ISSN 0280-5316
ISRN LUTFD2/TFRT--5807--SE

Utvärdering av latens och jitter för LQG och PID

Haitham Zaben

Department of Automatic Control
Lund University
December 2007

Lund University
Department of Automatic Control
Box 118
SE-221 00 Lund Sweden

Document name
MASTER THESIS

Date of issue
December 2007

Document Number
ISRNLUTFD2/TFRT--5807--SE

Author(s)
Haitham Zaben

Supervisor
Anton Cervin, Automatic Control in Lund
Karl-Erik Årzén, Automatic Control in Lund (Examiner)

Sponsoring organization

Title and subtitle
Utvärdering av latens och jitter för LQG och PID. (Evaluation of Latency and Jitter for LQG and PID Controllers)

Abstract
Digital control theory normally assumes equidistant sampling intervals and a negligible or constant control delay from sampling to actuation, so called input/output latency. However this can seldom be achieved in practice. In the computer tasks interfere with each other through preemption and blocking when waiting for common resources. The execution time for each task may be data-dependant and therefore differ from one sample to another, which introduces a so called jitter in the latency. The latency and jitter introduced by the computer can lead to significant performance degradation, and may also lead to instability. This thesis discusses how computer-induced latency and jitter affects computer-controlled systems. Control systems, with various processes and control structure of either PID or LQG, are analyzed under various latency and jitter conditions. The systems are analyzed and simulated with the
Matlab-based tools Jitterbug and TrueTime. Jitterbug allows the computation of a cost function for linear systems under various timing conditions. With this tool, one can estimate how sensitive a control
system is to latency and jitter. With TrueTime the user can simulate a control system in a real-time kernel under various timing conditions. The results from the analysis and the simulations are presented and discussed.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
Swedeish

Number of pages
47

Recipient's notes

Security classification

Innehållsförteckning

1 Inledning

Dagens datorstyrda reglersystem är inbäddade i stora och växande grupper av produkter, allt från mindre underhållningsprodukter till stora flygplan. Produkter som bilar och flygplan är utrustade med avancerade datorstyrda reglersystem och har stora krav på tillförlitlig och säker drift. En gemensam egenskap för dessa system är att datorsystemen blir alltmer komplexa då funktionaliteten utökas. Ett datorstyrt system realiseras av ett antal samtidiga aktiviteter, som kan bero på varandra, och som måste uppfylla ett antal fördefinierade villkor. Därför är det ofta svårt att på förhand säga om en implementation av en design kommer att uppfylla dessa villkor.

I digital reglering antas vanligtvis att samplingsintervallet är konstant och att det råder en försumbar eller konstant fördröjning mellan sampling och utställning av kontrollsignal. Från tidpunkten då en mätsignal samplas till dess att den nya kontrollsignalen beräknats kommer det att vara ett en liten fördröjning som är konstant, och denna brukar kallas för insignal/utsignal fördröjning (latens), eller input/output latency (se [5]).

Detta är dock väldigt sällan verkligheten. Inom samma nod kommer olika tasks i konflikt med varandra genom preemption och blockering när de väntar på gemensamma resurser. Exekveringstiden för varje task kan även skilja sig från en sampling till en annan. Det gör att fördröjningen blir tidsmässigt slumpartad, och denna slumpmässiga fördröjning kallas *jitter* [5].

Fördröjningen och jittret som uppkommer p.g.a. det datorstyrda systemet kan leda till en betydande prestandaförsämring. För att få bra prestanda i system med begränsade datorresurser, måste begränsningarna tas med i beräkningarna när man designar sitt reglersystem. För att undersöka detta har institutionen för reglerteknik på Lunds universitet utvecklat två verktyg som analyserar och simulerar hur fördröjningar och jitter påverkar regulator prestandan, Jitterbug [7] och TrueTime [8]

2 Mål

Målet med arbetet är att undersöka hur känsliga olika regulatorer är för fördröjning och jitter. Med hjälp av Jitterbug ska en kostnadsfunktion beräknas vilken ger uppskattning av hur latensen påverkar reglersystemet. Reglersystemet ska även simuleras i TrueTime för att se hur det beter sig i en realtidskärna.

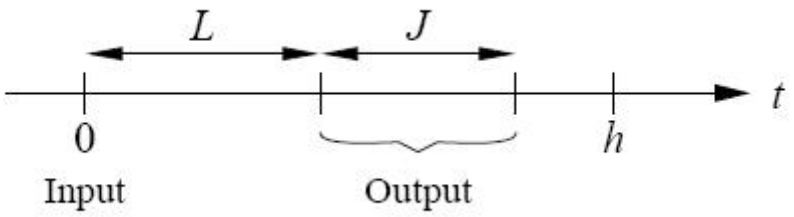
3 Bakgrund

I ett datorstyrt reglersystem så kommer det alltid att finnas en fördröjning från det ögonblick då mätsignalen, y(k), samplas till det ögonblick då kontrollsignalen, u(k), genereras. Det finns många anledningar till fördröjningen. I det enklaste fallet består det av exekveringstiden för maskinkoden. Men det kan också bero på preemptions från tasks med högre prioritet, avbrottsbehandling, och i det fallet då systemet är distribuerat över ett nätverk, kommunikationsfördröjningar. I det generella fallet är fördröjningen inte konstant. Det finns flera anledningar till detta: alternativa kodexekveringar, variationer i preemptionmönstret, etc. Input/output latensen har samma effekt på processen som en tidsfördröjning. Latensen i regler loopen orsakar ett fasskift som reducerar fasmarginalen, och i de fall som fördröjningarna är stora nog kan hela systemet bli instabilt.

3.1 Jitter i latensen

Input/output-latensen kan delas in i 2 delar: konstant latens, L, och jitter, J, enligt figur 1 [6]. En konstant latens går att kompensera för i reglerdesignen om man vet hur stor den är. När det finns jitter i latensen är det emellertid mycket svårare att kompensera för den.
Att det finns jitter innebär att latensen kan variera mellan olika samplingar. I detta arbete kommer följande latens fördelningar att studeras:

- Konstant: latensen är ett konstant värde mellan 0 och samplingstiden h varje sampling
- Varierande latens (α är mellan 0 och 1) för varje sampling:
    - Rektangelfördeladfördelning mellan 0 och αh
    - Normalfördelad runt αh/2
    - Hoppar mellan 0 och αh med 50 % sannolikhet



Figur 1 *Input/Output latensen kan delas in i en konstant del, L, och jitter, J.*

3.2 Tumregler för samplingstider

Valet av samplingstiden, h, för ett datorstyrt system är resultatet av en kompromiss mellan en rad olika faktorer. En av dessa faktorer är kostnad. Att öka samplingstiden gör att det finns mer tid som är tillgänglig för datorn att beräkna en kontrollsignal, och då kan en sämre och billigare dator användas. Idealet är att ha en dator som klarar av så små samplingstider som möjligt.

I många fall måste samplingstiden vara liten nog för att systemet ska förbli stabilt. Det finns en rad olika tumregler för hur samplingstider ska väljas för att ”försäkra sig” om att det datorstyrda systemet blir stabilt.

- 4 till 10 samplingar per stigtid T_r för det slutna systemet: $\frac{T_r}{h} = 4 - 10$ [3]
- Sampla utefter bandbreddsfrekvensen, ω_b: $0.2 \le \omega_b \cdot h \le 0.6$ [7]
- Minskningen av fasmarginalen när man diskretiserar en kontinuerlig process kan uppskattas som $\omega_c h/2$, där ω_c är skärfrekvensen. Man kan tillåta mellan 5° och 15° minskning av fasmarginalen vilket ger tumregeln: $0.15 \le \omega_c \cdot h \le 0.5$ [1]

Ett exempel som illustrerar de olika tumreglerna följer.

Antag att processen är en enkel andra ordningens process, $P = \frac{1}{(s+1)^2}$ och att

regulatorn är en PID regulator enligt, $C = K\left(1 + \frac{1}{s \cdot T_i} + \frac{s \cdot T_d}{\left(1 + \frac{s \cdot T_d}{N}\right)}\right)$.

PID parametrarna är valda som: K = 2.63, T_d = 0.908, T_i = 1.486, N = 10.
Då erhålles följande samplingsintervall:
h (T_r) = 0.172 – 0.430 sek
h (ω_b) = 0.157 – 0.470 sek
h (ω_c) = 0.073 – 0.243 sek

3.3 Jitterbug

Jitterbug ger användaren möjlighet att undersöka ett linjärt reglersystem under olika tids betingelser genom att beräkna ett kvadratiskt prestandakriterium. Reglersystemet beskrivs med hjälp av ett antal kontinuerliga och diskreta linjära system. En stokastisk tidsmodell med slumpmässiga tidsfördröjningar används för att beskriva exekveringen av systemet. Jitterbug kan användas för att undersöka aperiodiska, periodiska och jitterkompenserande regulatorer. Det ger snabbt och enkelt en uppskattning av hur känsligt ett reglersystem är för fördröjningar, jitter, förlorade samplingar, mm. Användning av Jitterbug förutsätter att man känner till samplingsperioderna och latensfördelningarna.

Jitterbug ger användaren möjligheten att bygga och analysera enkla tidsmodeller av datorstyrda system. Ett reglersystem byggs upp genom att sammanbinda ett antal kontinuerliga och diskreta system. För varje undersystem kan man ge valfria störnings- och kostnadsspecifikationer. I det enklaste fallet antas att de diskreta systemen uppdateras i ordning under varje tidsperiod. För varje diskret system kan en slumpmässig fördröjning (beskriven av en diskret fördelningsfunktion) specificeras, och denna måste ha fortlöpt innan nästa system kan uppdateras. Den totala kostnaden för systemet, summerad över alla undersystem, beräknas algebraiskt om systemet är periodiskt och iterativt om systemet är aperiodiskt.

Jitterbug kan endast hantera ett begränsat antal systemklasser. Reglersystemet är uppbyggt av linjära system, drivna av vitt brus. Prestandakriteriet som ska utvärderas är specificerat som en kvadratisk kostnadsfunktion. Tidsfördröjningarna i en tidsperiod antas vara oberoende av tidigare perioders tidsfördröjningar. De slumpmässiga fördröjningssannolikhetsfunktionerna är diskretiserade genom användandet av en för det totala systemet gemensam tidsfaktor, δ.

Även om en kvadratisk kostnadsfunktion knappast kan återskapa alla aspekter hos reglersystemet så kan den fortfarande vara värdefull när man snabbt vill jämföra olika regulatorimplementationer. Ett högre värde hos kostnadsfunktionen brukar bero på att det slutna systemet är mindre stabilt, och en oändlig kostnad innebär att systemet är instabilt. Kostnaden kan beräknas för många olika designparametrar och kan användas som hjälp i regulator designen.

Signal modell

Ett kontinuerligt system beskrivs av:

$$\frac{dx_c(t)}{dt} = Ax_c(t) + Bu(t) + \upsilon_c(t)$$
$$y(t) = Cx_c(t),$$

där A, B och C är konstanta matriser, och υ_c är vitt brus i kontinuerlig tid med kovariansen $R1_c$. Kostnaden för systemet än specificerad som:

$$J_c = \lim_{T\to\infty} \frac{1}{T} \int_0^T \begin{bmatrix} x_c(t) \\ u(t) \end{bmatrix}^T Q_c \begin{bmatrix} x_c(t) \\ u(t) \end{bmatrix} dt$$

Ett system i diskret tid beskrivs av:

$$x_d(t_{k+1}) = \Phi x_d(t_k) + \Gamma u(t_k) + \upsilon_d(t_k)$$
$$y(t_k) = Cx_d(t_k) + Du(t_k) + e_d(t_k)$$

Kovariansen för det diskreta vita bruset e_d och υ_d ges av:

$$R_d = E\begin{bmatrix} \upsilon_d(t_k) \\ e_d(t_{k)} \end{bmatrix} \cdot \begin{bmatrix} \upsilon_d(t_k) \\ e_d(t_{k)} \end{bmatrix}^T$$

Kostanden för systemet är specificerad som:

$$J_d = \lim_{T\to\infty} \frac{1}{T} \int_0^T \begin{bmatrix} x_d(t) \\ u(t) \end{bmatrix}^T Q_d \begin{bmatrix} x_d(t) \\ u(t) \end{bmatrix} dt$$

Det totala systemet formas genom att koppla ihop de olika kontinuerliga och diskreta processerna och den totala kostnaden blir då summan över alla undersystem:

$$J = \sum J_c + \sum J_d$$

Tidsmodell

Tidsmodellen består av ett antal tidsnoder. Varje nod kan associeras med en eller flera diskreta system i signalmodellen, som uppdateras när noden blir aktiv. Vid tiden noll aktiveras den första noden. Om den första noden deklareras som periodisk kommer exekveringen att starta om i denna nod varje h sekunder (motsvarar en tidsperiod) oavsett om den hunnit igenom alla noder.

Till varje nod kan man lägga till en tidsfördröjning, och denna måste ha fortlöpt innan nästa nod kan bli aktiv. Tidsfördröjningen kan också vara noll. Fördröjningen kan modellera olika sorters fördröjningar såsom beräkningsfördröjningar, mm. En fördröjning ges av den diskreta sannolikhetsfunktionen:

$$P_\tau = [P_\tau(0)\ P_\tau(1)\ P_\tau(2)\ \ldots\ ],$$

där $P_\tau(k)$ motsvarar en fördröjning på $k\delta$ sekunder. Tidsfaktorn δ är en konstant som definieras för hela modellen.

Beräkning av kostnaden

Beräkningen av kostnaden görs i tre steg: Först så samplas kostnadsfunktionerna, de kontinuerliga störningarna, och de kontinuerliga systemen med hjälp av tidsfaktorn för systemet. Sedan så formuleras det slutna systemet som ett hopplinjärt (jump linear) system. Slutligen beräknas den stationära variansen av alla tillstånd i systemet.

3.4 TrueTime

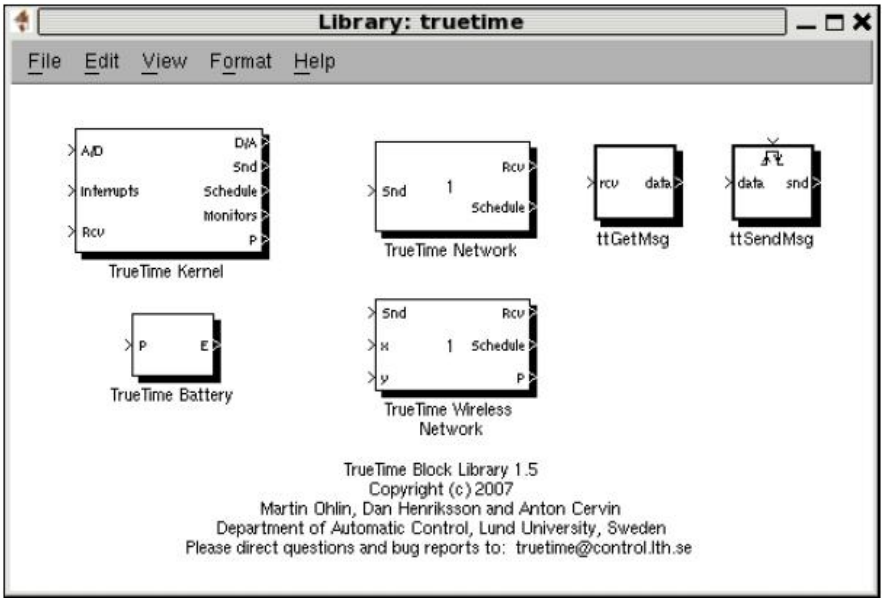
TrueTime är ett Matlab/Simulink-baserat verktyg som gör det möjligt att undersöka generella och detaljerade tidsmodeller av datorstyrda system. Verktyget innehåller två Simulink block: en Realtidskärna och ett Realtidsnätverk, se figur 2. I detta arbete har enbart kärnan använts.
Simulink-blocken är händelsedrivna och det behövs därför inte anges någon tidsfaktor för modellen. Exekveringen av en task kan simuleras i en godtycklig tidsskala genom att dela upp koden i olika segment. Typiskt så räcker det att dela upp koden i några olika segment, så som *Calculate* och *Update*, för att återge dess tidsenliga beteende. Alla blockens insignaler antas vara i diskret tid, förutom insignalen till A/D omvandlaren som kan vara i kontinuerlig tid. Alla utsignaler är i diskret tid.

Realtids kärnan

Detta block simulerar en dator med en enkel men flexibel realtids kärna, A/D och D/A omvandlare, och externa avbrottskanaler. Internt så innehåller kärnan många datastrukturer som återfinns i en verklig realtidskärna: en ready queue, en time queue, och datastrukturer för tasks, avbrottsbehandlare, monitorer och timers som skapats för simuleringen.Exekveringen av tasks och avbrottsbehandlingen är definierade i form av användarskriven kod. Denna kod kan skrivas i både C++ och som Matlab m-filer. Algoritmerna kan även definieras grafiskt med hjälp av Simulinks vanliga diskreta block.

Tasks är den huvudsakliga konstruktionen i TrueTimes simuleringsmiljö. Tasks används både för att simulera periodiska aktiviteter, såsom regulator och I/O tasks, och aperiodiska aktiviteter såsom kommunikationstasks och händelsedrivna regulatorer.
Varje task som skapas är definierad av ett antal attribut och av sin kod. Attributen innehåller ett namn, en release tid, en worst-case exekveringstid, relativa och absoluta deadlines, prioritet, och en period (om en task är periodisk).

Koden som är associerad med varje task exekveras av kärnan allteftersom simuleringen fortlöper. Koden delas vanligtvis in i flera segment. Koden kan samverka med andra tasks och med omgivningen i början av varje kodsegment. Denna exekveringsmodell gör det möjligt att modellera input/output fördröjningar, blockering när man ska ha tillgång till gemensamma resurser, mm. Den simulerade exekveringstiden för varje segment kan modelleras som konstant, slumpmässig, eller som databeroende. Exekveringen fortsätter i nästa segment när tiden för föregående segment har passerat.



Figur 2 *TrueTime block bibliotek*

4 Experiment

Två sorters regulator strukturer har undersökts: PI/PD/PID och LQG. Målet med experimenten är att undersöka hur känsliga dessa regulatorer är för input/output latens genom att beräkna en kostnadsfunktion i Jitterbug och sedan simulera i realtidsverktyget TrueTime. Undersökningen görs på följande processer:

$$P_1(s) = \frac{1}{(s+1)^n}, n = 3,4,5$$

$$P_2(s) = \frac{1}{(s+1)\cdot(1+\alpha s)^2}, \alpha = 0.01, 0.1, 0.5$$

$$P_3(s) = \frac{1}{(1+s)\cdot(1+\alpha s)\cdot(1+\alpha^2 s)\cdot(1+\alpha^3 s)}, \alpha = 0.1, 0.5, 0.9$$

$$P_4(s) = \frac{(1-\alpha s)}{(1+s)^3}, \alpha = 0.1, 0.3, 0.5$$

$$P_5(s) = \frac{1}{(s+1)\cdot\left(1+0.14ks+\left(\frac{ks}{10}\right)^2\right)}, k = 1,5,10$$

Tabell 1 *Processer som undersöks.*

PI, PD och PID regulatorerna representeras som:

$$PI = K\cdot\left(1+\frac{1}{s\cdot T_i}\right)$$

$$PD = K\cdot\left(1+\frac{s\cdot T_d}{\left(1+\frac{s\cdot T_d}{N}\right)}\right)$$

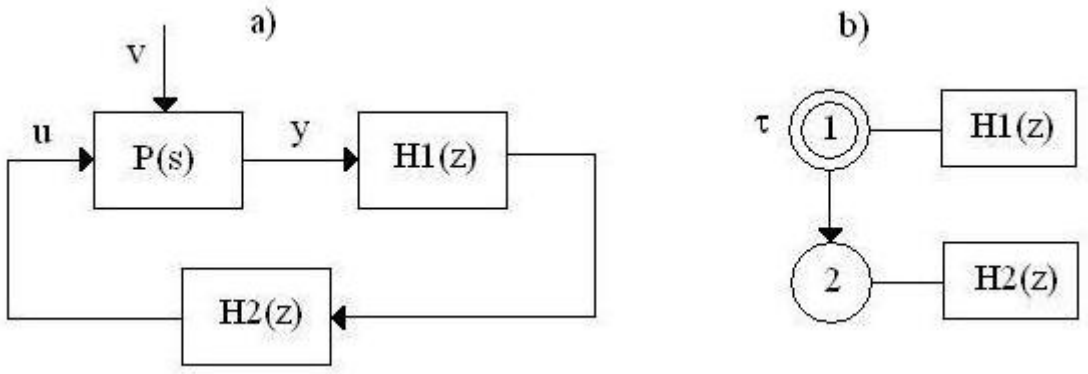
$$PID = K\left(1+\frac{1}{s\cdot T_i}+\frac{s\cdot T_d}{\left(1+\frac{s\cdot T_d}{N}\right)}\right)$$

Tabell 2 *PI, PD och PID regulatorerna*

Dessa processer är hämtade från en process batch, och återfinns i m-filen *processer*. Parametrarna K, T_i och T_d, som finns i m-filen *parametrar*, är optimerade till processerna i batchen på sådant sätt att regulatorerna motverkar last- och mätstörningar på ett tillfredställande sätt, och är någorlunda snabba. Båda dessa m-filer följer med processbatchen [9]. Eftersom en ”ren” derivatadel ger stora förstärkningar av mätbrus vid höga frekvenser så används ett filter som begränsar förstärkningen vid höga frekvenser till värdet N. Regulatorerna är i kontinuerlig tid och måste därför diskretiseras innan de kan användas i Jitterbug. Detta görs med Matlab kommandot *c2d* som tar regulatorn, samplingstiden och diskretiseringsmetod (Tustin i detta fall) som indata.

