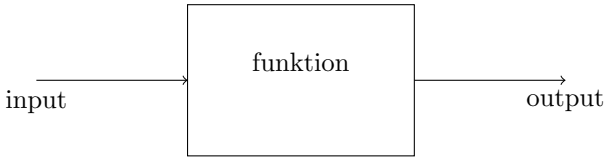
TMV176 Inledande matematik TD

MATLAB TD 2.2 — FUNKTIONER

Funktioner. I MATLAB-programmering spelar funktioner en viktig roll. De gör det möjligt att ”förpacka” och ”dölja” programkod och bryta ned ett längre program i överskådliga delar, där varje del är en funktion. Man kan se en funktion som en ”black box”:



När man använder funktionen ser man inte vad som händer inuti boxen, utan bara vad man stoppar in (input) och vad som kommer ut (output).

Vi börjar med ett enkelt exempel, funktionen

$$y = \frac{x + a}{x^2 + b^2}$$

Vi öppnar en ny m-fil av typen ”function” i katalogen `matlab2` och ger den namnet `funk1.m`. I filen står det nu:

```
function [ output_args ] = funk1( input_args )
%FUNK1 Summary of this function goes here
%   Detailed explanation goes here


end
```

Skillnaden mellan en script-fil och en funktionsfil är att funktionsfilen inleds med en funktionsdeklaration:

```
function [ output_args ] = funk1( input_args )
```

som anger funktionens namn (här `funk1`) och dess output och input variabler. Obs att funktionens namn ska vara detsamma som m-filens namn bortsett från ändelsen `.m`. Vår funktion har en input x och en output y. Vi fyller i det samt ett tillfälligt värde på y:

```
function y = funk1(x)
%FUNK1 Summary of this function goes here
%   Detailed explanation goes here

y=1
end
```

Vi provkör i kommandofönstret:

```
>> funk1(5)
```

Ett script kan köras med den gröna ”Run”-knappen i editor-fönstret, men en funktion måste köras i kommando-fönstret så att man kan ge input variablerna.

Stegvis programmering. Vi bygger upp funktionen genom stegvis programmering, dvs vi skriver en rad i taget och provkör innan vi skriver nästa rad. Vi börjar med att ge värden till a och b och räkna ut täljaren:

```
function y = funk1(x)
%FUNK1 Summary of this function goes here
%   Detailed explanation goes here
```

Date: 2011–09–08, Stig Larsson, Matematiska vetenskaper, Chalmers tekniska högskola.

```
a=1
b=2
taljare=x+a
y=1
end
```

Vi provkör

```
>> funk1(5)
>> ut=funk1(5)
```

och ser att vi får rätt värden på a, b och täljaren. Sedan skriver vi nämnaren:

```
function y = funk1(x)
%FUNK1 Summary of this function goes here
%   Detailed explanation goes here

a=1
b=2
taljare=x+a
namnare=x^2+b^2
y=1
end
```

Vi provkör och fortsätter sedan med slutresultatet:

```
function y = funk1(x)
%FUNK1 Summary of this function goes here
%   Detailed explanation goes here

a=1
b=2
taljare=x+a
namnare=x^2+b^2
y=taljare/namnare
end
```

Vi provkör tills vi är säkra på att funktionen är korrekt:

```
>> funk1(1)
>> y=funk1(0)
>> z=5
>> u=funk1(z)
```

Nu sätter vi in semikolon efter varje beräkning i filen och ändrar till elementvis operationer:

```
function y = funk1(x)
%FUNK1 Summary of this function goes here
%   Detailed explanation goes here

a=1;
b=2;
taljare=x+a;
namnare=x.^2+b^2;
y=taljare./namnare;
end
```

Den stegvisa programmeringen innebär att man kontrollerar och rättar alla fel innan man fortsätter. Gör alltid så.

Vi kan nu rita grafen:

```
>> x=linspace(0,10);
>> y=funk1(x);
```

```
>> plot(x,y)
```

eller

```
>> fplot(@funk1, [-20, 20])
```

Observera funktionshandtaget `@funk1`.

Interna variabler. Vilka variabler har du i ”Workspace” nu? Vilket värde har `a` och `taljare`? Vi provar:

```
>> a
>> taljare
```

Dessa variabler finns inte i Workspace. De används inuti `funk1` men de kan inte ses utanför funktionen. De kallas *interna variabler*. Om vi vill se vad som händer inuti funktionen kan vi ta bort ett eller flera semikolon. Detta är bra när man söker efter fel, men annars vill man inte se de interna variablerna.

De variabler som finns i Workspace kan man kalla *externa*.

Flera input och output variabler. För ändra värden på `a` och `b` måste vi ändra i m-filen. Alternativt kan vi låta dem vara input variabler:

```
function y = funk1(x,a,b)
%FUNK1 Summary of this function goes here
%   Detailed explanation goes here

taljare=x+a;
namnare=x.^2+b^2;
y=taljare./namnare;
end
```

Prova:

```
>> funk1(0,1,2)
```

Grafen:

```
>> x=linspace(0,10);
>> y=funk1(x,1,2);
>> plot(x,y)
```

eller

```
fplot(@(x)funk1(x,1,2), [0, 10])
```

Här blir vi tvungna att använda en anynom funktion för att tala om att vi ska plotta som funktion av x.

Om man vill veta vad nämnaren blir kan man låta den vara en output variabel:

```
function [y, namnare] = funk1(x,a,b)
%FUNK1 Summary of this function goes here
%   Detailed explanation goes here

taljare=x+a;
namnare=x.^2+b^2;
y=taljare./namnare;
end
```

Prova:

```
>> z=5
>> alfa=1
>> beta=2
>> [kvot, nmr]=funk1(z,alfa,beta)
```

Man måste ange båda output variablerna på kommandoraden annars får man bara den första. Men man kan kalla dem vad man vill.

Dokumentation. Till sist fyller vi funktionens dokumentation:

```
function y = funk1(x,a,b)
%FUNK1 computes y=(x+a)/(x^2+b^2)
% Syntax:  y = funk1(x,a,b)
% Input:   x - number or vector; a, b - numbers
% Output:  y - number or vector
% Example: funk1(0,1,2) gives 0.25

taljare=x+a;
namnare=x.^2+b^2;
y=taljare./namnare;
end
```

och provar den:

```
>> help funk1
```

Observera att dokumentationen skrivs som kommentarer (med procent-tecken) omedelbart efter funktionsdeklarationen utan tomma rader emellan. Dokumentationen kan även skrivas före deklarationen.

Övning 2.1. Skriv en funktion med deklarationen

```
function y=kast(t)
```

som beräknar kasthöjd enligt övningen från vecka 1. Här ska `t` vara ett tal eller vektor. Plotta från kommandofönstret med

```
>> t=linspace(0,2); y=kast(t); plot(t,y)
```

och med `fplot`. Plottningen ska alltså inte ingå i funktionen.

Övning 2.2. Samma som ovan men med deklarationen

```
function y=kast2(t,m,y_0,v_0)
```

Övning 2.3. Samma som ovan men med deklarationen

```
function y=kast3(T,m,y_0,v_0)
```

Funktionen ska beräkna och plotta kasthöjden på intervallet $[0, T]$. Nu ska plottningen skrivas in i funktionen.

`/stig`