Kort innføring i MATLAB

Tom Lindstrøm og Klara Hveberg

Matematisk institutt
Universitetet i Oslo

Innhold

1 Innføring i MATLAB **1**
A.1 Det aller enkleste 1
A.2 Matriser og vektorer 4
A.3 Komponentvise operasjoner 7
A.4 Grafer 9
A.5 Tredimensjonale grafer 12
A.6 Mer om matriser 13
A.7 Radoperasjoner i MATLAB 16
A.8 Ta vare på arbeidet ditt 19
A.9 Programmering i MATLAB 21
A.10 m-filer, funksjoner og script 25
A.11 Anonyme funksjoner og linjefunksjoner 28
A.12 Symbolske beregninger 30
A.13 Feilsøking 32

Kapittel 1

Innføring i MATLAB

Dette appendikset gir en kort innføring i MATLAB med tanke på behovene i denne boken. Hensikten er å gi deg litt starthjelp slik at du kommer i gang med å bruke programmet. Vi forutsetter at du sitter ved maskinen og taster inn kommandoene vi gjennomgår, men det kan kanskje være lurt å ha bladd fort gjennom seksjonene du skal arbeide med på forhånd slik at du vet hva som vil bli tatt opp. Vær oppmerksom på at du også kan få hjelp fra MATLABs innebygde hjelpemenyer og demo'er (kommandoen `>> help help` gir deg en oversikt over hva som er tilgjengelig). Ønsker du å arbeide med Python istedenfor MATLAB, finnes det et tilsvarende appendiks for Python på bokens nettsider. Python er et generelt programmeringsspråk der matematikkommandoene er lagt tett opptil MATLAB.

Appendikset er ganske uavhengig av resten av boken, og du kan lese det når du selv vil. Siden MATLAB er et matrisebasert program, kan det være lurt å vente til du vet hva en matrise er, og siden MATLAB brukes mest intensivt i kapittel 4 og 5, er det sikkert lurt å være godt kjent med programmet før du kommer så langt. Hvis du begynner å lese tidlig, må du regne med at du av og til støter på matematikkbegreper du ikke har lest om i boken ennå. Da er det bare å hoppe over dem og heller komme tilbake senere.

I de første seksjonene i appendikset (1-6) konsentrerer vi oss hovedsakelig om hvordan du kan bruke MATLAB til å utføre enkle beregninger. I de siste seksjonene (7-13) ser vi på litt mer avanserte ting som hvordan du tar vare på (utvalgte deler av) arbeidet ditt, hvordan du programmerer i MATLAB, og hvordan du lager dine egne MATLAB-rutiner som du kan kalle på igjen og igjen. Det kan hende at du har bruk for noe av dette stoffet før du er ferdig med de første seks seksjonene.

I dette appendikset vil du se mange eksempler på MATLAB-kode, altså på MATLAB-kommandoer og MATLAB-programmer. Vi har prøvd å legge inn mange kommentarer i koden slik at det skal være mulig å lese seg til hva som skjer. En kommentar begynner alltid med symbolet `%` og er ikke en del av selve programmet. Når du begynner å lage dine egne programmer, er det en god vane å legge inn forklarende kommentarer i koden, både for din egen del og andres.

A1: Det aller enkleste

Du åpner MATLAB som et hvilket som helst annet program. I Unix skriver du `matlab &`, mens du i Windows-lignende omgivelser klikker på MATLAB-ikonet. I begge tilfeller får du opp et kommandovindu med et MATLAB "prompt" `>>` (avhengig av oppsettet på maskinen er dette kommandovinduet enten ett av flere MATLAB-vinduer eller en del av et

større vindu). Etter promptet kan du skrive en kommando du vil at MATLAB skal utføre. Skriver du f.eks.

```
>> 3+4
```

og skifter linje, svarer MATLAB med

```
ans=7
```

og gir deg et nytt "prompt" til neste regnestykke (prøv!). Du kan prøve de andre standard regneoperasjonene ved å skrive inn etter tur `3-4, 3*4, 3/4, 3^4` (husk å avslutte med et linjeskift for å utføre kommandoene). MATLAB bruker den vanlige prioriteringsordningen mellom regneoperasjonene slik du er vant til fra avanserte lommeregnere. Kommandoen

```
>> 8^2/3
```

gir svaret 21.3333, dvs. 64/3, mens

```
>> 8^(2/3)
```

gir svaret 4.0000. Vær forøvrig klar over at på enkelte maskiner er `^`-tasten litt trøblete når man bruker MATLAB. Som regel løser problemet seg hvis man taster "mellomrom" etter `^` og så fortsetter med neste symbol.

MATLAB kjenner alle vanlige (og mange uvanlige!) funksjoner, så skriver du

```
>> sin(0.94)
```

svarer MATLAB med sinus til 0.94 radianer. Vær oppmerksom på at MATLAB er nøye på multiplikasjonstegn: Skriver du `>> 15sin(0.94)`, svarer MATLAB med en feilmelding – du må skrive `>> 15*sin(0.94)` for å få regnet ut $15 \sin 0.94$.

En funksjon du må passe litt på, er eksponentialfunksjonen e^x som du må skrive som `exp(x)`. Vil du regne ut $e^{2.5}$, skriver du altså `exp(2.5)` (og ikke `e^2.5`). I tillegg til eksponentialfunksjonen må du passe litt på arcusfunksjonene som i MATLAB heter `asin`, `atan` osv, og *ikke* `arcsin`, `arctan` som man kanskje skulle vente. I tillegg kan det være greit å vite at kvadratrotfunksjonen heter `sqrt` og at absoluttverdifunksjonen heter `abs`. Skriver du

```
>> exp(sqrt(abs(-0.7)))
```

regner MATLAB ut $e^{\sqrt{|-0.7|}}$. De andre standardfunksjonene heter det du er vant til: `sin, cos, tan, cot, log`.

Vår gamle venn π kalles i MATLAB for `pi`. Kommandoen

```
>> pi
```

gir svaret `3.1416`. Ønsker du flere (synlige) desimaler i utregningene, kan du skrive

```
>> format long
```

Kommandoen `>> pi` blir da besvart med `3.14159265358979`. Vil du tilbake til kort format (det blir fort uoversiktlig med mange desimaler i alle tall), skriver du

```
>> format short
```

Ber du MATLAB regne ut $\sin \pi$, får du deg kanskje en overraskelse. Istedenfor 0, gir MATLAB deg svaret `1.2246e-16` (dvs. $1.2246 \cdot 10^{-16}$). Dette skyldes at MATLAB primært er et numerisk beregningsprogram som regner med avrundede tall.

Skal du bruke et tall flere ganger i en beregning, kan det være greit å gi det et navn. Ønsker du at 0.3124 heretter skal betegnes med `a`, skriver du rett og slett

```
>> a=0.3124
```

Navnsetter du på tilsvarende måte

```
>> b=2.41
```

kan du regne ut produktet ved å skrive

```
>> a*b
```

La oss avslutte denne innledende seksjonen med noen MATLAB-knep det kan være greit å vite om (blir det for mye på en gang, får du heller komme tilbake til denne delen senere). I MATLAB kan du ikke endre på en kommando som er blitt eksekvert. Har du laget en liten trykkfeil, må du derfor føre inn kommandoen på nytt. Dette kan du gjøre ved klipping-og-liming, men du kan også bruke piltastene opp og ned. En oversikt over tidligere kommandoer ligger i det lille vinduet som heter "Command History", og du kan navigere frem og tilbake i disse kommandoene ved å bruke piltastene. Klikker du på den gamle kommandoen du vil bruke på nytt, kommer den opp i arbeidsvinduet i redigerbar form.

Dersom du ønsker at MATLAB skal vente med å eksekvere en linje til senere, holder du skift-tasten (for skifte til stor bokstav) nede mens du slår linjeskift. MATLAB vil nå vente med å utføre kommandoen til du slår et linjeskift uten å bruke skift-tasten. Synes du at en kommando blir svært lang og vil fortsette den på en ny linje, kan du skrive tre punktum etter hverandre og så skifte linje. MATLAB skjønner da at du ikke er ferdig med kommandoen og venter med å utføre den.

Får du problemer med en kommando og ønsker å avbryte, trykker du Ctrl-c (altså kontrolltasten og c samtidig).

Et siste nyttig triks: Ønsker du at MATLAB skal utføre en kommando uten å skrive resultatet i vinduet, taster du et semikolon etter kommandoen. Skriver du f.eks.

```
>> c=a*b;
```

vil c få verdien ab, men resultatet kommer ikke opp på skjermen. Du vil ofte ha behov for å gi MATLAB input i form av lange ramser av tall, og da er det lurt å bruke dette trikset så skjermen ikke fylles opp av uinteressante tall.

Oppgaver til seksjon 1

1. Bruk MATLAB til å regne ut: $2.2 + 4.7$, $5/7$, 3^2, $\frac{2\cdot 3-4^2}{13-2.2^2}$

2. La MATLAB regne ut verdiene, og sjekk at resultatene er rimelige: e^1, $\sqrt{16}$, $\cos\pi$, $\sin\frac{\pi}{6}$, $\tan\frac{\pi}{4}$, $\arcsin\frac{1}{2}$, $\arctan 1$.

3. Definer $x = 0.762$ og $y = \sqrt{9.56} + e^{-1}$ og regn ut: $x + y$, xy, $\frac{x}{y}$ og $\sin x^2 y$.

4. MATLAB-funksjonene `floor`, `ceil` og `round` runder av desimaltall til heltall på ulike måter. Eksperimenter med uttrykk av typen `floor(3.3)`, `ceil(-7.23)` osv. for å finne ut hvordan funksjonene fungerer.

