Introduction in MATLAB (TSRT04)

HT1 2019

Håkan Johansson

Division of Communication Systems
Department of Electrical Engineering (ISY)
Linköping University, Sweden

www.commsys.isy.liu.se/en/student/kurser/TSRT04

Purpose of the course

The course intends to give basic practical knowledge about the programming language MATLAB, which is used for technical computations. The course can both be seen as a preparation to a large number of courses where MATLAB is used, and as a possibility to learn a powerful engineering language.

After completing this course, the student is expected to be able to:

- *Use MATLAB as a calculator both for scalars and matrices. Use elementary functions and define variables.*
- *Construct simple scripts and functions.*
- *Use control structures (if-clauses, for- and while-loops) in simple examples.*
- *Graphically present results of computations and data sets.*
- *Use the help system to learn new functions.*

Course Outline

- 1 lecture, 2×2 h lessons, 5×2 h lab exercises
  - Notice discrepancy: 2 course credits $\sim$ 53 study hours!
  - 37 h of self studies — emphasis on “own excursions”.
  - Course material, instructions, information and links available at course homepage. Read through the whole homepage!
  - Download MATLAB from the Student portal!
  - Course book not required — some book options and video tutorials on the homepage.

- Lessons and lab exercises
  - Learn by doing — best way to learn MATLAB.
  - Work in pairs: Find someone with a similar programming background.
  - We expect both of you to actually write and understand the code.
  - Lessons are optional, lab exercises are mandatory!

Register for lessons and lab exercises

2 groups

- Group A and B: Maria Posluk
- Group C and D: Martin Allander

Registration

- You choose your group — you should do it right away!
- Registration to one lesson group (covers two lessons) and one lab group (covers five labs)
- Registration is done in Lisam

Examination

1) Quiz

- Individual test on the MATLAB basics.
- At first lab exercise. Examines material from lessons.

2) Plot assignment

- Plot and visualize a data set in different ways.

3) "Mini-project"

- Solve a small problem and visualize the solution.
- Attendance on labs is mandatory until project is finished.

General guidelines

- Multiple options for plot assignment and project (choose 1!)
- All sessions are opportunities to ask questions.
- In between sessions: Work at home (5-6h per week)!
- Project presentation is done only at lab occasions 3 and 5.

Level of Difficulty

- Prerequisites: Linear algebra, Programming
- Swedish courses have *expected learning outcome*.
- Difficulty *always* depends on previous experience.
- Difference in programming experience matters.
- MATLAB will be useful in many courses.

What is MATLAB?

MATrix LABoratory (MATLAB)

- Advanced calculator for technical computing
- Simple but powerful programming language
- Numerical calculations (not symbolic as Mathematica)
- Available for Windows, Mac, Unix, Linux, etc.
- New versions twice/year: 2016a, 2016b, 2017a

- Pros: Easy to get started, easy to visualize results
- Pros: Many examples and toolboxes for various topics (e.g., math, statistics, optimization, telecom, control, biology, finance)
- Cons: Not the fastest code - but usually fast enough!

- Suitable for testing ideas, solving scientific problems, developing/validating algorithms

