Funktionsfiler

En funktionsfil är en fil som beskriver en funktion som anropas från huvudprogrammet. Den sparas separat.
En funktionsfil har det allmänna utseendet
Funktionsfiler används t.ex. vid beräkning av nollställen, lokala extrempunkter och integraler.

```
function y=filnamn(x1,x2,...,xn);
y=funktionsuttryck;
```

`y` kan även vara en vektor med flera komponenter.

Funktionsfilen har ett antal in-parametrar från huvudprogrammet och ett antal ut-parametrar som efter anropet kan användas i huvudprogrammet.

Funktionsfilen skall sparas under det filnamn man angivit på första raden.

Nollställen

För att bestämma nollställena till en funktion $f(x)$ beskrives denna först med en funktionsfil enligt ovan.
Om vi betraktar en funktion av en variabel har denna fil utseendet

```
%nollställe a
function y=fun(x);
y=funktionsuttryck;
```

Här har vi kallat filen `fun` och under detta namn skall den alltså sparas.

För att bestämma önskat nollställe hos $f(x)$ anropar vi funktionsfil med instruktionen

```
%nollställe b
fzero('fun', x0)
```
, där `fun` är vår funktionsfil som beskriver funktionen f.
`x0` är det startvärde matlab använder för att hitta nollstället. Detta skall vara ett värde i närheten av det förväntade nollstället och kan erhållas ur f:s graf.
Anropet kan göras från en m-fil eller från kommandofönstret.

Exempel

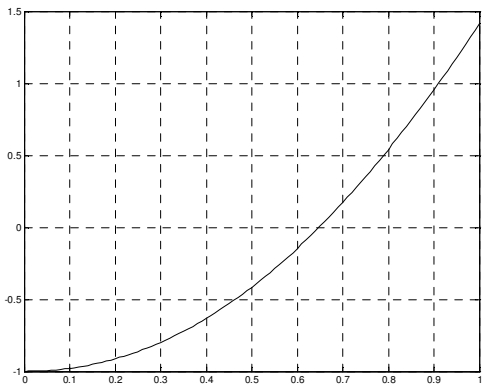
Bestäm det nollställe till $f(x)=x^3-\cos 2x$ som ligger i intervallet $(0,1)$.

Lösning

Rita först grafen för $f(x)$ i det aktuella intervallet.

```
%p11
x=0:0.01:1
y=x.^3-cos(2*x)
plot(x,y);
grid on;
```

Detta ger grafen t.h.



Nollstället ligger som synes nära $x=0{,}6$.

Skriv sedan en funktionsfil som beskriver $f(x)$.

```
%fun1
function y=fun(x);
y=x.^3-cos(2*x);
```

Spara denna fil under namnet `fun1`.
Det kommando som beräknar nollstället anropar denna funktionsfil enligt följande (från kommandofönstret)

```
>>n=fzero('fun1',0.6)
```

När programmet körts redovisas resultatet i kommando-fönstret

```
n =

    0.6478
>>
```

Nollstället är alltså $x = 0{,}6478$

Ett alternativ till att ge ett starvärde är att ge det intervall det förväntade nollstället ligger i. Detta intervall måste anges så funktionsvärdet har olika tecken i intervallets ändpunkter.
Intervallet anges som en vektor enligt nedan där intervallet $(0,1)$ används.

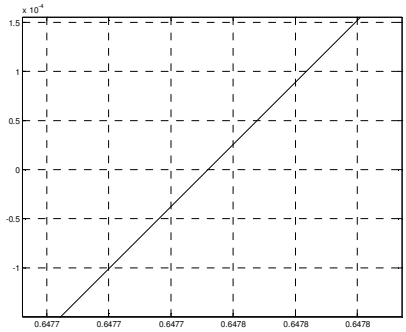
```
n=fzero('fun1',[0 1])
```

Detta ger resultatet

```
n =

    0.6478

>>
```

Om vi förstorar grafen ser vi att nollstället är något mindre än det angivna.
Genom att ge kommandot `format long` i kommando-fönstret kan vi få nollstället med fler siffror

```
n =

   0.64776543345820

>>
```



Lokala min- och max-punkter

Matlab har ett kommando för att bestämma det x-värde för vilket funktionen $f(x)$ har lokalt minimum i ett angivet intervall $(x1, x2)$ samt motsvarande min-värde. Detta kommando kräver att funktionen beskrivs med en funktionsfil.
Kommandot har följande utseende

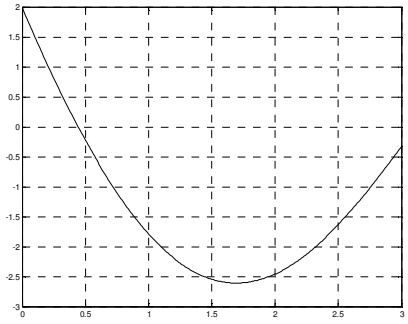
```
[x,y]=fminbnd('fun1',x1,x2).
```

I vektorn `[x,y]` returneras x-värde och motsvarande min-värde y.
Om funktionen saknar lokalt minimum i intervallet returneras funktionens ändpunktsminimum.

För att hitta en maxpunkt söks minpunkten för $-f(x)$.

Exempel

Bestäm det x-värde för vilket $f(x) = 2 \cdot e^{-x} - 3\sin x$ har en min-punkt i intervallet $(1,2)$.



