

Lathund för Matlab

Denna lathund, omfattande fyra sidor, är tillåten att använda vid tentamen i
"Programmering med Matlab", MVG300, MVGF30.

Aritmetiska operatörer

+ - * / \ ^
.* ./ .\ .^

Relationsoperatörer

< mindre än
> större än
<= mindre än lika med
>= större än lika med
== lika med
~= skilt från

Prioritetsordningen för **logiska operatör**:

~	negation	högsta
&	och	
	eller	↓
&&	lat variant	
	lat variant	lägsta

Vanliga datastrukturer

```
vektor(index)
matris(rad, kolonn)
cell_vektor{index}
cell_matris{rad, kolonn}
post.medlems_namn
post_vektor(index).medlems_namn
sträng = 'sekvens av tecken'
[], {} tom matris (den andra cell)
```

Index kan vara positiva heltal, heltalsvektorer/
matriser, samt logiska vektorer/matriser.

Kolon, :, står för alla.

Kan skapa större vektorer/matriser från delar.

Använd då [], semikolon, ;, för radbyte och
komma eller blank för kolonnbyte.

`matris(:)` ger en kolonnvektor med kolonnerna
staplade i följd. `vektor(:)` ger en kolonnvektor.

' markerar transponat.

`end` ger största index i indexuttryck.

```
index = find(logisk_vektor)
[rader, kolonner] = find(logisk_matris)
```

Repetitionssatser

```
for var = matris
    satser
end
```

var antar `matris(:, 1)`, `matris(:, 2)`, etc.
i loopen. Specialfall när `matris` är en radvektor:

```
matris = start:steg:stopp
matris = start:stopp        % om steg = 1
```

```
while logiskt_uttryck
    satser
end
```

`break` hoppar ur den innersta loopen.

`continue` fortsätter med nästa iteration
(i den innersta loopen).

`return` hoppar tillbaka till anropande funktion.

Kontrollsatser

```
if logiskt_uttryck
    satser
elseif logiskt_uttryck
    satser
else
    satser
end
```

Man kan ha flera `elseif`-delar. Här ett specialfall:

```
if logiskt_uttryck
    satser
end

switch switch_uttryck
    case case_uttryck
        satser
    case case_uttryck
        osv.
    otherwise
        satser
end
```

`case_uttryck` kan vara ett tal eller en sträng eller en
cellvektor av tal eller strängar.

Funktioner

Här ett exempel med två in- och utparametrar.

```
[ut1, ut2] = funk_namn(in1, in2)
ut1 = ...
ut2 = ...
```

Exempel på anonym funktion:

```
funk_handtag = @(x,y) x+sin(x*y)
```

Exempel på funktion som parameter:

```
funk(@(x,y) x+sin(x*y), osv. )  
funk(funk_handtag, osv. )  
funk1(@funk2, osv. )
```

Globala variabler:

```
global var1 var2 var3 etc.
```

Deklarera både i anropad och anropande funktion.

Några vanliga funktioner

På följande sidor betecknar v , u reella rad- eller kolonnvektorer, och M är en reell matris. s är en sträng. För de funktioner som gör skillnad mellan vektor- och matrisargument har jag visat båda fallen. När det inte är någon skillnad visas bara matrisfallet.

Om en funktion tar en rad/kolonn som argument och ger en resultatvektor, har resultatet samma orientering som indatavektorn.

-> rad betyder att resultatet hamnar i en radvektor och -> kolonn att resultatet hamnar i en kolonn.

Funktioner för att skapa matriser

```
zeros(m, n) nollmatris  
ones(m, n) matris av ettor  
eye(m, n) enhetsmatris  
rand(m, n) likf. slumpstal i [0, 1)  
randn(m, n) normalförd. slumpstal
```

Om $m = n$ anropa med `zeros(m)` etc.

`diag(M, k)` ger diagonal k som en kolonnvektor. $k = 0$ är huvuddiagonal, $k > 0$ ger super- och $k < 0$ subdiagonaler.

`diag(M)` ger huvuddiagonalen. `diag(v)` ger en diagonalmatris och `diag(v, k)` ger en matris med v som diagonal nummer k .

Funktionerna sum, prod, min, max, sort

```
sum(v) summan av elementen  
sum(M) kolonnsummor -> rad  
sum(M, 2) radsummor -> kolonn
```

Analogt för prod, produkten.

```
min(v) minimum  
min(M) kolonnminima -> rad  
min(M, [], 2) radminima -> kolonn  
[u, ind] = min(v) min och index  
[u, ind] = min(M) min och index  
[u, ind] = min(M, [], 2) min och index
```

Analogt för max. `ind` betecknar ett index eller en indexvektor. `ind` ger index till *första* minvärdet (om det finns kopior).

```
sort(v) sortera i växande ordning  
sort(M) sortera varje kolonn  
sort(M, 2) sortera varje rad
```

I avtagande ordning

```
sort(v, 'descend')  
sort(M, 'descend')  
sort(M, 2, 'descend')
```