LQG regulatorn designas med hjälp av kommandot *lqgdesign* som ger en regulator i diskret tid. *Lqgdesign* tar följande som indata: P, Q, R1, R2, h och tau. Tau används för att skapa en latenskompenserande regulator, om tau sätts till noll är kompenseringen också noll.

Jitterbug modellen är enkel, se figur 3, där a) är signal modellen och b) är tidsmodellen. Processen är beskriven av det kontinuerliga systemet P(s), och regulatorn är beskriven av de två diskreta systemen H1(z) och H2(z) som representerar samplingen respektive regleralgoritmen. H1(z) har en tidsfördröjning τ som motsvarar input/output-latensen. De diskreta systemen exekveras enligt den periodiska tidsmodellen, där den extra ringen i den första cirkeln indikerar att reglersystemet är periodiskt.



Figur 3 *En enkel Jitterbug modell av ett datorstyrt regelsystem,*

P(s) är någon av processerna i tabell 1. Reglersystemet är periodiskt med intervallet h. Samplern är beskriven av det enkla diskreta systemet: H1(z) = 1. Regleralgoritmen, H2(z), är en av: LQG, PI, PD och PID. Fördröjningen i systemet är modellerat med τ. Det antas att τ aldrig överstiger h, vilket i så fall skulle innebära att nod 2 aldrig aktiveras.
Kostnadsfunktionen som ska utvärderas är specificerad som:

$$J = \lim_{T\to\infty} \frac{1}{T} \int_0^T \left(y^2(t) + \rho \cdot u^2(t) \right) dt \;\rightarrow\; Q = \begin{bmatrix} 1 & 0 \\ 0 & \rho \end{bmatrix}$$

4.1 Jämförelse mellan olika regulatorer

Experimentet ligger till grund för övriga Jitterbug experiment. I detta experiment skall kostnadsfunktionen beräknas och jämföras för alla processer när var och en av de 4 regulatorerna används. Tre olika samplingstider studeras vilka fås ur tumregeln: $0.2 \le \omega_b \cdot h \le 0.6$.

Brusspecifikationerna är:

- Processbrus: R1 = 1
- Mätbrus: R2 = 0.01

Experimentets utformning är enligt följande:

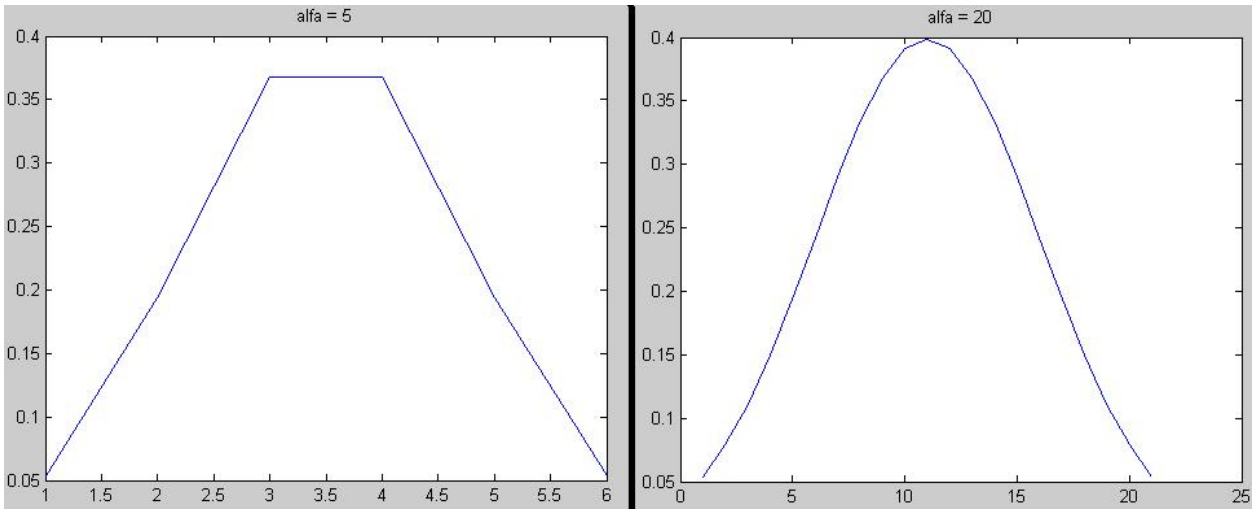
- Beräkna bandbredden ω_b

- Välj ut tre olika samplingstider (med hjälp av tumregeln) enligt:
  $h = \left[0.2/\omega_b \;\; 0.4/\omega_b \;\; 0.6/\omega_b\right]$ (1)
- Designa en regulator för var och en av dessa samplingstider
- Beräkna känslighets- och komplementära känslighetsfunktionen
- Utvärdera kostnaden då latensen är (α sveps mellan 0 och 1):
  - Konstant, $0 \rightarrow \alpha \cdot h$
  - Rektangelfördelad, $U(0, \alpha \cdot h)$
  - Normalfördelad, runt $\alpha \cdot h/2$
  - Ändpunktsfördelad, hoppar mellan 0 och $\alpha \cdot h$ med 50 % sannolikhet

Matlab kod

Tidsfaktorn är vald till: $\delta = h/20$, vilket innebär att α ovan kommer att vara en heltalssiffra mellan 0 och 20. Detta medför att den diskreta sannolikhetsfunktionen P_τ kommer att ha 21 värden och se ut på följande sätt för de olika latenserna:

- Konstant: en etta i kolumn nummer α och nollor i resten.
- Rektangel: en etta i alla kolumner från kolumn 0 till och med kolumn α, resten nollor.
- Normal: med hjälp av Matlab kommandot *normpdf* så kan man få en normalfördelad kurva för ett visst antal punkter, med medelvärdet och standardavvikelsen angivna av användaren, se figur 4. I detta fall är medelvärdet 0 och standardavvikelsen 1, och antalet punkter är lika med talet (α+1), enligt följande: p = normpdf (-2: 4/α: 2, 0, 1), $1 \le \alpha \le 20$, se figur 2. p kommer då att innehålla kurvans värde, och då blir $P_\tau(\alpha)$ = p(α). Resterande kolumner är 0. Observera P_τ = [1 0 … 0] då α = 0, och att kurvan är ”klippt” i båda ändarna.
- Ändpunkt: En etta i kolumn nummer 1 och i nummer α, resten nollor.
- Summan av $P_\tau(k)$ ska vara lika med 1 → $P_\tau = P_\tau$/summan(P_τ)



Figur 4 *Normalfördelning med normpdf för två olika α (alfa), den högra har 21 punkter medan den vänstra har 6 punkter.*

Först bestäms bandbredden för det återkopplade systemet med hjälp av Matlab kommandot *bandwidth*. Detta kräver att regulatorn är känd. LQG regulatorn skapas med kommandot

lqgdesign vilket kräver att man känner samplingstiden, men denna fås ju från tumregeln. För att lösa detta sätts LQG regulatorn först till C = 1. Sedan beräknas en bandbredd för det återkopplade systemet med denna regulator vilket då, med hjälp av tumregeln, ger ett samplingsintervall. Som samplingstid för *lqgdesign* används nu mittpunkten på det erhållna samplingsintervallet. Nu kan bandbredden beräknas, och därmed också samplingsintervallet.
Jitterbug initieras genom att man anropar *initjitterbug*, som tar δ och h som indata. Sedan lägger man till tidsnoderna med tillhörande P_τ med hjälp av kommandot *addtimingnode*. Sedan lägger man till processen som ett kontinuerligt system med *addcontsys*, som tar Q, R1 och R2 som indata. Samplern och regulatorn läggs till som diskreta system med kommandot *adddiscsys*. Till slut anropas *calcdynamics* och *calccost* som beräknar kostnaden.

Kostnadsfunktionen beräknas och plottas för de olika latenserna när latensen ökar från 0 till 100 % av samplingstiden. Känslighets- och komplementära känslighetsfunktionen beräknas med hjälp av kommandona *calcsens* och *calccompsens* och båda tar P och C som indata. Dessa plottas och maximala känsligheten och komplementära känsligheten beräknas.

LQG regulator

```
s = tf('s');
P = någon av processerna i tabell 1.
tau = 0;
proc_delay = 0;
P.Inputdelay = proc_delay;
titel = ['processen P'];

scenario = 1;     % scenario 1 = konstant
                          % scenario 2 = rektangel
                          % scenario 3 = normalfördelad
                          % scenario 4 = ändpunkt

% hitta en samplingstid för att skapa lqg-reg
% återkopplat system med C = 1, h = mittpunkten på 0.2-
0.6=wb*h
wb1 = bandwidth(feedback(P,1));
h1 = 0.4/wb1;

R1 = 1;
R2 = 0.01;
rho = 0.01;
Q = diag([1 rho]);

C1 = lqgdesign(P,Q,R1,R2,h1,tau);

wb = bandwidth(feedback(-d2c(C1,'tustin')*P,1))

hvec = [0.2/wb 0.4/wb 0.6/wb];
```

```
for i=1:1:3
    dt = hvec(i)/20;
    Jvec = zeros(1,round(hvec(i)/dt)+1);
    Jmat = zeros(4,length(Jvec));
    taumaxvec = 0:dt:hvec(i);
    for scenario = 1:1:4
        for taumax = taumaxvec

            Ptau = zeros(1,round(hvec(i)/dt)+1);
            if scenario = 1
                Ptau(1,round(taumax/dt)+1) = 1;
            End

            if scenario = 2
                Ptau(1:round(taumax/dt)+1) = 1;
            End

            if scenario = 3
                if taumax = 0
                    Ptau(1,1) = 1;
                else
                    p = normpdf(-2:4/(taumax/dt):2,0,1);
                    for t=1:1:taumax/dt+1
                        Ptau(1,t) = p(t);
                    end
                end
            end

            if scenario = 4
                Ptau(1,1) = 1;
                Ptau(1,round(taumax/dt)+1) = 1;
            end

            Ptau = Ptau/sum(Ptau);
            S = 1;
            C = lqgdesign(P,Q,R1,R2,hvec(i),tau);
            N = initjitterbug(dt,hvec(i));
            N = addtimingnode(N,1,Ptau,2);
            N = addtimingnode(N,2);
            N = addcontsys(N,1,P,3,Q,R1,R2);
            N = adddiscsys(N,2,S,1,1);
            N = adddiscsys(N,3,C,2,2);
            N = calcdynamics(N);
            J = calccost(N)
            Jvec(1,find(taumax==taumaxvec)) = J;
        end
        Jmat(scenario,:) = Jvec;
    end
```

```
    figure(i)
    taumaxvec = taumaxvec/hvec(i)*100;
    plot(taumaxvec,Jmat(1,:),'b')
    hold on
    plot(taumaxvec,Jmat(2,:),'g')
    hold on
    plot(taumaxvec,Jmat(3,:),'r')
    hold on
    plot(taumaxvec,Jmat(4,:),'k')

    title([titel,',h=',num2str(hvec(i)),'s,blå=constant,
         grön=uniform,röd=normalfördelad,svart=ändpunkt']);

    xlabel('Maximum Delay (in % of h)');
    ylabel('Cost J');

    figure(i+3)
    sens = calcsens(P,-C);
    sigma(sens)                              % Plotta sens
    Ms = norm(sens,inf);
    title(['Sensitivity            curve            ',titel,',
         h=',num2str(hvec(i)),', Ms=',num2str(Ms)]);

    figure(i+6)
    compsens = calccompsens(P,-C);
    sigma(compsens)             % Plotta compsens
    Mt = norm(compsens,inf);    % Compute the maximum
                                % complementary sensitivity
    title(['Complementary     sensitivity     curve',titel,',
         h=',num2str(hvec(i)),', Mt=',num2str(Mt)]);

end
```

PI/PD/PID regulator

Samma som ovan förutom att regulatorn är annorlunda. Parametriseringen för de olika regulatorerna finns i tabell 2. Dessutom måste regulatorn diskretiseras innan den används i Jitterbug. I derivatafiltret är N = 10.
Parametrarna finns i de olika ResL matriserna. L:et står för den grupp av processer som gäller, radnumret står för vilken process i gruppen som gäller, och kolumnnumret är var de olika parametrarna finns enligt följande:
PD: K – kolumn 6, T_d kolumn 7
PI: K – kolumn 10, T_i kolumn 11
PID: K – kolumn 15, T_i kolumn 16, T_d kolumn 17
K ska vara negativ eftersom regulatorn ska vara negativ då den läggs till i Jitterbug, därav minustecknet vid K.

I m-filen skall följande läggas till:

```
processer;                  % innehåller processerna
load parametrar   % innehåller parametrarna

% PI parametrar för t.ex process nummer 1
K = -Res4(1,10); Ti = Res4(1,11);
C1 = K*(1 + 1/(s*Ti));

% PD parametrar
K = -Res4(1,6); Td = Res4(1,7); N = 10;
C1 = K*(1 + s*Td/(1+s*Td/N));

% PID parametrar
K = -Res4(1,15); Ti = Res4(1,16); Td = Res4(1,17); N = 10;
C1 = K*(1 + 1/(s*Ti) + s*Td/(1+s*Td/N));

% Innanför den första for-satsen skall diskretiseringen
läggas:
C = c2d(C1,hvec(i),'Tustin');
```

4.2 LQG med latenskompensering

I fortsättningen kommer enbart $P(s)=\frac{1}{(s+1)^3}$ att studeras.

LQG regulatorn kan som ovan nämnts skapas med en latenskompensering genom att sätta parametern tau till ett värde större än noll. Här ska kostnadsfunktionen beräknas och jämföras med kostnadsfunktionen för LQG utan kompensering för 3 olika samplingstider som tas fram enligt ovan med tumregeln.
Kompenseringen ska vara för medellatensen. Medellatensen för de olika latenserna är:

- Konstant: tau = taumax
- Rektangel: tau = taumax/2
- Normalfördelad: tau = taumax/2
- Ändpunkt: tau = taumax/2

Matlab kod

Tidigare Matlab kod utökas med det som står bredvid ”% average latens”.

```
if scenario == 1
        Ptau(1,round(taumax/dt)+1) = 1;
        tau = taumax;       % average latens
end
```

```
if scenario == 2
        Ptau(1:round(taumax/dt)+1) = 1;
        tau = taumax/2;    % average latens
end
if scenario == 3
        if taumax == 0
                Ptau(1,1) = 1;
        else
                p = normpdf(-2:4/(taumax/dt):2,0,1);
                for t=1:1:taumax/dt+1
                        Ptau(1,t) = p(t);
                end
        end
        tau = taumax/2;    % average latens
end
if scenario == 4
        Ptau(1,1) = 1;
        Ptau(1,round(taumax/dt)+1) = 1;
        tau = taumax/2;    % average latens
end
```

4.3 Samma samplingstid för alla regulatorer

Samplingsintervallen blir olika för de olika regulatorerna. Därför ska kostnadsfunktionen utvärderas då alla regulatorer har samma samplingsintervall för att se vilken skillnad det är mellan regulatorerna.

Samplingstiderna för de olika regulatorerna för denna process ska enligt (1) vara (i sekunder):
LQG: 0.0747, 0.1495, 0.2242
PI: 0.8488, 1.6976, 2.5465
PID: 0.2990, 0.5979, 0.8969
PD: 0.3048, 0.6095, 0.9143

Samplingstiderna som kostnadsfunktionen ska utvärderas för väljs enligt:
hvec = $\begin{bmatrix}0.08 & 0.26 & 0.44 & 0.62 & 0.80\end{bmatrix}$.

4.4 Olika filter i derivata delen

Derivata delen i PD- och PID-regulatorn förstärker mätbruset vid höga frekvenser. Därför används en modifierad derivata del i dessa regulatorer där ett filter begränsar derivatadelen, se tabell 2. Filtret gör att maximala förstärkningen hos derivatadelen blir talet N.
I detta experiment ska inverkan av olika N undersökas för PD-regulatorn, det ska testas för
N = $\begin{bmatrix}2 & 5 & 10 & 20 & 30 & 50\end{bmatrix}$.

4.5 LQG regulator känslig för brus

Hittills har LQG-regulatorn designats med specifikationen att R2 = 0.01. Undersökning av hur systemet uppför sig när R2 = 0 görs i detta experiment. Detta bör göra regulatorn mer känslig för mätbruset. R2 i utvärderingen (*addcontsys*) ändras inte, enbart i *lqgdesign* sätts R2 till 0.

4.6 Process- och mätbrus var för sig

Utvärderingen av kostnadsfunktionen för brusen ska vara adderbar. Alltså ska summan av kostnaden för utvärdering av fallet R1 = 0, R2 = 0.01 och fallet R1 = 1, R2 = 0 vara lika stor som utvärderingen då R1 = 1 och R2 = 0.01 samtidigt. I koden ska alltså i *addcontsys* först R1 vara 0 och sedan ska R2 vara 0.

4.7 Modifierad ändpunktsfördelning

En liten ändring av ändpunktslatensen undersöks. I Matlabkoden ska sannolikhetsfunktionen se ut enligt följande:

- Så länge α (se experiment 1) är mindre än eller lika med 3 så ser det ut som tidigare
- När α är större än 3, sätt:
    - $P_\tau(1) = P_\tau(\alpha) = 1$
    - $P_\tau(2) = P_\tau(\alpha\text{-}1) = 0.25$

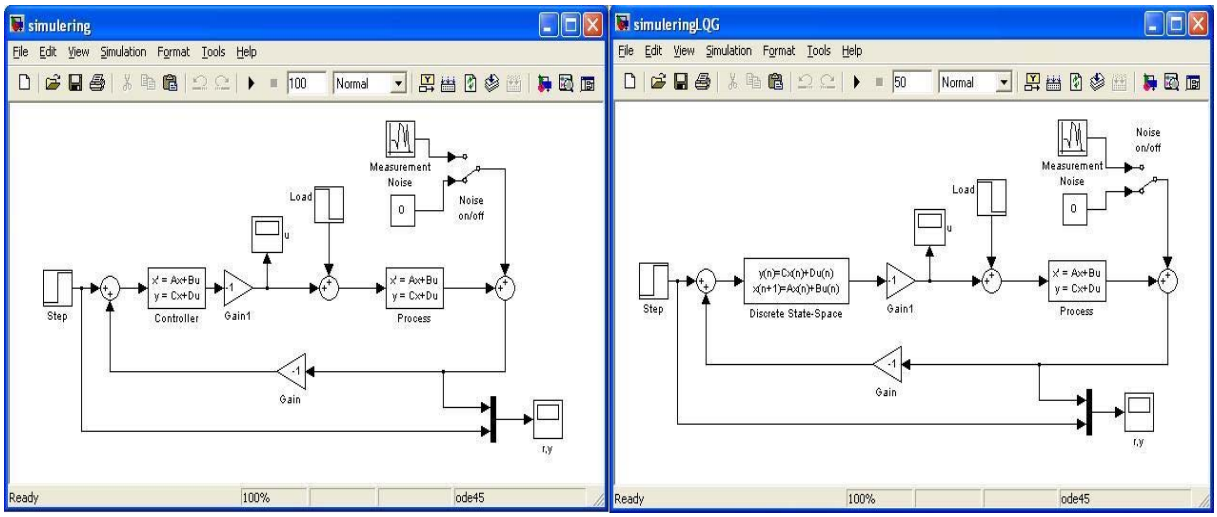
Detta medför att latensfördelningen kommer att se ut så här (α = 20):

$$P_\tau = \begin{bmatrix} 0.4 & 0.1 & 0 \ldots 0 & 0.1 & 0.4 \end{bmatrix}$$

```
if scenario = 4
          Ptau(1,1) = 1;
          Ptau(1,round(taumax/dt)+1) = 1;

          if ((taumax/dt)+1)>3    % modifiering av ändpunkt
          % sätter andra och näst sista till 1/4 av ändarna
          Ptau(1,2) = 0.25;
          Ptau(1,round(taumax/dt)) = 0.25;
          end
end
```

4.8 Simulering utan latens i Simulink

Regulatorerna simuleras i Matlab/Simulink modellen enligt figur 5:

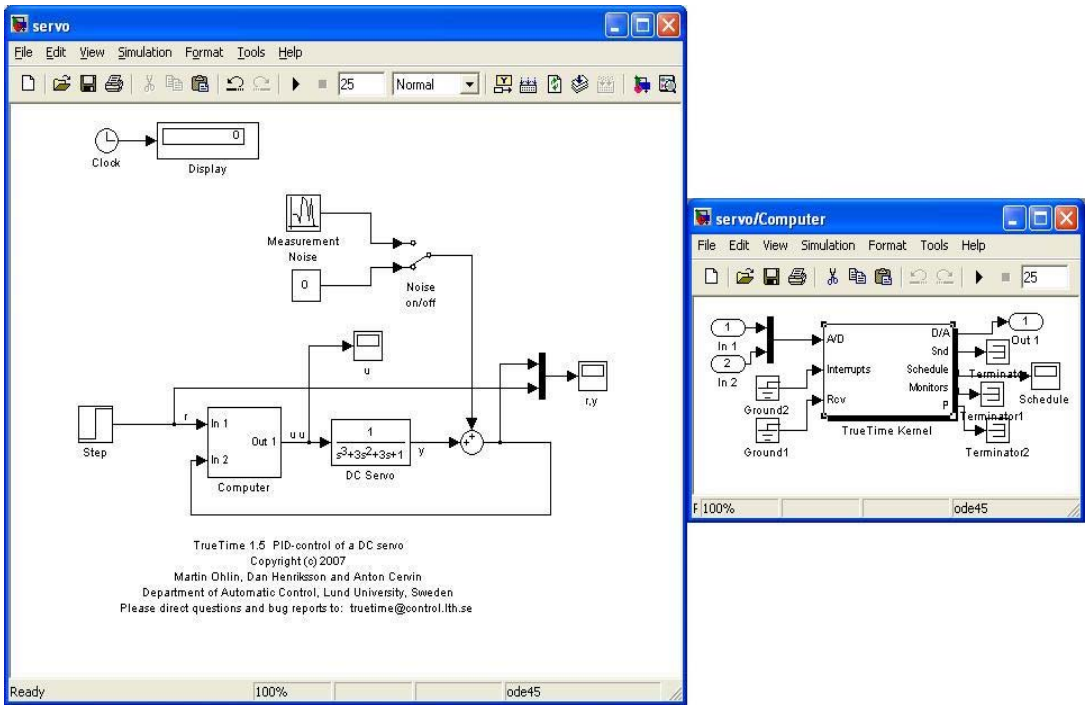


Figur 5 *Simuleringsmodell i Matlab, PI/PD/PID i den vänstra och LQG i den högra*

Stegsvar, lastsvar och brus undersöks för att se att regulatorerna beter sig som de har designats att göra. Latensen i experimentet är 0. Regulatorerna körs i den tidsdomän som de skapats, d v s PID/PI/PD i kontinuerlig tid och LQG i diskret tid.

