FORORD

Denne lefsa er ikke ment som en lærebok, men gir en kortfattet innføring i de mest brukte kommandoer og konstruksjoner i Matlab basert på eksempler. Den siste versjonen av Matlab er 6.1 som er innstallert på datasalen. Denne lefsa bruker ingen av de nye tingene som er introdusert i den siste versjonen.

MTF-NTNU Februar 2002

INNLEDNING

Matlab er et interaktiv program som er laget spesielt for tekniske og vitenskapelige beregninger. Det ble opprinnelig skrevet av Cleve Moler i slutten i 70-årene for å teste store Fortran programpakker i numerisk lineær algebra som Linpack og Eispack. Programmet har siden utviklet seg sterk med blant annet et eget programmeringspråk, sterk grafikk og objektprogrammering. I tillegg fåes spesialskrevne moduler innen en rekke fagområder (Toolboxes)

Det finnes mye litteratur om Matlab og her skal vi bare nevne noen få.

[1] A.Strømhylden og E. Wahl : *Introduksjon til Matlab. Tilleggshefte til Innføring i Informasjonsteknologi*
Tapir 1999

[2] F. Haugen : *Lær Matlab trinn for trinn*
TechTeach 2001

[3] E. P. Enander & A. Sjöberg : *Användarhandledning för Matlab5*
Uppsala Universitetet 1998

[4] Higham & Higham : *Matlab Guide*
SIAM 2000

[5] C. F. Van Loan : *Introduction to Scientific Computing*
Prentice-Hall , 2.utgave 2000

Referansene [4] og [5] inneholder Matlabprogrammer som kan lastes ned.
For liste over bøker som bruker/omhandler Matlab :
www.mathworks.com/support/books

HJELPESYSTEM

Start med `help help` dersom du ikke vet hvordan hjelpesystemet brukes. Istedenfor (eller i tillegg til) `help` kan vi bruke `doc`. Hjelpesystemet åpner nå et eget vindu. `doc` gir vanligvis mer omfattende informasjon enn `help`.

Skriv `help` for å gi en liste over søkeemner :
```
» help

HELP topics:

matlab\general       -  General purpose commands.
matlab\ops           -  Operators and special characters.
matlab\lang          -  Programming language constructs.
matlab\elmat         -  Elementary matrices and matrix manipulation.
matlab\elfun         -  Elementary math functions.
matlab\specfun       -  Specialized math functions.
matlab\matfun        -  Matrix functions - numerical linear algebra.
matlab\datafun       -  Data analysis and Fourier transforms.
matlab\audio         -  Audio support.
matlab\polyfun       -  Interpolation and polynomials.
matlab\funfun        -  Function functions and ODE solvers.
matlab\sparfun       -  Sparse matrices.
matlab\graph2d       -  Two dimensional graphs.
matlab\graph3d       -  Three dimensional graphs.
matlab\specgraph     -  Specialized graphs.
matlab\graphics      -  Handle Graphics.
matlab\uitools       -  Graphical user interface tools.
matlab\strfun        -  Character strings.
matlab\iofun         -  File input/output.
matlab\timefun       -  Time and dates.
matlab\datatypes     -  Data types and structures.
matlab\verctrl       -  Version control.
matlab\winfun        -  Windows Operating System Interface Files
(DDE/ActiveX)
matlab\demos         -  Examples and demonstrations.
toolbox\local        -  Preferences.
MATLAB6p1\work       -  (No table of contents file)

For more help on directory/topic, type "help topic".
For command syntax information, type "help syntax".
```

`help general` og `help ops` gir mange nyttige kommandoer

Lista viser at dersom vi f. eks ønsker å vite hvilken elementære matematiske funksjoner som finnes, skriver vi `help elfun`.
En annen nyttig søkekommando er `lookfor`. (skriv `help lookfor`)

VARIABLE

Navn på en variabel må begynne med en bokstav, fulgt av en vilkårlig rekke av bokstaver, tall og understrekningsymbolet (_). Mellomrom er ikke tillatt. Bare de 31 første tegnene er signifikante. Merk at Matlab skiller mellom store og små bokstaver slik at f. eks. Pi og pi er to forskjellige variabler.

Spesielle symboler

[ ]	Ved definisjon av matriser
( )	Indekser
' '	(To apostrofer). Markerer en streng
,	(Komma): Skiller indekser eller matriseelementer
;	(Semikolon): 1. Hindrer utskrift i kommandovinduet 2. Skiller linjer i matriser 3. Skiller programsetninger på en linje
%	Markerer begynnelsen på en kommentar-setning
:	Kolonoperator. Liste og matrise-generator
+	Addisjon
–	Subtraksjon
*	Multiplikasjon
.*	Elementvis multiplikasjon
/	(Høyre) divisjon
./	Elementvis (høyre) divisjon
\	Venstre divisjon
^	Eksponentiering
.^	Elementvis eksponentiering
'	(Enkel apostrof) : Transponering
...	Tre prikker etterhverandre ved skriving i kommandovinduet angir fortsettelse på neste linje.

Predefinerte variable

pi	3.141592...
`i`	Imaginær enhet, $\sqrt{-1}$
`j`	Imaginær enhet, $\sqrt{-1}$
`eps`	Relativ nøyaktighet, $2^{-52} \approx 2.22 \cdot 10^{-16}$
`realmin`	Minste flyttall, $2^{-1022} \approx 2.23 \cdot 10^{-308}$
`realmax`	Største flyttall. $(2 - \text{eps}) \cdot 2^{1023} \approx 1.8 \cdot 10^{308}$
`Inf`	Uendelig, eks 1/0
`NaN`	Ikke-et-tall, eks 0/0

Disse variable er ikke beskyttet og kan redefineres. Får tilbake sine opprinnelige verdier ved å bruke `clear`.

Eksempel :

```
» pi
ans =
    3.1416
» pi = 3.0
pi =
     3
» clear pi
» pi
ans =
    3.1416
```

Det er normalt ikke noen god ide å redefinere `pi`, mens `i` og `j` derimot brukes ofte som indekser slik at det kan bli nødvendig å omdefinere disse. Generelt bør en forsøke å unngå redefinering.

FILKOMMANDOER

Når vi er i kommando-vinduet, finnes det en rekke kommandoer vi kan bruke til å gi liste over filer, hoppe fra en mappe til en annen osv. Alle disse operasjonene kan selvfølgelig gjøres i Windows, men i enkelte tilfeller er det raskere å utføre dem som kommandoer istedenfor å bruke musa. Mange av disse kommandoene har ekvivalente kommandoer hentet fra det underliggende operativsystemet. Ved å starte kommandolinja med et utropstegn, antar Matlab at det som følger er en operativsystemkommando.

Liste over noen kommandoer : (skriv `help general`)

`what`	Gir en liste over M-filer i den mappa du er i
`dir`	Gir liste over filer og mapper i den mappa du er i
`cd`	brukes til å hoppe fra en mappe til en annen
`copyfile`	Kopierer innholdet av en fil til en annen. Dersom den andre fila ikke eksistere, blir den opprettet. Flytter også filer fra en mappe til en annen.
`delete`	Sletter filer
`mkdir`	Lag en ny mappe.

Nedenfor er vist en seanse der vi bruker noen av disse kommandoene.
Bruk `help` til å finne den nøyaktige syntaksen for hver av kommandoene.

```
» dir
.  ..  Blasius   Eks193    Falkner   Kap2      tull.m    Øving3
» what
M-files in the current directory c:\myfiles\matlabprog\sio1054
tull
```

```
» mkdir test
» copyfile tull.m test
» cd test
» dir
.      ..     tull.m
» delete tull.m
» cd ..
» dir
  . .. Blasius   Eks193    Falkner   Kap2      test tull.m    Øving3
```

Merk : en prikk er en forkortelse for navnet til den mappa du er i, mens to prikker er en forkortelse for den overliggende mappa (parent directory)
Vi nevnte ovenfor at dersom du setter et utropstegn foran en kommando, oppfattes dette som kall av en systemkommando.(Se forskjellen på `dir` og `!dir`). Den første er en Matlab-kommando mens den siste er en Windows-kommando. Dersom du ønsker å døpe om en fil, kan du bruke Windows-kommandoen `ren` : `!ren minfil.m dinfil.m` som døper om filen `minfil.m` til `dinfil.m`.
Dersom du eksempelvis ønsker å omdøpe alle filene som ender på `for` til filer som ender på `m`, kan dette gjøres enkelt ved : `!ren *.for *.m`

MATRISER OG VEKTORER

Skriver inn en 3×3 matrise. Semikolon angir slutt på hver matriselinje.

```
» A = [1 2 3; 4 5 6; 7 8 9]
A =
     1     2     3
     4     5     6
     7     8     9
```

Kan plukke ut hoveddiagonalen ved :

```
» diag(A)
ans =
     1
     5
     9
```

`diag` kan brukes til å plukke ut andre diagonaler (Se `help diag`)

Linjevektor :

```
» a = [10 11 12]
a =
    10    11    12
```

Kolonnevektor :

```
» b = [13; 14 ;15]
b =
    13
    14
    15
```

Får en kolonnevektor ved å transponere en linjevektor :

```
» a'
ans =
    10
    11
    12
```

Får en linjevektor ved å transponere en kolonnevektor :

```
» b'
ans =
    13    14    15
```

Ved å transponere `A` bytter vi om linjer og kolonner

```
» A = A'
A =
     1     4     7
     2     5     8
     3     6     9
```

Velger ut den første kolonna i `A` :

```
» a = A(:,1)
a =
     1
     2
     3
```

Velger ut den første linja i `A`

```
» b = A(1,:)
b =
     1     4     7
```

Velger det siste elementet i den første kolonna i `A`:

```
» A(end,1)
ans =
     3
```

Velger det siste elementet i den tredje kolonna i A:

```
» A(end,3)
ans =
     9

» b = b'
b =
     1
     4
     7
```

Lager en ny B matrise ved å sette sammen to kolonnevektorer :

```
» B = [a b]
B =
     1     1
     2     4
     3     7
```

Lager en ny C matrise ved å sette sammen de to matrisene A og B

```
» C = [A B]
C =
     1     4     7     1     1
     2     5     8     2     4
     3     6     9     3     7
```

Finner antall elementer i vektoren b:

```
» length(b)
ans =
     3
```

Finner dimensjonene av matrisa C:

```
» size(C)
ans =
     3     5
```

Lager en linjevektor av elementene 2,3,4 i første linje av C:

```
» d = C(1,2:4)
d =
     4     7     1
```

Lager en kolonne-vektor av elementene 1,2 i første kolonne av C:

```
» d = C(1:2,1)
```

```
d =
     1
     2
```

Setter `d` lik siste kolonne i `C`:

```
» d = C(:,3)
d =
     7
     8
     9
```

Adderer et element til begynnelsen og slutten av `d` . Merk at
Matlab øker dimensjonene dynamisk:

```
» d = [0;d;0]
d =
     0
     7
     8
     9
     0
```

Merk at vi kan addere en skalar direkte til en vektor selv om dette strengt tatt ikke er en tillatt vektor-operasjon

```
» b = d +1
b =
     1
     8
     9
    10
     1
```

Vi har også tomme matriser. Disse markeres med [] (to hakeparenteser).
Kan f.eks brukes til å fjerne en kolonne eller linje fra en matrise.

```
»A
A =
     1     4     7
     2     5     8
     3     6     9
» A(:,3) = []
A =
     1     4
     2     5
     3     6
```

AUTOMATISK GENERERING AV VEKTORER OG MATRISER.

