Matlab för Nybörjare

Author: Charlie Pelland

Updated by:
Marc Meunier IT-amanuens

Introduktion till Matlab

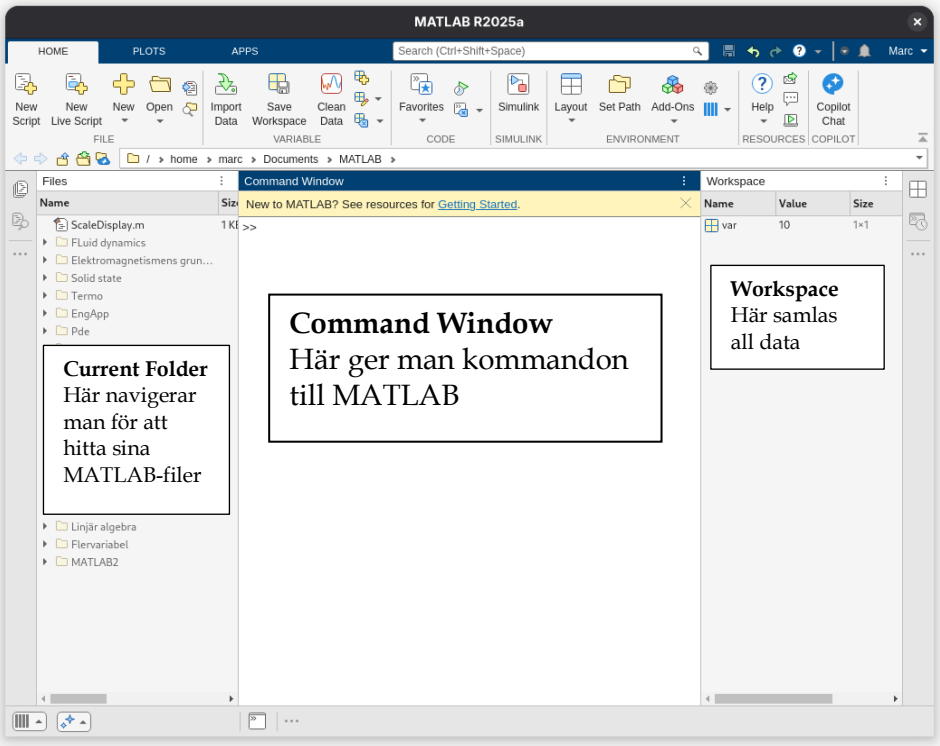
Matlab (matrix laboratory) är ett datorprogram och ett programspråk som används av ingenjörer runt om i världen. Ni kommer att använda er av det under hela er utbildning och mycket möjligt i arbetslivet därefter. Programmet används för matematiska och tekniska beräkningar, kontrolldesign och system, signalprocesser, finansiell modellering, applikationsutveckling, bildbehandling med mera.

Här kommer en kort introduktion samt ett par övningar som förklaras steg för steg för att ni ska få en mjukstart till att programmera och förstå vad man kan göra i Matlab. Ni kommer märka att koden är på engelska, vilket kan vara att föredra då Å,Ä,Ö ibland kan ställa till problem.

Om det känns som att det går väldigt fort fram så oroa er inte. Vi kommer att komma tillbaka och gå igenom grunderna igen under Metoder och Verktyg.

Inledning

Börja med att öppna MATLAB.



Figur 1: Så här kan Matlab se ut när programmet startas.