4.9 Simulering med latens i TrueTime

För att verifiera resultaten från Jitterbug har även simuleringar i TrueTime gjorts, se figur 6.



Figur 6 *TrueTime modell för simulering av input/output latens, till höger TrueTime kärnan.*

Matlab kod

En initierings m-fil, *init_reg,* sätter upp TrueTime-simuleringen. TrueTime kärnan måste initieras och det görs med kommandot *ttInitKernel* som tar antal insignaler och antal utsignaler som indata, i detta fall två insignaler (r och y) och en utsignal (u). Sedan definieras attributen: periodtiden, deadline, release-tiden, och prioriteten. Slutligen så skapas en periodisk TrueTime task som tar bl.a. regulatorkoden och *data* som indata. *Data* innehåller bl.a. varje regulators olika parametrar.

Regulator koden är indelad i två segment. I det första segmentet läses referens- och mätsignalen in (r och y i figur 3). Sedan beräknas styrsignalen och tillstånden uppdateras. I det andra segmentet ställs styrsignalen ut.
Regulatorkoden, *lqgcode*, följer nedan. I lqgcode beräknas styrsignalen och tillstånden uppdateras direkt i segment 1, och i segment 2 ställs styrsignalen ut.

Init_reg

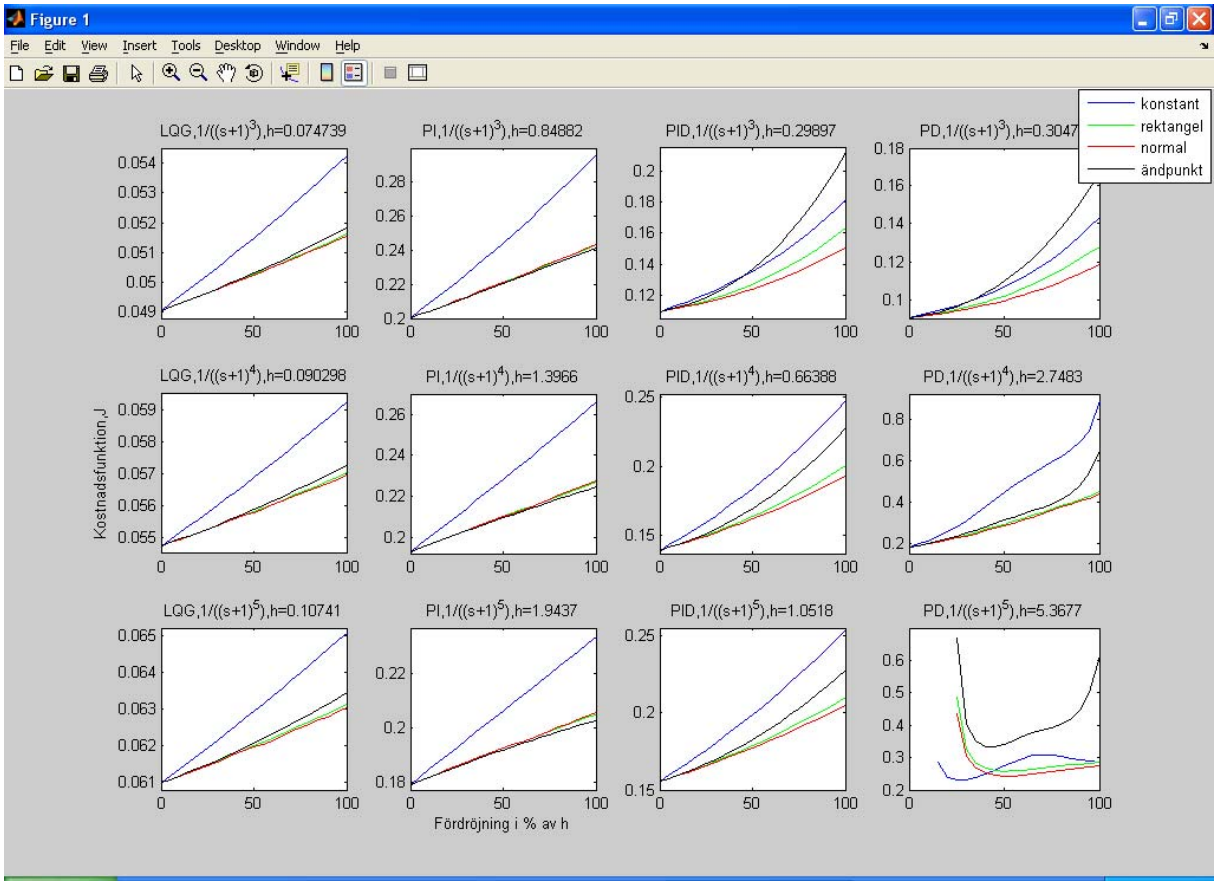
```
function init_reg

% Initiera TrueTime
ttInitKernel(2,  1,  'prioFP');  %  nbrOfInputs,  nbrOfOutputs,
                                 %  fixed priority

% Ladda regulator parametrar och samplingstid
load P2n3/H1
% Task attribut
period = hvec(i);           %   (i)   är   beroende   på   vilken
                            % samplingstid från (1)
deadline = period;
offset = 0.0;               % startar från början
prio = 1;

% Skapa task data
% C = ss(C)                 % PID, PI och PD regulatorn är en
                            % överföringsfunktion
data.a = C.a;
data.b = C.b;
data.c = C.c;
data.d = C.d;
data.x = zeros(size(C.a,1),1);
data.h = period;
data.u = 0;
data.yold = 0;
data.rChan = 1;
data.yChan = 2;
data.uChan = 1;
ttCreatePeriodicTask('reg_task', offset, period, prio,
'regcode', data);
```

```
function [exectime, data] = regcode(seg, data)

switch seg,
 case 1,
  r = ttAnalogIn(data.rChan);                % Läs referens
  y = ttAnalogIn(data.yChan);                % Läs mätsignalen

  % Beräkna kontrollsignalen och uppdatera tillstånden
  data.u = -(data.c * data.x + data.d * (r-y));
  data.x = data.a * data.x + data.b * (r-y);

  % någon av följande latenser (exectime) används
  exectime = α␣data.h(1);  % konstant(α väljs mellan 0 och 1)
  exectime = data.h(1)*rand;                  % rektangel
  exectime = data.h(1)*rand;                  % normal
  exectime = data.h(1)*round(rand);           % ändpunkt

 case 2,
  ttAnalogOut(data.uChan, data.u);  %Ställ ut styrsignalen
  exectime = -1;
end
```

5 Resultat

5.1 Jämförelse mellan olika regulatorer

(Se bilaga A, B och C för fler plottar)

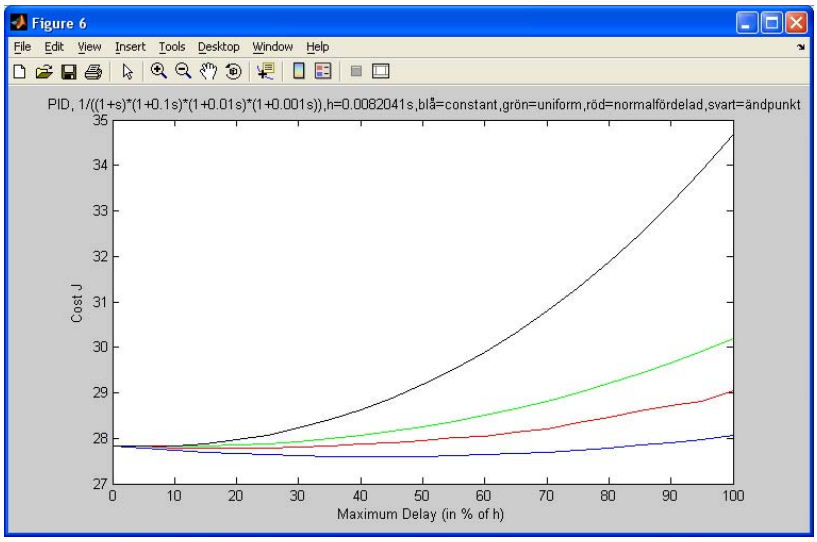


Figur 7 *Kostnadsfunktion för P_1 i tabell 1 för olika regulatorer med den snabbaste samplingstiden i samplingsintervallet (1).*

Figur 7 visar kostnadsfunktionen för olika processer och regulatorer. Figurtexten till respektive delfigur beskriver vilken regulator, process och samplingstid som använts. I varje figur står den blå linjen för kostnadsfunktionen då latensen är konstant, den gröna står för rektangelfördelad latens, den röda står för normalfördelad latens, medan den svarta står för ändpunktsfördelad latens. Skalan på x-axeln (0 - 100) är fördröjningen i procent av aktuell samplingstid.

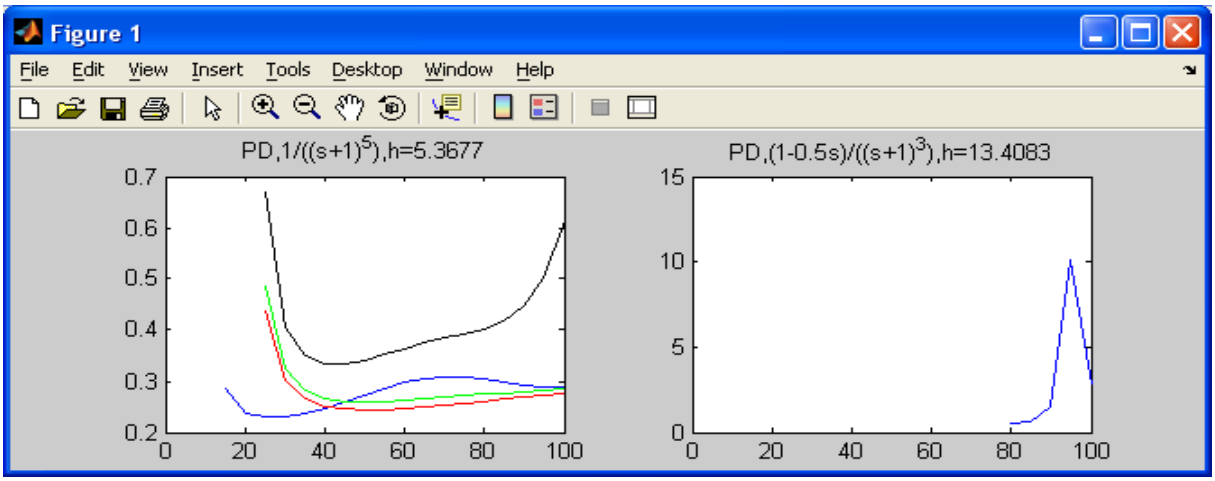
Figuren visar typiskt utseende för processerna i tabell 1 då den snabbaste samplingstiden från (1) används. LQG regulatorn är den regulator som ger klart bäst prestanda (lägst kostnad). PI regulatorn ger också god prestanda, medan PID och PD ger varierande resultat.LQG och PI regulatorn har ett tillsynes linjärt beroende mellan kostnadsfunktionen och latensen. För båda regulatorer ger den konstanta fördelningen störst kostnad. Inbördesrankningen mellan fördelningarna för de båda regulatorerna skiljer sig markant på en punkt: ändpunktsfördelad latens är nummer 2 (ger näst högst kostnad av de fyra fördelningarna) för LQG men är nummer 4 (lägst kostnad) för PI.

För PD och PID regulatorn är beroendet mellan kostnadsfunktionen och latensen mer kvadratiskt. Inbördesrankingen är för det mesta likadan för PD och PID: konstant latens ger störst kostnad, sen kommer ändpunkt, rektangel och sist normalfördelad latens. För vissa processer är kostnaden för ändpunktslatensen vid mindre fördröjning mindre än kostnaden för konstant latens, men blir mer när latensen närmar sig 100 % av samplingstiden.
När samplingstiden för dessa regulatorer som erhålles från (1) är liten, mindre än 10 ms, så ser dock kostnaden ut som i figur 8. Nu har den ändpunktsfördelade latensen störst kostnad medan den konstanta ger lägst kostnad. Dessutom är kostnaden, även fast den är stabil, mycket högre och är inte enbart ökande med ökande fördröjning som tidigare, utan har ett slags minimum som är olika för olika fördröjnings typer.



Figur 8 *Kostnadsfunktion för P_3 (α = 0.1) i tabell 1 för PID regulator med den snabbaste samplingstiden i samplingsintervallet (1).*

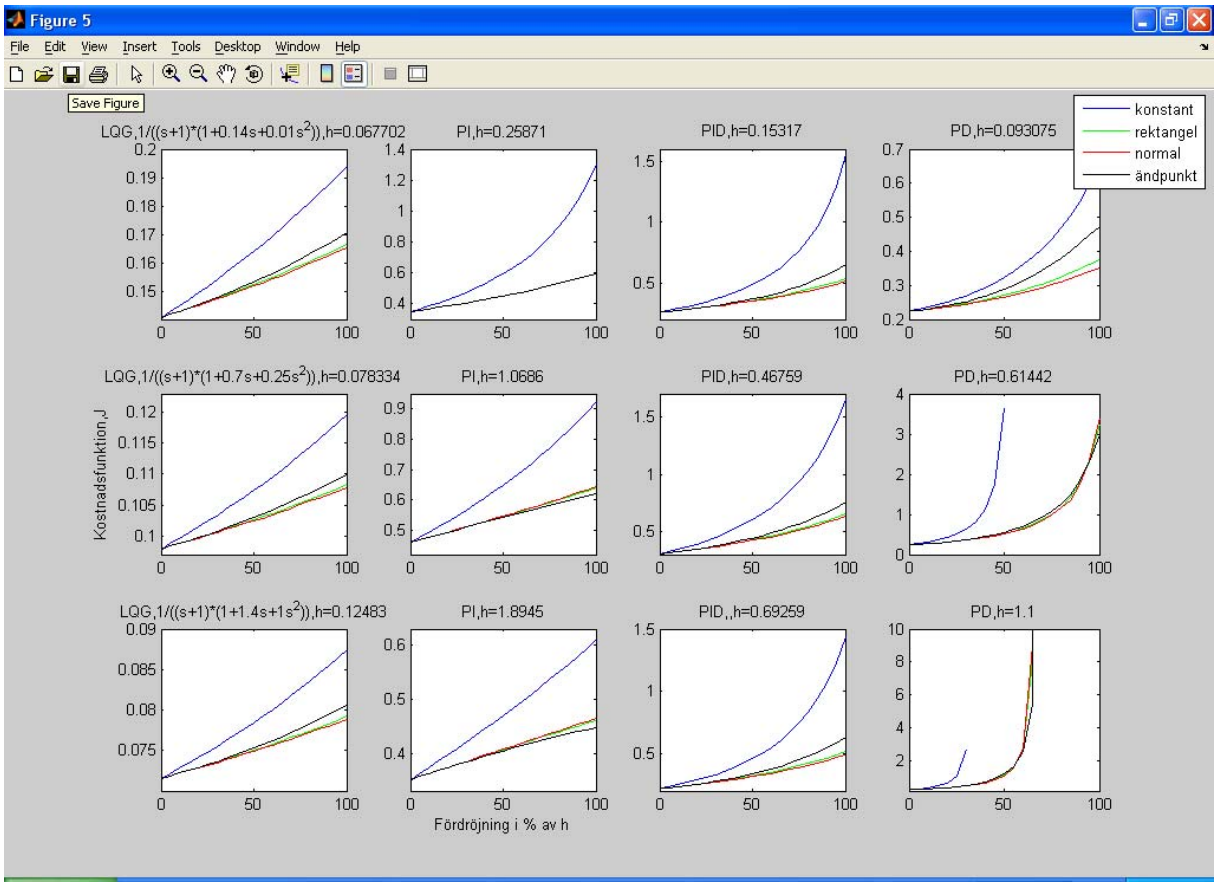
För två processer med PD som regulator ser kostnadsfunktion något udda ut, se figur 9.



Figur 9 *Kostnadsfunktion för P_1 (n = 5) med PD som regulator till vänster, och P_4 (α = 0.5) till höger*

I den högra figuren är kostnaden oändlig för alla typer utom den konstanta latensen som inte är oändlig när latensen är mer än 80 % av samplingstiden, som här är väldigt långsam: 13 sekunder.

När samplingstiden väljs som mittpunkten på (1) ser kostnadsfunktionen typiskt ut som figur 10. LQG ger fortfarande lägst kostnadsfunktion.



Figur 10 *Kostnadsfunktion för P_5 i tabell 1 för olika regulatorer med den mittersta samplingstiden i samplingsintervallet (1).*

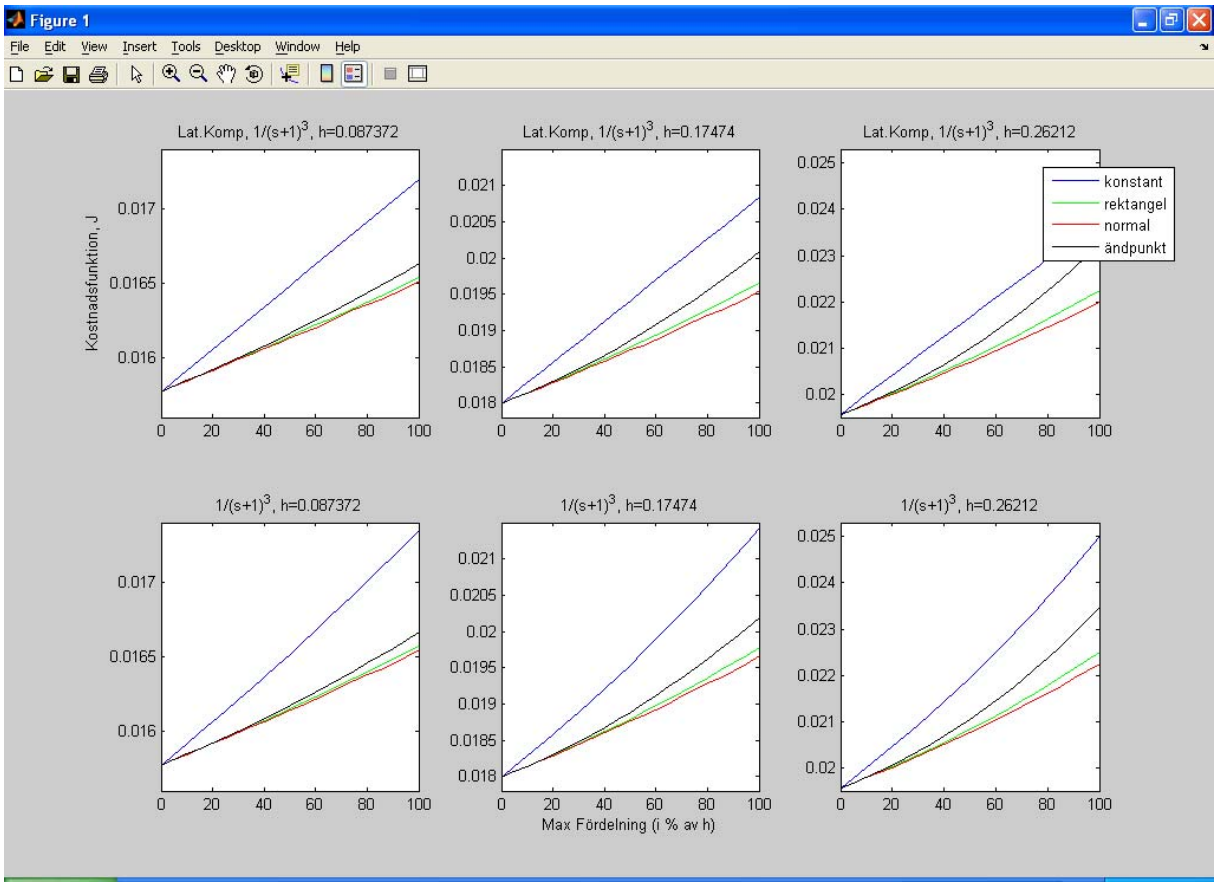
För LQG är beroendet fortfarande ganska linjärt och inbördesrankingen mellan fördelningarna är också oförändrad. För PI är det också oförändrat så när som på vissa processer där den konstanta fördelningen ibland är kvadratisk.
För PID är inbördesrankingen nu alltid så att den konstanta ger störst kostnadsfunktion och beroendet är fortfarande kvadratiskt. Den konstanta fördelningen tenderar även att ”sticka iväg” (ge instabilitet) när latensen närmar sig 100 % av samplingstiden.
PD regulatorn ger en kostnad som för vissa latensfördelningar och processer är oändlig vilket innebär att reglersystemet är instabilt. Detta beror till stor del på att samplingstiden är för långsam, men även en hel del på att D-delens förstärkning av mätbruset.

