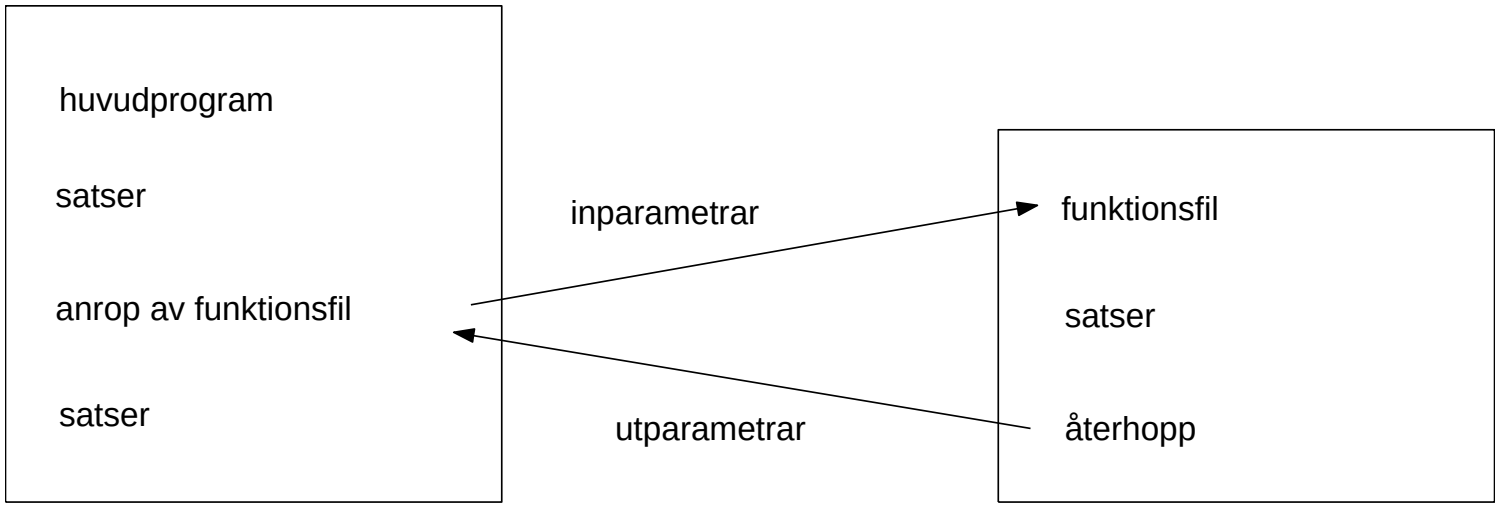
I dag

- Programstruktur
- Felsökning

Huvudprogram och underprogram

- När man löser stora problem försöker man ofta dela upp problemet i delproblem.
- Sedan skriver man en M-fil för varje del.
- Typiskt skriver man då en kommandofil/scriptfil (som kallas *huvudprogram*) som anropar funktionsfiler (som också kallas *subrutiner* eller *underprogram*).
- Huvudprogram och funktionsfiler kommunicerar via in- och utargument (parametrar).



- Anrop av funktionsfil.
- Efter att satserna i funktionsfilen utförts sker ett återhopp till huvudprogrammet.

- *Exempel.* För att räkna och skriva ut antal negativa element i varje rad i en matris kan vi skriva programmet (scriptfilen) **negativ.m**:

```
% negativ.m
A = input('Mata in en matris: ');
[m,n] = size(A);              % m = antal rader i A.
                              % n = antal kolonner i A.
for i = 1:m                   % Loopa över raderna.
    antal = 0;                % Nollställ räknare.
    for j = 1:n               % Loopa över kolonnerna.
        if A(i,j) < 0         % A(i,j) är elementet i rad i, kolonn j
            antal = antal + 1;
        end
    end
    fprintf('Negativa element i rad %i: %i\n',i,antal)
end
```

- *Exempel. (forts.)*

```
>> negativ
Mata in en matris: [5 -1 2; 0 4 2; -4 3 -8]
Negativa element i rad 1: 1
Negativa element i rad 2: 0
Negativa element i rad 3: 2
```

- Vi vill nu räkna både antal positiva element, antal negativa element, och antal element som är lika med noll i varje rad i en matris.
- Vi delar upp problemet i delproblem, och börjar med att skriva en funktion **tecken.m** som gör detta endast för en vektor.

