Computer Algebra and Technical Computing (MTH1006)

B. Vorselaars

bvorselaars@lincoln.ac.uk

School of Mathematics and Physics, University of Lincoln

Notes

Reminder dates

- Later today (24/10): hand-out first coursework Matlab
- In two weeks time (10/11): deadline hand-in first coursework.

Today

- functions (including recap).
- More flow control: loops

Recap if – elseif – else statement

```
>> 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
```

Recap function

```
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 local variable, which contains the input value

Recap function

Define your own function:

```
edit my_polynomial.m
```

Then insert the following text in the file:

```
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

Recap function

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

Example of calling the function in Matlab:

```
>> z=my_polynomial(2);
```

- 2: this value will be stored in the input argument `x_in` within the function
- `my_polynomial`: function name, also known as calling name
- `z`: the value of the output argument `y_out` will be stored in this variable `z`.

Function clarified

Why a Matlab function over a Matlab script? A function is similar to a script, but it has specific *input* and *output* arguments

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

If we want to test the script for another value of x, we need to *alter* the script. If instead we would write it as a function, we don't need to change the function. You wouldn't want to change a script called cos every time you want to calculate the cosine of a number!

Function clarified

Why a function over a script? A function is similar to a script, but it has specific *input* and *output* arguments

```
% Script
x=1;
if x>0
  disp('Number is positive')
end
Number positive
```

```
% Function
function msg=test_positive(x)
if x>0
  msg='Number is positive';
else
  msg='';
end
end
```

function clarified

```
% Function
function msg=test_positive(x)
if x>0
  msg='Number is positive';
else
  msg='';
end
end
```

Now we can test the function in a separate script:

```
x=4;
msg=test_positive(x);
disp(msg);
Number is positive
```

Loops

Sometimes we want to repeat a statement many times, maybe with slightly different conditions Consider the following script

```
x=1;
y=x^2+1
x=2;
y=x^2+1
x=3;
y=x^2+1
```

Running gives

```
y =
     2
y =
     5
y =
    10
...
```

How to do this more efficient?

```
x=1;
y=x^2+1
x=2;
y=x^2+1
x=3;
y=x^2+1
```

- Use vectors

  ```
  x=1:3
  y=x.^2+1 % The ^ is now replaced by .^,
      to allow for vector operations
  ```

  This is similar but not exactly the same as the original script, since y now becomes a vector

How to do this more efficient?

```
x=1;
y=x^2+1
x=2;
y=x^2+1
x=3;
y=x^2+1
```

- Use a *loop*

  ```
  for x=1:3
    y=x^2+1 % not necessary to do vector
        operation .^
  end
  ```

  This is entirely equivalent.

Loops

Loops allow us to repeat a set of statements many times.

`for` loops

With a *for* loop we know the number of loop iterations beforehand. The general syntax is

```
for x = a_vector
   statement(s) that may involve x
end
```

within the loop the content of `x` changes every iteration, it goes through all the elements of the `a_vector` one by one (so the `a_vector` is fed to `x` one element at a time.

for loop example

How to perform the following sum:

$$S = 1 + 2 + \ldots + 10$$

- Direct

  ```
  S=1+2+3+4+5+6+7+8+9+10
  ```

  Drawback: a lot of typing
- Using vectors

  ```
  x=1:10
  S=sum(x)
  ```

  ```
  x =
          1        2        3        4        5        6
              7        8        9       10
  S =
      55
  ```

for loop example

- Using a variable that is updated element by element.

```
S=0; % initialize

S=S+1;
S=S+2;
S=S+3;
...
S=S+10;
```

Here `S` is also known as the *accumulator*. So you add the element to `S` and update `S` at the same time. At the end all the elements are added to the variable `S`.

for loop example

- Again, using a variable acting as an accumulator, but now with a `for` loop, where you add every element of a vector to a variable.

```
S=0; % initialize

for x=1:10
    S=S+x;
end
```

More complex loop

How to add the square of the numbers ranging from 1 to 20?

$$S = \sum_{n=1}^{20} n^2$$

```
S=0; % initialize
for n=1:20
   S=S+n^2;
end
S
```

More complex loop: continue

It is possible to skip elements, using the keyword continue

$$S = \sum_{n=1, n\neq 5}^{10} n$$

```
S=0; % initialize
for n=1:10
    if n==5
        continue
    end
    S=S+n;
end
```

- continue: implies go to the next iteration at that point.
- Here: because of the if statement, if $n = 5$, the number n will *not* be added to the variable S.
- The result is therefore $S = 1 + 2 + 3 + 4 + 6 + 7 + 8 + 9 + 10 = 50$.

More complex loop: continue

What is wrong with the following code, if one wants to calculate $S = \sum_{n=1, n\neq 10}^{20} n$?

```
S=0; % initialize
for n=1:20
    S=S+n;
    if n==10
        continue
    end
end
```

- The `continue` statement is too late in the program, it should be before the line that adds the element $n = 10$ to S.

More complex loop: break

It is also possible to early exit a `for` loop.

```
S=0;
for n=1:20
    S=S+n;
    if n==10
        break
    end
end
```

- The `break` statement causes the for loop to exit when $n = 10$. So it only adds the numbers 1 to 10 to S, so $S = 1 + 2 + 3 + \ldots + 10$.
- Another (more practical) example: loop over a vector of numbers. If an odd number is found, break.