



MALMÖ HÖGSKOLA

Inbyggda system och signaler
Reglerteknik

*Kompendium om
reglerteknikutrustningen*

Gion Koch Svedberg
februari 2016

Innehåll

1 Vattentankmodellen	1
1.1 Översikt över Vattentankmodellen.....	1
1.2 Anslutning till vattentankmodellen som reglerobjekt	2
2. Arduino Due.....	3
2.1 Skydda och anpassa analoga ingångarna A0 och A1	3
2.2 Skydda och anpassa analoga utgången på pin='DAC0'	4
2.3 Skydda och anpassa analoga PWM-utgången på pin='DAC1'	5
3. Kommunikation mellan Matlab och Arduino Due.....	5
3.1 Förberedning av Arduino Due.....	5
3.2 Förberedning av Matlab	6
3.3 Arduino-kommandon i Matlab.....	6
4. Matlabprogrammering.....	7
4.1 Realtidskrav på samplingstid	7
4.2 Spara sessionsfiler och variabelfiler.....	8
4.3 Nyttiga tips angående matriser och vektorer.....	10
4.4 Matlab Control Toolbox.....	13

1 Vattentankmodellen

Vi använder oss i reglertekniklabbet av en fysisk vattentankmodell, såsom det är ganska vanligt i reglerteknikkurser världen över. Vattentankmodeller är tacksamma för de är enkla att förstå sig på, har lagom långsamma processförlopp samtidigt som de tillåter tillämpningen av enkla och mycket avancerad teori. På its learning finns en sammanställning av olika vetenskapliga forskningsartiklar som alla använder sig av liknande vattentankmodeller som vi använder oss i våra laborationer.

1.1 Översikt över Vattentankmodellen

Vattentankmodellen består av två vattentankar i serie till varandra med en övre och en nedre behållare, se figur 1.1a nedan. Utflödet av första tanken är inflödet till andra tanken.

Vattennivån i första tanken kan störas genom en separat utgång (röda klammer i bilden nedan).

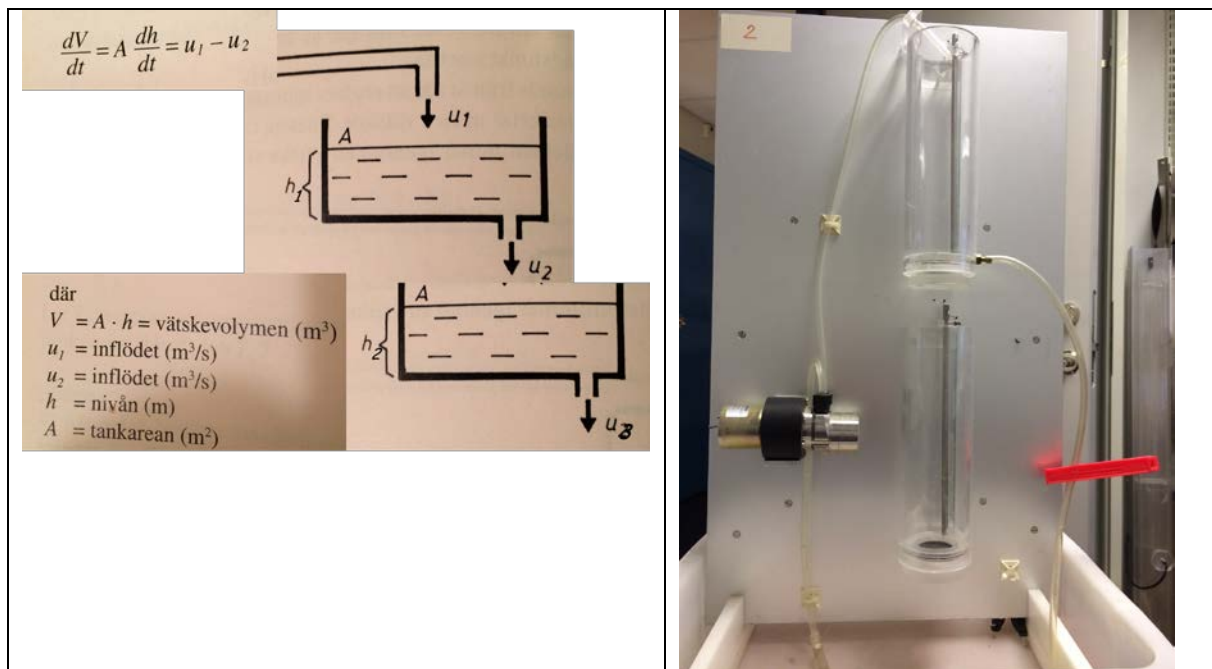


Fig. 1.1a: Illustration och kort av vattentankmodellen

En pump tillför vatten från baljan till första behållaren (tank). Pumpen styrs med en signal mellan 0- ca. 3V. Vattennivåmätare i varje tank genererar en signal mellan 0V (tom tank=0cm) och 10V (full tank=20cm). Anslutningen till pumpen och till nivåmätarna finns via panelen på framsidan, se figuren 1.1b.



Fig. 1.1b: Anslutningar av pumpstyrningen (blå) och nivåmätningarna av över och undre tankarna (blå) samt gemensam jordning (svart).

1.2 Anslutning till vattentankmodellen som reglerobjekt

Bilden nedan försöker visa hur labbtrusningen relaterar till teorin om enkel reglerkretsen. Processen eller reglerobjektet som ska regleras är vattentankmodellen med två behållare och en pump. Nivån i både behållare eller tankar kan mätas i Matlab med hjälp av anslutningar till Arduino Due. Pumpen styrs direkt via Matlab genom pumpens anslutning till Arduino. Dvs att ingen extern spänningskälla eller motor-shield är nödvändig!

