F4: Datastrukturer (kap. 6–7)

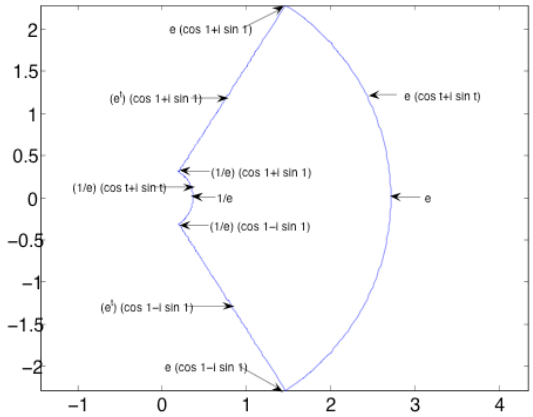
komplexa tal
strängar och textbehandling
glesa tabeller
celltabeller
posttabeller
logisk tabell och mask
formelbehandling

Komplexa tal

Den imaginära enheten skrivs `i` eller `j` om man inte har gjort om `i` eller `j` till vanlig variabel. Kommandot `clear i j` återställer. Kommandot `isreal` undersöker om alla element har imaginärdelen 0. Funktioner som är definierade i matematiken för komplexa argument eller kan ge komplexa resultat fungerar så i Matlab. Skriv en funktion som ritar en funktion av z för z=1+t i, –t+i, –1–t i, t–i för –1<=t<=1.

```
function ritakomplex(fun)
step = 0.02;
t = [-1:step:-step step:step:1];
z =[1+t*i -t+i -1-t*i t-i];
f = feval(fun,z);
re = real(f);
im = imag(f);
plot(re,im,'-');
```


Testa med `ritakomplex('exp'); % Texten handredigerad`




```
exp(x+i*y)=exp(x)*(cos y+i*sin y)
De 4 delarna blir:
e*(cos t+i*sin t)
exp(t)*(cos 1+i*sin 1)
(1/e)*(cos t+i*sin t)
exp(t)*(cos 1-i*sin 1)
Några funktioner som har med komplexa tal att göra
är:
```

`real(z)`	realdelen
`imag(z)`	imagimärdelen
`isreal(z)`	imaginärdelen är 0
`abs(z)`	beloppet
`angle(z)`	argumentet
`conj(z)`	negera imaginärdelen

Strängar och textbehandling

Lagring, utskrift, operatorer, funktioner, ischar, double/int16/abs, char, […;…], char('…','…'), deblank, strcat, strvcat (med utfyllnad), strcmp, strcmpi, strncmp, strncmpi, findstr, lower, upper, strings, tecken enligt Latin1 konverteras ok, strtok,

Radplanering

Skriv en funktion som radplanerar en text, *t*, till angiven radlängd.

```
t = ['Här har vi ett  exempel på en    text ' ...
'som  ska radplaneras.   Jag hoppas  att det ' ...
'går bra.'];
paragraph(t,12)
```

- Hitta alla blanktecken. Ord kan börja efter ett blanktecken och kan sluta med textens sista tecken, lägg därför till 0 och numel(*t*)+1.
- Antag att nästa ord börjar efter nästa blanktecken och slutar före följande blanktecken. Ignorera tomma ord (flera blanka i rad).
- Om ordet ryms på raden lägg till blanktecken och ordet och håll reda på radens längd.
- Om ordet inte ryms, lägg till raden i textmatrisen och lägg det nya ordet först på raden.
- När texten är slut, lägg sista raden till textmatrisen.


```
function text = paragraph(t,llength)
% Radplanerar texten t till radlängden llength.
bpos = [0 findstr(t,' ') numel(t)+1];
% Index för blanktecknen
text = ''; % Det blivande stycket
line = ''; % Aktuell rad
lpos = llength+1;
% Antalet använda platser på aktuell rad
% För varje möjligt ord
for bno = 1:numel(bpos)-1
    % Sök nästa ord,
    % deblank ignorerar avslutande blanka
    word = t(bpos(bno)+1:bpos(bno+1)-1);
    if ~isempty(word)
        if llength<lpos+1+numel(word);
```



```
            % Ordet ryms ej, initiera ny rad
            text = strvcat(text,line);
            line = word; lpos = numel(word);
        else
            % Ordet ryms
            % Lägg ordet till aktuell rad
            line =[line ' ' word];
            lpos = lpos+1+numel(word);
        end
    end
end
% Hantera ev. icke-fylld slutrad
if ~isempty(line)
    text = strvcat(text,line);
end
```