Lösning

Skriv först funktionsfilen som beskriver den funktion vars minvärde vi söker.

```
%fun2
function y=fun2(x);
y=2*exp(-x)-3*sin(x);
```

Programmet som ger svaret anropar funktionsfilen och ser ut enligt

```
[x,y]=fminbnd('fun2',1,2)
```

Körs detta program ges resultatet i kommandofönstret

```
x =

    1.6937


y =

   -2.6097

>>
```

Funktionen har alltså minvärdet -2,6097 för x-värdet 1,6937.

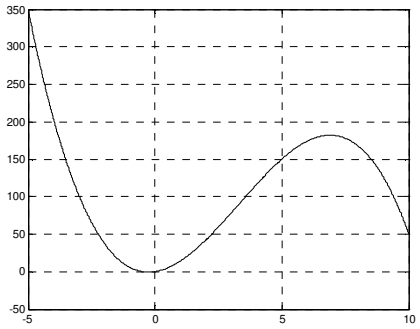
Derivata

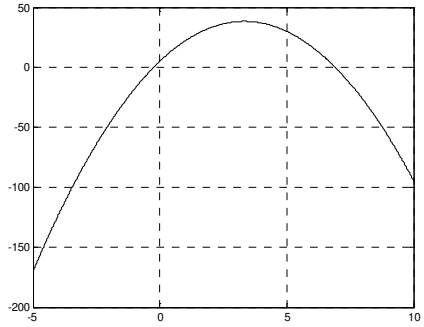
Matlab kan approximativt ge derivatan av en funktion. Detta görs genom att man beräknar differenskvoten i varje punkt. Om funktionen anges med värden i *N* punkter fås alltså derivatan som en vektor med *N-1* element.

Exempel

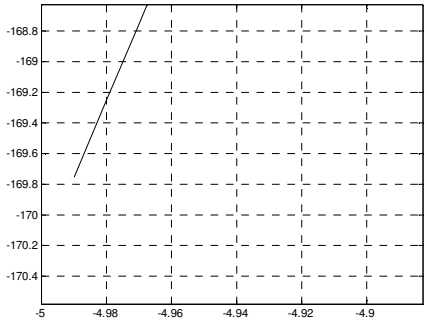
I diagrammet t.v. visas grafen för $y = 5x - x^3 + 10x$.
Nedan följer ett program som beräknar och visar grafen för derivatan y' som visas i det högra diagrammet.

```
%p12
x=-5:0.01:10
y=5*x-x.^3+10*x.^2
z=diff(y)./diff(x) %derivatan beräknas
t=-4.99:0.01:10 %vektor med en punkt mindre än hos vektorn x
plot(t,z)
grid on
```





Observera att derivatan för *x* = 0 inte kan beräknas! När vi ger plot-kommandot för *z* måste x-vektorn göras kortare!
detta kan vi se om grafen förstoras, se fig.



Integraler

Med kommandot `quad('fun',a,b)` kan integralen av en funktion given i en funktionsfil beräknas.

Exempel

Beräkna integralen $\int_1^4 \sqrt{x}dx$.

Lösning

Först måste vi skriva en funktionsfil som beskriver integranden, i detta fall $\sqrt{x}$.

```
%fun3
function y=fun3(x)
y=sqrt(x)
```

Härefter kan ge vi alla kommandon i kommandofönstret enligt

```
>> I=quad('fun3',1,4)
```

Detta ger utskriften

```
I =

    4.6667

>>
```

Alltså $\int_1^4 \sqrt{x}dx = \underline{\underline{4{,}6667}}$

På motsvarande sätt kan dubbelintagraler över ett rektangulärt område beräknas med kommandot
`dblquad('fun',a,b,c,d),`
Där integralen beräknas över rektangeln $(a,b)\times(c,d)$.

Exempel

Beräkna integralen $\int_0^1 \int_{-1}^3 x\cdot\sin\left(xy^2\right)dxdy$.

Lösning

Funktionsfilen skrivs enligt

```
%fun4
function z=fun4(x,y)
z=x.*sin(x*y.^2)
```

Kommandot i kommandofönstret ges då som

```
>> I=dblquad('fun4',0,1,-1,3)
```

Detta ger utskriften

```
I =

    0.5158

>>
```

Lösning av ordinära differentialekvationer, (ODE), med Matlab.

Vi börjar med att studera en ordinär differentialekvation av första ordningen

$y'(t)+f(t)\cdot y(t)=g(t)$, $y(0)=a$

Detta kan skrivas $y'=F(t,y)$, $y(0)=a$, där $F(t,y)=g(t)-f(t)\cdot y$

Eulers stegmetod innebär att vi beräknar värdet hos y i diskreta punkter där *y*:s värde i en punkt bestäms ur värdena på *y* och y' i föregående punkt.
Om vi stegar fram med konstant steglängd = *h* gäller approximativt skrivas
$\Delta y \approx h\cdot y' = h\cdot F(t,y)=h\cdot(g(t)-f(t)\cdot y(t))$ enligt ovan.