- **Octave:** Open source option — compatible with MATLAB

Division's research: 5G Wireless Communications

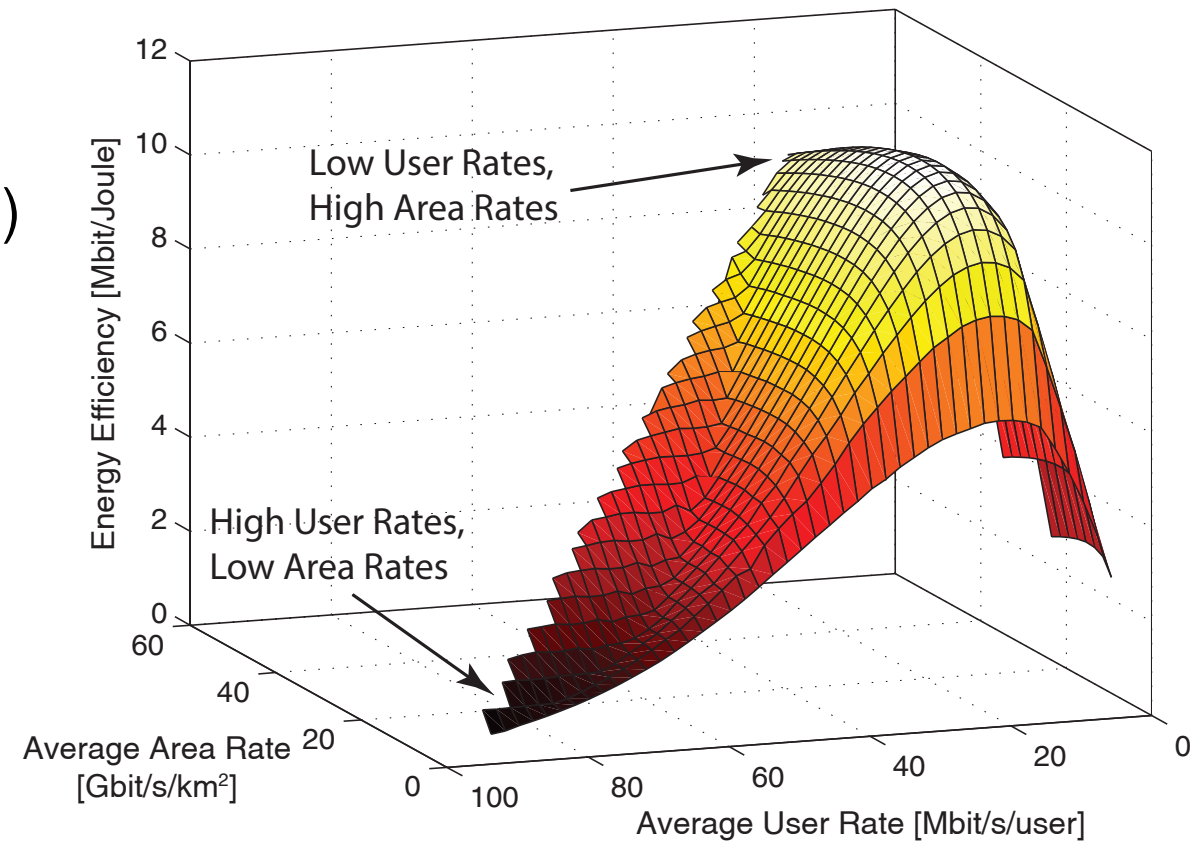
Goal: Develop design principles for the next generation cellular networks.

