Computer Algebra and Technical Computing (MTH1006)

B. Vorselaars
bvorselaars@lincoln.ac.uk

School of Mathematics and Physics, University of Lincoln

General notes

- Next week (24/10): hand-out first coursework Matlab
- 10/11: deadline hand-in first coursework.

Today

- Recap
- Output strings to screen
- `if` statement
- Creating your own `function`

Recap I

- Vector operations: $1 + 1/2 + 1/3 + \cdots + 1/10$

```
>> a=1:10; sum(1./a)
```

- Strings

```
>> txt3='H';
>> txt3(2:5)='ello'
txt3 =
Hello
```

Recap II

- Boolean algebra:

```
>> ~(10<3)
ans =
  logical
     1
```

- Is 2 larger than 1 or 10 smaller than 3?

```
>> (2>1) || (10<3)
ans =
  logical
     1
```

Yes

Recap III

- Plotting

```
>> x=-3:0.01:3
>> plot(x,x.^2)
>> plot(x,cos(x),'--')
>> xlabel('x')
```

Output to screen

How to display a string to screen?

- Method so far

```
>> 'Hi'
ans =
Hi
```

Drawbacks?

- Extra part is outputted:

```
ans =
```

- The output is assigned to the variable `ans`.

Output to screen

- Use the command `disp`

```
>> disp('Hi')
Hi

>> disp('This string will be printed to
   screen')
This string will be printed to screen

>> disp(['Hello,', ' how are you?'])
Hello, how are you?

>> x='a string';
>> disp(x)
a string
```

Converting numbers to strings

How to output the number contained in a variable in Matlab?

- So far we typed the name of a variable:

```
>> x=pi
x =
    3.1416

>> x
x =
    3.1416
```

Converting numbers to strings

- ```
  >> T=33;

  >> disp(['Temperature equals ',T]) % BUG
  Temperature equals !
  ```

  What goes wrong?? The number is interpreted as an ASCII character. Convert number to string using `num2str`

  ```
  >> T
  T =
       33
  >> num2str(T)
  ans =
  33

  >> disp(['Temperature equals ',
     num2str(T)])
  Temperature equals 33
  ```
  ```

If statement

Till now command execution is in a fixed sequence:

```
x=1:10

y=x.^2;

plot(x,y)

title('simple plot')
```

However, sometimes execution depends on a condition

```
if condition
  action
end
```

This is called an *if* statement. The *action* is only executed if the *condition* is fullfilled.

If statement

Consider the script

```
temperature=-10;
if temperature<0
   'It freezes'
end
```

Execution gives:

```
It freezes
```

In this *if* statement the condition is fullfilled and therefore the action is executed.

If statement

Consider the script

```
temperature=0;
if temperature<0
   'It freezes'
end
```

Execution gives:

```
% no output
```

In this *if* statement the condition is false, and therefore the action is not executed.

If statement

Example

```
x=sqrt(10)
y=3
if x>y
  'x is larger than y'
end
if x<y
  'x is smaller than y'
end
```

Test

```
x is larger than y
```

If statement

Example

```
x=sqrt(10)
y=3
if x>=y
  'x is larger than or equal to y'
end
if x<y
  'x is smaller than y'
end
```

Test

```
x is larger than or equal to y
```

Notice that the second condition is exactly the opposite of the first.

If statement

In pseudo-code (not real code, but sketchy)

```
if condition
  action1
end
if NOT condition
  action2
end
```

This occurs very often: shorthand using `else` has been introduced