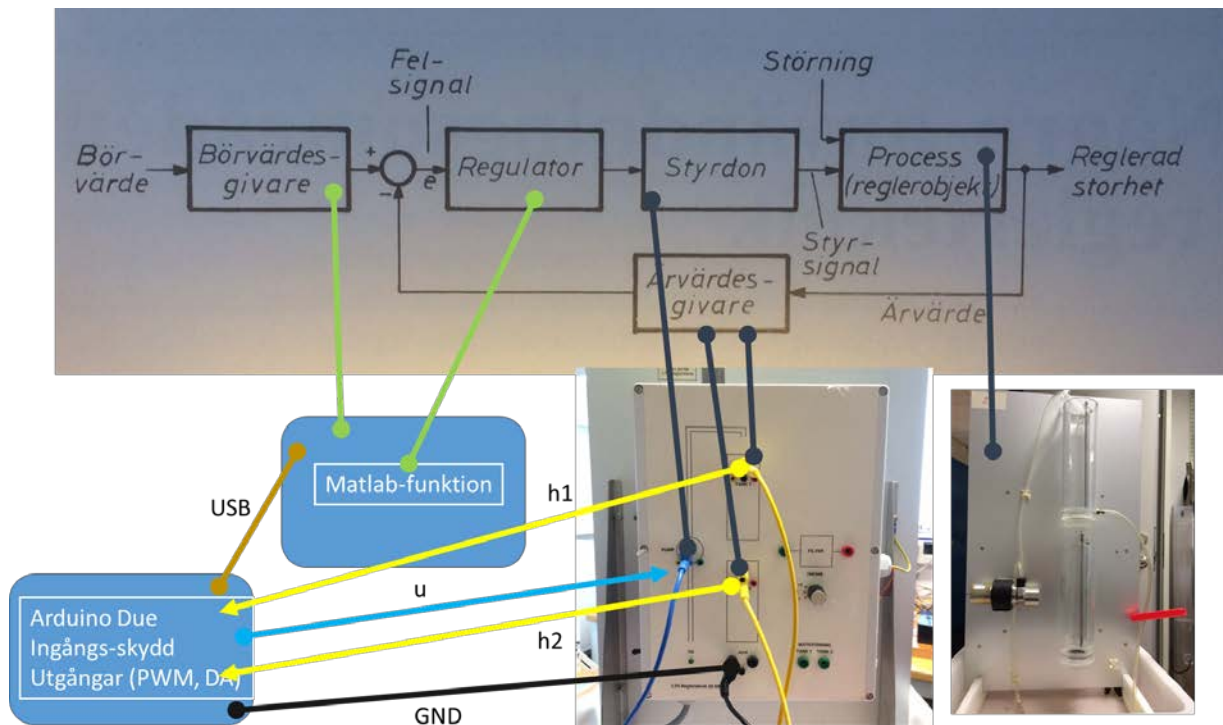


Fig.1.2: Illustration av hela reglerkretsen

2. Arduino Due

2.1 Skydda och anpassa analoga ingångarna A0 och A1

Alla signalerna omkring en Arduino Due behöver begränsas till 3,3V max.

Ingångsspänningarna som fås från vattenmodellen ligger dock mellan 0-10V och motsvarar vattennivåerna i tankarna som vi behöver mäta. Genom en spänningsdelare (seriekoppling av ca 33kOhm och 10kOhm) får man ner signalnivån till 0-3,3V över 10kOhm-resistorn. Innan signalerna läggs på ingångarna A0 och A1 ska de skyddas med op-amp kretsarna CA3240 på liknande sätt som tidigare. Utgångsspänningen vid op-amp ska begränsas vid ca. 2,87V. Testa med voltmeteren först att så är fallet innan ni ansluter kretsen till Arduinon!

Skillnaden i scheman från figur 2.1 nedan jämfört med den som användes i signalbehandlingsdelen är spänningsdelaren samt att kondensatorn tagits bort för att tillåta DC-andelen i signalen.

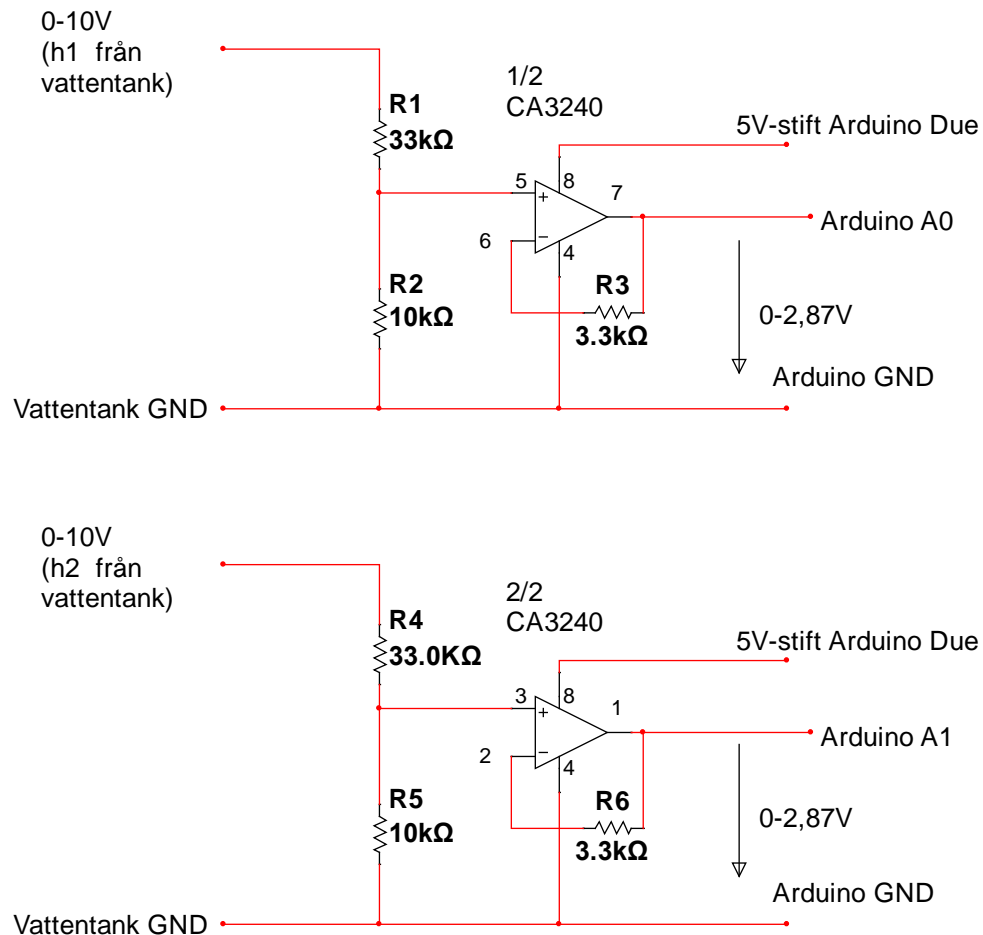
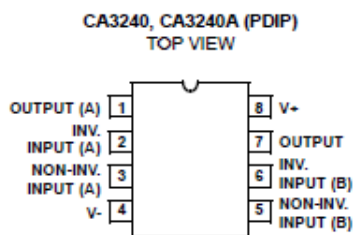


