delarna skall kopplas samman till ett system beror mycket på i vilken ordning beräkningarna måste utföras. Enligt specifikationerna får processen ej innehålla direktterm. Detta medför att återkopplingen i systemet ej leder till någon algebraisk loop. Men fortfarande är det ej godtyckligt i vilken ordning ekvationerna skall beräknas. Man måste sätta upp ett beräkningsschema eller en beräkningssekvens, där de ekvationer som skall lösas

utförs i en sådan ordning så att inga algebraiskaloopar uppstår. Detta beräkningsschema får sedan genomlöpas en gång per sampel. Frågan är alltså hur denna sekvens skall byggas upp och hur den skall se ut.

I [2] , kap 11, diskuteras praktiska aspekter och implementering av adaptiva regulatorer. Den beräkningssekvens som föreslås där, återfinns i lista 2.1. Detta schema är tänkt att användas då den adaptiva regulatorn skall verka på en riktig process.

1 `A/D omvandling av processens utsignal.`
2 `Beräkna styrsignalen.`
3 `D/A omvandling av styrsignalen.`
4 `Filtrering.`
5 `Skattning av parametrar.`
6 `Design av regulatorparametrar.`
7 `Förberedande beräkningar av styrsignalen.`

Lista 2.1 Beräkningsschema för en adaptiv regulator.

För att passa syftet för simulering måste schemat omarbetas. Steg 1 och 3 behöver självklart ej vara med, istället skall dessa steg ersättas med ett där processens utsignal och tillstånd beräknas. Likaså kan steg 2 och steg 7 slås ihop till ett steg. Att man här föreslagit en uppdelning, är för att minimera tiden från det man mäter processens utsignal till dess att styrsignalen läggs ut.

Då processen ej innehåller någon direktterm, behöver man inte $u(n)$ för att beräkna $y(n)$. Det innebär att beräkning av utsignalen och tillstånden kan ske som

$$\begin{aligned} x(n+1) &= \phi x(n) + \Gamma u(n) \\ y(n+1) &= Cx(n+1) \end{aligned} \tag{2.7}$$

Genom att uppdatera processen enligt (2.7), kan följande beräkningsschema användas vid simuleringen.

1 `Beräkna styrsignalen.`
2 `Filtrering.`
3 `Skattning av parametrar.`
4 `Design av regulatorparametrar.`
5 `Uppdatera processens utsignal och tillstånd.`

Lista 2.2 Beräkningsschema vid simulering av adaptiva regulatorer.

I avsnitt 2.1 nämns behovet av regler för hur systemet skall kopplas ihop och simuleras. Beräkningsschemat i 2.2 är liktydligt med de regler för hur simuleringen skall utföras. Hur man skall se till att detta schema följs återstår att lösa, liksom hur systemet skall kopplas ihop.

Under arbetet att utveckla toolboxen, har två möjliga sätt utkristalliserats hur signaler skall skickas mellan funktionerna.

1. Systemstrukturen ligger i funktionerna.
2. Systemstrukturen ligger i funktionsanropen.

Systemstrukturen i funktionerna

Då toolboxen utgår ifrån *STR:n* (se figur 2.1), finns en klar systemstruktur att följa. Det innebär att alla signalvägar i systemet är definierade från början. Funktionerna som skall beskriva de ingående blocken i systemet, kan alltså skrivas utifrån dessa signalvägar. Man behöver m.a.o. ej ha några signaler som in- och utargument till funktionerna. Det enda man behöver ange är de specifikationer som algoritmerna eventuellt behöver. Ett funktionsanrop får då ett principiellt utseende som i 2.8. Observera att utargumentet är valfritt.

$$utargument = funktion(spec) \tag{2.8}$$

Som nämnts i avsnit 2.2 kan funktioner endast jobba med de variabler som är funktionens argument, undantaget de variabler som är definierade som *global.* Alla variabler som skall beskriva signaler måste alltså vara *globala.* Den datastruktur man bör välja, är att organisera matriser och vektorer enligt (2.3). Genom att välja denna datastruktur slipper man att använda tillståndsvektorer, som då också hade varit tvungna att vara globala.

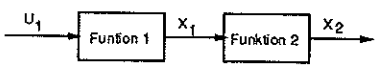
Genom att utnyttja den fasta systemstrukturen, möjliggör man intressanta lösningar för hur beräkningsschemat i lista 2.2 skall följas. Idén är att användaren endast behöver ange vilken av de färdigskrivna funktionerna, de olika blocken i systemet i fig 2.1 skall använda. Detta skall ske i en *definitionsrutin.* Därefter kan simulering av systemet utföras genom att använda en färdigskriven *simuleringsrutin.* Denna skall vara uppbygd enligt lista 2.2, och använder sig av de funktioner som definierats för de olika blocken. På detta sätt får man en toolbox som mer bygger på använda kommando.

Att skriva funktionerna för de olika blocken, är inga problem. Däremot blir det mycket svårt att skriva de rutinerna som skall definiera och simulera systemet. Framförallt då programeringsarbetet skall göras i Matlab. Man måste betänka att Matlab först och främst är ett matematikprogram, och ej något programspråk.

Efter mycket jobb med att försöka skriva ovannämnda rutiner, insåg jag att det inte var lösbart i Matlab. Idèn i sig själv är ändå intressant, helst med tanke på att man kan knyta funktioner till Matlab som är skrivna i programspråken C eller Fortran.

Systemstruktur i funktionsanropen

Detta är den lösning som är den enklast, och kanske mest logisk. Den kräver däremot mer arbete av en användare. Om två funktioner skall kopplas ihop som i fig 2.3, skall Matlabkoden för detta skrivas som i lista 2.3.



Figur 2.3 Sammankoppling av två funktionsblock.

```
x1=funktion1(u1)
x2=funktion2(x1)
```

Lista 2.3 Princip för Matlabkod, för att åstadkomma sammankopplingen i fig 2.3.

Denna metod innebär att användaren själv får skriva den beräkningsrutin, vilken skall följa lista 2.3. Man måste skapa en sekvens, innehållande de valda funktionerna för de olika blocken. Och placera dessa funktioner så som lista 2.3 föreskriver. Sekvensen byggs lätt upp med hjälp av *FOR* eller *WHILE* kommando, som finns att tillgå i Matlab.

Det är med hjälp av denna metoden jag valt att bygga upp toolboxen. Fördelen är att framförallt att man ej binder sig till någon systemstruktur. Man slipper nu också att arbeta med globala variabler. Datastrukturen jag valt att arbeta med är att införa tillståndsvektorer. Vilket medför att funktionsanropen ser ut som i (2.5).

2.4 Jämförelse Simnon/Matlab

En direkt jämförelse mellan Simnon och Matlab kan egentligen inte göras, då de utvecklats för olika syften. Det man kan göra är att försöka besvara frågan: kan Matlab användas som simuleringspråk? Svaret är ja, fast inte utan svårigheter.

Det är mycket man saknar i Matlab, jämfört med Simnon. Eftersom Simnon är ett simuleringsprogram för dynamiska system, finns där det mesta man kan tänkas behöva då det gäller att definiera och koppla ihop systemet. Dessutom finns där sorteringsrutiner som automatiskt skapar det beräkningsschema som skall genomlöpas. Det är dessa möjligheter som jag framförallt saknar i Matlab. Däremot har ju Matlab sin styrka i matrisberäkningarna. Detta gör Matlab överlägset Simnon, då det gäller att skriva de ekvationer som beskriver systemet. Dessutom kan man skriva egna funktioner i Matlab, vilket möjliggör generellt skrivna algoritmer.

Det är egentligen bara en smakfråga om man skall önska sig Simnon med matrisoperationer, eller Matlab med de kommando Simnon har för att binda ihop och simulera ett system. Fördelen med att utföra simuleringar i Matlab är att det finns en mängd hjälpmedel att tillgå, då det gäller att räkna på och analysera system. Under våren 1991 kommer ett nytt hjälpmedel kallat *SIMULAB* att introduceras. Detta är ett program för simulering av dynamiska system, där ingående funktionsblock bindes ihop grafiskt. Funktionsblocken kan skapas antingen i C, Fortran eller i Matlab. Tyvärr har jag ej haft möjligheten att testa detta program. Men det är tänkbart att detta är ett sätt att utöka Matlab med de möjligheter som finns i Simnon.

3. Beskrivning av *Adaptive-Toolbox*

Simulering av adaptiva regulatorer i Matlab, med hjälp av *Adaptive Toolbox*, kan enklast beskrivas så som att en simuleringsfil byggs upp innehållande ett beräkningsschema som utför de rekursiva beräkningarna. De i systemet ingående delarna, exemplevis regulator, estimator etc, beskrivs med hjälp av funktioner från toolboxen. Utseendet på denna simuleringsfilen bestämms utifrån definitioner och regler som gäller för datastruktur samt struktur för simuleringsfilen.

Adaptive Toolbox består av följande delar:

- Regler och definitioner för simuleringsfilen samt datastruktur
- Funktioner som beskriver ingående delar i systemet
- Implementerad algoritm för den inidirekta STR:n
- Implementerad algoritm för den direkta STR:n

Definitioner på datastruktur återfinns i avsnitt 3.1, medan de regler som gäller för simuleringsfilen finns beskrivna i avsnitt 3.2. Funktioner som ingår i *Adaptive Toolbox* finns kortfattat beskrivna i avsnitt 3.3, komplett Matlabkod finns i appendix A.

Algoritmerna för den direkta och indirekta STR:n återfinns i två stycken kompletta simuleringsfiler. Dessa filer är i detalj beskrivna i kapitel 4 och 5 och beskrivs ej närmare i detta kapitel.

3.1 Grundläggande definitioner

Begränsningar

Under utvecklingen av *Adapitve Toolbox* har ett antal begränsningar införts, både avseende processen och typen av regulator. Motivet till detta återfinns i kapitel 2. De begränsningar som införts är:

- Processen som skall simuleras får ej innehålla någon direktterm
- Processen som skall simuleras kan endast beskrivas i diskret tid
- Endast regulatorn av typen *STR* finns tillgänglig

Datastruktur

Som nämnts tidigare i kapitel 2, är den enda tillgängliga datatypen i Matlab matrisen. Möjlighet finns dock att fritt definiera antal rader och kolumner, vilket medför att även vektorer och skalärer är tillgängliga.

För att skapa en så användarvänlig miljö som möjligt, är det av största vikt att en entydig notationsprincip används. Speciellt viktigt är det att klargöra

vad antalet rader respektive antalet kolumner anger. Förvirring och onödiga fel kan lätt uppstå om en oklar notationsprincip tillämpas.

De parametrar som kan skapa problem är variabler som vilka innehåller resultat från simuleringen så som parameterskattningar och signaler.

Beskrivning av signaler

För beskrivning av signaler utnyttjas genomgående en radvektor enligt 3.1

$$Y = \begin{pmatrix} y(1) \\ y(2) \\ \vdots \\ y(n) \end{pmatrix} \tag{3.1}$$

Antalet rader i ovanstående vektor anger antal sampel som signalen genrerats.

Beskrivning av parameterskattningar

Om samma princip avänds för variabler som innehåller parameterskattningar under en simulering, kommer dessa att anta formen som i 3.2.

$$\theta = \begin{pmatrix} \theta_1(1) & \theta_2(1) & \dots & \theta_k(1) \\ \vdots & \vdots & & \vdots \\ \theta_1(n) & \theta_2(n) & \dots & \theta_k(n) \end{pmatrix} \tag{3.2}$$

Antalet rader i matrisen anger det antal sampel som har simulerats. Antal i koloner i matrisen anger antalet skattade parametrar.

3.2 Uppbyggnad av simuleringsfil

För att simulera system som innehåller en adaptiv regulator krävs att rekursiva beräkningar utförs. Enklaste sättet att åstadkomma detta i Matlab, är att skapa en *script-fil.* Denna fil, simuleringsfilen, skall innehålla en loop där de rekursiva beräkningarna utförs.

Det principiella utseendet för simuleringsfilen återfinns i lista 3.1

```
INITSIERING
BEGIN LOOP
   REKURSIV BERÄKNINGSALGORITM
END LOOP
PRESENTATION
```

Lista 3.1 Principiellt utseende för simuleringsfil.

Som framgår är filen uppdelad i tre huvuddelar:

- Initiering
- Beräkningar
- Pressentation

Genom denna uppdelning är det möjligt att skapa simuleringsfiler av mer generell karaktär, detta då den del som beskriver typ av regulator återfinns i beräkningsalgoritmen medan initieringsdelen innehåller bl a de delar som beskriver processen som skall simuleras. Detta innebär att med relativt enkla handgrepp kan en ny process simuleras, alternativt kan beräkningsalgoritmen bytas så att en ny typ av regulator kan simuelras.

Initiering

Initieringsfasen fyller ett flertal syften. Genom att göra en genomtänkt initiering ökar läsbarheten och återanvändningsgraden av simuleringsfilen. Ett antal funktioner kräver också initialvärden på vissa inparametrar. Exempelvis kräver funktionen `process`, vilken beräknar processens utsignal, initialtillståndet för processen. Variabler av denna typ måste alltså tilldelas initalvärden, innan beräkningsloopen genomlöps första gången. En annat syfte med initieringen är att försöka minimera de operationer som skall utföras i beräkningsloopen, vilket är ekvialent med att minimera exikveringstiden.

Initieringen kan delas upp i följande block:

1 Skapa processen som skall simuleras

2 Design av det slutna systemet

3 Signalgenerering

4 Initiering av estimering

5 Initiering av regulatorn

6 *Speedup*-rutin

Vad som utförs i respektive block, är beroende av vilka funktioner som används i beräkningsdelen och vilken typ av regulator som simuleras. Rent generellt kan dock sägas att tilldelning av värden till konstanter, så som antal parametrar som skall skattas osv, är exempel på operationer som skall göras under initieringen. Initieringsdelen för den indirekta och direkta STR:n beskrivs i detalj i kapitel 4 respektive 5.

Speedup-rutin

Genom att skapa variabler som innehåller resultat från simuleringen i initieringsfasen kan exikveringstiden drastiskt minskas. Detta beror på att Matlab jobbar så att när en ny rad eller kolumn skall läsas in, måste ny minnesarea allokeras. Den "nya" variabeln lagras därefter på det nya minnesutrymmet.

Då storleken på de matrisern och kolumner som innehåller simuleringsresultat är kända, kan dessa enkelt skapas redan i initieringsfasen. Enklast sker detta med funktionen `zero`, vilken fyller matrisen eller kolumnen med nollor.

Vid försök visade det sig att exikveringstiden kunde minskas med 5 - 10 gånger med denna metod, jämfört med att ej skapa dessa variabler i initieringen.

För att minimera exikveringstiden rekommenderas användaren alltså att skapa variabler som skall innehålla simuleringsresultat i initieringsfasen.

Allmänt om beräkningsalgoritmer

Med beräkningsalgoritm avsees här, det beräkningsschema som skall genomlöpas på ett sådant sätt att nödvändiga beräkningar utförs i en korrekt ordning. Under förutsättning att processen ej innehåller någon direktterm, kan följande grundschema för algoritmerna ställas upp:

```
1  Beräkning av styrsignalen
2  Filtrering
3  Skattning av processen
4  Design av regulatorparametrar
5  Beräkning av utsignal från processen och uppdatering av
   processens tillstånd
```

Lista 3.2 Beräkningsschema för simulering av adaptiva regulatorer under förutsättning att processen ej innehåller en direktterm.

En mer ingående diskussion om ovanstående beräkningsschema återfinns i kapitel 2. Beroende på vilken typ av regulator som simuleras kommer vissa smärre olikheter att uppträda. En genomgång av algoritmerna för den direkta och indirekta STR:n återfinns i kapitel 4 och 5.

Presentation av simuleringsresultatet

Den sista delen av simuleringsfilen är den del där simuleringsresultatet presenteras. I *Adaptive toolbox* finns två stycken funktioner, speciellt anpassade för presentation av vektorer och matriser beskrivna på formen enligt (3.1) och (3.2).

Utöver dessa funktioner, har man dessutom tillgång till de otaliga funktioner som finns tillgängliga i Matlab för analys och presentation av simuleringsresultat. I simuleringsfilerna för den direkta och indirekta STR:n har en plottrutin lagts in i beräkningsdelen. Här utnyttjas funktioner som plottar Bodediagrammet för verklig och skattad process samt Nygquistdiagrammet för verklig och skattad kretsöverföringsfunktion. Dessa operationer utförs i detta fall var 20:e sampel.

Genom att införa plotrutiner i beräkningsdelen, får man ett kraftfullt verktyg för att analysera och förstå systemet som simuleras. Nackdelen med en sådan rutin, är självklart att exikveringstiden ökar.

3.3 Funktioner

Under utvecklingen av toolboxen har strategin varit att utnyttja så mycket som möjligt av existerande Matlabfunktioner. På grund av detta har behovet av antalet nya funktioner kunnat minimeras.

Kompleterande funktioner till *Adaptive Toolbox* är framförallt hämtade från följande toolboxar:

- System Identification Toolbox
- PPBOX
- FRBOX

Vilka funktioner som har använts, framgår ur beskrivningen av simuleringsfilerna i kapitel 4 respektive 5. För mer information om dessa funktioner hänvisas till respektive manual, se [5] och [6].

I nedanstående avsnitt presenteras funktioner ingående i *Adaptive Toolbox*. Den fullständiga Matlabkoden återfinns i Appendix A.

RST1

Beskrivning

Beräknar styrsignalen *u(t)* utrifrån styrlagen:

$$R(q)u(k) = T(q)u_c(k) - S(q)y(k)$$

Det förutsätts här att R är monisk samt att:

$$degR \geq degT$$

$$degR \geq degS$$

Synops

```
[u,newstate]= rst1(r,s,t,y,uc,state)
```

`r,s,t`	:	Radvektorer med reglerpolynomen R,S och T $R = [1\ r_1\ \ldots\ r_{degR}]$ $S = [s_0\ s_1\ \ldots\ r_{degS}]$ $T = [t_o\ t_1\ \ldots\ r_{degT}]$
`y`	:	Processens utsignal $y(k)$
`uc`	:	Referenssignal $u_c(k)$
`state`	:	Tillståndsvektor innehållande gamla u, y och u_c $[u(k-1)\ u(k-2)\ \ldots\ y(k-1)\ \ldots\ u_c(k-1)\ \ldots\ ]$
`u`	:	Styrsignal $u(k)$
`newstate`	:	Uppdaterad tillståndsvektor $[u(k)\ u(k-1)\ \ldots\ y(k)\ \ldots\ u_c(k)\ \ldots\ ]$

Anmärkning

Om `state = []`, tom vektor, returneras `newstate` som en nollvektor med rätt längd.

RST2

Beskrivning

Implemetering av en RST regulator med *anitwindup*. Styrsignalen $u(k)$ beräknas utifrån styrlagen:

$$A_o(q)v(k) = Tu_c(k) - S(q)y(k) + (A_o(q) - R(q))u(k)$$
$$u(k) = sat\ v(k)$$

Det förutses att R och A_o är moniska, samt att följande är uppfyllt:

$$degR \geq degT$$

$$degR \geq degS$$

$$degA_o = degR$$

Synops

```
[u,newstate]= rst2(r,s,t,ao,lim,y,uc,state)
```

`r,s,t`	:	Radvektorer med reglerpolynomen R, S och T
`ao`	:	Radvektor innehållande oberverarpolynomet A_o
`lim`	:	Mättningsgränser $[u_{min}\ u_{max}]$
`y`	:	Processens utsignal $y(k)$
`uc`	:	Referenssignal $u_c(k)$
`state`	:	Tillståndsvektor med gamla u, y och u_c
`u`	:	Styrsignal $u(k)$
`newstate`	:	Uppdaterad tillståndsvektor

Anmärkning

Om `state = []`, tom vektor, returneras `newstate` som en nollvektor med rätt längd.

Då observerarpolynomet sätt till [] skapas en deadbeat observerare $A_o^* = 1$.