Kan bruke kolonnotasjon `i:j:k` der `i` = startverdi, `j` = inkrement og `k` = sluttverdi. Dersom inkrementet utelates, antas inkrement = 1

```
» a = [1:5]
a =
     1     2     3     4     5

» a = [1 :0.5 : 3]

a =
    1.0000    1.5000    2.0000    2.5000    3.0000
```

Kan bruke vanlige parenteser eller helt utelate parenteser for linjevektorer :

```
» a = 1:0.1:1.5
a =
    1.0000    1.1000    1.2000    1.3000    1.4000    1.5000

» b = 7:-1:1
b =
     7     6     5     4     3     2     1
```

Kan også bruke **linspace**:

```
» d = linspace(1,2,4)
d =
    1.0000    1.3333    1.6667    2.0000
```

Syntaks : `linspace`(venstre endepunkt, høyre endepunkt, antall punkt)
Dette betyr at `linspace` genererer `n` ekvidistante punkt der `d(1) = a` og `d(n) = b` . Dersom antall punkt utelates, velges 100. (Se også `logspace`)

De følgende kommandoene brukes ofte til å initialisere vektorer og matriser

```
» a = zeros(5,1)
a =
     0
     0
     0
     0
     0

» b = ones(1,5)
b =
```

```
     1     1     1     1     1
```

```
» D = eye(3)
D =
     1     0     0
     0     1     0
     0     0     1
```

Her er et eksempel på bruk av indeksvektor. Merk at `a` blir en kolonnevektor fordi vi har brukt `a = zeros(5,1)` ovenfor. Om indeksvektoren er kolonne- eller linjevektor betyr ikke noe i dette tilfellet.

```
» j = 1:5
j =
     1     2     3     4     5
```

```
» a(j) = j*0.1
a =
    0.1000
    0.2000
    0.3000
    0.4000
    0.5000
```

Indeksvektorer brukes også til såkalt indirekte addressering.

```
»a = a';
» v(1) = 1; v(2) = 5; v(3) = 2;
» v
v =
     1     5     2
```

```
» b = a(v)
b =
    0.1000    0.5000    0.2000
```

Elementvise operasjoner

```
» x = linspace(pi/3,pi,6)
x =
    1.0472    1.4661    1.8850    2.3038    2.7227    3.1416
```

```
» y = x.*x
y =
    1.0966    2.1494    3.5531    5.3077    7.4132    9.8696
```

```
» y = sin(x).*x
y =
    0.9069    1.4580    1.7927    1.7121    1.1074    0.0000
```

```
» y = sin(x)./x
y =
    0.8270    0.6784    0.5046    0.3226    0.1494    0.0000

» y = sin(x).^2
y =
    0.7500    0.9891    0.9045    0.5523    0.1654    0.0000
```

Tilslutt sorterer vi y:

```
»  y = sort(y)
y =
    0.0000    0.1654    0.5523    0.7500    0.9045    0.9891
```

Her er flere måter å beregne skalarproduktet på

```
» y = sum(x.*x)
y =
   29.3895

» y = x'*x
y =
   29.3895

» y = dot(x,x)
y =
   29.3895
```

LØSNING AV LIGNINGSYSTEM

Vi skal løse systemet $\boldsymbol{A} \cdot \boldsymbol{x} = \boldsymbol{b}$. Løsningen kan skrives symbolsk ved $\boldsymbol{x} = \boldsymbol{A}^{-1} \cdot \boldsymbol{b}$. Det er mulig å beregne den inverse av $\boldsymbol{A}$-matrisa og deretter multiplisere med $\boldsymbol{b}$, men dette er ineffektivt. Løsning av ligningsystemet i Matlab skrives `x = A\b` der `b` er en kolonnevektor. Operatoren \ kalles *venstre divisjon* (heller mot matrisa som skal «divideres» på ; symbolsk kan $\boldsymbol{A}^{-1}$ oppfattes som divisjon.
Merk at `b/A` ikke er definert. Derimot kan vi beregne `b'/A` der `b'/A = b'*inv(A)`. (Husk at her er `b'` en linjevektor)

Eksempel 1

```
» A = [-3 3/2 0; 3/4 -9/4 5/4; 0 5/6 -19/9]
A =
   -3.0000    1.5000         0
    0.7500   -2.2500    1.2500
         0    0.8333   -2.1111
```

```
» b = [1/2 1 3/2]'
b =
    0.5000
    1.0000
    1.5000

» x = A\b

x =
   -0.8952
   -1.4571
   -1.2857

» format rat
» x
x =
   -94/105
   -51/35
    -9/7
```

Da matriseelementene her er rasjonale tall, kan vi få løsningen på pen brøkform. Dette gjøres ved å forandre utskriftsformatet for kommando-vinduet fra `format` til `format rat`. Går deretter tilbake til `format short` (Se `help format`)

```
» format
```

Eksempel 2

```
A =
    0.4096    0.1234    0.3678    0.2943
    0.2246    0.3872    0.4015    0.1129
    0.3645    0.1920    0.3728    0.0643
    0.1784    0.4002    0.2786    0.3927

b =
    0.4043
    0.1550
    0.4240
   -0.2557

>> x = A\b
x =
   -0.0061
   -1.5556
    2.0315
   -0.5043
```

PROGRAMMERINGSDEL

Til nå har vi sett på bruk av Matlab som kalkulator. De konstruksjonene og kommandoene som følger her, blir vanligvis brukt i program selvom de også kan skrives direkte i kommandovinduet. Programmene skrives med en tekst-editor Den innebygde editoren er vanligvis god nok, da Matlabprogrammene normalt er små. Er det behov for en mer sofistikert editor, finnes **TextPad** ved NTNU. Programmene blir lagt i filer med endelse m. Kalles derfor M-filer. Eks. : Prog.m Har to typer M-filer: Skript M-filer, også kalt kommando-filer samt funksjons M-filer som inneholder en funksjonsdefinisjon.

LOGISKE UTTRYKK

Matlab har ikke logiske variable. Alle logiske uttrykk får verdien 1 eller 0 der sant = 1 og usant = 0.

Sammenligningsoperatorer (Skriv `help/doc ops`)

<	mindre enn
<=	mindre eller lik
==	lik
>=	større eller lik
>	større enn
~=	ikke lik (ulik, forskjellig)

Når `a` og `b` er tall, blir `a == b, a > b , a ~= b` eksempler på logiske uttrykk.

Logiske operatorer (Skriv `help ops`)

&	både og (and)
|	enten eller (or)
xor	eksklusiv eller(exclusive or)
~	negasjon (not)

I tillegg har vi `any` og `all`

Husk at sant = 1 og usant = 0. La *A* og *B* være to logiske uttrykk.Sannhetstabellen nedenfor viser verdiene ved bruk av operatorene.

		and	**or**	**xor**	**not**
A	*B*	*A*&*B*	*A*|*B*	xor(*A*,*B*)	~*A*
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

Eksempler

a = 1, b = 2, c = 3 ,d = 4

```
» y = (a<=1)| (c>a)
y =1

» y = ( a > 1)& (d >b)
y =
     Ø

» y = ( a == 1)& (d ~=b)
y =
     1

» y = xor(( a == 1) , (d ~=b))
y =
     Ø
```

De logiske uttrykkene brukes ved forgreininger og løkker.

FORGREININGER

Kommandoer: **if, else, elseif .** (Dessuten kommandoen **switch**)

Variant 1

```
if   logisk uttrykk
        setninger
end
```

Setninger utføres bare dersom det logiske uttrykket = *sant*.
Eksemplet kan også skrives på en linje:

`if` , *logisk uttrykk, setninger*, `end`

Denne siste formen kan lett bli uoversiktelig.

Variant 2

```
if    logisk utrykk
        setninger1
else
        setninger2
end
```

setninger1 utføres dersom det logiske uttrykket = *sant* ellers
utføres *setninger2*.

Variant 3

```
if    logisk utrykk1
        setninger1
elseif  logisk uttrykk2
        setninger2
else
        setninger3
end
```

setninger1 utføres bare dersom *logisk uttrykk1* = *sant*. *setninger1* og *2* blir da ikke utført. Dersom *logisk uttrykk1* = *usant* utføres bare *setninger 2* dersom *logisk uttrykk2* = *sant*. Dersom både *logisk uttrykk1* og *2* = *usant* utføres *setninger3* (og bare da).

Ved bruk av sammenligningsoperatorer kan de variable være vektorer og matriser. La `x` og `y` være vektorer med `n` komponenter. Da vil if `x < y` bare være sant dersom $x_i < y_i$, `i = 1,2, .. n` . Tilsvarende for matriser.