Understand interplay between

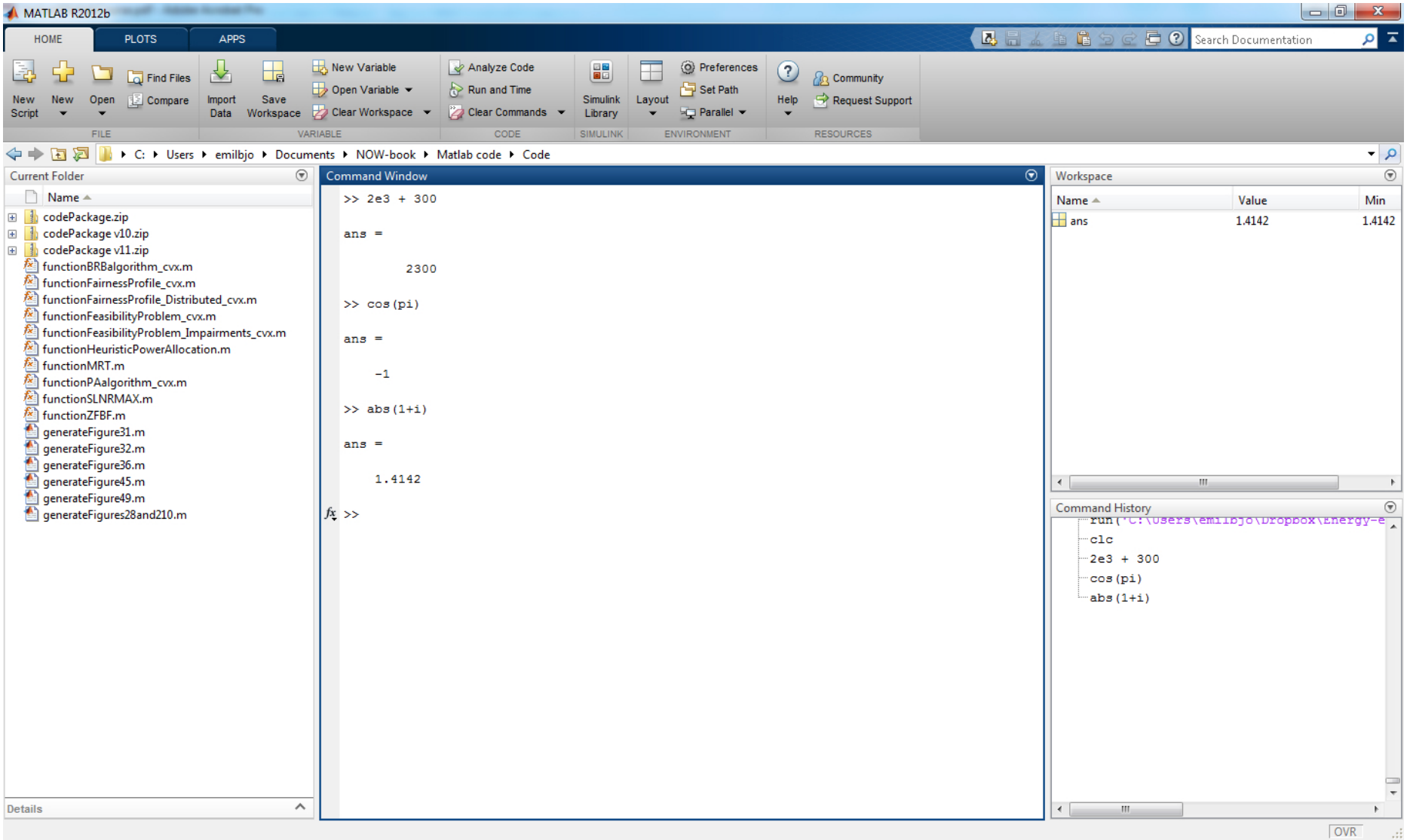
- Data rate per user (bit/s/user)
- Area data rate (bit/s/km^2)
- Energy efficiency (bit/Joule)

Role of MATLAB:

- Test models
- Develop algorithms
- Visualize tradeoffs



MATLAB Interface



MATLAB as Pocket Calculator

Use *Command Window* as a scientific pocket calculator

- Simple numbers: `30`, `pi` (π), `1e2` ($1 \cdot 10^2$)
- Simple operators: `+ - / *`
- Simple functions: cosine ($\cos()$), absolute value ($\mathrm{abs}(\cdot)$)

Examples:

```
>> 2e3 + 300

ans = 2300

>> cos(pi)

ans = -1

>> abs(1+1i)

ans = 1.4142
```

Variables

- A “container” to save values in.
- Has a name and a value.

```
>> a = 5

a = 5

>> b = a + 3

b = 8
```

(That to the right of `=` is computed first, and the result stored in `b`.)

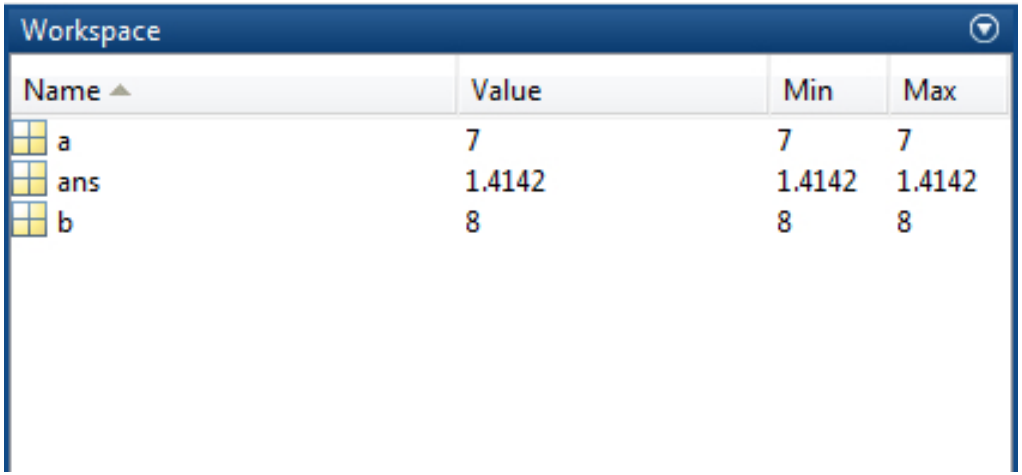
What is the result of:

```
>> a = a + 2

a = 7
```