RLS

Beskrivning

Skattar A och B, med minstakvadratmetoden med glömskefaktor, i processen som beskrivs av:

$$A^*(q^{-1})y(k) = B^*(q^{-1})u(k - nk) + C^*(q^{-1})e(k)$$

A och B beskriver polynomen:

$$A^*(q^{-1}) = 1 + a_1q^{-1} + \ \ldots\ a_{na}q^{-na}$$

$$B^*(q^{-1}) = b_0 + b_1q^{-1} + \ \ldots\ b_{nb}q^{-nb}$$

Skattningen ges av:

$$\begin{aligned}
\hat{\theta}(k) &= \hat{\theta}(k-1) + K(k)\epsilon(k) \\
\epsilon(k) &= y(k) - \varphi^T(k-1)\hat{\theta}(k-1) \\
K(k) &= P(k-1)\varphi(k-1)(\lambda + \varphi^T(k-1)P(k-1)\varphi(k-1))^{-1} \\
P(k) &= (I - K(k)\varphi^T(k-1))P(k-1)/\lambda
\end{aligned}$$

där

$$\begin{aligned}
\hat{\theta}^T &= [a_1 \ \ldots \ a_{na} \ b_0 \ b_1 \ \ldots \ b_{nb}] \\
\varphi^T(k-1) &= [-y(k-1) \ \ldots \ -y(k-na) \ u(k-nk) \ \ldots \ u(k-nk-nb)]
\end{aligned}$$

Synops

```
[theta,p,nstate ]= rls(y,u,nn,lambda,thetao,po,state)
```

`y,u`	:	$y(k)$ respektive $u(k)$
`nn`	:	Vektor som definierar antalet skattade parametrar i A och B samt fördröjningen: `nn = [ na nb nk ]`
`lambda`	:	Glömskefaktor
`thetao`	:	Kolumnvektor som anger $\hat{\theta}(k-1)$
`po`	:	$P(k-1)$
`state`	:	Radvektor som anger tillståndsvektorn $[-y(k-1) \ \ldots -y(k-na) \ u(k-1) \ \ldots \ u(k-nk-nb)]$
`theta`	:	Radvektor som anger $\hat{\theta}(k)$
`p`	:	$P(k)$
`nstate`	:	Uppdaterad tillståndsvektor $[-y(k) \ \ldots -y(k-na+1) \ u(k) \ \ldots \ u(k-nk-nb+1)]$

Anmärkning

$\varphi(k-1)$ skapas med hjälp av tillståndsvektorn `state`. Om `state = []`, tom vektor, returneras `nstate` som en nollvektor med rätt antal element baserat på na, nb och nk.

Observera att den nya paramaterskattningen, `theta`, ges som en radvektor och ej som kolumnvektor. Detta beroende på den definition som gjorts, se ekvation 3.2, för en variabel innehållande parameterskattning under ett antal sampel.

FILT

Beskrivning

Filtrerar en signal med ett filter som beskrivs av:

$$\begin{aligned}
H_f(q) &= \frac{B(q)}{A(q)} \\
y(k) &= H_f(q)u(k)
\end{aligned}$$

Det förutsätts att A är monisk.

Synops

```
[y,newstate ] = filt(u,state,b,a)
```

`u`	:	Insignalen $u(k)$
`state`	:	Tillståndsvektor som ges av radvektorn: $[u(k-1) \ldots u(k-nb) \ -y(k-1) \ldots \ -y(k-na)]$
`a,b`	:	Radvektorer som beskriver polynomen A och B
`y`	:	Filtrerad utsignal $y(k)$
`newstate`	:	Uppdaterad tillståndsvektor

Anmärkning

Om `state = []`, tom vektor, returneras `newstate` som en nollvektor med rätt antal element.

PROCESS

Beskrivning

Beräknar utsignalen från en process, utan direktterm, beskriven på tillståndsform.

$$x(k+1) = \Phi x(k) + \Gamma u(k)$$
$$y(k+1) = Cx(k+1)$$

Synops

```
[y,x ] = process(x,y,procpar)
```

`x`	:	Tillståndsvektor $x(k)$
`u`	:	Styrsignal $u(k)$
`procpar`	:	Beskriver processen som `procpar = [`$\Phi \ \Gamma \ C^T$`]`
`y`	:	Utsignal från processen $y(k+1)$
`x`	:	Uppdaterad tillståndsvektor $x(k+1)$

SAT

Beskrivning

Beskriver funktionen:

$$sat\ u = \begin{cases} u_{low} & u \leq u_{low} \\ u & u_{low} < u < u_{high} \\ u_{high} & u \geq u_{high} \end{cases}$$

Synops

```
[usat] = sat(u,ulim)
```

`u`	:	Insignal $u(k)$
`ulim`	:	Mättnadsgränser: `ulim =` $[u_{min} \ u_{max}]$
`usat`	:	Begränsad utsignal $sat\ u$

SHIFT

Beskrivning

Funktionen utför högerskift på en vektor.

Synops

```
newX = shift(oldX,x)
```

`oldX`	:	Radvektor `oldX` = $[x_1\ x_2\ \ldots\ x_{nx}]$
`x`	:	x_0
`newX`	:	Uppdaterad vektor `newX` = $[x_0\ x_1\ \ldots\ x_{nx-1}]$

TH2POL

Beskrivning

Konverterar vektorn $\hat{\theta}$ till A och B polynom.

Synops

```
[A,B] = th2pol(theta,nn,sh)
```

`theta`	:	Radvektor `theta` = $[a_1\ a_2\ \ldots a_{na}\ b_0\ b_1\ \ldots\ b_{nb}]$
`nn`	:	Vektor som definierar antalet parametrar i A och B samt fördröjningen: `nn` = $[\ na\ \ nb\ \ nk\ ]$
`sh`	:	Anger om A och B skall returneras i framåt- eller bakåtskift. `sh = 'fs'` ger framåtskift `sh = 'bs'` ger bakåtskift
`A,B`	:	Polynomen A och B

TH2RST

Beskrivning

Konverterar en vektor som innehåller skattningarna av reglerparametrarna R, S och T i styrlagen:

$$R^*u(k) = -S^*y(k - nsk) + T^*u_c(k - ntk)$$

Synops

```
[R,S,T] = th2rst(th,nr,ar,ns,nsk,nt,ntk,sh)
```

`th`	:	Radvektor `th` = $[r_1\ r_2\ \ldots\ r_{nr}\ s_0\ s_1\ \ldots\ s_{ns}\ t_0\ t_1\ \ldots\ t_{nt}]$
`nr,ns,nt`	:	Antalet skattade parametrar i R, S respektive T
`nsk,ntk`	:	Fördröjning för S och T

`ar`	:	Fördefinierad pol i R polynomet. Ex. Integrator $A_r = [1 - 1]$
`sh`	:	Anger om R, S och T skall returneras i framåt- eller bakåtskift. `sh = 'fs'` ger framåtskift `sh = 'bs'` ger bakåtskift
`R,S,T`	:	R, S och T polynomen

PLOTYU

Beskrivning

Plottar en vektor som har formen enligt (3.1).

Synops

```
plotyu(yin,uin,rin,time)
plotyu(yin,uin,rin,time,tstart)
plotyu(yin,uin,rin,time,tstart,tend)
```

`yin,uin,rin`	:	Vektorer som plottas
`time`	:	Vektor som innehåller tidpunkter för sampel
`tstart`	:	Starta plott från tidpunkten `tstart`
`tend`	:	Avsluta plott vid vid tidpunkten `tend`

PLOTTH

Beskrivning

Plottar en matris som har formen enligt (3.2).

Synops

```
plotth(th,Pd,mark,time)
plotth(th,Pd,mark,time,tstart)
plotth(th,Pd,mark,time,tstart,tend)
plotth(th,Pd,mark,time,tstart,tend,tit)
```

`th`	:	Parameterskattning på formen 3.2
`Pd`	:	Diagonalelementen i P-matrisen på formen 3.2
`mark`	:	Då `mark = 1` sker märkning av parametrarna
`time`	:	Vektor som innehåller tidpunkter för sampel
`tstart`	:	Starta plott från tidpunkten `tstart`
`tend`	:	Avsluta plott vid vid tidpunkten `tend`
`tit`	:	Textsträng med valfri titel.

4. Indirekt adaptiv reglering

I detta kapitel kommer ett större exempel att visa, hur den indirekta självinställande regulatorn kan simuleras. I avsnitt 4.1 presenteras den process som skall simuleras och under vilka betingelser det skall ske. Avsnitt 4.2 visar hur systemet skall implementeras i Matlab. Hur simuleringar och experiment utförs visas i avsnitt 4.3 och avslutningsvis finns en kort sammanfattning i avsnitt 4.4.

4.1 Processen

Den process som skall simuleras, beskrivs av överföringsfunktionen

$$G_p(s) = \frac{B(s)}{A(s)} = K\frac{\omega_0^2(Ts-1)}{(sT+1)(s^2+2\zeta\omega_0 s+\omega_0^2)}e^{-\tau s} \tag{4.1}$$

med följande värden på parametrarna

$K = K_n + \Delta K$	$K_n = 1$	$-0.5 \leq \Delta K \leq 2$
$\tau = \tau_n + \Delta\tau$	$\tau_n = 0.2$	$0.0 \leq \Delta\tau \leq 0.2$
$T = T_n + \Delta T$	$T_n = 1$	$-0.2 \leq \Delta T \leq 0.5$
$\omega_0 = \omega_{0n} + \Delta\omega_0$	$\omega_{0n} = 15$	$-5 \leq \Delta\omega_0 \leq 1$
$\zeta = 1$		

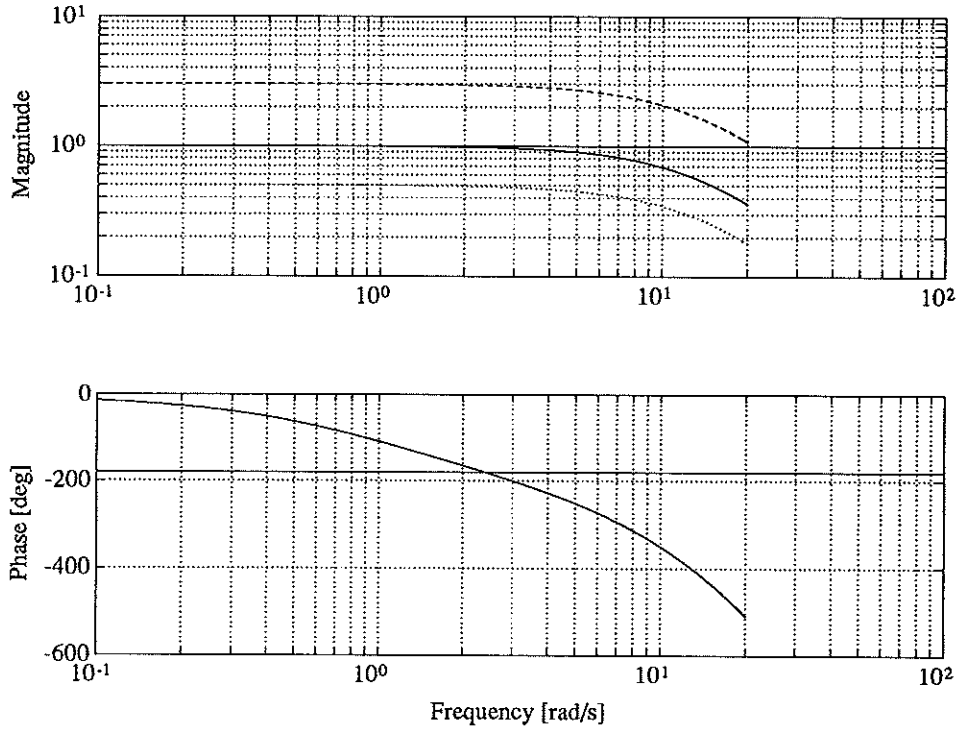
Det öppna systemets uppförande

Processen har tre poler, vilka varierar i snabbhet. Det innebär att problem kan uppstå när man skall försöka identifiera parametrarna. Om man studerar bodediagrammet för det öppna system, se figurerena 4.1 - 4.4, finner man att:

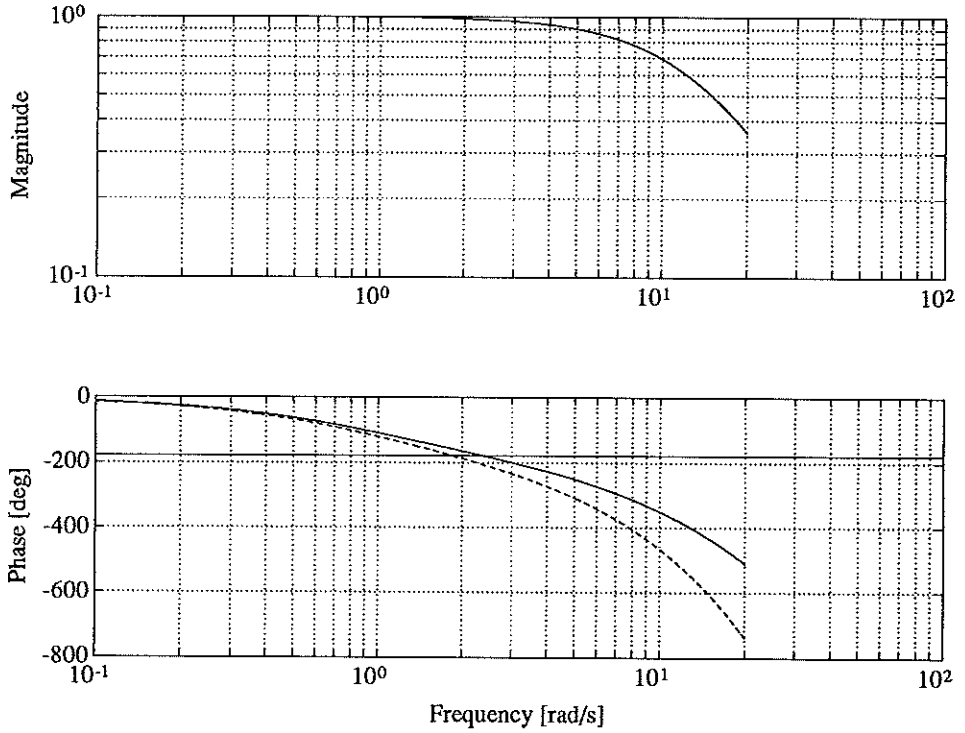
$$\omega_{-180} \approx 2\ rad/s$$

Där ω_{-180} är frekvensen då $arg(G_p(i\omega)) \approx -180°$. Dessa värden gäller för de flesta paramtrar.

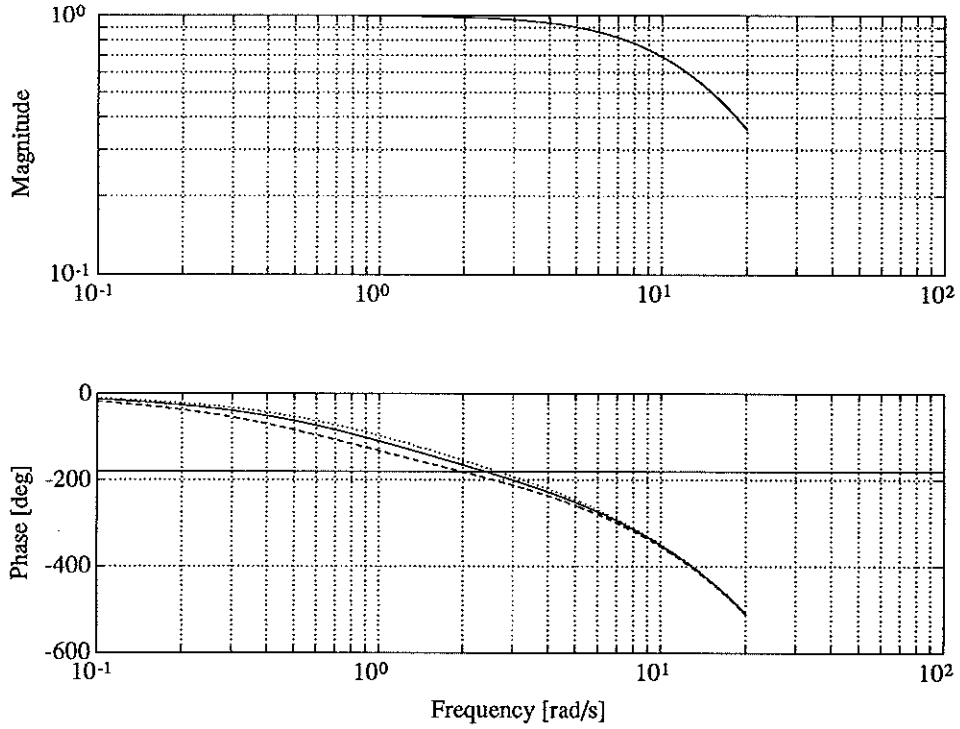
Slutsatserna man kan drar från bodediagrammet är att de intressanta frekvenserna ligger runt 1 till 2 rad/s. Samplingsfrekvensen f_s bör väljas så att $f_s/2$ ligger över detta område. Då det intressanta frekvensområdet ligger mycket lägre än de snabba polerna, kan man fråga sig om dessa behöver identifieras. Studerar man bodediagramen, finner man att det borde räcka att identifiera en modell av ordningen ett eller två. Man kan alltså se de komplexkonjugerade polerna som omodellerad dynamik. Tidsfördröjningen går däremot ej att bortse ifrån.



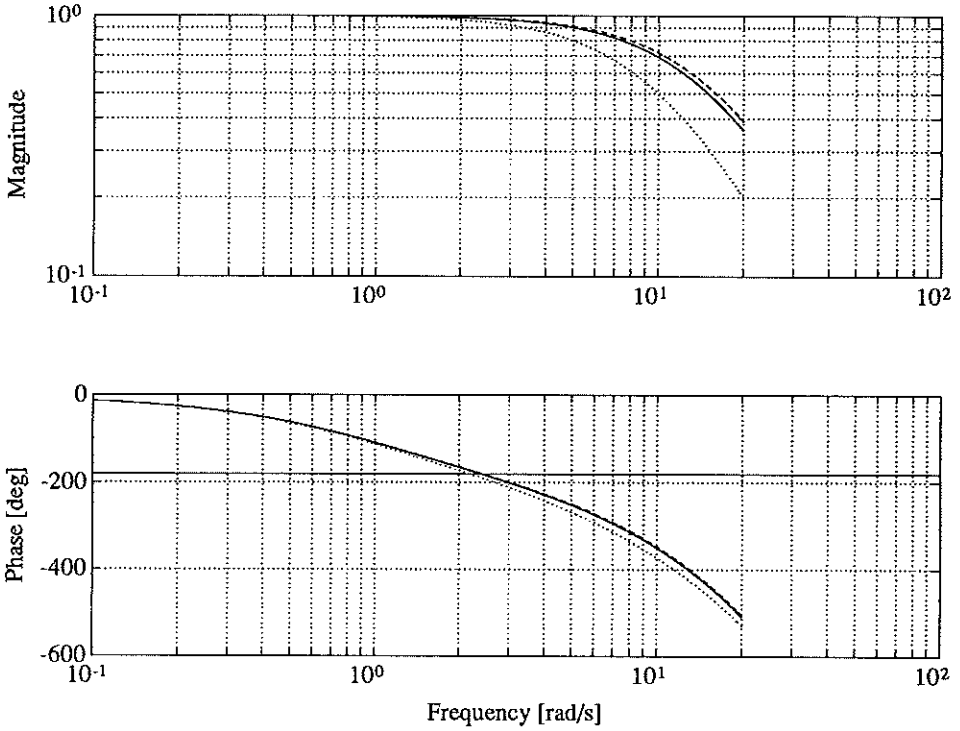
Figur 4.1 Bodediagram för $G_p(i\omega)$ då ΔK varierar. (—) = nominellt. (- - -):$\Delta K = 2$ och $(\cdots)$:$\Delta K = -0.5$



Figur 4.2 Bodediagram för $G_p(i\omega)$ då $\Delta\tau$ varierar. (—) = nominellt. (- - -):$\Delta\tau = 0.2$



Figur 4.3 Bodediagram för $G_p(i\omega)$ då ΔT varierar. (—):nominellt. (- - -):$\Delta T = 0.5$ och $(\cdots)$:$\Delta T = -0.2$

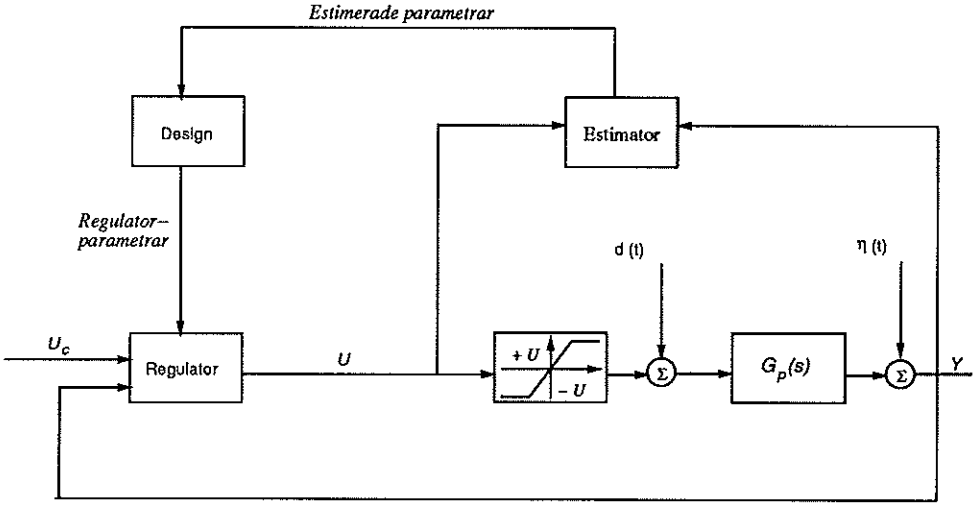


Figur 4.4 Bodediagram för $G_p(i\omega)$ då $\Delta\omega_0$ varierar. (—) = nominellt. (- - -):$\Delta\omega_0 = 1$ och $(\cdots)$:$\Delta\omega_0 = -5$

Regulatorn och reglermål

Processen skall regleras med hjälp av den indirekta självinställande regulatorn. Systemet har ett blockschema enligt figur 4.5. Observera att Processen innehåller en olinjäritet, som begränsar styrsignalen till $\pm U$. Designmetoden som används är polplacering.