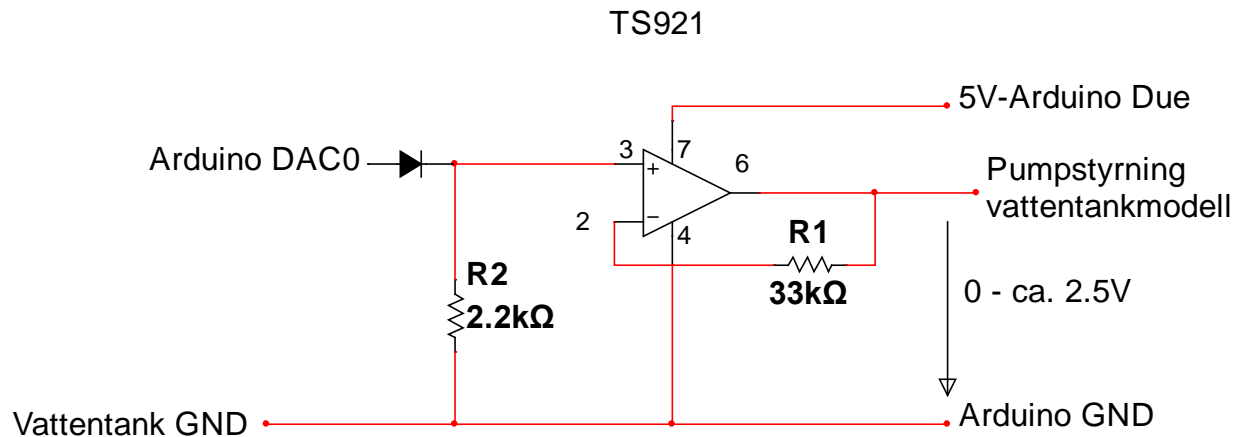
Fig. 2.1: Scheman av skyddselektroniken för Arduino Dues analoga ingångar A0 och A1.

Pinout

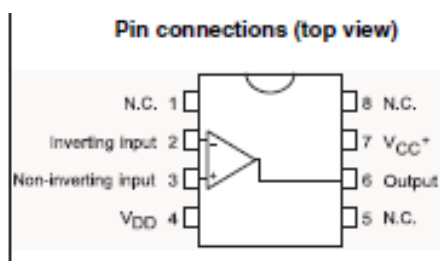


2.2 Skydda och anpassa analoga utgången på pin='DAC0'

AnalogWrite-kommandon på pin 'DAC0' genererar en analog signal mellan 0.55V-2.7V, beroende på 8-bits värdet mellan 0-255, se också kap. 3.3. För att skydda utgången och för att få 0V för värdet 0, ska kretsen med dioden och op-amp TS921 enligt fig 2.2.1 nedan användas.



2.2.1 Förstärkningskrets med diod och op-amp (TS921)



2.3 Skydda och anpassa analoga PWM-utgången på pin='DAC1'

AnalogWrite-kommandon på pin 'DAC1' genererar en PWM signal mellan 0-2.7V, beroende på 8-bits duty-cycle värdet, med 0=0% och 255=100%. För att skydda utgången kan samma krets med op-amp TS921 användas som illustrerad i figuren 2.2.1 ovan. Dioden behövs inte.

3. Kommunikation mellan Matlab och Arduino Due

Matlab och Arduino kommunicerar via USB-comporten med varandra. Man måste först ladda ner ett litet "server"-program till Arduino Due som sedan kör hela tiden. Server-programmet lyssnar på com-porten och tar emot Matlabs kommandon och tolkar dem i en finite state machine med switch/case-anvisningar.

På Matlabsidan byggs upp kommunikationen med Arduinon genom att skapa ett kommunikationsobjekt "a" genom kommandon:

```
>> a=arduino_com('COMx').
```

På its learning finns i mappen "Reglerteknik" ett zip-fil "Matlab-Arduino-mjukvara" med all programvara som behövs. Ladda ner och packa ut zip-filen på datorn först. Följ sedan anvisningarna nedan för att ladda ner server-programmet till Arduino Due och för att kopiera arduino_com funktionen till rätt ställe.

3.1 Förberedning av Arduino Due

Innan Arduino och Matlab kan kommunicera med varandra, behöver man ladda ner en liten kommunikations-server applikation till Arduino Due. Det görs genom följande steg:

- 1 Anslut Arduino Due till PC:n och starta upp Atmel Studio
- 2 Hitta stället med utpackade zip-filen
- 3 Öppna projektet "DueSrv" i Atmel Studio
- 4 Ladda ner projektet till Arduino Due, OBS: Bossac behöver vara installerad på rätt sätt och rätt ställe!
- 5 När allt har fungerat som det ska kan ni stänga Atmel Studio

3.2 Förberedning av Matlab

Matlab skapar ett kommunikationsobjekt när den ska kommunicera med Arduino Due. För detta krävs att rätt programvara finns i användarkatalogen. Genomför följande steg:

- 1 Ta reda på Matlabs aktuella användarkatalog. Byt den till en katalog som du har kontroll över, dvs som inte finns på labdatorns hårddisk (annars kommer allt att försvinna när ni loggar ut). Bra val är M:/-katalogen eller ett USB-minne.
- 2 Kopiera alla Matlabfiler från den utpackade zip-filen till denna arbetskatalog. (Kolla att `arduino_com.m`, och `test_prog.m` finns med).
- 3 Kolla upp på vilken com-port Arduino Due kortet är ansluten till PC/MAC. (T.ex. på PC genom "Enhetshanteraren-> Portar". Till Enhetshanteraren kommer man genom att trycka "<Windows-fönster + X>".
- 4 Testa kommunikationen genom att skriva följande kommandon i Matlabs kommandofönster (med x= nummer av com-porten från steg 3):
`>> a=arduino_com('COMx')`

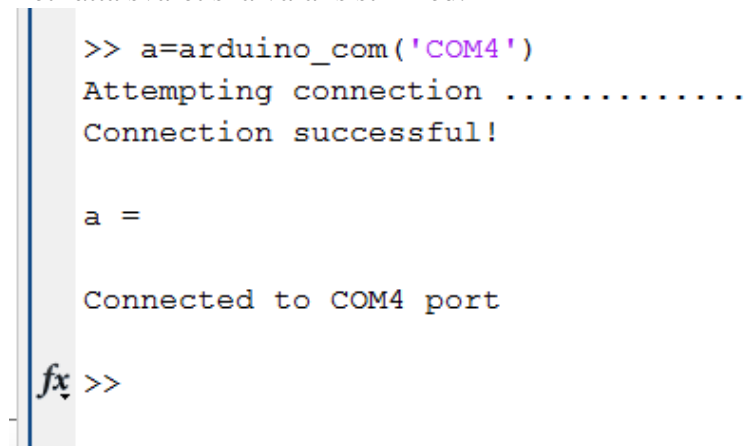
Om ni får följande felmeddelande:

```
>> a=arduino_com('COM4')
```

Undefined function or variable 'arduino_com'.

Så betyder det att Matlab inte hittar `arduino_com.m`-filen i den valda arbetskatalogen. Kopiera dit filen eller ändra arbetskatalogen till rätt ställe.