På bilden ser vi hur Matlab kan se ut vid uppstart. Om ni använder MATLAB R2025A eller senare går de att ändra till ovan vy genom att välja ***Three Column*** under ***HOME -> Layout***. Vi börjar med att fokusera på Command Window. Testa att skriva in ett simpelt kommando här, som tex:

```
>> (3+4)*5
```

och avsluta med ENTER. Matlab kommer då svara med:

```
ans =

    35
```

Nu ser vi att **ans** har sparats i Workspace fönstret, där all vår data samlas. Fortsätt med att testa lite vanliga matematiska kommandon:

Exempel	Kommando
$\sqrt{9}$	sqrt(9)
$\sin(\frac{\pi}{2})$	sin(pi/2)
$\lvert -100 \rvert$	abs(-100)
4^2	4^2

Hjälp-funktionen

Är man osäker på vad ett kommando gör kan man använda sig av hjälp-funktionen. Testa exempelvis hjälp-funktionen för absolutvärden:

```
>> help abs
```

För mer information om hur hjälp-funktionen fungerar, ge kommandot:

```
>> help help
```

Vet man däremot inte hur kommandot lyder kan man använda sig av doc-funktionen, tex:

```
>> doc absolute value
```

För mer information om hur doc-funktionen fungerar, ge kommandot:

```
>> doc doc
```

Ett annat bra sätt att hitta hjälp på är att söka på nätet och var aldrig rädd för att fråga om hjälp.

Variabler

Att använda sig av variabler är en viktig del av Matlab, bla för att det är enkelt sätt att ändra ett värde om det används på flera ställen i en kod. Börja med att skapa två variabler.

```
>> a=4
```

```
>> b=3
```

Testa att göra en ekvation med hjälp av variablerna. Tex Pythagoras sats:

```
>> c=sqrt (a^2+b^2)
```

Eller cirkelns omkrets:

```
>> d=pi*c
```

Notera att alla variabler samlas i Workspace fönstret och att en variabel går att skriva över. Kom också ihåg att döpa variablerna väl, så att ni kommer ihåg vad variabeln är för något. Undvik redan upptagna namn, så som **help, pi**.

Vektorer & Matriser

När vi arbetar i Matlab så använder man sig mycket av vektorer och matriser (namnet på programmet är ju trots allt en förkortning på *matrix laboratory*) vilket är nödvändigt om man tex ska simulera i flera dimensioner eller bearbeta data. En vektor, kan ses som en lista med värden och kan programmeras in såhär:

```
>> vector1 = [23, -65, 9, 15, 35, 0, 35, 65]
```

Vektorn sparas då som vanligt i Workspace. Testa att dubbelklicka på vektorn i Workspace för att få upp datan i ett rutnät. Detta är ett bra sätt att undersöka och få en överblick.

En matris kan ses som flera vektorer i ett rektangulärt schema med värden och programmeras på ett lik- nande sätt. Värden i en rad skiljs åt med kommatecken och vill man byta rad använder man semikolon. Testa att göra en 3x3 matris:

```
>> matrix1 =[9, 8, 7; 6, 5, 4; 3, 2,1]
```