- *Exempel.* Funktionsfilen **tecken.m** räknar antal element av olika tecken i en vektor.

```
function [neg,noll,pos] = tecken(v)
% Inparameter: v
% Utparametrar: neg  (antal negativa element i v)
%               noll (antal element som är noll i v)
%               pos  (antal positiva element i v)
neg = 0; noll = 0; pos = 0;
n = length(v);  % n: antal element i v
for j = 1:n   % Loopa över samtliga element.
    if v(j) < 0
        neg = neg + 1;
    elseif v(j) == 0
        noll = noll + 1;
    else
        pos = pos + 1;
    end
end
```

- *Exempel. (forts.)* Vi anropar funktionen **tecken.m** från scriptfilen **prog1.m**:

```
% prog1.m
A = input('Mata in en matris: ');
[m,n] = size(A);  % m: antal rader   n: antal kolonner
disp('Rad   #Negativa   #Noll   #Positiva')
for i = 1:m
  [neg,noll,pos] = tecken(A(i,:));     % A(i,:) betyder rad i.
  fprintf(' %i.        %i        %i         %i \n',i,neg,noll,pos)
end
```

- *Exempel. (forts.)* Körning av **prog1.m**:

```
>> prog1
Mata in en matris: [1 -4 0 0 -5 3 2 0; -2 0 -3 4 2 -1 -6 0]
Rad   #Negativa   #Noll   #Positiva
 1.        2        3         3
 2.        4        2         2
```

Funktion som inargument till funktion

- En funktion kan ha en annan funktion som inargument.
- Inargumentet skall då vara ett *funktionshandtag* (function handle).
- Funktionshandtag skapas genom att skriva `@` före funktionens namn, och fungerar som referens till denna.

- *Exempel.* Funktionsfilen **vardetabell.m** skriver ut en värdetabell för funktionen *fun* som skickas med som en inparameter.

```
function vardetabell(fun,x)
% Denna funktion skriver ut en värdetabell för funktionen fun
% Inparametrar: fun (funktionshandtag), x (radvektor)
% Utparametrar: Inga
y = fun(x);              % Beräkna funktionsvärden.
disp('     x     y')     % Skriv ut värdetabell.
disp([x' y'])            % x' transponerar x, d.v.s. omvandlar
                         %   x från radvektor till kolonnvektor.
```

- *Exempel. (forts.)* Vi anropar **vardetabell.m** för några olika funktioner:

```
>> vardetabell(@sqrt, [1 4 9 16 25])
     x     y
     1     1
     4     2
     9     3
    16     4
    25     5

>> vardetabell(@funk1, 1:4)  % funk1.m (se nästa sida)
     x     y
     1     2
     2     5
     3    10
     4    17
```

- **funk1.m**

```
function y = funk1(x)
% Denna funktionsfil beräknar y = x^2 + 1.
% Inparameter: x
% Utparameter: y
y = x.^2 + 1;  % Elementvis potens så kan inparameter
               % också vara en vektor.
```

Programmeringsfel och avlusning

- *Exempel.* Skrivfel/stavfel.

```
% fel1.m
x = linspace(0,2*pi);
y = sin(x);
plit(x,y)
```

Vid körning:

```
>> fel1
??? Undefined command/function 'plit'.

Error in ==> fel1 at 4
plit(x,y)
```

- *Exempel.* Liten och stor bokstav. Odefinierad variabel.

```
% fel2.m
x = linspace(0,2*pi);
Y = sin(x);    % STORT Y.
plot(x,y)      % litet y.
```

Vid körning:

```
>> fel2
??? Undefined function or variable "y".

Error in ==> fel2 at 4
plot(x,y)      % litet y.
```

- *Exempel.* Logiskt fel. Felaktig variant av **negativ.m**.

```
% fel3.m
A = input('Mata in en matris: ');
[m,n] = size(A);              % m = antal rader i A.
                              % n = antal kolonner i A.
antal = 0;                    % RÄKNARE NOLLSTÄLLS UTANFÖR YTTE LOOP.
for i = 1:m                   % Loopa över raderna.
    for j = 1:n               % Loopa över kolonnerna.
        if A(i,j) < 0
            antal = antal + 1;
        end
    end
    disp(['Negativa element i rad ' num2str(i) ': ' num2str(antal)])
end
```

- *Exempel. (forts.)* Vid körning fås istället *ackumulerat* antal negativa element:

```
>> fel3
Mata in en matris: [5 -1 2; 0 4 2; -4 3 -8]
Negativa element i rad 1: 1
Negativa element i rad 2: 1
Negativa element i rad 3: 3
```

- I MATLABs editor finns verktyg för att avlusa ett program, och se att det fungerar som man tänkt sig.

Övningar

1. Skriv in funktionsfilen **vardetabell.m**. Beräkna och skriv ut värdetabeller för funktionerna (skapa funktionsfiler för var och en):

   a) $f(x) = 4x + 1$

   b) $g(x) = (x - 1)^2$

   c) $h(x) = \sqrt{x^2 + 1}$

   Hitta på några egna också.

2. Modifiera scriptfilen **negativ.m** så att programmet i stället räknar hur många element i varje rad som tillhör intervallet $(2, 7)$, det vill säga är större än 2 och mindre än 7.

   Tips. Använd den logiska operatorn &. Se sid. 22–24 i förra veckans föreläsning.