Det rätta svaret ska vara is stil med:



```
>> a=arduino_com('COM4')
Attempting connection .....
Connection successful!

a =

Connected to COM4 port

fx >>
```

3.3 Arduino-kommandon i Matlab

När allt är på plats ska det gå att kommunicera mellan Matlab och Arduino Due. **Glöm dock inte att skydda analoga in- och utgångar enligt kapitel 2 innan ni sätter igång med dessa kommandon!**

Tabell 2.3 Översikt över Matlab instruktioner

Matlab-instruktion	argument	Beskrivning
<code>pinMode(a,pin,state)</code>	a: arduino-objektet pin: pin-nummer (2-13) state: 'OUTPUT', 'INPUT'	Initialiserar port med pin-nummer till in- eller utgång.
<code>digitalWrite(a,pin,level)</code>	a: arduino-objektet pin: pin-nummer (2-13) level: 0, 1	Sätter en pinne till 0 eller 1. Måste först deklareraras som utgång
<code>digitalRead(a,pin)</code>	a: arduino-objektet pin: pin-nummer (2-13)	Läser av status (digitalt värdet) av en pin. Bör deklareraras som ingång innan den läses.
<code>analogWrite(a, val, pin)</code>	a: arduino-objektet val: 8-bit värde 0..255 Pin: 'DAC1' -> PWM Pin: 'DAC0' -> DAC	PWM signal ('DAC1') eller analog DAC ('DAC0')-utgång. Välj vad som ska gälla genom val av pin.
<code>analogRead(a,pin)</code>	a: arduino-objektet pin: 'A0', 'A1', ('A3')	Läser av en analog pinne. Skall ej deklareraras som ingång innan läsning. AD-upplösning 1024bit

4. Matlabprogrammering

4.1 Realtidskrav på samplingstid

Det finns en missuppfattning att realtidskrav betyder att ett system ska vara så snabbt som möjligt. Inom reglerteknik betyder det dock att samplingstiden är alltid likadant samt att systemet kan garantera en svarstid, oftast inom en eller två samplingstidpunkter. (Se https://en.wikipedia.org/wiki/Real-time_computing för en överblick över begreppet).

För att uppnå realtidskraven i våra Matlabprogram använder vi följande struktur i exekveringscykel:

```
function .....
```

```
% förklaring av funktionen med alla argument (t.ex. börvärdet r) och resultat (t.ex. y)
```

```
% Initialisering av variablerna
```

```
N= ... % antal samplingar
```

```
Ts= ... % samplingstid i sekunder
```

```
e=zeros(1, N);
```

```
h1=zeros(1, N);
```

```
u=zeros(1, N);
```

```
y=zeros(1, N);
```

```
ok=0;
```



```

...
% Konfigurering av in- och utgångar
...

for k=1:N % slinga kommer att köras N-gångar, varje gång tar exakt Ts-sekunder

    start = cputime; %startar en timer för att kunna mäta tiden för en loop
    if ok < 0 % testar om samplingen är för kort
        k % sampling time too short!
        disp('samplingstiden är för lite! Ök värdet för Ts');
        return
    end

    % läs in sensorvärden
    h1(k)= analogRead(a, 'A0'); % mät nivån i behållaren 1

    % beräkna något, t.ex. styrvärdet u(k)
    e(k)=r-h1(k);
    u(k)=kp*e(k); %p-regulator

    % begränsa styrvärdet till lämpliga värden (t.ex. >0 och <255, samt heltal)
    u(k)=min(max(0, round(u(k)), 255);

    % skriva ut styrvärdet
    analogWrite(a,u(k), 'DAC0');

    elapsed=cputime-start; % räknar åtgångsen tid i sekunder
    ok=(Ts-elapsed); % sparar tidsmarginalen i ok

    pause(ok); %pausar resterande samplingstid
end % -for

```

4.2 Spara sessionsfiler och variabelfiler

Spara variablerna i en fil med namn "filnamn":

```
>> help save
```

save Save workspace variables to file.

save(FILENAME) stores all variables from the current workspace in a MATLAB formatted binary file (MAT-file) called FILENAME.

save(FILENAME,VARIABLES) stores only the specified variables.

save(FILENAME, ..., '-append') adds new variables to an existing file.

You can specify '-append' with additional inputs such as VARIABLES, '-struct', FORMAT, or VERSION.

save(FILENAME, ..., FORMAT) saves in the specified format: '-mat' or '-ascii'.

Inputs:

FILENAME: If you do not specify FILENAME, the save function saves to a file named matlab.mat. If FILENAME does not include an extension and the value of format is '-mat' (the default), MATLAB appends .mat. If filename does not include a full path, MATLAB saves in the current folder. You must have permission to write to the file.

VARIABLES: Save only selected variables from the workspace. Use one of the following forms:

V1, V2, ... Save the listed variables. Use the '*' wildcard to match patterns. For example, save('A*') saves all variables that start with A.

Examples:

```
% Save all variables from the workspace to test.mat:
save test.mat
```

```
% Save two variables, where FILENAME is a variable:
savefile = 'pqfile.mat';
p = rand(1, 10);
q = ones(10);
save(savefile, 'p', 'q');
```

```
% Save the fields of a structure as individual variables:
s1.a = 12.7;
s1.b = {'abc', [4 5; 6 7]};
s1.c = 'Hello!';
save('newstruct.mat', '-struct', 's1');
```

```
% Save variables whose names contain digits:
save myfile.mat -regexp \d
```

See also load, matfile, whos, regexp, hgsave, saveas, workspace, clear.

Reference page for save
Other functions named save

```
>> help load
```

load Load data from MAT-file into workspace.

S = load(FILENAME) loads the variables from a MAT-file into a structure array, or data from an ASCII file into a double-precision array.

S = load(FILENAME, VARIABLES) loads only the specified variables from a MAT-file. VARIABLES use one of the following forms:

VAR1, VAR2, ... Load the listed variables. Use the '*'

wildcard to match patterns. For example, `load('A*')` loads all variables that start with A.

Notes:

If you do not specify `FILENAME`, the load function searches for a file named `matlab.mat`.

Examples:

```
gongStruct = load('gong.mat')    % All variables
load('handel.mat', 'y')         % Only variable y
load('accidents.mat', 'hwy*')   % Variables starting with "hwy"
load('topo.mat', '-regexp', '\d') % Variables containing digits
```

`>> help diary`

`diary` Save text of MATLAB session.

`diary FILENAME` causes a copy of all subsequent command window input and most of the resulting command window output to be appended to the named file. If no file is specified, the file `'diary'` is used.

`diary OFF` suspends it.

`diary ON` turns it back on.

`diary`, by itself, toggles the diary state.

Use the functional form of `diary`, such as `diary('file')`, when the file name is stored in a string.

See also `save`.