Radplanering forts.

Testkör:

```
t = ['Här har vi ett  exempel på en    text ' ...
'som  ska radplaneras.   Jag hoppas  att det ' ...
'går bra.'];
paragraph(t,12)
ans =
Här har vi
ett exempel
på en text
som ska
radplaneras.
Jag hoppas
att det går
bra.
```


Radplanering testversion

```
function text = paragraphdb(t,llength)
% Radplanerar stycken till given radlängd.
% t = ['Här har vi ett  exempel på en text   som
ska ',...
%    'radplaneras.   Jag hoppas att det  går
bra.'];
bpos = [0 findstr(t,' ') numel(t)+1] % Index för
blanktecken
text = ''; % Det blivande stycket
line = ''; % Aktuell rad
lpos = llength+1; % Antalet använda platser på
raden
% Så länge ord återstår
for bno = 1:numel(bpos)-1
    % Sök nästa ord
```



```
    it = bpos(bno)+1:bpos(bno+1)-1
    word = t(it);
    if ~isempty(word)
        if llength<lpos+1+numel(word);
            disp('Ordet ryms ej');
            % Initiera ny rad
            text = strvcat(text,line)
            line = word, lpos = numel(word)
        else
            disp('Ordet ryms');
            % Lägg odet till aktuell rad
            line =[line ' ' word]
            lpos = lpos+1+numel(word),
        end
    end
end
% Hantera ev. icke-fylld slutrad
if ~isempty(line)
```



```
    text = strvcat(text,line);
end
```

Testkör:

```
t = ['Här har vi ett  exempel på en    text ' ...
'som  ska radplaneras.   Jag hoppas  att det ' ...
'går bra.'];
paragraphdb(t,12)
bpos =
  Columns 1 through 13
     0     4     8    11    15    16    24    27
30    31    32    33    38
  Columns 14 through 26
    42    43    47    60    61    62    66    73
74    78    82    86    91
it =
     1     2     3
Ordet ryms ej
```

```
text =
     ''
line =
Här
lpos =
     3
it =
     5     6     7
Ordet ryms
line =
Här har
lpos =
     7
it =
     9    10
Ordet ryms
line =
Här har vi
```



```
lpos =
    10
it =
    12    13    14
Ordet ryms ej
text =
Här har vi
line =
ett
lpos =
     3
it =
   Empty matrix: 1-by-0
it =
    17    18    19    20    21    22    23
Ordet ryms
line =
ett exempel
```



```
lpos =
    11
it =
    25    26
Ordet ryms ej
text =
Här har vi
ett exempel
line =
på
lpos =
     2
it =
    28    29
Ordet ryms
line =
på en
lpos =
```



```
     5
it =
   Empty matrix: 1-by-0
it =
   Empty matrix: 1-by-0
it =
   Empty matrix: 1-by-0
it =
    34    35    36    37
Ordet ryms
line =
på en text
lpos =
    10
it =
    39    40    41
Ordet ryms ej
text =
```

```
Här har vi
ett exempel
på en text
line =
som
lpos =
     3
it =
   Empty matrix: 1-by-0
it =
    44    45    46
Ordet ryms
line =
som ska
lpos =
     7
it =
    48    49    50    51    52    53    54    55
```



```
56    57    58    59
Ordet ryms ej
text =
Här har vi
ett exempel
på en text
som ska
line =
radplaneras.
lpos =
    12
it =
   Empty matrix: 1-by-0
it =
   Empty matrix: 1-by-0
it =
    63    64    65
Ordet ryms ej
```



```
text =
Här har vi
ett exempel
på en text
som ska
radplaneras.
line =
Jag
lpos =
     3
it =
    67    68    69    70    71    72
Ordet ryms
line =
Jag hoppas
lpos =
    10
it =
```



```
   Empty matrix: 1-by-0
it =
    75    76    77
Ordet ryms ej
text =
Här har vi
ett exempel
på en text
som ska
radplaneras.
Jag hoppas
line =
att
lpos =
     3
it =
    79    80    81
Ordet ryms
```

```
line =
att det
lpos =
     7
it =
    83    84    85
Ordet ryms
line =
att det går
lpos =
    11
it =
    87    88    89    90
Ordet ryms ej
text =
Här har vi
ett exempel
på en text
```



```
som ska
radplaneras.
Jag hoppas
att det går
line =
bra.
lpos =
     4
ans =
Här har vi
ett exempel
på en text
som ska
radplaneras.
Jag hoppas
att det går
bra.
```


