

# Introduktionskurs i MATLAB

## Datorlektion 2

Denna version: 11 januari 2010

Den här datorlektionen handlar mestadels om programmering. De mest grundläggande begreppen introduceras, och vi ger ett förslag på hur man kan angripa den typ av programmeringsproblem som man vanligtvis vill lösa i MATLAB. För dig som programmerat förut är antagligen det mesta bekant.

### 1 Förberedelseuppgifter

1. Gå igenom momenten från lektion 1 och se till att du har koll på dem.
2. Läs igenom avsnitt 2 och gör uppgiften.

### 2 Dokumentation

Redan under datorlektion 1 nämnde vi vikten av att kommentera sin kod. Hur mycket man bör dokumentera och kommentera sin kod beror delvis på användningsområdet:

1. Man vill kunna använda funktionen en annan dag, och förstå vad den gör.
2. En kompis vill kunna använda funktionen, och vill då kunna förstå vad den gör så att han eller hon är säker på att den fungerar. Han eller hon vill också att resultatet presenteras på ett tydligt sätt.

3. Du vill övertyga chefen om att funktionen är så användbar att det lönar sig att köpa in MATLAB till ert företag. Koden bör då inte bara vara väldokumenterad, utan också presentera resultatet på ett snyggt, prydligt och tydligt sätt.

Byt ut "kompis" mot "kursledningen" eller "labassistenten" och du får en uppfattning om den ambitionsnivå du bör använda åtminstone på laborationen i den här kursen.

Några saker att tänka på för att åstadkomma tydlig och prydlig kod:

- Skriv din kod i en logisk ordning: "Först gör vi detta, sedan gör vi detta". Använd rader som börjar med %% för att dela upp koden i delberäkningar, och kommentarer för att tala om vad som händer i de olika delarna.
- Använd beskrivande namn på dina variabler. Följande kod gör samma uträkning som exemplet med lånberäkning från lektion 1, men är något svårare att förstå:

```
function y = test(x1,x2)
% Ett test
a = 0.01;
b = 0.1;
y = x1 + x1*a - x2*b;
```

- Skriv hjälptexter och förklaringar till de in- och utmatningar du gör. Hjälptexten skrivs som kommentarer på raden/raderna efter funktionsdeklarationen (`function y = ...`) och är det som skrivs ut om du skriver `help filnamn` på din m-fil. (Testa det på exemplet `lanberakning`.)
- Glöm inte att ändra kommentarerna om du ändrar i koden. Felaktiga kommentarer är sämre än inga alls, och gör koden mer kryptisk istället för tvärtom.

**Uppgift:** Komplettera funktionen `kurvplot` från lektion 1 med kommentarer och hjälptexter (gör först klart funktionen om du inte redan har gjort det).

### 3 Kontrollstrukturer

#### Villkor — if-satser

Ofta vill man göra olika saker beroende på värdet på någon variabel. I låneexemplet från lektion 1 kan man t ex vilja kontrollera om skulden är färdigbetalad. I MATLAB, liksom i de flesta andra programspråk finns därför kontrollstrukturen `if`. Ett exempel på hur den kan användas i Pelles program för att kontrollera om skulden är färdigbetalad och i så fall sätta den till noll:

```
function ny_skuld = lanberakning(skuld,lon,ranta,...
                                betalningsgrad)
% ny_skuld = lanberakning(skuld,lon,ranta,betalningsgrad)
% Beräknar skulden nästa månad.
% Mata in skuld, lön, ränta och betalningsgrad

ny_skuld = skuld + skuld*ranta - betalningsgrad*lon;

% Testa om skulden är färdigbetalad:
if ny_skuld <= 0
    ny_skuld = 0;
    disp('Skulden är färdigbetalad!')
else
    disp('Skulden är inte färdigbetalad ännu.')
end
```

Tre nyckelord: `if`, `else` och `end`. Efter `if` står vårt villkor: `ny_skuld<=0` (`<=` står för mindre än eller lika med). I det här exemplet är villkoret en olikhet, men man kan också kontrollera t ex om två variabler är lika genom ett dubbelt likhetstecken (`==`), eller om de är olika genom `~=`. Om villkoret är sant utförs raden (eller raderna) mellan `if` och `else`, annars utförs de mellan `else` och `end`. Allmänt kan man skriva syntaxen som

```
if villkor
    programrader
else
    programrader
end
```

MATLAB tolkar nollskilda värden som sanna och 0 som falskt. Villkoret kan alltså vara ett uttryck som beräknas till något nollskilt eller 0. Använd `help relop`, MATLAB-dokumentationen eller boken [1] för att ta reda på vilka jämförelseoperationer som finns.