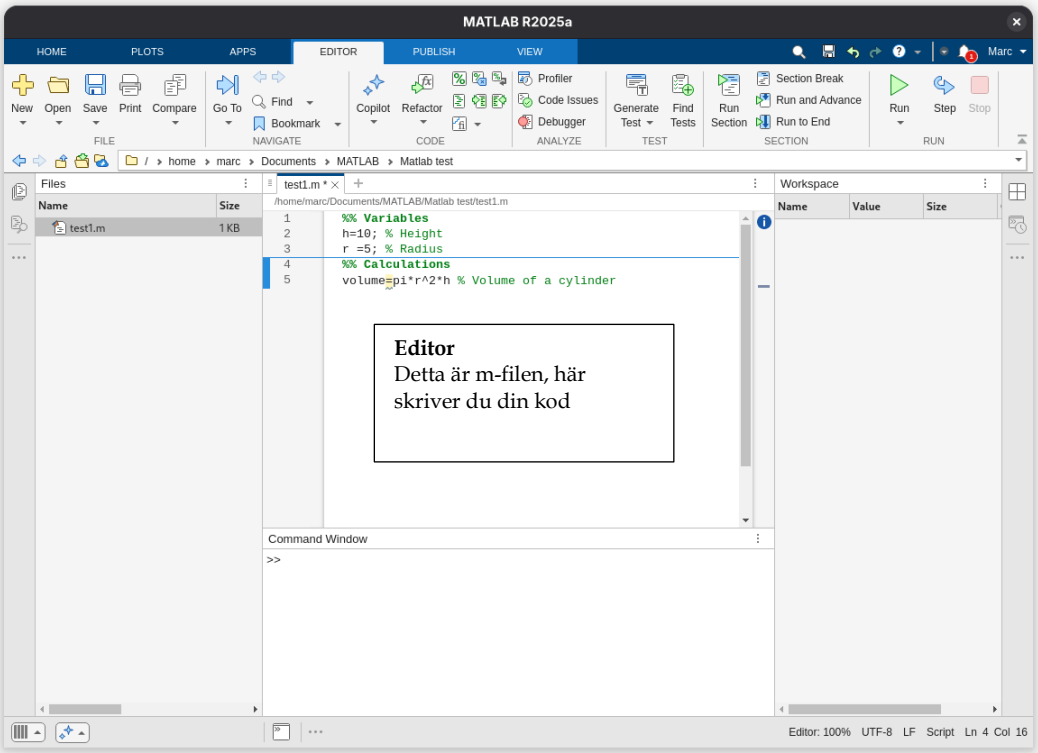
Det finns många sätt att programmera in och hantera vektorer och matriser i Matlab, mer info finns i Matlab-guiden på https://tekniskfysik.se/student/hjalpmedel/matlab-och-python/. Innan vi går vidare kan det vara bra att veta om kommandon som **clc** som rensar Command Window och **clear all** som rensar Workspace.

M-filer

När man programmerar skriver man sällan koden direkt in i Command Window, utan använder sig av en **m-fil**, också kallat **script**. En m-fil är en textfil som skrivs på Matlabs programmeringsspråk och som sedan körs automatiserat, rad för rad, genom Command Window. Vi skriver alltså klart koden innan den körs, istället för att ge enskilda kommandon direkt in i Command Window.

Vi skapar en m-fil med namnet **test1.m** genom att ge kommandot:

```
>> edit test1
```



Figur 2: Matlab med en m-fil startad

Filen lägger sig i **Current Folder**. Testa att leta upp den så att ni vet vart era filer hamnar. Det går också att skapa en ny m-fil genom att klicka på **New**,  , uppe i vänstra hörnet. Går man den här vägen behöver man manuellt spara filen innan man kan köra koden (glöm inte .m när du sparar).

Vi gör tillsammans en enkel kod där vi räknar ut volymen på en cylinder. Börja med att kommentera vad som kommer att hända. Detta är viktigt eftersom det är lätt att gå vilse i en stökig kod, speciellt när den är lång. Med %% startar du en ny sektion i koden och med % startar du en kommentar. Ansätt variablerna och skriv sedan ekvationen för volymen av en cylinder. Koden kan tex se ut såhär:

```
%% Variables
h=10; % Height
r =5; % Radius

%% Calculations
volume=pi*r^2*h  % Volume of a cylinder
```

Lägg märke till semikolonen efter variablerna. Dessa förhindrar att kommandon skrivs ut i Command Window. Eftersom vi är intresserade av volymen, är det den enda information vi vill ha utskriven. Testa att köra koden med snabbkommandot **F5** eller tryck på **Run**,  . Utskriften borde se ut så här:

```
volume =

    785.3982
```

samt att variablerna och svaret ska ha sparats i Workspace.

Funktioner

En stor fördel med att använda Matlab är att det redan finns inprogrammerade funktioner redo att användas. Ni har redan använt några av dessa, tex funktionen som tar roten ur ett värde, **sqrt()**. Man kan också själv göra egna funktioner som man sedan anropar. Fördelen med detta är att samma funktion kan användas flera gånger och därmed sparar man datakraft, tid och koden blir kortare. Ni ska själva få testa på att göra en funktion i en av kommande uppgifter.