```
if condition
  action1
else
  action2
end
```

If – else statement

Example

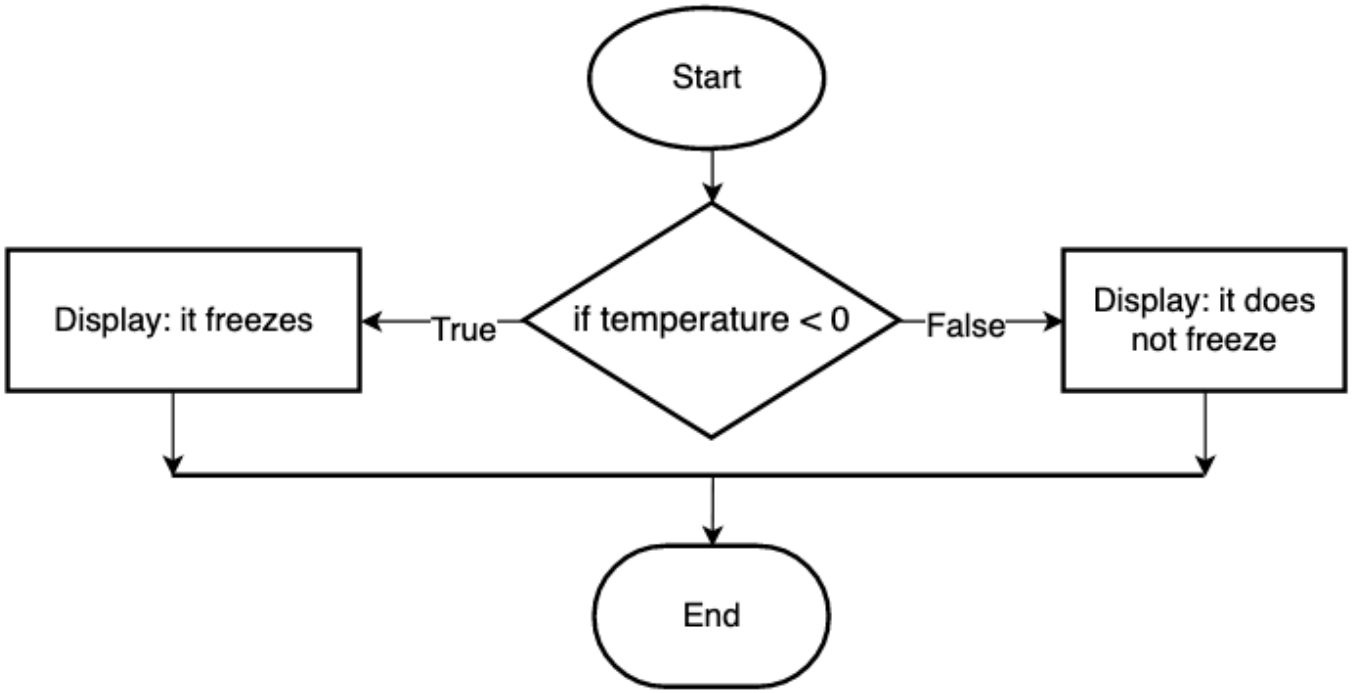
```
if x>0
  disp('Number is positive')
else
  disp('Number is zero or negative')
end
```

Test

```
>> x=3;
if x>0
  disp('Number is positive')
else
  disp('Number is zero or negative')
end
Number is positive
```

Alternative view: using a flowchart

Flowchart



corresponds to

```
if temperature<0
 'it freezes'
else
 'it does not freeze'
end
```

If – elseif – else statement

What if we have more than two options? Example:

- If $x > 0$: number is positive
- If $x = 0$: number is zero
- If $x < 0$: number is negative

If – elseif – else statement

Example

```
if x>0
  disp('Number is positive')
elseif x==0
  disp('Number is zero')
else
  disp('Number is negative')
end
```

If – elseif – else statement

Test

```
>> x=0;
if x>0
  disp('Number is positive')
elseif x==0
  disp('Number is zero')
else
  disp('Number is negative')
end
Number is zero
```

If – elseif – else statement

Extended if statement

```
if condition1
  statements 1
elseif condition2
  statements 2
elseif condition3
  statements 3
  .
  .
  .
else
  statements n
end
```

Functions

What is a function, mathematically?

- Example of defining a function:

$$f(x) = x^2 - 1$$

  - x: input
  - f: function
  - $f(x)$: output
  - $f(x) = x^2 - 1$: relating the input to the output. (similar to variable assignment!)

  A function f takes an input x and returns an output $f(x)$.
  Commonly: input and output are numbers
- Example of calling a function: The above function squares numbers. If the input is 3, the output is 8.

$$f(3)$$

will return 8

Functions

What is a function in Matlab?

- Example of defining a function:
  1. Create a `.m` file (similar to a script):

     ```
     edit f
     ```

  2. Fill in the `.m` file:

     ```
     function y_out=f(x_in)
     y_out=x_in^2-1
     end
     ```

     - `x_in`: state this is an input (variable)
     - `f`: state the name of the function. Has to be the *same* as the file name.
     - `y_out`: state this is an output (variable)
     - `y_out=x_in^2-1`: relating the input variable to the output variable

Function

- Example of calling our own function in Matlab

```
f(3)
```

will return

```
ans =
      8
```

Functions

Example of calling a known function in Matlab

```
>> x=cos(3)
```

- 3: input argument
- `cos`: function name, also known as calling name
- `x`: output argument

Different types of functions:

- Matlab intrinsic function: `cos`, `log`, `disp`
- Matlab `.m` file function. Example `factorial(n)`, which is $n!$. Note: the function's calling name is the name of the file (without `.m`).
- Your *own* function inside a dedicated function file (for example `my_function.m`).
- Your *own* function at the end of a script file (for example, `myscript_with_functions.m`)

Why define your own function?