5. Kommandoen `rand` returnerer et "tilfeldig" tall mellom 0 og 1. Bruk den noen ganger så du ser hvordan den fungerer. Du kan også kombinere kommandoen med funksjonene i oppgave 4, f.eks. ved å skrive `floor(100*rand)`. Hva slags tilfeldige tall får du nå? Lag en "terningkastsimulator" som returnerer tallene fra 1 til 6 med sannsynlighet $\frac{1}{6}$.

A2: Matriser og vektorer

MATLAB egner seg utmerket for matriseregning. Ordet MATLAB er faktisk en forkortelse for "MATrix LABoratory", og programmet startet sin karriere som et undervisningsverktøy i lineær algebra.

Dersom vi ønsker å definere matrisen

$$A = \begin{pmatrix} 2 & -3 & 1 \\ 0 & 1 & -3 \\ -4 & 2 & 1 \end{pmatrix}$$

i MATLAB, skriver vi

```
>> A=[2 -3 1
0 1 3
-4 2 1]
```

(legg merke til at vi bruker hakeparenteser for å definere matriser). Synes du dette tar for stor plass, kan du isteden skrive

```
>> A=[2 -3 1;0 1 -3;-4 2 1]
```

der du bruker semikolon til å skille en rad fra den neste. Mellom to elementer på samme rad er det nok med et mellomrom. Synes du dette blir uoversiktlig, kan du bruke komma isteden:

```
>> A=[2,-3,1;0,1,-3;-4,2,1]
```

MATLAB utfører regneoperasjoner for matriser med enkle kommandoer. Har du lagt inn matrisene A og B, vil kommandoene

```
>> A+B
>> A-B
```

```
>> A*B
```

få MATLAB til å regne ut henholdsvis summen $A+B$, differensen $A-B$ og produktet AB. Dette forutsetter at matrisene har riktige dimensjoner slik at regneoperasjonene er definert (hvis ikke svarer MATLAB med en feilmelding av typen: `Matrix dimensions must agree`). Kommandoen

```
>> 3*A
```

ganger matrisen A med tallet 3, og kommandoen

```
>> A^7
```

regner ut sjuende potensen til A (dvs. A ganget med seg selv 7 ganger). For å finne den transponerte matrisen A^T skriver du

```
>> A'
```

og for å finne den inverse matrisen A^{-1} skriver du

```
>> inv(A)
```

MATLAB regner ut determinanten til A når du skriver

```
>> det(A)
```

Husk at kommandoene `inv` og `det` forutsetter at A er en kvadratisk matrise.

MATLAB oppfatter vektorer som spesialtilfeller av matriser. En radvektor

$$\mathbf{b} = (b_1, b_2, \ldots, b_n)$$

er altså en $1 \times n$-matrise, og vektoren $\mathbf{b} = (-2, 5, 6, -4, 0)$ lastes derfor inn med kommandoen

```
>> b=[-2 5 6 -4 0]
```

En søylevektor

$$\mathbf{c} = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{pmatrix}$$

oppfattes som en $m \times 1$-matrise. Du kan laste inn vektoren $\mathbf{c} = \begin{pmatrix} 7 \\ -3 \\ 2 \end{pmatrix}$ ved for eksempel å skrive

```
>> c=[7;-3;2]
```

(husk at semikolon markerer linjeskift). Når du skriver vektorer i MATLAB, er det viktig å ha tenkt igjennom om du ønsker radvektorer eller søylevektorer; du kan ikke regne med at MATLAB skjønner at du mener en søylevektor når du skriver en radvektor!

Har du lastet inn en matrise A og en søylevektor $\mathbf{c}$, kan du få MATLAB til å regne ut produktet $A\mathbf{c}$ ved å skrive

```
>> A*c
```

MATLAB har spesialkommandoer for å regne ut skalar- og vektorprodukt:

```
>> dot(a,b)
>> cross(a,b)
```

Disse kommandoene fungerer både for rad- og søylevektorer, men den siste forutsetter naturlig nok at vektorene er tredimensjonale.

Av og til er det nyttig å eksperimentere med matriser av tilfeldige tall, f.eks. for å teste en hypotese du har. Kommandoen `>> rand(m,n)` produserer en $m \times n$-matrise der komponentene er tilfeldige tall mellom 0 og 1.

Oppgaver til seksjon 2

1. La

$$A = \begin{pmatrix} 1 & 3 & 4 & 6 \\ -1 & 1 & 3 & -5 \\ 2 & -3 & 1 & 6 \\ 2 & 3 & -2 & 1 \end{pmatrix} \text{ og } B = \begin{pmatrix} 2 & 2 & -1 & 4 \\ 2 & -1 & 4 & 6 \\ 2 & 3 & 2 & -1 \\ -1 & 4 & -2 & 5 \end{pmatrix}.$$

Bruk MATLAB til å regne ut; A^T, B^T, $(AB)^T$, A^TB^T, B^TA^T, A^{-1}, B^{-1}, $(AB)^{-1}$, $A^{-1}B^{-1}$, $B^{-1}A^{-1}$. Blir noen av resultatene like?

2. Bruk kommandoen `rand` til å generere noen tilfeldige matriser av ulike dimensjoner.

3. Vi har to vektorer $a = (1, -2, 3)$ og $b = (2, 2, -4)$. Sjekk at lengdene til vektorene kan finnes ved kommandoene `>> norm(a)` og `>> norm(b)`. Forklar at du kan finne vinkelen mellom vektorene ved kommandoen

```
acos(dot(a,b)/(norm(a)*norm(b)))
```

Finn vinkelen.

4. La

$$A = \begin{pmatrix} 1 & 2 & -1 \\ 3 & -1 & 0 \\ -4 & 0 & 2 \end{pmatrix}$$

Finn A^{-1}, A^T og $\det(A)$.

5. (Denne oppgaven forutsetter at du er kjent med egenverdier og egenvektorer.) Skriver du

```
>> [u,v]=eig(A)
```

vil MATLAB definere to matriser u og v. Søylene i matrisen u er egenvektorene til A, mens v er en diagonalmatrise der elementene på diagonalen er egenverdiene til A. Egenvektorene og egenverdiene kommer i samme rekkefølge slik at den første egenverdien tilhører den første egenvektoren osv. Kjør

```
>> A =[3     1     2
 3     0     4
 2     1     4];
>> [u,v]=eig(A)
```

og se hva som skjer. Sjekk at det virkelig er egenverdier og egenvektorer du har fått ut. Vær oppmerksom på at MATLAB alltid skalerer egenvektorene slik at de har lengde 1.

A3: Komponentvise operasjoner

Ovenfor har vi sett på de algebraiske operasjonene som brukes mest i vanlig matriseregning. Det finnes imidlertid andre operasjoner slik som det komponentvise matriseproduktet (eller *Hadamard-produktet*) der produktet av matrisene

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \cdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \quad \text{og} \quad \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \cdots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{pmatrix}$$

er

$$\begin{pmatrix} a_{11}b_{11} & a_{12}b_{12} & \ldots & a_{1n}b_{1n} \\ a_{21}b_{21} & a_{22}b_{22} & \ldots & a_{2n}b_{2n} \\ \vdots & \vdots & \ldots & \vdots \\ a_{m1}b_{m1} & a_{m2}b_{m2} & \ldots & a_{mn}b_{mn} \end{pmatrix}$$

Selv om slike komponentvise operasjoner ikke brukes mye i lineær algebra, er de viktige i MATLAB siden MATLAB bruker matriser til mer enn tradisjonell matriseregning. MATLAB har derfor egne kommandoer for slike operasjoner. Disse kommandoene har et punktum foran det "naturlige" kommandonavnet. Skriver du f.eks.

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

vil MATLAB regne ut det komponentvise produktet av A og B som beskrevet ovenfor. Tilsvarende gir

```
>> A./B
```

en matrise der den ij-te komponenten er $\frac{a_{ij}}{b_{ij}}$ mens

```
>> A.^B
```

gir en matrise der den ij-te komponenten er $a_{ij}^{b_{ij}}$. Vi kan også bruke disse operasjonene når den ene matrisen erstattes med et tall; f.eks. vil

```
>> 3./B
```

produsere matrisen med komponenter $\frac{3}{b_{ij}}$, mens

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

produserer matrisen med komponenter ${a_{ij}}^2$. Du kan til og med anvende funksjoner på matriser;

```
>> sin(A)
```

gir en matrise med komponenter $\sin a_{ij}$. Operasjonene kan selvfølgelig kombineres; kommandoen

```
>>exp(A.*(B.^2))
```

gir oss matrisen med komponenter $e^{a_{ij}b_{ij}^2}$. Som vi skal se i neste seksjon, er de komponentvise operasjonene spesielt nyttige når vi skal tegne grafer, men det finnes også noen enklere anvendelser som vi kan se på allerede nå.