Uppgift 1: Flödeshastighet

Problem

En ny simhall ska fylla sina två bassänger med hjälp av en pump. Pumpens flödeshastighet är 0.5 m^3/s. **Hur lång tid kommer det sammanlagt att ta för bassängerna att fyllas?**

	Bassäng 1	Bassäng 2
Bredd [m]	25	10
Längd [m]	50	20
Djup [m]	2	1

Lösningsförslag

Med flödeshastighetsekvationen

$$Q = \frac{V}{t} \Rightarrow t = \frac{V}{Q}, \tag{1}$$

där Q är flödeshastighet, V är volym och t är tid, kan vi få fram vårt svar. Med hjälp av en funktion kan vi räkna ut tiden för båda bassänger.

Huvudkod

- Vi startar med att göra en ny m-fil som vi tex döper till task1.

```
>> edit task1
```

- Vi skriver in vår data i vektorer.

```
%% Data
B1 =[25, 50, 2]; %Pool 1 [width, length, depth]
B2 =[10, 20, 1]; %Pool 2 [width, length, depth]
Pump =0.5; %flowrate
```

- Härnäst kallar vi på vår funktion (som vi ännu inte programmerat). Vi skickar in dimensionerna för bassängerna en och en samt pumpens hastighet. Funktionen kommer att ge tillbaka tiden det tar att fylla en bassäng.

```
%% Flowrate time function
time_B1=myfunction (B1, pump);
time_B2=myfunction (B2, pump);
```

- Sist så adderar vi de båda tiderna och delar på 60 för att få tiden i minuter.

```
TotalTime=(time_B1+time_B2)/60
```

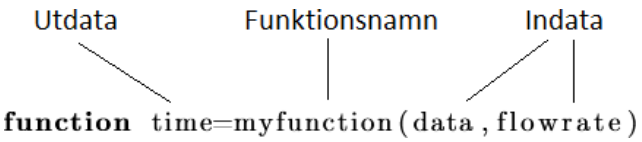
Flödesfunktionen

- En funktion kodas också i en m-fil. Vi döper den till **myfunction**.

```
>> edit myfunction
```

- Väl inne i den nya m-filen måste vi specifiera att det är en funktion, **function**. Härnäst vad funktionen kommer att skicka tillbaka till vår huvudkod, **time** (detta är en variabel). Efter det, namnet på funktionen, **myfunction**, samt vad den importerar, **data** (bassängens dimensioner) och **flowrate** (pumpens flödeshastighet).

- Först räknar vi ut volymen. I vektorn **data** ligger dimensionerna, först bredden , **data(1)** på position 1, sen längden, **data(2)**, osv. Så det är bara för oss att multiplicera dessa tre för att få volymen.




```
volume=data(1)*data(2)*data(3);
```

- Nu kan vi använda oss av formeln (1) längre upp för att få ut tiden.

```
time=volume/flowrate;
```

- Funktionen avslutas med **end** och ska ungefär se ut såhär:

```
function time=myfunction(data,flowrate)
    volume=data(1)*data(2)*data(3);

    time=volume/flowrate;
end
```

Gå nu tillbaka till task1.m och kör koden. Hur lång tid tar det?

Uppgift: Raketacceleration

Problem

En rymdraket skickas rakt uppåt från jorden med en konstant acceleration. Ombord finns en hastighetsmätare som varje sekund mäter och sparar ner hastigheten $[m/s]$. På grund av vibrationer under uppskjutningen uppstår mätfel i hastighetsdatan som gör det svårt att uppskatta accelerationen.

Vad är raketens konstanta acceleration?