Detta ger att $y(t+h)=y(t)+h\cdot y'(t)=y(t)+h\cdot F(t,y)=y(t)+h\cdot(g(t)-f(t)\cdot y(t))$

Om vi studerar lösningen vid distinkta tidpunkter $t=0,h,2h,3h,.....,(n-1)\cdot h,\ n\cdot h,.....$
får vi

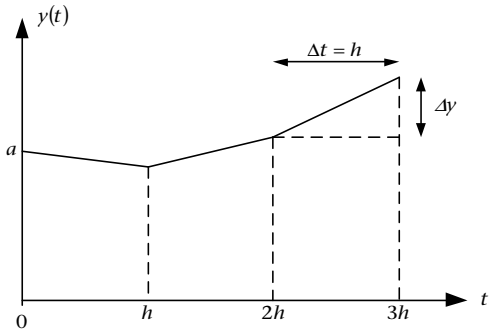
$y((n+1)\cdot h)\approx y(n\cdot h)+h\cdot(g(n\cdot h)-f(n\cdot h)\cdot y(n\cdot h))$,

vilket ger att vi kan beräkna $y(t)$ stegvis ur de föregående värdena

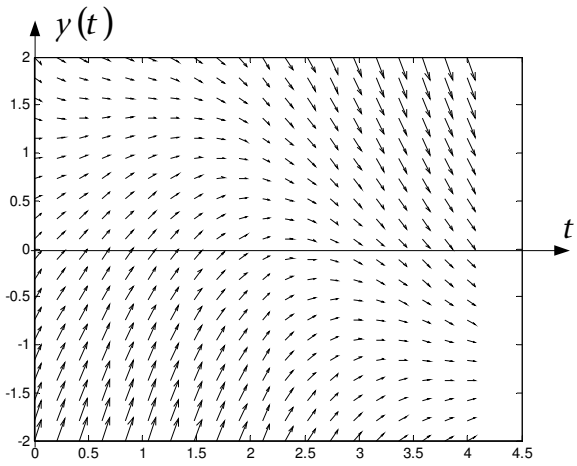
$n=0$ $\quad y(h)\approx y(0)+h\cdot(g(0)-f(0)\cdot y(0))$

$n=1$ $\quad y(2h)\approx y(h)+h\cdot(g(h)-f(h)\cdot y(h))$

$n=2$ $\quad y(3h)\approx y(2h)+h\cdot(g(2h)-f(2h)\cdot y(2h))$



$y'=F(t,y)$ kan också åskådliggöras i ett riktningsfält enligt fig t.h. Detta visar alla möjliga förlopp som $y(t)$ kan ha beroende på begynnelsevillkoret.



Denna metod kallas Eulers stegmetod och ger en approximativ lösning till differentialekvationen. En noggrannare metod tar hänsyn till funktionens variation i stegintervallen och väger samman dessa värden, denna metod kallas Runge-Kuttas metod.
I Matlab finns flera algoritmer för lösning av ODE och vi skall utnyttja den som bygger på denna metod nämligen `ode45`.

Lösning av ODE i Matlab tillgår så att man med instruktionen `ode45` anropar en funktionsfil som innehåller differentialekvationen.

Antag att vi vill lösa differentialekvationen $y' = -y + \sin 2t + \frac{t}{t^2+1}$, $y(0) = 1$.

Vi skriver först den funktionsfil som beskriver diff-ekvationen. Den får då följande utseende.
De två parametrarna *t* och *y* är vår oberoende resp. beroende variabel i den lösning på ode:n vi vill ha.
Rad 2 beskriver diff-ekvationen. `dy` motsvarar alltså y'.

```
%fun5
function dy=func1(t,y);
dy=-y+sin(2*t)+t./(t.^2+1);
```

En funktionsfil har det allmänna utseendet

```
function y=filnamn(x1,x2,...,xn);
y=funktionsuttryck;
```

`y` kan även vara en vektor med flera komponenter som vi ser nedan vid lösning av högre ordningens differentialekvationer.
Funktionsfilen skall sparas under det filnamn man angivit på första raden.

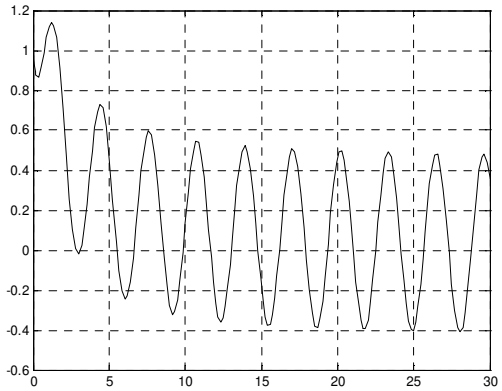
För att få lösningen anropar vi funktionsfilen `func1` med ett program som löser differentialekvationen och anger i detta anrop det tidsintervall vi är intresserade av samt ekvationens begynnelsevillkor.
I detta program har vi även ett kommando som skriver ut lösningens graf.

```
%p13 diffekv1
[t y]=ode45('fun5',[0 30],1);
plot(t,y,'k');grid on
```

I instruktionen `[t y]=ode45('fun5',[0 30],1);` anges att tidsintervallet är
[0 30] samt begynnelsevillkoret $y(0) = 1$. Observera hakparenteserna!
Formen är alltså

```
[t,y]=ode45('funktionsnamn',tidsintervall,begynnelsevillkor);
```

Om vi kör programmet `diffekv1` erhålles lösningen som en graf.



ODE av ordning 2

Matlab kan endast hantera ode eller system av ode av ordning 1. Därför måste differentialekvationer av andra ordningen eller högre skrivas om till ett system av ode av ordning 1.