Syftet med regleringen är att det slutna systemet skall följa referensvärdet utan översläng. Stigtiden till 95 % skall vara runt $4T$, där T ges av (4.1). Regulatorn skall också klara av laststörningar och mätbrus.



Figur 4.5 Blockschema för systemet.

Beteckningarna i figuren, står för följande signaler.

U_c	Referenssignal
$u(t)$	Processens insignal
$y(t)$	Mätbar utsignal från processen
$d(t)$	Laststörning (ej mätbar)
$\eta(t)$	Mätbrus
U	Mättningsnivå på processingång

Val av samplingsintervall

Ett rimligt val av samplingsintervall är att låta

$$N_r = \frac{T_r}{h} \approx 4 - 10$$

där N_r antal sampel per stigtid. Om man antar att det räcker att ha en modell av första ordningen ger det i detta fallet

$$h = \frac{T_r}{N_r} = \approx 0.15 - 0.2$$

uträknat för de värsta fallen. Ett val av samplingsintervall kan vara

$$h = 0.2 \Leftrightarrow f_s = 5Hz$$

vilket medför att nyquistfrekvensen ligger runt 16 rad/s.

Samplad process

Då processen innehåller en tidsfördröjning, kommer den tidsdiskreta överföringsfunktionens ordning att bero på valt samplingsintervall. Antalet extra poler d i origo, kan bestämmas genom sambandet

$$\tau = (d-1)h + \tau' \qquad \text{där} \quad 0 < \tau' \leq h \tag{4.2}$$

Vid sampling av processen (4.1), övergår överföringsfunktionen till

$$H(q) = \frac{B(q)}{A(q)} = \frac{b_0 q^3 + b_1 q^2 + b_2 q + b_3}{q^d(q^3 + a_1 q^2 + a_2 q + a_3)} \tag{4.3}$$

eller uttryckt i bakåtskiftopertorn

$$H(q^{-1}) = \frac{b_0 + b_1 q^{-1} + b_2 q^{-2} + b_3 q^{-3}}{1 + a_1 q^{-1} + a_2 q^{-2} + a_3 q^{-3}}\, q^{-d} \tag{4.4}$$

Observera att $d \geq 1$ för att processen ej skall innehålla någon direktterm. Med det valda samlingsintervallet, kommer d i detta fallet vara

$$d = \begin{cases} 1, & \Delta\tau = 0 \\ 2, & 0 < \Delta\tau \leq 0.2 \end{cases}$$

Val av slutet system och regulatordesign

Med utgångspunkt ifrån specifikationerna och det öppna systemet, skall det önskade slutna systemet bestämmas. Då systemet är ett icke-min-fas system, kan ej några nollställen förkortas bort. Tidsfördröjningen i systemet går ej att bortse ifrån, utan modellen måste innehålla någon form av tidsfördröjning.

Det öppna systemet har tre poler. En långsam reell, och två komplexkonjugerade snabba. Som nämnts tidigare vore det intressant att försöka göra ett slutet system av första ordningen, innehållande tidsfördröjningar. De snabba polerna betraktas då som omodellerad dynamik. Ett förslag på det slutna systemets karakteristiska polynom är.

$$A_m(s) = T_m s + 1 \tag{4.5}$$

I tidsdiskret form fås

$$A_m(q) = q^{d_m}(q + a_{m_1}) \tag{4.6}$$

Där d_m anger den valda tidsfördröjningen. Det slutna systemets nollställen väljs som

$$B_m = B^- B'_m \qquad \text{där} \quad \begin{matrix} B^- = B \\ B^+ = 1 \end{matrix} \tag{4.7}$$

B'_m väljs så att systemet får stationära förstärkningen 1 . Gradtalet på observeraren bestäms via sambandet

$$deg A_o \geq 2deg A - deg A_m - deg B^+ + l - 1 \tag{4.8}$$

där l anger antalet integratorer i regulatorn. Om man utgår ifrån att processen är av ordningen 5, d.v.s $\Delta\tau > 0$, kan följande värden väljas.

$$d_m = 4$$
$$deg A_o = 5$$

Ett förslag på observeraren är att låta den ha samma struktur som A_m i (4.6).

Process som skall skattas

Som nämnts tidigare uppträder processen som ett först ordningens system med tidsfördröjningar, i frekvensintervallet som ligger under nyquistfrekvensen. En lämplig modell att skatta är då

$$H(q^{-1}) = \frac{b_0 + b_1 q^{-1} + b_2 q^{-2} + b_3 q^{-3}}{1 + a_1 q^{-1}} \, q^{-2} \tag{4.9}$$

4.2 Implementering av systemet

I appendix B finns den kompletta filen som visar hur hela systemet är implementerat i Matlab. I detta avsnitt presenteras hur denna fil är uppbyggd. Först visas hur de olika delarna i systemet skall initialiseras därefter hur beräkningssekvensen skall byggas upp.

Processen

För att beräkna processens utsignal, används funktionen

```
[y(k+1),x(k+1) ]= process(x(k),u(k),procpar)
```

där

$$procpar = [\Phi \; \Gamma \; C^T]$$

För att skapa processmodellen, görs följande i Matlab.

```
%**** make process model ****

Apc           = conv([1 1/T],[1 2*z*wo wo^2]); % process denominator
Bpc           = - K*wo^2*[1 -1/T];             % process nominator
[Bpd,Apd]     = sample(Bpc,Apc,tau,h)          % sampled process
[Phi,Gam,C,D] = tf2ss(Bpd,Apd);                % transform to state -
                                               % space representation
procpar       = [Phi Gam C'];
x             = zeros(length(Phi),1);          % initial state
```

Lista 4.1 Matlabkod för att initialisera processen som skall simuleras

Variablerna T, z, ω_o o.s.v måste tilldelas värden innan dessa kommando utförs. Observera också att processens initialtillstånden sätts till noll.

Slutna systemet

För att skapa de variabler som anger hur det slutna systemet skall se ut, utförs följande.

```
%***** Design the Closed Loop *****

Tm  = 1;                                       % closed loop pole
dm  = 4;                                       % closed model time delay
Amc = [1 1/Tm];                                % closed loop den
Amd = polyc2d(Amc,h);                          % sample Amc
Amd = [Amd zeros(1,dm)];                       % closed loop den
To  = 1.5;                                     % Observer pole
Aoc = [1 1/To];                                % Observer cont. time
Aod = [polyc2d(Aoc,h) zeros(1,dm)];            % sample observer
Ar  = [1 -1];                                  % integrator
```

Lista 4.2 Matlabkod för att initialisera slutna systemet och observeraren.

Initialisering av estimatorn

För att skatta processen används minstakvadratskattning. Denna metod finns implementerad i funktionen RLS. För att initialisera estimatorn, så den valda processmodellen skattas se (4.9), exekveras följande Matlabkod.

```
%***** Initialization of estimator *****

na       = 1;                  % number of estimated
nb       = 4;                  % a and b parameters.
nk       = 2;                  % numbers of time delay.
nn       = [na nb nk];
lam      = 1.00;               % forgetting factor.
th0      = 0.01*ones(na+nb,1); % initial value on theta
P0       = 1000*eye(na+nb);    % vector and P matrix.
rlsstate = [];                 % default value.
```

Lista 4.3 Matlabkod för att initialisera slutna systemet och observeraren.

Observera att variabeln *rlsstate* sätts till en tom matris, vilket innebär att funktionen RLS skapar en vektor med rätt antal element. Ur denna vektor skapas sedan regressorvektorn. Värt att notera är att *th*0 är en kolumnvektor med na + nb element.

Initialisering av Regulatorn

Det finns två regulatorfunktioner implementerade som båda utnyttjar polplacering. Den ena utan *antiwindup* den andra med. Jag väljer att använda regulatorn med *antiwindup* här, vilken är implementerad i funktionen RST2. Den initialiseringen som måste göras, är att beräkna de första R, S och T polynomen. Detta kan göras på många sätt. Här görs en design utifrån initialvärdena på de skattade processparametrarna. Designfunktionen som används är RSTD.

```
%***** Initialize regulator *****

[A,B]   = th2pol(th0,nn,'fs');          % unpack th to A and B polynomial
[r,s,t] = rstd(1,Bpd,Apd,1,Amd,Aod,Ar); % design regulator parameters
```

Lista 4.4 Initiering av R, S och T polynomen som används av regulatorn.

Det första kommandot omformar th vektorn till A och B polynom. Inparametern `'fs'` talar om att A och B skall skrivas i framåtskift.

Signaler

Det som återstår nu är att generera de signaler som skall påverka systemet. Funktionen som används för att skapa signalerna är YUSIGNALS. För att generera en signalsekvens som:

$$u_c(t) = \begin{cases} 10 & \text{för} \quad T_0 \le t \le T_0 + 10T \\ -5 & \text{för } T_0 + 10T \le t \le T_0 + 20T \\ 10 & \text{för } T_0 + 20T \le t \le T_0 + 30T \end{cases}$$

$$\sigma = 0.2 \qquad \text{för alla t}$$

utförs de Matlabkommando som finns i lista 4.5. T_0 är längden för uppstart, i detta fall valt till 0 sekunder.

```
%***** Signals *****

time = 0:h:30;                                          % Simulation time.
rlev=  [10  -5 10];                    % Reference signal.
rch  = [ 0  10 20];
dlev=0;                                                 % Load disturbance.
dch=0;
nvar=0.2;                                               % Measure Noise.
nch=0;

trldn=yusignals(time,rlev,rch,0,0,dlev,dch,nvar,nch);% Make ref,load and
                                                        % noise signals.
uc=trldn(:,2);                                          % Reference signal
d =trldn(:,4);                                          % Load disturbance.
n =trldn(:,5);                                          % Measure Noise.
```

Lista 4.5 Matlabkod för generering av signaler

YUSIGNALS skapar en matris utifrån variablerna *lev och *ch. Variablerna *ch anger vid vilken tidpunkt signalen skall ha amplituden *lev. Undantaget är nvar som anger vilken varians bruset har. Anledningen till att skapa separata variabler för referenssignal, laststörning och brus är att öka läsbarheten.

Beräkningssekvensen

Det återstår nu att bygga upp den sekvens som skall utföra beräkningarna. Sekvensen innehåller de funktioner, man valt skall representera de ingående blocken i figur 4.5.

```
%***** Simulation block *****

for k=1:length(time)

%---- Control signal ----
   [u(k),regstate]= rst2(r(k,:),s(k,:),t(k,:),[],ulim,y(k),uc(k),regstate);

%---- Estimation ----
   [th(k,:),P0,rlsstate] = rls(yf(k),uf(k),nn,lam,th0,P0,rlsstate);
   th0                   = th(k,:)';
   Pd(k,:)               = diag(P0)';

%---- Design ----
   [A,B]                          = th2pol(th(k,:),nn,'fs');
   [r(k+1,:),s(k+1,:),t(k+1,:)]   = rstd(1,B,A,1,Amd,Aod,Ar);

%---- Update Process ----
   up              = sat(u(k),ulim) + d(k);
   [y(k+1),x]      = process(x,up,procpar);
   y(k+1)          = y(k+1) + n(k);
end
```

Lista 4.6 Matlabkod för beräkningssekvensen

Det första som sker är beräkning av styrsignalen. Regulatorpolynomen lagras i var sin matris. Matrisen har lika många rader som antal sampel. Antalet kolumner är lika med antalet parametrar. R(k,:) betyder rad k och alla kolumner. Den tomma matrisen som matas in innebär att regulatorn får en *deadbeat* antiwindup. ULIM är en vektor som innehåller mättningsgränserna. REGSTATE är en vektor innehållande gamla värden på y, u och u_c. Utparametrarna är styrsignalen och uppdaterad tillståndsvektor.

Andra steget i sekvensen är skattning av processen. Funktionen som utför det är RLS. Raden `th0 = th(k,:)'` är till för att RLS skall få de gamla parameterskattningarna. Det skulle vara bättre att ha `th(k-1,:)'` som inparameter till RLS men det ger problem då $k = 1$, se kapitel 2. Alternativet skulle vara att låta k gå från 2 och uppåt. Observera att de ny parameterskattningarna kommer ut i en radvektor. Detta är anledningen varför man måste göra transponat på `th(k,:)` när `th0` tilldelas dess värde. Sista raden i detta block är till för att spara diagonalelementen i P matrisen.

Tredje steget är design av regulatorparametrarna. Koden är precis den samma som vid initialiseringen av regulatorn.

Sista steget i sekvensen är uppdateringen av processen. Första raden representerar mättningen i processen. Här adderas också laststörningen till. Processen uppdateras sedan med PROCESS och sist adderas mätbruset till.

Kommentaren till den fullständiga koden

Den fullständiga Matlabkoden för detta exempel återfinns i Appendix B. En del tillägg har gjorts som är värda att kommentera. För att kunna trimma regulatordesignen och skattaren är det värdefullt att kunna stänga av och på både estimator- och designrutinerna. Genom att införa två parametrar ESTIM och ADAPT, kan man testa på dessa och på så sätt utföra de kommando som är behövliga för de olika fallen.

Då man har tillgång till alla rutiner i Matlab, finns det oanade möjligheter att analysera regulatorn under simuleringen. Här har rutiner lagts in som plottar bodediagram och nyquistdiagram för skattat och verkligt system. Dessa rutiner utförs var 20:e sampel. Detta ger tillgång till ovärderlig information under simuleringen, hur systemet uppför sig i frekvensplanet.

4.3 Experiment

Intrimning av Regulatordesignen

Detta experiment är till för att se hur regulatorn upp för sig och hur väl systemet följer referensvärdet. Försöken utgår ifrån att processen är känd. Referenssignalen, laststörningen och mätbruset genereras som:

$$
\begin{aligned}
u_c(t) &= \begin{cases} 10 & \text{för} \quad T_0 \le t \le T_0 + 10T \\ -5 & \text{för } T_0 + 10T \le t \le T_0 + 20T \\ 10 & \text{för } T_0 + 20T \le t \le T_0 + 30T \end{cases} \\
d(t) &= \begin{cases} 0 & \text{för} \quad T_0 \le t \le T_0 + 20T \\ -5 & \text{för } T_0 + 20T \le t \le T_0 + 30T \end{cases} \\
\sigma &= \begin{cases} 0 & \text{för} \quad T_0 \le t \le T_0 + 20T \\ 0.2 & \text{för } T_0 + 20T \le t \le T_0 + 30T \end{cases} \\
T_0 &= 0
\end{aligned}
\tag{4.10}
$$

Matlab koden för att generera denna signalsekvens bli:

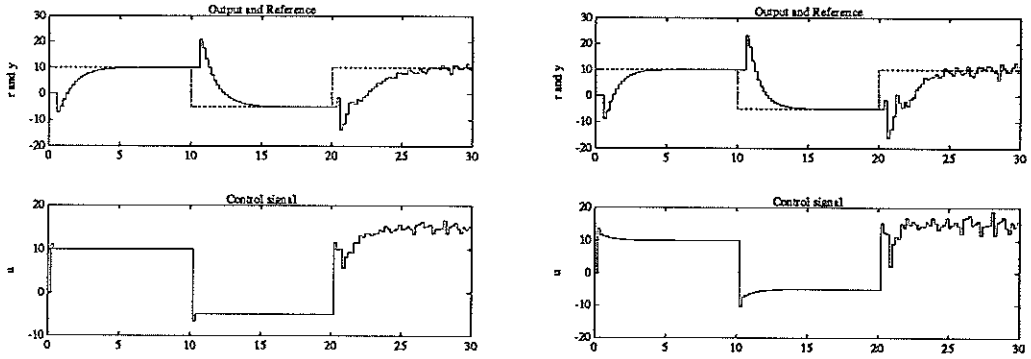
```
%***** Signals *****

time = 0:h:30;                          % Simulation time.
rlev= [10  -5 10];                      % Reference signal.
rch  = [ 0  10 20];
dlev=-5;                                % Load disturbance.
dch=20;
nvar=0.2;                               %  Measure Noise.
nch=20;
```

Lista 4.7 Matlabkod för generering av signaler

Funktionsanropet till `YUSIGNALS` är motsvarande som i lista 4.5.

Design av systemet diskuterades i avsnitt 4.1. De inledande simuleringarna bygger på resultaten som framkom där.

$$h = 0.2$$
$$A_m(q) = q^4(q + a_{m_1})$$
$$A_o(q) = q^4(q + a_{m_2})$$



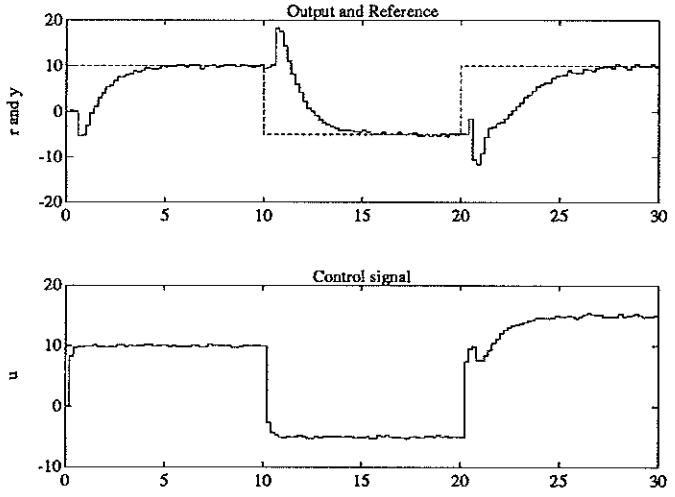
Figur 4.6 Utsignal och styrsignal för a) $a_m = 1$, $a_o = 1$ (vänstar kolumnen) b) $a_m = 1.25$, $a_o = 2$(högra kolumnen)

Det är ingen större skillnad på snabbheten för systemet med de olika modell- och observerarpolerna. De högfrekventa signalerna förstärks mer med de snabbare polerna vilket får till följd att styrsignalen och därmed utsignalen får ett ryckigare utseende. Med tanke på att man ej vinner något i snabbhet bör därför de långsamma polerna väljas.

Med den valda strukturen på modell- och observerarpolynom, får man ett system med liknande egenskaper som ett system med *deadbeat* observerare. För att slippa detta kan man ansätta A_m och A_o som:

$$A_m(s) = (s + \frac{1}{T_m})(s + \frac{10}{T_m})^2(s + \frac{100}{T_m})^2$$
$$A_o(s) = (s + \frac{1}{T_o})(s + \frac{10}{T_o})^2(s + \frac{100}{T_o})^2$$

Resultatet av simuleringen med val av A_m och A_o som ovan, visas i figur 4.7. Snabbheten på systemet har ej ändrats nämnvärt, men styrsignalen har fått ett mycket lugnare utseende.