Lösningsförslag

Med hjälp av rörelseekvationen,

$$v = at + u, \tag{2}$$

där v är hastigheten, a är accelerationen, t är tiden och u är initialhastigheten kan vi med hjälp av en **linjär regressionanalys** räkna ut accelerationen. En regressionanalys är en metod där man skapar den ekvation som bäst passar mätdatan, i detta fall en linjär ekvation (räta linjens ekvation),

$$y = kx + m. \tag{3}$$

- Vi startar med att göra en ny m-fil. Kör koden efter varje steg för att upptäcka eventuella fel så tidigt som möjligt. Skriv i Command Window:

```
>> edit task2
```

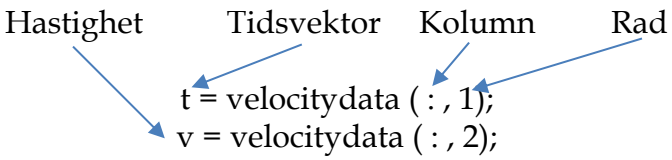
- Hastighetsdatan ligger på https://tekniskfysik.se/student/hjalpmedel/nyborjarmaterial/ under Matlab - Datorövning och måste laddas ner. Enklast är att öppna velocitydata.txt i webbläsare, högerklicka på sidan och klicka på **Spara som**. Det är fördelaktigt att lägga filen i samma mapp som eran nuvarande MATLAB-kod, till exempel kan de vara C:\User*USERNAME*\Documents\MATLAB.

Om ni använder stationära datorerna i datorsalarna sparar ni filerna på servern **H** i exempelvis en ny mapp ”MATLAB” så kommer ni åt datat även ifrån andra datorer (kan dock saknas innan ni skrivit på datorkontrakt). Exempelvis använder ni:

load velocitydata.txt
eller:
load H:\MATLAB\velocitydata.txt

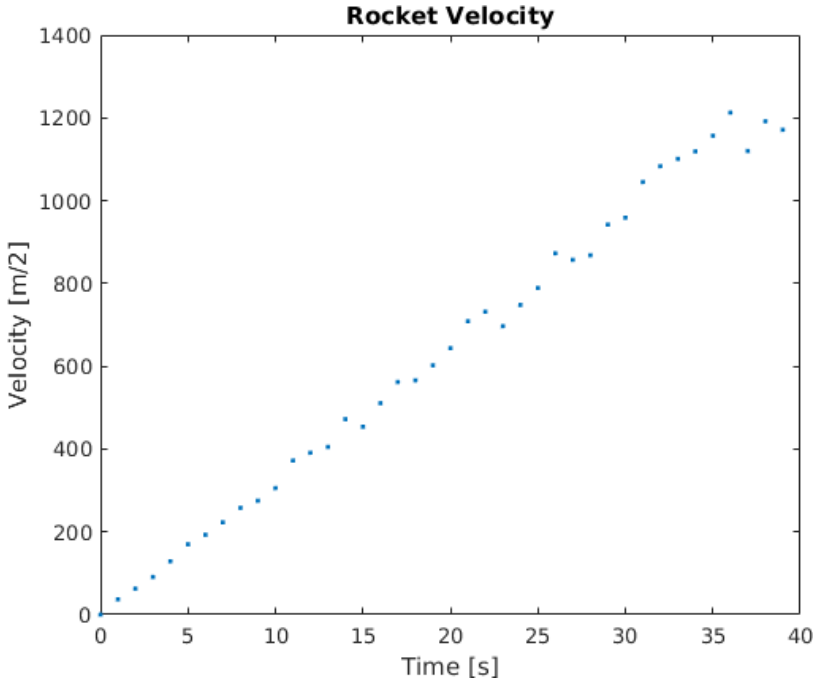
beroende på ifall ni sparat de i samma mapp eller på servern **H.**

- Packa upp datan i två nya vektorer, en för **tid** och en för **hastighet**. Kolonet i parentesen betyder att samtliga värden förs över. Så för tiden, *t*, bildar samtliga värden i kolumn 1 en ny vektor (dubbelklicka på **velocitydata** i Workspace för bättre förståelse).