När samplingstiden väljs som den långsammaste tiden i (1) är kostnadsfunktionen för LQG fortfarande den som ger lägst kostnad.
Inbördesförhållandena för de olika fördelningarna är oförändrade för alla regulatorer. För PI är kostnaden fortfarande tillfredställande så länge fördröjningen är mindre än ca 70 % av samplingstiden. Då är det ett fåtal processer som får oändlig kostnad för den konstanta latensen.
För PID blir kostnaden för den konstanta latensen ofta oändlig då latensen är mer än ca 70 % av samplingstiden. För PD så är det som för den mittersta samplingstiden, kostnaden är ofta oändlig.

Slutsats

- LQG ger alltid bättre kostnad än PID
  - Naturligt, eftersom den är designad för samma brus och kostnadsfunktion som den utvärderas emot
  - Inte nödvändigtvis sant om brus och/eller kostnad är olika mellan design och utvärdering
- För LQG och PI så är för det mesta en linjär approximation bra
- För PD och PID så verkar det vara bättre med en kvadratisk approximation
- Det är i regel medellatensen som avgör kostnaden
  - De varierande fördröjningarna med medelvärde h/2 ger bättre resultat än en konstant fördröjning på h
- När h ökar inom tumregeln:
  - LQG ger stabilitet över hela tumregeln
  - PI ger mestadels stabilitet
  - PID ger mer instabilitet, konstant latens medför ofta instabilitet när latensen närmar sig 100 % av h
  - PD är för det mesta instabil

5.2 LQG med latenskompensering



Figur 11 *LQG med latenskompensering de övre 3, utan kompensering de undre 3*

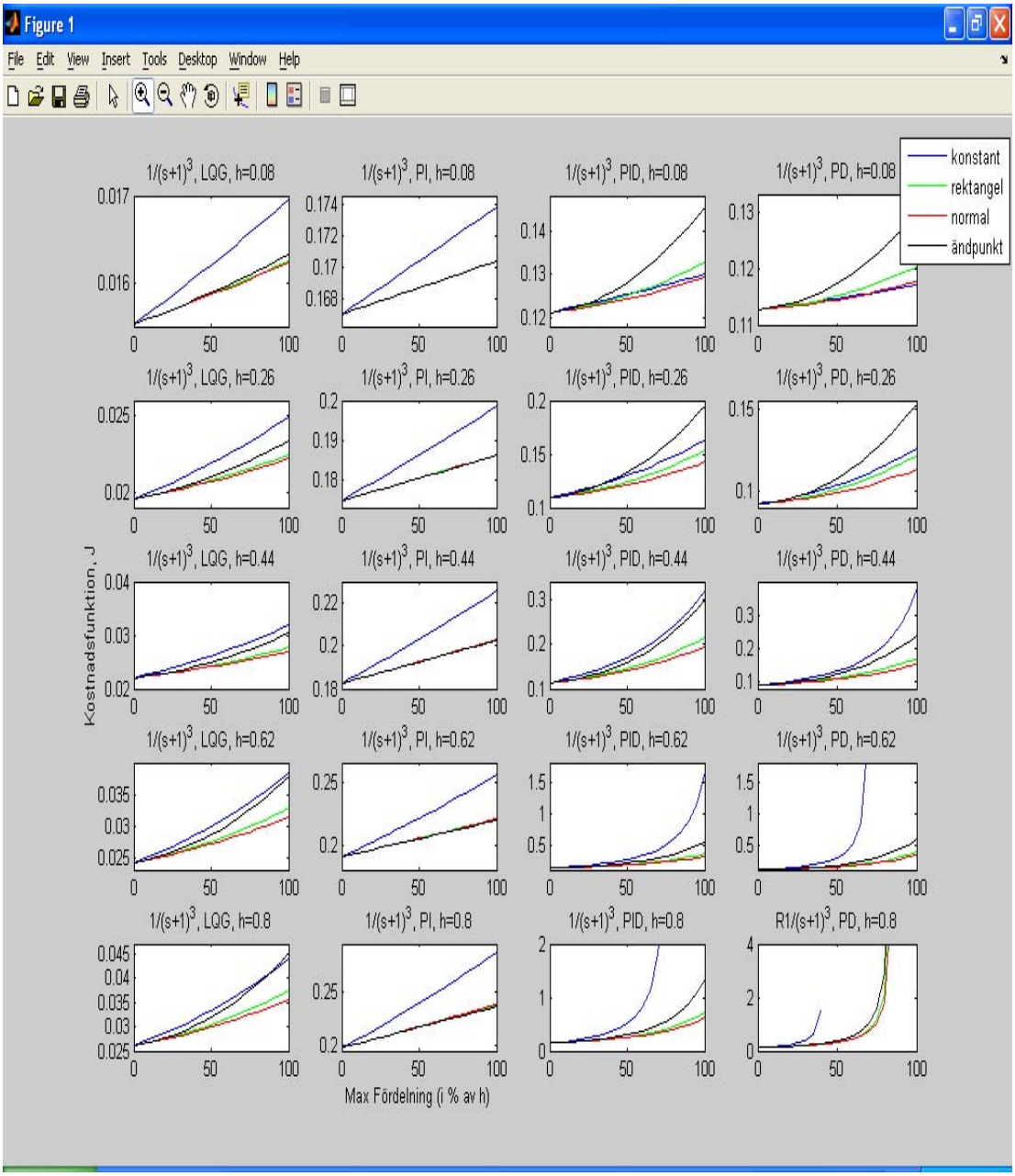
Man ser i figur 11 att långsammare samplingstid ger större skillnad mellan latenskompenserad LQG regulator och regulator utan kompensering, dock är skillnaden ganska så försumbar.

Slutsats

Latenskompenserande LQG regulator ger ingen större skillnad mot icke-latenskompenserande LQG regulator.

5.3 Samma samplingstid för alla regulatorer

Resultatet av simuleringarna visas i figur 12. Resultatet för PID och PD liknar tidigare simuleringar, se bilagorna. För PI så är regler systemen ännu stabilare eftersom samplingstiden nu är lägre. LQG ger fortfarande överlägset stabilast system trots att största samplingstiden är nästan 10 gånger större än den största som fås från (1). Dock blir förhållandet nu mera kvadratiskt. Dessutom ger ändpunktslatensen högst kostnad när h ökar.

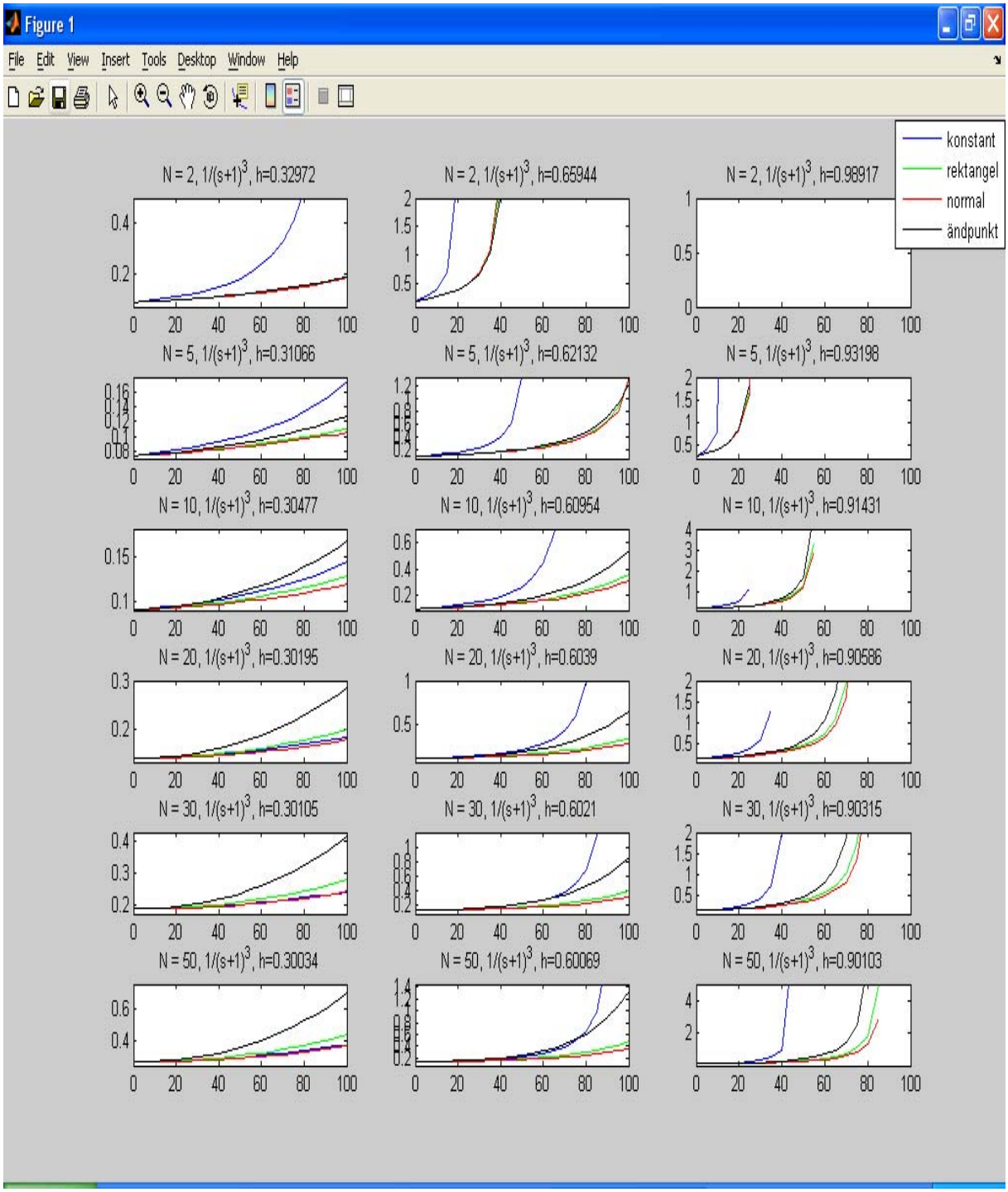


Figur 12 *Alla regulatorer med 5 olika samplingstider*

Slutsats

Att resultatet med PID och PD liknar tidigare resultat beror på att samplingsintervallet i detta fall liknar samplingsintervallet som fås med (1). Lägre samplingstid än vad (1) ger, resulterar i ett stabilare system, vilket blir fallet med PI regulatorn. Högre samplingstid än vad (1) ger, vilket blir fallet med LQG regulatorn, resulterar i ett mindre stabilt system, vilket är naturligt.

5.4 Olika filter i derivata delen



Figur 13 *Kostnadsfunktion för PD regulator för olika N och samplingstider från (1)*

Resultatet ser annorlunda ut beroende på i vilken del av samplingsintervallet (1) vi befinner oss. När samplingstiden väljs som den kortaste, vilket är fallet i kolumnen till vänster i figur 13, så verkar det som att N = 5 och N = 10 ger lägst sammanlagd kostnad för de 4 latenserna. Kostnaden verkar också öka med ökande N, förutom för N = 2, då filtret inte fungerar lika bra. Alla fördröjningar är stabila utom den konstanta latensen då N = 2. Inbördes rangordningen mellan ändpunkt (störst kostnad) och rektangel (näst störst) och normalfördelad (minst) är samma för alla N. Den konstanta latensfördelnings kostnad varierar i rangordning och går från att ha störst kostnad till att ha nästan lägst när N ökar.
När samplingstiden väljs som den mittersta punkten i (1) ger alla latenserna instabilitet då N = 2. När N ökar försvinner instabiliteten för normal-, rektangel- och ändpunktsfördelad latens, och den konstanta latensens stabilitet ökar.

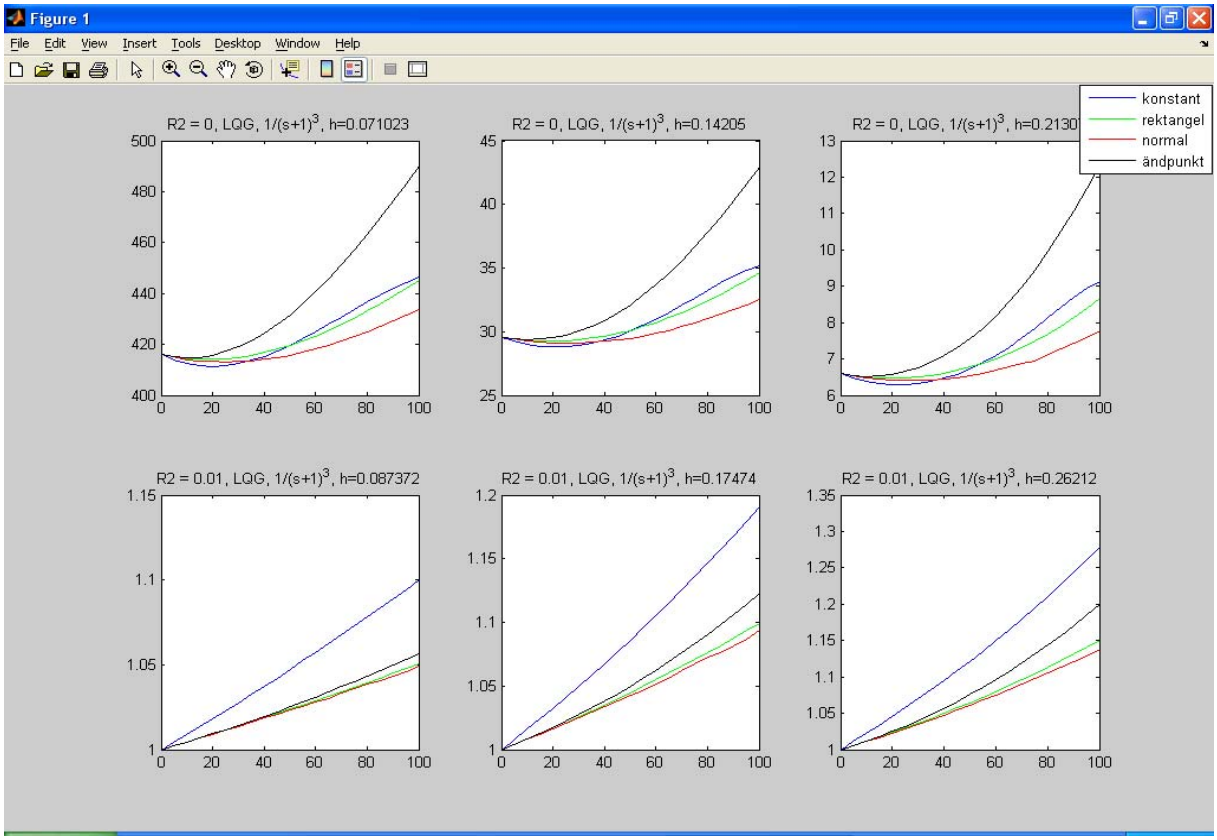
Då samplingstiden väljs som den längsta i (1) ger alla latenser instabilitet för alla N. Den konstanta latensen blir instabil först, följt av i tur och ordning: ändpunkt, rektangel och rektangel. Instabiliteten minskar dock med ökande N.

Slutsats

N-värdet i derivata-delen bör vara:

- 10-20 om h är liten
- mer än 30 annars

5.5 LQG regulator känslig för brus



Figur 14 *Kostnadsfunktion för LQG regulator designad med R2 = 0 i rad 1, och LQG med R2 = 0,01 i rad 2, utvärderad för tre olika samplingstider*

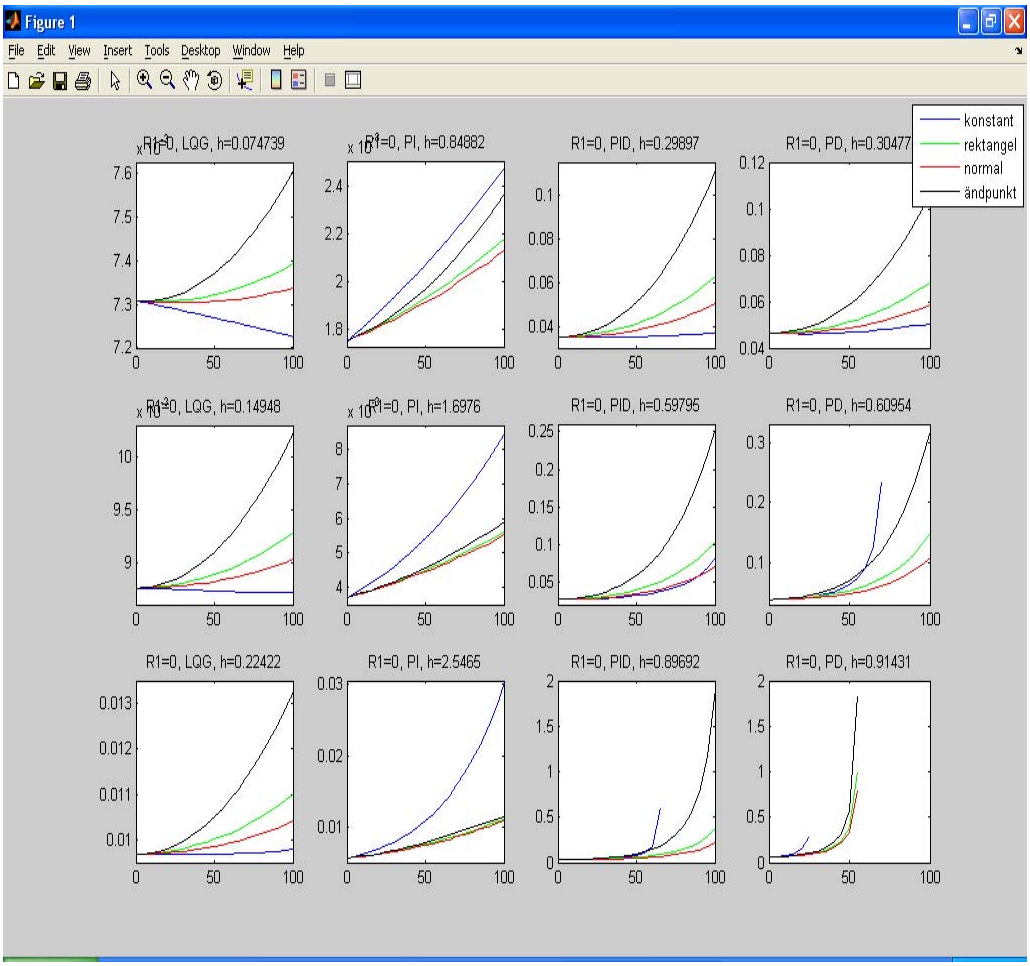
Rad 1 i figur 14 visar LQG regulatorn när den designats för att vara känslig för högfrekvent brus (R2 = 0), medan rad 2 visar LQG regulatorn designad för att motverka bruset (R2 = 0,01). LQG regulatorn med R2 = 0 beter sig som en PID eller en PD regulator som samplas väldigt snabbt, jämför med figur 8, och är klart mycket känsligare än för högfrekvent mätbrus än regulatorn som designats med R2 = 0,01. Kostnaden är hög men minskar när samplingstiden ökar (vilket den gör när från vänster till höger i figur 14), och detta skiljer sig från alla tidigare resultat. Sambandet mellan kostnaden och latensen är kvadratiskt då R2 = 0, medan den är linjär då R2 = 0,01. Ändpunktslatensen då R2 = 0 ger högst kostnad medan övriga ligger nära varandra, vilket är en stor skillnad från då R2 = 0,01 där det är den konstanta latensen som ger högst kostnad med övriga ligger nära varandra.