Antag att vi vill lösa följande begynnelsevärdesproblem (van der Pols ekvation).

$$\begin{cases} x''(t) + \left(x^2(t) - 1\right) \cdot x'(t) + x(t) = 0 \\ x(0) = 0.25 \\ x'(0) = 0 \end{cases}$$

Vi börjar med en omskrivning $\begin{cases} x''(t) = \left(1 - x^2(t)\right) \cdot x'(t) - x(t) \\ x(0) = 0.25 \\ x'(0) = 0 \end{cases}$

Sätt sedan $x = x_2$ och $x' = x_1 \quad \Rightarrow \quad x_1' = x'' \,,\; x_2' = x' = x_1$

Då får problemet följande utseende.

$$\begin{cases} x_1' = \left(1 - x_2^2\right) \cdot x_1 - x_2 \\ x_2' = x_1 \\ x_1(0) = 0 \\ x_2(0) = 0{,}25 \end{cases}$$

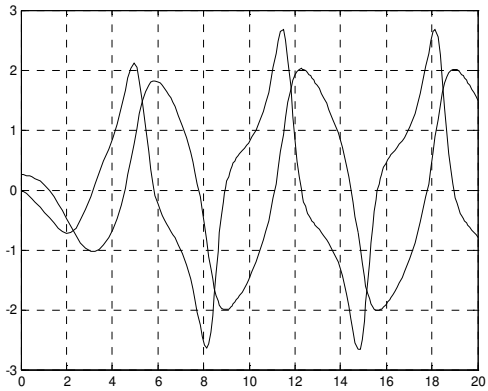
Här har vi alltså ett system av differentialekvationer av ordning 1.

Detta beskrivs med följande funktionsfil. Observera att `dx` måste skrivas som en kolumnvektor, alltså på två rader!

```
%fun6  vanderpol.m
function dx=fun6(t,x)
dx=  [x(1).*(1-x(2).^2)-x(2);
      x(1);]
```

För att lösa ekvationen anropas den med följande program

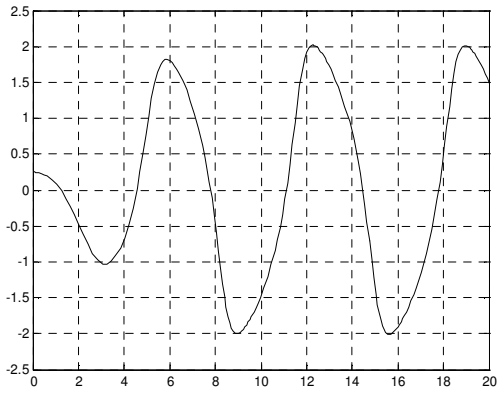
```
%p14 diffekv2_vanderpol_1
t0=0;
tslut=20;
x0=[0 0.25];
[t,x]=ode45('fun6',[t0 tslut],x0);
plot(t,x,'k');grid on;
```



Körs programmet erhålles grafen t.h.

Om vi endast vill studera vår sökta funktion `x=x2`, anger vi detta i `plot`-instruktionen.

```
%p15 diffekv2_vanderpol_2
t0=0;
tslut=20;
x0=[0 0.25];
[t,x]=ode45('vanderpol',[t0 tslut],x0);
plot(t,x(:,2),'k'); % endast x2 skrivs ut
grid on
```



Detta program ger alltså

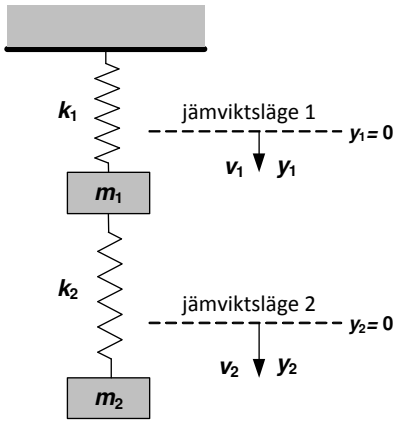
I nästa exempel visas en instruktion, `legend`, som namnger kurvorna i diagrammet.

Ytterligare ett exempel (system av två ODE)

Exempel

(*Hämtat från Per Jönsson, Matlab, beräkningar inom teknik och naturvetenskap*)

Betrakta fjädersystemet i figuren t.h.
Vi skall bestämma vikternas lägen som funktion av tiden för $t > 0$ då



$y_1(0) = 0 \quad \dot{y}_1(0) = 0 \quad y_2(0) = 1 \quad \dot{y}_2(0) = 0$, dvs

$y_1(0) = 0 \quad v_1(0) = 0 \quad y_2(0) = 1 \quad v_2(0) = 0$.

Detta innebär alltså att m_2 höjs 1m varvid m_1 hålls kvar i viloläget, därefter släpps båda vikterna och får svänga fritt.