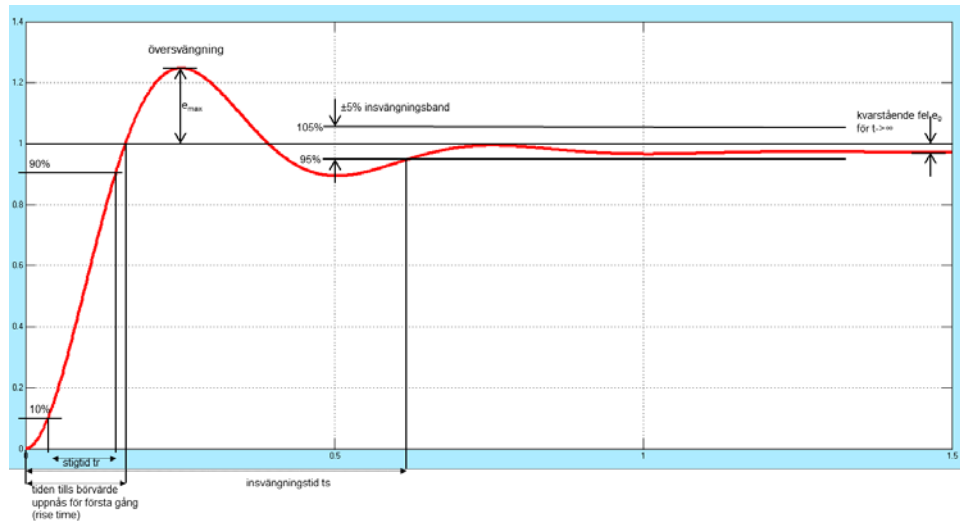
Reference page for `diary`

4.3 Nyttiga tips angående matriser och vektorer

”>>” betyder Matlab-prompten i command window

Antagande:

- `h2` är en vektor med alla samplade mätningar för nivån i tank 2
- `b` är börvärde för nivån i tank 2
- `N` är antal samples i `h2`
- `h2` har samplats tillräckligt länge så att signalen till slut är nästan konstant
- `dT` samplingstid



Följande uppskattningar beror på hur signalerna ser ut. Ibland finns mätfel och andra störningar som ställer till det. I så fall måste mätfel först tas bort eller filtreras bort eller uppskattningarna måste göras från hand.

Uppskattning av **maximal översvängning** (= maximal värde i h_2 – börvärde b)

```
>> max(h2)-b % om det är negativ så finns ingen översvängning
```

Uppskattning av **stigtid t_r** (tiden det tar för signalen att stiga från 10% av börvärdet till 90%)

```
>> s = find((h2 >= b*0.1) & (h2 <= b*0.9)) % samplingar när h2 är mellan 10% och 90%
```

```
>> tr = (max(s) - min(s)) * dT % stigtid i sekunder
```

Uppskattning av **rise time t_r** (tiden det tar för signalen från nullpunkten till börvärde)

```
>> find(h2 >= b) % samples med värden av h2 lika med eller högre än börvärde. Om resultatet är null så betyder det att signalen aldrig når upp till börvärde
```

```
>> rt = min(find(h2 >= b)) * dT
```

Uppskattning av **insvängningstid t_s** (tiden det tar tills signalen är i en intervall mellan 95% och 105% av börvärdet)

```
>> it = min(find((h2 >= b*0.95) & (h2 <= b*1.05))) * dT % hittar första sample som uppfyller insvängningsvillkoren, multiplicera samplenummer med samplingstiden
```

Uppskattning av **kvarstående fel** (skillnaden av insvängde, stationär signalen till börvärde)

```
>> mean(h2(length(h1)-15:length(h2))) - b % ta medelvärde av sista 16 samples minus börvärde
```

Description

The colon is one of the most useful operators in MATLAB®. It can create vectors, subscript arrays, and specify for iterations.

The colon operator uses the following rules to create regularly spaced vectors for scalar values i , j , and k :

$j:k$

is the same as $[j, j+1, \dots, k]$, or empty when $j > k$.

You can use the colon to create a vector of indices to select rows, columns, or elements of arrays, where:

$A(:,j)$

is the j th column of A .

$A(i,:)$

is the i th row of A .

$A(:, :)$

is the equivalent two-dimensional array. For matrices this is the same as A .

$A(j:k)$

is $A(j)$, $A(j+1)$, ..., $A(k)$.

$A(:,j:k)$

is $A(:,j)$, $A(:,j+1)$, ..., $A(:,k)$.

$A(:)$

is all the elements of A , regarded as a single column. On the left side of an assignment statement, $A(:)$ fills A , preserving its shape from before. In this case, the right side must contain the same number of elements as A .

When you create a vector to index into a cell array or structure array (such as `cellName{ : }` or `structName(:).fieldName`), MATLAB returns multiple outputs in a comma-separated list. For more information, see [How to Use the Comma-Separated Lists in the MATLAB Programming Fundamentals documentation](#).

<code>A=zeros(n)</code>	ger en $n \times n$ -matris med nollor
<code>A=zeros(m,n)</code>	ger en $m \times n$ -matris med nollor
<code>B=ones(n)</code>	ger en $n \times n$ -matris med ettor
<code>B=ones(m,n)</code>	ger en $m \times n$ -matris med ettor
<code>C=eye(n)</code>	ger en enhetsmatris av dimensionen $n \times n$
<code>d=rand</code>	ger ett rektangulärfördelat slumptalsvärde i intervallet $[0,1]$
<code>d=randn</code>	ger ett normalfördelat slumptalsvärde med medelvärde 0 och standardavvikelse 1.
<code>D=rand(n)</code>	ger en $n \times n$ -matris med slumptalsvärden.
<code>a=diag(A)</code>	ger en kolonnvektor a med diagonalelementen i matrisen A .
<code>B=inv(A)</code>	inversen av matrisen A .
<code>f=det(A)</code>	determinanten av matrisen A .
<code>G=eig(A)</code>	egenvärdena till matrisen A .

4.4 Matlab Control Toolbox

Följande utdrag är från kap. 24 i kursboken.

24.2 Matlab Control Toolbox

Matlab Control Toolbox är ett tilläggspaket till Matlab, som innehåller extra kommandon avsedda för reglertekniska beräkningar. Paketet behandlar såväl tidskontinuerliga som tidsdiskreta system. Det behandlar också såväl system beskrivna med överföringsfunktioner, som system beskrivna på tillståndsform. I detta avsnitt ges en mycket kort introduktion till programmet, samt ett antal enkla exempel. Observera att alla kommandon som beskrivs här finns utförligare beskrivna i den inbyggda hjälpfunktionen i Matlab-programmet. Genom att ge kommandot » `help feedback` fås ytterligare information om kommandot `feedback`.