Workspace

Variables are stored in the “Workspace”, cf., a filing cabinet.



Investigate your workspace

- If you don’t give a variable name: Result is stored in `ans`
- You can click on variables in workspace to find out more.
- You can list all available variables with `>>whos`.

Vectors and Matrices

Vectors and matrices are a fundamental to MATLAB.

- $a = \begin{bmatrix} 4 & 5 & 6 \end{bmatrix}$ is written as `>>a = [4 5 6]` (or `[4, 5, 6]`)
- $b = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$ is written as `>>b = [1; 2; 3]`
- $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ is written as `>>A = [1 2; 3 4]`

These are stored in Workspace — just as any variable:

Vectors and Matrices

- Suppress output from a MATLAB command by semicolon:

```
>> a = [4 5 6];
```

- To get the matrix transpose write `.'`:

```
>> a.'

ans =
      4
      5
      6
```

Generate special matrices and vectors:

- `>>C = eye(2)` yields $C = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$.
- `>>x = 3:6` yields $x = \begin{bmatrix} 3 & 4 & 5 & 6 \end{bmatrix}$.
- `>>y = 2:3:11` yields $y = \begin{bmatrix} 2 & 5 & 8 & 11 \end{bmatrix}$.

Matrix Operations

Original purpose of MATLAB: Matrix operations

- Define matrices:

```
>> A = [1 2; 3 4];
>> B = eye(2);
```

- Compute multiplications:

```
>> A*B

ans =

     1   2
     3   4
```

i.e., $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

(normal matrix multiplication)

```
>> A.*B

ans =

     1   0
     0   4
```

i.e., $\begin{bmatrix} 1 \cdot 1 & 2 \cdot 0 \\ 3 \cdot 0 & 4 \cdot 1 \end{bmatrix}$

(element-wise multiplication)

- Similar: `^2` vs. `.^2`, and `/` vs. `./`

Matrix Operations

There are tons of functions that handle matrices:

- Classic functions: `exp()` `log()` `sin()` `cos()` `tan()`
- Ordering functions: `min()` `max()` `mean()` `sort()`

Some functions work element-wise:

```
>> x = 0:(pi/2):(2*pi)

x = 0    1.5708    3.1416    4.7124    6.2832

>> y = sin(x)

y = 0    1.0000    0.0000   -1.0000   -0.0000
```

Some functions process all elements at once:

```
>> z = max(x)

z = 6.2832
```

Matrix Indexing

How to access specific elements in vectors and matrices?

```
>> y = [0 1 0 -1 0];
>> y(4)

ans = -1

>> A = [3 5 2; 7 8 6];
>> A(1,2)

ans =  5
```

$$A = \begin{bmatrix} 3 & 5 & 2 \\ 7 & 8 & 6 \end{bmatrix}$$

How to find a function?

If you are looking for a function:

- How do you know if it exists in MATLAB?
- “lookfor term” searches the documentation for the string “term”
- Example: `lookfor determinant` to look for the matrix determinant function

How do you know how it works?

- “help command” displays a help text for “command”
- “doc command” gives more thorough information

General documentation:

- “doc” opens up the MATLAB documentation
- “help” gives a list of “toolboxes” (collections of commands organized by usage)

Beyond the Pocket Calculator: Scripts

- A way to perform several commands at once.
- Save some commands in an m-file (the filename must end with `.m`) and run all at once by simply typing the name of the file at the command line.
- `>>edit` start an editor suitable for writing m-files.
- Documentation: Comments are written as `% Comment`

Strong recommendation:

- Always use scripts!
- Easy to reproduce result and write documentation.
- Easy to make small changes and rerun everything.