Slutsats

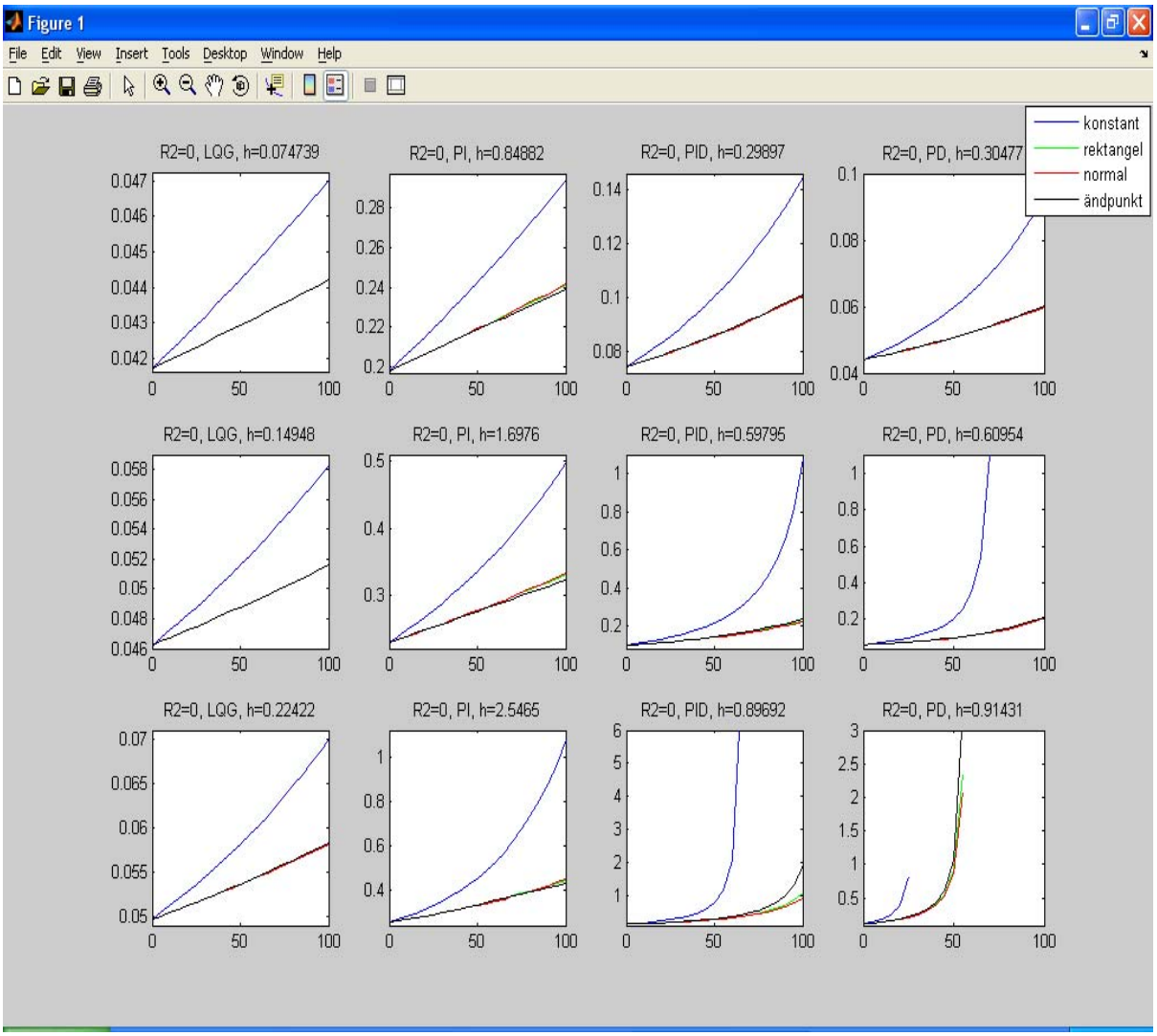
LQG som designas för lågt mätbrus (snabb observerare) men utvärderas med högt mätbrus ger högst kostnad för ändpunktslatens.

5.6 Process- och mätbrus var för sig

Beräkningar i Matlab ger att summan av de två utvärderade fallen blir exakt som utvärderingen då R1 = 1 och R2 = 0.01. Från figur 15 visar kostnadsfunktionen utvärderad då R1 = 0, och figur 16 visar kostnadsfunktionen utvärderad då R2 = 0.

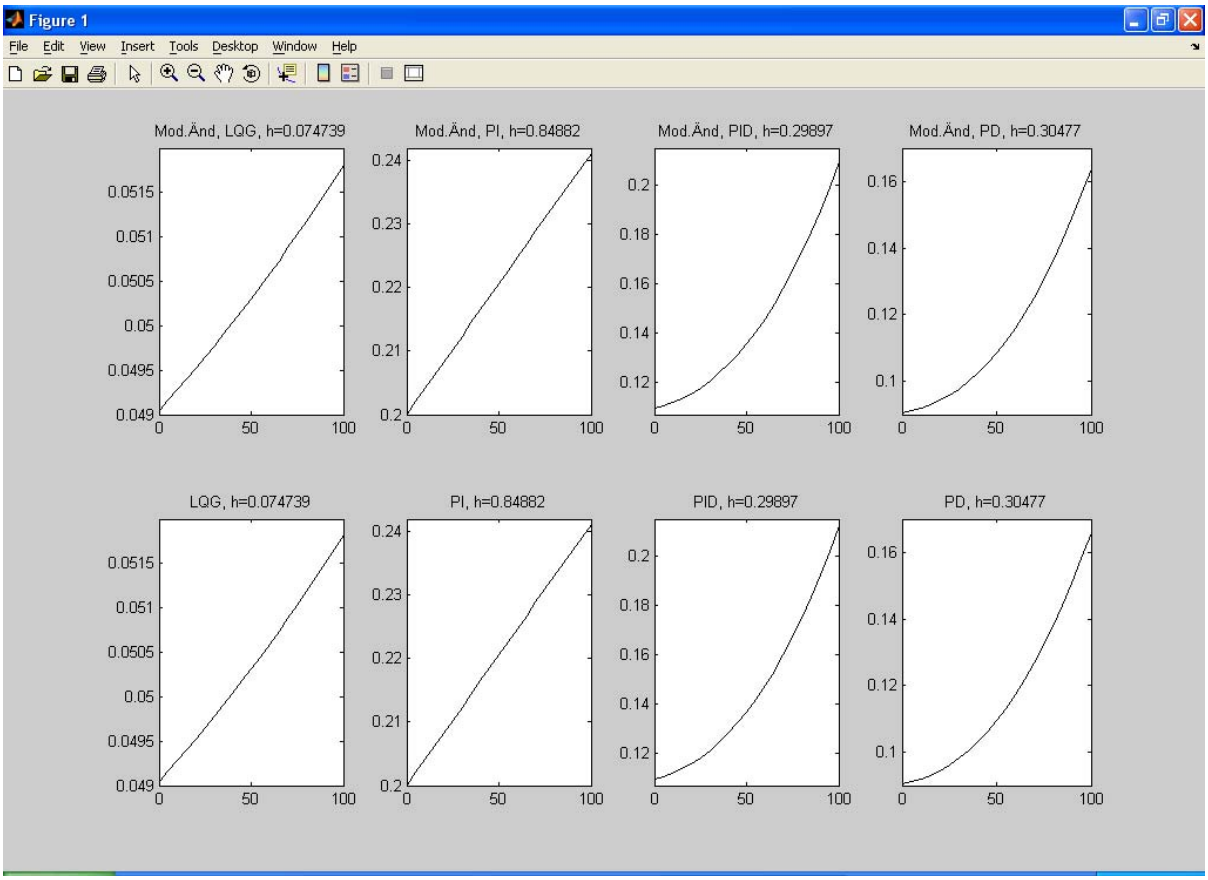


Figur 15 *Kostnadsfunktion utvärderad då R1 = 0 (y-värdena för LQG och PI ska delas med 1000 för de två första raderna)*



Figur 16 *Kostnadsfunktion utvärderad då R2 = 0*

5.7 Modifierad ändpunktsfördelning



Figur 17 *Kostnad utvärderad för en modifierad ändpunktsfördröjning.*

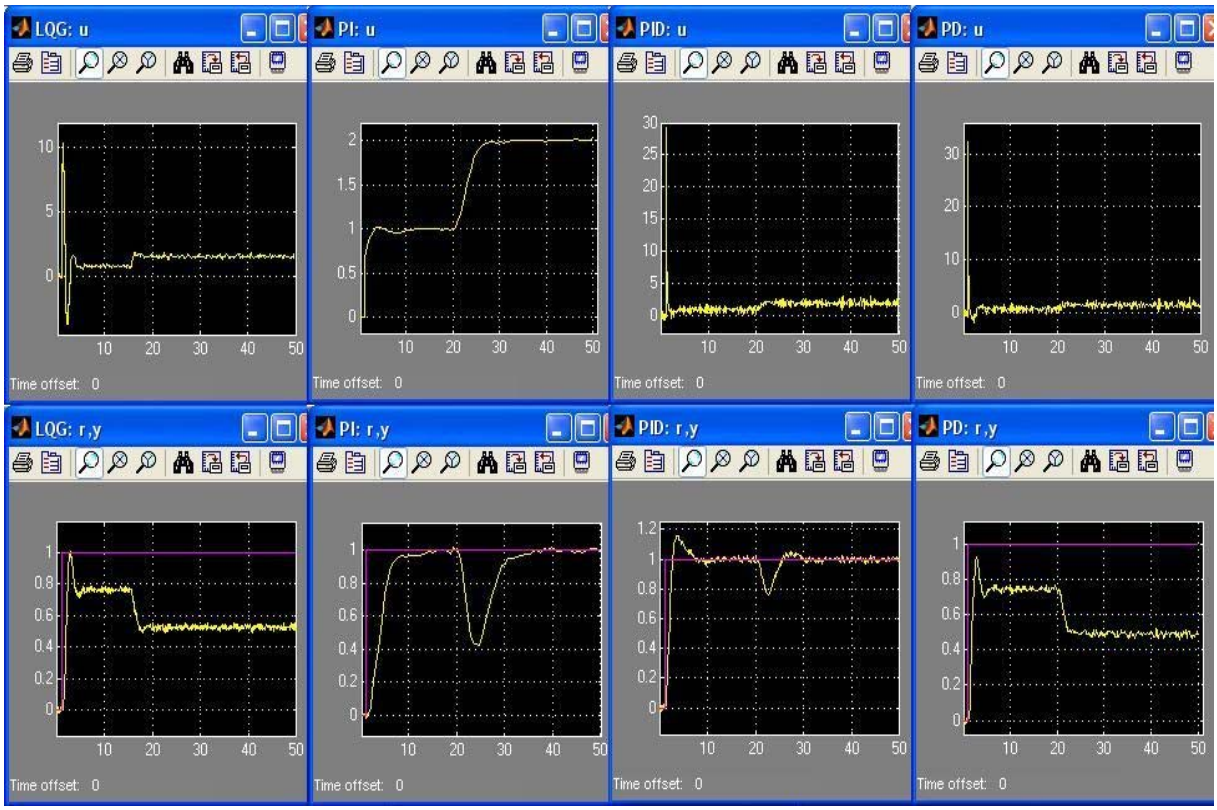
Skillnaden mellan de båda fallen är ytterst liten, se figur 17. Skillnaden är dock större för PID och PD regulatorn i jämförelse med LQG och PI regulatorn, som knappt har någon skillnad.

5.8 Simulering utan latens i Simulink

Simuleringarna i figur 18 visar att PI regulatorn inte är lika aggressiv som övriga regulatorer utan når steget i referenssignalen långsammare. Den återhämtar sig från laststörningen långsammare än PID. Samtidigt så motverkar den mätstörningarna på ett bättre sätt.

Slutsats

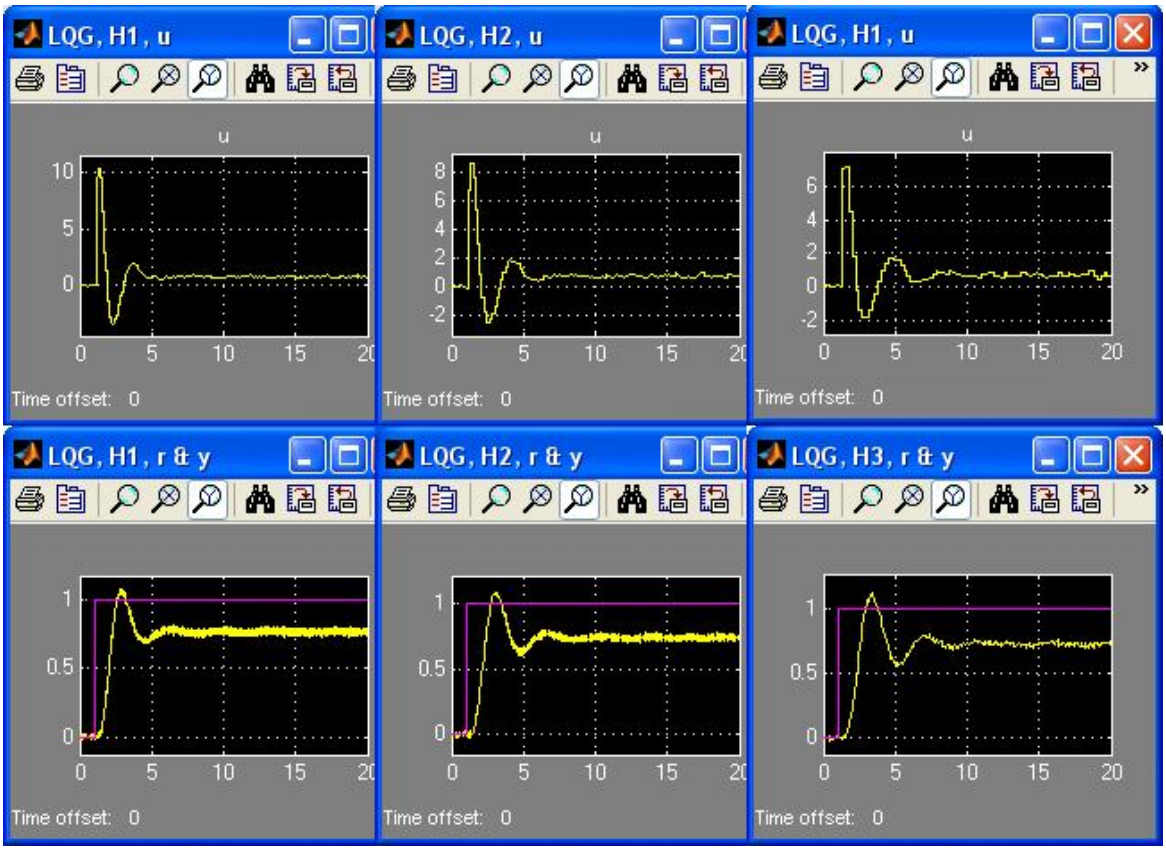
Simuleringarna visar att regulatorerna ger ett tillfredsställande uppförande.



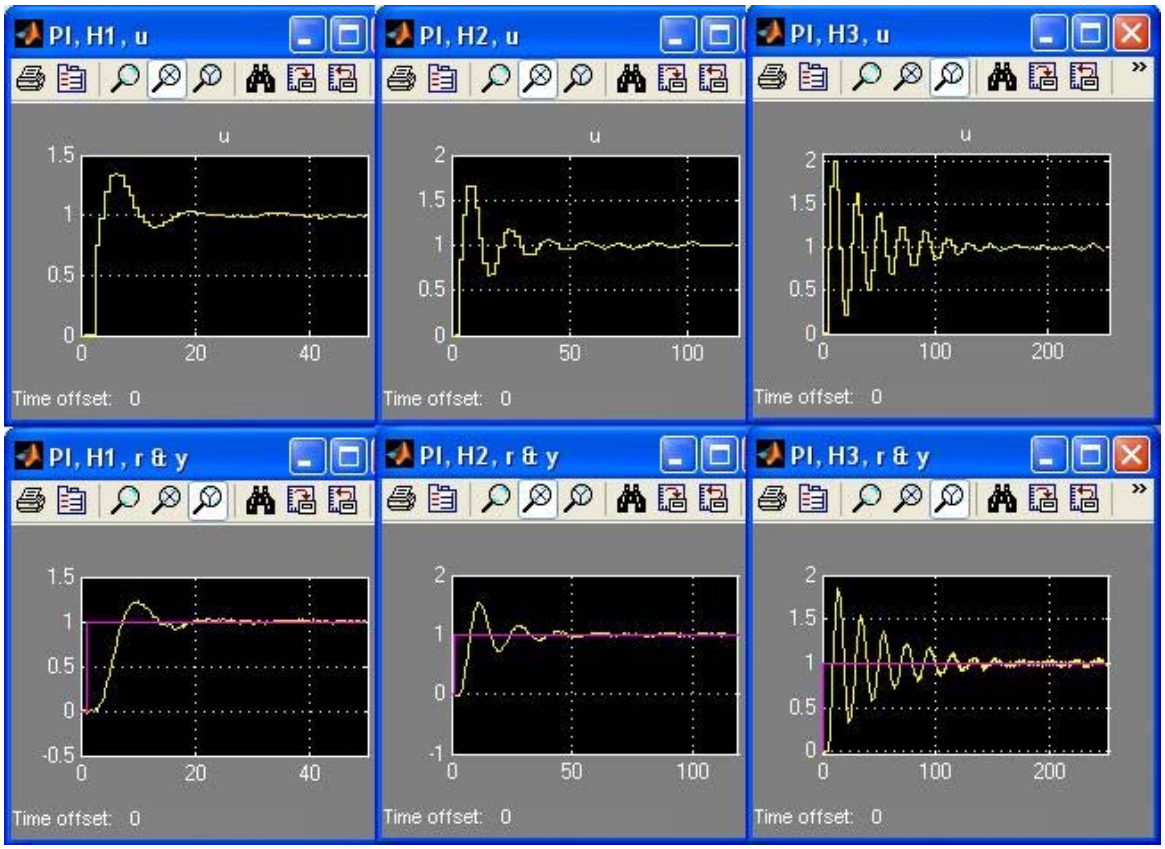
Figur 18 *Simuleringar av de olika regulatorerna*

5.9 Simulering med latens i TrueTime

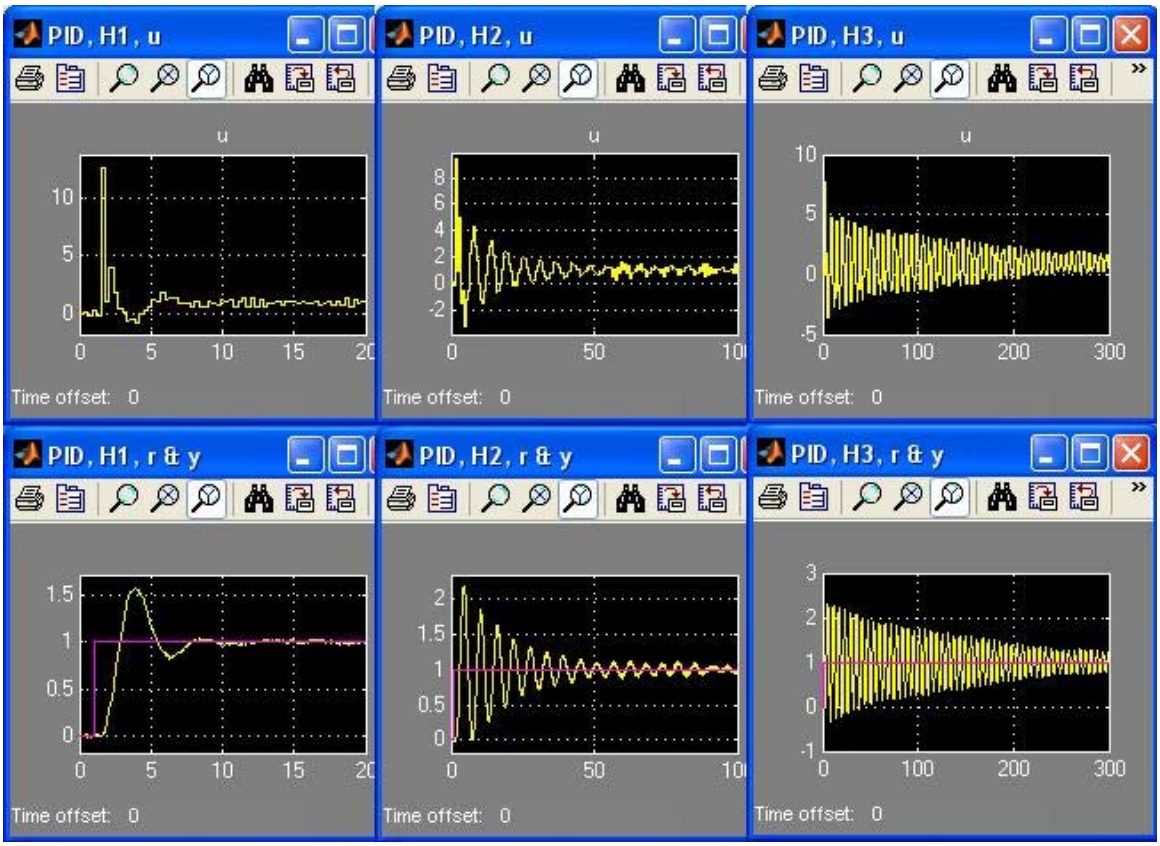
När latensen är konstant och latensen är maximal latens eller då systemet blir instabilt, ser styrsignalen och stegsvaret för de olika regulatorerna ut enligt figurerna 19, 20, 21 och 22. Som synes ger LQG regulatorn stabilast system. Detta framgick ju redan innan då kostnadsfunktionen utvärderades i Jitterbug, jämför med bilagorna A, B och C. När samplingstiden ökar så ökar också instabiliteten. Från bilagorna framgår att LQG och PI regulatorn aldrig blir instabil, vilket också överrensstämmer med TrueTime simuleringarna. För PID så blir systemet instabilt när den längsta samplingstiden används, men först när latensen är mer än runt 65 % av samplingstiden, jämför bilagorna. PD regulatorn ger ett instabilt system med den mittersta samplingstiden då latensen är mer än 65 % av samplingstiden, och med den längsta samplingstiden då latensen är mer än 25 %, detta också enligt Jitterbug.



