TMV156/TMV155E Inledande matematik E, 2009

MATLAB — REPETITION OCH FÖRDJUPNING

Instruktioner

Den här skriften är en genomgång av grunderna i Matlab. Det mesta är att betrakta som repetition men jag adderar några tips och försöker förtydliga några saker. Det kan därför vara en god idé för alla att titta igenom detta även om det främst riktar sig till de som av olika anledningar tycker att de tidigare laborationerna varit svåra. Det gäller att vara vaken här. Det står i princip hela tiden exakt vad du ska göra. Om det ska vara någon nytta med detta så måste du hela tiden fråga dig vad som händer. Innan du skriver in ett kommado och trycker enter, försök alltid att förutsäga vad resultatet blir. Om det inte blir vad du trodde, se till att du kan förklara det för dig eller din lab-kamrat i efterhand!

1. Uppgifter

1.1. **Matlabs olika fönster etcetera.** Den här uppgiften kommer som en beskrivning. Se till att du kan identifiera alla nämnda fönster och dylikt!

När vi kör igång Matlab så kommer, med grundinställningar, fyra olika (del-)fönster att öppnas. Först och främst *kommandofönstret* (Command Window). Där kan man deklarerar variabler och manipulera dessa med Matlabs alla kommandon. Det bör främst användas för att beräkna väldigt enkla saker och framför allt för att söka reda på nya kommandon (t.ex. med `lookfor`) och läsa om dem (`help`).

Vi har också fönstret innehållande *Workspace*. Där visas alla deklarerade variabler och deras typ och storlek samt i viss utsträckning variabelns värde. Om man dubbelklickar på en variabel så öppnas *Array Editor*. Där kan man ändra i variablerna "för hand". Det gör man mest när man hanterar mätdata med mera. I de flesta övriga fall vill vi att olika program ska sköta detta med minimal inblandning av användaren.

Ett tredje fönster visar alla filer och kataloger som finns i den katalog som du befinner dig i. Detta fönster heter *Current Directory*. Med hjälp av det kan du med musen flytta dig runt i katalogträdet samt öppna filer. Det är ibland placerat ovan eller under Workspace. I detta fall kan man klicka fram det dolda med musknappen. I *verktygsfältet* finns en rullist med samma namn, *Current Directory*. Den visar vilken katalog du befinner dig i. Klickar du på pilen så kommer de kataloger du har befunnit dig i nyligen fram. Genom att klicka på någon av dessa flyttar du dig snabbt till önskad katalog.

Ett fönster visar kommandohistorien d.v.s. de kommandon du tidigare har skrivit i kommandofönstret. Det har fått namnet *Command History*. Om du klickar på ett kommando så dyker det upp i kommandofönstret exekveras/körs direkt.

Skriver man `edit` i kommandofönstret så öppnar man text-editorn där man kan skriva m-filer (se nedan). Skriver man `edit filnamn` så öppnas filen med namnet "filnamn". Man kan skapa en ny fil genom att skriva `edit filnamn.m`. Den får då just namnet "filnamn.m".

1.2. **Variabler, tilldelning och typer.** "En variabel är storhet som man själv inför och som man kan tilldela ett värde" skriver P. Jönsson i "Matlab - Beräkningar inom teknik och naturvetenskap".

Date: 2009–10–15, Fredrik Lindgren, Matematiska vetenskaper, Chalmers tekniska högskola.

Uppgift: Fundera över betydelsen av den meningen och fråga labhandledaren om det inte verkar vettigt.

Variabler är helt fundamentalt för all programmering. För att hålla ordning på de objekt som vi vill jobba med så måste de ha ett *namn*. De har alltid en *typ* och ofta ett *värde*, men de kan också vara tomma. Vi har stött på typerna *'doubles', 'function handles', 'logicals'* och *'strings'* eller dubblar, funktionshandtag, logiska variabler och strängar om vi försvenskar.

