

# MATLAB Introductory Course

## Computer Exercise Session 2

This version: January 11, 2010

This computer session is mostly about programming in MATLAB. The most basic concepts are introduced, and a suggestion is given on how to tackle the kind of programming tasks that are usually solved in MATLAB. If you are used to programming, most of the material will probably be familiar to you.

### 1 Preparatory Exercises

1. Look through the different parts of computer exercise 1, and make sure you have understood them.
2. Read through section 2 and do the exercise.

### 2 Documentation

Already in computer exercise 1 we pointed out the importance of commenting your code. Depending on the purpose of the code, you might choose different levels of documentation:

1. You might want to be able to use the function later, and need to understand what it does.
2. A friend will use the function, and needs to understand how it works. He or she would also like the result to be presented in a well-structured way.

3. You would like to convince your boss that your function is so useful that it is necessary to purchase a MATLAB license for the company. The code should be not only well-documented, but should also present the result in a nice and convincing way.

Substitute “lab assistant” for “friend”, and you will get a hint on the minimum level of ambition you need for the documentation on the project exercise in this course.

To obtain a nice code that is easy to read, bear in mind the following:

- Write your code in a logical order: “First we do this, then we do that.” Use lines starting with %% to group different parts of the code, and use comments to explain the purpose of each part.
- Use intuitive names for your variables. The following code performs the same computation as in the `loancomp` example from session 1, but is harder to understand:

```
function y = test(x1,x2)
% A test
a = 0.01;
b = 0.1;
y = x1 + x1*a - x2*b;
```

- Write help texts and explanations for the arguments and return variables. The help text is written as comments in the lines immediately following the function declaration. This is what is printed if you type `help filename` for your m-file. (Try it out on `loancomp`.)
- Do not forget to change the comments when you change the code. Incorrect comments are worse than no comments at all, and just makes the code hard to understand.

**Exercise:** Add comments and help texts to `curveplot` from computer session 1 (if you haven’t completed the function already, do that first).

### 3 Control structures

#### Conditions — if clauses

Often you would like to do different things depending on the value of some variable. In the loan example from session 1, you might for instance want to check whether the debt is settled. In MATLAB (like in most other programming languages), there is a control structure called `if` clauses. The following example show how to use it to check if the debt is settled, and in that case set the debt to zero:

```
function new_debt = loancomp(debt,salary,rate,paymentsize)
% new_debt = loancomp(debt,salary,rate,paymentsize)
% Computes the debt after this month.
% Arguments: debt, salary, rate of interest, and
%           percentage of salary to pay.

new_debt = debt + debt*rate - paymentsize*salary;

% Test if the debt is settled:
if new_debt <= 0
    new_debt = 0;
    disp('The debt is settled!')
else
    disp('The debt is not yet settled.')
end
```

Three keywords: `if`, `else` and `end`. Following `if`, we find our condition: `new_debt<=0` (`<=` means less than or equal to). In this example, the condition consists of an inequality, but you could also check whether two variables are equal by `==`, or if they are not equal by `~=`. If the condition is true, the

commands between `if` and `else` are executed. Otherwise the commands between `else` and `end` are executed. In general, we can write

```
if condition
    commands
else
    commands
end
```

MATLAB interprets nonzero values as true and 0 as false. Hence, the condition might be an expression that is evaluated to something which is either nonzero or zero. Use `help relop`, the MATLAB documentation, or [1] to find the relational operators of MATLAB.

.....

#### Error handling

Sometimes the `if` clause is used to find an error in the input. In the example above we might want to check that the rate of interest is positive. Otherwise we do not want to compute anything, but just print an error message and exit the function. This is obtained with the `error` command. Add the following test before `new_debt` is computed:

```
% Test if the rate of interest is positive:
if rate<0
    error('The rate of interest must be positive!')
end
```

The rest of the function is left unchanged. Now test what happens if you enter a negative rate of interest.

The command `warning` can be used to warn about minor issues without interrupting the function.

**Exercise:** Extend your function `curveplot` so that it checks that `xmin` is less than `xmax`. If the interval is given in another way, you could either print an error message and exit the function, or you could swap `xmin` and `xmax` and continue. In any case, a warning message should be given.

## Repetitions

As previously mentioned, often one would like to run the same code (at least approximately) a number of times. For this, there are better ways than just repeating the same code over and over again.