Example: Script

Lina has run 5 km in 23 min and 15 s.

- She wants to compute the time per km.
- She wants to do the same thing next week.

m-file computeRunPace.m

```
distance = 5; % Distance in km
minutes = 23; % Total time expressed in
seconds = 15; % minutes and seconds

% Compute time per km in minutes:
totalminutes = minutes + seconds/60;
minperkm = totalminutes/distance
```

Scripts vs. Functions

Nature of scripts

- Just a collection of commands.
- Uses MATLAB's general Workspace.
- Can overwrite previous variables (overlapping name).
- Can unintendedly use previous variables (coding error).
- Simplest solution: Begin scripts with `clear`, which empties workspace.

Nature of functions

- Another concept: Have their own local Workspaces.
- Works just like MATLAB's own functions.
- Excellent way to reusing the same code multiple times.

Example: Function

m-file computeRunPace.m

```
function minperkm = computeRunPace(dist, min, s)
% Computes the time per km in minutes, given
% the distance and the total time expressed
% in minutes and seconds.

  totalMinutes = min + s/60;
  minperkm = totalMinutes/dist;
end
```

- `function` — indicates the beginning of a function
- *function name* — should be the same as the m-file name
- *input* — data needed by the function
- *output* — result delivered by the function

Example: Function Execution

```
>>mpkm=computeRunPace(5,23,15)
```

Workspace: MATLAB

m-file computeRunPace.m

```
function minperkm = computeRunPace(dist, min, s)
% Computes the time per km...

  totalMinutes = min + s/60;
  minperkm = totalMinutes/dist;
end
```

Example: Function Execution

```
>>mpkm=computeRunPace(5,23,15)
```

Workspace: MATLAB

m-file computeRunPace.m

```
function minperkm = computeRunPace(dist, min, s)
% Computes the time per km...

  totalMinutes = min + s/60;
  minperkm = totalMinutes/dist;
end
```

Workspace: computeRunPace

```
dist = 5
min = 23
s = 15
```

Example: Function Execution

```
>>mpkm=computeRunPace(5,23,15)

mpkm = 4.65
```

Workspace: MATLAB

```
mpkm = 4.65
```

m-file computeRunPace.m

```
function minperkm = computeRunPace(dist, min, s)
% Computes the time per km...

  totalMinutes = min + s/60;
  minperkm = totalMinutes/dist;
end
```

Workspace: computeRunPace

```
dist = 5
min = 23
s = 15
totalMinutes =
23.25
minperkm = 4.65
```

Example: Function Execution

```
>>mpkm=computeRunPace(5,23,15)

mpkm = 4.65
```

Workspace: MATLAB

```
mpkm = 4.65
```

m-file computeRunPace.m

```
function minperkm = computeRunPace(dist, min, s)
% Computes the time per km...

  totalMinutes = min + s/60;
  minperkm = totalMinutes/dist;
end
```

Combine Scripts and Functions

Functions

- Create functions whenever a certain “algorithm” or multi-row computation takes place more than once
- Built-in MATLAB functions are written in this way (write `type functionName` to see)

Scripts

- Define input values
- Call different functions
- Process and visualize output from functions

Visualization

Suppose we want to plot (visualize) the mathematical function $y = \sin(x)$ for $0 \leq x \leq 10$:

m-file plotSine.m

```
x = 0:0.1:10; % The x for which y should be computed
y = sin(x);

figure; % Open a new figure ready for plotting
plot(x,y) % Plot y as a function of x
xlabel('x') % Give a name to the horizontal axis
ylabel('y = sin(x)') % Give a name to the vertical axis
title('My first plot') % Give a name to the whole figure
```

Visualization: Many types

Many functions for plotting data:

- 2D line graphs: `plot`, `semilogx` (horizontal log-scale)
- 2D bar graphs: `bar`, `histogram`
- 3D line graphs: `plot3`
- 3D bar and mesh graphs: `bar3`, `mesh`
- 3D surface graphs: `surf`, `sphere`, `ellipsoid`

Use `help` to read more!