De differentialekvationer som beskriver rörelsen bildar då ett ekvationssystem enligt

$$\begin{cases} m_1\ddot{y}_1 = -k_1 y_1 + k_2(y_2 - y_1) \\ m_2\ddot{y}_2 = \quad\quad -k_2(y_2 - y_1) \end{cases}$$

För att kunna lösa detta andra ordningens system med Matlab måste vi, enligt tidigare, skriva om det till ett system av första ordningens ekvationer. Detta görs enligt följande variabelbyte

$$\begin{cases} z_1 = y_1 \\ z_2 = \dot{y}_1 \\ z_3 = y_2 \\ z_4 = \dot{y}_2 \end{cases} \Rightarrow \begin{cases} y_1 \to z_1 \\ \dot{y}_1 \to z_2 \\ \ddot{y}_1 \to \dot{z}_2 \\ y_2 \to z_3 \\ \dot{y}_2 \to z_4 \\ \ddot{y}_2 \to \dot{z}_4 \end{cases}$$

Observera speciellt hur andraderivatorna blivit förstaderivator i de nya variablerna. Detta på bekostnad av fyra ekvationer i stället för två.

$$\begin{cases} \dot{z}_2 = \ddot{y}_1 \\ \dot{z}_4 = \ddot{y}_2 \end{cases}$$

$$\begin{cases} \dot{z}_1 = z_2 \\ \dot{z}_2 = -\dfrac{k_1}{m_1} z_1 + \dfrac{k_2}{m_1}(z_3 - z_1) \\ \dot{z}_3 = z_4 \\ \dot{z}_4 = -\dfrac{k_2}{m_2}(z_3 - z_1) \end{cases}$$

Som visats tidigare gör vi först en funktionsfil som beskriver ekvationssystemet, denna fil kallas **dubbel** och sparas även under detta namn, därefter skriver vi ett program som anropar funktionsfilen och löser ekvationssystemet. Detta program kallar vi **ode_dubbel**.
Här har vi även deklarerat parametervariablerna som globala så att de bär med sig sina värden.

```
%fun7
function dz=fun7(t,z)
global m1 m2 k1 k2  % globala variabler
dz=[z(2)
-k1/m1*z(1)+k2/m1*(z(3)-z(1))
z(4)
-k2/m2*(z(3)-z(1))];
```

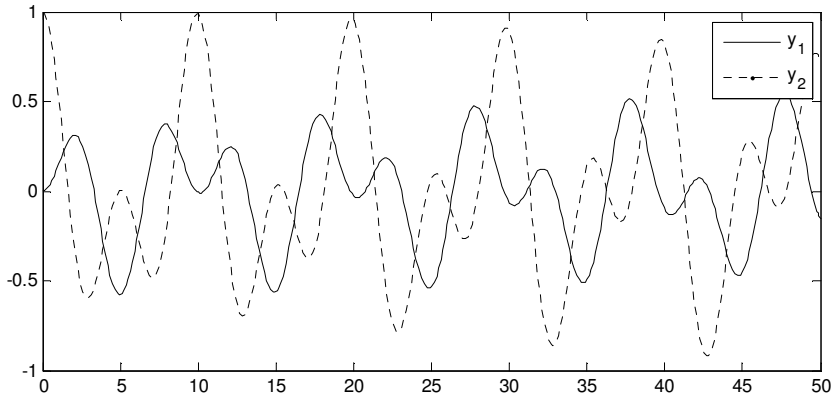
```
% p16
global m1 m2 k1 k2  % globala variabler
m1=3;m2=1;k1=2;k2=1;
[t,z]=ode45('fun7',[0 50],[0 0 1 0]);
plot(t,z(:,1))
hold on
plot(t,z(:,3),'--')
legend('y_1','y_2')  %Kurvornas namn anges i diagrammet
```

Observera hur begynnelsevärdena anges

```
[t,z]=ode45('dubbel',[0 50],[0 0 1 0]);
```

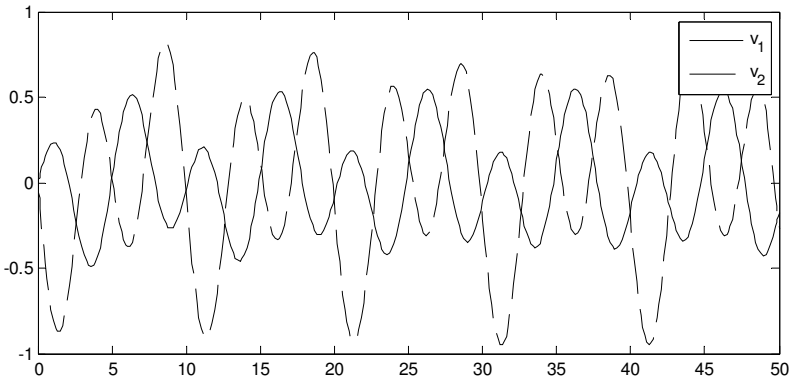
`[0 0 1 0])` betyder $z_1(0)=0 \quad z_2(0)=0 \quad z_3(0)=1 \quad z_4(0)=0$, dvs
$y_1(0)=0 \quad \dot{y}_1(0)=0 \quad y_2(0)=1 \quad \dot{y}_2(0)=0$.

När detta program körs erhålles följande utskrift



Övning

Ändra programmet så att massornas hastigheter skrivs ut. Det skall ge följande diagram.



Något om Matlab Control Toolbox

Till Matlab finns tillägg för olika tillämpningar, tex **Signal processing toolbox** för signalbehandling och **Control System Toolbox** för reglertekniska beräkningar.
Vi skall här stifta en ganska ytlig bekantskap med den senare.
Matlab **Control System Toolbox** kan hantera kontinuerliga och diskreta signaler och system.