Man kan också använda `ind`, här ett ex:

```
>> [u, ind] = sort([3 1 -2 4 -6])  
u = -6 -2 1 3 4  
ind = 5 3 2 1 4
```

Funktioner som tar reda på dimensioner

```
[m, n] = size(M) dimensionerna  
v = size(M) -> [m, n]  
length(M) -> max(size(M))  
numel(M) antalet element i M  
isempty(M) 1 om tom matris,  
0 annars
```

Matematiska funktioner

Följande funktioner tar vektor/matris-argument och arbetar elementvis:

```
cos sin cot tan trigonometriska  
acos asin acot atan inverserna  
cosh sinh coth tanh hyperboliska  
exp log log10  
sqrt abs
```

`abs` beräknar *elementvis* absolutbelopp. Använd `norm` för vektorlängd. Det finns ingen `ln`-funktion, använd `log`.

Linjära ekvationssystem

Antag att M är kvadratisk, då är:

```
inv(M) inversen  
det(M) determinanten  
A \ b lösning av  $Ax = b$ 
```

Mera matrishantering

```
norm(v) vektorlängd  
mean(v) medelvärde  
median(v) median  
std(v) standardavvikelse  
cumsum(v) ackumulerad summa  
cumprod(v) ackumulerad produkt  
diff(v) successiva differenser
```

För matrisargument fungerar `mean`, `median` analogt med `sum` och `cumsum`, `cumprod` analogt med `sort`.

`M = reshape(v, m, n)` skapar $m \times n$ -matrisen `M` med kolonnelementen tagna från `v`.

Det måste gälla att `m * n = numel(M)`.

Reella och komplexa konstanter

<code>Inf</code>	Infinity
<code>-Inf</code>	-Infinity
<code>NaN</code>	Not-a-Number
<code>intmax</code>	ger största heltalet
<code>intmin</code>	ger mesta negativa heltalet
<code>realmax</code>	ger största flyttalet (double)
<code>realmin</code>	minsta positiva normaliserade flyttalet

Exempel på konstanter:

```
-1.23e-45      12.3      2.35e100
2 + 3i         2.3e14 - 56.44e100j
```

Antag att `M` är en *komplex* matris:

<code>real(M)</code>	<code>realdel</code>
<code>imag(M)</code>	<code>imaginärdel</code>
<code>conj(M)</code>	komplexbesläktad
<code>angle(M)</code>	argument (vinkeln)
<code>M.'</code>	reellt transponerat av komplex <code>M</code>

Avrundning och liknande

Följande funktioner avrundar till heltal (på olika vis):

<code>floor(M)</code>	avruta mot minus oändligheten
<code>ceil(M)</code>	avruta mot plus oändligheten
<code>fix(M)</code>	avruta mot noll
<code>round(M)</code>	avruta mot närmaste heltal
<code>sign(M)</code>	-1 om negativt, 1 om positivt, noll om noll

Om `x` och `y` $\neq 0$ är två heltal (eller heltalsfält av samma dimension) så är

<code>mod(x, y)</code>	resten vid divisionen <code>x / y</code>
<code>mod(x, 0)</code>	är <code>x</code>

Några vanliga grafikfunktioner (via exempel):

```
plot(x, y, 'r-')    t.ex.
plot(x, y1, 'r*', x, y2, 'b:')
plot3(x, y, z, 'r--')  rymdkurva
semilogx(x, y)         log i x-led
semilogy(x, y)         log i y-led
loglog(x, y)           log i både x- och y-led
```

Färger:	<code>r</code> <code>g</code> <code>b</code> <code>c</code> <code>m</code> <code>y</code> <code>k</code>
Linjetyper:	<code>-</code> <code>:</code> <code>-.</code> <code>--</code>
Symboler:	<code>.</code> <code>o</code> <code>x</code> <code>+</code> <code>*</code> <code>v</code> <code>^</code> <code><</code> <code>></code>

samt `s`, `d`, `p`, `h` (för `star`, `diamond`, `pentagram`, `hexagram`).

<code>figure</code>	öppna nytt fönster
<code>figure(n)</code>	aktivera fönster nr. <code>n</code>
<code>clf</code>	‘‘rensa’’ aktuellt fönster
<code>xlabel(sträng)</code>	x-rubrik
<code>title(sträng)</code>	rubrik
<code>text(x, y, s)</code>	skriv <code>s</code> med start i <code>(x, y)</code>
<code>axis equal</code>	lika skalade axlar
<code>axis off</code>	stäng av axlarna
<code>grid on</code>	slå på grid
<code>fill(x, y, 'r')</code>	rita röd polygon
<code>hold on</code>	håll kvar plot
<code>hold off</code>	släpp plot

```
linspace(start, stopp, antal) -> rad
linspace(start, stopp), om antal=100
```

```
[X, Y] = meshgrid(x, y) skapa grid
mesh(X, Y, Z)    rita z = f(x, y)
surf(X, Y, Z)    rita z = f(x, y)
```

In- och utmatning

```
format    utskriftsformat för tal
disp(var) utskrift utan ans
tal = input(prompt) läser ett tal
sträng = input(prompt, 's') läser en sträng
           prompt är en frågesträng
clear      tag bort alla variabler
clear var  tag bort variabeln var
```

Filhantering

```
save fil var1 var2 spara variabler på fil
load fil          ladda variabler från fil
```

Låt strängvariabeln `fil` innehålla ett filnamn.