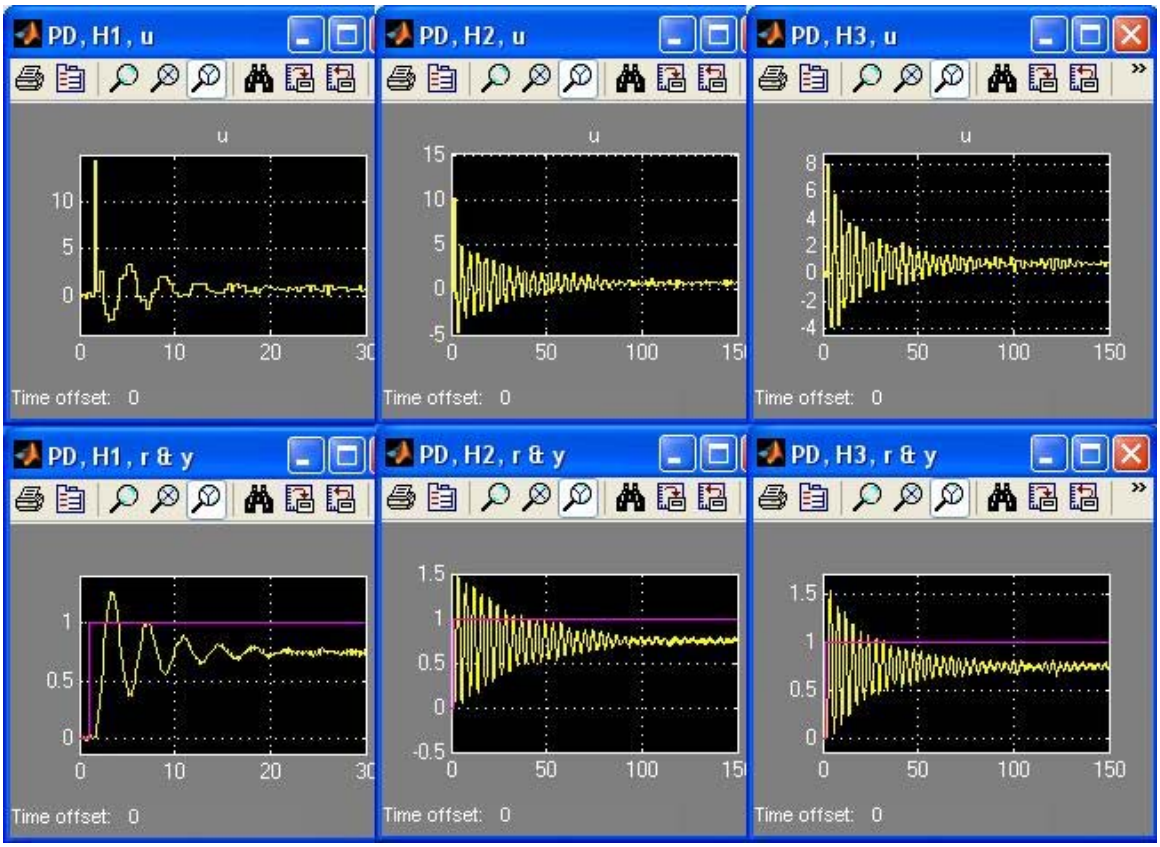
Figur 19 *LQG med konstant latens, 100 % av samplingstiden, för de tre samplingstiderna*



Figur 20 *PI med konstant latens, 100 % av samplingstiden, för de tre samplingstiderna*

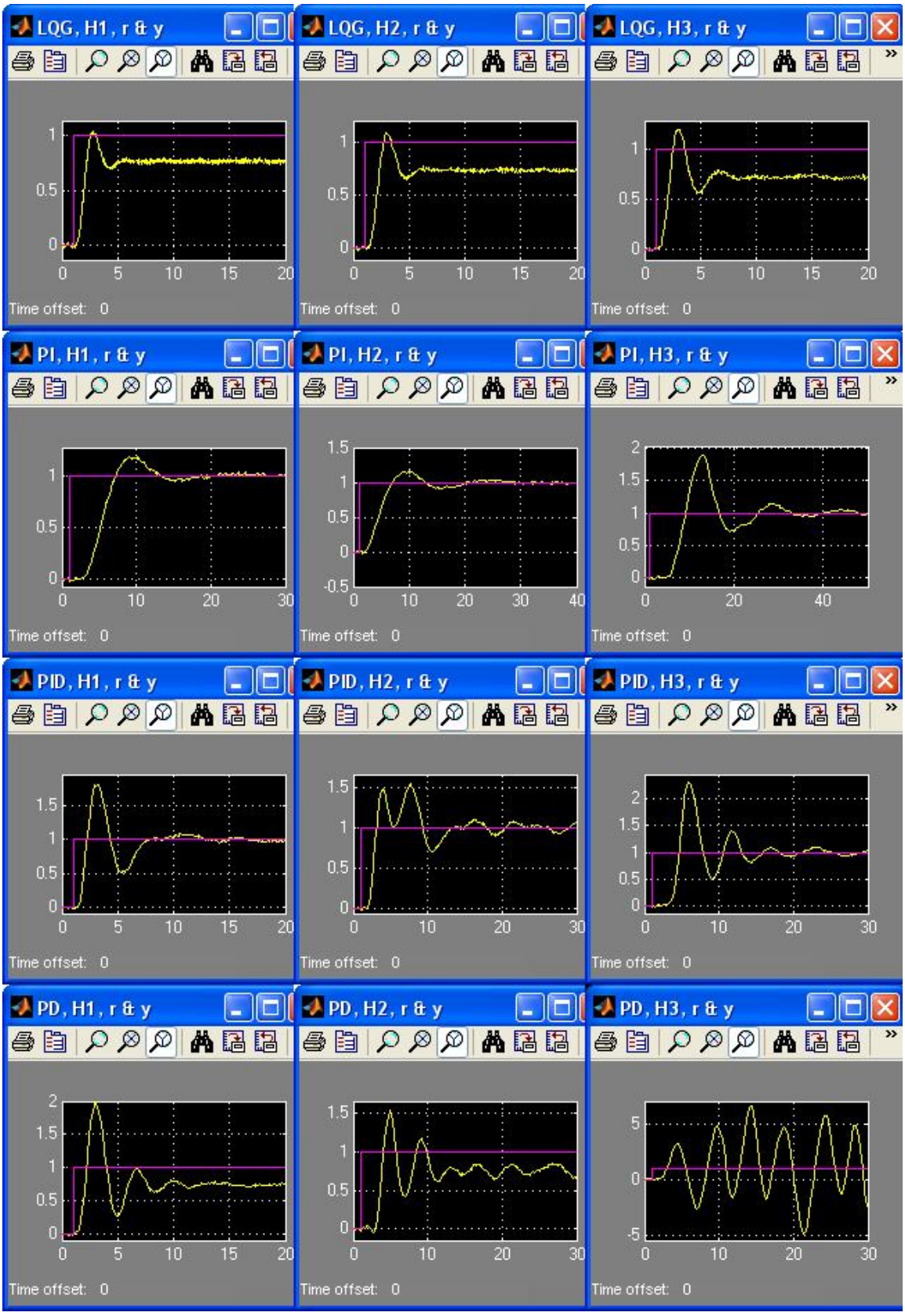


Figur 21 *PID med konstant latens, 100 % av samplingstiden för de två först samplingstiderna, och 65 % för den sista samplingstiden*



Figur 21 *PD med konstant latens, 100 % av samplingstiden för den första samplingstiden, 65 % av den andra, och 20 % av den sista samplingstiden*

För ändpunktsfördelad latens blir resultatet enligt figur 22:



Figur 22 *Simulering av änpunktsfördelad latens*

Enligt Jitterbug ska alla regulatorer för alla samplingstider från (1) ge stabilt system med undantag för PD regulatorn med den längsta samplingstiden. Detta är också fallet av simuleringarna i TrueTime. Observera att resultatet inte alltid ser ut enligt figur 22, utan varierar från en körning till en annan eftersom latensen är slumpmässig nu och inte längre konstant som i föregående fall.

Simulering av normal- och rektangelfördelad latens ger liknande resultat som simulering av ändpunktsfördelad latens.

Slutsats

Simuleringarna i Jitterbug och TrueTime ger ett väldigt överrensstämmande resultat.

6 Sammanfattning

Simuleringarna i Jitterbug och TrueTime har gett en större insikt i hur jitter påverkar ett digitalt reglersystem. Det finns många tumregler för hur samplingstiden ska väljas, i detta fall har bandbreddsregeln använts. Undersökningen har gjorts på 4 olika regulatorer: LQG, PI, PID och PD. Jittret som undersökts har varit av olika fördelningar: konstant, normal, rektangel och ändpunkt.

Simuleringarna har givit följande slutsatser:

- LQG ger alltid bättre kostnad än PID
  - Naturligt, eftersom den är designad för samma brus och kostnadsfunktion som den utvärderas emot
  - Inte nödvändigtvis sant om brus och/eller kostnad är olika mellan design och utvärdering
- För LQG och PI så är för det mesta en linjär approximation bra mellan kostnaden och samplingstiden
- För PD och PID så verkar det vara bättre med en kvadratisk approximation
- När h ökar inom tumregeln:
  - LQG ger stabilitet över hela tumregeln
  - PI ger mestadels stabilitet
  - PID ger mer instabilitet, konstant latens medför ofta instabilitet när latensen närmar sig 100 % av h
  - PD är för det mesta instabil
- Det är i regel medellatensen som avgör kostnaden
  - De varierande fördröjningarna med medelvärde h/2 ger bättre resultat än en konstant fördröjning på h
- Ovan gäller inte om man har ett ”högfrekvent” jitter (ändpunktsfördelningen) och man har en regulator med hög förstärkning för höga frekvenser vilket är fallet i:
  - PD och PID
  - LQG om den designas för lågt mätbrus (snabb observerare) men utvärderas med högt mätbrus
- N-värdet i derivata-delen bör vara:
  - 10-20 om h är liten
  - mer än 30 annars
- Simuleringarna från Jitterbug och TrueTime ger överrensstämmande resultat

Analytiska uttryck för kostnadsfunktionen går att beräkna i mycket enkla fall, som t ex för en integrator med fördröjning och en minimalvariansregulator enligt följande:

$\frac{dx(t)}{dt} = u(t-L) + \upsilon_c(t)$, $0 \le L \le h$, υ_c är vitt brus

Kostnadsfunktionen är: $J = \frac{1}{h} E\left\{ \int_0^h x^2(t)\,dt \right\}$

Beräkningarna leder till att kostnaden kommer att bero på ett linjärt samband enligt: $J = \frac{3+\sqrt{3}}{6} h + L$, L är latensen. Detta skulle innebära att en linjär approximation av sambandet mellan kostnaden och samplingstiden enligt ovan ändå är ganska bra.

7 Referenser

[1] Karl J Åström och Björn Wittenmark, *Adaptive Control*, 2:a upplagan, Addison-Wesly 1995

[2] Karl J Åström och Björn Wittenmark, *Computer-Controled Systems*, 3:e upplagan, Upper Saddle River, NJ: Prentice Hall, 1997

[3] Karl-Erik Årzén, *Real-Time Control Systems*, Avdelning för Reglerteknik, Lunds Tekniska Högskola, Sverige, 2003

[4] Gene F Franklin & J David Powell & Abbas Emami-Naeini, *Feedback Control of Dynamic Systems*, 3:e upplagan, Addison-Wesly 1993

[5] Anton Cervin, Dan Henriksson, Johan Eker och Karl-Erik Årzén, ”*Analysis and Simulation of Controller Timing*”, Avdelning för Reglerteknik, Lunds Tekniska Högskola, Sverige, 2003

[6] Anton Cervin, Bo Lincoln, Johan Eker, Karl-Erik Årzén, Giorgio Buttazzo, ”*The Jitter Margin and its Application in the design of Real-Time Control Systems”*, Avdelning för Reglerteknik, Lunds Tekniska Högskola, Sverige, augusti, 2004

[7] Anton Cervin och Bo Lincoln, *Jitterbug 1.1 – Referens Manual*, Avdelning för Reglerteknik, Lunds Tekniska Högskola, Sverige, januari 2003

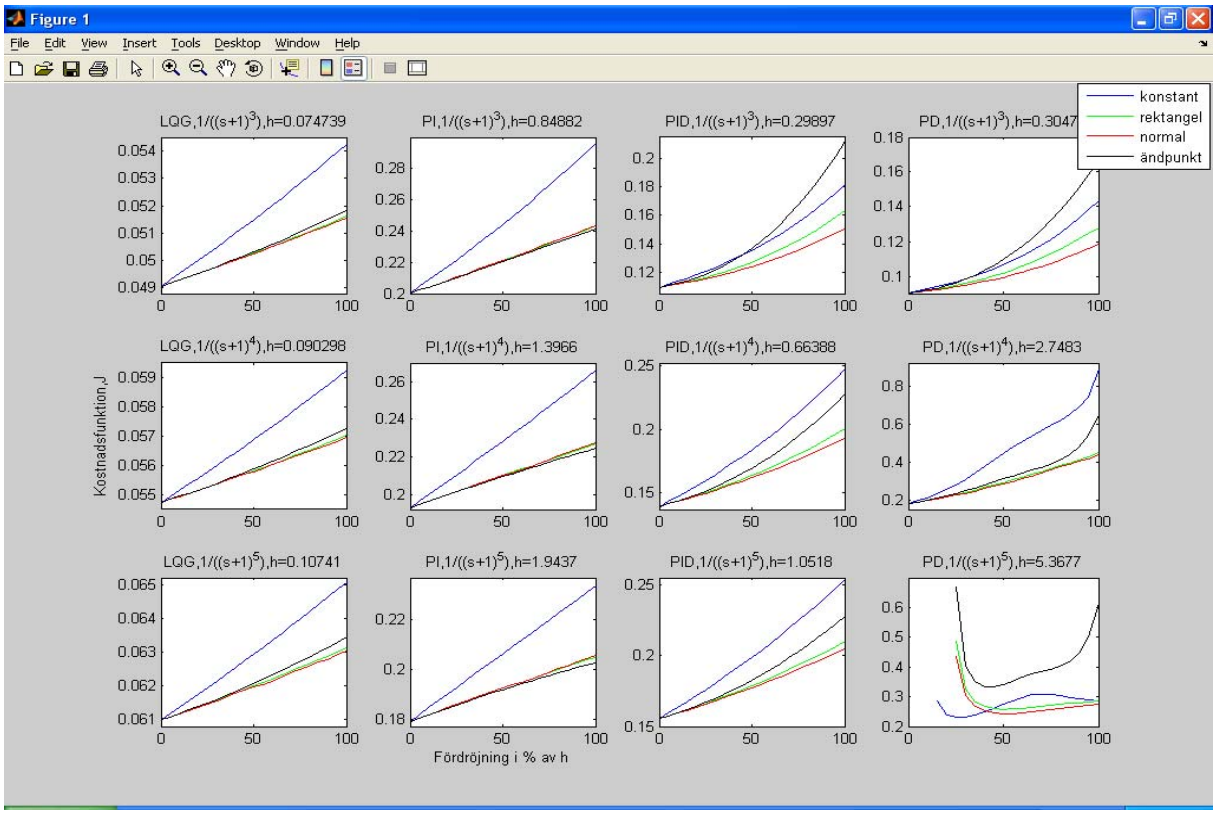
[8] Anton Cervin, Dan Henriksson och Martin Ohlin, *TrueTime 1.5 – Referens Manual*, Avdelning för Reglerteknik, Lunds Tekniska Högskola, Sverige, januari 2007

[9] Tore Hägglund, *Process Batch och PID Parametrar*, Avdelning för Reglerteknik, Lunds Tekniska Högskola, Sverige, 2007

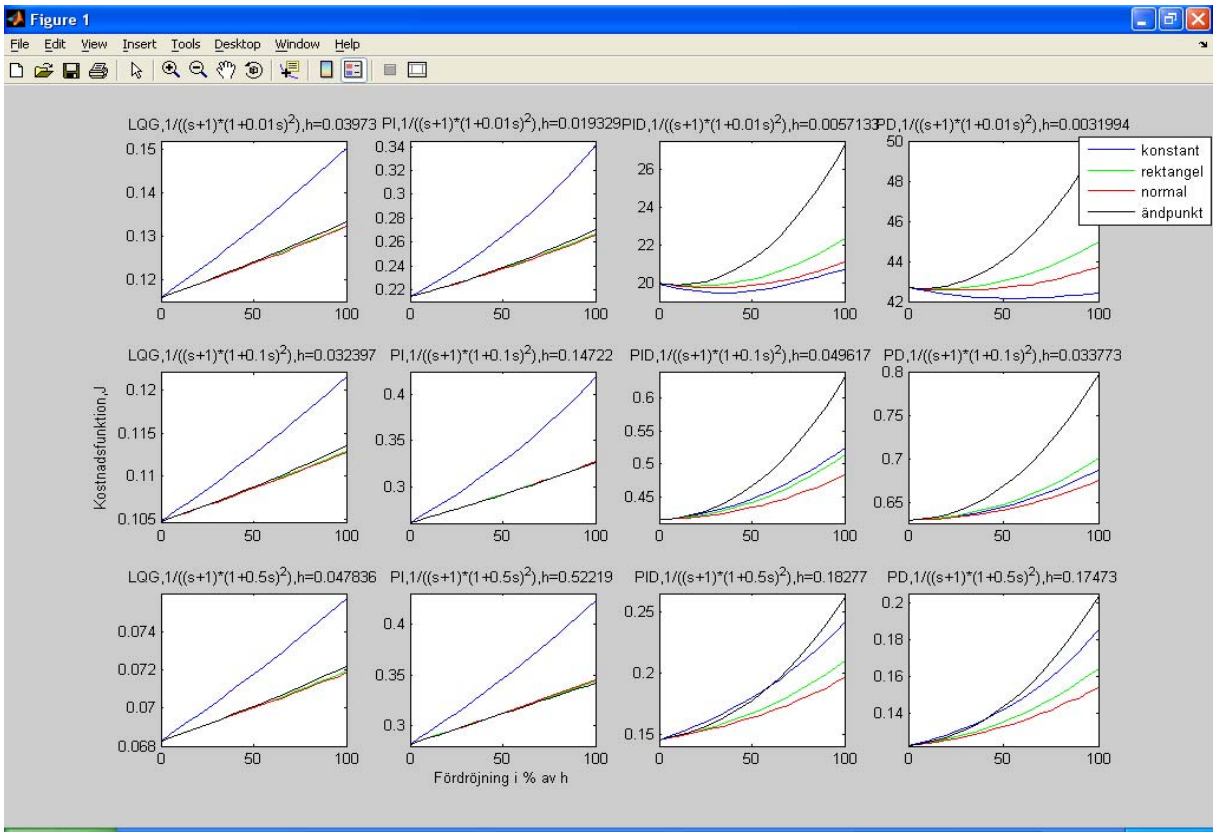
8 Bilagor

I bilagorna A, B och C återfinns utvärderingen av kostnadsfunktionen för vart och ett av reglersystemen som undersökts enligt experiment 4.1. I bilaga A utvärderas kostnadsfunktionen då samplingstiden är den kortaste som erhålles från (1), i bilaga B sker utvärderingen då den mittersta samplingstiden ur (1) används, och i bilaga C så är det den längsta samplingstiden ur (1) som används. I varje bilaga visas 5 figurer, 1 för var och en av grupp av processer (P_1, P_2, P_3, P_4 och P_5) i tabell 1. Raderna i varje figur visar de olika processerna inom en grupp av processer (t ex P_1 med n = 3), medan kolumnerna visar de olika regulatorerna (från vänster till höger) LQG, PI, PID och PD som har designats till gällande process. Varje delfigur inom dessa figurer har en rubrik som talar om vilken regulator och process det är som gäller och vad samplingstiden är.