MATLAB har egne kommandoer for å finne den største og minste komponenten til en vektor c. Du skriver bare

```
>> max(c)
>> min(c)
```

På tilsvarende måte kan du finne summer og produkter. Har du lagt inn en vektor c, finner du summen og produktet til komponentene i c ved å skrive

```
>> sum(c)
>> prod(c)
```

Skal vi finne summen av de 10 første naturlige tallene, kan vi altså skrive

```
>> a=[1 2 3 4 5 6 7 8 9 10];
>> sum(a)
```

Hvis vi også vil ha summen av de 10 første kvadrattallene, kan vi bruke en av de "punkterte" operasjonene:

```
>> sum(a.^2)
```

Dette er vel og bra så lenge vi har korte summer, men hva hvis vi ønsker summen av de hundre første kvadrattallene? Det blir ganske kjedelig å taste inn vektoren $(1, 2, 3, \ldots, 100)$ for hånd. MATLAB har løsninger for dette. Taster du

```
>> a=1:100
```

definerer du a til å være vektoren $(1, 2, 3, \ldots, 100)$. Skriver du isteden

```
>> a=1:2:100
```

blir a vektoren bestående av alle oddetallene mindre enn 100. Generelt vil

```
>> a=b:h:c
```

definere a til å være vektoren med b som førstekomponent, $b + h$ som annenkomponent, $b + 2h$ som tredjekomponent osv. inntil vi kommer til c (den siste komponenten vil altså være det største tallet på formen $a + nh$ som er mindre enn eller lik c). Skal vi regne ut summen av kvadratene av alle partall opptil 100, kan vi altså skrive

```
>> a=2:2:100;
>> sum(a.^2)
```

Til slutt nevner vi kommandoen `>> linspace` som ofte er nyttig hvis intervallet vi skal dele opp har "uregelmessige" endepunkter. Skriver vi f.eks.

```
>> a=linspace(0,pi,50);
```

får vi definert `a` som en vektor med 50 komponenter der første komponent er 0, siste komponent er π og avstanden mellom én komponent og den neste allltid er den samme.

0ppgaver til seksjon 3

1. Legg inn disse n-tuplene i MATLAB: $(1, -9, 7, 5, -7)$, $(\pi, -14, e, 7/3)$, $(1, 3, 5, 7, 9, \ldots, 99)$, $(124, 120, 116, 112 \ldots, 4, 0)$.

2. Legg inn tuppelet $(1, 2, 4, 8, 16, \ldots 4096)$ i MATLAB og finn summen.

3. Legg inn $(0, \frac{1}{100}, \frac{2}{100}, \ldots, 1)$ i MATLAB. Bruk denne partisjonen til å lage en nedre og øvre trappesum for funksjonen $f(x) = x^2$ over intervallet $[0, 1]$, og regn ut disse summene. Sammenlign med integralet $\int_0^1 x^2\, dx$.

4. Bruk MATLAB til å regne ut produktet $\frac{1}{2} \cdot \frac{3}{4} \cdots \frac{99}{100}$.

5. Anta at du har lastet inn en matrise A i MATLAB. Da vil kommandoen `>> sum(A)` gi deg en radvektor der hver komponent er summen av den tilsvarende søylen, mens kommandoen `>> sum(A,2)` gir deg en søylevektor der hver komponent er summen av tilsvarende rad. Prøv disse kommandoene på noen matriser.

A4: Grafer

Anta at $x = (x_1, x_2, \ldots, x_n)$ og $y = (y_1, y_2, \ldots, y_n)$ er to n-tupler. Kommandoen

```
>> plot(x,y)
```

får MATLAB til å lage en strektegning der punktet (x_1, y_1) er forbundet til (x_2, y_2) med en strek, (x_2, y_2) er forbundet til (x_3, y_3) med en strek osv. Dersom du bare skriver

```
>> plot(x)
```

(altså én vektor istedenfor to), lager MATLAB en strektegning mellom punktene $(1, x_1)$, $(2, x_2)$, $(3, x_3)$ osv. Vær oppmerksom på at figuren kommer i et eget vindu med navn "Figure 1", og at dette vinduet ofte gjemmer seg bak andre vinduer.

Vi kan utnytte metoden ovenfor til å plotte grafer av funksjoner. Trikset er å velge punktene som skal forbindes til å være (tettliggende) punkter på funksjonsgrafen. La oss se hvordan vi kan bruke metoden til å tegne grafen til $\sin \frac{1}{x}$ over intervallet $[-\pi, \pi]$. Vi velger først et tuppel som gir oppdeling av intervallet i mange små biter:

```
>> x=linspace(-pi,pi,100);
```

(det er lurt å avslutte med et semikolon så ikke MATLAB lister opp alle punktene i intervallet som output). Så velger vi tuppelet av tilhørende funksjonsverdier:

```
>> y=sin(1./x);
```

(Legg merke til punktumet – siden vi egentlig opererer på vektorer, må vi bruke de "punkterte" operasjonene `.*`, `./` og `.^` når vi lager grafer!) Til slutt plotter vi punktene

```
>> plot(x,y)
```

Men hva hvis vi har lyst til å tegne en graf til i samme vindu? Da gir vi først kommandoen

```
>> hold on
```

og taster så inn en ny funksjon. Hvis den nye funksjonen er $z = x^2 \sin \frac{1}{x}$, skriver vi først

```
>> z=x.^2.*sin(1./x);
```

og så plotter vi punktene:

```
>> plot(x,z)
```

Grafene du lager vil fortsette å komme i samme vindu inntil du gir kommandoen

```
>> hold off
```

Det hender at du vil lage en ny figur i et nytt vindu og samtidig beholde den gamle i det gamle vinduet. Da kan du bruke kommandoen

```
>> figure
```

som får MATLAB til å lage et nytt vindu med navn "Figure 2". De nye plottene du lager, kommer nå i dette vinduet. Vil du ha enda et nytt vindu, bruker du kommandoen `figure` på ny og får et vindu "Figure 3". Ønsker du senere å gå tilbake til vinduet "Figure 2" for

å gjøre endringer, aktiverer du dette vinduet med kommandoen

```
>> figure(2)
```

Vi skal se på noen tilleggskommandoer som kan være nyttige å kunne. Dersom du selv vil velge størrelsen på vinduet grafene skal vises i, kan du bruke en kommando av typen

```
>> axis([-pi, pi, -2.5, 2.5])
```

Vil du ha et kvadratisk vindu, skriver du

```
>> axis('square')
```

mens

```
>> axis('equal')
```

gir deg samme målestokk på begge aksene. Du kan gi figuren en tittel ved å skrive noe slikt som

```
>> title('Grafen til en vakker funksjon')
```

og du kan sette navn på aksene ved å skrive

```
>> xlabel('x-akse')
```

```
>> ylabel('y-akse')
```

Ønsker du tekst på figuren, kan du skrive

```
>> text(a,b,'tekst')
```

der (a, b) er koordinatene til det punktet på figuren der teksten skal begynne. Du kan få farge på grafene ved å bruke fargekoder;

```
>> plot(x,y,'r')
```

gir en rød graf, mens

```
>> plot(x,y,'g')
```

gir en grønn. En rød, stiplet graf får du ved

```
>> plot(x,y,'r--')
```

og en grønn, prikket graf ved

```
>> plot(x,y,'g:')
```

Av og til ønsker man å ha flere grafer side om side i den samme figuren. Kommandoen

```
>> subplot(m,n,p)
```

deler vinduet i $m \times n$-delvinduer, og sørger for at den neste plot-kommandoen blir utført i det p-te delvinduet.

Hvis du ønsker å ta vare på en figur for å kunne bruke den i et annet dokument senere, må du lagre den som en egen fil. Det enkleste er å bruke menyene i figurvinduet. Vær klar over at du kan velge mellom forskjellige formater, f.eks. EPS eller JPEG.

Oppgaver til seksjon 4

1. Legg inn 6-tuplene $a = (3, 1, -2, 5, 4, 3)$ og $b = (4, 1, -1, 5, 3, 1)$ i MATLAB og utfør kommandoen `>> plot(a,b)`. Utfør også kommandoene `>> plot(a)` og `>> plot(b)`, og bruk `>> hold on` til å sørge for at de to siste figurene kommer i samme vindu.

2. Bruk kommandoen `>> plot` til å lage en enkel strektegning av et hus.

3. Bruk MATLAB til å tegne grafen til $f(x) = x^3 - 1$ over intervallet $[-1, 1]$. Legg så inn grafen til $g(x) = 3x^2$ i samme koordinatsystem, og velg forskjellig farge på de to grafene.

4. Bruk MATLAB til å tegne grafen til funksjonen $f(x) = \sin\frac{1}{x}$ over intervallet $[-1, 1]$. Bruk først skrittlengde $\frac{1}{100}$ langs x-aksen. Tegn grafen på nytt med skrittlengde $\frac{1}{10000}$.

A5: Tredimensjonale grafer

MATLAB har flere kommandoer som kan brukes til å lage tredimensjonale figurer. Vi skal se på et par av de enkleste. Grafen til en funksjon $z = f(x, y)$ tegnes ved å plotte funksjonsverdiene over et nett av gitterpunkter i xy-planet.

Kommandoen `mesh` lager konturer av grafen ved å forbinde plottepunktene på grafen med rette linjestykker. Resultatet ser ut som et fiskegarn med knuter i plottepunktene. Varianten `meshc` tegner i tillegg nivåkurver til funksjonen i xy-planet. La oss se hvordan vi kan tegne grafen til funksjonen $z = f(x, y) = xy\sin(xy)$ over området $-4 \leq x \leq 4, -2 \leq y \leq 2$.

Vi lager først en oppdeling av de to intervallene vi er interessert i, ved å skrive

```
>> r=-4:0.05:4;
>> s=-2:0.05:2;
```

(husk semikolon etter kommandoene, ellers vil du få lange tallremser som output!) Her har vi valgt å dele opp begge intervallene i skritt med lengde 0.05, men du kan godt velge en finere eller grovere oppdeling. Neste skritt er å lage et rutenett av oppdelingene våre. Dette gjør vi med kommandoen

```
>> [x,y]=meshgrid(r,s);
```

Vi kan nå definere funksjonen:

```
>> z=x.*y.*sin(x.*y);
```

(husk å bruke .-versjonene av de algebraiske operasjonene!) Dermed er vi klare til selve plottingen som utføres av kommandoen

```
>> mesh(x,y,z)
```

Grafen kommer opp i et eget figurvindu akkurat som for todimensjonale figurer. Velger du `>> surf(x,y,z)` istedenfor `>> mesh(x,y,z)`, får du en graf der flateelementene er fargelagt. Ved å gå inn på menyen i grafvinduet, kan du dreie flaten i rommet (aktiver musa ved å klikke på et ikon som symboliserer dreiing). Dette er ofte nødvendig for å få et godt inntrykk av hvordan grafen ser ut. Ønsker du bare å få ut nivåkurvene til en flate, kan du bruke kommandoen

```
>> contour(x,y,z).
```

Du kan regulere antall nivåkurver ved å skrive

```
>> contour(x,y,z,n)
```

der n er antall nivåkurver du ønsker.