Et eksempel der vi kan bruke konstruksjonen fra variant 3

Anta at vi skal regne om tallkarakterer til bokstavkarakterer og følgende relasjoner er gitt :

$$[1.0-1.75] \Rightarrow A \text{ , } (1.75-2.25] \Rightarrow B \text{ , } (2.25-2.75] \Rightarrow C$$
$$(2.75-3.25] \Rightarrow D \text{ , } (3.25-4.25] \Rightarrow E \text{ , } > 4.25 \Rightarrow F \text{ (stryk)}$$

Som Matlab-kode :

```
if (tallk <= 1.75)
   disp ('Karakter er A');
elseif (tallk <= 2.25)
   disp ('Karakter er B');
elseif (tallk <= 2.75)
   disp ('Karakter er C');
elseif (tallk <= 3.25)
   disp ('Karakter er D');
elseif (tallk <= 4.25)
   disp ('Karakter er E');
else
    disp ('Karakter er F (stryk)');
end
```

Merk at parentesen rundt f.eks `(tallk < 1.75)` ikke behøves.

ITERASJONSLØKKER

Matlab har to kommandoer, **for** og **while**, for gjentatt utførelse av setninger.

Den generelle syntaksen for en **for**- løkke er :

```
for variabel = uttrykk
        setninger
end
```

variabel er navnet på løkke-variabelen (Løkka kan, om ønskelig , skrives på en linje).

Eksempel

```
>> for k = [1 2.5 3 4.5]
x= k^2
end

x =
     1
x =
    6.2500
x =
     9
x =
   20.2500
>>
```

Her er *uttrykk* en vektor slik at løkkevariabelen `k` forløpende går gjennom verdiene i vektoren.
Det vanligste er at *uttrykk* tilordner en startverdi, en inkrement-verdi og en sluttverdi til løkke-variabelen. Inkrementet kan være både positivt og negativ eller det kan utelates. I det siste tilfellet antas 1 som inkrement-verdi. Vanligvis brukes kolonnotasjonen til å definere løkke-variabelen. Vi kan ha flere løkker inne i hverandre :

```
for variabel1 = uttrykk1
        setninger1
        for variabel2 = uttrykk2
                setninger2
        end
end
```

Eksempel

```
» A = zeros(4);
» for k = 1:3
      A(k,k) = 4;
      A(k,k+1) = 1;
```

```
      A(k+1,k) = 1;
  end
» A(4,4) = 4;
» A
A =
     4     1     0     0
     1     4     1     0
     0     1     4     1
     0     0     1     4
```

Kommandoen **while** utfører setninger så lenge et logisk uttrykk er sant.
Den generelle syntaksen for en **while**-løkke er :

```
while logisk uttrykk
      setninger
end
```

setninger blir ikke utført dersom det logiske uttrykket er usant. Også her kan vi ha flere løkker inne i hverandre.

Merk at både **for** og **while** tester på den første linja i løkka .Dersom man ønsker å hoppe ut av løkkene andre steder, kan det gjøres ved å bruke kommandoen **break**. Merk at **break** bare gir uthopp fra den innerste løkka dersom det er flere løkker. Dersom **break** brukes utenfor **for** og **while**-løkker, stoppes eksekveringen av programmet. Dette kan være nyttig ved feilfinning.

Nedenfor er noen typiske varianter på bruken av **for** og **while** i iterasjonsløkker. Som eksempel ser vi på en iterasjonsprosess der vi skal finne kvadratrota x av et tall c : $x = \sqrt{c}$. Matematisk er dette ekvivalent md å finne nullpunktet av funksjonen $f(x) = 0$ der $f(x) = x^2 - c$. Ved bruk av Newton-Raphsons metode får vi følgende iterasjonsprosess :

$$x_{m+1} = x_m + \Delta x_m \ , \ m = 0,1,...$$

$$\Delta x_m = -\frac{(x_m^2 - c)}{2x_m}$$

For å starte iterasjonsprosessen, må vi tippe en startverdi x_0 . Dessuten må vi ha et stoppkriterium. For å være spesifikk, velger vi $c = 20$; vi skal finne $x = \sqrt{20}$. Velger startverdi $x_0 = 5$.

Variant 1

```
% Program VARIANT1
itmax = 10; epsi = 1.0e-5; x = 5.0;
c = 20.0;
it = 0;
while 1
    it = it + 1;
    if it > itmax
        disp('*** Maks. antall iterasjoner ***');
        break
    end
    dx = -(x^2 - c)*0.5/x;
    x = x + dx
    if abs(dx) < epsi
        break
    end
end

dx =-0.5000
x = 4.5000

dx = -0.0278
x = 4.4722

dx =-8.6266e-005
x = 4.4721

dx =-8.3203e-010
x = 4.4721

>>
```

Dette er en evighetsløkke da betingelsen *while 1* alltid er oppfylt. Uthopp fra løkka skjer enten når maks. antall iterasjoner er overskredet eller når konvergens er oppnådd. Selv om vi skriver ut verdien av *it* og *dx*, er det lurt å gjøre oppmerksom på at maksimalt antall iterasjoner er overskredet.

Variant 2

```
% Program VARIANT2
itmax = 10; epsi = 1.0e-5; x = 5.0;
c = 20.0;
for it = 1 : itmax
    dx = -(x^2 - c)*0.5/x;
    x = x + dx;
    if abs(dx) < epsi
        break
    end
```

```
end
if it >= itmax
        disp('*** Maks. antall iterasjoner ***');
        break
    end
```

(Resultat som for **variant1**)

Denne løkka som utføres maksimalt *itmax* ganger, er mer kompakt. Uthopp fra løkka før maks. antall iterasjoner er oppnådd, skjer dersom vi har konvergens. Utskrift om maks. antall iterasjoner ligger nå etter løkka.

Variant 3

```
% Program VARIANT3
itmax = 10; epsi = 1.0e-5; dx = 1.0;
x = 5.0; c = 20.0;
it = 0;
while (it <= itmax) & (abs(dx) > epsi)
    it = it + 1;
    dx = -(x^2 - c)*0.5/x;
    x = x + dx;
end
if it > itmax
    disp('*** Maks. antall iterasjoner ***')
    break
end
```

(Resultat som for **variant1**)

Denne løkka utføres bare dersom *it* <= *itmax* og samtidig *dx* > *epsi*. Forat løkka skal utføres minst en gang, må *dx* ved inngang settes til en verdi større enn *epsi*. Det skjer ingen uthopp fra løkka. Antall iterasjoner må nå telles dersom vi vil følge iterasjonsprosessen. Utskrift om maks. antall iterasjoner ligger nå som i **variant2**, etter løkka. Vi skal se på noen eksempler på bruk av variant 2 og 3 etter vi har sett litt nærmere på funksjoner.

FUNKSJONER

Funksjoner programmeres i funksjons M-filer og tilsvarer det som generelt kalles subprogram. Dette er program som brukes av andre program.

Generell syntaks : **function** [*ut1,ut2,...utn*] = *fnavn*(*inn1,inn2,....innk*)

Linja over definerer en funksjon med navn *fnavn*. Vi har *k* inn-parametre og *n* ut-parametre. Dersom vi bare har en ut-parameter, kan vi utelate hake-parentesene.

Vi ser på et enkelt eksempel der vi har laget en funksjon som beregner parabelen $y = x^2$. Funksjonen lagres som *func.m*

```
function y = func(x)
y = x^2;
» y = func(2)

y =
     4
```

`x` og `y` kan være vektorer eller matriser, men da må funksjonen skrives slik at den opererer rett på denne typen variable.

```
function y = func(x)
y = x.^2;
» z = [0 : 0.1 : 0.5];
» v = func(z)
v =
         0    0.0100    0.0400    0.0900    0.1600    0.2500
```

Legg merke til at det ikke er noen sammenheng mellom navnene på de variable i definisjonen av funksjonen og navnene som benyttes når den brukes. (Når vi bruker en funksjon, sier vi at vi *kaller* funksjonen). Men vi må passe på at de er av samme type. Legg også merke til at da `x` er en linjevektor blir også `y` en linjevektor. I mange tilfeller er det lurt å initialisere ut-parametrene dersom de er vektorer og matriser. I tilfellet over kan vi f. eks. skrive :

```
function y = func(x)
y = zeros(size(x));
y = x.^2;
```

Ved å bruke `size` sørger vi for at funksjonen virker for både linje- og kolonne-vektorer.

Dersom du redefinerer dine inngangsvariable inne i funksjonen, vil disse bli mellomlagret til funksjonen er utført. (Det blir ikke laget kopi av inngangsdata som ikke redefineres). Matlab bruker her en overførings-metode for parametre som kalles "call by value". Dermed blir ikke de opprinnelige verdiene ødelagt. Dette betyr også at du ikke sparer plass ved å redefinere inngangsdata. Merk også at alle lokale variable som defineres i en funksjon, forsvinner når funksjonen er utført.

```
function y = func(x)
y = zeros(size(x));
a = 1;
y = x.^2;
x = 2*x;
```

```
» v = func(z)
v =
         0    0.0100    0.0400    0.0900    0.1600    0.2500
» z
z =
         0    0.1000    0.2000    0.3000    0.4000    0.5000
» a
??? Undefined function or variable 'a'.
```

Vi ser her at `z` har fremdeles sin opprinnelige verdi. Den lokale variable `a` som vi definerte, blir slettet når funksjonen er utført.

Vi kan også overføre parametre ved bruk av kommandoen **globa**l.

```
function y = func(x)
y = zeros(size(x));
global P Q;
y = x.^2 + P + Q ;

» global Q P;
» Q = 1; P = 2;
» v = func(z)
v =
    3.0000    3.0100    3.0400    3.0900    3.1600    3.2500
```

Med `Q = 1` og `P = 2` har vi addert 3 til vektoren `v` . Merk at rekkefølgen av de globale variable kan være forskjellig i det kallende programmet og det kalte programmet. (Godt nytt for dem som er vant til Fortran). Flere globale variable skal skilles med mellomrom; ikke komma. Sletting av globale variable ved å bruke `clear` virker bare i kommandovinduet.