```
>> 3.^2+3
ans =
     12

>> 2.5.^2+3
ans =
     9.2500

>> x=4;
>> x.^2+3
ans =
     19

>> [1, 4].^2+3
ans =
     4      19
```

Too much repetition

User-defined functions: function header

Define your own function. First create a new file by typing `edit my_polynomial`. The content of the file adheres to a strict format

```
function y_out=my_polynomial(x_in)
% Square the input and add 3
y_out=x_in.^2+3;
end
```

- `function`: keyword, denoting the start of a function definition. Here it implies the `.m` file is a function and not a script
- `y_out`: output argument, a variable that will contain the output value. When calling the function, this will indicate the output of the function.
- $=$: assignment operator, implying that the value of the variable before will be returned
- `my_polynomial`: name of our function, same as the name of the `.m` file
- `(x_in)`: parenthesis with in between the input argument; a

User-defined function

Define your own function:

```
function y_out=my_polynomial(x_in)
% Square the input and add 3
y_out=x_in.^2+3;
end
```

- `%`: comment line. Contains a description of the function.
- `y_out=x_in^2+3;` In this place the *function body* is given. This could be much longer and contains the actual instructions. Here the output `y_out` is calculated from the input `x_in`.
- `end`: keyword, denoting the end of a function definition.

demo

Calling our function

The function can simply be called by specifying it's name and an argument within brackets:

- Calling our function using a number

```
>> my_polynomial(0)
ans =
     3

>> my_polynomial(10)
ans =
     103
```

- Storing the output in a variable:

```
>> x=my_polynomial(10)
x =
     103
```

Calling our function

- Calling with a variable

```
>> a=3; b=my_polynomial(a)
b =
     12
```

- Calling using a vector

```
>> my_polynomial(1:3)
ans =
      4      7     12
```

User-defined function

Define your own function:

```
function result=is_freezing(temperature)
% Returns true if the temperature is below
    zero
if temperature<0
    result=true;
else
    result=false;
end
end
```

- Test for positive temperature

```
>> is_freezing(10)
ans =
  logical
   0
```

User-defined function

- Test for negative temperature

```
>> is_freezing(-10)
ans =
  logical
   1
```

User-defined function

A function does not need to have an output argument

```
function my_plot(c)
% Plot the curve x^2+c
x=linspace(-5,5,100);
y=x.^2+c
plot(x,y)
end
```

- Calling the function from the prompt

```
>> my_plot(0);
>> hold on;
>> my_plot(2);
>> my_plot(4);
>> hold off;
```

Function with multiple arguments

How to implement multiple input and/or output arguments?

- Maths example, two input arguments:

$$f(x, y) = x + y - 2$$

- Multiple input arguments: separated by comma's.
- Output arguments: simply return a vector.

Functions with multiple arguments

- Example of more input and output arguments

```
function
    [x_out,y_out]=polar_to_Cartesian(r_in,
    theta_in)
%Converts polar coordinates to Cartesian
    coordinates.
x_out=r_in.*cos(theta_in);
y_out=r_in.*sin(theta_in);
end
```

- Input arguments: `r_in` and `theta_in`
- Output arguments: `x_out` and `y_out`

Functions with multiple arguments

- Example of more input and output arguments

```
function
    [x_out,y_out]=polar_to_Cartesian(r_in,
    theta_in)
%Converts polar coordinates to Cartesian
    coordinates.
x_out=r_in.*cos(theta_in);
y_out=r_in.*sin(theta_in);
end
```

- Executing the function

```
>> [x,y]=polar_to_Cartesian(1,pi/4)
x =
    0.7071
y =
    0.7071
```

which are the values we expect (for both $1/\sqrt{(2)}$).

Types of functions in Matlab

The type of user-defined functions in Matlab have different properties.

- Function files: `.m` files with typically a single function (so not a script). They can be called from anywhere (other scripts, command prompt).
- Local functions within a script. These functions reside at the end of the script, and can only be called from within the script.
- Local functions within a function. These functions reside at the end of the function, and can only be called from within the function.

In this module we will be mostly using the first construct.