Kommandoen

```
>> plot3
```

er nyttig når du skal plotte parametriserte kurver i tre dimensjoner. Skriver du

```
>> t=linspace(0,10*pi,100);
>> x=sin(t);
>> y=cos(t);
>> z=t;
>> plot3(x,y,z)
```

tegner MATLAB kurven $\mathbf{r}(t) = (\sin t, \cos t, t)$ for $t \in [0, 10\pi]$.

Oppgaver til seksjon 5

1. Tegn grafene til disse funksjonene med MATLAB: $f(x,y) = x^2y^2$, $g(x,y) = \frac{\sin x}{y^2} + x^2$, $h(x,y) = \sin(e^{x+y})$. Vri på flatene for å få et best mulig inntrykk.

2. Bruk MATLAB til å tegne kurvene $\mathbf{r}_1(t) = (t, t^2, \sin t)$ og $\mathbf{r}_2(t) = (\sin^2 t, \cos^2 t, e^{-t})$. Vri på koordinatsystemet for å se kurvene best mulig.

3. Bruk kommandoen `plot3` til å lage en tredimensjonal strektegning av en terning.

A6: Mer om matriser

Siden matriser står så sentralt i MATLAB, kan det være greit å vite hvordan man kan manipulere dem på en effektiv måte. MATLAB gir deg blant annet mange muligheter til å sette sammen og ta fra hverandre matriser.

Dersom A er 3×3-matrisen

$$A = \begin{pmatrix} 2 & -3 & 1 \\ 0 & 1 & -3 \\ -4 & 2 & 1 \end{pmatrix}$$

og B er 3×2-matrisen

$$B = \begin{pmatrix} 7 & 4 \\ 2 & 5 \\ -1 & 3 \end{pmatrix}$$

kan vi skjøte sammen A og B til 3×5-matrisen

$$C = \begin{pmatrix} 2 & -3 & 1 & 7 & 4 \\ 0 & 1 & -3 & 2 & 5 \\ -4 & 2 & 1 & -1 & 3 \end{pmatrix}$$

ved å gi kommandoen

```
>> C=[A B]
```

Du kan skrive ut komponentene til en matrise med enkle kommandoer:

```
>> A(1,2)
```

skriver ut elementet i første rad, andre søyle i A (legg merke til at vi nå bruker runde parenteser, og ikke de firkantede vi bruker til å bygge matriser). Vil du skrive ut hele den første raden, gir du kommandoen

```
>> A(1,:)
```

mens

```
>> A(:,2)
```

skriver ut den andre søylen. Du kan også bruke kolon-notasjon til å plukke ut andre deler av en matrise. Starter vi med 3×5-matrisen C ovenfor, vil

```
>> C(2:3,1:4)
```

gi deg undermatrisen

$$\begin{pmatrix} 0 & 1 & -3 & 2 \\ -4 & 2 & 1 & -1 \end{pmatrix}$$

som består av komponentene som ligger fra annen til tredje rad (det er dette `2:3` står for) og fra første til fjerde søyle (det er dette `1:4` står for). Kommandoen

```
>> C([1 3],[2 5])
```

gir deg matrisen

$$\begin{pmatrix} -3 & 4 \\ 2 & 3 \end{pmatrix}$$

bestående av elementene som ligger i første og tredje rad og annen og femte søyle. Du kan også bruke denne notasjonen til å bytte om på radene eller søylene til en matrise. Skriver du

```
>> C([3 1 2],:)
```

får du ut matrisen

$$\begin{pmatrix} -4 & 2 & 1 & -1 & 3 \\ 2 & -3 & 1 & 7 & 4 \\ 0 & 1 & -3 & 2 & 5 \end{pmatrix}$$

der radene i den opprinnelige matrisen C nå kommer i rekkefølgen $3, 1, 2$. Kommandoen `>> C(:,[3 1 2 5 4])` vil på tilsvarende måte bytte om på søylene i C.

Som tidligere nevnt, kan du *transponere* matrisen C (dvs. bytte om rader og søyler) ved å skrive

```
>> C'
```

Resultatet er

$$C^T = \begin{pmatrix} 2 & 0 & -4 \\ -3 & 1 & 2 \\ 1 & -3 & 1 \\ 7 & 2 & -1 \\ 4 & 5 & 3 \end{pmatrix}$$

Legg merke til at hvis

$$a = (a_1, a_2, \ldots, a_n)$$

er en radvektor, vil den transponerte være søylevektoren

$$a^T = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix}$$

Det finnes mange andre spesialkommandoer for å manipulere matriser i MATLAB. De er samlet i verktøykassen ELMAT som du får oversikt over ved å skrive

```
>> help elmat
```

Her følger noen kommandoer man ofte kan ha nytte av. Noen av de første har vi mer eller mindre vært borti i tidligere, men vi gjentar dem her for å ha alt samlet på et sted:

`x=linspace(2,4,12)` % 12-tuppelet med 12 tall jevnt fordelt fra 2 til 4

`x=1:10` % 10-tuppelet med alle heltallene fra 1 til 10

`x=1:2:10` % 5-tuppelet med annethvert tall fra 1 til 10 (2 angir steglengden)

`A=zeros(3,4)` % 3×4-matrisen med bare nuller

`A=ones(3,4)` % 3×4-matrisen med bare enere

`A=eye(3,4)` % 3×4-matrisen med enere på hoveddiagonalen, nuller ellers

`A=rand(3,4)` % 3×4-matrise med tilfeldiggenererte tall mellom 0 og 1

Som et enkelt eksempel tar vi med at `>> A=eye(3,4)` gir matrisen

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Oppgaver til seksjon 6

1. Skriv inn matrisene

$$A = \begin{pmatrix} 2 & -3 & 1 \\ 4 & 1 & -5 \\ 1 & 2 & -1 \end{pmatrix}$$

og

$$B = \begin{pmatrix} -1 & 3 & 2 \\ 4 & 5 & 1 \\ 0 & 2 & -1 \end{pmatrix}$$

i MATLAB og gjennomfør operasjonene `>> C=[A,B]`, `>> C(2,4)`, `>> C(:,[2 3])`, `>> C([1 3],3:5)`.

2. Bruk matrisen C fra forrige oppgave. Undersøk hva som skjer med matrisen når du skriver `>> C(:,3)=2*C(:,3)` og `>> C([1,3],:)=4*C([1,3],:)`.

3. Undersøk hva kommandoen `>> [A;B]` gjør når A og B er to matriser.

A7: Radoperasjoner i MATLAB

Denne seksjonen forutsetter at du kjenner til radoperasjoner for matriser, og at du kan bringe en matrise til (redusert) trappeform. Har du ikke lært dette ennå (det kommer i kapittel 4), kan du trygt hoppe over denne seksjonen så lenge.

MATLAB har en egen kommando som bringer en matrise på redusert trappeform. Den heter `rref` (for *reduced row echelon form*). Starter vi med matrisen

$$A = \begin{pmatrix} 1 & 4 & -5 & 7 & 3 & 2 & 8 & -1 \\ 2 & 3 & -2 & 5 & 3 & 3 & 7 & 8 \\ -1 & 1 & 2 & 3 & -1 & 4 & 4 & 4 \end{pmatrix}$$

leder kommandoen

```
>> B=rref(A)
```

til den reduserte trappeformen

$$B = \begin{pmatrix} 1 & 0 & 0 & -0.48 & 0.88 & -0.2 & -0.04 & 3.36 \\ 0 & 1 & 0 & 2.12 & 0.28 & 1.8 & 2.76 & 2.16 \\ 0 & 0 & 1 & 0.2 & -0.2 & 1 & 0.6 & 2.6 \end{pmatrix}$$

Du kan også bruke MATLAB til å foreta nøyaktig de radoperasjonene du vil. Lar du

$$C = \begin{pmatrix} 2 & -3 & 1 & 7 & 4 \\ 0 & 1 & -3 & 2 & 5 \\ -4 & 2 & 1 & -1 & 3 \end{pmatrix}$$

være matrisen fra forrige seksjon, har vi allerede sett hvordan vi kan bytte om på to rader ved å bruke en kommando av typen `>> C([3,2,1],:)`. Denne kommandoen gir deg en ny matrise der rad 1 og 3 er ombyttet. Taster du

```
>> C(1,:)=2*C(1,:)
```

svarer MATLAB med

```
C=
4    -6    2    14    8
0    1    -3    2    5
-4    2    1    -1    3
```

Den har altså ganget den første raden med 2. Skriver du så

```
>> C(3,:)=C(3,:)+C(1,;)
```

legger MATLAB den første raden til den siste, og du får

```
C=
4    -6    2    14    8
0    1    -3    2    5
0    -4    3    13    11
```

Du kunne ha slått sammen disse operasjonene til én ved å skrive

```
>> C(3,:)=C(3,:)+2*C(1,:)
```

(dersom du prøver den siste kommandoen, må du huske å tilbakeføre C til den opprinnelige verdien først!)