Figur 4.7 Utsignal och styrsignal för $T_m = 1, T_o = 1$

Intrimning av Skattaren

I detta experimentet undersöks hur den adaptiva regulatorn klarar av att skatta processen, då systemet utsätts för brus och laststörning. De okända processparametrarna hålls konstanta under simuleringen. Signalerna som skall påverka systemet genereras som:

$$u_c(t) = \begin{cases} 10 & \text{för} \quad T_0 \le t \le T_0 + 10T \\ -5 & \text{för } T_0 + 10T \le t \le T_0 + 20T \\ 10 & \text{för } T_0 + 20T \le t \le T_0 + 30T \end{cases}$$

$$d(t) = \begin{cases} 0 & \text{för} \quad T_0 \le t \le T_0 + 20T \\ -5 & \text{för } T_0 + 20T \le t \le T_0 + 30T \end{cases}$$

$$\sigma = 0.2 \quad \text{för alla t}$$

$$T_0 = 10$$

Då man har tillgång till Matlabs övriga funktioner, finns möjlighet att under simuleringen göra analys av systemet. Genom att infoga Matlabkod enligt lista 4.8 i beräkningssekvensen, kan man plotta bodediagram och/eller nyquistdiagram för:

- Verklig process $\frac{B(q)}{A(q)}$
- Skattad process $\frac{\hat{B}(q)}{\hat{A}(q)}$
- Verklig kretsöverföringsfunktion $\frac{S(q)}{R(q)}\frac{B(q)}{A(q)}$
- Skattad kretsöverföringsfunktion $\frac{S(q)}{R(q)}\frac{\hat{B}(q)}{\hat{A}(q)}$

Variablerna `FR1`, `FR2` och `FR3` innehåller frekvenssvaret för verklig process, skattad process samt frekvenssvaret för $S(q)/R(q)$. Genom att använda funktionen `FMUL` kan frekvenssaret för kretsöverföringsfunktionen erhållas. Funktionsanropet `FMUL(FR1,FR3)` ger frekvenssvaret för den verkliga kretsöverföringsfunktionen. Genom att byta ut `FR1` mot `FR2` erhålls den skattade.

```
if rem(k,20)==0,
   k
   fr1 = frd(Bpd,Apd,h,-2,[],80);
   fr2 = frd(B,A,h,-2,[],80);
   fr3 = frd(s(k+1,:),r(k+1,:),h,-2,[],80);
   clg
   bopl(fr1,fr2,[-1 2 -1 1 -500 180]);
   pause(1);
   nypl(fmul(fr1,fr3),fmul(fr2,fr3),'125',[-4 4 -4 4]);
end
```

Lista 4.8 Matlabkod för plot av nyquist- och bodediagram

Figur 4.8 visar bodediagrammet för den riktiga processen (heldragen linje) och den skattade (streckad) samt nyquistdiagrammet för riktig och skattad kretsöverföringsfunktion. Diagrammen är beräknade vid $t = 36sek$, d.v.s 30 sampel efter det att laststörningen börjat påverka systemet. Man ser att processmodellen underskattar amplituden i hela frekvensområdet.

Systemets utsignal, styrsignal och de skattade parametrarna visas i figur 4.9. Det intressantaste är att titta på plotten för skattningen. När man studerar denna verkar skattningen vara bra. Den ställer in sig snabbt och ändrar sig inte nämnvärt under simuleringen. Slutsatsen man drar här är den motsatta som man drar när man studerar bodediagrammet. Bodediagrammet visar ju tydligt att skattningen inte alls är speciellt bra i vissa frekvensområden.

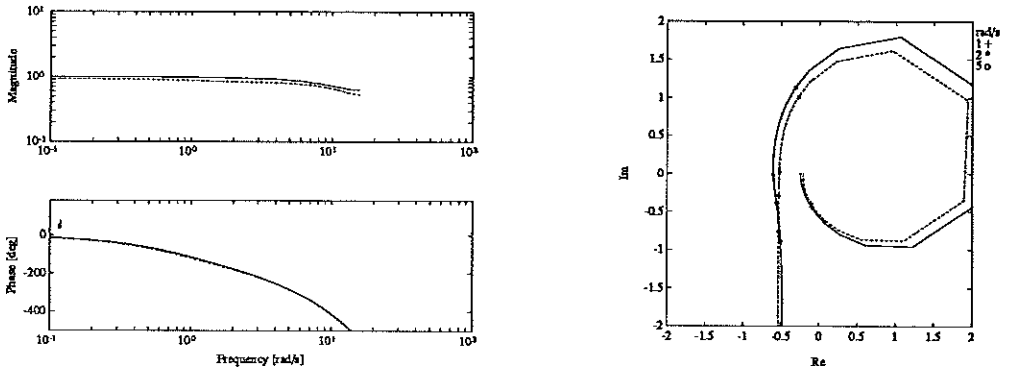
Genom att filtrera signalerna som skattningen bygger på, bör man få en bättre skattning. Detta filter bör vara av typen bandpass. I Matlab finns där funktioner som beräknar olika typer av filter av godtycklig ordning. Lista 4.9 visas hur man i Matlab skapar och lägger in filtret i blockschemat.

```
%*****  Include in Initialization of estimator *****
[bf,af]  = butter(2,[0.01,0.5]);            % bandpassfilter


%*****  Include in simulation Loop *****

[yf(k),yfstate] = filt(y(k),yfstate,bf,af);
[uf(k),ufstate] = filt(u(k),ufstate,bf,af);
[th(k,:),P0,rlsstate] = rls(yf(k),uf(k),nn,lam,th0,P0,rlsstate);
```

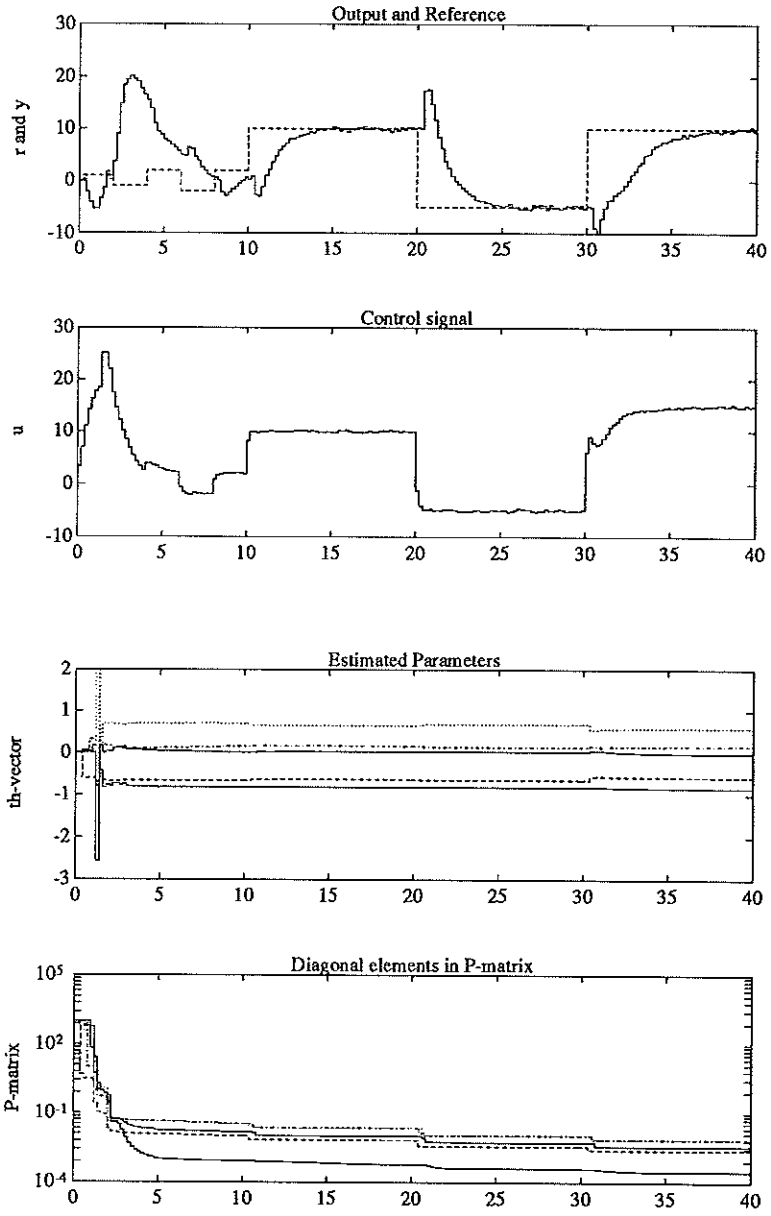
Lista 4.9 Matlabkod för filtering av signaler

Funktionen BUTTER skapar ett Butterworth filter, här av andra ordningen. Observera att brytfrekvenserna `[0.01,0.5 ]` är normerad frekvens. Skapande av filtret kan läggas in i initieringen av estimatorn. De två följande raderna skall inkluderas i beräkningssekvensen. FILT filtrerar en signal med filtret som har överföringsfunktionen b_f/a_f. Inargument till RLS skall nu vara de filtrerade ut- och insignalerna.

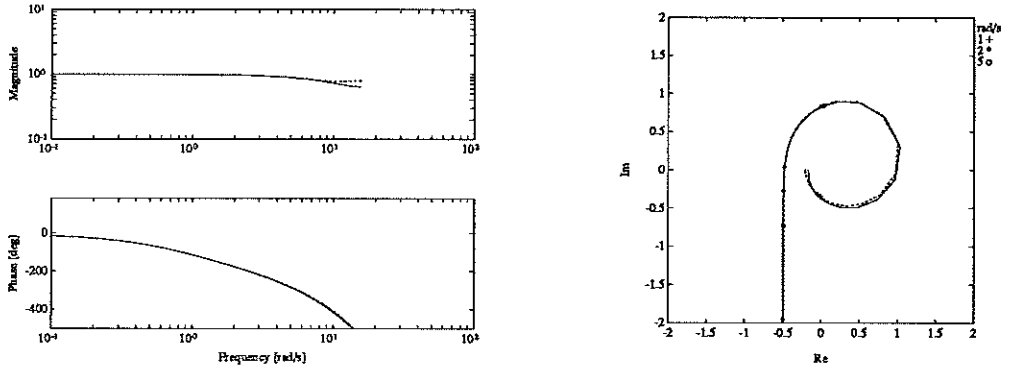
Exakt samma simulering utförs på systemet, men med förfiltrerad data till skattaren. Figur 4.10 visar en serie bode- och nyquistdiagram som är beräknade vid olika tidpunkter under simuleringen. Bodediagramet visar verklig och skattad process. Nyquistdiagramet visar verklig och skattad kretsöverföring. Då dessa digram visas kontinuerligt under simuleringen, får man en god möjlighet att förstå vad som händer i de olika faserna.



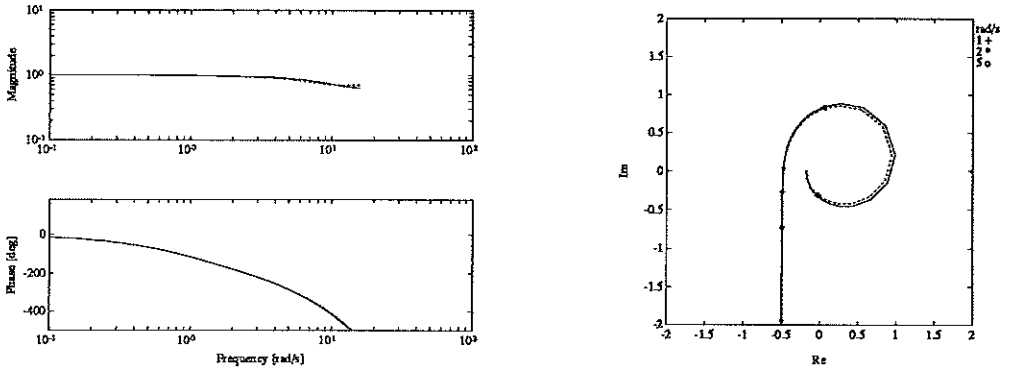
Figur 4.8 Bodedigram för process och nyquistdiagram för kretsöverföringsfunktion vid $t = 36sek.$(Verklig = (—) skattad = (···)



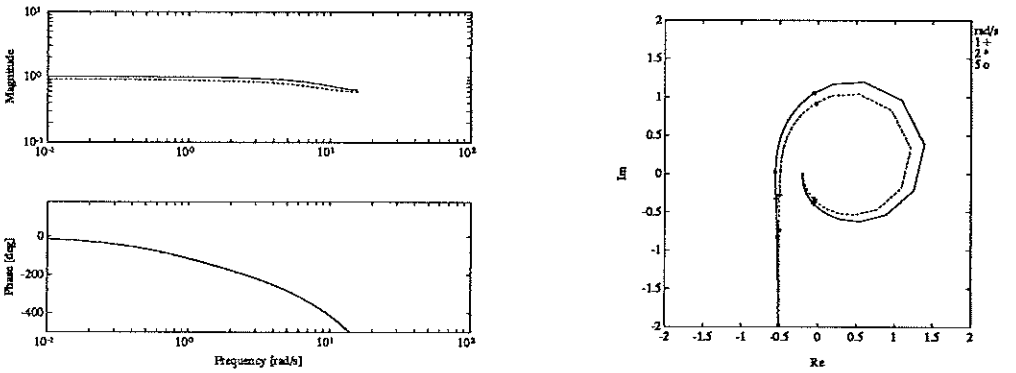
Figur 4.9 Utsignal, referensvärde och styrsignal för systemet. Ofiltrerad estimeringsdata.



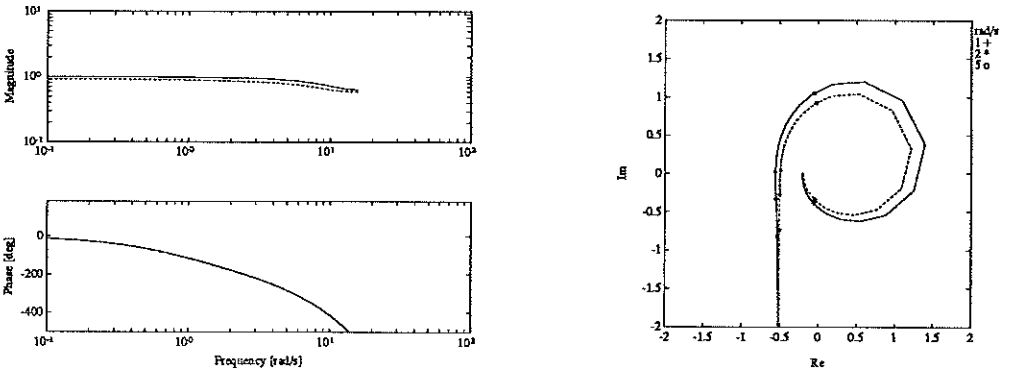
Figur 4.10 a) $t = 8sek$



Figur 4.10 b) $t = 24sek$

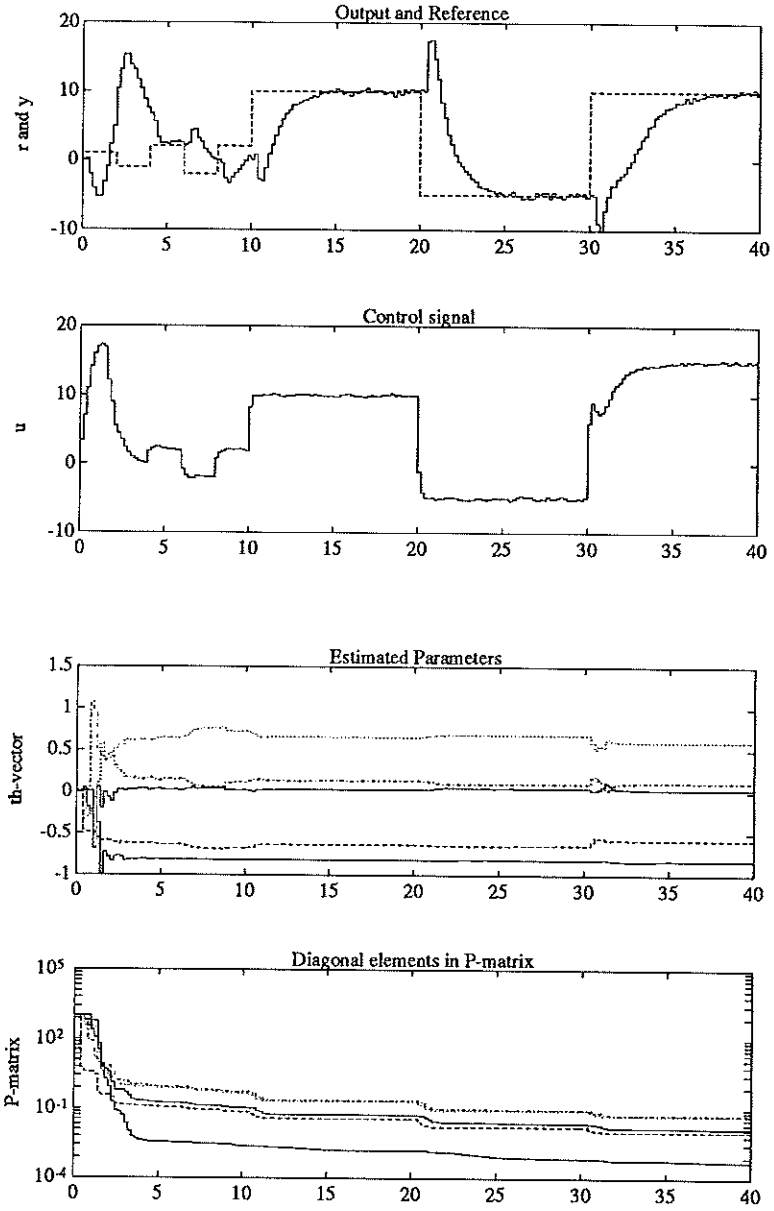


Figur 4.10 cc) $t = 36sek$



Figur 4.10 d) $t = 40sek$

Figur 4.10 Bodediagram för process och nyquistdiagram för kretsöverföringsfunktionen. (Verklig = (—) skattad = (···). Filtrerad



Figur 4.11 Utsignal, referensvärde och styrsignal för systemet. filtrerad estimeringsdata.

Jämförs figur 4.9 och 4.11 kan någon större skillnad mellan de olika fallen ej upptäckas vad gäller styrsignal och utsignal. En direkt jämförelse mellan de olika fallen kan göras vid $t = 36sek$, d.v.s 6 sekunder efter laststörning börjar påverka systemet. Bodediagramen (figur 4.8 och 4.10 c) för skattad och verklig process, är förvillande lika. Här syns ingen förbättring i skattningen. Nyquistkurvan visar däremot att processen skattas lite bättre vid låga frekvenser, med förfiltrering.

4.4 Slutsatser om simuleringen

Detta kapitel har visat vilka möjligheter man har att implementera och simulera ett adaptivt system i Matlab. Den största fördelen framför Simnon är

att man kan skatta många parametrar och analysera systemet under själva simuleringen. På detta sätt får man en ovärderlig insikt i hur systemet uppför sig under olika faser av simuleringen.

Den fullständiga Matlabfilen finns i Appendix B. Tanken med denna fil är att fungera som en arbetsbänk (*Workbench*), vilken kan användas som utgångspunkt för att implementera och simulera andra indirekta adaptiva system.

5. Direkt adaptiv reglering

Detta kapitel kommer att visa med hjälp av ett experiment, hur den direkta självinställande regulatorn kan implementeras och simuleras i Matlab. Avsnitt 5.1 tar upp bakomliggande teori för den algoritm som används. Avsnitt 5.2 visar hur systemet skall implementeras i Matlab. Den process som används i simuleringarna, presenteras i avsnitt 5.3. Experimentförfarandet och resultaten avhandlas i avsnitt 5.4.

5.1 Algoritm

I den indirekta självinställande regulatorn erhålls regulatorparametrarna genom att lösa den Diophantiska ekvationen

$$AR_1 + B^- S = A_o A_m \tag{5.1}$$

där

$$\begin{aligned} B &= B^+ B^- \\ B_m &= B^- B'_m \\ T &= A_o B'_m \\ R &= B^+ R_1 \end{aligned} \tag{5.2}$$

B^+ är processens stabila nollställen. Processen beskrivs av ekvationen

$$Ay(t) = Bu(t) + Ce(t) \tag{5.3}$$

eller uttryck i bakåtskiftsopperatorn

$$A^*(q^{-1})y(t) = B^*(q^{-1})u(t - d_0) + C^*(q^{-1})e(t)$$

Genom att multiplicera den Diophantiska ekvationen med $y(t)$ och utnyttja (5.3) kan (5.1) skrivas om som

$$A_o A_m y(t) = B^-(Ru(t) + Sy(t)) + R_1 Ce(t) \tag{5.4}$$

Låt det önskade systemets svar bestämmas som

$$y_m(t) = \frac{B_m}{A_m}\, u_c(t) = \frac{B^- B'_m}{A_m}\, u_c(t) = \frac{B^- T}{A_m A_o}\, u_c(t) \tag{5.5}$$

Felet $\varepsilon(t) = y(t) - y_m(t)$ ges då av

$$\varepsilon(t) = \frac{B^-}{A_o A_m}(Ru(t) + Sy(t) - Tu_c(t)) + \frac{R_1 C}{A_o A_m} e(t) \tag{5.6}$$

Antag att bruset $e(t)$ kan sättas till noll. Ekvation (5.6) kan då skrivas som

$$\begin{aligned} \varepsilon(t) &= \frac{B^-}{A_o A_m}(Ru(t) + Sy(t) - Tu_c(t)) \\ &= B^{*-}(R^* u_f(t) + S^* y_f(t) - T^* u_{cf}(t)) \end{aligned} \tag{5.7}$$

där

$$\begin{aligned} u_f(t) &= H_f(q^{-1})u(t) \\ y_f(t) &= H_f(q^{-1})y(t) \\ u_{cf}(t) &= H_f(q^{-1})u(t) \end{aligned} \tag{5.8}$$

$$H_f(q^{-1}) = \frac{q^{-d_0}}{A_o^* A_m^*}$$

Om B^- sätts till en konstant ($B^- = b_0$), dvs alla processnollställen förkortas, kan algoritm 5.3 i [2] användas. Denna algoritm innebär att B^- i föregående ekvationer skall sättas till b_0. Man har sedan möjlighet att skatta R, S och T polynomen. Används denna algoritm på processer med dåligt dämpade nollställen eller processer av icke-minfas typ, kommer man att få problem med styrsignalen.