Inmatning

Vi börjar med inmatning av system i **Command Window** och utgår från överföringsfunktionen

$$G1(s) = \frac{s+2}{3s^2+2s+4}$$

För att mata in denna ger vi kommandot

```
» G1=tf([1 2],[3 2 4])
```

följt av RETURN (alla kommandon skall åtföljas av RETURN)

Då svarar programmet i **Command Window** med

```
Transfer function:
     s + 2
--------------
3 s^2 + 2 s + 4
»
```

Överföringsfunktionen anges alltså med två vektorer, den första med täljarpolynomets koefficienter och den andra med nämnarpolynomets koefficienter med högst gradtal först.
Det finns också kommandon för att bilda summan och produkten mellan överföringsfunktioner

```
» H=G1+G2
```

```
» H=G1*G2
```

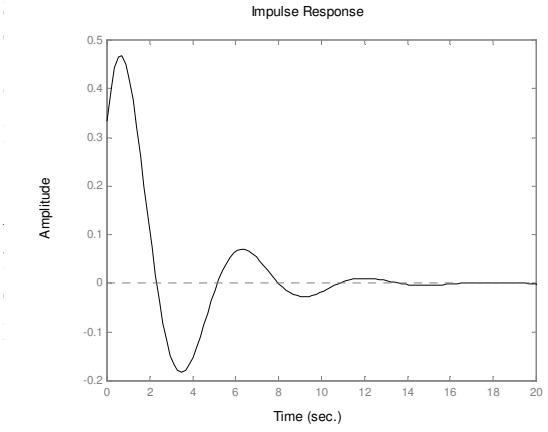
Impuls- och stegsvar

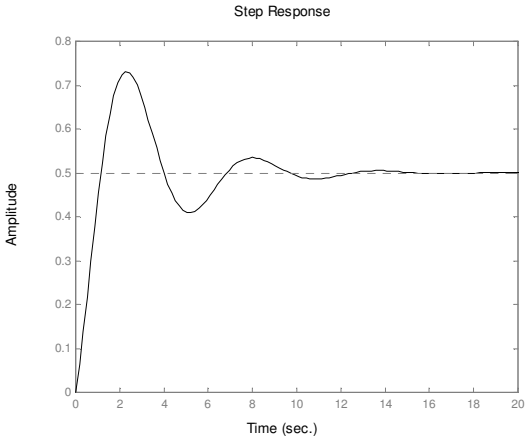
Kommandot **`impulse(G1,20)`** bildar G1:s impulssvar och ritar det i grafikfönstret för $0 \le t \le 20$.

På motsvarande sätt ritas stegsvaret för G1 ut med kommandot

```
step(G1,20)
```

Dessa kommandon ger följande grafer i **grafikfönstret**





Rita amplitud och fasfunktion för $H(s)=\dfrac{2s+5}{s^2+8s+25}$

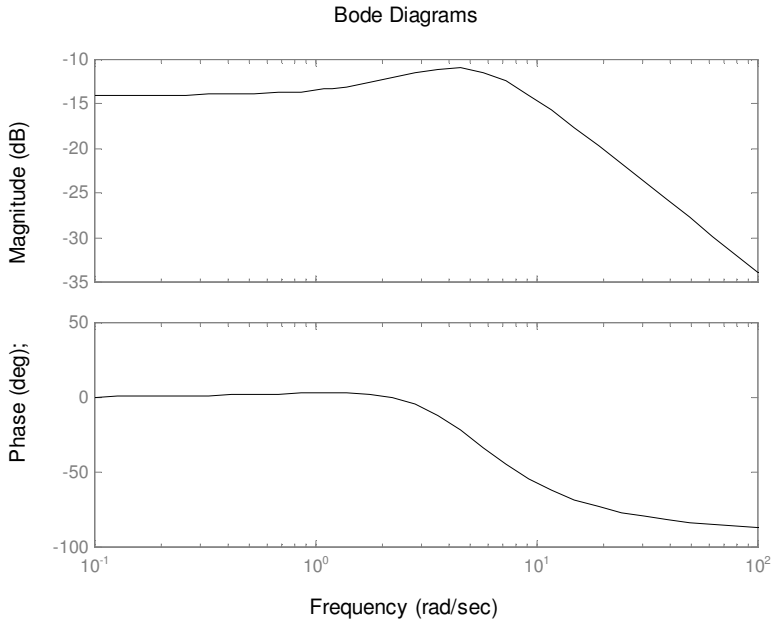
Lösning

Bilda först $H(s)$ enligt

```
» H=tf([2 5],[1 8 25])

Transfer function:
   2 s + 5
--------------
s^2 + 8 s + 25
```