Man kan lure på hva som er vitsen med å kunne foreta radoperasjoner "for hånd" på denne måten når MATLAB har kommandoen `>> rref` innebygget. Når man bruker MATLAB som et verktøy, støter man imidlertid ofte på matriser med spesiell struktur, f.eks. svært mange nuller. Da er det ofte mer effektivt selv å programmere hvilke radoperasjoner MATLAB skal gjøre enn å kjøre en standardkommando som `>> rref` som ikke tar hensyn til strukturen til matrisene.

Radoperasjoner i MATLAB kan brukes til å løse lineære ligningssystemer, men du har også muligheten til å finne løsningen med én kommando. Dersom A er en ikke-singulær, kvadratisk matrise og b er en vektor, kan du løse vektorligningen $Ax = b$ ved kommandoen

```
>> x=A\b
```

(legg merke til at skråstreken heller bakover `\` og ikke fremover /). Velger vi for eksempel

$$A = \begin{pmatrix} 1 & -1 & 4 & 3 \\ 2 & 1 & -4 & 5 \\ 6 & 3 & 1 & -2 \\ 3 & 3 & -2 & 4 \end{pmatrix}$$

og

$$b = \begin{pmatrix} 1 \\ 3 \\ 0 \\ -2 \end{pmatrix}$$

gir kommandoen `>> x=A\b` svaret

$$x = \begin{pmatrix} 1.3537 \\ -2.4784 \\ -0.7023 \\ -0.0076 \end{pmatrix}$$

Oppgaver til seksjon 7

1. Skriv inn matrisene

$$A = \begin{pmatrix} 2 & -3 & 1 & 1 & 4 \\ 4 & 1 & -5 & 2 & -1 \\ 1 & 2 & -1 & 2 & -1 \end{pmatrix}$$

og

$$B = \begin{pmatrix} -1 & 3 & 2 & 3 & 1 \\ 4 & 5 & 1 & 4 & 4 \\ 0 & 2 & -1 & -3 & -1 \end{pmatrix}$$

i MATLAB og gjennomfør operasjonene `>> rref(A)` og `>> rref(B)`.

2. Bruk MATLAB til å løse ligningssystemet

$$\begin{aligned} x + 3y + 4z + 6u &= 3 \\ -x + y + 3z - 5u &= 5 \\ 2x - 3y + z + 6u &= -2 \\ 2x + 3y - 2z + u &= 0 \end{aligned}$$

3. Finn den generelle løsningen av ligningssystemet

$$\begin{aligned} 2x - 3y + 4z + 5u &= 6 \\ x + y + 3u &= 4 \\ 4x - 3y + 2z + 3u &= 5 \end{aligned}$$

ved hjelp av MATLAB-kommandoen `rref`.

4. Skriv matrisen

$$A = \begin{pmatrix} 0 & 3 & -2 & 7 & 1 & 2 & 7 & 0 \\ 2 & 0 & 1 & -4 & 3 & 2 & 0 & 4 \\ -1 & 3 & 4 & -1 & 2 & -5 & 6 & 2 \end{pmatrix}$$

på redusert trappeform ved å la MATLAB utføre radoperasjonene én for én (du får altså ikke lov til å bruke `rref` eller en lignende kommando). Beskriv den generelle løsningen til ligningssystemet som har A som utvidet matrise.

5. a) Figuren viser spillebrettet for et enkelt terningspill. Spillerne starter på "Start" og kaster en vanlig terning for å flytte. De trenger eksakt riktig antall øyne for å gå i mål (står de på 11 og kaster en femmer, 'spretter" de altså ut til 10 ved å telle på denne måten 12-mål-12-11-10). La t_i være antall kast du må regne med å gjøre før du går i mål (dvs. det forventede antall kast) dersom du står på felt i.

Mål	12	11	10	9	8	7
Start	1	2	3	4	5	6

Forklar hvorfor

$$\begin{aligned}
t_1 &= \frac{1}{6}t_2 + \frac{1}{6}t_3 + \frac{1}{6}t_4 + \frac{1}{6}t_5 + \frac{1}{6}t_6 + \frac{1}{6}t_7 + 1 \\
t_2 &= \frac{1}{6}t_3 + \frac{1}{6}t_4 + \frac{1}{6}t_5 + \frac{1}{6}t_6 + \frac{1}{6}t_7 + \frac{1}{6}t_8 + 1 \\
\vdots \;\; &\vdots \;\; \vdots \qquad \vdots \qquad \vdots \qquad \vdots \qquad \vdots \\
t_6 &= \frac{1}{6}t_7 + \frac{1}{6}t_8 + \frac{1}{6}t_9 + \frac{1}{6}t_{10} + \frac{1}{6}t_{11} + \frac{1}{6}t_{12} + 1 \\
t_7 &= \frac{1}{6}t_8 + \frac{1}{6}t_9 + \frac{1}{6}t_{10} + \frac{1}{6}t_{11} + \frac{1}{6}t_{12} + 1 \\
t_8 &= \frac{1}{6}t_9 + \frac{1}{6}t_{10} + \frac{1}{6}t_{11} + \frac{1}{3}t_{12} + 1 \\
t_9 &= \frac{1}{6}t_{10} + \frac{1}{3}t_{11} + \frac{1}{3}t_{12} + 1 \\
t_{10} &= \frac{1}{6}t_{10} + \frac{1}{3}t_{11} + \frac{1}{3}t_{12} + 1 \\
t_{11} &= \frac{1}{6}t_9 + \frac{1}{6}t_{10} + \frac{1}{6}t_{11} + \frac{1}{3}t_{12} + 1 \\
t_{12} &= \frac{1}{6}t_8 + \frac{1}{6}t_9 + \frac{1}{6}t_{10} + \frac{1}{6}t_{11} + \frac{1}{6}t_{12} + 1
\end{aligned}$$

Forklar videre hvorfor dette er et lineært ligningssystem og bruk kommandoen `rref` til å finne løsningen. Hvor mange kast må du regne med å bruke når du står på start?

b) Løs problemet i a) når du ikke trenger eksakt antall øyne for å komme i mål.

A8: Ta vare på arbeidet ditt

Vi skal nå se på noen mer datatekniske ting om hvordan du kan ta vare på det du har gjort i MATLAB. Hvis du vil ta vare på MATLAB-kjøringene dine med tanke på senere

redigering, kan du skrive

```
>> diary filnavn
```

Innholdet i kommandovinduet blir nå fortløpende skrevet til filen du oppga som blir opprettet i katalogen du står i. For å avslutte dagbokføringen, skriver du

```
>> diary off
```

Hvis du senere ønsker å skrive til den samme dagbokfilen, gir du kommandoen

```
>> diary on
```

Dagbokfiler kan redigeres på vanlig måte, og egner seg derfor bra til obliger og lignende. Du kan gjøre de MATLAB-kjøringene du ønsker, lagre dem i dagboken, og etterpå legge til kommentarer, stryke uaktuelle utregninger osv. Vær oppmerksom på at figurer ikke lagres i dagboken!

Noen ganger ønsker vi ikke å ta vare på så mye av det vi har gjort, alt vi trenger er å beholde verdien på visse variable til senere kjøringer. Skriver du

```
save var x y A
```

lagres verdiene til `x`, `y` og `A` på filen `var.mat`. Neste gang du kjører MATLAB, kan du lese inn variablene fra denne filen ved å skrive

```
>> load var
```

Ønsker du å lagre *alle* variabelverdier, kan du skrive

```
>> save
```

Når du skal lagre variable, er det ofte lurt å gi dem mer beskrivende navn enn `a, y, A`. Et variabelnavn i MATLAB kan bestå av bokstaver og tall, men må begynne med en bokstav. MATLAB skiller mellom store og små bokstaver. Du bør unngå å bruke innebygde funksjons- og kommandonavn på variablene dine.

Når du har arbeidet en stund, kan du fort miste oversikten over hvilke variabelnavn du har brukt. Kommandoen

```
>> who
```

gir deg en oversikt. Hvis du vil slette innholdet i variabelen `x`, skriver du

```
>> clear x
```

Vil du slette alle variablene, skriver du

```
>> clear
```

Oppgave til seksjon 8

1. Lag en diary-fil og rediger den.

A9: Programmering i MATLAB

Skal du få virkelig nytte av MATLAB, må du lære deg å programmere. Har du programmert i et annet språk tidligere, bør ikke dette være særlig problematisk – alt du behøver, er å lære deg litt MATLAB-syntaks. Har du aldri programmert før, står du overfor en litt større utfordring, men vi skal prøve å hjelpe deg igang.

Et program er egentlig ikke noe annet enn en sekvens av kommandoer som du vil at MATLAB skal utføre, men det er to ting som gjør at virkelige programmer er litt mer kompliserte enn de kommandosekvensene vi hittil har sett på. Den ene er at virkelige programmer gjerne inneholder *løkker*, dvs. sekvenser av kommandoer som vi vil at maskinen skal gjennomføre mange ganger. Vi skal se på to typer løkker: *for-løkker* og *while-løkker*. Den andre tingen som skiller virkelige programmer fra enkle kommandosekvenser, er at hvilken utregning som skal utføres på et visst punkt i programmet, kan avhenge av resultatet av tidligere beregninger som programmet har gjort. Dette fanger vi opp med såkalte *if-else-setninger*.