Subfunksjoner
Det er mulig å bruke funksjoner inne i funksjoner. En slik funksjon, kalt *subfunksjon*, må ligge helt tilslutt i hovedfunksjonen.

```
function y = func(x)
y = zeros(size(x));
a = f(x);
y = a*x.^2;
%------------------
function fac = f(t)
fac = sum(t);
» v = func(z)
v =
         0    0.0150    0.0600    0.1350    0.2400    0.3750
```

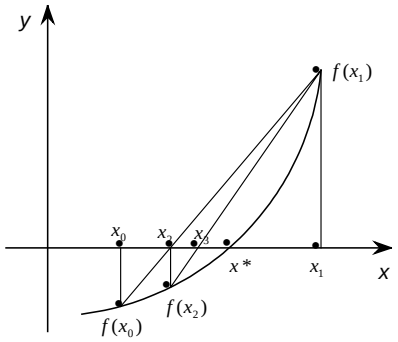
Her er `func` hovedfunksjonen og `f` subfunksjonen. Skriv `help function` for et

annet eksempel.

EKSEMPEL PÅ BRUK AV FUNKSJONER

Nullpunksbestemmelse ved bruk av sekantmetoden.

Sekantmetoden er beskrevet i avsnitt 2.2 i kompendiet. For oversiktens skyld gjentar vi beskrivelsen her.

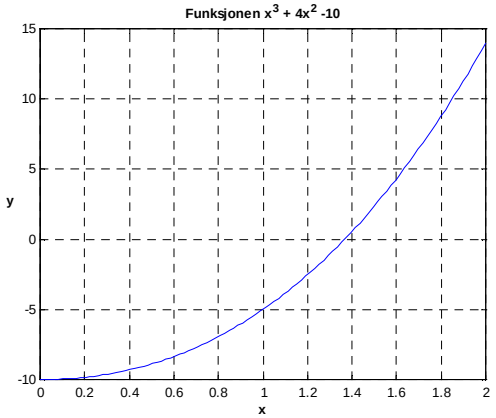


Vi skal finne nullpunktet $x*$ (kan være flere) av funksjonen $y = f(x)$.
Utfører en iterasjonsprosess etter følgende skjema :

$$x_{m+1} = x_m + \Delta x \ , \ m = 1, 2, \ldots$$

$$\Delta x = -f(x_m) \cdot \left[\frac{x_m - x_{m-1}}{f(x_m) - f(x_{m-1})} \right]$$

Figuren viser prosessen for $m = 2$. Vi har valgt å sette iterasjonsindekset m nede da det ikke er noen fare for kollisjon med andre indekser.
For å komme i gang, må vi tippe to verdier x_0 og x_1. Prosessen gjentas til et konvergenskriterium er oppfylt, f.eks. $|\Delta x| < \varepsilon_1$ eller $|\Delta x| < |x| \cdot \varepsilon_2$. Kombineres gjerne med $|f(x_{m+1})| < \varepsilon_3$.
La oss bruke sekantmetoden til å finne den reelle rota av tredjegradpolynomet $f(x) = x^3 + 4x^2 - 10$. (Et tredjegradspolynom med reelle koeffisienter har alltid minst en reell rot). Figuren nedenfor viser polynomet tegnet for $x \in [0 \ , \ 2]$.



Vi ser at vi har et nullpunkt i intervallet [1.2 , 1.4] , og på en lommekalkulator med nullpunktsløser finner vi nullpunktet $x^* = 1.365200134$ med 10 korrekte siffer. La oss gjenta beregningen med bruk av sekantmetoden og bruker startverdier $x_0 = 0$ og $x_1 = 2$.

I den første versjonen utfører vi iterasjonsprosessen et visst antall ganger uten å spesifisere noe iterasjonkriterium. Dette er ofte lurt når vi begynner på et problem der vi er usikker på iterasjonsforløpet.

Programmet **sekant1** gir følgende resultat :

```
 m         dx              f(x)              x
-------------------------------------------------
 1    -1.17e+000       1.40e+001       0.8333333333
 2     3.75e-001      -6.64e+000       1.2087912088
 3     2.11e-001      -2.39e+000       1.4196238401
 4    -5.87e-002       9.22e-001       1.3608992155
 5     4.22e-003      -7.14e-002       1.3651166138
 6     1.14e-004      -1.87e-003       1.3652302546
 7    -2.41e-007       3.98e-006       1.3652300134
 8     1.34e-011      -2.21e-010       1.3652300134
```

Vi ser at vi har fått 10 korrekte siffer etter 7 iterasjoner. Legg merke til at prosessen går langsomt i begynnelsen; prosessen skyter fart først etter 6 iterasjoner. Det gjelder generelt både for Newton-Raphsons metode og sekantmetoden at vi må være nær rota for å få rask konvergens.
Listingen av programmet er vist nedenfor.

```
% Program sekant1
clear
x0 = 0.0 ; x1 = 2.0; % Startverdier
f0 = func(x0);
mmax = 8;
s1 = '%2.0f    % 7.2e     % 7.2e      %13.10f \n';
disp(' m         dx              f(x)              x ' )
disp('-------------------------------------------------')
for m = 1 : mmax
   f1 = func(x1);
   dx = - f1*(x1 - x0)/(f1 - f0);
   x = x1 + dx;
   fprintf(s1,m,dx,f1,x);
   x0 = x1;
   x1 = x;
   f0 = f1;
end

function y = func(x)
y = x^3 + 4*x^2 - 10;
```

Når vi nå vet at iterasjonsprosessen forløper greit, kan vi legge inn et konvergenskriterium. Dette er gjort i versjonen **sekant2** nedenfor.
(Se avsnittet *Utskrift av tabeller* angående `fprintf`)

```
% Program sekant2
x0 = 0.0 ; x1 = 2.0; % Startverdier
f0 = func(x0);
itmax = 10; epsi = 1.0e-7; it = 0; dx = 1;
while (abs(dx) >= epsi ) & ( it <= itmax)
   it = it + 1;
   f1 = func(x1);
   dx = - f1*(x1 - x0)/(f1 - f0);
   x = x1 + dx;
   x0 = x1;
   x1 = x;
   f0 = f1;
end
if (it > itmax)
   disp( '*** Maks. antall iterasjoner ***')
end
fprintf( '\n x =  %13.10f \n',x);
```

FUNKSJON SOM INN-PARAMETER I EN FUNKSJON

Eksempel : Nullpunktsløsere

Anta at vi ønsker å lage en funksjon **sekant** som finner røtter av vilkårlige funksjoner. I dette tilfellet ønsker vi å bruke navnet på funksjonen som vi skal finne røttene av, som innparameter i **sekant**. Dette får vi til ved å bruke kommandoen **feval** (**f**unction **eval**uation). Ved å bruke **sekant2** som modell, kan **sekant** skrives som følger :

```
function rot = sekant(fname,x0,x1,itmax,epsi)
f0 = feval(fname,x0);
it = 0; dx = 1;
while (abs(dx) >= epsi ) & ( it <= itmax)
   it = it + 1;
   f1 = feval(fname,x1);
   dx = - f1*(x1 - x0)/(f1 - f0);
   x = x1 + dx;
   x0 = x1;
   x1 = x;
   f0 = f1;
end
if (it > itmax)
   disp( '*** Maks. antall iterasjoner ***')
end
```

Dersom vi bruker **sekant** på funksjonen **func** i det forrige eksemplet,får vi:

```
rot = sekant( 'func',x0,x1,itmax,epsi)
```

Legg merke til at istedenfor `fname`, setter vi inn det virkelige navnet på den funksjonen som vi skal finne rota av. Denne er lagret som en m-fil med navn `func.m` og må ligge på søkestien. Merk også apostrofene siden kommandoen **feval** antar at `fname` er en streng.
Fra og med versjon 6.0 av Matlab anbefales det å bruke @ (krøllnabla) foran funksjonsnavnet istedenfor å sette inn funksjonsnavnet som en streng. For tilfellet ovenfor med den nye notasjonen :

```
    rot = sekant(@func,x0,x1,itmax,epsi)
```

Da denne nye standarden ikk er kompatibel med tidligere versjoner av Matlab, velger vi å bruke den gamele måten med å angi navnet som en streng.

Det må nevnes at sekantmetoden i umodifisert versjon som vist ovenfor, ikke er noen god nullpunktsløser i generelle tilfeller. Det er bedre å bruke Matlabs egen nullpunktsløseren **fzero**. (Har kalkulatoren din en nullpunktsløser, er dette vanligvis en variant av **fzero**).
La oss bruke **fzero** på funksjonen ovenfor. Anta først at vi ikke kjenner noe intervall som inneholder nullpunktet, men vi har en anelse om at nullpunktet ligger i nærheten av 1.0

```
» format long
» x0 = 1.0;
» rot = fzero('func',x0)
Zero found in the interval: [0.54745, 1.4525].

rot =
   1.36523001341410
»
```

fzero kan kalles på en rekke måter(skriv `help fzero`). Dersom vi kjenner et intervall som rota ligger i, kan vi spesifisere dette.

```
» x0 = [1.2 1.4];
» rot = fzero('func',x0)
Zero found in the interval: [1.2, 1.4].
rot =
   1.36523001341410
»
```

Normalt vil den siste varianten være raskere. I begge tilfellene ovenfor har vi funne rota med full presisjon, noe vi kan tillate oss i dette enkle tilfellet. For mer kompliserte tilfeller kan dette være både tidkrevende og unødvendig. Anta at det er tilstrekkelig med ca. 5 siffers nøyaktighet.

```
» x0 = [1.2 1.4];
» options = optimset('Tolx',1.0e-5);
```

```
» rot = fzero('func',x0,options)
Zero found in the interval: [1.2, 1.4].
rot =
   1.36523001552733
»
```

Vi ser at vi har fått adskillig mer enn 5 korrekte siffer. (Skriv `help optimset` for å se den totale lista over opsjoner)

Eksempel : Bestemte integral

La oss lage en funksjon som integrerer en funksjon $y = f(x)$ fra $y = a$ til $y = b$ ved bruk av trapesmetoden. Deler intervallet [a , b] i n deler der hver del har lengde $h = \frac{(b-a)}{n}$. Med $x_1 = a$ og $x_{n+1} = b$ blir $x_k = a + h(k-1)$, $k = 1,2,..,n+1$.
Fra figuren ovenfor får vi følgende uttrykk for arealet :

$$A = \frac{h}{2}(y_1 + y_{n+1}) + h(y_2 + y_3 + ... + y_n)$$
$$= -\frac{h}{2}(y_1 + y_{n+1}) + h(y_1 + y_2 + ... + y_{n+1})$$

