Matlab-repetition

- Substitutionschiffer
- Permutationer
- Rekursion

”Function functions”

Fil-skrivning och läsning med formatstyrning

Formelbehandling

Ank-race

Substitutionschiffer

Caesar-rullning

Ex. Alfabete: 1234;

text: 2 3 1 1 2;

Substitution: Ett stegs cykliskt skift

1->2

2->3

3->4

4->1

```
% subst-chiffer
alfab = [1 2 3 4];
indx  = [2 3 4 1]; % en permutation av 1..n
txt   = [2 3 1 1 2]
txt2  = alfab(indx(txt))
% hur översätta tillbaka?
```

ASCII och char

Alla tecken :

```
% alla ASCII tecken
indx = 1:256;
char(indx)
pause
txt  = ' hej hej ö #Å'
txt2 = char(indx(double(txt)))
pause
indx = rand_perm(256); % slump-permutation
txt3 = char(indx(double(txt)))
pause
iindx(indx)=1:256;     % invers-permutationen
txt4 = char(iindx(double(txt3)))
```

Permutationer och rekursion

Exempel 1:

Beräkning av fakulteter, $n! = n\,(n\text{-}1)!$; $1! = 1$

Rekursion:

```
function nfac = fact_rec(n)
% compute n factorial = 1 2 3 ... n
% by recursion
%-------------------------------
if n == 1
    nfac = 1;
else
    nfac = n*fact_rec(n-1);
end
```

Det är enkelt att konstruera en icke-rekursiv algoritm,

```
nfac = 1;
for k = 2:n
   nfac = nfac*k;
end
```

som implementeras av matlabs **`prod`**-funktion,

```
nfac = prod(1:n);
```

Exempel 2: Beräkna det *n*:te Fibonacci-talet,

$$f_n = f_{n-1} + f_{n-2};\ f_1 = 1;\ f_2 = 2;$$

1. Hur ser en rekursiv algoritm ut?

2. Iterativ:

```
function fn = fibo(n)
% assume n is a positive integer
if n == 1, fn = 1;
elseif n == 2, fn = 2;
else f1 = 1; f2 = 2;
    for k = 3:n
        fn = f1 + f2;
        f1 = f2;      % no need to save all fk,
        f2 = fn;      % just the two last
    end
end
```

Formel: $f_n = \frac{3+\sqrt{5}}{5+\sqrt{5}}\left(\frac{\sqrt{5}+1}{2}\right)^n + \frac{2}{5+\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n \approx \frac{3+\sqrt{5}}{5+\sqrt{5}}\left(\frac{\sqrt{5}+1}{2}\right)^n \quad n >> 1$

Permutationer

Hur generera en slumpmässig permutation av *n* objekt – talen 1,2,…,*n*?

1. ”Dragning utan återläggning”: Tag objekt slumpmässigt tills inga är kvar: **`rand_perm2`**
2. ”Kortblandning”: Byt obj 1 mot slumpmässigt annat, sedan 2 mot annat, etc. Upprepa *n* gånger: **`rand_perm`**

```
function indx = rand_perm2(n)
% by selection w/o return
tmp  = 1:n;
L    = n; % number of items
indx = tmp;
for k = 1:n
    k1     = floor(L*rand)+1; % random selection
    indx(k)= tmp(k1);         % save the item
    tmp(k1)= [];              % remove it from list (*)
    L      = L-1;             % length decreases by 1
end
```

```
function indx = rand_perm(n)
% indx is a random permutation of 1...n
% algorithm: do n times
%            swap two elements
indx = 1:n;
for k = 1:n
    k1 = floor(n*rand)+1;
% swap indx(k1), indx(k)
    tmp     = indx(k1);
    indx(k1)= indx(k);
    indx(k)= tmp;
end
```

Denna är mycket snabbare än **`rand_perm2`**. Det beror på borttagningen av element (rad (*)) som kräver omflyttning. Tidtagning och användning av profil-verktyget bekräftar denna hypotes: 70% av tiden i **`rand_perm2`** ägnas åt borttagningen!

Exempel 3

Generera lista med alla permutationer av elementen i en vektor **`v(1:n)`**

Det är ganska enkelt att göra en rekursiv algoritm.