La oss se på et typisk problem som kan løses ved hjelp av løkker. Vi skal ta for oss *Fibonacci-tallene*, dvs. den tallfølgen $\{F_n\}$ som begynner med $F_1 = 1$, $F_2 = 1$ og tilfredsstiller

$$F_n = F_{n-1} + F_{n-2} \qquad \text{for } n \geq 3 \tag{1.0.1}$$

Ethvert tall i følgen (bortsett fra de to første) er altså summen av de to foregående. Det er lett å bruke formelen til å regne ut så mange Fibonacci-tall vi vil:

$$F_3 = F_2 + F_1 = 1 + 1 = 2$$
$$F_4 = F_3 + F_2 = 2 + 1 = 3$$
$$F_5 = F_4 + F_3 = 3 + 2 = 5$$
$$F_6 = F_5 + F_4 = 5 + 3 = 8$$

og så videre. Vi ser at vi hele tiden utfører regnestykket i formel (A.0.1), men at vi for hvert nytt regnestykke oppdaterer n-verdien.

La oss nå lage et lite MATLAB-program som regner ut de 20 første Fibonacci-tallene. Programmet gjør dette ved å lage en 20-dimensjonal vektor der F_n er den n-te komponenten. (Kommentarene etter %-tegnet forklarer hensikten med linjen — de er ikke del av programmet.)

```
>> F=[1 1];              %forteller MATLAB at F_1 = 1 og F_2 = 1
>> for n=3:20;           %starter for-løkken og angir hvor
                          %langt den skal løpe
      F(n)=F(n-1)+F(n-2); %regner ut neste Fibonacci-tall
   end                    %avslutter for-løkken
```

Det burde ikke være så vanskelig å skjønne hvordan programmet fungerer — det utfører de samme beregningene som vi gjorde ovenfor fra $n = 3$ til og med $n = 20$ (dvs. mellom grensene angitt i for-løkken). Legg forøvrig merke til at vi skrev semikolon etter linjene i programmet; det er ofte lurt for å unngå mye rar output. Vil du nå vite hva det 17. tallet i

følgen er, kan du nå skrive

```
>> F(17)
```

I for-løkker bestemmer vi på forhånd hvor mange ganger løkken skal gjennomløpes. Ofte ønsker vi å fortsette beregningene til et visst resultat er oppnådd uten at vi på forhånd vet hvor mange gjennomløpninger som trengs. I disse tilfellene er det lurt å bruke en while-løkke. Det neste programmet illustrerer dette ved å regne ut Fibonacci-tallene inntil de når 10 000.

```
>> F=[1 1];
>> n=3;
>> while F(n-1)<10000
      F(n)=F(n-1)+F(n-2);
      n=n+1;
   end
```

Legg merke til at i en while-løkke må vi selv oppdatere fra n til $n+1$, mens denne oppdateringen skjer automatisk i en for-løkke. Dette skyldes at while-løkker er mer fleksible enn for-løkker, og at man også ønsker å tillate andre typer oppdatering enn at n går til $n+1$.

La oss også se et enkelt eksempel på bruk av en if-else-setning. Det litt tossete progammet printer ut de like Fibonacci-tallene samt indeksen til alle odde Fibonacci-tall. Legg merke til funksjonen `rem(m,k)` som gir oss resten når m deles på k (dersom k er lik 2 som i programmet, blir resten 0 når m er et partall, og 1 når m er et oddetall).

```
>> F=[1 1];
   >> for n=3:20            %starter for-løkke
      F(n)=F(n-1)+F(n-2);   %regner ut neste Fibonacci-tall
         if rem(F(n),2)==0  %innleder if-else setningen ved å
                            %sjekke om F(n) er delelig med 2
            F(n)            %skriver ut F(n) hvis det er et partall
         else               %innleder else-delen av setningen
            n               %skriver ut n dersom F(n) er et oddetall
         end                %avslutter if-else setningen
      end                   %avslutter for-løkken
```

Legg merke til det dobbelte likhetstegnet i fjerde linje (`if rem(F(n),2)==0`). I MATLAB brukes det vanlige likhetstegnet til å tilordne verdier — skriver du

```
>> C=D,
```

får C den (forhåpentligvis allerede definerte) verdien til D. For å sjekke om to allerede definerte verdier er like, må du derfor bruke et annet symbol, nemlig det dobbelte likhetstegnet `==`. Nyttige symboler av denne typen, er

`<` mindre enn

`<=` mindre enn eller lik

`>`	større enn
`>=`	større enn eller lik
`==`	lik
`~=`	ikke lik

Når du skriver programmer, får du også bruk for logiske operatorer som *og, eller* og *ikke*. I MATLAB skriver du dem slik:

`&`	og
`|`	eller
`~`	ikke

Som et eksempel tar vi med et lite program som skriver ut Fibonacci-tall som er partall, men ikke delelige på 4:

```
>> F=[1 1];
>> for n=3:50
     F(n)=F(n-1)+F(n-2);
        if (rem(F(n),2)==0) & (rem(F(n),4)~=0)
           F(n)
        end
   end
```

I litt større program har vi ofte behov for å avslutte en løkke og fortsette eksekveringen av den delen av programmet som kommer umiddelbart etter løkken. Til dette bruker vi kommandoen `break`. Det neste programmet illustrerer bruken. Det skriver ut alle Fibonacci-tall mindre enn 1000 (siden størrelsen på Fibonacci-tallene når 1000 før indeksen kommer til 50):

```
>> F=[1 1];
>> for n=3:50
     F(n)=F(n-1)+F(n-2);
        if F(n)>1000
           break
        end
     F(n)
    end
```

La oss ta en nærmere titt på den generelle syntaksen for for-løkker, while-løkker, if-setninger og break-kommandoer (ikke bry deg om dette hvis du synes det ser rart og abstrakt ut på det nåværende tidspunkt — det kan likevel hende du får nytte av det når du har fått litt mer programmeringstrening).

Syntaksen til en for-løkke er:

```
>> for n=start:steg:stopp
      setninger
   end
```

Hvis steglengden er 1, trenger vi bare å oppgi start- og stoppverdiene.

Syntaksen til en while-løkke er:

```
 >> while feil < toleranse
      setninger
    end
```

Syntaksen for if-setninger er:

```
>> if n < nmaks
      setninger
   end
```

eller

```
>> if n < middel
      setninger
   elseif n > middel
      setninger
   else
      setninger
   end
```

Syntaksen for break-kommandoer er:

```
>> for n=1:antall
      setninger
      if feil > toleranse
         break
      end
      setninger
   end
```

Vi avslutter denne seksjonen med noen ord om effektivitet. I programmene ovenfor har vi begynt med å la F være 2-tuppelet `[1 1]` og så latt MATLAB utvide lengden på tuppelet etter hvert som programmet kjører. Det viser seg at MATLAB går fortere dersom vi gir n-tuppelet den "riktige" størrelsen fra starten av. Det første programmet vårt blir altså raskere om vi skriver det om på denne måten:

```
>> F=zeros(1,20);
>> F(1)=1;
>> F(2)=1;
>> for n=3:20;
      F(n)=F(n-1)+F(n-2);
   end
```

Inntjeningen spiller selvfølgelig ingen rolle når programmet er så kort som her, men for store utregninger kan den ha betydning.

La oss helt til slutt nevne at det egentlig er enda mer effektivt å unngå for-løkker der det er mulig; ofte kan vi erstatte dem ved å bruke vektorer og komponentvise operasjoner isteden. Ønsker vi for eksempel å lage en vektor med de 5 første positive oddetallene som elementer, vil følgende for-løkke gjøre jobben:

```
>> for i=1:5
     a(i)=2*i-1;
   end
```

Men det er mer effektivt å lage vektoren på følgende måte:

```
>> a=2*(1:5)-1;
```

Vi skal ikke legge vekt på effektiv programmering i dette kurset, men med tanke på senere kurs er det greit å være oppmerksom på at slik bruk av "vektor-indeksering" kan gjøre store programmer mye raskere.

Oppgaver til seksjon 9

1. En følge er gitt ved $a_1 = 1$, $a_2 = 3$ og $a_{n+2} = 3a_{n+1} - 2a_n$. Skriv et program som genererer de 30 første leddene i følgen.

2. Skriv et program som skriver ut hvert fjerde Fibonacci-tall mindre enn 10 000.

3. Skriv et program som regner ut tallene $G_n = \frac{F_n}{F_{n-1}}$, $n = 2, \cdots, 30$, der F_n er det n-te Fibonacci-tallet. Tegn en graf som viser følgen.

4. I denne oppgaven skal vi se på en modell for samspillet mellom rovdyr og byttedyr. Vi lar x_n og y_n betegne hhv. antall rovdyr og antall byttedyr etter n uker, og vi antar at

$$x_{n+1} = x_n(r + cy_n) \qquad\qquad y_{n+1} = y_n(q - dx_n)$$

der r er litt mindre enn 1, q er litt større enn 1, og c og d er to små, positive tall.

a) Forklar tankegangen bak modellen

b) Velg $r = 0.98$, $q = 1.04$, $c = 0.0002$, $d = 0.001$, $x_1 = 50$, $y_1 = 200$. Lag et MATLAB-program som regner ut x_n og y_n for $n \leq 1000$. Plott følgene x_n og y_n i samme koordinatsystem. Hvorfor er toppene til x_n forskjøvet i forhold til toppene til y_n?

A10: m-filer, funksjoner og script

Hittil har vi programmert i kommandovinduet til MATLAB. Det er greit hvis du arbeider med korte programmer du bare vil bruke én gang, men svært upraktisk for lengre programmer og programmer du vil bruke om igjen senere. Slike programmer bør du skrive inn på egen kommandofil som du kan ta frem og bruke på nytt når du har bruk for den. I MATLAB kalles slike filer gjerne m-filer siden de skal ha endelsen .m. En m-fil inneholder

MATLAB-kommandoer (uten prompten `>>`) og kan skrives inn i en hvilken som helst editor, f.eks. emacs eller den innebygde editoren i MATLAB som ligger i et eget vindu (i noen versjoner av MATLAB spretter dette vinduet opp når du starter programmet, i andre må du klikke på et ikon i MATLAB-vinduet for å få det opp).