La oss som et eksempel integrere sin(x). Funksjonen **trapes1** gjør dette.

```
function intsin = trapes1(a,b,n)
h = (b-a)/n; % Intervall-lengde
np = n + 1 ; % Antall knutepunkt
s = 0;
for k = 2 : n
   x = a + (k-1)*h; y = sin(x);
   s = s + y*h;
end
intsin = s + h*(sin(a) + sin(b))*0.5;
```

Vi bruker **trapes1** til å beregne $\int_0^{\pi} \sin(x)\,dx = 2.$

```
» a = 0; b = pi; n = 50;
» value = trapes1(a,b,n)
value =
    1.9993
```

Problemet med denne versjonen er at vi kan bruke den bare til å integrere sinus-funksjonen. Dersom vi ønsker å integrere en annen funksjon, må vi gjøre en forandring i programmet. Bruker derfor kommandoen **feval** som vi brukte i programmet **sekant** til å sette funksjonsnavnet i parameterlista. Benytter samtidig anledningen til å gjøre **trapes1** mer effektiv ved å beregne alle

funksjonsverdiene før vi går inn i summasjonsløkka. Bruker dessuten den innebygde kommandoen **sum**. Den nye versjonen er vist nedenfor.

```
function trapint = trapes2(fname,a,b,n)
% Integrerer en funksjon med bruk av trapesmetoden
h = (b-a)/n; % Intervall-lengde
np = n + 1 ; % Antall knutepunkt
x = linspace(a,b,np); % Knutepunktsverdier
y = feval(fname,x); % Funksjonsverdier
s = h*sum(y);
trapint = s - h*(y(1) + y(np))*0.5;
```

`fname` er navnet (på funksjonen som skal integreres.

Vi gjentar beregningen som vi gjorde ovenfor :

```
» a = 0; b = pi; n = 50;
» value = trapes2('sin',a,b,n)
value =
    1.9993
```

La oss deretter bruke **trapes2** på funksjonen som vi har brukt under nullpunkts-eksemplene:

```
function y = func(x)
y = x.^3 + 4.*x.^2 - 10;
```

La oss integrere denne mellom `x = 1` og `x = 2` der det analytiske resultatet er $\frac{37}{12} = 3.08333\ldots$. Legg merke til at **trapes2** må ha alle funksjonsverdiene samlet i en vektor. Derfor må vi nå bruke elementvise operatorer i **func**.

```
» a = 1.0; b = 2.0; n = 50;
» value = trapes2('func', a, b, n)
value =
3.0839
```

Vi gjentar at fra og med versjon 6.0 av Matlab anbefales det å bruke @ (krøllnabla) foran funksjonsnavnet istedenfor å sette inn funksjonsnavnet som en streng. For tilfellet ovenfor : `value = trapes2(@func, a, b, n)`

Eksemplene ovenfor er ment å vise programmering i Matlab. Når en gitt funksjon skal integreres, velger vi heller å bruke metodene som er innebygget i Matlab, f.eks **quad** og **quadl**.

Ordinære differensialligninger. ODE45

I lefsa *ODE* ser vi på litt avansert bruk av odeløsere i Matlab**.** La oss her se på enkel bruk av **ode45** som på mange måter er arbeidshesten blant odeløserne i Matlab. Som eksempel bruker vi en ikke-lineær svingeligning med dempning.

$$\frac{d^2z}{dt^2} = -\alpha \cdot \left(z + \frac{dz}{dt}\left|\frac{dz}{dt}\right| \right) \tag{1}$$

Startbetingelser : $z(0) = h \ , \ \frac{dz}{dt}(0) = 0$ (b)

(1) skrevet som system :

$$\begin{aligned} \frac{dz}{dt} &= v \\ \frac{dv}{dt} &= -\alpha \cdot (z + v|v|) \\ z(0) &= h \ , \ v(0) = 0 \end{aligned} \tag{2}$$

Når vi skal bruke **ode45** eller de andre løserne i Matlab, må (2) kodes som en Matlab-funksjon og legges i en egen m-fil. Istedenfor z og v bruker vi nå en vektor y med 2 elementer. Setter $z = y(1)$ *og* $v = y(2)$ slik at systemet i (2) blir :

$$\begin{aligned} \frac{dy(1)}{dt} &= y(2) \\ \frac{dy(2)}{dt} &= -\alpha \cdot (y(1) + y(2)|y(2)|) \\ y(1)(0) &= h \ , \ y(2)(0) = 0 \end{aligned} \tag{3}$$

I den videre beregningen setter vi $\alpha = 0.07$ og $h = 8$ (b)

(3) skrevet i Matlab kan f.eks se slik ut :

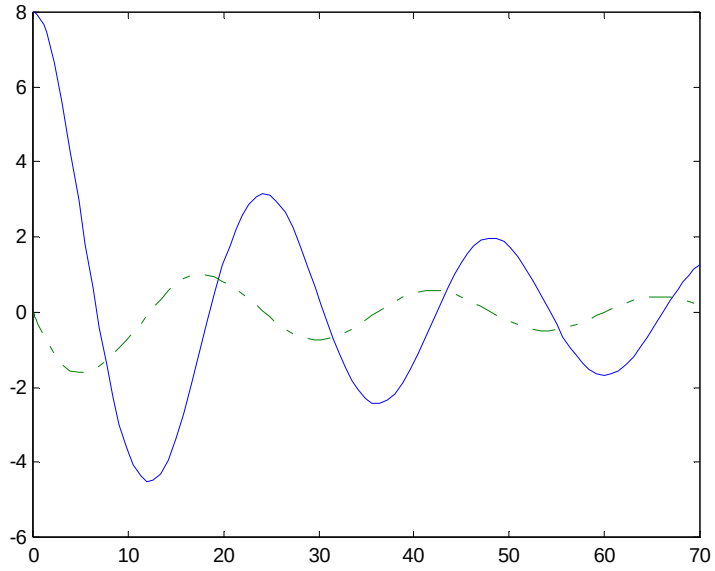
```
function dydt = fcn(t,y)
dydt = zeros(2,1)
dydt(1) = y(2);
dydt(2) = -0.07*(y(1) + y(2)*abs(y(2));
```

Noen kommentarer : Denne funksjonen heter *fcn* med argumenter *t* og *y* . Når funksjonen blir kalt, returnerer den med de deriverte i vektoren *dydt*. Navnene *dydt, fcn , t* og *y* kan velges fritt. Merk at vi har spesifisert *dydt* som en kolonne-vektor. (Bruk *help ode45* i Matlab). Denne funksjonen legges i en egen m-fil. Anbefaler å bruke samme navn på fila og funksjonen slik at navnet på fila i dette tilfellet blir *fcn.m*.
La oss så skrive første versjon av programmet. Bestemmer oss for å plotte z og v som funksjon av t opptil t = 70s.

```
% Program versjon1
% Bruker ode45
clear
y0 = [8 ; 0]; % Startverdier
tintervall = [ 0  70.0];
[t,y] = ode45('fcn',tintervall,y0);

% Plotter z og v som funksjon av t
plot(t,y(:,1),t,y(:,2),'-.');
```

Når programmet uføres, får vi følgende plott :



La oss se nærmere på programmet. Anbefaler å starte all programmer med kommandoen *clear*. Denne nullstiller alle variable (Unntaket er variable definert med kommandoen *global*. Disse må nullstilles fra kommando-vinduet.). Deretter setter vi startverdiene i kolonne-vektoren *y0*. (Alternativ: *y0* = [8 0]' som transponerer linjevektoren til en kolonnevektor). Tidsintervallet angis i linjevektoren *tintervall*. Det første tallet angir startverdien og det siste sluttverdien. Deretter kalles **ode45**. Strengen `'fcn'` angir egentlig navnet på m-fila og ikke funksjonen. Derfor er det om tidligere nevnt lurt å bruke samme navn på fil og funksjon. (Fra versjon 6.0 av Matlab, anbefales `@fcn` som nevnt tidligere). Resultatet av kallet er en vektor *t* og en matrise *y*. Vektoren *t* inneholder alle tidene som **ode45** har funnet løsning for. Lengden kan finnes ved å bruke kommandoen `length(t)` eventuellt `size(t).` I dette tilfellet finner vi `length(t)` = 117. (Se kommentar angående parameteren `Refine` i lefsa *ODE*). Første kolonne av matrisa *y* inneholder *z*-verdiene (antall 117) og den andre kolonna *v*-verdiene. (De deriverte av første kolonne). Vi er så klar til å plotte. Når vi skriver *t, y(:,1)*, plottes *t* langs den horisontale aksen og *z* langs den vertikale. Merk at *y(:,1)* betyr 1. kolonne der kolonet henviser til alle elementene. Deretter plotter vi hastigheten *v* i samme figuren ved å skrive *t, y(:,2)* der 2

henviser til den andre kolonnen som inneholder hastighetene. Tilslutt har vi lagt til strengen '–.' Dette har vi gjort for at den siste kurven skal tegnes med strekpunktert linje istedenfor heltrukket. Plotting er behandlet mer detaljert senere. (Skriv `help plot` for flere muligheter)

Versjon 2

Vi skal nå i tillegg skrive tabell for z og v. Tabellen skal gå til 13s da vi ser av plottet at dette tilsvarer noenlunde maks. negativ z-amplityde ($v \approx 0$)

```
% Program versjon2
% Dette er versjon1 med tillegg av tabeller
% Bruker ode45
clear
y0 = [8; 0]; % Startverdier
tintervall = [0 70.0];
[t,y] = ode45('fcn',tintervall,y0);

% Plotter z og v = dz/dt som funksjon av t
plot(t,y(:,1),t,y(:,2),'-.');

% Tabeller for z og v opptil 13s
options = odeset('RelTol',1.0e-5);
y0 = [6; 0]; % Startverdier
tintervall = [0 : 1 : 13];
[t,y] = ode45('fcn',tintervall,y0,options);
fprintf(' %5.1f  %13.4e  %13.4e \n',[t y]');
```

```
   0.0    8.0000e+000    0.0000e+000
   1.0    7.7234e+000   -5.4642e-001
   2.0    6.9327e+000   -1.0171e+000
   3.0    5.7320e+000   -1.3607e+000
   4.0    4.2598e+000   -1.5597e+000
   5.0    2.6582e+000   -1.6226e+000
   6.0    1.0526e+000   -1.5717e+000
   7.0   -4.5573e-001   -1.4324e+000
   8.0   -1.7903e+000   -1.2276e+000
   9.0   -2.8955e+000   -9.7630e-001
  10.0   -3.7324e+000   -6.9318e-001
  11.0   -4.2752e+000   -3.8998e-001
  12.0   -4.5089e+000   -7.6313e-002
  13.0   -4.4276e+000    2.3733e-001
```