* Om bara ett objekt finns har listan bara ett element.
* Givet en lista över permutationer av *n*-1 objekt, så får man listan för *n* genom att ersätta var och en av dem med de *n* olika man får genom att sätta det nya objektet i position 1, 2, …, *n*.

1: 1

1,2: **2** 1;

1 **2**

1,2,3: **3** 2 1;

2 **3** 1;

2 1 **3**;

3 1 2;

1 **3** 2;

1 2 **3**

```
function A = recperm(v)
% generate all permutations of elements in
% array v by recursion
%-----------------------------------------------
n = length(v);
if n == 1                                % just return V
    A = v;
else
    B   = recperm(v(1:n-1));             % permute all
                                         % but the last,
    col = v(n)*ones(size(B(:,1)));       % make a column of
                                         % correct size
    A   = [col,B];                       % first block of
                                         % rows of A
    for k = 1:n-1                        % n-1 next blocks
        A = [A;...
            [B(:,1:k),col,B(:,k+1:end)]];
    end
end
```

Rekursionen kräver att hela tillståndet för funktionen sparas för n, n-1,…, tills man kommer till n = 1. Men här är tillståndet litet: data växer först i **`else`**-delen ovan. Då först sänds något tillbaka och listorna växer.

Övning: Hur mycket lagras med **`recperm(n)`**?

Visa, att $S_n = \sum_{k=1}^{n} k \cdot k! = 1 + 2.2 + 3.6 + 4.24 + ... = (n+1)!-1$

Skrivning och läsning med formatstyrning

- Läs en text-fil med **`fileread`**(*filnamn*) som ger filens innehåll som en sträng. Använd **`uigetfile`**:

```
>> fname = uigetfile('*') % ’*’ - filter för vilka filnamn
                          % som ska visas, ex. ’*.m’
fname =
fiboan.m
>> txt = fileread(fname)
txt =
function f = fiboan(n)
fi = (sqrt(5)+1)/2;
```

Skrivning som text med filreferens:

- **`fprintf`**(*filreferens*,*formatsträng*,*uttryck*,…)

Utelämnad filreferens = standard output = skärm

- **`sprintf`**(*formatsträng*, *uttryck*,…) returnerar en sträng

Funktion som parameter (function functions)

I många tillämpningar har man en metod (function method) som man ska kunna använda med många olika funktioner

1 Uttryck som text och `eval`

Fil: **`method2a.m`**

```
function method2a(f)
for x=3:3:10
    eval(f);
end
```

Fil: **`use2a.m`**

`use2a.m`

```
m_line = 'disp([''f2 x='' num2str(x)]);';
method2a(m_line);
============
```

```
>use2a
f2 x=3
f2 x=6
f2 x=9
```

2 Enkel funktionsreferens (`function handle`)

Fil: `use3a.m`

```
f=@f3a;
method3a(f);
```

```
>use3a
f3a x=3
f3a x=6
f3a x=9
```

Fil: `f3a.m`

```
function f3a(x)
disp(['f3a x='num2str(x)]);
```

Fil: `method3a.m`

```
function method3a(f)
for x=3:3:10
   f(x);
end
```

2b Funktionsreferens med parameter

Fil: use3b.m

```
for y = [5 10]
   f = @(x)f3b(x,y);
   method3b(f);
end
```

Fil: f3b.m

```
function f3b(x,y)
disp(['f3b x=' num2str(x)
' y=' num2str(y)]);
```

Fil: method3b.m

```
function method3b(f)
for x=3:3:10
   f(x);
end
```

```
>use3b;
f3b x=3 y=5
f3b x=6 y=5
f3b x=9 y=5
f3b x=3 y=10
f3b x=6 y=10
f3b x=9 y=10
```

Formelbehandling

Med Matlabs »Symbolic algebra toolbox» kan man göra formelbehandling. Matlab känner till deriverings- och integrationsregler och kan göra viss formelförenkling.
Med **`syms`** deklarerar man vilka variabler som ska vara symboliska.

```
S = solve('eqn1','eqn2',...,'eqnN',. . .
          'var1','var2',...'varN')
```

löser **`{vari},i=1:N`** 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.