#### Felhantering

Ibland använder man `if`-satsen för att hitta ett inmatningsfel. Eftersom vi i exemplet ovan lät räntan och betalningsgraden vara inargument kan vi t ex vilja kolla att räntan är positiv. I annat fall vill vi inte göra någon beräkning, utan skriva ut en varningstext och avsluta funktionen. Till det används kommandot `error`. Lägg till följande test innan `ny_skuld` beräknas:

```
% Testa om den inmatade räntan är positiv:
if ranta<0
    error('Räntan måste vara positiv!')
end
```

Resten av funktionen ser ut som förut. Prova vad som händer om du matar in en negativ ränta.

Kommandot `warning` kan användas för att varna för mindre allvarliga fel, som inte avbryter funktionen.

**Uppgift:** Utöka din funktion `kurvplot` så att den kontrollerar att intervallet anges med det minsta värdet före det största. Om intervallet angetts på något annat sätt kan du antingen skriva ut ett felmeddelande och avsluta funktionen, eller byta plats på värdena och fortsätta kurvritningen. Även då bör du skriva ut en varning.

## Upprepningar

Som tidigare nämnts vill man ofta köra samma eller ungefär samma kod ett flertal gånger. Är detta flertal stort finns det bättre sätt än att köra funktionen ”för hand” om och om igen.

### for-loopar

Antag att Pelle inte bara vill veta skulden nästa månad utan om t ex 12 månader. Istället för att köra funktionen 12 gånger med olika inargument kan vi använda en så kallad loop. Vi inför också en ny inparameter: antal månader.

```
function ny_skuld = lanberakning(skuld,lon,ranta,...
    betalningsgrad,manader)
% ny_skuld = lanberakning(skuld,lon,ranta,...
%     betalningsgrad,manader)
% Beräknar skulden ett antal månader framåt.
% Mata in skuld, lön, ränta och betalningsgrad per
% månad samt antal månader.

% Testa om den inmatade räntan är positiv:
if ranta<0
    error('Räntan måste vara positiv!')
end
```

```
% Beräkna den nya skulden:
for t = 1:manader
    skuld = skuld + skuld*ranta - betalningsgrad*lon;
end
ny_skuld = skuld;

% Testa om skulden är färdigbetalad:
if ny_skuld <= 0
    ny_skuld = 0;
    disp('Skulden är färdigbetalad!')
else
    disp('Skulden är inte färdigbetalad ännu.')
end
```

Nyckelorden `for` och `end` avgränsar upprepningsstrukturen. Variabeln `t` stegas igenom det som står efter `=`, och för varje värde körs programraderna mellan `for` och `end`. Kolon-notationen, `:`, känner vi igen från datorlektion 1. Variabeln `t` kommer alltså vara 1 första gången programraden (eller raderna) körs, 2 andra gången osv upp till värdet på månader (`t` ex 12). Varje gång beräknas ett nytt värde på variabeln `skuld`. Prova att ta bort semikolonet efter beräkningen för att se delresultaten skrivas ut.

Ovan används `t` bara för att hålla räkningen. Man kan också använda den inne i loopen. I exemplet kan man vilja få ut skulden månad för månad istället för bara den sista månaden. Låt därför `skuld` bli en vektor, där `skuld(t)` anger skulden månad `t`. Initiera vektorn, före `for`-loopen, med

```
skuld = [skuld; zeros(manader,1)];
```

Detta skapar en vektor där vi kan spara skulden månad för månad. Inuti `for`-loopen beräknar vi nu skulden en viss månad med

```
skuld(t+1) = skuld(t) + skuld(t)*ranta - ...
    betalningsgrad*lon;
```

I övrigt ser funktionen likadan ut.

Testa den varianten av funktionen, och rita en figur över hur skulden ändras, månad för månad. Skriv ut varje iteration så att du ser hur vektorn skuld växer. Detta är också användbart vid felsökning.

**Uppgift:** Utöka och modifiera funktionen `kurvplot`, så att den ritar kurvorna  $f(x) = 1 - \frac{e^{-tx}}{\sqrt{1-t^2}} \sin(x\sqrt{1-t^2} + \arccost)$  för olika värden på  $t$ . Rita kurvorna i samma figur, där  $t$  varierar mellan 0.1 och 0.9 med steget 0.2. Du skall alltså använda en `for`-loop för hantera olika  $t$  inne i funktionen.