För att slippa en oscillerande styrsignal får inga instabila eller dåligt dämpade nollställen strykas. Problemet är att B är okänt. Två möjliga sätt finns att lösa detta problem:

- Approximera B^-
- Skatta B

Att approximera B^- kräver förkunskap om processen. Om man däremot skattar B och delar upp polynomet som

$$\begin{aligned} B^+ &= 1 \\ B^- &= \hat{B} \end{aligned} \tag{5.9}$$

kan man utnyttja ekvationerna på föregående sida. Följande algoritm erhålls då

ALGORITM 5.1— Direkta *STR:n* utan nollställesförkortning

steg 1: Estimera processen med en rekursiv skattningsalgoritm.

steg 2: Beräkna önskad utsignal som

$$y_m(t) = \frac{\hat{B} B_m'}{A_m} u_c(t) \tag{5.10}$$

där B_m' bestäms så att önskad stationär förstärkning erhålls på det slutna systemet.

steg 3: Estimera R^*, S^* och T^* i ekvationen

$$\varepsilon(t) = R^* u_f(t) + S^* y_f(t) - T^* u_{cf}(t) \tag{5.11}$$

där

$$\begin{aligned} u_f(t) &= H_f(q^{-1})u(t) \\ y_f(t) &= H_f(q^{-1})y(t) \\ u_{cf}(t) &= H_f(q^{-1})u(t) \end{aligned} \tag{5.12}$$

$$H_f(q^{-1}) = \frac{\hat{B}^* q^{-d_0}}{A_o^* A_m^*}$$

med en rekursiv skattningsalgoritm.

steg 4: Beräkna styrsignalen med styrlagen

$$R^*u(t) = -S^*y(t) + T^*u_c(t) \tag{5.13}$$

□

Då B är känt ($\hat{B}$) och R^* skall vara monisk, kan felet $\varepsilon(t)$ skrivas som

$$\varepsilon(t) = y(t) - y_m(t) - u_f(t) \tag{5.14}$$

Detta innebär att man ej behöver skatta den första koefficienten i R polynomet.

Integratorverkan

Om regulator skall innehålla en integrator, är det onödigt att skatta denna. R^* polynomet kan då skrivas som

$$R^* = (1 - q^{-1})R_1'^*$$

Nu kan felet $\varepsilon(t) = y(t) - y_m(t)$ skrivas som

$$\varepsilon(t) = R_1'^*\Delta u_f(t) + S^*y_f(t) - T^*u_{cf}(t)$$

där

$$\Delta u_f(t) = u_f(t) - u_f(t-1)$$

Beräkning av $\Delta u_f(t)$ kan ske på många sätt. Det enklaste, med tanke på implementeringen i Matlab, är att filtrera $u(t)$ med

$$\begin{aligned} H_{fu}(q^{-1}) &= H_f(q^{-1}) * (1 - q^{-1}) \\ H_f(q^{-1}) &= \frac{\hat{B}^*q^{-d_0}}{A_o^*A_m^*} \end{aligned} \tag{5.15}$$

$y(t)$ och $u_c(t)$ filtreras med H_f.

Algoritm 5.1 kan nu modifieras, så att den även klarar av fallet med integratorverkan.

ALGORITM 5.2—Direkta *STR:n* med integratorverkan utan nollställeförkortning

steg 1: Estimera processen med en rekursiv skattningsalgoritm.

steg 2: Beräkna önskad utsignal som

$$y_m(t) = \frac{\hat{B}B_m'}{A_m}u_c(t) \tag{5.16}$$

B_m' bestäms så att önskad stationär förstärkning erhålls på det slutna systemet.

steg 3: Estimera $R_1'^*$, S^* och T^* i ekvationen

$$\varepsilon(t) = R_1'^*\Delta u_f(t) + S^*y_f(t) - T^*u_{cf}(t) \tag{5.17}$$

där

$$\begin{aligned} u_f(t) &= H_{fu}(q^{-1})u(t) \\ y_f(t) &= H_f(q^{-1})y(t) \\ u_{cf}(t) &= H_f(q^{-1})u(t) \end{aligned} \qquad (5.18)$$

med en rekursiv skattningsalgoritm. H_{fu} och H_f enligt (5.15).

steg 4: Beräkna styrsignalen med styrlagen

$$R^*u(t) = -S^*y(t) + T^*u_c(t) \qquad (5.19)$$

där

$$R^* = R_1'^* * (1 - q^{-1})$$

□

Observera att $R_1'^*$ måste vara monisk, vilket innebär att felet kan skrivas som

$$\varepsilon(t) = y(t) - y_m(t) - \Delta u_f(t) \qquad (5.20)$$

vilket innebär att först koefficienten i $R_1'^*$ ej behöver skattas.

Algoritmerna 5.1 och 5.2 ger inte någon renodlad direkt *STR*. Det man erhåller är en hybrid mellan den indirekta och direkta. Största vinsten med denna metod, är att man slipper lösa den Diophantiska ekvationen. Lösningen av denna ekvation kan ge numeriska problem. Nackdelen med metoden, är att man måste skatta både processparametrarna och regulatorparametrarna. I Matlab är detta inget problem då skattningsalgoritmerna kan skriva på ett generellt sätt.

5.2 Implementering av systemet

Implementeringen av den direkta *STR:n* är uppbygd enligt samma princip som den indirekta. De delar av koden som sammanfaller för de båda typerna av regulatorer är:

- Initiering av processen.
- Skapa det önskade slutna systemet.
- Skattning av processparametrar.

För beskrivning av dessa delar hänvisas till kapitel 4.

I appendix B finns den kompletta filen hur systemet implementeras i Matlab. Systemet är implementerat så att både algoritm 5.1 och 5.2 kan användas utan större ändringar.

Skattning av regulatorparametrar

Skattningen av regulatorparametrarna byggs upp utifrån ekvationerna:

$$\begin{aligned} \varepsilon(t) &= R^*u_f(t) + S^*y_f(t) - T^*u_{cf}(t) \\ \varepsilon(t) &= y(t) - y_m(t) - u_f(t) \end{aligned}$$

Där $u_f(t)$ och R^* byts ut mot $\Delta u_f(t)$ och R_1^{-*} då regulatorn skall innehålla en integrator. Regressorn och parametervektorn kommer att se ut som

$$\theta = [r_1 \ldots r_{n_r} s_0 \ldots s_{n_s} t_0 \ldots t_{n_t}]^T$$
$$\varphi^T(t) = [u_f(t-1)..u_f(t-n_r) y_f(t-1)..y_f(t-n_s) u_{cf}(t-1)..u_{cf}(t-n_t)]$$
$$\epsilon(t) = \varphi^T(t)\theta$$

önskad regulatorstruktur skall anges på formen

$$\begin{aligned} R^*(q^{-1}) &= 1 + r_1 q^{-1} + \ldots \\ S^*(q^{-1}) &= (s_0 + s_1 q^{-1} + \ldots)\, q^{-nk_s} \\ T^*(q^{-1}) &= (t_0 + t_1 q^{-1} + \ldots)\, q^{-nk_t} \end{aligned}$$

Genom att ange antalet parametrar som skall skattas i R, S och T kan en godtycklig regulatorstruktur erhållas. För att garantera en kausal regulator måste nk_s och $nk_t \geq 0$.

Matlabkoden, för att initiera skattningen av regulatorparametrarna, finns i lista 5.1.

```
%***** Initialization of regulator estimation *****

Ar  = [1 -1];                          % pre-specified pole in R
nr  = 1;                               % numbers of estimated R parameters
ns  = 2;                               % numbers of estimated S parameters
nsk = 0;                               % delay in S polynomial
nt  = 2;                               % numbers of estimated T parameters
ntk = 0;                               % delay in T polynomial

nnreg  = [0 nr+ns+nt 1];

lamr   = 1.00;                         % forgetting factor.
thr0   = 1*ones(nr+ns+nt,1);           % initial value on theta
Pr     = 1000*eye(nr+ns+nt);           % vector and P matrix.

phiufd  = zeros(1,1+nr);               % phi vector for ufd
phiyf   = zeros(1,ns+nsk);             % phi vector for yf,
phiucf  = zeros(1,nt+ntk);             % phi vector for ucf
```

Lista 5.1 Matlabkod för initialisering av skattning av regulatorparametrar

Skattningsalgoritmen som används är RARX, som skattar en modell av typen

$$y(t) = \frac{b_0 + b_1 q{-}1 + \ldots}{1 + a_1 q^{-1} + \ldots}\, q^{-n_k}\, u(t)$$

För att kunna skatta parametrarna i regulatorn, måste dess struktur översättas för att passa RARX. För att göra detta sättes antalet skattade A parametrar till noll. Antalet B parametrar är lika med det totala antalet parametrar i regulatorn. Tidsfördröjningen n_k måste vara ≥ 1. Denna information finns lagrad i variabeln nnreg.

Då man skattar processen kan man utnyttja att skattningsalgoritmerna uppdaterar regressorvektorn, se kapitel 4. Uppdateringen görs med $u(t)$ och $y(t)$. φ vektorn skall i detta fallet ej uppdateras så här, utan uppdateringen får ske utanför skattningsfunktionerna. Regressorvektorn består av `phiufd`, `phiyf` och `phiucf`. Genom att dela upp regressorn så här, underlättas uppdateringen av regressorn. Initialt sätts alla element till noll.

Filtrering

Initieringen av de filter som finns i systemet har följande Matlab kod

```
%***** Filters *****

Af    = conv(Amd,Aod);              % Filter for estimation
ufdst = [];                         % empty state vectors for default
yfst  = [];                         % value.
ucfst = [];
ymst  = [];
```

Lista 5.2 Matlabkod för initialisering av filter som finns i systemet

$u(t)$, $y(t)$ Och $u_c(t)$ skall filtreras med

$$H_f(t) = \frac{\hat{B}^* q^{-d_0}}{A_o^* A_m^*}$$

Variabeln `Af` innehåller karakteristiska polynomet för detta filter. Variablerna som sätts till tomma matriser, är tillståndsvektorer till förekommande filter. Om denna tillståndsvektor är tom, skapar filterfunktionen `FILT` en vektor där alla element är noll.

Initialisering av regulatorn

Det först som händer i beräkningsloopen, är beräkning av styrsignalen. För att initiera R, S och T polynomen utförs följande kommando.

```
%****  Initialization regulator ****

[r,s,t]    = th2rst(thr0',nr,Ar,ns,nsk,nt,ntk,'fs') % unpack R,S,T
regst      = [];
reglim     = proclim;
```

Lista 5.3 Matlabkod för initialisering av regulatorparametrarna

Funktionen `TH2RST` skapar regulatorpolynomen utifrån strukturen (5.19). Inparametern `'fs'` medför att polynomen skapas i framåtskift. Styrsignalen beräknas med funktionen `RST2`, som har *antiwindup* verkan. `REGLIM` anger mättningsgränserna.

Beräkningssekvensen

Beräkningssekvensen byggs upp enligt samma princip som i den indirekta *STR:n*. Det som skiljer sig för den direkta, är framför allt uppdateringen av regressorvektorn för skattningen av regulatorparametrarna. Matlabkoden återfinns i lista 5.4. Då koden till stor del är den samma som för den indirekta, kommenteras endast vissa avsnitt.

Beräkningen av $y_m(t)$, som behövs för att skatta regulatorparametrarna, görs i blocket *Desired response*. `Bmp` är modellens B polynom. Detta polynom bestäms så att stationära förstärkningen blir 1. `ym` beräknas genom att filtrera u_c med B_m/A_m.

Skapandet av regressorvektorn för regulatorparametrarnas skattning, utförs i blocket *Build regressor vector*. `SHIFT` utför följande operation:

$$\begin{aligned} X &= [x(1)\, x(2) \ldots x(n)] \\ X &= shift(X, x(0)) \\ X &= [x(0)\, x(1) \ldots x(n-1)] \end{aligned}$$

På detta sätt uppdateras `PHIUFD`, `PHIYF` och `PHIUFC`. Innehållet i dessa variabler är:

```
PHIUFD = [ u(k)  .. u(k-nr)]
PHIYF  = [ y(k)  .. y(k-nks-ns)]
PHIUCF = [ uc(k) .. uc(k-nkt-nt)]
```

De element som skall plockas ut bestäms entydigt av `nr,..,nkt`. Att skapa `PHI` sker då så enkelt som i sista steget i blocket.

```
%**** Simulation loop ****
%*************************


for k=1:length(time),


%**** Control signal *****

  [u(k),regst]     = rst2(r,s,t,[],reglim,y(k),uc(k),regst);

%**** Estimation of process ****

  [thp(k,:),Pp,rlsst] = rls(y(k),u(k),[na nb nk],lamp,thp0,Pp,rlsst);
  thp0                = thp(k,:)';
  Ppd(k,:)            = diag(Pp)';
  [A,B]               = th2pol(thp(k,:),[na nb nk],'bs');


%**** Desired response ****

  Bmp             = B*sum(Amd)/sum(B);
  [ym(k) ymst]    = filt(uc(k),ymst,Bmp,Amd);


%**** Filtering ****

  [ufd(k),ufdst] = filt(u(k),ufdst,conv(B,Ar),Af);
  [yf(k),yfst]   = filt(y(k),yfst,B,Af);
  [ucf(k),ucfst] = filt(uc(k),ucfst,B,Af);

%**** Build regressor vector ****

  phiufd  = shift(phiufd,ufd(k));
  phiyf   = shift(phiyf,yf(k));
  phiucf  = shift(phiucf,ucf(k));
```

```
  phi     = [phiufd(2:nr+1) phiyf(1+nsk:ns+nsk) -phiucf(1+ntk:nt+ntk)]';

%**** Estimation of regulator polynomial ****

  epsi             = y(k) - ym(k) -ufd(k);
  [thr(k,:),yh,Pr] = rarx([epsi 0],nnreg,'ff',lamr,thr0,Pr,phi);
  thr0             = thr(k,:)';
  Prd(k,:)         = diag(Pr)';
  [r,s,t]          = th2rst(thr(k,:),nr,Ar,ns,nsk,nt,ntk,'fs');


%**** Update process ****

  up         = sat(u(k),[-25 25]) + d(k);
  [y(k+1),x] = process(x,up,procpar);
  y(k+1)     = y(k+1)+n(k);

end


%*******************************
%**** End simulation loop *****
```

Lista 5.4 Beräkningssekvensen för den direkta *STR:n*.

Kommentarer till den fullständiga koden

I Appendix B finns den fullständiga koden för den direkta regulatorn. Plottrutinerna som ingår, är de som förklarades i kapitel 4. Dessa kommenteras därför ej här.

5.3 Process och design

Samma process som simulerades i kapitel 4 används här.

$$G_p(s) = \frac{B(s)}{A(s)} = K\frac{\omega_0^2(Ts-1)}{(sT+1)(s^2+2\zeta\omega_0 s+\omega_0^2)}e^{-\tau s} \tag{5.21}$$

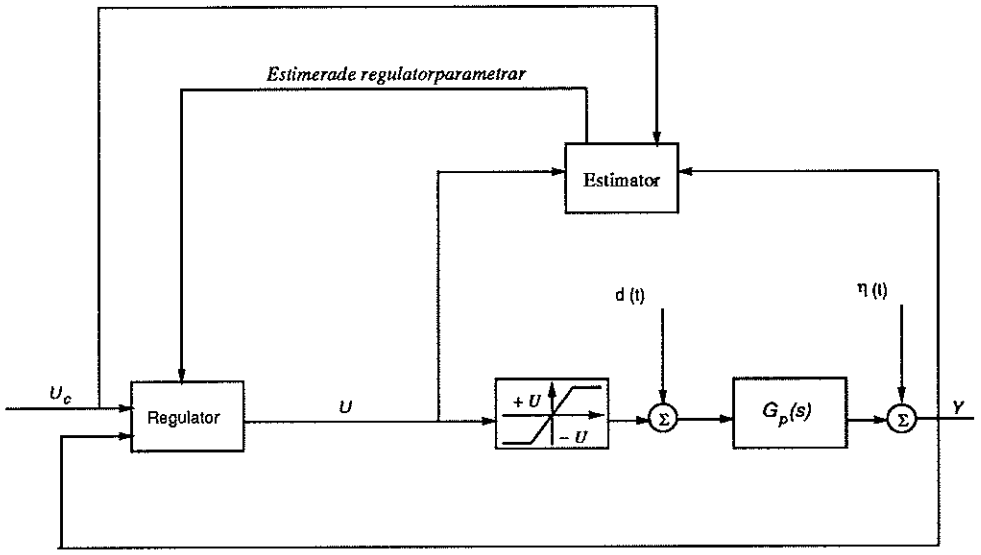
med följande värden på parametrarna

$$\begin{aligned}
K &= K_n + \Delta K & K_n &= 1 & -0.5 &\le \Delta K \le 2\\
\tau &= \tau_n + \Delta\tau & \tau_n &= 0.2 & 0.0 &\le \Delta\tau \le 0.2\\
T &= T_n + \Delta T & T_n &= 1 & -0.2 &\le \Delta T \le 0.5\\
\omega_0 &= \omega_{0n} + \Delta\omega_0 & \omega_{0n} &= 15 & -5 &\le \Delta\omega_0 \le 1\\
\zeta &= 1
\end{aligned}$$

Regulator och reglermål

Processen skall regleras med hjälp av den direkta självinställande regulatorn. Designmetoden som används är polplacering. Syftet med regleringen är att det slutna systemet skall följa referenssignalen utan översläng, med en stigtid runt $4T$. Regulatorn skall också klara av att systemet påverkas av laststörningar och mätbrus.

Systemet har ett blockschema som i figur 5.1.



Figur 5.1 Blockschema för systemet.

Val av samplingsintervall

För val av samplingsintervall, hänvisas till kapitel 4. Samplingsintervallet sätts till:

$$h = 0.2 \Leftrightarrow f_s = 5Hz$$

Slutet system och regulatordesign

Designen bygger på de antagande om processen, som gjordes i kapitel 4. Det innebär att ordningen på processen antas vara

$$degA = 5$$

Det önskade slutna systemet ordning sätts till

$$degA_m = 5$$

Observerapolynomet skall väljas som

$$degA_o \geq 2degA - degA_m - degB^+ + \ l - 1$$
$$degA_o = 5$$

där $degB^+ = 0$ och en $l = 1$ (antalet integratorer).

För att kunna lösa (5.1) krävs att

$$degR_1 = 4$$
$$degS < 6$$

Gradtalet på T polynomet sätts till $degA_o$.

Val av gradtal enligt ovan, ger följande utseende på regulatorparametrarna.

$$\begin{aligned} R^*(q^{-1}) &= (1+q^{-1})(1+r_1q^{-1}+r_2q^{-2}+r_3q^{-3}+r_4q^{-4}) \\ S^*(q^{-1}) &= s_0+s_1q^{-1}+s_2q^{-2}+s_3q^{-3}+s_4q^{-4}+s_5q^{-5} \\ T^*(q^{-1}) &= t_0+t_1q^{-1}+t_2q^{-2}+t_3q^{-3}+t_4q^{-4}+t_5q^{-5} \end{aligned} \tag{5.22}$$

Två olika strukturer på modell- och observerarpolynom testades i kapitel 4. Ett av alternativen var att polynomen hade *deadbeat* -karaktär, se (4.5). Med detta val fick man ringning på styrsignalen. Det andra alternativet som provades vid simuleringarna, var att välja A_m och A_o som:

$$\begin{aligned} A_m(s) &= (s + \frac{1}{T_m})(s + \frac{10}{T_m})^2(s + \frac{100}{T_m})^2 \\ A_o(s) &= (s + \frac{1}{T_o})(s + \frac{10}{T_o})^2(s + \frac{100}{T_o})^2 \end{aligned} \tag{5.23}$$

I detta exempel väljs A_m och A_o på denna form.