Exempel

```
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 en numme-laboration finns en differensapproximation till derivatan av en funktion i dess vänstra ändpunkt. Vi ska beräkna koefficienterna i den. Låt funktionsvärdena på gitter-punkterna vara

$u_i = u(x_i)$, $x_i = ih$, $i = 0,1, \ldots, N\text{-}1$, N där $h=L/N$.

Om vi approximerar funktionen med en rät linje får vi

$u'(x_0) = (u_1 - u_0)/h$

med fel prop. h^2

Approximera istället med en parabel ($x_0 = 0$!)

$u(x) = A(x-x_0)^2 + B(x-x_0) + C, u'(x) = 2A(x-x_0) + B$

och bestäm A, B, och C så att parabeln går genom

(x_0+kh,u_k), $k = 0,1,2$:

```
syms A B C u0 u1 u2 h d e
e = solve('A*0      +B*0    +C = u0',...
          'A*h^2    +B*h    +C = u1',...
          'A*(2*h)^2+B*(2*h)+C = u2','A','B','C');
d = 2*A*0+B % the derivative at x = 0
e.B
=== ==
d =
B
ans =
-(3*u0 - 4*u1 + u2)/(2*h)
```

Det stämmer med formeln i labb 5. Man kan göra motsvarande för funktionens högra ände.

Formelbehandling forts.

Man kan lösa polynomekvationer.

Exempel: Fibonacci-talen.

$f_{n+1} - f_n - f_{n-1} = 0$ ger karakteristisk ekvation

$x^2 - x - 1 = 0$

med rötter x_1, x_2. Då är

$f_n = A\, x_1^{\,n} + B\, x_1^{\,n}$

där A, B väljs så att det stämmer för $n = 0$ ($f_0 = 1$) och $n = 1$ ($f_1 = 1$)

```
syms x A B fibo
xs = solve('x^2 - x - 1 = 0','x')
x1 = xs(1); x2 = xs(2);
coeff = solve('A*x1^0 + B*x2^0 = 1',...
              'A*x1^1 + B*x2^1 = 1','A','B')
n = 7;
fibo = coeff.A*x1^n+coeff.B*x2^n
expand(fibo)
simplify(fibo)
double(subs(fibo))
```

Ankrace

1000 badankor åker i virvlande ström; ibland framåt, ibland bakåt. Svårt att känna till virvlarna i detalj, så vi gör en modell meds slump-hopp.
I varje tidssteg kan en anka hoppa fram ett steg, bakåt ett, eller stanna kvar, med sannolikheter p, q, och 1-p-q. Om $p > q$ kommer de netto att åka nedströms.

a) Hur ser flocken ut efter 500 tidssteg?

Rita ett ”histogram” som ger hur många ankor det finns i varje steg av forsen

b) Hur är tids-fördelningen när de passerar målbron, 100 steg nedströms?

Rita ett diagram över hur många ankor som passerat som funktion av tiden

=====

Olika angreppssätt:

1. Följ alla ankorna, tidssteg för tidssteg
2. Följ en anka i taget för alla tidssteg

Vi visar här 1.

```
% Ankrace: m ankor, n steg
% I varje steg: +1 med sannolikhet p, -1 med sannolikhet q
clear all, close all
p = 0.8; q = 0.18; % stor variation
n = 1000; % antal steg
m = 1000; % antal ankor
% Följ alla ankor på en gång,
% "histogram" N över ank-lägena X i varje steg
X        = zeros(m,n+1);
X(:,1)   = n;  % alla startar i 0
N        = zeros(2*n+1,n+1);
N(n+1,1) = m;  % alla startar i 0
for tid = 2:n+1      % kör n steg
    X(:  ,tid) = X(:  ,tid-1);   % antag att alla stannar,
                                 % rätta till felen:
    z  = rand(m,1);
    im1 = z<q;                    % ”logical” index (0 0 0 1 0 0 1 0…)
    ip1 = z>1-p;
    X(im1,tid) = X(im1,tid-1)-1; % de hoppar bakåt
    X(ip1,tid) = X(ip1,tid-1)+1; % de hoppar fram
    % Räkna ankorna i varje position
    for anka = 1:m
        var = X(anka,tid);
        N(var,tid) = N(var,tid)+1;
    end
end;
subplot(121); bar(N(:,800));
subplot(122); mesh(N/m); shading interp
```