Tidsdiskreta system

Matlab Control Toolbox kan också arbeta med tidsdiskreta system och även transformera mellan tidskontinuerliga och tidsdiskreta modeller. De flesta kommandon som vi tidigare diskuterat fungerar på samma sätt i det tidsdiskreta fallet och upprepas därför inte här. Vi ger bara några exempel på tidsdiskreta kommandon i sammanställningen på nästa sida.

Inmatning av överföringsfunktioner och block-schematransformering

Överföringsfunktioner matas in med hjälp av kommandot `tf` (transfer function). I argumentet till detta kommando ges först koefficienterna i täljaren, därefter koefficienterna i nämnaren i form av vektorer (med hakparenteser). Börja med den koefficient som har högst gradtal. Nedanstående exempel visar inmatning av följande två överföringsfunktioner:

$$G_1 = \frac{s+2}{s^2+s+10} \quad \text{och} \quad G_2 = \frac{2s+3}{s+2}$$

Som namn på överföringsfunktioner och konstanter kan man använda godtyckliga textsträngar upp till 19 tecken långa, bildade av små och stora bokstäver eller siffror. Det första tecknet måste dock vara en bokstav. Med enkla kommandon kan man sedan bestämma överföringsfunktionen för seriekopplade, parallellkopplade och återkopplade system. Vi visar detta med några exempel:

<p>» <code>G1=tf([1 2],[1 1 10])</code></p> <p>Transfer function:</p> $\frac{s+2}{s^2+s+10}$	<p>Inmatning av överföringsfunktionen G1</p> <p>Svar från Matlab</p>
<p>» <code>G2=tf([2 3],[1 2])</code></p> <p>Transfer function:</p> $\frac{2s+3}{s+2}$	<p>Inmatning av G2</p> <p>Svar från Matlab</p>
<p>» <code>G3=G1*G2</code></p> <p>Transfer function:</p> $\frac{2s^2+7s+6}{s^3+3s^2+12s+20}$	<p>Seriekoppling av G1 och G2</p> <p>Svar från Matlab</p>
<p>» <code>H=G1+G2</code></p> <p>Transfer function:</p> $\frac{2s^3+6s^2+27s+34}{s^3+3s^2+12s+20}$	<p>Parallellkoppling av G1 och G2</p> <p>Svar från Matlab</p>
<p>» <code>H=feedback(G1,G2)</code></p> <p>Transfer function:</p> $\frac{s^2+4s+4}{s^3+5s^2+19s+26}$	<p>G1 återkopplad med G2 (som default används negativ återkoppling)</p> <p>Svar från Matlab</p>

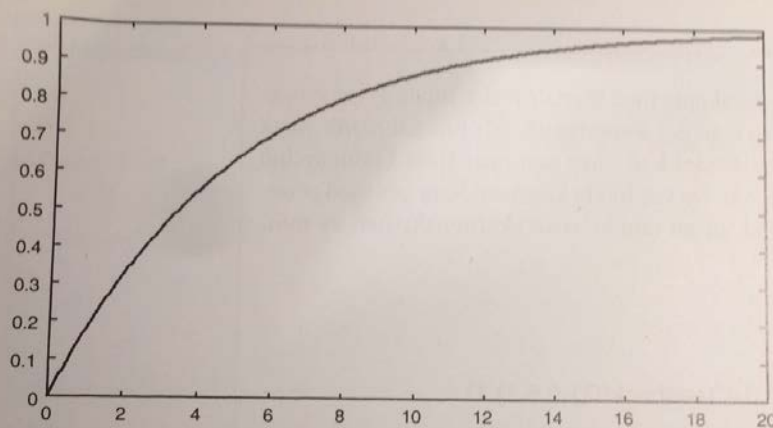
Andra närbesläktade kommandon:

<code>poly(x)</code>	ger ett polynom av grad n , dvs en vektor av grad $n + 1$, bildat av rötterna givna i vektorn x . Här är n längden på vektorn x .
<code>series(..)</code>	annat kommando för seriekoppling av överföringsfunktioner.
<code>parallel(..)</code>	annat kommando för parallellkoppling av överföringsfunktioner.
<code>conv(p,q)</code>	multipliserar polynomen p och q med varandra.

Tidsförlopp, stegsvarsberäkning med mera

Med enkla kommandon kan man bestämma stegsvaret, impulssvaret m m för olika system. Vi visar detta med några exempel. Vi börjar med att mata in en ny överföringsfunktion $G1$. Stegsvaret erhålls sedan genom att ge kommandot `step(G1,t)`, där parametern t bestämmer skalningen (dvs maxvärdet) på tidsaxeln. Resultatet ges av följande kurva.

- » `G1=tf([1],[5 1]);` Inmatning av överföringsfunktionen $G = 1/(1+5s)$.
- » `step(G1,20)` Beräkning och plotning av stegsvaret upp till $t = 20$.



I stället för ovanstående kommando kan man också skriva enbart `step(G1)` för att erhålla stegsvaret. I detta fall bestämmer dock systemet skalningen (slutvärdet) på tidsaxeln.

Impulssvaret fås på liknande sätt med kommandot `impulse(G1,t)` eller kommandot `impulse(G1)`, se nedan. Däremot finns det inget "direkt-kommando" för att beräkna rampsvaret. För att göra detta måste man först skapa en vektor med insignalen (en rampfunktion). Därefter används kommandot `lsim` för att beräkna hur utsignalen ser ut till följd av den aktuella insignalen. (Kommandot `lsim` används för att beräkna utsignalen för ett visst system till följd av godtycklig insignal.) Se nedan:

» <code>t=[0:0.05:20];</code>	Skapande av tidsvektor för vilken rampsvaret ska beräknas. Tidsvektorn går från $t = 0$ till $t = 20$ med intervallet 0.05.
» <code>ramp=2*t;</code>	Skapande av insignalvektor (en rampfunktion med lutningen 2).
» <code>y=lsim(G1,ramp,t);</code>	Skapande av vektor med utsignalvärden, följt av plottning av utsignalen (rampsvaret).
» <code>plot(t,y)</code>	

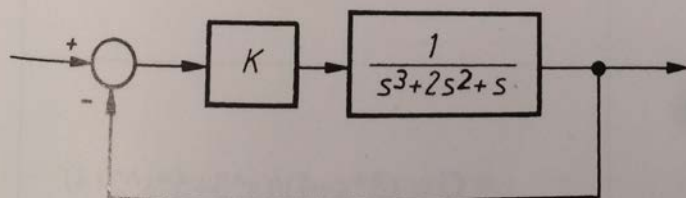
Andra närbesläktade kommandon:

<code>[amp,fas]=bode(G1);</code>	ger en matris med amplitud- och fasvärden.
<code>nyquist(G1)</code>	ger Nyquistdiagrammet.
<code>[z,p,k]=tf2zp([1 2],[1 1 10])</code>	ger nollställena och poler till ett system.
<code>freqresp(G,w)</code>	ger frekvensfunktionens värde vid aktuell frekvens.
<code>pzmap(G)</code>	ger en pol-nollställeskarta för överföringsfunktionen G .