La oss se nærmere på siste delen av dette programmet. Vi har nå tatt med en linje *options = odeset('RelTol',1.0e-5). RelTol* er den relative nøyaktigheten vi ønsker. I plottedelen av programmet satte vi ikke *RelTol*. I **ode45** settes da *RelTol* = 1.0e-3. Dette er ofte godt nok for plotting. (Se lefsa *ODE* for flere detaljer). Vi ønsker utskrift for hvert sekund. Derfor skriver vi *tintervall* = [0 : 1: 13] (eventuellt [0 : 13]) som er en linjevektor med 14 elementer: 0, 1,2,…13. Merk at *t*-vektoren nå bare består av disse 14 verdiene. (Under beregningen brukes om nødvendig flere). I kallet av **ode45** legges parameteren *options* etter startvektoren *y0*. For å få pen utskrift, bruker vi format-kommandoer. (Flere detaljer under avsnittet *Utskrift av tabellert* i lefsa). Konstruksjonen [*t y*] gir en matrise med tre kolonner der første kolonne er *t*-vektoren osv. Når vi bruker formatkommandoer, skrives en matrise ut kolonne for kolonne. Det vi ønsker er å skrive ut matrisa linje for linje. For å få til dette, må vi transponere matrisa ved å skrive [*t y*]' . Nå blir kolonnene linjer og linjene kolonner.

UTSKRIFT AV TABELLER

Når vi skriver til skjerm , papir eller en annen fil, bruker vi kommandoene **disp**, **fprintf** og **sprintf**. **disp** og **fprintf** sørger for **utskrift** mens **sprintf** lager en utskriftstreng som så kan skrives ut med **fprintf** og **disp**.

Syntaks :

disp(streng og/eller variable)
sprintf (<Streng med format specifikasjoner>, <Liste av variable>)

Syntaksen for **fprintf** er som for **sprintf**, men i tillegg kan det spesifiseres at det skal skrives til en fil.

Et format ser ut som følger : **% mW.PF**
Tegnet % angir starten av et format og må alltid være med. **m** er en markør som kan ha følgende verdier (kan utelates):

minus tegn (–)	Venstrejusterer tallet. Ellers høyrejustert
+	Skriv alltid tallets fortegn (pluss eller minus)
0	Utfyll tallet med ledende nuller isteden for mellomrom

W angir antall plasser som er avsatt til tallet. (Kan utelates). **P** angir antall desimaler etter desimalpunktum (presisjonen). (Kan utelates).
F er formatbeskriver.(Må være med). En liste over verdier for **F** er gitt nedenfor

F	Beskrivelse
c	Et enkelt tegn
d	Heltall med fortegn
e	Eksponentform (liten e)
E	Eksponentform (stor E)
f	Desimaltall
g	Kompakt versjon av f eller g
G	Som ovenfor, men med stor G
o	Oktalt tall (uten fortegn)
s	Streng av tegn
u	Heltall uten fortegn
x	Heksadesimalt tall(småbokstaver)
X	Som ovenfor , men med store bokstaver

La oss se på et eksempel der vi skriver en tabell over feilfunksjonen `erf(x)` og den komplementære feilfunksjonen `erfc(x) = 1 - erf(x)` for $0 \le x \le 2$

```
x = [0 : 0.25 : 2.0]';
y1 = erf(x);
y2 = erfc(x);
disp('      x       erf(x)     erfc(x)')
disp('    ------------------------------')
disp([x y1 y2]);
```

som gir utskrift :

```
       x       erf(x)     erfc(x)
    ------------------------------
         0          0    1.0000
    0.2500     0.2763    0.7237
    0.5000     0.5205    0.4795
    0.7500     0.7112    0.2888
    1.0000     0.8427    0.1573
    1.2500     0.9229    0.0771
    1.5000     0.9661    0.0339
    1.7500     0.9867    0.0133
    2.0000     0.9953    0.0047
```

Legg merke til at `x` er generert som en kolonnevektor slik at også `y1` og `y2` blir kolonnevektorer. Deretter bruker vi **disp** til å lage tabell-overskrift. Ber deretter om utskrift av matrisa `[x y1 y2]`. Merk at matrisa skrives ut linje for linje slik som vi ønsker. Tallene i tabellen blir utskrevet med den valgte versjonen av kommandoen `format.` I dette tilfellet er dette `short` som gir 4 desimaler. Dersom vi hadde valgt `format long` , ville vi fått utskrift med 14 siffer for alle kolonnene. Vi ser derfor at vi ikke kan velge forskjellig format for de enkelte kolonnene med ensidig bruk av **disp.** Kombinerer derfor **disp** med **sprintf**.

```
x = [0 : 0.25 : 2.0]';
y1 = erf(x);
y2 = erfc(x);
disp('      x       erf(x)       erfc(x)')
disp('    -------------------------------');
disp(sprintf('   %5.2f     %7.4f   %12.4e \n',[x y1 y2]'));
```

```
      x       erf(x)       erfc(x)
    -------------------------------
    0.00      0.0000    1.0000e+000
    0.25      0.2763    7.2367e-001
    0.50      0.5205    4.7950e-001
    0.75      0.7112    2.8884e-001
    1.00      0.8427    1.5730e-001
    1.25      0.9229    7.7100e-002
    1.50      0.9661    3.3895e-002
    1.75      0.9867    1.3328e-002
    2.00      0.9953    4.6777e-003
```

I **sprintf** har vi nå skrevet en streng som inneholder formatteringskoder. Disse starter som vi vet med %. Strengen avsluttes med `\n` som betyr ny linje. Hadde vi utelatt `\n` , ville hele matrisa blitt skrevet på en linje. Legg merke til at vi har transponert matrisa `[x y1 y2]`. Når vi bruker formatteringskoder, skrives en matrise ut kolonne for kolonne; ikke linje for linje som vi ønsker. Derfor må matrisa transponeres. Dette er ikke nødvendig når vi bare bruker **disp**, som i det forrige eksemplet.

I stedenfor **disp** og **sprintf**, kan vi her greie oss med **fprintf** alene som vist nedenfor.

```
x = [0 : 0.25 : 2.0]';
y1 = erf(x);
y2 = erfc(x);
fprintf('      x       erf(x)       erfc(x)\n');
fprintf('    ------------------------------\n');
fprintf('   %5.2f     %7.4f   %12.4e \n',[x y1 y2]');
```

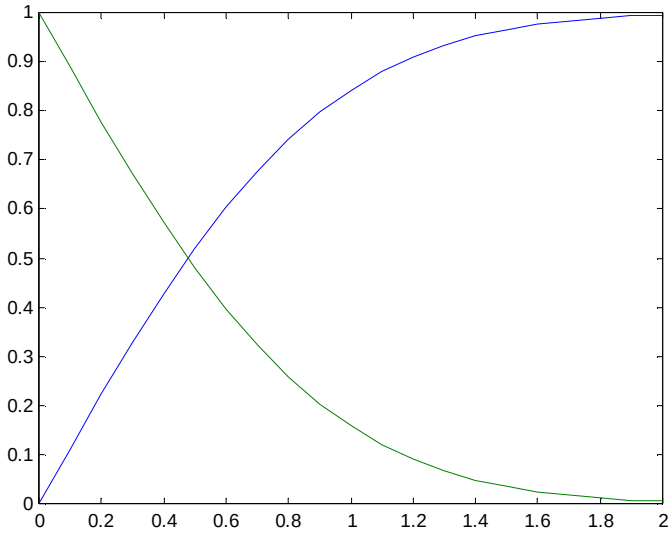
```
      x       erf(x)       erfc(x)
    ------------------------------
    0.00      0.0000    1.0000e+000
    0.25      0.2763    7.2367e-001
    0.50      0.5205    4.7950e-001
    0.75      0.7112    2.8884e-001
    1.00      0.8427    1.5730e-001
    1.25      0.9229    7.7100e-002
    1.50      0.9661    3.3895e-002
    1.75      0.9867    1.3328e-002
    2.00      0.9953    4.6777e-003
```

sprintf er nyttig når formatteringsstrengene blir lange samt i forbindelse med tekst på plott.