Skattad process

Den process som skattas har följande överföringsfunktion

$$\hat{H}(q^{-1}) = \frac{b_0 + b_1 q^{-1} + b_2 q^{-2} + b_3 q^{-3}}{1 + a_1 q^{-1}} q^{-2}$$

5.4 Experiment

Simuleringarna i detta avsnitt visar att hybriden mellan den direkta och indirekta regulatorn fungerar utmärkt. Experimenten kommer också visa *Toolboxens* styrka.

De gradtal på regulatorparametrarna som räknades fram i avsnit 5.3 var

$$\begin{aligned} DegR &= 5 \\ DegS &= 5 \\ DegT &= 5 \end{aligned}$$

Skall man utnyttja dessa gradtal, innebär det att 5 R, 6 S och 6 T parametrar kan skattas. Matriserna man jobbar med har då storleken 17×17. Att skriva en skattningsalgoritm i Simnon för det antalet parametrar, är ingen rolig uppgift. Här hade man från början fått försöka dra ner på antalet parametrar, och istället laborera med vilka parametrar som behövs skattas.

I Matlab har man däremot inga problem med detta, koden blir lika svår eller lätt oavsett antal parametrar som skall skattas. Självklart strävar man efter att få en så enkel regulatorstruktur som möjligt, men man har möjlighet att på ett enkelt sätt se hur antalet skattade parametrar påverkar regleringen.

Simulering av det nominella fallet

Inledningsvis simuleras systemet då processparametrarna har nominella värden. Signalerna som påverkar systemet genereras som:

$$u_c(t) = \begin{cases} 10 & \text{för } 10 \le t \le 20 \\ -5 & \text{för } 20 \le t \le 30 \\ 10 & \text{för } 30 \le t \le 40 \\ -5 & \text{för } 40 \le t \le 50 \\ 10 & \text{för } 50 \le t \le 60 \end{cases}$$

$$d(t) = \begin{cases} 0 & \text{för } 0 \le t \le 30 \\ -5 & \text{för } \quad t \ge 30 \end{cases}$$

$$\sigma = 0 \qquad \text{för alla} \quad t$$

Uppstartskedet är precis samma som i kapitel 4.

Strukturen på A_m och A_o väljs som i (5.23), där T_m och T_o sätts till 1. En A parameter och fyra B parametrar, skattas i processen. Glömskefaktorn sätts till ett.

Experiment 1 I första simuleringen sätts

$$\begin{aligned} nr &= 5 \\ ns &= 6 \\ nks &= 0 \\ nt &= 6 \\ nkt &= 0 \\ \lambda_{reg} &= 0.995 \end{aligned}$$

Detta ger regulator strukturen:

$$\begin{aligned} R^*(q^{-1}) &= 1 + r_1 q^{-1} + r_2 q^{-2} + r_3 q^{-3} + r_4 q^{-4} + r_5 q^{-5} \\ S^*(q^{-1}) &= s_0 + s_1 q^{-1} + s_2 q^{-2} + s_3 q^{-3} + s_4 q^{-4} + s_5 q^{-5} \\ T^*(q^{-1}) &= t_0 + t_1 q^{-1} + t_2 q^{-2} + t_3 q^{-3} + t_4 q^{-4} + t_5 q^{-5} \end{aligned}$$

Observera att ingen integratorpol fördefinieras i R polynomet. Resultatet av simuleringen visas i figur 5.2 och 5.3 .Om man studerar figur 5.3, ser det ut som om regulatorn innehåller en integratorpol. Vid undersökning av det skattade R polynomet, visade det sig att en pol skattades i 1.0026. För att avgöra om detta var en ren tillfällighet, gjordes en simulering med laststörningen $d(t() = -10$, $\Delta\tau = 0.2$. Med dessa ändringar, skattades den reella polen i R polynomet till 0.94. Detta visar att vill man säkert ha integratorverkan, måste denna fördefinieras.

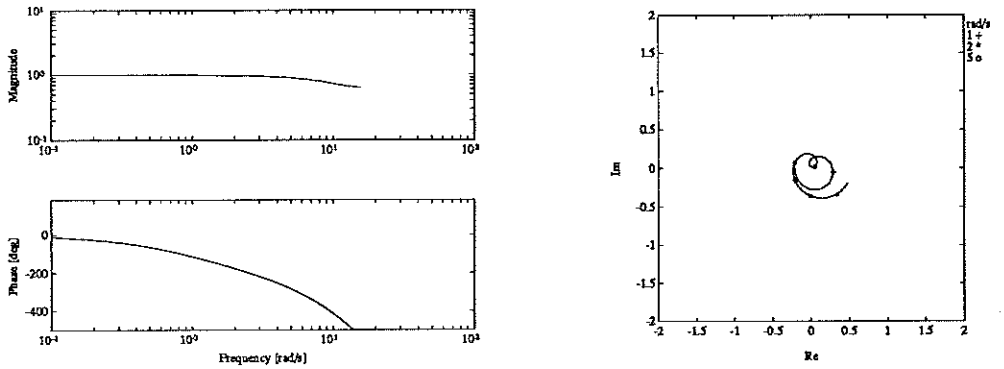
Att skatta många parametrar, medför numeriska problem. För att få ner antalet parametrar undersöktes slutvärdet på skattningarna.

```
R  = [1.0000   -1.3346    1.0065   -0.9055    0.4254   -0.1953]
Pr = [   0      0.0026    0.0542    0.2597    0.2769    0.0525]

S  = [0.7782   -1.0879    0.6009   -0.1840   -0.0023   -0.0072]
Ps = [0.0734    0.4696    0.5481    0.1640    0.0257    0.0025]

T  = [0.8523   -0.9036    0.1776   -0.0533    0.1065   -0.0781]
Pd = [0.0026    0.0208    0.0443    0.0504    0.0303    0.0044]
```

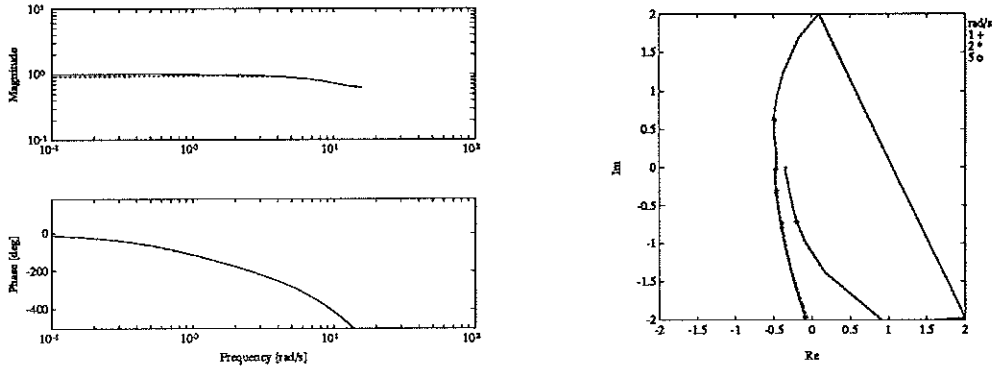
`Pr`, `Ps` och `Pt` är diagonalelementen i P matrisen. En jämförelse mellan skattningen och osäkerheten ger att s_3, s_4 och s_5 kan sättas till noll. Samma gäller för T polynomet.

Experiment 2 I detta experimentet simuleras ett system, med en regulatorstruktur:

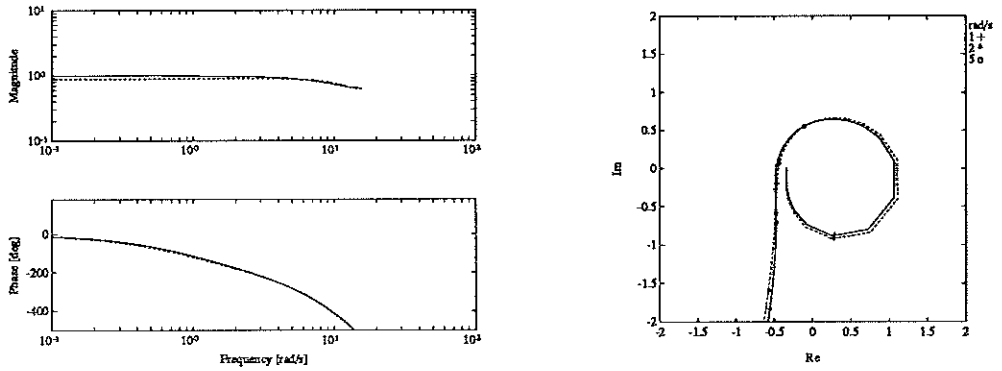
$$\begin{aligned} R^*(q^{-1}) &= (1 - q^{-1})\,(1 + r_1 q^{-2} + r_2 q^{-3} + r_3 q^{-4} + r_4 q^{-5}) \\ S^*(q^{-1}) &= s_0 + s_1 q^{-1} + s_2 q^{-2} \\ T^*(q^{-1}) &= t_0 + t_1 q^{-1} + t_2 q^{-2} \end{aligned}$$



Figur 5.2 a) $t = 24sek$



Figur 5.2 b) $t = 36sek$

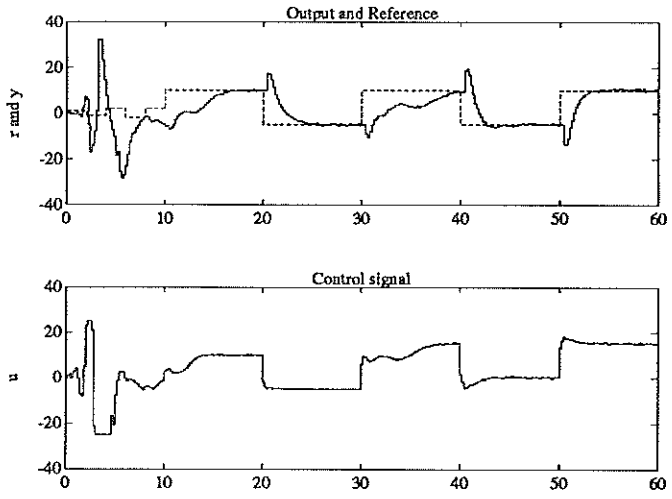


Figur 5.2 c) $t = 52sek$

Figur 5.2 *Experiment 1* : Processen har nominella parametervärden. 5 R, 6 S och 6 T parametrar skattas för regulatorn. Ingen integratorpol är fördefinierad i R polynomet.Vänstra kolumnen visar bodediagram för verklig och skattad process vid olika tidpunkter för simuleringen. Högra kolumnen visar nyquistdiagram för verklig och skattad kretsöverföringsfunktion vid samma tidpunkter. I båda fallen gäller att verklig = (—) och skattad = (···).

Processen har fortfarande nominella värden. Resultaten för simuleringen visas i figur 5.4 och 5.5. Inga principiella skillnader kan upptäckas, mellan experiment ett och två, i nyquist- och bodediagrammen (se fig 5.2 och 5.4). I figur 5.6 ser man att systemet som har en inlagd integratorpol, hämtar sig snabbare från laststörningen som börjar verka vid $t = 30$ s.

Slutsatsen man drar ifrån de två inledande experimenten, är att det räcker att skatta 4 R, 3 S och 3 T parametrar.



Figur 5.3 *Experiment 1* : Processen simuleras med nominella parametervärden. 5 R, 6 S och 6 T parametrar skattas i regulatorpolynomet. Ingen integratorpol fördefinieras i R polynomet. Den övre figuren visar systemets utsignal och börvärde, den undre visar styrsignalen.

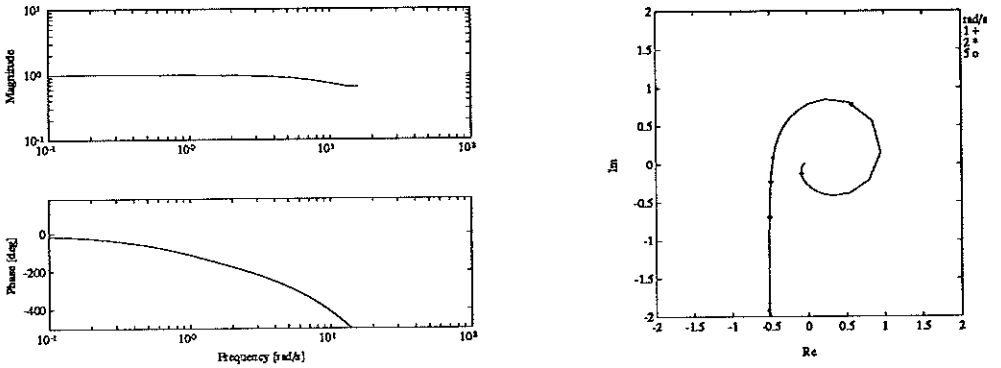
Icke nominella fallet

Dessa experiment skall visa hur regultorn klarar av att reglera processen, då denna aviker ifrån det nominella fallet. Ändringarna som görs i processen är:

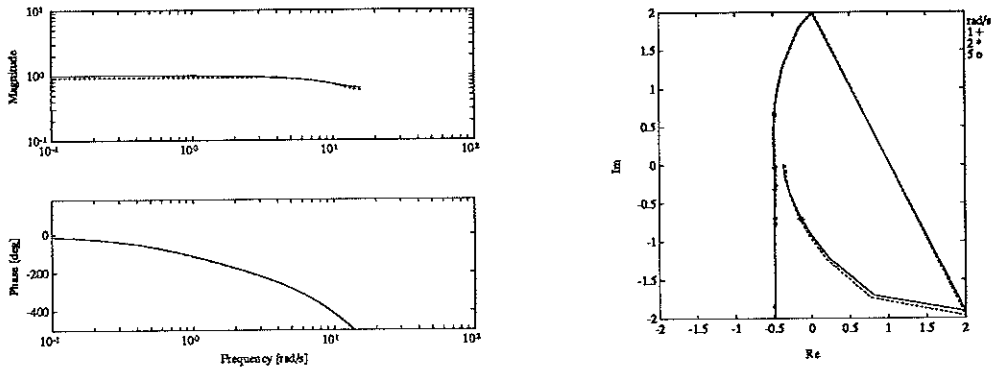
$$\Delta K = -0.4$$
$$\Delta \tau = 0.2$$
$$\Delta T = 0.5$$
$$\Delta \omega_0 = 1$$

Signalerna som påverkar systemet, är de samma som i det nominella fallet.

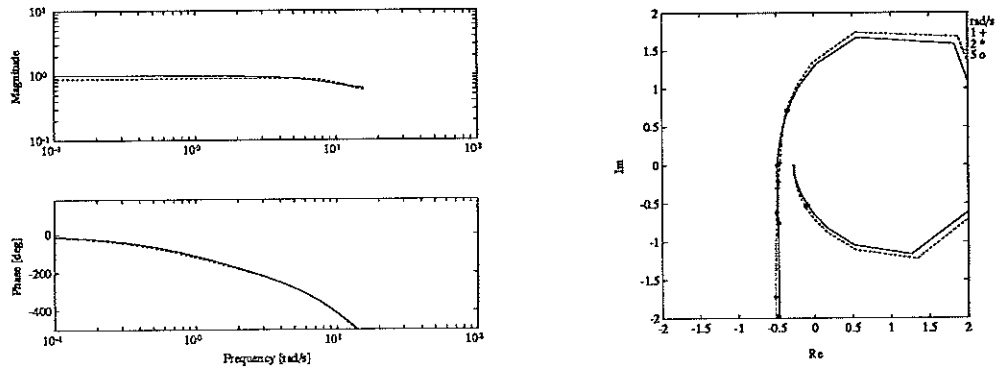
Experiment 3 Bodediagrammet i figur 5.6 visar att man inte klarar av att skatta processen i lågfrekvensområdet. Detta beror på att stationära förstärkningen är låg. Då det finns en begränsning på ± 25 i processen medför det att styrsignalen kommer att bottna. Nyquistdigrammet skiljer sig nu också ifrån det nominella fallet. Kretsöverföringsfunktionen omsluter -1, vilket innebär att man har fått en instabil regulator.



Figur 5.4 a) $t = 24sek$

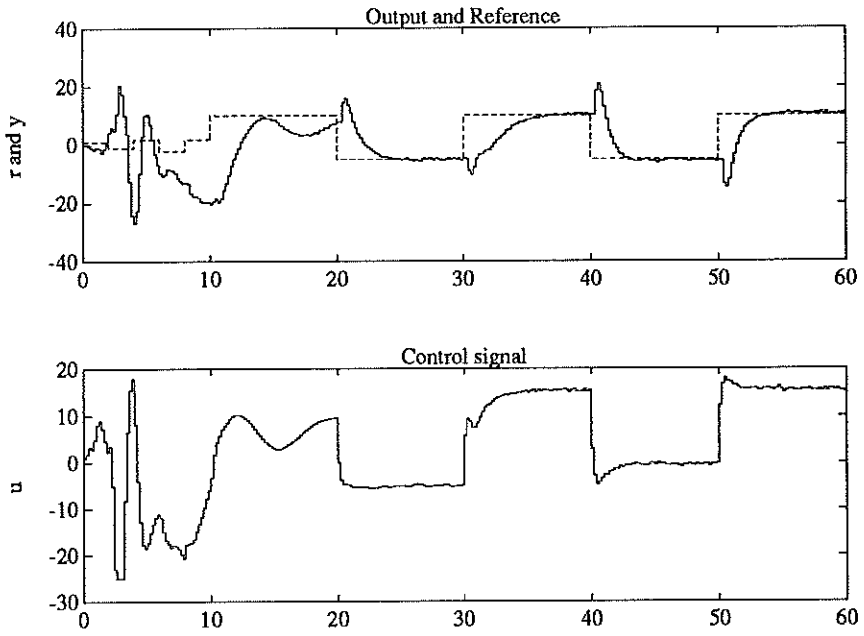


Figur 5.4 b) $t = 36sek$

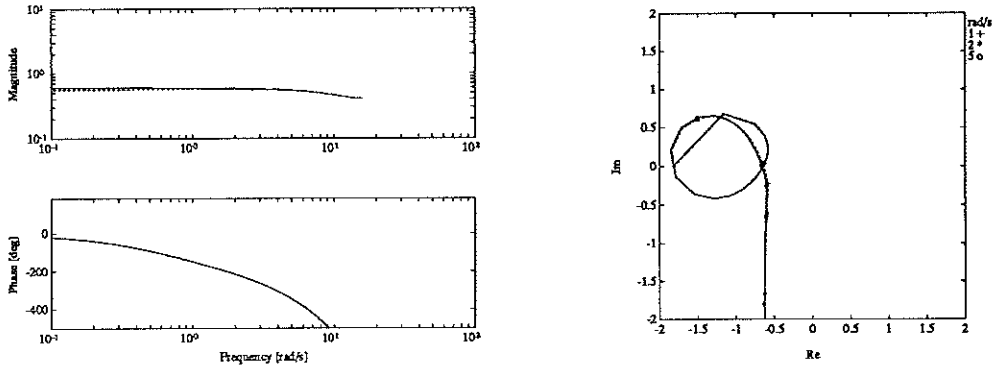


Figur 5.4 c) $t = 52sek$

Figur 5.4 *Experiment 2* : Simulering då processen har nominella paramatervärden. 4 R, 3 S och 3 T parametrar skattas för regulatorn. Integratorpol är fördefinierad i R polynomet. Vänstra kolumnen visar bodediagram för verklig och skattad process vid olika tidpunkter för simuleringen. Högra kolumnen visar nyquistdiagram för verklig och skattad kretsöverföringsfunktion vid samma tidpunkter. I båda fallen gäller att verklig = (—) och skattad = (···).