Beräkning av kurvskaror

I många fall vill man beräkna kurvskaror för olika system, t ex hur stegsvaret förändras då förstärkningen i ett reglersystem varierar. För att kunna plotta flera kurvor i samma diagram måste kommandot `hold` användas. Kommandot `göör` att gamla kurvor i ett diagram sparas och att upprepad plottning gör att nya kurvor läggs ovanpå de gamla.

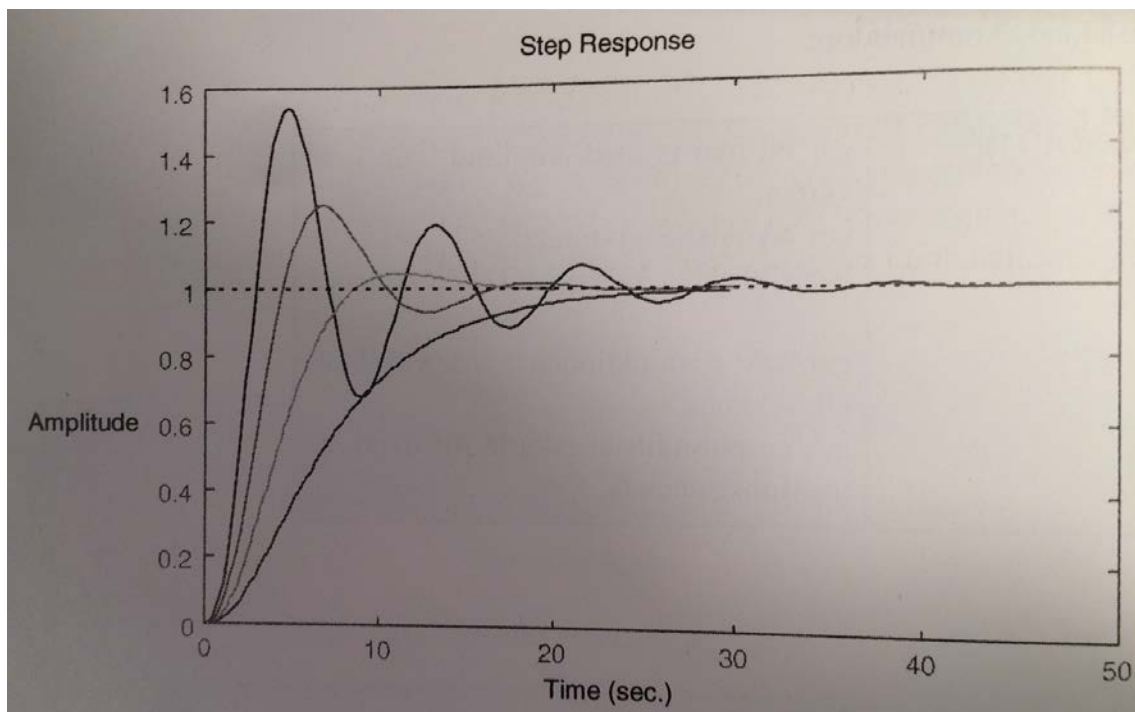
Nedanstående kommandosekvens visar hur man tar fram en kurvskara för följande system då förstärkningen i regulatorn är $K = 0,125$; $K = 0,25$; $K = 0,5$ resp $K = 1$.



```

» G=tf([1],[1 2 1 0]);
» GT=feedback(G,1);
» step(GT)                                (ger första kurvan)
» hold
Current plot held                          (svar från Matlab)
» GT=feedback(0.5*G,1);
» step(GT)                                (ger andra kurvan)
» GT=feedback(0.25*G,1);
» step(GT)                                (ger tredje kurvan)
» GT=feedback(0.125*G,1);
» step(GT)                                (ger fjärde kurvan)

```



Nya Matlab-kommandon, processer med dödtid mm.

De senaste versionerna av Matlab erbjuder ett alternativt och ofta bekvämare sätt att mata in överföringsfunktioner enligt nedan. Med dessa versioner kan man även arbeta med processer med dödtid.

Definition av laplace-variabeln s (Detta måste göras först innan man kan mata in överföringsfunktioner enligt nedan)	» $s = tf('s')$
Inmatning av överföringsfunktioner i Matlab Exempel: $G = \frac{3s + 4}{s^3 + 5s^2 + 4}$	» $G = (3*s+4)/(s^3+5*s^2+4)$
Inmatning av process med dödtid Exempel: $G = \frac{5e^{-2s}}{1 + 4s}$	» $G = 5*\exp(-2*s)/(1+4*s)$
Matlab-kommando som beräknar ett antal mått på stegsvaret för processen G , t ex stigtid, insvängningstid och översväng	» $\text{stepinfo}(G)$
Alternativt kommando för lösning av lineära ekvationssystem $y = A \cdot r$	» $r = \text{linsolve}(A,y)$

Med "z" och "H" istället för "s" och "G"

$Z = tf('z',TS)$ specifies $H(z) = z$ with sample time TS .

You can then specify transfer functions directly as expressions in S

or Z , for example,

Discrete-Time Conventions

The control and digital signal processing (DSP) communities tend to use different conventions to specify discrete transfer functions. Most control engineers use the z variable and order the numerator and denominator terms in descending powers of z , for example,

$$h(z) = \frac{z^2}{z^2 + 2z + 3}.$$

The polynomials z^2 and $z^2 + 2z + 3$ are then specified by the row vectors $[1 \ 0 \ 0]$ and $[1 \ 2 \ 3]$, respectively. By contrast, DSP engineers prefer to write this transfer function as

$$h(z^{-1}) = \frac{1}{1 + 2z^{-1} + 3z^{-2}}$$

and specify its numerator as 1 (instead of $[1 \ 0 \ 0]$) and its denominator as $[1 \ 2 \ 3]$.

tf switches convention based on your choice of variable (value of the 'Variable' property).

Variable	Convention
'z' (default), 'q'	Use the row vector $[a_k \dots a_1 a_0]$ to specify the polynomial $a_k z^k + \dots + a_1 z + a_0$ (coefficients ordered in <i>descending</i> powers of z or q).
'z^-1'	Use the row vector $[b_0 b_1 \dots b_k]$ to specify the polynomial $b_0 + b_1 z^{-1} + \dots + b_k z^{-k}$ (coefficients in <i>ascending</i> powers of z^{-1}).