PLOTTING

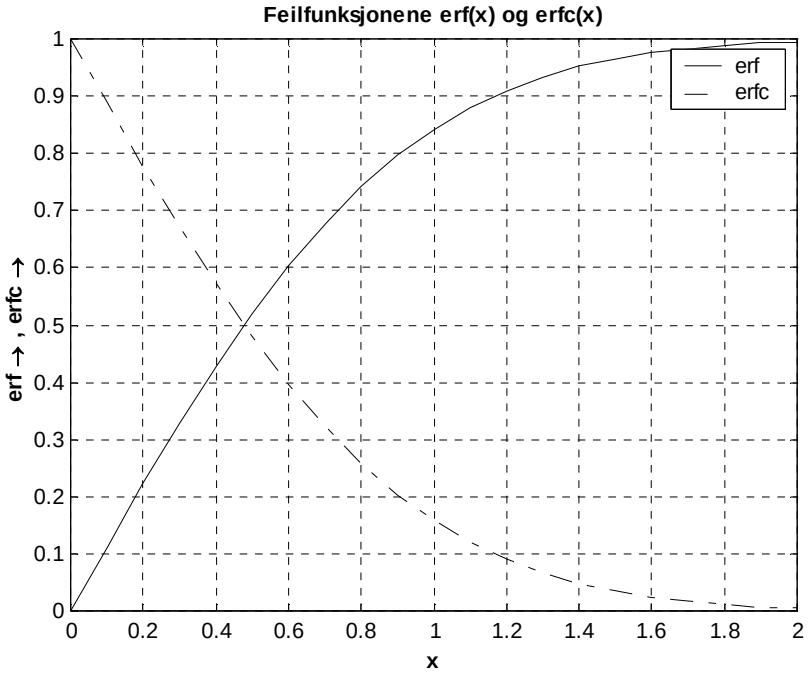
La oss plotte feilfunksjonene i det forrige eksemplet.

```
» x = 0 : 0.1 : 2;
» y1 = erf(x);
» y2 = erfc(x);
» plot (x,y1,x,y2);
```



Den første vektoren plottes langs x-aksen mens den andre plottes langs y-aksen. I dette tilfellet har vi plottet to funksjoner i samme plottet. Dette kan vi gjøre fordi begge har samme utstrekning langs både x- og y-aksen. I praksis har vi bruk for å fortelle hva vi har plottet, samt å skille de to funksjonene fra hverandre. Dette gjøres med `title` og `legend` samt plottesymboler.

```
1    % Program plotting
2    x = 0 : 0.1 :2;
3    y1 = erf(x);
4    y2 = erfc(x);
5    plot(x,y1,'k',x,y2,'k-.');
6    grid on
7    title('Feilfunksjonene erf(x) og erfc(x)','Fontweight','Bold')
8    xlabel('x','Fontweight','Bold')
9    ylabel('erf \rightarrow , erfc\rightarrow','Fontweight','Bold')
10   legend('erf','erfc')
```



Forklaring til det meste finnes ved bruk av `help plot`, men noen kommentarer kan være nyttige. I linje 5 har vi brukt `'k'` og `'k-.'`. `'k'` brukes for å angi at plottefargen skal være svart. Uten denne vil plottefargen være blå for den første kurva og grønn for den andre. (Spesielt den grønne vises dårlig på en gråtone - printer). For den andre kurva bruker vi `'k-.'` der - . angir strekpunktert linje. I line 6 har vi spesifisert nett. Vi setter overskrifta i linje 7 og angir uthevet skrift. I linje 8 og 9 setter vi tekst på aksene. Legg merke til `\rightarrow` som ganske riktig blir en høyrepil på figuren. Det finnes en mengde symboler, deriblant de greske bokstavene, som kan brukes på figuren. En liste over disse er

gitt i appendiks . I linje 10 setter vi tegnforklaring til kurvene (legend) der `erf(x)` har heltrukket linje og `erfc(x)` har en strekpunktert. Denne boksen kan flyttes rundt på figuren med musa. Dersom du har bruk for å sette annen tekst på figuren, prøv `help text` og `help gtext`.
Ovenfor har vi tegnet begge kurvene samtidig ved å plotte dem i den samme plot-setningen. Samme effekten kan vi få ved følgende sekvens :

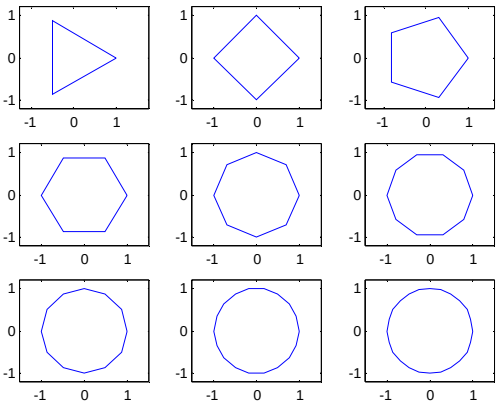
```
plot(x,y1)
hold on
plot(x,y2)
hold off
```

Vi bruker denne framgangsmåten dersom det skal plottes mange kurver på samme figur , f. eks. i en iterasjonsprosess.
Vi kan også dele plottevinduet opp i delplot ved å bruke kommandoen **subplot**. Eksempel : `subplot(3,3,k)` betyr at plottevinduet skal deles opp i en 3×3 matrise av delvindu og at det neste plottet skal plasseres i delvindu `k`.
Nummereringen av delplottene er som følger :

[1] [2] [3]
[4] [5] [6]
[7] [8] [9]

Programmet nedenfor , hentet fra Van Loan[5], viser dette.

```
% Script File: Polygons
% Plots selected regular polygons.
close all
theta = linspace(0,2*pi,361);
c = cos(theta);
s = sin(theta);
k=0;
for sides = [3 4 5 6 8 10 12 18 24]
   stride = 360/sides;
   k=k+1;
   subplot(3,3,k)
   plot(c(1:stride:361),s(1:stride:361))
   axis([-1.2 1.2 -1.2 1.2])
   axis equal
end
```



BANDMATRISER

TRIDIAGONALE MATRISER

Bruker ligning (2.3.16) i kompendiet som eksempel.

$$y''(x) + 2xy'(x) = 0 \qquad (1)$$

med randbetingelser : $y(0) = 0\ ,\ y(x_\infty) = 1$

(Mer nøyaktig randbetingelse : $y \to 1$ for $x \to \infty$).
Med den mer nøyaktige randbetingelsen har (1) løsningen $y(x) = \mathrm{erf}(x)$.

Diskretisering

Med henvisning til figuren, får vi :

$$x_i = h \cdot i\ ,\ i = 0,1,\ldots,N+1 \text{ der } h = \frac{x_\infty}{N+1} \qquad (2)$$

Diskretisering med sentraldifferanser vil gi en ligningsystem med tridiagonal koeffisientmatrise :

$$\begin{bmatrix} b_1 & c_1 & & & & & \\ a_2 & b_2 & c_2 & & & & \\ & \cdot & \cdot & \cdot & & & \\ & & \cdot & \cdot & \cdot & & \\ & & & \cdot & \cdot & \cdot & \\ & & & & a_{N-1} & b_{N-1} & c_{N-1} \\ & & & & & a_N & b_N \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ x_{N-1} \\ x_N \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \cdot \\ \cdot \\ \cdot \\ d_{N-1} \\ d_N \end{bmatrix} \qquad (3)$$

Ved bruk av sentraldifferanser i (1) samt bruk av (2) :

$$\frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} + 2h \cdot i \cdot \left(\frac{y_{i+1} - y_{i-1}}{2h} \right) = 0$$

som ordnet gir :

$$(1-h^2\cdot i)y_{i-1} - 2y_i + (1+h^2\cdot i)y_{i+1} = 0 \qquad (4)$$

Skriver ut (4) for $i = 1$:

$$-2y_1 + (1+h^2)y_2 = 0 \text{ da } y_0 = 0 \text{ fra (1b)}$$

Skriver ut (4) for $i = N$:

$$(1-h^2N)y_{n-1}) - 2y_N = -(1+h^2N) \text{ da } y_{N+1} = 1 \text{ fra (1b)}$$

Totalt har vi da fått disse uttrykkene for diagonalene :

$$\begin{aligned} &b_i = -2\ ,\ i = 1,2,\ldots,N \quad ,\quad a_i = 1-h^2\cdot i\ ,\ i = 2,3,..,N \\ &c_i = 1+h^2\cdot i\ ,\ i = 1,2,..,N-1 \\ &d_i = 0\ ,\ i = 1,2,..,N-1\ ,\ d_N = -(1+h^2N) \end{aligned} \qquad (5)$$

Matlab-programmet nedenfor viser fremgangsmåten der vi bruker subrutina **tdma** til å løse ligningsystemet i (5).

```
%============= Program triv1 ==============
clear
xmax = 4.Ø;
h = Ø.Ø5; % skrittlengde
ni = round(xmax/h) ; %Antall intervall
xmax = ni*h; % For sikkerhets skyld
neq = ni - 1 ; % Antall ligninger
%--- Initialisering(kolonnevektorer)
b = -2*ones(neq,1);
c = zeros(neq,1);
d = c; a = c ;
% --- Genererer over-og under-diagonaler
for k = 1:neq
   fac = k*h^2;
   a(k) = 1 - fac;
   c(k) = 1 + fac;
end
d(neq) = -(1 + fac);
%-----------------------
% Løser ligningsystemet
%-----------------------
y = tdma(a,b,c,d);
y = [Ø;y ; 1];
%---- Utskrift av y ----
x = [Ø :h: xmax]'; % Som kolonnevektor
ya = erf(x); % Analytisk løsning
fprintf('\n      x         y         ya  \n\n')
fprintf('  %7.3f %1Ø.5f  %1Ø.5f \n',[x y ya]');
```

Vektorisering

Matlab er laget for å operere raskt på vektorer uten bruk av løkker. For eksempel er operasjonen `y = x` raskere enn

```
for k = 1: n
  y(k) = x(k);
end
```

Denne teknikken kan utvides til bruk av indeksvektorer. Eksemplet ovenfor kan også skrives :

```
k = 1:n; % Indeksvektor
y(k) = x(k);
```

Indeksvektoren `k` går gjennom alle verdiene fra 1 til `n` slik at effekten blir som i tilfellet `y = x` . Vi kan også bruke indeksvektoren på deler av vektoren. Merk at indeksvektoren `k` også krever lagerplass etter at operasjonen er fullført. Går vi tilbake til løkka i **triv1**, kan vi generere diagonalen `a` og `c` ved å bruke en indeksvektor :

```
k = 1: neq;  % Indeksvektor
a(k) = 1 - k*h^2;
c(k) = 1 + k*h^2;
```

Denne versjonen er ca. 25 ganger raskere enn løkka. Generelt er bruk av indeksvektor mye mer effektivt enn bruk av løkke. Når vi derimot ser på den virkelige eksekveringstiden, føles kanskje ikke effekten så stor. Med `neq = 40000` og en PC med 266MHz prosessor, bruker løkka 2.14s mens bruk av indeksvektor krever 0.09s. Indeksvektoren trenger i dette tilfellet 312.5 KB lagerplass.