Strängar forts.

En sträng är en radvektor med teckenelement. Konkatenera (sammanfoga) strängar horisontellt:

`t = ['abc' 'def ' 'ghi  ']` då inkluderas avslutande blanktecken (och andra »vittecken») i delsträngarna,

`t = strcat('abc','def ','ghi  ')` som ignorerar avslutande »vittecken» i delarna.

Konkatenera vertikalt:

`t = ['12345';'67890']` raderna måste vara lika långa

`t = strvcat('123',' ','45678')` alla rader fylls med blanka till längsta radens längd, tomma strängar ignoreras.

`t = char('123',' ','45678')` som `strvcat` men tomma strängar ignoreras ej.

Man konverterar mellan tal och tecken med `char` och t.ex. `abs`. Man testar om tecken är bokstav med `isletter` och om ett tecken är

»vittecken» med `isspace`. Man konverterar mellan versaler och gemena med `lower` och `upper`.

Det finns olika sätt att söka och jämföra texter. Några funktioner är:

`findstr(`*text1*`,` *text2*`)` ger de index i *text1* där *text2* börjar,

`strcmp(`*text1*`,` *text2*`)` ger sant om texterna är lika,

`strcmpi(`*text1*`,` *text2*`)` fungerar som `strcmp(upper(`*text1*`), upper(`*text2*`))` dvs. ignorerar skiftläge i jämförelsen,

`strncmp(`*text1*`,` *text2*`,` *n*`)` jämför de första *n* tecknen i texterna,

`[`*t*`,`*r*`] = strtok(`*s*`,`*d*`)` letar nästa »token» (symbol) i *s*. Hoppa över inledande tecken ur *d* (delimiters, avskiljare), låt *t* vara följande tecken som inte finns i *d* och låt sedan *r* vara resten av *s*. Man kan skriva programmet paragraph med `strtok`.

16:15 är en tom vektor.

Datarepresentation

Talet sjutton kan representeras som strängen '17' som lagras som koden för '1' som är 49 följt av koden för '7' som är 55 eller som 17 som lagras som binärt heltal 0…010001 eller som flyttal som ett annat bitmönster med delarna *t1 m t2 e* ≈*t1*m**2^(*t2*e*). Man kan konvertera mellan strängform och intern binär form med `num2str`, `str2num` och andra funktioner.

Glesa tabeller

I vissa sammanhang använder man tabeller där de flesta elementen är 0. Man kan ange att tabeller ska lagras som glesa tabeller dvs. radnummer, kolumnnummer och värde för de element som inte är 0.

Många numeriska metoder använder glesa matriser. Algoritmer för nät (grafer) kan skrivas med glesa matriser t.ex. för att hitta billigaste väg eller gå igenom en labyrint.