### for loops

Suppose that Pelle would like to know his debt, not only in one month, but in, for instance 12 months. Instead of running the function 12 times with different arguments, we can use a `for` loop. We also introduce a new input argument: number of months.

```
function new_debt = loancomp(debt,salary,rate,...
    paymentsize,months)
% new_debt = loancomp(debt,salary,rate,...
%     paymentsize,months)
% Computes the debt after a number of monts.
% Arguments: debt, salary, rate of interest,
%             percentage of salary to pay and
%             number of months.

% Test if the rate of interest is positive:
```

```
if rate<0
    error('The rate of interest must be positive!')
end

% Compute the new debt:
for t = 1:months
    debt = debt + debt*rate - paymentsize*salary;
end
new_debt = debt;

% Test if the debt is settled:
if new_debt <= 0
    new_debt = 0;
    disp('The debt is settled!')
else
    disp('The debt is not yet settled.')
end
```

The keywords `for` and `end` enclose the repetitive structure. The variable `t` will, step by step, take the values given after the `=` sign, and for each value, the lines between `for` and `end` are executed. We recognize the colon notation, `:`, from computer session 1. The first time the `for` loop is run, `t` will be 1, the second time `t` will be 2 etc., until `t` has reached the value of `months` (e.g. 12). Every time, `debt` gets a new value. Try removing the semicolon after the computation to see the debts for each month.

In the example above, `t` is only used to keep track of the number of iterations, but it can also be used inside the loop. For instance, let us make `debt` a vector, where `debt(t)` gives the debt for month `t`. Before the `for` loop, add

```
debt = [debt; zeros(months,1)];
```

to create a vector where we can save the size of the debt month by month. The line inside the for loop is replaced by

```
debt(t+1) = debt(t) + debt(t)*rate - paymentsize*salary;
```

Otherwise, the function is left as it is.

Try this version of the function, and plot a figure of how the debt changes over the months. If you skip the semicolon in the loop, you can also see how the values of debt are entered one by one. Printouts like that may be useful when looking for bugs in the code.

**Exercise:** Extend and modify the function `curveplot`, so that it plots the curves  $f(x) = 1 - \frac{e^{-tx}}{\sqrt{1-t^2}} \sin(x\sqrt{1-t^2} + \arccost)$  for different values of  $t$ . Plot the curves in the same figure. You can let  $t$  vary between 0.1 and 0.9 with a step size of 0.2. You are supposed to use a for-loop to handle varying  $t$  inside your function.

## while

You do not always know in advances how many times to repeat a computation. For instance, Pelle might want to find out after how many months the debt will be settled. For this, one can use a while loop. `while` checks if a condition is satisfied (in the same way as `if`) and executes a number of commands if it is. Then the condition is checked again, and the whole procedure is repeated until the condition is not satisfied anymore. We can modify our example to compute at which month the debt is settled using `while`:

```
function finalmonth = loancomp(debt,salary,rate,paymentsize)
% finalmonth = loancomp(debt,salary,rate,paymentsize)
% Computes at which month the debt is settled
% Arguments: debt, salary, rate of interest, and
```

```
%           percentage of salary to pay

% Test if the rate of interest is positive:
if rate<0
    error('The rate of interest must be positive!')
end

month_no = 1;
while debt(month_no)>0
    % Compute the new debt:
    debt(month_no+1) = debt(month_no) + ...
        debt(month_no)*rate - paymentsize*salary;
    month_no = month_no+1;
end
finalmonth = month_no;
```

When using a while loop, it is important to make sure that the condition is not always satisfied. In that case, the function would get stuck in an infinite loop<sup>1</sup>. This should be checked before entering the loop. In our example, the loop will be infinite if the debt does not decrease every month. We can check this just like we test if the rate of interest is positive. Give a condition which ensures that the loop will be finite (and the debt will be settled):

.....

Modify the code so that we do not risk an infinite loop. Apart from the condition above, we could also allow a maximum number of iterations, i.e., we can require that `month_no` does not exceed 200 (for instance). The upper limit could also be an argument of the function.

Summarize, or give an example of when to use for-loops be used, and when it is better to use while-loops?

<sup>1</sup>If this would occur, you can stop the execution by pressing Ctrl-c.

.....