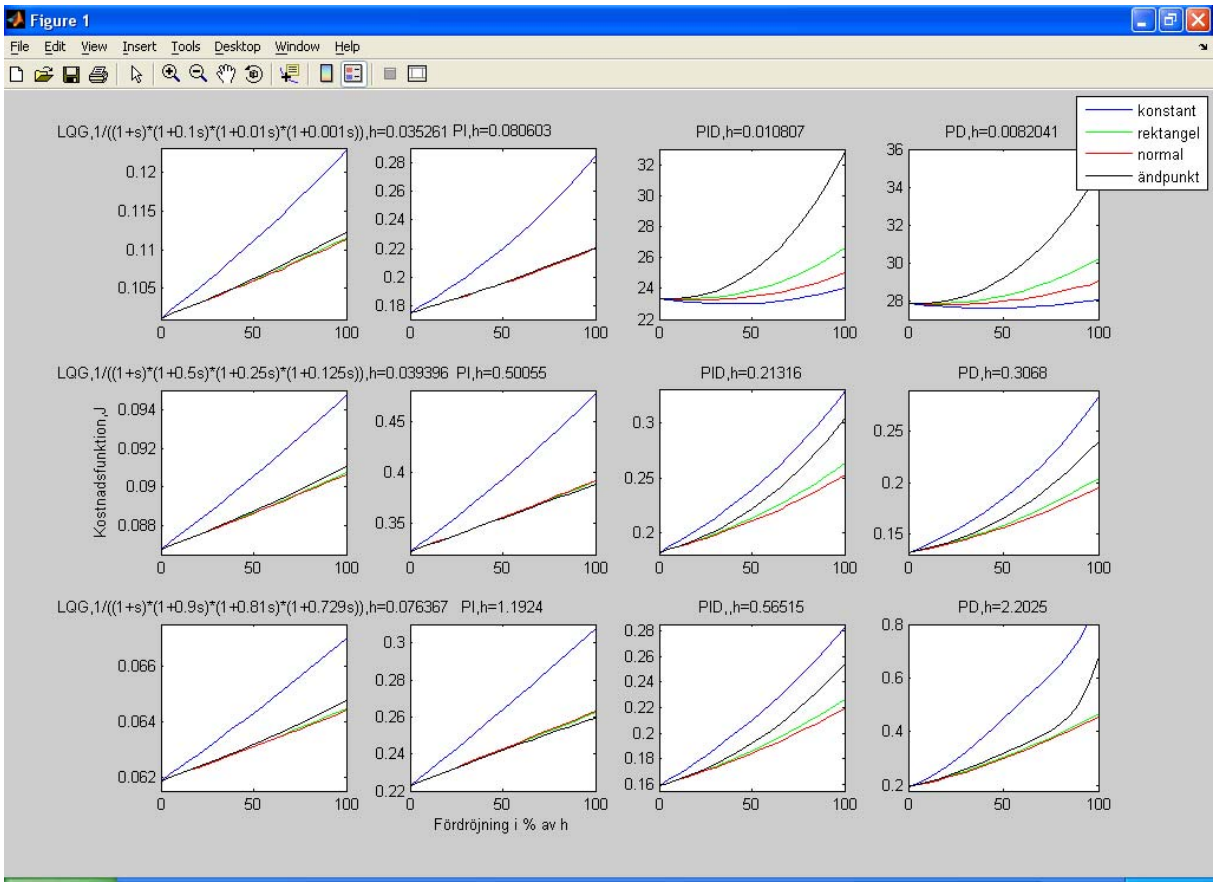
A. Kostnadsfunktion för den kortaste samplingstiden



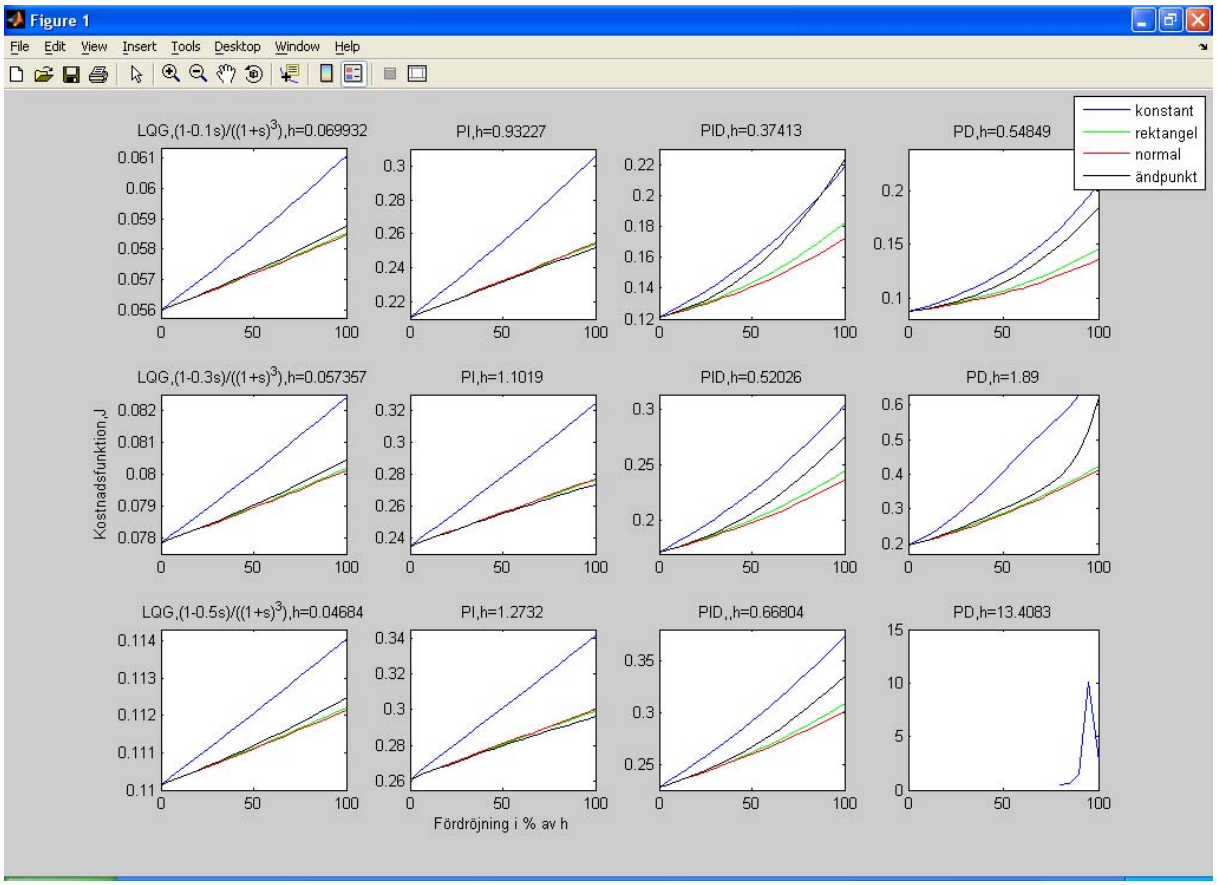
Figur 23 *P_1 med n = 3 i rad 1, n = 4 i rad 2 och n = 5 i rad 3*



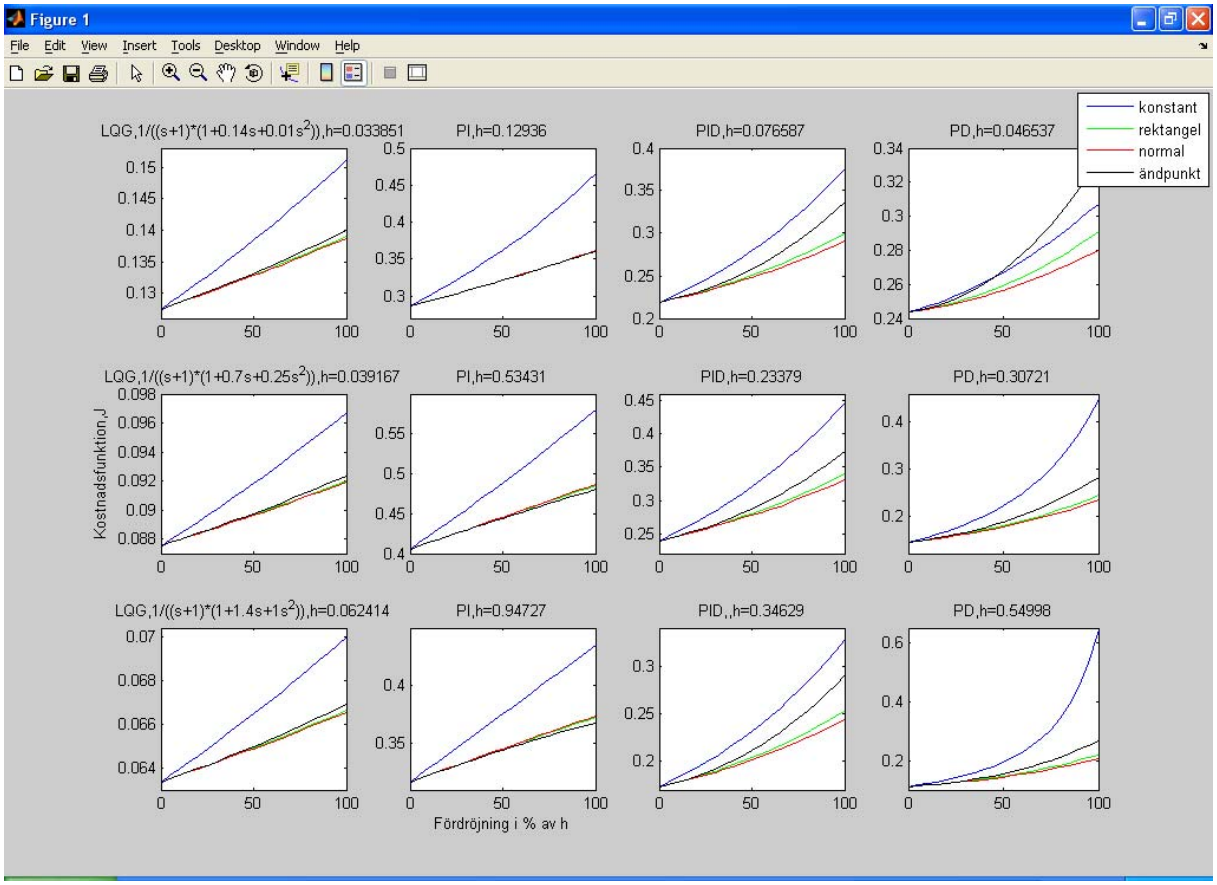
Figur 24 *P_2 med α = 0,01 i rad 1, α = 0,1 i rad 2 och α = 0,5 i rad 3*



Figur 25 *P_3 med α = 0,1 i rad 1, α = 0,5 i rad 2 och α = 0,9 i rad 3*

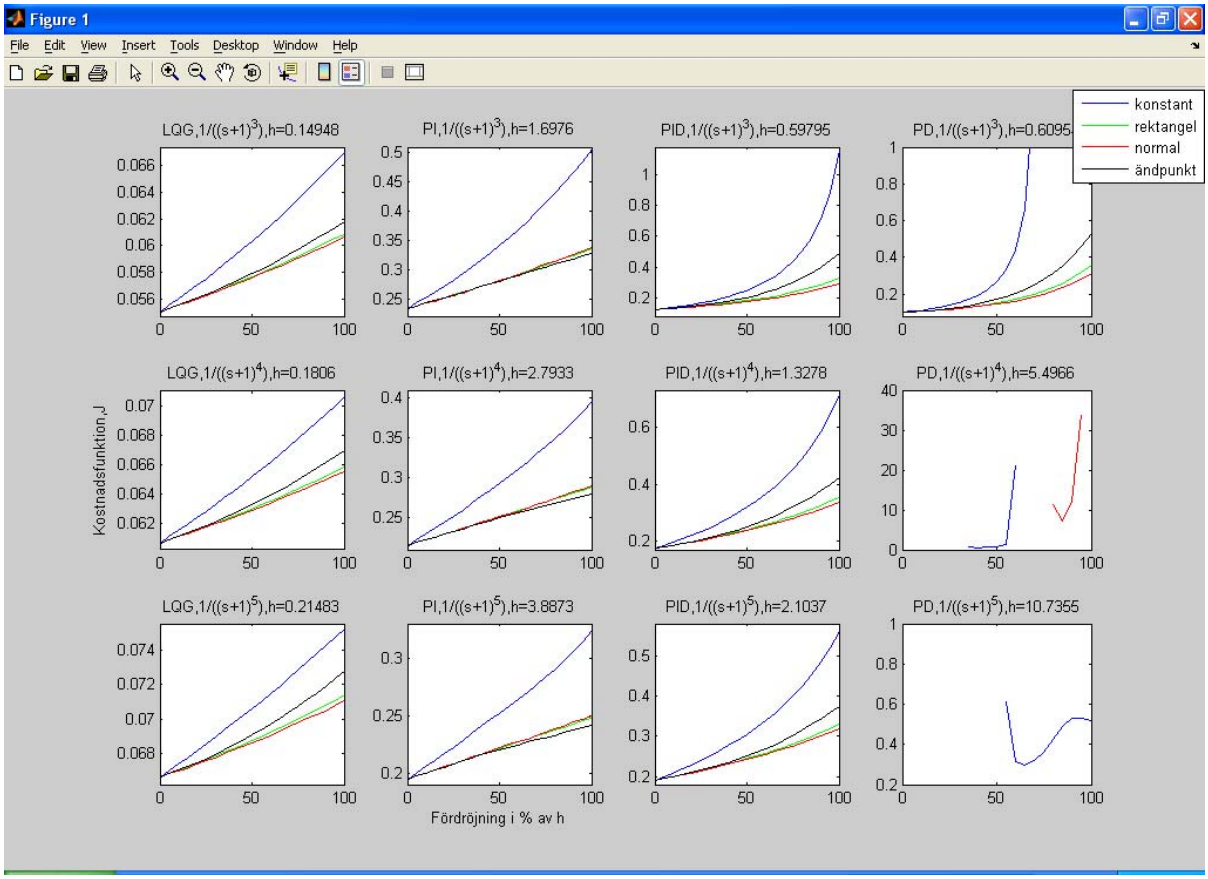


Figur 26 *P_4 med α = 0,1 i rad 1, α = 0,3 i rad 2 och α = 0,5 i rad 3*

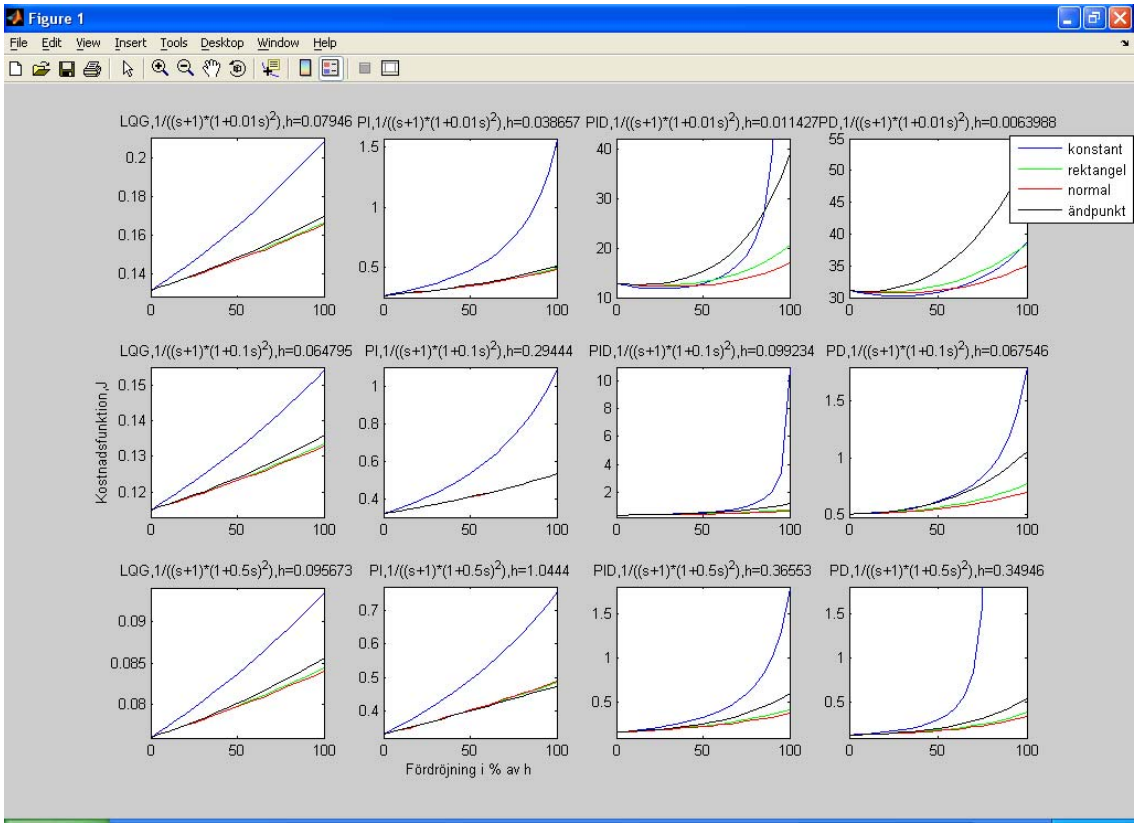


Figur 27 *P_5 med k = 1 i rad 1, k = 5 i rad 2 och k = 10 i rad 3*

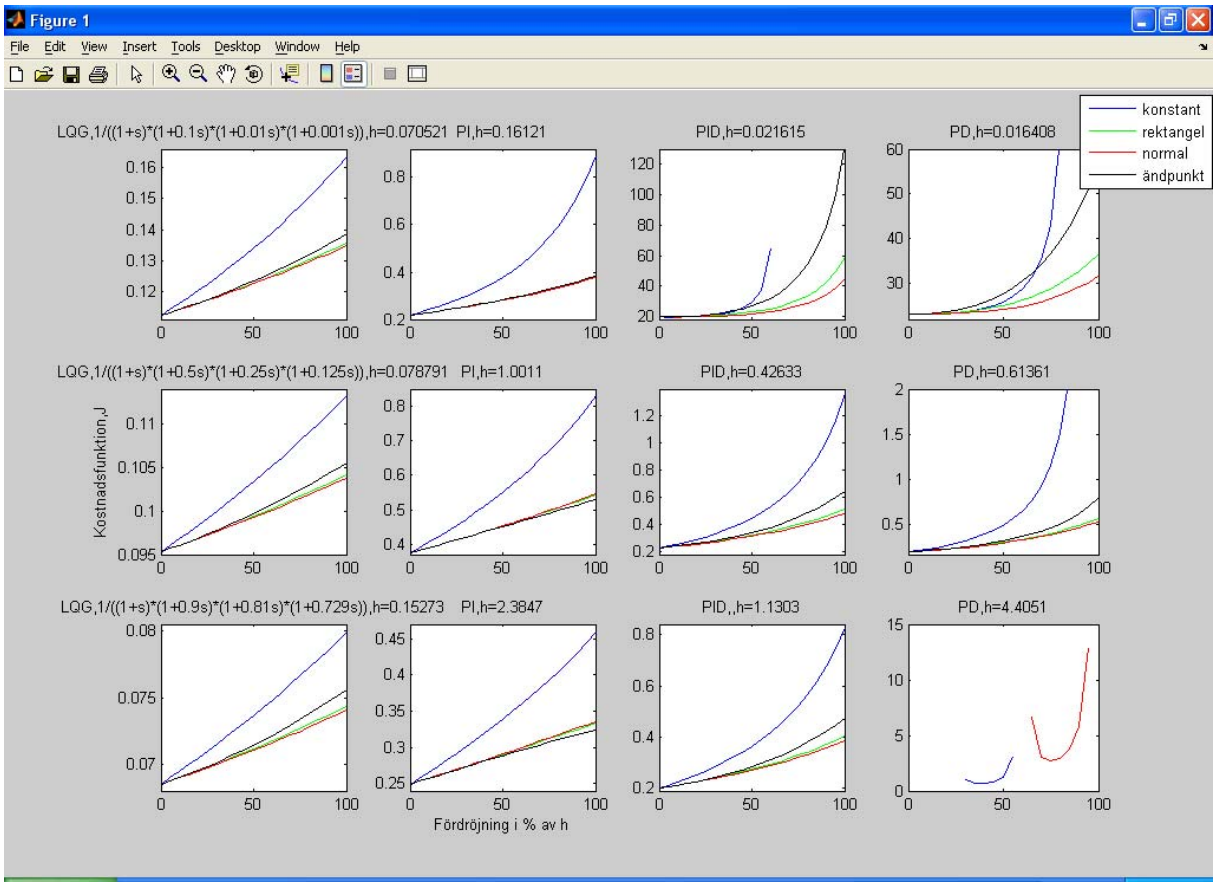
B. Kostnadsfunktion för den mittersta samplingstiden



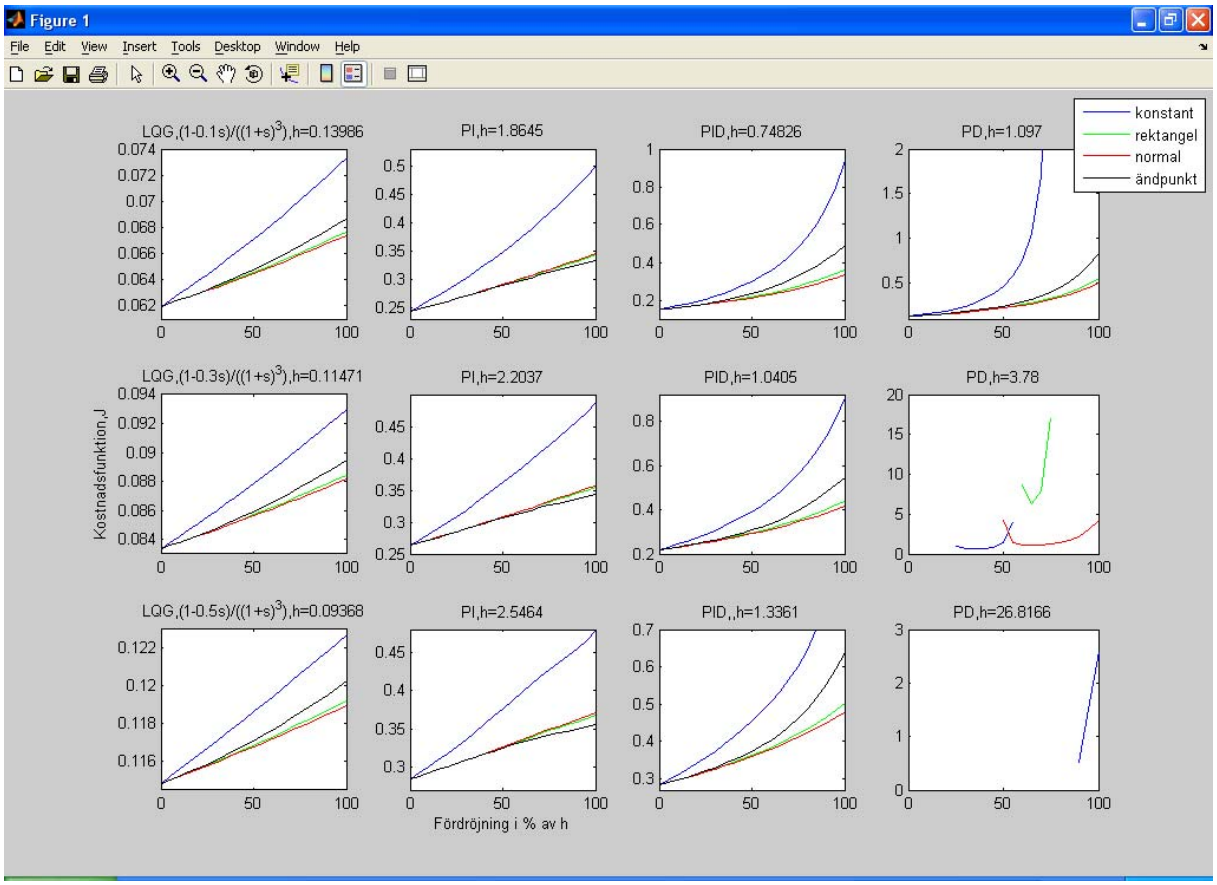
Figur 28 *P_I med n = 3 i rad 1, n = 4 i rad 2 och n = 5 i rad 3*