- Gör en graf av datan med hjälp av **plot-funktionen** för att få en överblick över problemet. **close all** är ett kommando som stänger alla redan öppna grafer från förra gången koden kördes. I plot-funktionen har vi först är x-axeln (tiden, *t*), andra är y-axeln (hastigheten, *v*) och det sista, ’.’, är ett kommando för att skriva ut punkter för varje datavärde istället för en linje (skriv **help plot** i Command Window för mer info). För att lättare förstå vad grafen visar lägg till titel och axelnamn, glöm inte enheter. För att vi senare ska kunna inkludera saker i grafen används kommandot **hold on**.

```
close all
plot(t,v,'.')
title('Rocket Velocity')
xlabel('Time [s]')
ylabel('Velocity [m/2]')
hold on
```



Figur 3: Hastighetsdatan visualiserad.

Grafen visar att ju fortare raketen färdas ju större verkar mätfelet bli.

- Gör en linjär regressionanalys. Skicka in datan för x och y samt en 1:a, som visar att det är en ekvation av första graden vi vill passa in.
-

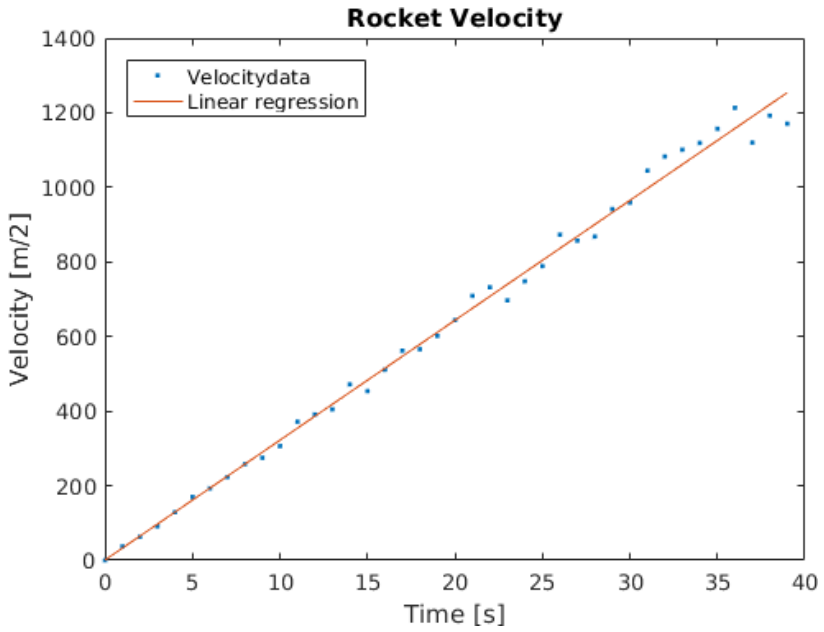
```
LinReg=polyfit (t, v, 1)
```

Det första värdet vi får tillbaka motsvarar accelerationen, a. Är den rimlig? Det andra värdet motsvarar den initiala hastigheten, u. Denna borde vara nära noll, vilket är rimligt eftersom den första mätningen i datan visar att raketen stod stilla.

- För att visualisera den linjära anpassningen används **polyval()** som genererar en ny anpassad vektor. Denna kan vi sedan lägga in i grafen. För att hålla koll på vad det är vi ser läggs också en textförklaring in med hjälp av **legend()**. Med **'location', 'northwest'** lägger sig textförklaringen uppe i vänstra hörnet.

```
p=polyval( LinReg, t, 1) ;

plot (t, p)
legend ('Velocitydata', 'Linear regression', 'location', 'northwest')
```



Figur 4: Hastighetsdatan med en linjäranpassning.

Vi ser hur den linjära ekvationen har anpassat sig efter hastighetsdatan.

Vill ni fortsätta med svårare problem finns det en mer ingående introduktion samt svårare övningar under Blivande student -> Nybörjarmaterial på Teknisk Fysiks hemsida, https://tekniskfysik.se/student/hjalpmedel/nyborjarmaterial/.