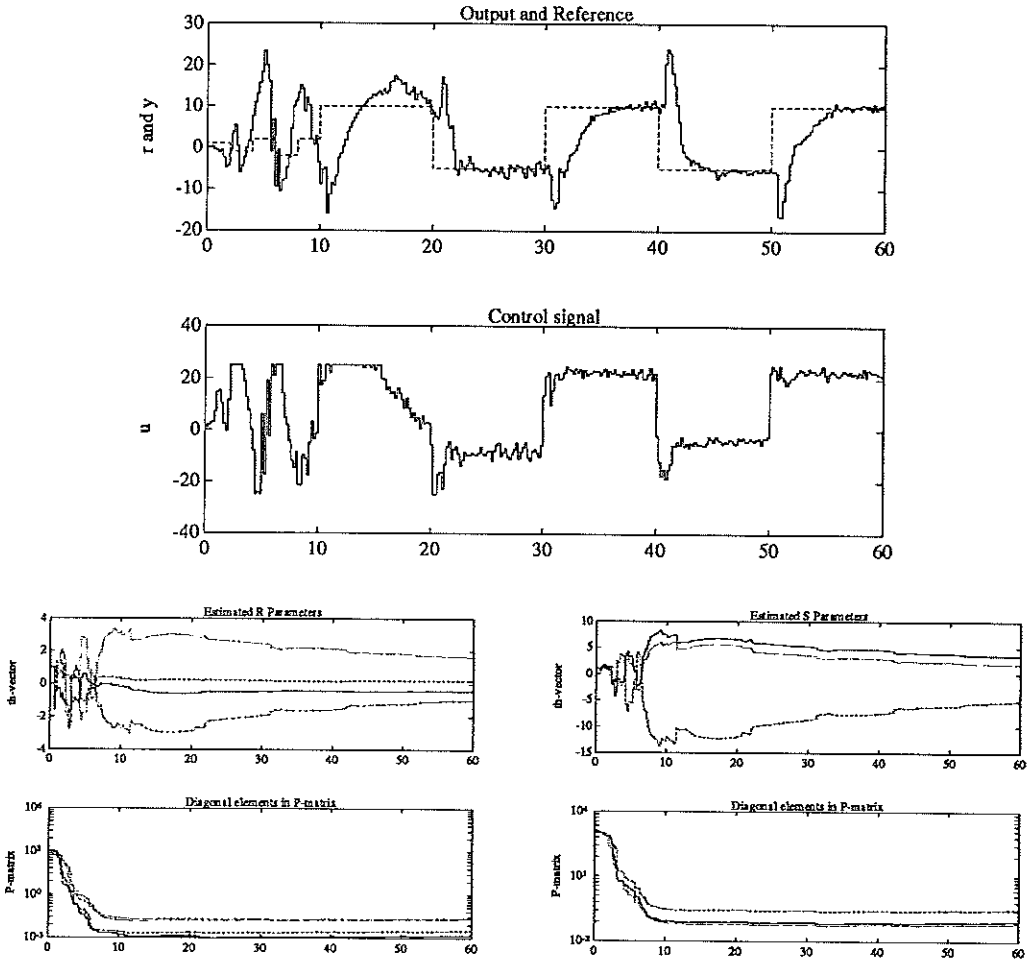
Figur 5.5 *Experiment 2* : Processen simuleras med nominella parametervärden. 4 R, 3 S och 3 T parametrar skattas i regulatorpolynomet. Integratorpolen är fördefinierad i R polynomet. Den övre figuren visar systemets utsignal och börvärde, den undre visar styrsignalen.



Figur 5.6 *Experiment 3* : Simulering av systemet då processens parametrar har ändrat sig med: $\Delta K = -0.4$, $\Delta\tau = 0.2$, $\Delta T = 0.5$ och $\Delta\omega_0 = 1$. 4 R, 3 S och 3 T parametrar skattas för regulatorn. Integratorpolen i R polynomet är fördefinierad. Bodediagrammet visar verklig och skattad process. Nyquistdiagrammet visar verklig och skattad kretsöverföring. I båda diagrammen gäller att verklig = (—) och skattad = (···).

Figur 5.7 visar dels utsignal och styrsignal, dels parameterskattningen för R och S polynomen. Figuren visar tydligt att regulatorn nu är instabil. Visserligen klarar man fortfarande av att följa referenssignalen, men det sker på bekostnad av en ryckigare styrsignal. R och S parametrarna driver under hela simuleringstiden. En anledning till detta kan vara att man skattar för få R och S parametrar.

Experiment 4 För att få bukt med de drivande regultorparametrarna, gjordes en simulering där $4R$, S och T parametrar skattades. Bodediagrammen



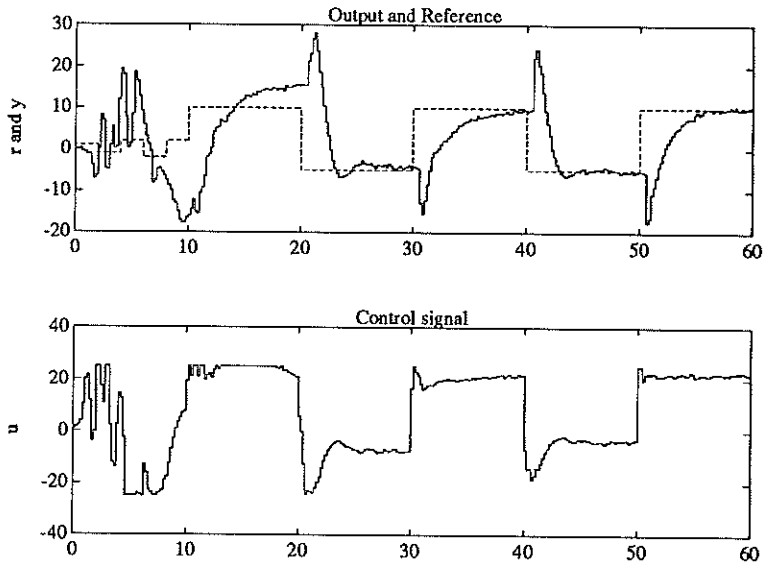
Figur 5.7 *Experiment 3* : Simulering av systemet då processparametrarna har ändrat sig med: $\Delta K = -0.4$, $\Delta\tau = 0.2$, $\Delta T = 0.5$ och $\Delta\omega_0 = 1.4$ R, 3 S och 3 T parametrar skattas. Integratorpolen i R polynomet är fördefinierad. Den övre figuren visar systemets utsignal och börvärde samt styrsignal. De undre figurerna visar skattningen av R och S parametrarna

och nyquistdiagrammen uppvisar inga principiella skillnader. Skattningen i det lågfrekventa området, avviker fortfarande från processens värde. Kretsöverföringsfunktionen omsluter fortfarande -1. Den stora skillnaden är att regulatorparametrarna ej längre driver.

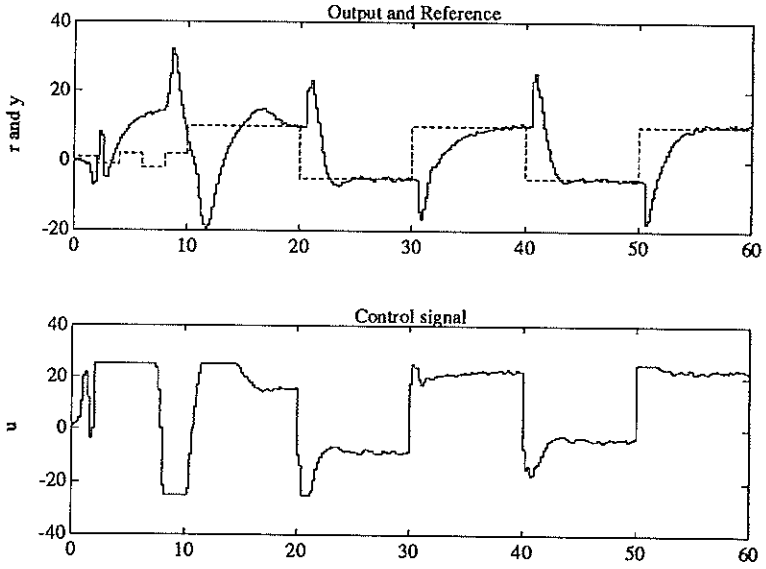
Styrsignalen och utsignalen visas i figur 5.8. En jämförelse mellan figur 5.8 och 5.7, visar att styrsignalen har fått ett lugnare beteende.

Experiment 5 Fortfarande har processen svårt att följa referenssignalen för $10 \leq t \leq 20$, tiden fram till 10 är en initieringsfas. Om man studerar styrsignalen i experiment 4, kan man se att denna har lite av *deadbeat* karaktär över sig, framför allt under tiden $10 \leq t \leq 20$. Det skulle vara önskvärt att få lite dynamik på styrsignalen under denna tid. Regulatorn som använts under alla experimenten, har varit med *antiwindup*. Denna har haft en *dead-beat-observerare* .

I figur 5.9, visas styr- och utsignalen då regulatorn har samma *antiwindup* polynom som används vid designen av det slutna systemet. Ingen skillnad kan



Figur 5.8 *experiment 4* : Simulering av systemet då processparametrarna har ändrat sig med: $\Delta K = -0.4$, $\Delta\tau = 0.2$, $\Delta T = 0.5$ och $\Delta\omega_0 = 1.$ 4 R, 4 S och 4 T parametrar skattas. Integratorpolen i R polynomet är fördefinierad. Den övre figuren visar systemets utsignal och börvärde. Den undre visar styrsignalen.



Figur 5.9 *experiment 5* : . Simulering av systemet då processparametrarna har ändrat sig med: $\Delta K = -0.4$, $\Delta\tau = 0.2$, $\Delta T = 0.5$ och $\Delta\omega_0 = 1.$ 4 R, 4 S och 4 T parametrar skattas. Integratorpolen i R polynomet är fördefinierad. Regulatorns *antiwindup* -polynom är satt till:$A_{or} = (s+1)(s+10)^2(s+100)^2$ (kontinuerliga modell). Den övre figuren visar systemets utsignal och börvärde, den undre visar styrsignalen.

ses för tider ≥ 30 däremot är det stor skillnad före detta. Systemet hänger med mycket bättre nu, utan att man förlorat något i styrsigalsuppförande. Styrsignalen ligger fortfarande nära bottningsgränsen, men den har fått ett betydligt lugnare beteende framför allt för tider ≤ 30

5.5 Avslutning

Detta kapitel har visat hur en direkt självinställande regulator implementeras i Matlab. Regulatorn är inte en renodlad direkt *STR*, utan en hybrid mellan den direkta och indirekta. Denna typ av adaptiva regulatorer, har inte varit föremål för större undersökningar. Simuleringsresultaten visar dock att regulatorn fungerar.

Fördelarna med att simulera i Matlab har också framkommit. Genom att ändra endast ett fåtal parametrar, är det möjligt att skatta valfritt antal parametrar. Detta är möjligt tack vara Matlabs matrisoperationer. Detta ökar dels möjligheterna att simulera system med hög ordning dels undersöka hur systemet uppför sig relaterat till antal parametrar som skattas.

6. Slutsatser

Baserat på de resultat som framkommit, blir slutsatsen att Matlab lämpar sig mycket väl som miljö för simulering av adaptiva regulatorer. Dock finns vissa problem, bland annat beroende på den tillgängliga datastruktur som finns i Matlab. Begränsningar på datastrukturen leder till vissa problem med implementeringen av rekursiva beräkningsalgoritmer.

Ett annat problem som uppkommer är den metodik som skall användas för att definiera och koppla ihop systemet. Jämförs Matlab med exempelvis Simnon utifrån denna aspekt, går det att peka på ytterligare svagheter för Matlab som simuleringsmiljö för adaptiva regulatorer. I Simnon har man tillgång till alla de funktioner som krävs för att definiera och koppla ihop systemet. Dessutom finns där sorteringsrutiner, vilka automatiskt skapar det beräkningsschema som skall genomlöpas. För att ersätta dessa faciliteter har det varit tvunget att införa bl. a. begränsningar på processen som skall simuleras. Det har också varit nödvändigt att definiera ett schema, som anger i vilken ordning beräkningarna måste utföras så inga algebraiska loopar uppstår. Dessa brister i Matlab, sänker självklart användarvänligheten samt ökar programmeringsinsatsen.

Matlabs styrka ligger i matrisberäkningarna och den stora tillgången av funktioner för analys och syntes av reglersystem. De extremt goda möjligheterna till matrisberäkningar gör Matlab mycket lämplig som miljö att implementera de ekvationerna som beskriver de i systemet ingående delarna. Då funktionerna kan skivas generellt, blir det mycket enkelt för en användare att laborera med exemplevis ordningen på skattad process, gradtal för regulatorpolynomen etc. Att simulera system med hög ordning är inte heller det något problem. Detta är något som är mycket svårt med simuleringsverktyg som saknar möjlighet att utföra matrisberäkningar.

En annan ovärderlig tillgången i Matlab är den mängd funktioner som finns att tillgå för analys. I de simuleringsfiler som beskriver den indirekta och direkta STR:n används exempelvis funktioner som plottar bodediagrammet för simulerad och skattad process samt nyquistdiagrammet för verklig och skattad kretsöverföringsfunktion. Denna facilitet ökar möjligheterna för en användare att få insikt om och ökad förståelse för systemet som simuleras.

7. Referenser

[1] THE MATHWORKS (1990): *Pro-Matlab, User's Guide.*

[2] ÅSTRÖM, K. J. and WITTENMARK, B. (1989): *Adaptive Control*, Addison-Wesley, Reading, Mass.

[3] ÅSTRÖM, K. J. and WITTENMARK, B. (1990): *Computer Controlled Systems, Theory and Design, 2nd ed*, Prentice-Hall, Englewood Cliffs.

[4] SÖDERSTRÖM, T. and STOICA, P. (1989): *System Identification*, Prentice-Hall, Englewood Cliffs.

[5] GUSTAFSSON, K., LILJA, M. and LUNDH, M (1990): "A Collection of Matlab Routines for Control System Analysis and Synthesis," Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

[6] LJUNG, L (1991): *System Identification Toolbox, User's Guide.*

Appendix A - Matlabkod

RST 1

```
function [u,newstate] = rst1(R,S,T,y,uc,state)
%
% RST  Implementation of an RST regulator
%
%      U = RST1(R,S,T,Y,Uc,STATE)
%
%      Input  : R,S,T      : Regulator polynomial in forward shift
%
%               Y          : Process output
%
%               Uc         : Reference signal
%
%               STATE      : [u(k-1) u(k-2) ... y(k-1) ... uc(k-1) ...]
%                            If STATE=[], it's filed with zeros
%
%      Output : U          : Control Signal
%
%               NEWSTATE   : Updated state vector
%
%      The control signal is determined from the control law
%
%      U(k) = -r(2)u(k-1)- r(3)u(k-2) ... - s(1)y(k) - s(2)y(k-1) ...
%             +t(1)Uc(k) + t(2)Uc(k-1)
%
%
%      Restriction:    R must be monic
%                      degS and degT must be less or equal degR

%      Last edit 91-01-30

nR=length(R);nS=length(S);nT=length(T);
S = [zeros(1,nR-nS) S];                  % Transform to backward shift
T = [zeros(1,nR-nT) T];
if isempty(state)
    state=zeros(1,3*nR-3);
end

U     = [0 state(1:nR-1)];
Y     = [y state(nR:2*nR-2)];
Uc    = [ uc state(2*nR-1:3*nR-3)];

u        = -R*U' - S*Y' + T*Uc';

U(1) = u;
U(nR)=[];
Y(nR)=[];
Uc(nR)=[];

newstate = [U Y Uc];
```

RST 2

```
function [u,newstate]=rst2(R,S,T,Ao,lim,y,uc,state);
%
% RST2  Implementation of RST regulator with anti-wind-up
%
%       [U,NEWSTATE] = RST2(R,S,T,AO,LIM,Y,Uc,STATE)
%
%       Input  : R,S,T      : Regulator polynomial in forward shift
%
%                Ao         : Observer polynomial in forward shift
%                             for the regulator. Ao must be monic
%                             and degAo = degR .
%
%                LIM        : Saturation limits [umin umax]
%
%                Y          : Process output
%
%                Uc         : Reference signal
%
%                STATE      : Contains oldU, oldY, oldUc and oldV.
%                             The vector is formed as:
%                             [u(k-1) u(k-2)..y(k-1)..uc(k-1) ..v(k-1)..]
%
%       Output : U          : Control Signal
%
%                NEWSTATE   : Updated state vector
%
%       The control signal is determined from the control law
%
%                AoV = TUc - SY + (Ao - R)U
%                  U = sat(V)

%       Last edit 91-01-30

nR=length(R);nS=length(S);nT=length(T);nA=length(Ao);
S = [zeros(1,nR-nS) S];          % Transform to backward shift
T = [zeros(1,nR-nT) T];
if isempty(state)
   state=zeros(1,4*nR-4);
end
if isempty(Ao)
  Ao = [1 zeros(1,nR-1)];         % Dead Beat Antiwindup default
end

U        = [0  state(1:nR-1)];
Y        = [y  state(nR:2*nR-2)];
Uc       = [uc state(2*nR-1:3*nR-3)];
V        = [0  state(3*nR-2:4*nR-4)];
v        = -Ao*V' + T*Uc' - S*Y' + Ao*U' - R*U';
u        = min(max(v,lim(1)),lim(2));

U(1) = u;
V(1) = v;
U(nR)=[];
Y(nR)=[];
Uc(nR)=[];
V(nR)=[];

newstate = [U Y Uc V];
```

RLS

```
function [theta,P,nstate]=rls(y,u,nn,lambda,theta0,P0,state)
%RLS  Recursive Least Square with forgetting factor
%
%     [THETA,P,NSTATE] = RLS(Y,U,NN,LAMBDA,THETA0,P0,STATE)
%
%     Inputs:  Y,U        : Input and output signals
%
%              NN         : Vector [nA nB nK]
%
%              LAMBDA     : Forgetting factor
%
%              THETA0     : Old estimated parameters.
%                           Input format is column vector
%
%              STATE      : Old state vector that contains
%                           [-y(k-1)..-y(k-na) u(k-1)..u(k-nk-nb)]
%                           Regressor vector is formed by STATE
%                           If STATE=[] => it will be filled with
%                           zeros
%
%              P0         : Old P-Matrix
%
%
%     Outputs: THETA      : New estimated parameters
%                           Output format is row vector
%
%              P          : New P-Matrix
%
%              NSTATE     : New STATE vector
%                           [-y(k)..-y(k-na+1) u(k)..u(k-nb-nk+1)]

%     Last edit 90-02-1


if isempty(state)
   state=zeros(1,sum(nn)-1);
end

%---- Transform regression vector -----

i   = [1:nn(1) (nn(1)+nn(3)):(sum(nn)-1)];
phi = state(:,i)';
I   = eye(nn(1)+nn(2));

%---- RLS algorithm ----

K               = P0*phi/(lambda + phi'*P0*phi);  % Weight factors
resid           = y - phi'*theta0;                % Residual
theta           = (theta0 + K*resid)';            % New theta
P               = (I - K*phi')*P0/lambda;         % Updating P-Matrix


% Create new regression vector and transform theta
%
% phi= [-y(k-1). . .u(k-nk-nB+1)]'

nstate=[-y state(1:nn(1)-1) u state(nn(1)+1:sum(nn)-2)];
```

Filt

```
function [y,newstate]=filt(u,state,b,a)
%FILT  Filters the signal u(n) with the filter described
%      by vectors a and b.
%
%      [Y,NEWSTATE] = FILT(U,STATE,B,A)
%
%      The filter structure is described by the difference
%      equation:
%
%      y(n) = b(1)u(n) + b(2)u(n-1) + ... + b(nb)u(n-nb-1)
%
%                      - a(2)y(n-1) - ... - a(na)y(n-na-1)
%
%
%      Inputs   U        : Input signal u(n)
%
%               STATE    : A vector that contains old U and Y.
%                          The vector is formed as:
%                          [u(n-1)...u(n-nb-1) -y(n-1)...-y(n-na-1)]
%                          If state is empty it will be filed with
%                          zeros
%
%               A,B      : Vectors that describe the filter
%                          A = [a(1) a(2) ... a(na)]
%                          A must be monic.
%
%      Outputs  Y        : Output from the filter y(n)
%
%               NEWSTATE : Updated state vector, that contains
%                          U(n) and Y(n).

%      Last edit date  91-01-26

if isempty(state)
  state = zeros(1,length(b)-1+length(a)-1);
end
state=[u state];

y = [b a(2:length(a))]*state';

ystate=[-y state(length(b)+1:length(state))];

newstate = [state(1:length(b)-1) ystate(1:length(a)-1)];
```

Process

```
function [y,x]=process(x,u,procpar)
%
% PROCESS  Calculate output and updating process state of the system:
%
%          x[k+1] = Phi x[k] + Gam u[k]
%          y[k+1] = C x[k+1]
%
%          No direct term is allowed.
%
%
%
%          [Y,X] = PROCESS(X,U,PROCPAR)
%
%          Inputs:  X       : old process state
%                   U       : control signal
%                   PROCPAR : Vector  [PHI GAM C^T]
%
%          Outputs: Y,X     : Output and updated state

%          Last edit 90-01-16

[m,nn] = size(procpar);
Phi    = procpar(:,1:m);
Gam    = procpar(:,m+1);
C      = procpar(:,nn)';

x = Phi*x +Gam*u;      % X(k+1)
y = C*x;               % Y(k+1)
```