För att tilldela en variabel ett (nytt) värde så används tecknet = som i det här sammanhanget heter *tilldelningsoperatorn*. Kommandosekvensen

```
>>A=1+1;
>>funktionen=@(x)(x.^2);
>>sant=(A==2);
>>B=[];
```

skapar fyra variabler nämligen variablerna med namnen A, $funktionen$, *sant* och B. De tre första är av typerna double, function handle respektive logical och har värdena 2, $@(x)(x.\hat{}2)$ respektive 1 (sant). Uttrycket $A == 2$ är den logiska satsen "variablen A har värdet två" vilket är sant. Därför får *sant* värdet 1 som ju betyder sant. Om det istället hade stått `sant=(A==1)` så hade *sant* fått värdet 0 som ju betyder falskt. Mer om det i uppgifterna 1.10-1.12. Notera skillnaden på tilldelningsoperatorn = och *jämförelseoperatorn* ==. Variablen B har inget värde alls. Den får då typen double.

Uppgift: Skriv först `clear` i kommandofönstret och se att Workspace är tomt. Skriv sedan kommandosekvensen ovan och notera hur variablerna dyker upp i Workspace och betrakta symbolerna för de olika typerna samt värdena de får.

1.3. **Räkna.** Du bör känna till räkneoperationerna /, *, +, - och ^ samt matematiska funktioner på utdelad lista (t.ex `sin` och `exp`).

1.4. **Skapa och bygga matriser och vektorer.** Matriser är listor med tal, bokstäver eller logiska ettor och nollor om matrisen är av typen double, string respektive logical. Om vi först koncentrerar oss på dubblar så skapas en godtycklig matris genom att tal skrivs in innanför hakparenteser. Kolumner åtskiljs med mellanslag eller kommatecken medan rader åtskiljs med semikolon eller genom att man när man kommit till slutet av en rad trycker enter och så att säga skriver den nya raden på en ny rad. Följande kommandon gör alltså alla samma sak:

```
>>A=[1 2 3;4 5 6]
>>B=[1,2,3;4,5,6]
>>C=[1 2 3
4,5,6]
```

Matriserna A, B och C blir likadana. De har två rader och tre kolumner.

Vi kan sätta ihop matriser som har lika många rader till en ny matris på följande sätt:

```
>>D=[7 8;9 10]
>>E=[A D]
```

Matrisen E har nu två rader och fem kolumner, eller hur? Försöker vi däremot sätta ihop A och D så här,

```
>>F=[A;D];
```

så får vi ett felmeddelande. Vad står det med röd text i kommandofönstret? Vad betyder det?

Transponatet av en matris A betecknas i matematiken med A^T. Det är definierat så att det har den första raden i den ursprungliga matrisen som sin första kolumn, den andra raden som sin andra kolumn och så vidare. Föra att deklarera en variabel AT och ge den värdet av A:s transponat så skriver vi i Matlab

```
>>AT=A'
```

Tecknet ' är alltså transponeringsoperatorn. Nu kan vi skriva

```
>>F=[D;AT];
```

Uppgift: Genomför alla kommandona ovan och se att de stämmer!

Ofta vill man ändra eller plocka ut ett enstaka element i en matris. För att detta ska gå smidigt så har Matlab två olika typer av numrering av matriser. Den första ger positionen som (radnummer,kolumnnummer) på så sätt att

```
>>F(3,2)
```

skriver ut det tal som finns i den tredje raden uppifrån och den andra raden från vänster räknat.

Uppgift: Testa detta och se till att du förstår hur numreringen fungerar! Ändra sedan värdet på element (3,2) till taler 100 genom att skriva

```
F(3,2)=100
```

vid kommandoprompten. Det betyder alltså att element (3,2) i variabeln F tilldelas värdet 100.