Adapt plots:

- Almost everything can be tailored.
- Use the “Property Editor” in menu “View” of a figure

Control Structures

Some “behaviors” depend strongly on the input:

- Does your bank account have enough money or not?

Some pieces of code is repeated:

- Do you need to run the same lines of code multiple times?
- Do you know how many times in advance?

MATLAB has several *control structures*:

- if statements
- while loops
- for loops

These are similar to other programming languages.

if Clauses

General syntax:

```
if condition
  % statements/commands if condition is true
else
  % statements/commands if condition is false
end
```

Writing conditions using logics

- Use operators such as: `> >= == && || ~= < <=`
- Suppose `savings` is a variable with the amount on your bank account.
- Examples: `savings >= 10000 && savings <= 20000`

Example: if Clauses

Example

A bank account has 2% interest on savings and charges 14% interest for credits. Write a function to compute the interest given an amount.

m-file computeBankInterest.m

```
function interest = computeBankInterest(amount)
% Computes annual interest for a given amount

if amount >= 0
  interest = 0.02*amount;
else
  interest = 0.14*amount;
end

end
```

Guard Towards Errors

If statements can be used to avoid unexpected behaviors

- Example: `computeBankInterest(amount)` cannot handle complex numbers
- Can be checked and handled as:

```
if imag(amount) ~= 0
  error('There is no imaginary money!');
end
```

- `imag()` gives the imaginary part of a scalar/vector/matrix
- `error()` displays an error message
- Text strings are written as `'message'`
- Alternative: `disp()` displays a non-error-related message

while Loops

- Repeat similar computations *while* a condition is fulfilled
  - Condition is checked only at beginning of each loop
  - Be sure that the condition will eventually be false — otherwise the loop runs forever!

- General syntax:

```
while condition
  % statement/commands to be repeated
end
```

Example: while Loops

Example

Suppose you have borrowed 1 million kr from the bank. The bank charges 0.25% interest per month. You amortize 5,000 kr per month. How many months will it take to repay the loan?

m-file predictLoan.m

```
currentLoan = 1e6; % The initial loan is 1,000,000 kr
monthlyPayment = 5000; % You pay 5000 kr each month
montlyInterest = 0.0025; % The bank charges 0.25% per month
monthNumber = 0; % Keep track of month number

while currentLoan >= 0
  currentLoan = currentLoan + currentLoan*montlyInterest; %Apply interest rate
  currentLoan = currentLoan - monthlyPayment; %Reduce loan by monthly payment
  monthNumber = monthNumber + 1;
end

% monthNumber will now contain the month when you have repaid your loan
% Be sure that monthlyPayment > currentLoan*montlyInterest, otherwise it never stops!
```

for Loops

- If you know how many time to repeat commands
  - More compact to use `for`-loops instead of `while`

- General syntax:

```
for var = vector with values
  % statement/commands to be repeated
end
```

Example: for Loops

Example

Suppose you start saving 500 kr per month when your kid is born. The monthly interest is 0.17% (2% per year). How much will the kid have at the age of 18?

m-file predictSavings.m

```
currentSaving = 0; % Bank account is empty in advance
monthlySaving = 500; % You save 500 kr per month
montlyInterest = 0.0017; % The bank interest is 0.17% per month

numberOfMonths = 12*18; % Compute number of months before turning 18

for index = 1:numberOfMonths
  currentSaving = currentSaving + currentSaving*montlyInterest; %Apply interest rate
  currentSaving = currentSaving + monthlySaving; % Add monthly saving
end

% currentSaving will now contain the savings at the age of 18
```

Summary

- MATLAB is useful in many different computations
- Standard tool at universities and many companies — more than 1 million users
- Choose variable names carefully — and write comments
- Use scripts and functions, it *will save you time*
- Control statements:
  - `if` statements — do different things depending on a condition
  - `for` loops — repeat computations for a predetermined set of values
  - `while` loops — repeat computations until a condition is no longer fulfilled
- Make use of the help system to extend your knowledge!!!

Good luck with the course!

Have fun with MATLAB!

Learn by exploration!