Celltabeller

I vanliga tabeller har alla element samma typ. I celltabeller kan elementen ha olika typ.

```
a = [{[1 3 –7;2 0 6;0 5 1]}, ...
{'This is a text string.'}; ...
{[3+4*i –5;-10*i 3-4*i]} {[]}];
a(1,1) % cellen med index (1,1)
ans =
[3x3 double]
a{1,1} % innehållet i cellen med index (1,1)
ans =
[1 3 –7
 2 0 6
 0 5 1]
```

En celltabell är en tabell med celler som element. Varje cell är en variabel. Man kan använda ett förenklat skrivsätt:

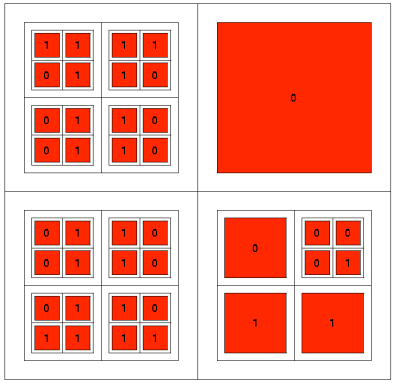

```
a = {[1 3 –7;2 0 6;0 5 1], ...
'This is a text string.'; ...
[3+4*i –5;-10*i 3-4*i], []}
```

En cell kan innehålla en celltabell. Använd celltabell för att kvadrantkoda en bild.

```
p =
{{{1,1;0,1},{1,1;1,0};{0,1;0,1},{1,0;1,0}},0;...
{{0,1;0,1},{1,0;1,0};{0,1;1,1},{1,0;1,1}},{0,{0,0;
0,1};1,1}};
```

Celltabell forts.

```
cellplot(p);
```



Posttabeller

I vanliga tabeller och celltabeller når man elementen med index. I posttabeller använder man namn för att nå elementen. Vi kan använda en posttabell för att lagra information om bankkonton. Beskrivningen av ett konto är en »post», ett element i posttabellen. Varje post har samma uppsättning fält med varsitt namn. Vi väljer fältnamnen `kontonummer`, `namn`, `ranta` och `saldo` och skriver en funktion, `trans`, för att göra en transaktion och låter `trans` ha en lokal funktion för att hitta en post med ett visst kontonummer.

```
function saldo = trans(accountno,amount)
% Utför en banktransaktion.
% saldo = trans(accountno,amount)
% Om amount >=0 insättes amount på kontot.
% Om amount<0 kontrolleras saldot och om
%  saldo+amout<0 skrivs ett varningsmeddelande
%  annars ändras saldo till saldo+amount.
```


Bankexempel forts.

```
global REG
% hitta kontot
index = regfind(accountno);
if index<=0
  disp('Kontonumret är okänt.');
  saldo = -1; return;
else
  strvcat(['Transaktion med belopp ' ...
  num2str(amount)], ['på konto ' ...
  num2str(accountno) ' med '], ...
  ['innehavare ' REG(index).namn ...
  ' och saldo före: ' ...
  num2str(REG(index).saldo) '.'])
end
if REG(index).saldo+amount<0
```


Bankexempel forts.

```
  disp(['Övertrassering. Transaktionen ' ...
  'utförs ej.']);
  saldo = REG(index).saldo;
else
  saldo = REG(index).saldo+amount;
  REG(index).saldo = saldo;
end
function index = regfind(accountno)
global REG
for no = 1:numel(REG)
  if REG(no).kontonummer==accountno
    index = no; return;
  end
end
index = -1;
```

Bankexempel forts.