Sat

```
function [usat]=sat(u,ulim)
%SAT       Saturation block
%
%          [USAT] = SAT(U,ULIM)
%
%          Inputs:  U      : Input Signal
%                   ULIM   : Saturation limits [umin umax]
%
%          Outputs: USAT   : Output between umin and umax

% Last Edit Date 91-02-01

usat = min(max(u,ulim(1)),ulim(2));
```

Shift

```
function newX=shift(oldX,x)
% SHIFT Makes right shift in vector
%
%       NEWX = SHIFT(OLDX,X)
%
%       Inputs:  OLDX   : Vector that contains old x value
%                         e.g OLDX = [x(1) x(2) ... x(nx)]
%                X      : new value x(0)
%
%       Outputs: NEWX   : Updated X vector
%                         e.g NEWX = [x(0) x(1) ... x(nx-1)]

%Last Edit Date 91-01-16

if ~isempty(oldX)
 n=length(oldX);
 newX=[x oldX(1:n-1)];
end
```

th2pol

```
function [A,B]=th2pol(theta,nn,sh)
%TH2POL    Unpack the theta vector and form
%          A and B polynom
%
%          [A,B] =TH2POL(THETA,NN,SH)
%
%          Inputs:  THETA     : Vector that contains estiamted
%                               A and B polynomial.
%
%                   NN        : The orders and delay of the model
%                               NN = [na nb nk]
%
%                   SH = 'fs' : A and B polynmial is returned in
%                               forward shift.
%
%                   SH = 'bs' : A and B polynomial is returned in
%                               backward shift.
%
%          Outputs: A,B       : Process polynomial
%
%          See also RLS and SQRLS

%          Last Edit Date 91-01-28


if sh == 'fs'
   A=[1 theta(1:nn(1)) zeros(1,nn(3)+nn(2)-1-nn(1))];
   B=[theta(nn(1)+1:nn(1)+nn(2)) zeros(1,nn(1)+1-nn(2)-nn(3))];
else
   A=[1 theta(1:nn(1))];
   B=[zeros(1,nn(3)) theta(nn(1)+1:nn(1)+nn(2))];
end
```

th2rst

```
function [R,S,T]=th2rst(th,nr,ar,ns,nsk,nt,ntk,sh)
%TH2RST  Unpack theta vector and form R, S and T
%        polynomial in forward or backward shift to the
%        control law
%
%        Ru(k) = -Sy(k-ntk) + Tuc(n-nkt)
%
%        Assumptions: TH = [r(1).. s(1).. t(1)]
%                     R  = [1 r(1) .. r(nr)]
%                     S  = [s(1) ...  s(ns)]
%                     T  = [t(1) ...  t(nt)]
%
%        [R,S,T] = TH2RST(THETA,NR,NS,NSK,NT,NTK,SH)
%
%        Inputs:  THETA     : Parameter vector
%                 NR,NS,NT  : Number of estimated parameters
%                 NSK,NTK   : Delay
%                 AR        : pre-specified pole in R polynomial.
%                             e.g integrator Ar = [1 -1]
%                 SH        : string SH='fs' formed in forward shift
%                                    'bs'   "     " backward  "
%
%        Outputs: R,S,T     : Polynomial to an RST regulator
%
%

%        Last Edit Date 91-02-06

[m,n]=size(th);
nar = length(ar)-1;
if sh =='bs',
   R = conv([1 th(1:m,1:nr)],ar);
   S = [zeros(m,nsk) th(nr+1:nr+ns)];
   T = [zeros(m,ntk) th(nr+ns+1:nr+ns+nt)];
else
   R = conv([1 th(1:nr) zeros(m,max(nsk,ntk))],ar);
   S = [th(nr+1:nr+ns) zeros(m,nr+1+nar-ns-nsk+ntk-nsk)];
                      % zeros(m,ntk-nsk)];
   T = [th(nr+ns+1:nr+ns+nt) zeros(m,nr+1+nar-nt-ntk+nsk-ntk)];
                           %zeros(m,nsk-ntk)];
end
%R = theta(1:m,1:degr+1);
%S = [theta(1:m,degr+2:degr+degs+2)]./R(:,1);
%T = [theta(1:m,degr+degs+3:degr+degs+degt+3)]./R(1);
%R = R./R(1);
```

Plotyu

```
function plotyu(yin,uin,rin,time,tstart,tend)

clg
h=time(2)-time(1);
if nargin < 5
   nstart=1;
else
   nstart = find((time>=tstart) & (time <= (tstart+h)));
   nstart=nstart(1);
end

if nargin < 6
   nend = length(time);
else
   nend = find((time >= tend) & (time <= (tend+h)));
   nend=nend(1);
end


y = yin(nstart:nend);
r = rin(nstart:nend);
u = uin(nstart:nend);

n  = length(y);
nn = 2*n;
yy = zeros(nn-1,1);
rr = yy;
uu = yy;
tt = yy;

yy(1:2:nn)   = y;
yy(2:2:nn-1) = y(1:n-1);
rr(1:2:nn)   = r;
rr(2:2:nn-1) = r(1:n-1);
uu(1:2:nn)   = u;
uu(2:2:nn-1) = u(1:n-1);
tt(1:2:nn)   = time(nstart:nend);
tt(2:2:nn-1) = time(nstart+1:nend);
subplot(211),plot(tt,yy,tt,rr);
title('Output and Reference')
ylabel('r and y')
subplot(212),plot(tt,uu);
title('Control signal')
ylabel('u')
```

plotth

```
function plotth(th,Pd,mark,time,tstart,tend,tit)

clg
h=time(2)-time(1);

if nargin < 5
   nstart=1;
elseif isempty(tstart)
   nstart=1;
else
   nstart = find((time>=tstart) & (time <= (tstart+h)));
   nstart = nstart(1);
end

if nargin < 6
   nend = length(time);
elseif isempty(tend)
   nend = length(time);
else
   nend = find((time >= tend) & (time <= (tend+h)));
   nend = nend(1);
end

if nargin < 7
  tit=[];
end

tht=th;
st=[];
[n,m]=size(th);
k    = (nend -nstart)/2+nstart;
dk   = h*(nend-nstart)/80;
if mark
   for i = 1:m
     mm = 1:i;
     if i < 10,
        st = [st; ' ' num2str(i)];
     else
        st = [st; num2str(i)];
     end
     x(i) = i*dk+k*h;
     y(i) = round(x(i)/h);
   end
end


th = th(nstart:nend,:);
Pd = Pd(nstart:nend,:);
[n,m]=size(th);
nn = 2*n;
th2 = zeros(nn-1,m);
Pd2 = th2;
tt = zeros(nn-1,1);

th2(1:2:nn,:)   = th;
th2(2:2:nn-1,:) = th(1:n-1,:);
Pd2(1:2:nn,:)   = Pd;
Pd2(2:2:nn-1,:) = Pd(1:n-1,:);
tt(1:2:nn)   = time(nstart:nend);
tt(2:2:nn-1) = time(nstart+1:nend);
subplot(211),plot(tt,th2);
```

```
tit=['Estimated ',tit,' Parameters'];
title(tit)
ylabel('th-vector')
text(x,diag(tht(y,:)),st)
subplot(212),semilogy(tt,Pd2);
title('Diagonal elements in P-matrix')
ylabel('P-matrix')
```

Appendix B

1. Indirekta STR:n

Detta är den fullständiga Matlabkoden för exemplet som finns i kapitel 4. Denna fil finns i toolboxen under namnet `WORKBENCH1.M`

```
frbox                   % initialize plot routines
ppbox
estim = 1;              % 1 = estimation on,
                        % 0 = estimation off
adapt = 1;              % 1 = parameter update on
                        % 0 =     "       "    off
%***** Process initialization *****

Kn   = 1;
taun = 0.2;
Tn   = 1;
won  = 15;

dK   = 0;       % [-0.5 2]
dtau = 0;       % [0.0 0.2]
dT   = 0;       % [-0.2 0.5]
dwo  = 0;       % [-5 1]

K    = Kn + dK;
tau  = taun + dtau;
T    = Tn + dT;
wo   = won + dwo;
z    = 1;


%***** make process model *****

h             = 0.2;                          % sampling interval
Apc           = conv([1 1/T],[1 2*z*wo wo^2]);% process denominator
Bpc           = - K*wo^2*[1 -1/T];            % process nominator
[Bpd,Apd]     = sample(Bpc,Apc,tau,h);        % sampled process
[Phi,Gam,C,D] = tf2ss(Bpd,Apd);               % transform to -
                                              % state space representation
procpar       = [Phi Gam C'];
proclim       = [-25 25];                     % saturation
x             = zeros(length(Phi),1);         % initial state
y             = C*x;                          % initial output

%***** Design the Closed Loop *****
Tm  = 1;                                      % closed loop pole
Amc = poly(1/Tm*[-1 -10 -10 -100 -100]);
Amd = polyc2d(Amc,h);                         % sample Amc
To  = 1;                                      % Observer pole
Aoc = poly(1/To*[-1 -10 -10 -100 -100]);
Aod = polyc2d(Aoc,h);                         % sample observer
Ar  = [1 -1];                                 % integrator

%*****  Initialization of estimator *****
```

```
if estim
 na       = 1;                                  % number of estimated.
 nb       = 4;                                  % a and b parameters.
 nk       = 2;                                  % numbers of time delay.
 nn       = [na nb nk];
 lam      = 1.00;                               % forgetting factor.
 th0      = 0.01*ones(na+nb,1);                 % initial value on theta
 P0       = 1000*eye(na+nb);                    % vector and P matrix.
 rlsstate = [];                                 % default value.
 [bf,af]  = butter(2,[0.01,0.5]);               % bandpassfilter
end


%***** Signals *****

time = 0:h:40;                                  % Simulation time.
rlev=  [1 -1 2 -2 2  10 -5 10 -5];              % Reference signal.
rch  = [0  2 4  6 8  10 20 30 40];
%rlev=  [ 10 -5 10];                            % Reference signal.
%rch  = [0 10 20 ];
dlev=-5;                                        % Load disturbance.
dch=30;
nvar=0.2*h;                                     % Measure Noise.
nch=0;

trldn=yusignals(time,rlev,rch,0,0,dlev,dch,nvar,nch);% Make ref,load and
                                                % noise signals.
uc = trldn(:,2);                                % Reference signal
d  = trldn(:,4);                                % Load disturbance.
n  = trldn(:,5);                                % Measure Noise.

%***** Initialize regulator *****
if adapt
  [A,B]   = th2pol(th0',nn,'fs');       % unpack th to A and B polynomial
  [r,s,t] = rstd(1,B,A,1,Amd,Aod,Ar);   % design regulator parameters
else
  [r,s,t] = rstd(1,Bpd,Apd,1,Amd,Aod,Ar);
  r = ones(length(time),1)*r;
  s = ones(length(time),1)*s;
  t = ones(length(time),1)*t;
end
ulim     = [-25 25];
regstate = [];
%***** speed up routine *****

y  = [y;zeros(length(time),1)];                 % Fill vectors
u  = zeros(length(time),1);                     % with zeros
if estim
  th = zeros(length(time),na+nb);
  Pd = zeros(length(time),na+nb);
end
if adapt
  r = [r;zeros(length(time),length(r))];
  s = [s;zeros(length(time),length(s))];
  t = [t;zeros(length(time),length(t))];
end;


%***** The Simulation Loop *****

for k=1:length(time)
  [u(k),regstate] = rst2(r(k,:),s(k,:),t(k,:),[],ulim,y(k),uc(k),regstate);
  if estim,
```

```
    [yf(k),yfstate] = filt(y(k),yfstate,bf,af);
    [uf(k),ufstate] = filt(u(k),ufstate,bf,af);
    [th(k,:),P0,rlsstate] = rls(yf(k),uf(k),nn,lam,th0,P0,rlsstate);
    th0 = th(k,:)';
    Pd(k,:) = diag(P0)';
  end

  if adapt,
    [A,B]                          = th2pol(th(k,:),nn,'fs');
    [r(k+1,:),s(k+1,:),t(k+1,:)]  = rstd(1,B,A,1,Amd,Aod,Ar);
    if rem(k,20)==0,
      k
      [A,B]=th2pol(th(k,:),nn,'fs');
      fr1=frd(Bpd,Apd,h,-1,[],80);
      fr2=frd(B,A,h,-1,[],80);
      fr3 = frd(s(k+1,:),r(k+1,:),h,-1,[],80);
      fr4 = frd(t(k+1,:),r(k+1,:),h,-1,[],80);
      clg
      bopl(fr1,fr2,[-1 2 -1 1 -500 180]);
      pause(1);
      nypl(fmul(fr1,fr3),fmul(fr2,fr3),'125',[-2 2 -2 2]);pause(0.5)
    end
  end

  up                               = sat(u(k),ulim) + d(k);
  [y(k+1),x]                       = process(x,up,procpar);
  y(k+1)                           = y(k+1) + n(k);

end

%**** Plot routines *****

plotyu(y,u,uc,time);
pause
plotth(th,Pd,1,time,[],[],'Process');
```

2. Direkat STR:n

Detta är den fullständiga Matlabkoden för exemplet som finns i kapitel 5. Denna fil finns i toolboxen under namnet `WORKBENCH2.M`

```
%WORKBENCH for DIRECT ADAPTIVE CONTROL
%
%CASE: NO ZEROS IS CANCELED
%
%This shows how an direct adaptive regulator can be implemented in Matlab.
%For further information see  chapter 5 in rapport:
%ADAPTIVE TOOLBOX IN MATLAB

%Last Edit Date 91-02-10

frbox
ppbox

%***** Process initialization *****

Kn   = 1;
taun = 0.2;
Tn   = 1;
won  = 15;
```

```
dK   = 0;       % [-0.5 2]
dtau = 0;       % [0.0 0.2]
dT   = 0;       % [-0.2 0.5]
dwo  = 0;       % [-5 1]

K    = Kn + dK;
tau  = taun + dtau;
T    = Tn + dT;
wo   = won + dwo;
z    = 1;


%***** make process model *****

h              = 0.2;                                  % sampling interval
Apc            = conv([1 1/T],[1 2*z*wo wo^2]);        % process denominator
Bpc            = - K*wo^2*[1 -1/T];                    % process nominator
[Bpd,Apd]      = sample(Bpc,Apc,tau,h);                % sampled process
[Phi,Gam,C,D] = tf2ss(Bpd,Apd);                        % transform to -
                                                       % state space representation
procpar        = [Phi Gam C'];
proclim        = [-25 25];                             % saturation
x              = zeros(length(Phi),1);                 % initial state
y              = C*x;                                  % initial output


%***** Design the Closed Loop *****

Tm  = 1;                                               % closed loop pole
Amc = poly(1/Tm*[-1 -10 -10 -100 -100]);
Amd = polyc2d(Amc,h);                                  % sample Amc
To  = 1;                                               % Observer pole
Aoc = poly(1/To*[-1 -10 -10 -100 -100]);
Aod = polyc2d(Aoc,h);                                  % sample observer


%***** Signals *****

time = 0:h:60;                                         % Simulation time.
rlev=  [1 -1 2 -2 2  10 -5 10 -5 10 -5 10 -5];         % Reference signal.
rch  = [0  2 4  6 8  10 20 30 40 50 60 70 80];
dlev=-5;                                               % Load disturbance.
dch=30;
nvar=0.2*h;                                            %  Measure Noise.
nch=0;

trldn=yusignals(time,rlev,rch,0,0,dlev,dch,nvar,nch);% Make ref,load and
                                                       % noise signals.
uc = trldn(:,2);                                       % Reference signal
d  = trldn(:,4);                                       % Load disturbance.
n  = trldn(:,5);                                       % Measure Noise.


%*****  Initialization of regulator estimation *****

Ar  = [1 -1];                          % pre-specified pole in R
nr  = 4;                               % numbers of estimated R parameters
ns  = 4;                               % numbers of estimated S parameters
nsk = 0;                               % delay in S polynomial
nt  = 4;                               % numbers of estimated T parameters
ntk = 0;                               % delay in T polynomial
```

```
nnreg  = [0 nr+ns+nt 1];

lamr   = 0.995                          % forgetting factor.
thr0   = 1*ones(nr+ns+nt,1);            % initial value on theta
Pr     = 1000*eye(nr+ns+nt);            % vector and P matrix.

phiufd  = zeros(1,1+nr);                % phi vector for uf
phiyf   = zeros(1,ns+nsk);              % phi vector for yf,
phiucf  = zeros(1,nt+ntk);              % phi vector for ucf


%*****  Initialization of process estimation *****

na    = 1;                              % numbers of estimated A parameters
nb    = 4;                              % numbers of estimated B parameters
nk    = 2;                              % delay in process
lamp  = 1.00;                           % forgetting factor.
thp0  = 0.001*ones(na+nb,1);;           % initial value on theta
Pp    = 100*eye(na+nb);;                % vector and P matrix.
rlsst = [];


%***** Filters *****

Af    = conv(Amd,Aod);                  % Filter for estimation
yfst  = [];                             % empty state vectors for default
ufdst = [];
ucfst = [];
ymst  = [];


%****  Initialization regulator ****

[r,s,t]    = th2rst(thr0',nr,Ar,ns,nsk,nt,ntk,'fs') % unpack R,S,T
regst      = [];
reglim     = proclim;


%**** Simulation loop ****
%************************


for k=1:length(time),


%**** Control signal *****

  [u(k),regst]     = rst2(r,s,t,[],reglim,y(k),uc(k),regst);

%**** Estimation of process ****

  [thp(k,:),Pp,rlsst] = rls(y(k),u(k),[na nb nk],lamp,thp0,Pp,rlsst);
  thp0                = thp(k,:)';
  Ppd(k,:)            = diag(Pp)';
  [A,B]               = th2pol(thp(k,:),[na nb nk],'bs');


%**** Desired response ****

  Bmp            = B*sum(Amd)/sum(B);
  [ym(k) ymst]   = filt(uc(k),ymst,Bmp,Amd);
```

```
%**** Filtering ****

  [ufd(k),ufdst] = filt(u(k),ufdst,conv(B,Ar),Af);
  [yf(k),yfst]   = filt(y(k),yfst,B,Af);
  [ucf(k),ucfst] = filt(uc(k),ucfst,B,Af);

%**** Build regressor vector ****

  phiufd  = shift(phiufd,ufd(k));
  phiyf   = shift(phiyf,yf(k));
  phiucf  = shift(phiucf,ucf(k));

  phi     = [phiufd(2:nr+1) phiyf(1+nsk:ns+nsk) -phiucf(1+ntk:nt+ntk)]';


%**** Estimation of regulator polynomial ****

  epsi             = y(k) - ym(k) -ufd(k);
  [thr(k,:),yh,Pr] = rarx([epsi 0],nnreg,'ff',lamr,thr0,Pr,phi);
  thr0             = thr(k,:)';
  Prd(k,:)         = diag(Pr)';
  [r,s,t]          = th2rst(thr(k,:),nr,Ar,ns,nsk,nt,ntk,'fs');


%**** Update process ****

  up         = sat(u(k),[-25 25]) + d(k);
  [y(k+1),x] = process(x,up,procpar);
  y(k+1)     = y(k+1)+n(k);

%**** Plot routine ****

  if rem(k,20)==0,
     disp(sprintf('simulation time : %g' ,k*h))
     [Ae,Be] = th2pol(thp(k,:),[na nb nk],'fs');
     fr1     = frd(Bpd,Apd,h,-1,[],80);
     fr2     = frd(Be,Ae,h,-1,[],80);
     fr3     = frd(s,r,h,-1,[],80);
     clg
     bopl(fr1,fr2,[-1 2 -1 1 -500 180]);
     pause(1);
     nypl(fmul(fr1,fr3),fmul(fr2,fr3),'125',[-2 2 -2 2]);
     pause(0.5);
  end
end


%*******************************
%**** End simulation loop *****

%**** plot rutin *
plotyu(y,u,uc,time);
pause
plotth(thr(:,1:nr),Prd(:,1:nr),0,time,[],[],'R')
pause
plotth(thr(:,nr+1:nr+ns),Prd(:,nr+1:nr+ns),0,time,[],[],'S')
pause
plotth(thr(:,nr+ns+1:nr+ns+nt),Prd(:,nr+ns+1:nr+ns+nt),0,time,[],[],'T')
pause
plotth(thp,Ppd,0,time,[],[],'Process')
```