Det finnes to typer m-filer: script og funksjoner. Et script er en sekvens av MATLAB-kommandoer slik vi ville ha tastet dem inn i kommandovinduet (men altså uten prompten). Hvis vi samler disse kommandoene på en fil som heter `filnavn.m`, vil MATLAB utføre dem når vi gir kommandoen `>> filnavn` (uten endelsen .m) i kommandovinduet. For at MATLAB skal finne filen, må vi stå i samme katalog som filen er lagret i (med mindre filen ligger i søkestien til MATLAB). Vi kan sjekke hvilken katalog vi står i med kommandoen pwd (present working directory) og skifte katalog med kommandoen cd (change directory).

Hvis vi f.eks ønsker å lage et script som finner røttene til annengradsligningen $2x^2 + 3x + 1 = 0$, kan vi legge følgende kommandoer på filen `annengradsligning.m`

```
a=2;
b=3;
c=1;
x1=(-b+sqrt(b^2-4*a*c))/(2*a)
x2=(-b-sqrt(b^2-4*a*c))/(2*a)
```

Når vi skriver

```
>> annengradsligning
```

utfører MATLAB alle kommandoene i scriptet og returnerer svarene

```
x1          -0.5000
x2           -1
```

En funksjon er en m-fil som begynner med ordet 'function' og har én eller flere inn- og ut-parametre. Disse parametrene er tall eller matriser (eller i noen få tilfeller funksjoner). Variablene som brukes i funksjoner er lokale (med mindre vi definerer en global variabel, se `help global`) og lagres ikke i det ordinære arbeidsrommet.

Hvis vi ønsker å lage en funksjon som finner røttene til en vilkårlig annengradsligning $ax^2 + bx + c = 0$, kan vi la brukeren spesifisere ligningskoeffisientene a, b og c som inn-parametre til funksjonen, og la ut-parameteren som funksjonen returnerer, være en 2-dimensjonal radvektor som inneholder røttene `r1` og `r2`. På filen `annengrad.m` legger vi da følgende funksjonskode:

```
function [r1, r2]=annengrad(a,b,c)
if a~=0              % sjekker at a ikke er null
   r1=(-b+sqrt(b^2-4*a*c))/(2*a);
   r2=(-b-sqrt(b^2-4*a*c))/(2*a);
else
   disp('Koeffisienten til x^2 kan ikke være null')
end
```

(legg merke til kommandoen `disp('tekst')` som får MATLAB til å skrive ut tekst). Vi kan nå finne røttene til annengradsligningen $2x^2 + 3x + 1 = 0$, ved å gi kommandoen

```
>> [x1,x2]=annengrad(2,3,1)
```

MATLAB returnerer igjen svarene $x_1 = -0.5000$ og $x_2 = -1$. Det er en god vane å legge inn forklarende kommentarer i m-filene som forteller hva kommandoene gjør. I MATLAB kan vi skrive inn slike kommentarer etter prosenttegnet % som kommenterer ut resten av linjen.

Bemerkning: Når du starter MATLAB, vil du automatisk befinne deg i MATLABs hjemmeområde (MATLAB's default working directory) som ligger i søkestien til MATLAB. Hvis du vil lagre m-filer i en annen mappe og legge denne inn i søkestien til MATLAB, kan du bruke `set path` fra MATLABs `file`-meny. Etter hvert som du bruker MATLAB mye, vil du sannsynligvis lage mange m-filer til mange forskjellige formål. For å få bedre oversikt anbefaler vi at du lager en mappe for hvert formål, og legger hver enkelt mappe inn i søkestien.

Oppgaver til seksjon 10

1. Determinanten til en 2×2-matrise er definert ved

$$\det \begin{pmatrix} a & b \\ c & d \end{pmatrix} = ad - bc$$

Lag en m-fil (en funksjonsfil) som regner ut slike determinanter. Filen skal ha inn-parametre a, b, c, d og ut-parameter det $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$.

2. Skriv en m-fil (en funksjonsfil) som gir løsningen til ligningssystemet

$$\begin{aligned} ax + by &= e \\ cx + dy &= f \end{aligned}$$

når det har en entydig løsning. Filen skal virke på denne måten: Inn-parametrene er koeffisientene a, b, c, d, e, f, og ut-parametrene er løsningene x, y. Dersom ligningssystemet ikke har en entydig løsning, skal programmet svare: "Ligningssettet har ikke entydig løsning"(det behøver altså ikke å avgjøre om ligningssettet er inkonsistent eller har uendelig mange løsninger).

3. Lag en funksjonsfil som regner ut leddene i følgen $\{x_n\}$ gitt ved:

$$x_{n+2} = ax_{n+1} + bx_n \qquad x_1 = c, x_2 = d$$

Filen skal ha inn-parametre a, b, c, d, m og skal returnere de m første verdiene til følgen.

4. En dyrestamme består av tre årskull. Vi regner med at 40% av dyrene i det yngste årskullet lever videre året etter, mens 70% i det nest yngste årskullet lever videre året etter. Ingen dyr lever mer enn tre år. Et individ i det andre årskullet blir i gjennomsnitt forelder til 1.5 individer som blir født året etter. Et individ i det eldste årskullet blir i gjennomsnitt forelder til 1.4 individer som blir født året etter. La x_n, y_n, z_n være antall dyr i hvert årskull etter n år, og forklar hvorfor

$$\begin{aligned} x_{n+1} &= 1.5y_n + 1.4z_n \\ y_{n+1} &= 0.4x_n \\ z_{n+1} &= 0.7y_n \end{aligned}$$

a) Lag et MATLAB-program som regner ut x_n, y_n og z_n for $1 \leq n \leq 100$. Plott alle tre kurvene i samme vindu. Lag et nytt vindu der du plotter alle de relative bestandene $x'_n = \frac{x_n}{x_n+y_n+z_n}$,

$y'_n = \frac{y_n}{x_n+y_n+z_n}$, $z'_n = \frac{z_n}{x_n+y_n+z_n}$.

b) Gjenta punkt a), men bruk andre startverdier, f.eks. $x_1 = 300$, $y_1 = 0$, z_1=0. Sammenlign med resultatene i a). Gjør oppgavene enda en gang med et nytt sett av startverdier. Ser du et mønster?

c) La $A = \begin{pmatrix} 0 & 1.5 & 1.4 \\ 0.4 & 0 & 0 \\ 0 & 0.7 & 0 \end{pmatrix}$. Forklar at $\begin{pmatrix} x_n \\ y_n \\ z_n \end{pmatrix} = A^{n-1} \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix}$. Bruk dette til å regne ut x_{100}, y_{100} og z_{100} for starverdiene i a). Sammenlign med dine tidligere resultater.

5. Lag en funksjonsfil som regner ut leddene i følgen $\{x_n\}$ gitt ved:

$$x_{n+1} = ax_n(1 - x_n) \qquad x_1 = b$$

Filen skal ha inn-parametre a, b, m, og skal returnere de m første verdiene til følgen. Sett $m = 100$, se på tilfellene $b = 0.2$ og $b = 0.8$, og kjør programmet for hhv. $a = 1.5$, $a = 2.8$, $a = 3$, $a = 3.1$, $a = 3.5$, $a = 3.9$. Plott resultatene. Eksperimenter videre hvis du har lyst, men behold humøret selv om du ikke finner noe mønster; fenomenet du ser på er et av utgangspunktene for det som kalles "kaos-teori"!

A11: Anonyme funksjoner og linjefunksjoner

I forrige seksjon så vi hvordan vi kunne lagre funksjoner i MATLAB ved hjelp av funksjonsfiler. Trenger vi bare en enkel funksjon noen få ganger, er det imidlertid unødvendig komplisert å skrive og lagre en egen fil, og MATLAB har derfor innebygd to metoder for å definere funksjoner direkte i kommandovinduet – *anonyme funksjoner* ("anonymous functions") og *linjefunksjoner* ("inline functions"). Vær oppmerksom på at anonyme funksjoner først ble introdusert i MATLAB i versjon 7, og at de derfor ikke er tilgjengelig i eldre utgaver av programmet.

La oss begynne med linjefunksjoner. Hvis vi i kommandovinduet skriver

```
>> f=inline('x.*sin(1./x)');
```

lagrer MATLAB funksjonen $f(x) = x \sin \frac{1}{x}$ (legg merke til anførselstegnene rundt funksjonsuttrykket). Ønsker vi å vite hva $f(5)$ er, kan vi derfor skrive

```
>> f(5)
ans =
    0.9933
```

Ønsker vi å plotte grafen til f, går vi frem på vanlig måte:

```
>> x=0.01:0.01:1;
>> y=f(x);
>> plot(x,y)
```

Metoden fungerer også på funksjoner av flere variable:

```
>> f=inline('y.*sin(1./x)')
 f =
     Inline function:
     f(x,y) = y.*sin(1./x)
```

```
>> f(2,3)
ans =
    1.4383
```

Dette forteller oss at funksjonsverdien til funksjonen $f(x,y) = y\sin\frac{1}{x}$ i punktet $(2,3)$ er 1.4383. Legge merke til at i sitt svar på kommandoen

```
>> f=inline('y.*sin(1./x)')
```

har MATLAB ordnet variablene slik at vi vet at x er første og y er annen variabel. Vil vi tegne grafen til f, skriver vi

```
>> x=0.01:0.01:1;
>> y=0.01:0.01:1;
>> [x,y]=meshgrid(x,y);
>> z=f(x,y);
>> mesh(x,y,z)
```

La oss nå gå over til anonyme funksjoner. Disse definerer du ved hjelp av operatoren `@` på denne måten:

```
>> g=@(x,y)x.^2.*y+y.^3
g =
    @(x,y)x.^2.*y+y.^3
```

Funksjonen $g(x,y) = x^2y + y^3$ er nå lastet inn. Du kan regne ut verdier og tegne grafer på samme måte som ovenfor:

```
>> g(1,-2)
ans =
    -8
>> x=0.01:0.01:1;
>> y=0.01:0.01:1;
>> [x,y]=meshgrid(x,y);
>> z=g(x,y);
>> mesh(x,y,z)
```

Anonyme funksjoner er nyttige når du skal integrere funksjoner i MATLAB. Vil du f.eks. regne ut integralet $\int_0^2 e^{-\frac{x^2}{2}}\,dx$, skriver du

```
>> quad(@(x)exp(-x.^2),0,2)
ans =
       0.8821
```

(`quad` står for *quadrature*, et gammelt navn for integrasjon). Utelater du $@(x)$, får du en feilmelding:

```
>> quad(exp(-x.^2),0,2)
??? Error using ==> fcnchk
FUN must be a function, a valid string expression,
or an inline function object.
```

Det går imidlertid greit å bruke denne syntaksen til å integrere en linjefunksjon du allerede har definert:

```
>> h=inline('exp(-x.^2)');
>> quad(h,0,2)
ans =
    0.8821
```

Kommandoen fungerer fortsatt ikke dersom h er definert som en anonym funksjon!