Testkör:

```
global REG
REG.kontonummer = 12357;
REG.namn = 'Kloker';
REG.ranta = 2.2;
REG.saldo = 200;
REG(2).kontonummer = 314159;
REG(2).namn = 'Trötter';
REG(2).ranta = 1.9;
REG(2).saldo = 10500;
trans(12357,100)
Transaktion med belopp 100
på konto 12357 med
innehavare Kloker och
saldo före: 200.
```


Bankexempel forts.

```
ans = 300
trans(12357,100)
Transaktion med belopp 100
på konto 12357 med
innehavare Kloker och
saldo före: 300.
ans = 400
trans(12358,100)
Kontonumret är okänt.
ans = -1
trans(12357,-100)
Transaktion med belopp -100
på konto 12357 med
innehavare Kloker och
saldo före: 400.
```


Bankexempel forts.

```
ans = 300
trans(12357,-1000)
Transaktion med belopp -1000
på konto 12357 med
innehavare Kloker och
saldo före: 300.
Övertrassering. Transaktionen utförs ej.
ans = 300
```

Registret kan ju inte vara parameter för vi vill ju att de ändringar som görs av funktionen ska märkas utanför. Vi kan ha registret både som in- och utparameter men det innebär alltför mycket kopiering.

Mask

Man kan välja ut vissa element i en tabell så att endast dessa element påverkas av en operation och övriga element lämnas opåverkade. För detta använder man en »logisk tabell». Användningen liknar användningen av masker i Photoshop o.d.

Fördubbla de positiva elementen i `A`.

```
A = [0 1 –7;-2 4 8;3 –5 –17];
m = A>0;
A(m) = 2*A(m)
A =
  0     2    -7
 -2     8    16
  6    -5   -17
```

Formelbehandling

Med Matlabs »Symbolic algebra toolbox» kan man göra formelbehandling. Matlab känner till deriverings- och integrationsregler och kan göra viss formelförenkling.

```
n = 4; syms x;
A = x.^((0:n)'*(0:n))
A =
[1,   1,   1,    1,    1]
[1,   x, x^2,  x^3,  x^4]
[1, x^2, x^4,  x^6,  x^8]
[1, x^3, x^6,  x^9, x^12]
[1, x^4, x^8, x^12, x^16]
```


Formelbehandling forts.

```
D = diff(log(A))
D =
[0,    0,    0,    0,    0]
[0,  1/x,  2/x,  3/x,  4/x]
[0,  2/x,  4/x,  6/x,  8/x]
[0,  3/x,  6/x,  9/x, 12/x]
[0,  4/x,  8/x, 12/x, 16/x]
syms a b; expand((a^3+b^2)^3)
ans =
a^9+3*a^6*b^2+3*a^3*b^4+b^6
```


Formelbehandling forts.

I laboration 5 finns en differensapproximation till derivatan av en funktion i dess vänstra ändpunkt. Låt funktionen vara ui=u(xi), i=0, 1, …N-1, N där h=L/N. Om vi approximerar funktionen med en rät linje får vi du/dx(x0)=(u1–u0)/h. Approximera istället med en parabel y=Ax^2+Bx+C med y'=2Ax+B.

```
syms A B C x0 u0 u1 u2 h N d e
e = solve('A*x0^2+B*x0+C=u0',...
'A*(x0+h)^2+B*(x0+h)+C=u1',...
'A*(x0+2*h)^2+B*(x0+2*h)+C=u2','A','B','C');
d = 2*A*V+B;
simplify(subs(d, {A B}, {e.A e.B}))
```

dvs.

```
ans = 1/2*(-3*u0+4*u1-u2)/h
```

Det stämmer med formeln i labb 5.

Man kan göra motsvarande för funktionens högra ände.

Formelbehandling forts.

Med `syms` deklarerar man vilka variabler som ska vara symboliska.

`S = solve('eqn1','eqn2',...,'eqnN', 'var1','var2',...'varN')` löser `vari` ur ekvationssystemet `eqni` som `S.vari`.

`subs(f, {v1 v2 … vn}, {e1, e2, …, en})` ersätter i formeln `f` variablerna `vi` med uttrycken `ei`.

`simplify(f)` förenklar formeln `f`. Se hjälpen för information om hur man kan styra förenklingen.