Då öppnar `fid = fopen(fil, 'r')` filen för läsning och `fid = fopen(fil, 'w')` för skrivning. Om `fid = -1` så gick filen inte att öppna.

`fclose(fid)` stänger filen.

`str = fgetl(fid)` läser in en rad från filen. Radens lagras i teckenvektorn `str`. Vid filslut är `str` talet (inte strängen) `-1`.

`tal = fscanf(fid, '%e', 1)` läser in ett tal. Om det inte fanns något tal att läsa (man har nått filslut) sätts `tal` till `[]`.

`tal = fscanf(fid, '%e')` eller

`[tal, n] = fscanf(fid, '%e')` läser alla, `n`, talen i filen och lagrar dessa i kolonnvektorn `tal`, som blir `n` element lång.

`fprintf(fid, '%s\n', str);` skriver innehållet i strängvariabeln `str` (följt av ny rad, newline, `\n`).
`fprintf(fid, '%e\n', M);` skriver `M(:)` ett tal per rad, med "engineering"-format (decimalpunkt och exponentdel). Byter man `%e` till `%f` (för fixed) får man ingen exponentdel. `%d` används för heltal. `%10d` get fältvidden tio. `%20.5e` och `%20.5f` ger fältvidd 20 varav 5 decimaler.

Listning av filnamn

Om strängvariabeln `directory` innehåller namnet på en katalog så ger `d = dir(directory)` en `m x 1`-vektor av poster (structs) med datamedlemmarna:

```
name    filnamn
date    modifieringsdatum
bytes   antalet bytes i filen
isdir   1 om filen är en katalogfil,
        0 annars
datenum date med ett annat format
```

Wildcards (*) är tillåtna:

```
d = dir('../data/*.txt');
```

returnerar information om alla filer, vars namn slutar på `.txt`, och som ligger i katalogen `../data`.

Typkonverteringsfunktioner

`s = num2str(x)` konverterar talet `x` till en sträng `s`.
`s = num2str(x, n)` ger `n` siffror. T.ex.
`s = num2str(-123.5678, 4)` ger `s = '-123.6'`.
`x = str2num(s)` konverterar en sträng till ett tal (väsentligen inversen till `num2str`).

```
char(v)   teckenkoder -> tecken
double(s) sträng -> teckenkoder
lower(s)  konvertera till gemena
upper(s)  konvertera till versaler
```

Strängjämförelse

`strcmp(s1, s2, n)` jämför de `n` första tecknen i strängarna `s1` och `s2`, med beaktande av "case", och returnerar 1 om likhet och 0 annars.

Blanktecken är signifikanta. Om `n` är större än `length(s1)` eller `length(s2)` returneras värdet 0.
`strcmpi(s1, s2, n)` fungerar analogt men struntar

i "case". Om man vill jämföra hela strängarna (inget `n`) finns `strcmp(s1, s2)` med case och `strcmpi(s1, s2)` utan.

`strfind(s1, s2)` returnerar startindex (kan vara flera) av början av `s2` i `s1`. Om `s2` är längre än `s1` returneras `[]`.

```
>> strfind('Abracadabra', 'ra')
ans = 3    10
```

```
>> strfind('Abracadabra', 'ab')
ans = 8
```

`ischar(arg)` returnerar `true` om `arg` är en teckenvektor, annars returneras `false` (om `arg` är ett tal t.ex).
`isletter(s)` returnerar en logisk vektor med ettor där `s(k)` är bokstäver (noll i övriga positioner).

Logiska funktioner

```
all(v)   returnerar 1 om alla v(k) är skilda
         från noll, returnerar 0 annars
any(v)   returnerar 1 om något v(k) är skilt
         från noll, returnerar 0 annars
false    returnerar logical(0)
true     returnerar logical(1)
```

Funktioner som opererar på mängder

En mängd kan representeras som en vektor av tal eller en cellvektor av strängar. Låt `u` och `v` vara två (cell)vektorer. Utdata från de fyra första funktionerna sorteras också.

```
u = unique(v)      u blir v utan upprepningar
union(u, v)        unionen
intersect(u, v)     snittet
setdiff(u, v)       skillnaden u \ v
ismember(tal, v)    sant om tal tillhör v
ismember(s, v)      sant om strängen s tillhör
                    cellvektorn v
```

Unixanrop

```
[status, resultat] = unix('kommando')
```

exekverar kommandot i ett skal (normalt `bash` eller `tcsh`) och returnerar resultatet i strängen `resultat` och status i `status` (0 om det gick bra och 1 om det gick fel).