Det andra sättet Matlab numrerar sina matriselement på är genom skalära heltal. Det första talet är talet längst upp till vänster, det andra är talet nedanför (om matrisen har mer än en rad) och så vidare till kolumnen är slut. Då fortsätter den på talet högst upp i kolumn två etc. En matris med tre rader och två kolumner har alltså den numrering som ges av följande matris.

```
>>G=[1 4;2 5;3 6]
```

Det betyder alltså att `G(1)` ger svaret 1, `G(2)` ger svaret 2 och så vidare.

Uppgift: Testa detta! Testa också på matrisen F och se till att du kan förutsäga vad du får om du skriver `F(k)` om k är något heltal större än noll och mindre än 11. Vad händer om du skriver `F(11)`?

Den skalära numreringen är framförallt användbar när matrisen är en vektor, det vill säga när den bara har en rad eller en kolumn. Vi ahr sett att vi kan skapa vektorer med kommandona

```
>>x=0:0.01:.5;
>>y=linspace(.5,2,25);
```

Uppgift: Försäkra dig om att du vet vad som händer här ovan! Skapa sedan en ny vektor *hej* genom att sätta y efter x på följande sätt:

```
>>hej=[x y];
```

och lista ut varför följande blir fel.

```
>>hej=[x;y];
```

Storleken på en matris ges av något av kommandona

```
>>stlk=sixe(F)
>>[radnr,kolnr]=size(F)
```

Vi kommenterar det andra uttrycket i avsnitt 1.8. Om man skriver

```
>>length(F)
```

får man det maximala värdet av antalet rader och kolumner. Det är alltså samma sak som att skriva

```
>>max(size(F))
```

som är samma sak som att skriva

```
>>max(stlk)
```

kommandoprompten.

Uppgift: Bekräfta i Matlab!

1.5. **Elementvisa operationer.** Ofta vill man för två matriser (eller vektorer) X och Y addera, multiplicera, subtrahera eller dividera talen på samma plats i matriserna. För addition och subtraktion är det inget problem. Om X och Y är lika stora skriver vi

```
>>Z=X+Y;
```

varvid Z(n,m)=X(n,m)+Y(n,m). Division och multiplikation är dock något annat för matriser, så om vi vill att $Z(n,m) = X(m,n) \cdot Y(m,n)$ så måste vi använda operationen .* (punktgånger). Dvs vi får skriva

```
>>Z=X.*Y;
```

i Matlab. På samma sätt använder vi vid division ./ och om vi vill att $Z(m,n) = X(m,n)^3$ så får vi skriva

```
>>Z=X.^3
```

i Matlab.

1.5.1. *Uppgift:* Med matrisen A som i avsnitt 1.4, generera en ny matris B med slumpade heltal mellan 1 och 3 och med samma storlek som matrisen A med hjälp av kommandot

```
B=random('unid',3,size(A))
```

och se så att detta stämmer. Multiplicera nu A och B på följande sätt och notera skillnaden i resultat!

```
>>A.*B
>>A*B
>>B*A
```

Det första är vad vi vill göra i den här kursen medan de två andra resultaten får sin förklaring när vi läser linjär algebra. Jämför också

```
>>A.^2
>>A^2
>>D^2
```

där D är den kvadratiska matrisen från avsnitt 1.4.

1.6. **Anonyma funktioner och funktionshandtag.** Om vi deklarerar en variabel med namnet $fkn1$ på så sätt att vi skriver

```
>>fkn1=@(x)(x.^2)
```

så händer egentligen två saker. Dels skapas en så kallad *anonym funktion* som kan räkna ut kvadraten av ett tal och dels skapas ett funktionhandtag som är en adress till den anonyma funktionen. Variabeln $fkn1$:s värde är den här adressen, funktionshandtaget. När vi sedan skriver

```
>>y=fkn1(2)
```

så skickar vi talet 2 till den här adressen, det vill säga till den anonyma funktionen som sedan beräknar kvadraten på talet och tilldelar variablen y detta värde, 4. Att den kallas anonym är för att det inte finns någon m-fil där funktionen är definierad. Enda sättet att nå den är via funktionshandtaget. Man kan skapa ett funktionshandtag till en vanlig funktion också. Skriver vi

```
>>fkn2=@sin
```

så blir variablen $fkn2$ en adress till Matlab-funktionen `sin`, vilken inte är en anonym funktion. Skriver vi nu

```
>>fkn2(pi/2)
```

så räknas $sin(\pi/2)$ ut.