## while

Man vet inte alltid på förhand hur många gånger man vill upprepa en beräkning. Pelle vill t ex veta när lånet är färdigbetalt. Då kan man använda kommandot `while`. `while` kontrollerar ett villkor (på samma sätt som `if`) och utför vissa programrader om det är uppfyllt. Därefter kontrolleras villkoret igen, och programraderna utförs igen om villkoret är uppfyllt. När villkoret inte är uppfyllt längre avslutas upprepningen. Vi modifierar exemplet för att räkna ut vilken månad lånet är färdigbetalt, och använder `while`:

```
function slutmanad = lanberakning(skuld,lon,ranta,...
    betalningsgrad)
% slutmanad = lanberakning(skuld,lon,ranta,...
%     betalningsgrad)
% Beräknar vilken månad skulden är färdigbetald
% Mata in skuld, lön, ränta och betalningsgrad per
% månad.
```

```
% Testa om den inmatade räntan är positiv:
if ranta<0
    error('Räntan måste vara positiv!')
end
```

```
manad_nr = 1;
while skuld(manad_nr)>0
    % Beräkna den nya skulden:
    skuld(manad_nr+1) = skuld(manad_nr) + ...
        skuld(manad_nr)*ranta - betalningsgrad*lon;
    manad_nr = manad_nr+1;
end
slutmanad = manad_nr;
```

När man gör en `while`-loop är det viktigt att se till att villkoret för upprepning inte alltid är uppfyllt, för att undvika att hamna i en oändlig loop<sup>1</sup>. Detta bör man förstås kontrollera innan man går in i loopen. I fallet ovan finns den risken om skulden inte minskar för varje månad. Vi kan kontrollera att det inte är så i samband med att vi kontrollerar att räntan är positiv. Ange ett villkor för att loopen inte ska bli oändlig (och skulden så småningom bli betald):

.....

Modifiera koden så att vi inte riskerar att hamna i en oändlig loop. Förutom villkoret ovan kan man också sätta en maxgräns på antalet iterationer, dvs modifiera villkoret i `while`-satsen så att `manad_nr` inte är större än t ex 200. Man kan också tänka sig att införa denna maxgräns som ytterligare ett inargument till funktionen.

Sammanfatta eller ge ett exempel på när man kan/bör använda `for`-loopar och när det är bättre med `while`-loopar:

<sup>1</sup>Skulle man hamna i en oändlig loop kan man avbryta beräkningarna genom att trycka på Ctrl-c.

.....

**Uppgift:** Maxvärdet hos funktionen  $f(x) = 1 - \frac{e^{-tx}}{\sqrt{1-t^2}} \sin(x\sqrt{1-t^2} + \arccost)$  beror på vilket värde  $t$  har. Anta att vi bara är intresserade av positiva  $x$ -värden. Om vi vill ha  $t$  så litet som möjligt, men ändå ha ett maxvärde som är mindre än t ex 1.2 kan vi prova olika  $t$ -värden i en `while`-loop. Modifiera funktionen så att den tar fram det minsta  $t$ -värdet som ger ett visst maxvärde. Tänk på att undvika oändliga loopar.

I lånberäkningsfunktionen ovan räknar vi förutom slutmånaden ut skulden månad för månad. Eftersom funktionsvariabler är lokala finns dock detta inte kvar i Workspace när funktionen avslutat sin körning. Vill vi ha ut den kan vi ange två utargument:

```
function [slutmanad,skuldutveckling] = lanberakning(...  
    skuld,lon,ranta,betalningsgrad)
```

Vi måste då se till att variabeln `skuldutveckling` tilldelas ett värde någonstans i funktionen. Funktionen kan sedan anropas med två utargument, t ex:

```
>> [m, s] = lanberakning(100000,25000,0.01,0.1)
```

Modifiera funktionen så att den producerar två utargument.

## 4 Exempel

Nedanstående exempel är hämtat från kursen TATM18, Linjär algebra. Det är alltså inte primärt avsett att lösas med MATLAB, men varför inte tillämpa våra MATLAB-kunskaper på det.