Oppgaver til seksjon 11

1. Definer $f(x) = \frac{\sin x}{x}$ som en linjefunksjon. Tegn grafen til f og regn ut integralet av f fra $\frac{1}{2}$ til 2.

2. Gjenta oppgave 1, men definer nå f som en anonym funksjon.

3. Definer $f(x, y) = x^2y - y^2$ som en linjefunksjon og tegn grafen.

4. Gjenta oppgave 3, men definer nå f som en anonym funksjon.

A12: Symbolske beregninger

Alt vi har tatt for oss til nå, har dreid seg om hvordan vi kan beregne ting numerisk. Med MATLAB er det imidlertid også mulig å regne ut en del ting symbolsk ved hjelp av verktøykassen *Symbolic Math Toolbox*. Det enkleste eksemplet er kanskje brøkregning. Skriver vi

```
>> brok=sym(1)/sym(3)+sym(1)/sym(2)
>> eval(brok)
```

vil første linje returnere den symbolske verdien 5/6, mens andre linje vil beregne den numeriske tilnærmingen til denne som blir 0.8333 (sørg for at du forstår forskjellen mellom disse to!). Et mer avansert eksempel er derivasjon: Den deriverte til x^5 kan du regne ut ved å skrive

```
>> syms x
>> diff(x^5)
```

Kommandoen `syms` deklarerer en symbolsk variabel.

Et tredje eksempel på symbolske beregninger er integrasjon. Kommandoen

```
>> quad('x.^7.*exp(x.^2)',0,1)
```

regner ut en numerisk tilnærming til integralet $\int_0^1 x^7 e^{x^2} dx$. Numeriske beregninger hjelper oss imidlertid ikke til å finne det ubestemte integralet, men her kan vi isteden skrive:

```
>> syms x
>> uttrykk=int(x^7*exp(x^2))
```

Her vil `uttrykk` være en symbolsk formel for det ubestemte integralet. Legg merke til at kommandoen `int` erstatter `quad` når vi integrerer symbolsk. Argumentet til `int` er et MATLAB-uttrykk på samme form som i kallet til `quad`, bortsett fra at vi ikke trenger å skrive operasjonene punktvis og ikke tar med anførselstegnene rundt funksjonsuttrykket. Det bestemte integralet på symbolsk form kan vi regne ut ved å skrive

```
>> svar=int(x^7*exp(x^2),0,1)
```

Symbolske uttrykk kan ha mange variable, og de kan manipuleres. Vi kan for eksempel sette inn verdier for variablene i et symbolsk uttrykk ved hjelp av kommandoen `subs`. Hvis vi vender tilbake til integralet $\int_0^1 x^7 e^{x^2} dx$, og som før lar `uttrykk` være det eksakte uttrykket for det ubestemte integralet, så vil

```
>> svar=subs(uttrykk,'x',sym(1))-subs(uttrykk,'x',sym(0))
```

være en annen måte å regne ut det bestemte integralet på, der vi setter inn integralgrensene i det ubstemte integralet ved hjelp av kommandoen `subs`. Vi setter her inn `sym(1)` og `sym(0)`, siden `subs` produserer et nytt symbolsk uttrykk når vi setter inn en symbolsk verdi. Hadde vi satt inn `1` og `0` i stedet, ville `subs` produsere den tilhørende numeriske verdien.

I matriseregning er det også flere ting som er nyttig å gjøre symbolsk. La oss demonstrere det aller enkleste ved hjelp av matrisen

$$A = \begin{pmatrix} 1 & 2 & 0 \\ 0 & 1 & 1 \\ 0 & -2 & 1 \end{pmatrix}$$

Kommandoene

```
>> A=sym([1 2 0;0 1 1;0 -2 1])
>> inv(A)
```

definerer matrisen symbolsk og finner den eksakte inversen. Ligningssystemet

$$A\mathbf{x} = \begin{pmatrix} 5 \\ 3 \\ 3 \end{pmatrix}$$

kan nå løses eksakt ved kommandoen

```
>> A\sym([5;3;3])
```

Oppgaver til Seksjon 12

1. Deriver $f(x) = x \sin x$ og $g(x) = \sqrt{x}$ symbolsk.

2. Finn de ubestemte integralene $\int x^3\,dx$ og $\int x^3 \cos x\,dx$ ved hjelp av MATLAB. Finn også de eksakte verdiene til $\int_0^3 x^3\,dx$ og $\int_0^\pi x^3 \cos x\,dx$.

3. Inverter matrisen

$$A = \begin{pmatrix} 1 & 2 & 0 \\ 0 & 1 & 1 \\ 0 & -2 & 1 \end{pmatrix}$$

og løs ligningen

$$A\mathbf{x} = \begin{pmatrix} 5 \\ 3 \\ 3 \end{pmatrix}$$

som anvist ovenfor.

A13: Feilsøking

Ganske ofte inntreffer det en feil i et program, eller det ser ut som programmet aldri blir ferdig. I slike tilfeller må vi gjøre feilsøking. De vanligste feilene i MATLAB er at matrisedimensjoner ikke stemmer, eller at vi har brukt vanlige operasjoner (som $*$) istedenfor punkterte operasjoner (som $.*$).

Andre feil som ofte dukker opp, er:

1. Syntaktiske feil. Det er for eksempel ikke uvanlig å glemme en avsluttende `end` i en `for`- eller `while`-løkke.
2. Du aksesserer et element i en vektor som er utenfor grensene til vektoren. MATLAB sier da `index out of bounds...` Husk at du definerer en vektor med en gitt størrelse, og at posisjonen i en vektor som du angir, må være mindre enn denne.
3. Du ber om verdien til en variabel som ikke er definert. MATLAB sier `Undefined function or variable...`
4. I MATLAB kan en funksjon kalle seg selv, og man kan fort gjøre den feilen at programmet kaller seg selv i det uendelige. MATLAB vil da feile når programmet har kalt seg selv for mange ganger.

Det er også mulig å drive mer avansert feilsøking i MATLAB, men vi går ikke nærmere inn på dette her siden vi kun trenger å skrive korte programmer der MATLABs egne feilmeldinger som regel er nok til å skjønne hva som er galt.

Oppgaver til Seksjon 13

1. Finn feilen og rett opp programmet:

```
a=1:10;
sum=0;
for k=1:11
  sum=sum+a(k);
end
```

2. Finn feilen og rett opp programmet:

```
function fak=fakultet(n)
  fak=n*fakultet(n-1);
```

3.Finn feilen og rett opp programmet:

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

Kapittel 1

Register

`&`, 23
`.*`, 7
`./`, 7
`.^`, 8
`<`, 22
`<=`, 22
`==`, 23
`>`, 23
`>=`, 23
`A\b`, 17
`^`, 2
`a=b:h:c`, 9
`abs`, 2
`asin`, 2
`atan`, 2
`axis`, 11
`ceil`, 4
`clear`, 20
`contour`, 13
`cross`, 6
`det`, 5
`diary`, 20
`dot`, 6
`eig`, 6
`eval`, 30
`exp`, 2
`eye`, 16
`figure`, 10
`floor`, 4
`format long`, 2
`format short`, 3
`hold off`, 10
`hold on`, 10
`inline`, 28
`int`, 30
`inv`, 5
`linspace`, 9
`load`, 20
`max`, 8
`mesh`, 13
`meshgrid`, 12
`min`, 8
`ones`, 16
`pi`, 2
`plot`, 9
`plot3`, 13
`prod`, 8
`rand`, 4, 6, 16
`round`, 4
`rref`, 16
`save`, 20
`sqrt`, 2
`subplot`, 12
`subs`, 31
`sum`, 8
`sym`, 30
`syms`, 30
`who`, 20
`xlabel`, 11
`ylabel`, 11
`zeros`, 15
`~`, 23
`~=`, 23

anonym funksjon, 29

beregning, symbolsk, 30

feilsøking, 32
for-løkke, 21
funksjon
    anonym, 29
    linje-, 28
    m-fil, 26

if-else-setning, 21

komponentvis operasjon, 7

løkke, 21
linjefunksjon, 28

m-fil, 25
matrise
    i MATLAB, 4

prompt `>>`, 1

Python, 1

script, 26
Symbolic Math Toolbox, 30
symbolsk beregning, 30

transponert, 5

vektor
    i MATLAB, 5

while-løkke, 21