Rita sedan bodediagrammet med

```
» bode(H)
»
```



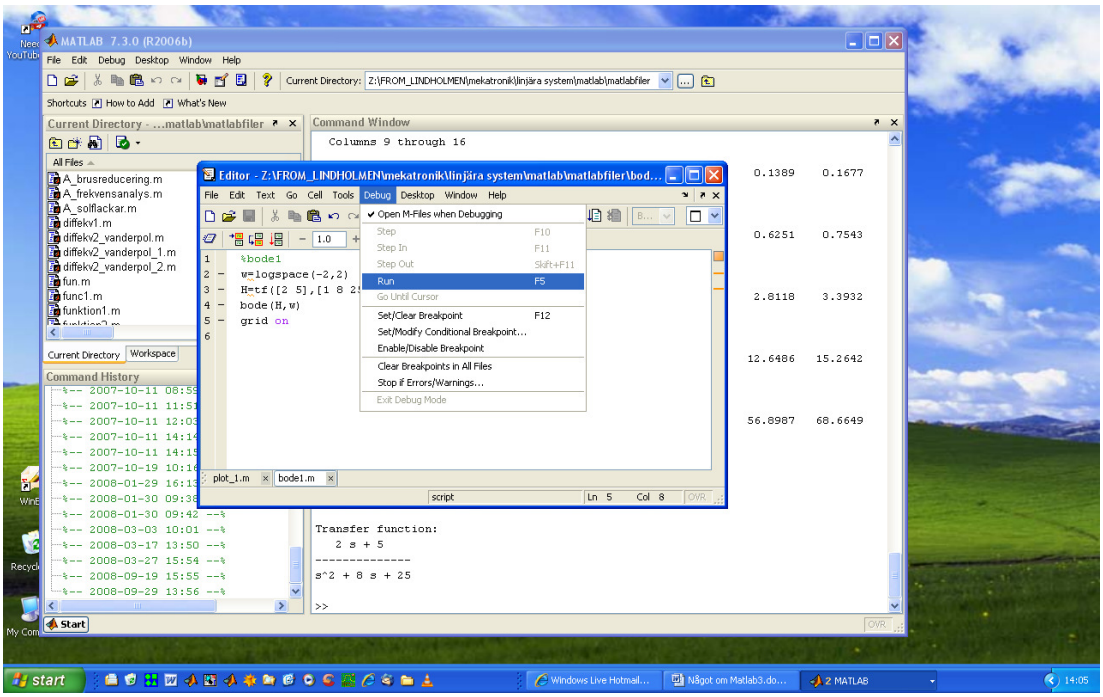
Vi kan påverka diagrammets utseende genom att ange att vi vill ha rutnät i diagrammet och dessutom välja en annan skala. Eftersom det är flera kommandon samlar vi dessa i ett program som vi skriver i **editorn**, då kan vi ju köra om programmet och slipper skriva in allt varje gång!

```
%p17 bode1
w=logspace(-2,2)
H=tf([2 5],[1 8 25])
bode(H,w)
grid on
```

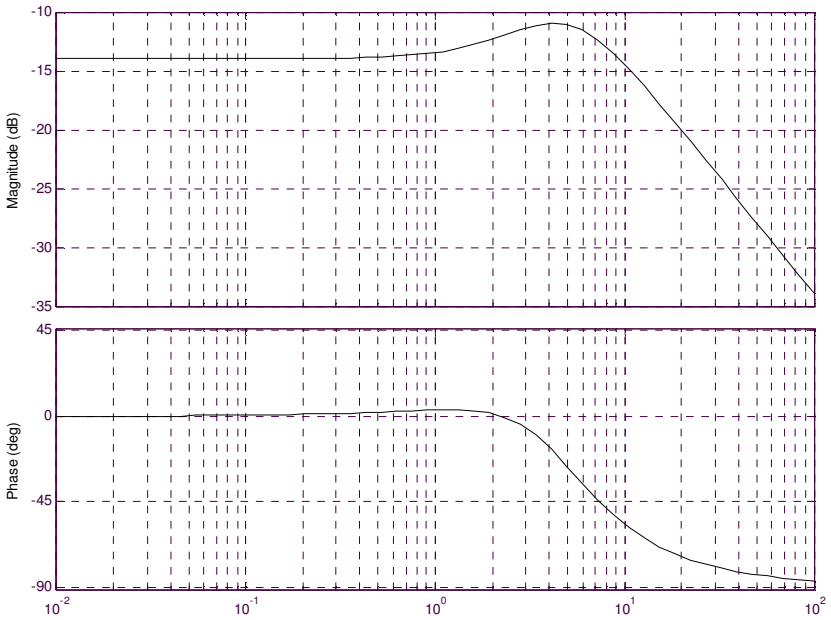
Första raden anger programnamnet. **%** anger att det är en kommentar och inget kommando.
Raden **w=logspace(-2,2)** anger att jag vill ha frekvensskalan $\left(10^{-2},10^{2}\right)$ rad/s.
Raden **grid on** anger att jag vill ha rutnät enligt skalan.

Programmet kan köras från kommandofönstret (**Command Window**) med följande kommando

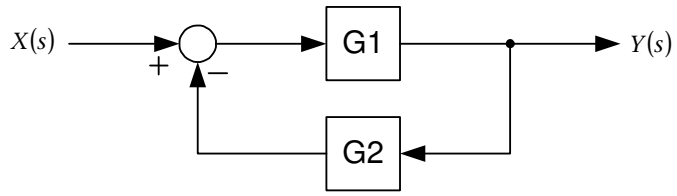
» bode1



Det kan även köras från editorn, under **Debug** finns alternativet **Run** om du redan sparat programmet eller **Save and run** om det inte är sparat, se figur ovan.



Ofta studerar man system där utsignalen återkopplas till utgången (feedback), se figur nedan.
Du kan läsa om **blockscheman för system** på s. 115-116 i läroboken.