GLISNE MATRISER I MATLAB

Med glisne matriser (eng: sparse matrices) mener vi matriser som har mange null-elementer i forhold til totalt antall elementer. Bandmatriser er typiske. Matlab har mange kommandoer for behandling av glisne matriser (bruk *help sparse*), men vi skal begrense oss til kommandoen *spdiags* som bygger matriser av diagonaler.
La oss se på et eksempel.

```
a = [1:5]';
A = spdiags([a a a],[-1 0 1], 5,5);
A = full(A)
A =
     1     2     0     0     0
     1     2     3     0     0
     0     2     3     4     0
     0     0     3     4     5
     0     0     0     4     5
```

Vi lager først en kolonne-vektor `a` med elementene 1 til 5. Denne vektoren bruker vi til å lage en tridiagonal matrise `[a a a]`. *spdiags* nummererer diagonalene på følgende måte: Hoveddiagonalen er 0, første overdiagonal er 1, neste overdiagonal 2 osv. Første underdiagonal er -1, neste -2 osv. For en tridiagonal matrise med en underdiagonal og en overdiagonal sammen med hoveddiagonalen, markeres dette ved `[-1 Ø 1]`. Tilslutt gis størrelsen på den totale matrisa; i vårt tilfelle 5×5. Matrisa `A` blir lagra uten nullelementene. For å kontrollere hva som lagres, går vi tilbake til en full matrise med kommandoen `A = full(A)`. (Bare ved små matriser, ellers forsvinner vitsen ved å bruke glisne matriser. Bruk kommandoen `spy(A)` når du bare vil kontrollere lagringsmønsteret). Legg merke til hvordan diagonalene lagres: Elementer med samme indeks blir lagret i samme *kolonne*, ikke samme *linje* som vi bruker; se lign. (3). For å gjøre dette helt klart, viser vi et eksempel med 5 ligninger:

Vår notasjon:

$$\begin{bmatrix} b_1 & c_1 & 0 & 0 & 0 \\ a_2 & b_2 & c_2 & 0 & 0 \\ 0 & a_3 & b_3 & c_3 & 0 \\ 0 & 0 & a_4 & b_4 & c_4 \\ 0 & 0 & 0 & a_5 & b_5 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ d_5 \\ d_5 \end{bmatrix} \qquad (6)$$

Lagring ved *spdiags*:

$$\begin{bmatrix} b_1 & C_2 & 0 & 0 & 0 \\ A_1 & b_2 & C_3 & 0 & 0 \\ 0 & A_2 & b_3 & C_4 & 0 \\ 0 & 0 & A_3 & b_4 & C_5 \\ 0 & 0 & 0 & A_4 & b_5 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ d_5 \\ d_5 \end{bmatrix} \qquad (7)$$

Vi har brukt store bokstaver i (7) for å markere de elementene som har forskjellig indeksering fra vår notasjon. Et program **triv3**, basert på bruk av *spdiags* og indeksvektoriserng er vist nedenfor.

```
%======== Program triv3 Sparse-versjon =======
clear
xmax = 4.Ø;
h = Ø.Ø5; % skrittlengde
ni = round(xmax/h) ; %Antall intervall
xmax = ni*h; % For sikkerhets skyld
neq = ni - 1 ; % Antall ligninger
%--- Initialisering(kolonnevektorer)
b = -2*ones(neq,1);
c = zeros(neq,1); % overdiagonal
d = c; a = c ;
% Lager indeksvektor
J = 2:neq;
```

```
% --- Genererer over-og under-diagonaler
a(J-1) = 1 - J*h^2;
c(J) = 1 + (J-1)*h^2;
d(neq) = -(1 + neq*h^2);
%-----------------------
% Løser ligningsystemet
%-----------------------
A = spdiags([a b c], [-1 Ø 1],neq,neq);
y = A\d;
y = [Ø;y ; 1];
%---- Utskrift av y ----
x = [Ø :h: xmax]'; %Som kolonnevektor
ya = erf(x); % Analytisk løsning
fprintf('\n      x          y         ya  \n\n')
fprintf('  %7.3f %1Ø.5f  %1Ø.5f \n',[x y ya]');
```

Vi har her lagret diagonalene direkte, tilsvarende lign. (7). Men kommandoen *sparse* nummererer også den tridiagonale matrise med standard matrise-notasjon, slik at for et system med fem ligninger kan (3) også skrives :

$$\begin{bmatrix} a_{11} & a_{12} & & & \\ a_{21} & a_{22} & a_{23} & & \\ & a_{32} & a_{33} & a_{34} & \\ & & a_{43} & a_{44} & a_{45} \\ & & & a_{54} & a_{55} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \end{bmatrix}$$

La oss vise dette :

```
a = (1:5)';
A = spdiags([a a a],[-1 Ø 1],5,5)

A =
   (1,1)        1
   (2,1)        1
   (1,2)        2
   (2,2)        2
   (3,2)        2
   (2,3)        3
   (3,3)        3
   (4,3)        3
   (3,4)        4
   (4,4)        4
   (5,4)        4
   (4,5)        5
   (5,5)        5
```

Legg merke til at null-elementene ikke lagres. Lign. (5) blir nå :

$$
\begin{aligned}
&b_i \rightarrow a_{i,i} = -2 \ , \ i = 1,2,\ldots,N \ \ , \\
&a_{i+1} \rightarrow a_{i+1,i} = 1 - h^2 \cdot (i+1) \ , \ i = 1,2,..,N-1 \\
&c_i \rightarrow a_{i,i+1} = 1 + h^2 \cdot i \ , \ i = 1,2,..,N-1 \\
&d_i = 0 \ , \ i = 1,2,..,N-1 \ , \ d_N = -(1 + h^2 N)
\end{aligned}
\tag{8}
$$

Merk at vi har brukt $i = 1,2,..,N\text{-}1$ også for `a`-vektoren slik at vi kan beregne `a` og `c` i samme løkke.

Programmet blir som vist nedenfor :

```
%======== Program triv4 Sparse-versjon =======
clear
xmax = 4.0;
h = 0.1; % skrittlengde
ni = round(xmax/h) ; %Antall intervall
xmax = ni*h; % For sikkerhets skyld
neq = ni - 1 ; % Antall ligninger
%--- Initialisering. Lagrer temporært b i d og
%--- bruker b-vektoren til å lage A
d = -2*ones(neq,1); % Hoveddiagonal
A = spdiags([d d d], [-1 0 1],neq,neq);
% --- Genererer over-og under-diagonaler
for k = 1:neq -1
   A(k+1,k) = 1 - (k+1)*h^2; % Underdiagonal
   A(k,k+1) = 1 + k*h^2;  % Overdiagonal
end
d(neq) = -(1 + neq*h^2);

y = A\d; %Løser ligningsystemet

y = [0;y ; 1];
%---- Utskrift av y ----
x = [0 :h: xmax]'; %Som kolonnevektor
ya = erf(x); % Analytisk løsning
fprintf('\n      x         y        ya  \n\n')
fprintf('  %7.3f %10.5f  %10.5f \n',[x y ya]');
```

Legg merke til at vi ikke kan bruke indeksvektor i dette tilfellet. Vi har brukt bare en vektor (høyresiden av systemet) til å generere matrisa slik at vi sparer lagerplass ved å bruke denne versjonen.

Det ikke noen fordel å bruke *spdiags* og `A\d` sammenlignet med å bruke **tdma** for løsning av et tridiagonalt ligningsystem. **tdma** er spesialskrevet for å behandle tridiagonale system mens *sparse* behandler generelle glisne system. Antall operasjoner i **tdma** er 2(`neq` -1) ≈ 2`neq` slik at tidsforbruket er proposjonalt med antall ligninger. Med en PC med 266Mhz prosessor, løser **tdma** et ligningsystem med 4000 ukjente på 0.45s og bruker følgelig 4.5s på et system med 40000. Ved å bruke *sparse*, trengs 8s for 4000 ligninger og hele 1457s (24 min og 17s) for et system på 40000. Regnetiden her er proposjonal med mer enn kvadratet av antall ligninger. Husk at **tdma** ikke utfører pivotering, noe som kan skape problemer dersom systemet ikke er diagonal-dominert; se (lign. (3.1.4) i kompendiet. I dette tilfellet kan **tripiv** brukes istedet. Regnetiden øker nå knapt 3 ganger (2.75) i forhold til **tdma**, men økningen er fremdeles proposjonal med antall ligninger. Spesialskrevne løsere for bandmatriser er normalt betydelig raskere enn de generelle løserne som er innbyggget i Matlab.

INNLESNING AV DATA (`help iofun`)

Den vanligste kommandoen er `input` som leser interaktivt .
Eksempel :

```
x = input('les x-verdi : ');
```

Skriver `les x-verdi :` på skjermen og venter på input fra brukeren.

Lese tabeller fra filer (`help load`**)**

Anta at vi har følgende tabell liggende på en fil med navn *func.dat*

```
0.00      0.0000
0.25      0.2763
0.50      0.5205
0.75      0.7112
1.00      0.8427
1.25      0.9229
1.50      0.9661
1.75      0.9867
2.00      0.9953
```

Følgende kommando leser tabellen inn i en 9×2 matrise :
`load func.dat`
Merk at matrisa får navnet `func`. Dersom fila `func` ikke har noen endelse (extension) og du skriver`load func` antar Matlab at dette er en mat-fil som er en fil på binær form. Du må da skrive`load func -ascii` for å få rett format.

APPENDIKS

Math Symbols

$\neq$	\neq	$\leftarrow$	\leftarrow	$\in$	\in
$\geq$	\geq	$\rightarrow$	\rightarrow	$\subset$	\subset
$\approx$	\approx	$\uparrow$	\uparrow	$\cup$	\cup
$\equiv$	\equiv	$\downarrow$	\downarrow	$\cap$	\cap
$\cong$	\cong	$\Leftarrow$	\Leftarrow	$\perp$	\perp
$\pm$	\pm	$\Rightarrow$	\Rightarrow	∞	\infty
∇	\nabla	$\Leftrightarrow$	\Leftrightarrow	$\int$	\int
$\angle$	\angle	∂	\partial	$\times$	\times

Greek Symbols

α	\alpha	ω	\omega	Σ	\Sigma
β	\beta	ϕ	\phi	Π	\Pi
γ	\gamma	π	\pi	Λ	\Lambda
δ	\delta	χ	\chi	Ω	\Omega
ε	\epsilon	ψ	\psi	Γ	\Gamma
κ	\kappa	ρ	\rho		
λ	\lambda	σ	\sigma		
μ	\mu	τ	\tau		
ν	\nu	υ	\upsilon		