TSRT04: Introduction in MATLAB Computer Exercise/Lesson 2

This version: March 20, 2015

This computer session is mostly about programming in MATLAB. The most basic programming 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, but it is still important to learn the syntax.

Note: The purpose of this computer exercise/lesson is that you should practice to learn the basics of MATLAB. There are plenty of questions in this document and we suggest that you write down the answers and your own reflections, so it is easy for you to go back. However, you don't have the present the results to anyone. The teachers are here to answer questions and give you feedback on your code. So don't be afraid — there are no stupid questions!

1 Preparatory Exercises

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

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, or that your idea deserves to become a patent. The code should be not only well-documented, but should also present the result in a nice and convincing way.

Substitute "friend" with "examiner" or "lab assistant", and you will get a hint on the minimum level of ambition you need for the documentation 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 in Section 7 of from computer exercise/lesson 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 (`function y = ...`). 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 harder to understand.

Exercise 1. Add comments and help texts to `curveplot` from computer session 1 (if you haven't completed the function already, do that first). Check with `help curveplot` that it looks alright.

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

new_debt = debt + debt*interest - 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 written between `if` and `else` are executed. Otherwise the commands between `else` and `end` are executed. In general, we can write the syntax as

```
if condition
  % statements/commands if condition is true
else
  % statements/commands if condition is false
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` to find the relational operators available in MATLAB.

Take the opportunity to try a few different conditions:

- Is $\sqrt{3}$ larger than 1.5? .....
- Is $\sin^2(t) + \cos^2(t)$ equal to 1 (for an arbitrary t)? What does it tell us? .....
- Check (with one expression) if e^2 is larger than 6 but smaller than 7. .....

Error handling

Sometimes `if` clauses are 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:

```
% Verify that the rate of interest is positive:
if interest<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.

Write an if-clause that checks if the monthly increase in dept, `debt*interest`, is larger than the monthly payment `paymentsize*salary`. If this is true, give a warning that the pay too little.

```
.....................................................................

.....................................................................

.....................................................................
```

Exercise 2. 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 after displaying a warning message. Choose yourself which one to implement.

Code Repetitions

As previously mentioned, one often 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 Emma would like to know her 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,interest,...
    paymentsize,months)
% new_debt = loancomp(debt,salary,interest,...
%      paymentsize,months)
% Computes the debt after a number of months.
% Arguments: debt, salary, rate of interest,
%            percentage of salary to pay and
%            number of months.

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

% Compute the new debt:
for t = 1:months
    debt = debt + debt*interest - 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 exercise/lesson 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. Verify by 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, we might want to store the debt month by month and provide it as an output. Let us therefore make `debt` a vector, where `debt(t)` gives the debt at the end of 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)*interest - 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 these are very useful when looking for bugs in the code!

Exercise 3. Extend and modify the function `curveplot`, so that it also plots the curves $f_2(x) = 1 - \frac{e^{-tx}}{\sqrt{1-t^2}} \sin(x\sqrt{1-t^2} + \arccos t)$ for different values of t in the interval $0 \leq t < 1$. Plot the new 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 loops

You do not always know in advances how many times to repeat a computation. For instance, Emma 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,interest,paymentsize)
% finalmonth = loancomp(debt,salary,interest,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 interest<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)*interest - 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.[1] 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 as described earlier; that is, check if monthly payment `paymentsize*salary` is larger than the monthly increase due to interest `debt*interest`. Write this so that the while-loop is only initiated if it is satisfied (i.e., there is no risk of an infinite loop), otherwise an error is displayed.

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.

[1] If this would occur, you can stop the execution by pressing Ctrl-c.

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

Exercise 4. The maximum of the function

$$f(x) = 1 - \frac{e^{-tx}}{\sqrt{1-t^2}} \sin\left(x\sqrt{1-t^2} + \arccos(t)\right)$$

depends on the value of $t \geq 0$. Suppose that we are only interested in positive x values. We want to make t as small as possible, but still let the maximum of $f(x)$ be 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.

A hint is that the maximum of $f(x)$ gets smaller when t is larger. If you start with $t = 0$ and increase t little by little, then the maximum should eventually drop below 1.2.

Exercise 5. 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 the main Workspace. To access the changes in the debt, we need to give two return values:

```
function [finalmonth,debtchange] = loancomp(...
    debt,salary,interest,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 Simulating a simple engineering problem

The problem formulation below is taken from a linear algebra course. It is not formulated to be solved in MATLAB, but why not apply our MATLAB skills to analyze 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 and reaches the analytical solution.

Exercise 6. (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 in each city (at the beginning of the first month) and number of months to simulate. Output: A vector/matrix with the results and a nice plot with describing text.
4. Generalize the function so that the percentages of car returns are given as arguments. Check that the percentages add up to 100, otherwise there should be an error.
5. Include a third town: Nyköping. Is there a way to rewrite the code so that it is easy to add additional towns? Can the number of towns even be an input parameter?
6. To be precise, only integer solutions are interesting (you cannot return half a car in Linköping and the other half in Norrköping). Modify your code to make sure that the number of cars is always an integer at the end of each month (e.g., using the function the `round`). There is a risk that the number of cars can increase/decrease in the rounding (particularly when having three cities) so you should use control structures to check if this occurs and then prevent it from happening.

The division into smaller subproblems above illustrates a common strategy for tackling the type of engineering problems that are 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 and the results have been visualized.

This approach resembles a common method of programming: 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.

Finally, you should open your scripts and functions in the MATLAB editor and look at the colored square in the upper right corner of the text field. What color does it have?

- Green: No warnings or errors found.
- Yellow: Warnings found.
- Red: Errors and warnings found.

Red light typically means that there is a syntax error in the code, thus it cannot be run. Green light means that you have done a terrific job! In most cases you will find a yellow light, which means that MATLAB has recommendations on how you can rewrite your code to be more efficient or get more precise results. Each recommendation can be spotted in the scroll bar at the right-hand side of the editor. Go through them all and see if you can make the appropriate changes. You will hopefully get a green light in the end!

5 For-loops versus matrix/vector computations

Since MATLAB is originally built for matrix computations, this is something that MATLAB does quickly and efficiently. For-loops has, on the other hand, traditionally been slow in MATLAB, even if it has been improved in recent years. There are many cases when matrix/vector operations can replace for-loops.

Define two large matrices A and B:

```
>> A = rand(1000,1000);
>> B = rand(1000,1000);
```

Compare the speed of computing the matrix addition $A + B$ as a matrix operation

```
>> C = A + B;
```

and by for-loops:

```
>> for row = 1:1000
>>     for col = 1:1000
>>         D(row,col) = A(row,col) + B(row,col);
>>     end
>> end
```

Which of the two implementations is the fastest?

.....

You can compute the time it takes to run your code by putting `tic` in front of the code and end by `toc`. This measures the time it takes from `tic` to `toc`. Which numbers do you get for the pieces of code above? (Remember to clear your workspace in between computations).

.....

.....

If you have to go through the elements of a matrix with for-loops, a good first step is to preallocate memory. This means adding the row

```
>> D = zeros(1000,1000);
```

in front of the for-loops, so that MATLAB knows how large D should be and doesn't have to change its dimension in every iteration of the for-loop.

How does this affect the run time?

.....

6 Homework exercises: Week 1-2

A set of homework exercises were given at the end of the material to computer exercise/lesson 1. Make sure to finish these exercises!