Överföringsfunktionen för detta system erhålles med kommandot

Feedback(G1,G2)

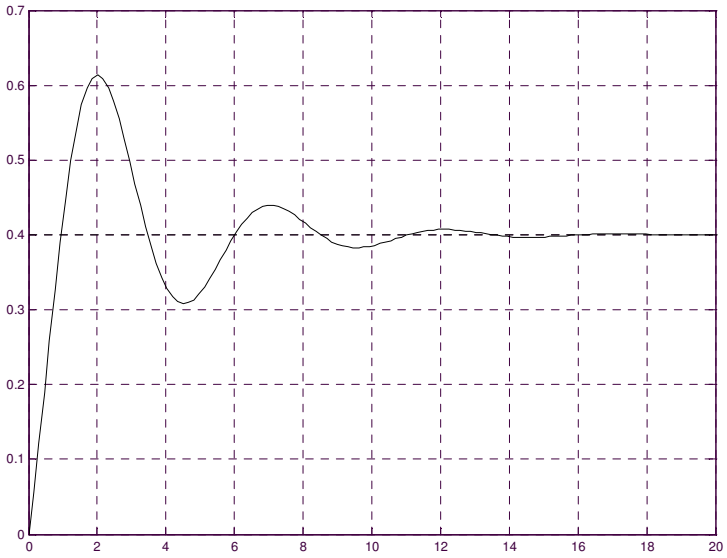
Låt G1 vara enligt ovan, alltså $G1(s)=\frac{s+2}{3s^2+2s+4}$ och låt G2 vara $G2=\frac{1}{s+2}$.

För att få överföringsfunktionen $H(s)=\frac{Y(s)}{X(s)}$ för detta återkopplade system ger vi alltså följande kommandon i **Command Window**

```
» G1=tf([1 2],[3 2 4])            %din inmatning

Transfer function:                %programmets svar
     s + 2
---------------
3 s^2 + 2 s + 4

»
» G2=tf([1],[1 2])                %din inmatning

Transfer function:                %programmets svar
  1
-----
s + 2


» H=feedback(G1,G2)               %din inmatning

Transfer function:                %programmets svar
     s^2 + 4 s + 4
------------------------
3 s^3 + 8 s^2 + 9 s + 10

»
```

Vi bestämmer stegsvaret för detta system med följande kommando, fortfarande i kommandofönstret

```
» step(H,20)


»
```



På nytt ett svängande stegsvar alltså!

Vi kan naturligtvis som i förra exemplet samla våra kommandon i ett program som vi skriver i **editorn**.

Exempel

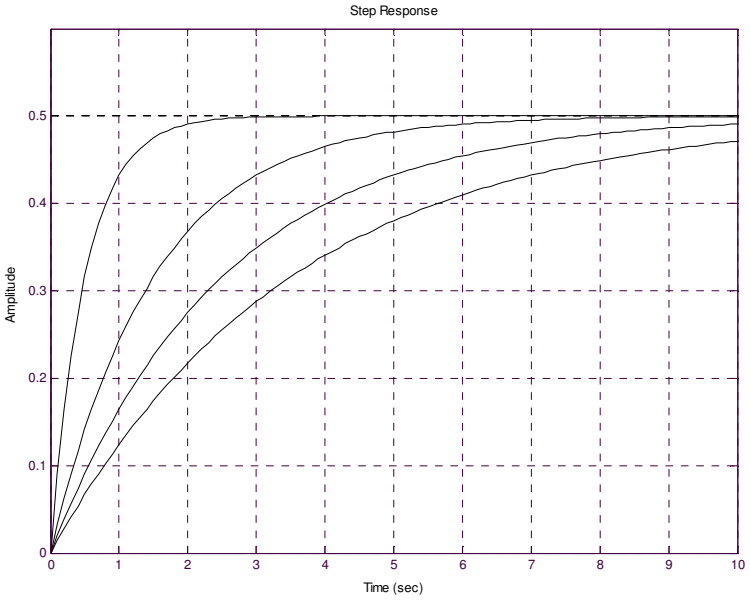
Rita stegsvaret för $G(s)=\frac{2}{a\cdot s+4}$ för a = 2, 6, 10, 14

Lösning

Stegsvaret beräknas av följande program där vi gör en loop vars loop-variabeln a får genomlöpa de önskade värdena. I varje varv bildas överföringsfunktionen för det aktuella a-värdet varefter dess stegsvar beräknas och ritas i grafikfönstret.
Programmet börjar med att stänga grafikfönstret för tidigare utskrifter. Varje loop-varv innehåller kommandot **`hold on`** som håller grafikfönstret öppet för följande a-värde.

```
%p18
hold off              %stänger grafikfönstret
for a=2:4:14          % a antar värdena 2,6,10,14.
G=tf([2],[a 4])       %bildar överföringsfunktionen för aktuellt a-värde
step(G,10)            %beräknar stegsvaret för denna överföringsfunktion
hold on               %håller grafikfönstret öppet för nästa a-värde
end
```

Följande utskrift erhålles (fundera över vilka värden på a som ger de olika kurvorna!)


Step Response
Amplitude
Time (sec)
0
0.1
0.2
0.3
0.4
0.5
0
1
2
3
4
5
6
7
8
9
10

Referenser

Eva Pärt-Enander	Matlab6
Per Jönsson	MATLAB beräkningar inom teknik och naturvetenskap