**Exercise:** The maximum of the function  $f(x) = 1 - \frac{e^{-tx}}{\sqrt{1-t^2}} \sin(x\sqrt{1-t^2} + \arccos t)$  depends on the value of  $t$ . Suppose that we are only interested in positive  $x$  values. If we want  $t$  as small as possible, but still would like a maximum that is less than, say, 1.2, we can test different  $t$  values in a `while` loop. Modify the function so that it finds the smallest  $t$  value for a certain maximum. Make sure that you avoid infinite loops.

In `loancomp`, we compute both the final month and the debt for each month. Since function variables are local, we cannot access the debt values from Workspace. To do that, we need to give two return values:

```
function [finalmonth,debtchange] = loancomp(...
    debt,salary,rate,paymentsize)
```

We must give `debtchange` the desired value somewhere in the function. The function can then be called with two return values, e.g.

```
>> [m, d] = loancomp(100000,25000,0.01,0.1)
```

Modify the function so that it takes two return values.

## 4 Example

The example below is taken from a linear algebra course, but why not apply our MATLAB skills to it?

A local car rental company has two offices, one in Linköping and one in Norrköping. It turns out that during a month, 70% of the cars rented in Linköping are also returned there, while 30% are returned in Norrköping. Of the cars rented in Norrköping, 40% are

returned there and 60% in Linköping. The company has 15 cars, each of which is rented once a month. How many should be placed in Linköping and Norrköping, respectively. What if more cars are purchased or if the percentages are changed?

The problem has an analytical solution, but we will solve it through simulation, since it could be interesting to see how the car distribution evolves over time.

**Exercise** (remember to document/comment your code):

1. Start with, e.g., 1 car in Linköping and 14 in Norrköping at month 1. What happens in month 2? 3? 4? Write a script simulating a number of months and test it. Plot the result.
2. Test the script for different numbers of cars and months.
3. Rewrite your script as a function. Input: number of cars and months. Output: A nice plot with describing text.
4. Generalize the function so that the percentages are given as arguments. Check that the percentages add up to 100.
5. Extra exercise: Include several towns. You need to think about what data format is appropriate.
6. To be precise, only integer solutions are interesting (you cannot return half a car in Linköping...). Include that in your function.

The division into smaller subproblems above illustrates a possible strategy for tackling the type of problems often solved in MATLAB. Start by solving an easy subproblem, where the answer can easily be checked. Visualize the answer with appropriate figures and document what you have done. The subproblem can then be gradually extended, until the entire problem is solved.

One common method of programming is called stepwise refinement. The basic concept here is to divide the problem into subproblems, that can be

solved one at a time. Hopefully, each of the subproblems is easier to solve than the original problem. If they cannot be solved directly, you can continue by dividing them into even smaller subproblems etc.

The wide variety of existing programming methods lies far beyond the scope of this course, but if you feel comfortable with a certain method, you can (probably) use it successfully in MATLAB.

## 5 Going further

In this section, the recommendation is to study general programming in some more detail. You can also investigate how to write efficient code, and/or how to use object-oriented programming in MATLAB. Use [1] or the MATLAB documentation.

### 5.1 General programming

Chapters 9 and 10 in [1] deal with functions.

**Suggestions for exercises:** 45, 48, 49, 51, 52, 53

You can also extend `loancomp` with features that you find useful. Some ideas could be to make fancier plots of different things, to give the rate of interest per year instead of per month, to account for changes in the salary once a year etc.

In “Getting started” in the MATLAB documentation there is a section on programming. You can also read about “Function functions” under “Mathematics”. Under “Programming” you can find “Programming tips”.

### 5.2 Efficient code

In [1], some different ways of making your code more efficient are explained.

Try to figure out whether vectorization can make the example in Section 4 more efficient. Exercise 48 in the book brings up some different ways of solving the problem. Which is the fastest method?

In the MATLAB documentation you can find a section on “Improving Performance and Memory Usage” under “Programming”

### 5.3 Object orientation

MATLAB has some support for object-oriented programming. Chapters 11 and 12 in [1] describe further data-types and object orientation. See also “Classes and Objects” under “Programming” in the MATLAB documentation.

## References

[1] F. Gustafsson and N. Bergman, *Matlab for Engineers Explained*, Springer Verlag, 2003.