For example,

```
g = tf([1 1],[1 2 3],0.1);
```

specifies the discrete transfer function

$$g(z) = \frac{z+1}{z^2+2z+3}$$

because z is the default variable. In contrast,

```
h = tf([1 1],[1 2 3],0.1,'variable','z^-1');
```

uses the DSP convention and creates

$$h(z^{-1}) = \frac{1+z^{-1}}{1+2z^{-1}+3z^{-2}} = zg(z).$$

See also [filt](#) for direct specification of discrete transfer functions using the DSP convention.

Note that `tf` stores data so that the numerator and denominator lengths are made equal. Specifically, `tf` stores the values

```
num = [0 1 1]; den = [1 2 3];
```

for g (the numerator is padded with zeros on the left) and the values

```
num = [1 1 0]; den = [1 2 3];
```

for h (the numerator is padded with zeros on the right).

Properties

`tf` objects have the following properties:

<code>num</code>	<p>Transfer function numerator coefficients.</p> <p>For SISO transfer functions, <code>num</code> is a row vector of polynomial coefficients in order of descending power (for Variable values <code>s</code>, <code>z</code>, <code>p</code>, or <code>q</code>) or in order of ascending power (for Variable values <code>z^-1</code> or <code>q^-1</code>).</p> <p>For MIMO transfer functions with <code>Ny</code> outputs and <code>Nu</code> inputs, <code>num</code> is a <code>Ny</code>-by-<code>Nu</code> cell array of the numerator coefficients for each input/output pair.</p>
<code>den</code>	<p>Transfer function denominator coefficients.</p> <p>For SISO transfer functions, <code>den</code> is a row vector of polynomial coefficients in order of descending power (for Variable values <code>s</code>, <code>z</code>, <code>p</code>, or <code>q</code>) or in order of ascending power (for Variable values <code>z^-1</code> or <code>q^-1</code>).</p> <p>For MIMO transfer functions with <code>Ny</code> outputs and <code>Nu</code> inputs, <code>den</code> is a <code>Ny</code>-by-<code>Nu</code> cell array of the denominator coefficients for each input/output pair.</p>
<code>Variable</code>	<p>String specifying the transfer function display variable. Variable can take the following values:</p> <ul style="list-style-type: none"> 's' — Default for continuous-time models 'z' — Default for discrete-time models 'p' — Equivalent to 's' 'q' — Equivalent to 'z' 'z^-1' — Inverse of 'z' 'q^-1' — Equivalent to 'z^-1' <p>The value of <code>Variable</code> is reflected in the display, and also affects the interpretation of the <code>num</code> and <code>den</code> coefficient vectors for discrete-time models. For <code>Variable</code> = 'z' or 'q', the coefficient vectors are ordered in descending powers of the variable. For <code>Variable</code> = 'z^-1' or 'q^-1', the coefficient vectors are ordered as ascending powers of the variable.</p> <p>Default: 's'</p>
<code>ioDelay</code>	<p>Transport delays. <code>ioDelay</code> is a numeric array specifying a separate transport delay for each input/output pair.</p> <p>For continuous-time systems, specify transport delays in the time unit stored in the <code>TimeUnit</code> property. For discrete-time systems, specify transport delays in integer multiples of the sample time, <code>Ts</code>.</p> <p>For a MIMO system with <code>Ny</code> outputs and <code>Nu</code> inputs, set <code>ioDelay</code> to a <code>Ny</code>-by-<code>Nu</code> array. Each entry of this array is a numerical value that represents the transport delay for the corresponding input/output pair. You can also set <code>ioDelay</code> to a scalar value to apply the same delay to all input/output pairs.</p> <p>Default: 0 for all input/output pairs</p>
<code>InputDelay</code>	<p>Input delay for each input channel, specified as a scalar value or numeric vector. For continuous-time systems, specify input delays in the time unit stored in the <code>TimeUnit</code> property. For discrete-time systems, specify input delays in integer multiples of the sample time <code>Ts</code>. For example, <code>InputDelay</code> = 3 means a delay of three sample times.</p>

<p>» $H = \text{tf}([0.3 \ 0 \ 0.1], [1 \ -0.9 \ 0.5 \ -0.1], 1)$</p> <p>Transfer function:</p> $\frac{0.3 z^2 + 0.1}{z^3 - 0.9 z^2 + 0.5 z - 0.1}$ <p>Sampling time: 1</p>	<p>Inmatning av en tidsdiskret överföringsfunktion. Det första argumentet anger täljarparametrarna, det andra anger nämnarparametrarna och det sista anger samplingstiden. Överföringsfunktionen presenteras med positiv representation.</p>
<p>» <code>step(H)</code> » <code>impulse(H)</code> » <code>bode(H)</code></p>	<p>Stegsvaret Impulssvaret Bodediagrammet</p>
<p>» $G = \text{tf}([5], [5 \ 2 \ 0]);$ » $H = \text{c2d}(G, 2)$</p> <p>Transfer function:</p> $\frac{1.558 z + 1.195}{z^2 - 1.449 z + 0.4493}$ <p>Sampling time: 2</p>	<p>Inmatning av en kontinuerlig överföringsfunktion. Därefter diskretisering med samplingstiden två sekunder. Kommandot <code>c2d</code> används (utläses: continuous to discrete). Som default används steginvariant transform, vilket är samma som transformering med 'zero order hold' (zoh) på signalen. Andra diskretiseringsmetoder fås genom att ange önskad metod med ett tredje argument (se nedan).</p>
<p>» $HH = \text{d2d}(H, 3);$</p>	<p>Kommando för att diskretisera om en tidsdiskret överföringsfunktion H med ny samplingstid. Ny samplingstid ges som argument till kommandot <code>d2d</code> (discrete to discrete).</p>
<p>» $G = \text{tf}([1], [1 \ 1]);$ » $H = \text{c2d}(G, 1, 'tustin')$</p> <p>Transfer function:</p> $\frac{0.3333 z + 0.3333}{z - 0.3333}$ <p>Sampling time: 1</p>	<p>Inmatning av en överföringsfunktion. Därefter diskretisering med Tustins metod (som är identisk med bilineär transform).</p>
<p>» $H = \text{tf}([0.3 \ 0 \ 0.1], [1 \ -0.9 \ 0.5 \ -0.1], 1);$ » $G1 = \text{d2c}(H)$</p> <p>Transfer function:</p> $\frac{0.4087 s^2 + 0.4397 s + 1.271}{s^3 + 2.303 s^2 + 2.71 s + 1.589}$	<p>Transformerering av en tidsdiskret modell till en tidskontinuerlig med kommandot <code>d2c</code> (discrete to continuous). Även här kan man specificera önskad metod för transformereringen med ett extra argument i kommandot: 'zoh' = zero order hold. (default) 'tustin' = Tustins metod etc.</p>