En lokal biluthyrningsfirma har två kontor, ett i Linköping och ett i Norrköping. Det har visat sig att under en månad återlämnas 70% av bilarna som hyrts i Linköping tillbaka dit, medan 30% av dem lämnas tillbaka i Norrköping. För Norrköpingskontoret gäller att 40% återlämnas där, medan 60% lämnas i Linköping. Företaget har 15 bilar. Varje bil hyrs ut en gång per månad. Hur många ska placeras i Linköping respektive Norrköping? Vad gäller om man köper in fler bilar eller om procentalen ändras?

Problemet har en analytisk lösning (bonusfråga för intresserade), men vi ska lösa det genom simulering. Det kan ju också vara intressant att se hur fördelningen av bilparken ändras över tid.

**Uppgift** (tänk på att dokumentera och kommentera din kod efter hand):

1. Börja med t ex 1 bil i Linköping och 14 i Norrköping månad 1. Hur ser det ut månad 2? 3? 4? Skriv ett skript som simulerar ett antal månader. Kolla att det stämmer. Plotta resultatet.
2. Prova skriptet för olika många bilar, olika många månader.
3. Generalisera skriptet till en funktion. In: antal bilar, antal månader. Ut: Prydlig figur, med beskrivande text.
4. Ännu generellare blir funktionen om man även kan mata in procentsatser. Kontrollera att samtliga bilar återlämnas (procentsatserna summerar till 100).
5. Fördjupningsuppgift: Inkludera flera städer. Här måste man fundera över vilket inmatningsformat som är lämpligt.
6. Ska man vara riktigt noga är det bara heltalslösningar som är intressanta, man lär ju inte lämna igen en halv bil i Linköping och en halv i Norrköping. Om du vill kan du inkludera även det i din funktion.

Uppdelningen i punkter illustrerar ett möjligt angreppssätt på den typ av problem som man ofta löser med MATLAB. Man börjar med att lösa ett mindre delproblem, där man lätt kan kontrollera svaret. Visualisera svaret med lämpliga figurer, som dokumenteras med lämplig text. Om hela problemet inte är löst fortsätter man med att lösa ett annat delproblem, som sedan kombineras med det man redan gjort.

En av de vanligaste programmeringsmetoderna kallas stegvis förfining. Stegvis förfining handlar om att dela upp sitt problem i delproblem, som kan lösas i tur och ordning. Förhoppningsvis är vart och ett av delproblemen lättare att lösas. Antingen är de så små att man vet direkt hur de ska lösas, eller också delar man upp dem i sin tur.

Den mängd metoder som finns för programmering ligger långt utanför den här kursens utrymme, men har du en metod som du tycker fungerar bra för dig finns (troligtvis) inget hinder att använda den när du programmerar i MATLAB.

## 5 Fördjupning

I fördjupningen till den här lektionen rekommenderar vi att du fördjupar dig allmänt om programmering, och i mån av intresse även undersöker hur man skriver effektiv kod, och/eller (företrädesvis om du programmerat tidigare) tittar på objektorientering i MATLAB. Använd boken [1] eller MATLAB-dokumentationen.

### 5.1 Allmänt om programmering

Kapitel 9 och 10 i boken [1] handlar om funktioner.

**Förslag på uppgifter:** 45, 48, 49, 51, 52, 53

Du kan också utöka funktionen `lanberakning` som vi tittat på tidigare, med saker du tycker kan vara användbara. Några idéer kan vara att rita skuldutvecklingen, att ange ränta per år istället för per månad, att lönen räknas upp en gång per år.

Under "Getting started" i MATLAB-dokumentationen finns ett avsnitt om programmering. Du kan också läsa vidare text om "Function functions" under "Mathematics". Under "Programming" finns också bla "Programming tips".

### 5.2 Effektiv kod

Några sätt att göra sin kod snabbare finns beskrivna i kapitel 14 i boken [1].

Fundera på om exemplet i avsnitt 4 kan göras effektivare genom att använda vektorisering. Uppgift 48 i boken tar upp några olika sätt att lösa problemet. Vilket är snabbast?

I MATLAB-dokumentationen hittar du ett avsnitt om "Improving Performance and Memory Usage" under "Programming".

### 5.3 Objektorientering

MATLAB har ett visst stöd för objektorienterad programmering. Fler datatyper och objektorientering finns beskriven i kapitel 11 och 12 i boken [1]. Se även "Classes and Objects" under "Programming" i MATLAB-dokumentationen.

## Referenser

- [1] F. Gustafsson och N. Bergman, *Matlab for Engineers Explained*, Springer Verlag, 2003.