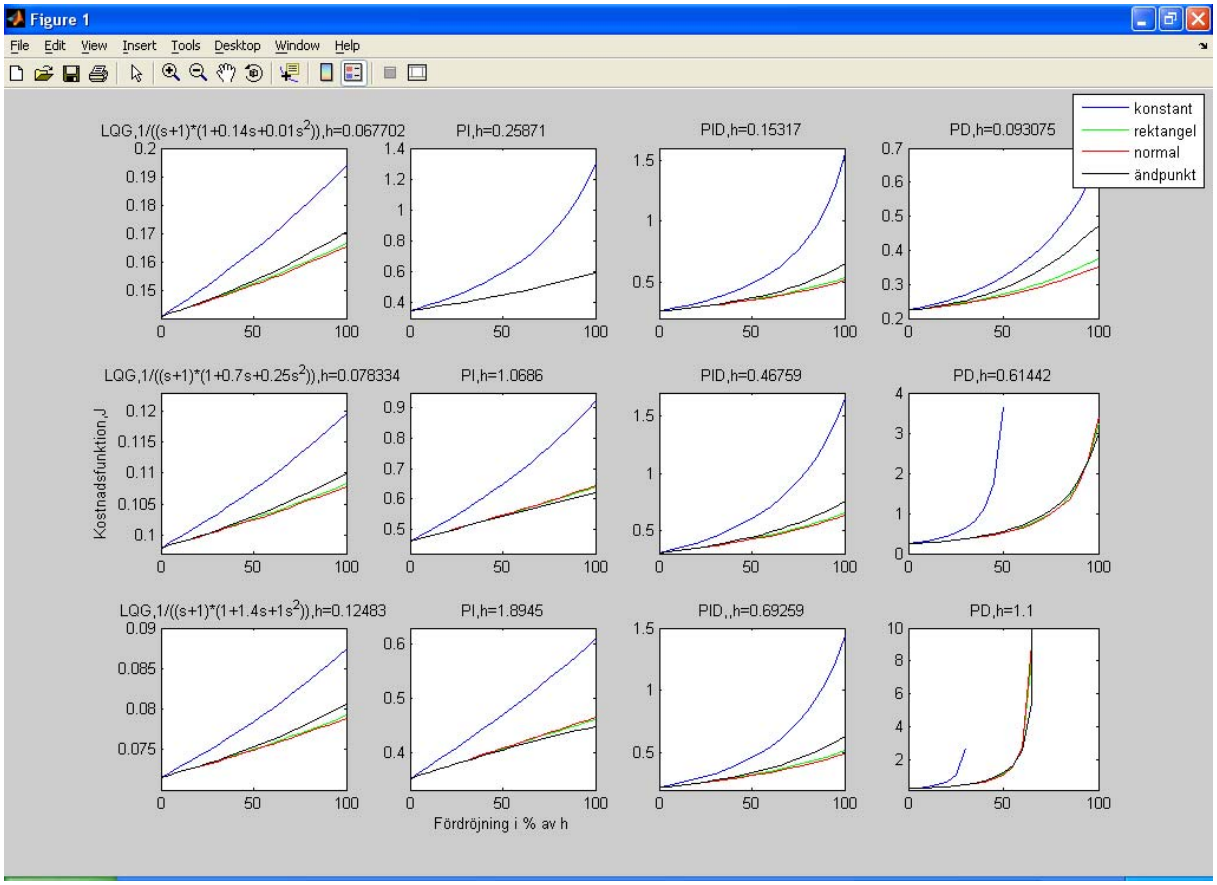
Figur 29 *P_2 med α = 0,01 i rad 1, α = 0,1 i rad 2 och α = 0,5 i rad 3*



Figur 30 *P_3 med α = 0,1 i rad 1, α = 0,5 i rad 2 och α = 0,9 i rad 3*

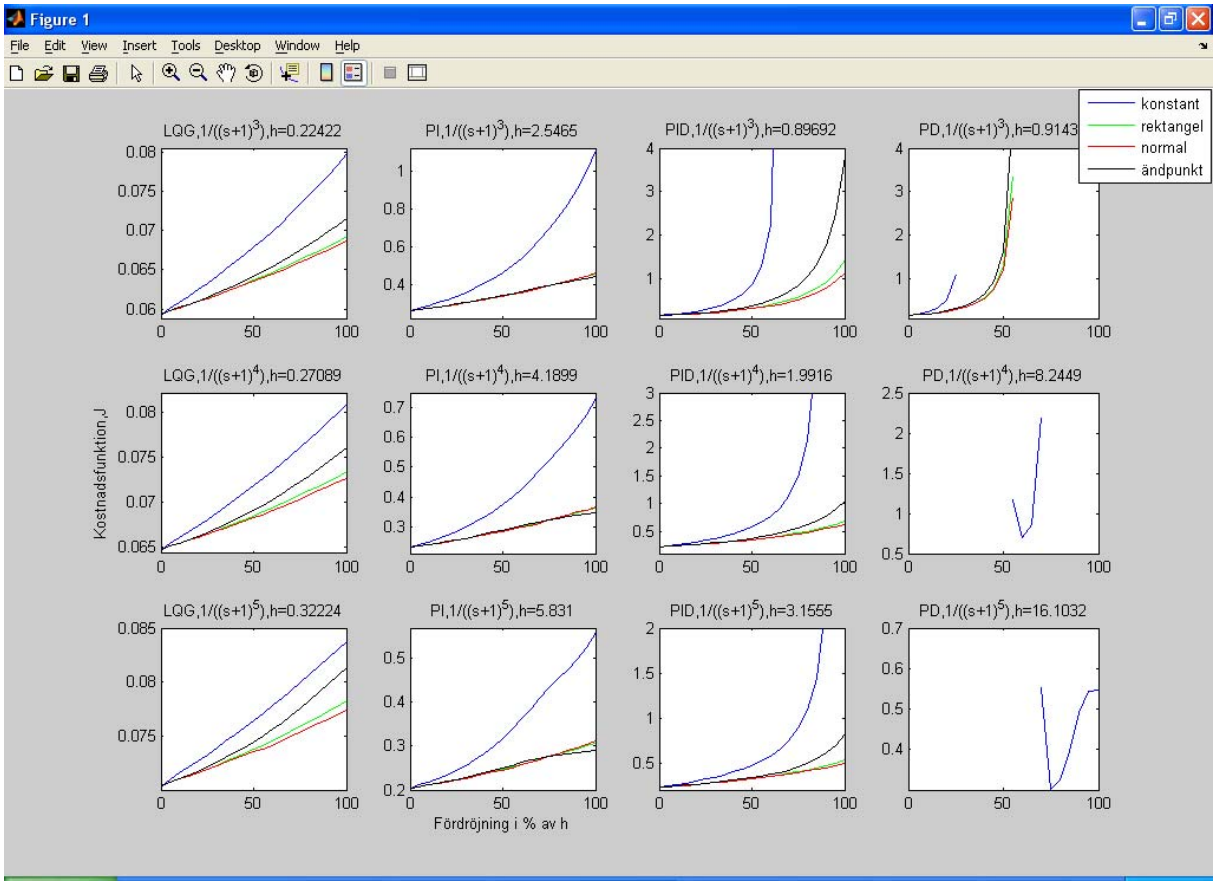


Figur 31 *P_4 med α = 0,1 i rad 1, α = 0,3 i rad 2 och α = 0,5 i rad 3*

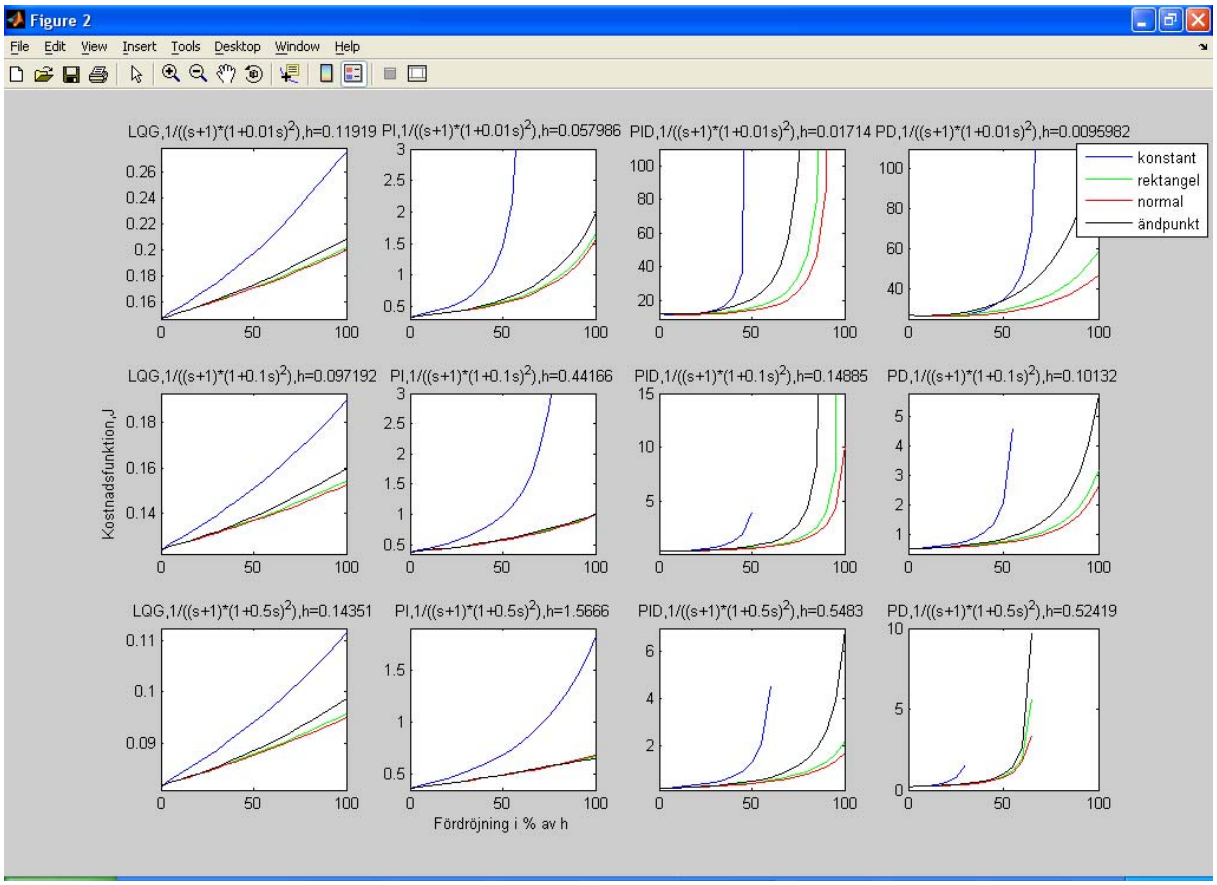


Figur 32 *P_5 med k = 1 i rad 1, k = 5 i rad 2 och k = 10 i rad 3*

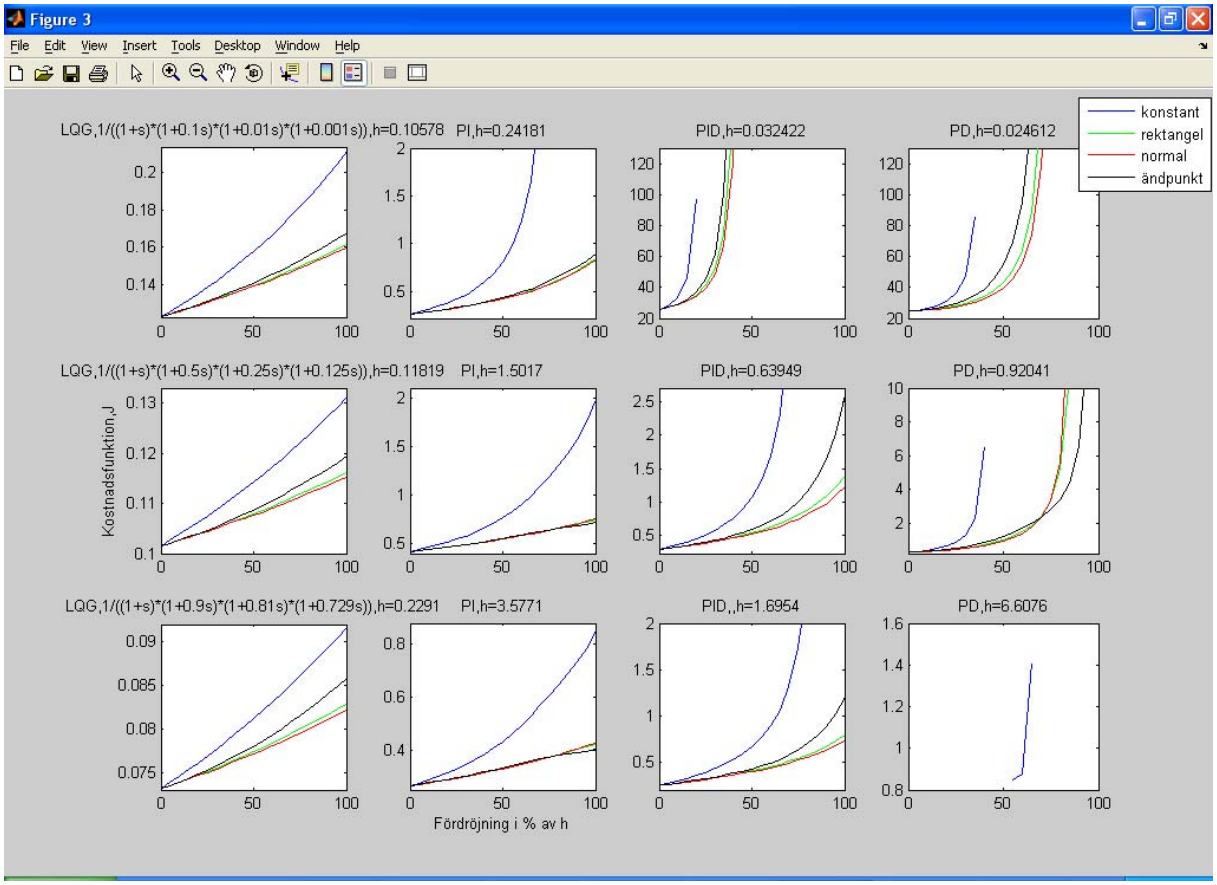
C. Kostnadsfunktion för den längsta samplingstiden



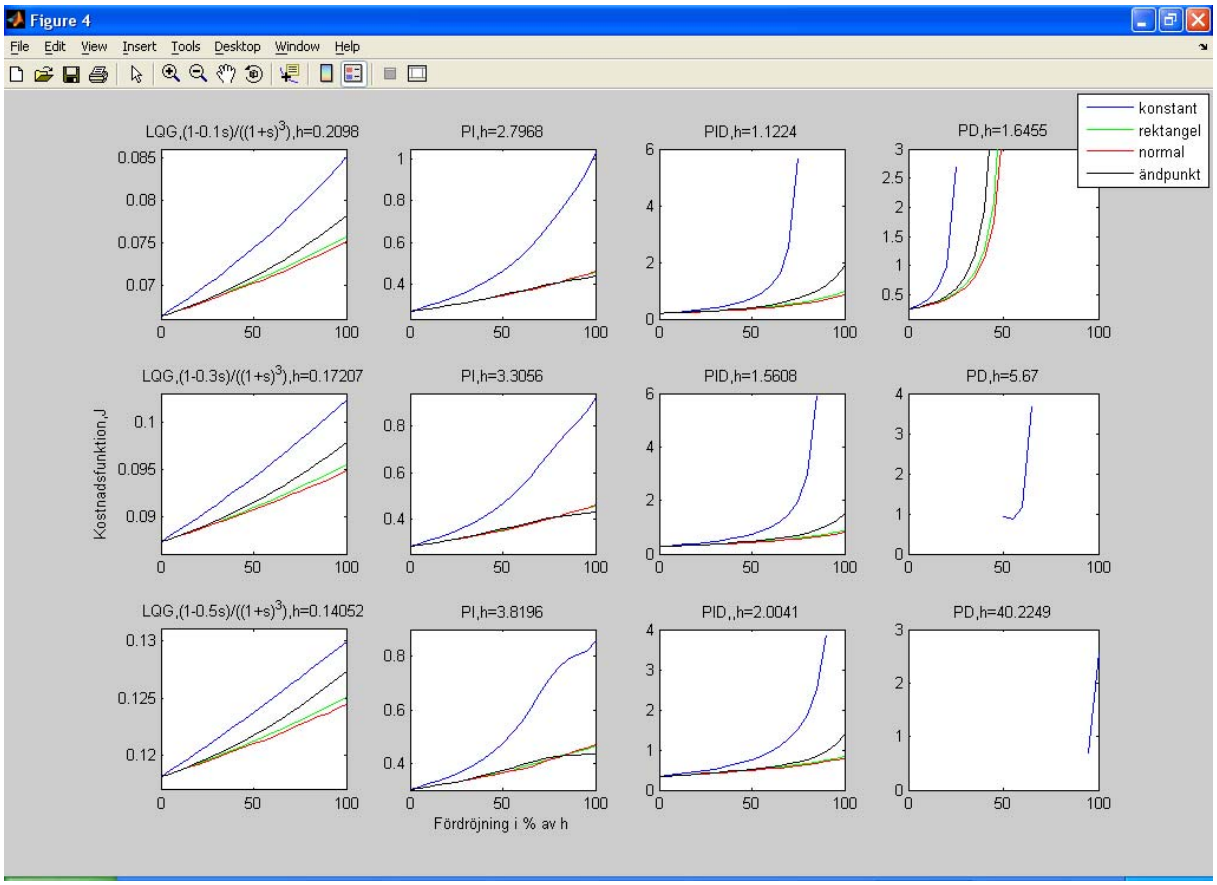
Figur 33 *P_I med n = 3 i rad 1, n = 4 i rad 2 och n = 5 i rad 3*



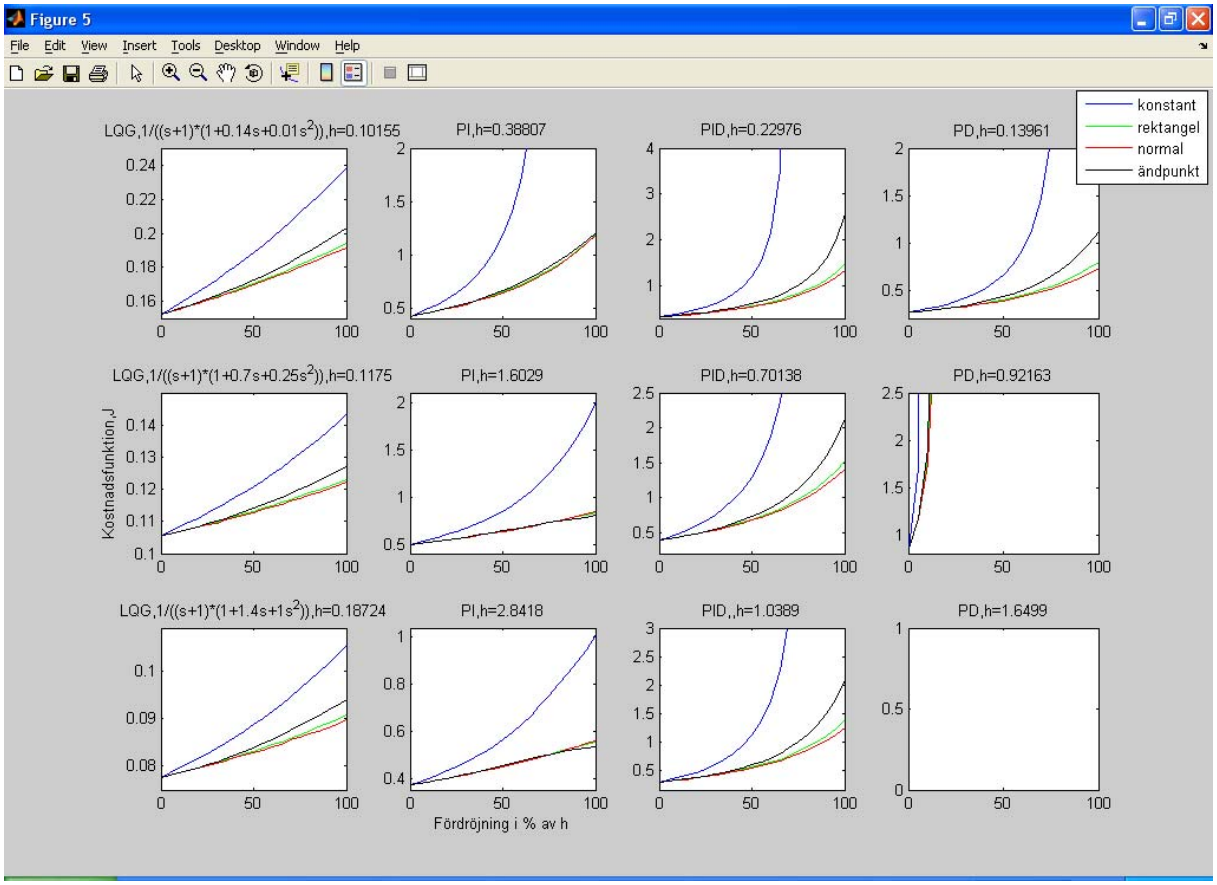
Figur 34 *P_2 med α = 0,01 i rad 1, α = 0,1 i rad 2 och α = 0,5 i rad 3*



Figur 35 *P_3 med α = 0,1 i rad 1, α = 0,5 i rad 2 och α = 0,9 i rad 3*



Figur 36 *P_4 med α = 0,1 i rad 1, α = 0,3 i rad 2 och α = 0,5 i rad 3*



Figur 37 *P_5 med k = 1 i rad 1, k = 5 i rad 2 och k = 10 i rad 3*