Eftersom den anonyma funktionen som $fkn1$ pekar på är definierad med punktupphöjt så kan vi anropa den med vektorer varvid den upphöjer varje element i vektorn till två. Alltså, om vi skriver

```
>>x=-2:.1:2;
>>y1=fkn1(x);
```

så tilldelas x värdena $\begin{pmatrix} -2 & -1,9 & -1,8 & \dots & 0 & \dots & 1,8 & 1,9 & 2 \end{pmatrix}$ och $y1$ tilldelas värdena $\begin{pmatrix} 4 & 3,61 & 3,24 & \dots & 0 & \dots & 3,24 & 3,61 & ,4 \end{pmatrix}$.

Uppgift: Diskutera funktionshandtag och anonyma funktioner med din labkamrat. Känns det svårt att förstå så tar du upp det med labhandledaren. Verifiera det ovan sagda genom att göra sekvensen i Matlab.

1.7. **Grafritning.** Kommandot `plot` tar obegränsat antal inargument (se avsnitt 1.8) på formen `plot(x1,y2,str1,x2,y2,str2,...xn,yn,strn)` där vektorn $x1$ måste vara lika lång som vektorn $y1$, $x2$ lika lång som $y2$ och så vidare. Variabeln $str1$ innehåller inforamtion om vilken färg som grafen ska, vad det ska vara för fason på linjen och hur punkterna ska se ut. Med x och $y1$ från föregående uppgift kan vi göra följande:

```
>>y2=x;
>>figure(1)
>>plot(x,y1,':r',x,y2,'k-s')
```

Här ritas grafen till $y = x^2$ med röd prickad linje medan grafen till $y = x$ ritas med svart heldragegen linje och kvadratiska punkter. Punkterna ritas ut i koordinaterna som ges av talparen som fås när man tar talen på samma plats i de båda vektorerna x och $y2$. Första koordinaten blir till exempel $(x(1), y2(1)) = (-2, -2)$. Hur blir det om vi istället skriver

```
>>figure(2)
>>plot(x,y1,'og--')
>>hold on
>>plot(x,x,'k-s')
```

kan man fråga sig?

Uppgift: Kontemplera över den frågan och utför sedan alla kommandon i detta avsnitt och se till så att resultaten blir vad du förväntar dig. Om inte, vad är det som har hänt?

Ett tips när man ritar grafen till funktioner med singulariteter är att göra som i följande exempel. Vi studerar funktionen $f(x) = 1/(x-2)$ som har en singularitet i $x = 2$. Vi vill rita grafen för $x \in [0, 4]$. Vi definierar följande variabler

```
>>close all
>>x1=linspace(0,2);
>>x2=linspace(2,4);
>>x=[x1 x2];
>>f=@(z)(z-2).^-1;
>>y=f(x);
>>plot(x,y)
```

Grafen vi får saknar den irriterande, nästan lodräta linjen i singulariteten som vi får om vi istället skriver

```
>>figure(2)
>>x2=linspace(0,4);
>>y2=f(x2);
>>plot(x,y)
```

Uppgift: Utför kommandona! Vad i hela friden är skillnaden? Vad beror den på?

I övrigt hänvisas till de tidigare laborationerna.

1.8. **Funktionsfiler med in- och utargument.** Funktioner i Matlab är definierade efter syntaxen `function [y1,y2,...,yn]=filnamn(x1,x2,..., xm)` där m är det *maximala* antalet inargument och n är det *maximala* antalet utargument.

Uppgift: Skapa en m-fil `inutvar.m` och klistra in följande programsekvens i den.

```
function [y1,y2]=inutvar(x1,x2,x3)
y1=x1;
y2=x2*x1;
```

Spara och anropa funktionen från kommandoprompten i tur och ordning med följande anrop:

```
>>inutvar(2)
>>inutvar(2,3)
>>inutvar(2,3,'hjälp!')
>>y=inutvar(2)
>>y=inutvar(2,3)
>>y=inutvar(2,3,'hjälp!')
>>[y,z]=inutvar(2,3)
>>[y,z,u]=inutvar(2,3,5)
```

Försök före varje anrop resonera dig fram till vad som kommer att hända!

1.9. **Färdiga funktioner och hjälp i Matlab.** I avsnitt 1.4 gjorde du detta:

```
>>stlk=sixe(F)
>>[radnr,kolnr]=size(F)
```

Här styrs tydligen utargumenten på ett mer avancerat sätt än i `inutvar.m`. Om vi säger åt funktionen `size.m` att mata ut två variabler så blir den första av en annan sort än om vi bara bad den mata ut en variabel. (Det är möjligt också för Matlab-användaren att skriva sådana program med hjälp av `nargout` och t.ex. en `if-else`-sats. Med kommandot `nargin` kan och `if-else`-satser så kan man få ett program att göra olika saker med olika antal inargument. Vi går inte in djupare på det här men se inlämningsuppgiften för ett exempel på hur det senare kan användas.) En poäng här är att Matlabs färdiga funktioner och de du skriver fungerar efter samma principer. Kan man skriva egna funktionsfiler så kan har man lättare att använda de färdiga och lättare att förstå hjälptexten.

Uppgift: Läs de två första styckena i hjälpen för `size`. Försök förstå de olika resultaten för kommandona ovan utifrån detta. Eller omvänt: Försök förstå hjälptexten utifrån vad kommandona returnerar!

1.10. **Logiska satser.** En logisk sats eller ett logiskt uttryck är ett påstående som antingen är sant (1) eller falskt (0). De vanligaste logiska satserna är en jämförelse mellan två tal. För att göra en sådan behövs en *jämförelseoperator*, se avsnitt 1.11. Vi kan operera på två logiska variabler med logiska operatorer. Med två satser skapar vi t.ex. en tredje genom att säga att båda två måste vara sanna samtidigt, se avsnitt 1.12

1.11. **Jämförelseoperatorer.** Jämförelseoperatorerna är beskrivna i föreläsningsanteckningarna. Titta där om du är osäker.

Uppgift: Låt a) $A = 1$ ovh $B = 2$ respektiveb) $A = B = 3$ och beräkna sanningshalterna i följande satser. Skriv först ner det och testa sedan i Matlab.

```
A==B
A~=B
A<=B
A>=B
A>B
A<B
```

1.12. **Logiska operatorer.** Även logiska operatorer är beskrivna i föreläsningsanteckningarna. Vi har också konstaterat det användbara med lite konstiga faktumet att doubles betraktas som logiska variabler om vi använder dem som om de vore det och logiska variabler betraktas som doubles om vi använder dem som det.

Uppgift: Låt a) $A = 1$ och $B = 0$ och b) $A = 1$ och $B = 1$. Bestäm i båda fallen sanningshalten av följande logiska operationer. Skriv först ner svad du tror svaret blir och testa sedan i Matlab. På ett ställe behandlas de båda variablerna som om de vore numeriska värden (t.ex. dubblar), var?

```
A&B
A~B
A|B
AxorB
A&(A<B)
```

1.13. **Loopar och alternativkonstruktioner.** Vi har under kursens gång berört `for`-loopar och `while`-loopar samt if-else-satser. Två av dessa konstruktioner kräver logiska uttryck. Vilka? Se i övrigt Laboration 2.