Computational PDE's II
Prof. Carsten Carstensen
Hella Rabus *Humboldt-Universität zu Berlin, Germany*

Matlab and Octave: Quick Introduction and Examples

1 Basics

1.1 Syntax and m-files

There is a shell where commands can be written in. All commands must either be built-in commands, functions, names of variables, or names of m-files within the current directory. Be aware, that the evaluation of commands is case-sensitive.

During the course you are supposed to write programs, more precisely m-files. An m-file called **Example.m** can be executed by typing `Example` on the prompt of Matlab (or Octave, respectively).
The program starts with the first line in the m-file.

Each line of an m-file is one command. If you write "**...**" at the end of a line, the command will be continued in the next line. If the command produces a result, it will be shown on the terminal. A semicolon behind a command suppresses the output.
If the output is more than one sceen long, Octave will sent it to a paging programm such as `more` or `less`.

```
h=3+4
```

produces output:

```
  h  =
        7
```

```
h=3+4;
```

or

```
h=3+...
 4;
```

produces no output

1.2 Conditional branching

You can use `if` to make decisions. An **if** statement has to be closed with `end`. Logical operators are: `<, <=, >, >=, ==, ~=`. Example:

```
if t<0
     [some code]
elseif t==0
     [some code]
else
     [some code]
end
```

1.3 Loops

There are two types of loops, `for` loops and `while` loops. Every loop has to be closed with an `end`.
Try the following examples

```
    s=0;                                        s=0;
    for i=1:4                                   for i=1:4
       s=s+i*i;                                    s=s+i*i;
    end                                         end;

    s=0;                                        s=2;
    for i=[1;4;6;3]                             while s<20
       s=s*i+i;                                    s=s*s;
    end                                         end;
```

2 Matrix and vector operations

2.1 Creating a matrix or a vector

It is pretty easy to define vectors and matrices and to use matrix operations. Some examples:

```
    >>A= [1 2 3; 4 5 6;7 8 9]                   >>b=[1;2;3]
    >>A= [1 2 3                                 >>c=[4,5,6]
          4 5 6                                 >>A=[b
          7 8 9]                                     c'
                                                     7 8 9]

    >>A=diag([1;5;9])+diag([2;3],1)+...
        diag([4;8],-1) + diag(3,2) +...
        diag(7,-2)
```

Note: The resulting matrix `A` in these examples is always the same. Entries in one row must be separated by a comma or a space. Rows are separated by a semicolon or a linebreak. `A'` denotes the transposed matrix of `A`. Other commands to create a matrix are: `diag, zeros, ones`.

Vectors are matrices with dimension (n,1) or (1,n), they can be created as follows

```
>>x=[4:2:14]

x =

     4     6     8    10    12    14
```

2.2 Operations

`>>C=A\B` means $C = A^{-1}B$, i.e. $AC = B$
`>>C=B/A` means $C = BA^{-1}$, i.e. $CA = B$
This holds for A being a regular matrix.

In the case that A is not regular, `C` is still returned as in the following equations. It is important to know, that the equation may not be solved exactly for A being a singular matrix. If the system of equations is underdetermined, the solution is not unique, a matrix C is returned without any warning:
`>>C=A\B` min $\|AC - B\|$ or one solution
`>>C=B/A` min $\|CA - B\|$ or one solution

`>>C=B+A`
`>>C=B*A`
`>>C=B.*A` means $c_{ij} = b_{ij}a_{ij}$
Almost every mathematical operator can be altered to a componentwise operation of matrices by putting a dot "." in front.
Some interesting functions for matrices are:

- `eig` for computing eigenvalues
- `lu` for computing an LU-decomposition
- `det` for computing the determinant of a matrix
- `inv` for inverting a matrix

If you do not know how to use a command, you may need some help. Some help is provided if you just type `help` on the command line, followed by the command. E.g.:
```
>>help lu
```

2.3 Accessing the elements of a matrix

The following examples are self explanatory.

```
>> I=[1,2]                              >>I(2)
I =                                     ans =
     1     2                                   2

>>A= [1 2 3                             >>A(2,3)
      4 5 6                             ans =
      7 8 9];                                  6
```

```
>> A(3,:)                                       >> A(1,2:3)
ans =                                           ans =
      7     8     9                                   2     3

>> A(I,:)
ans =
       1     2     3
       4     5     6

>>A([1,2],[2,3])=A([1,2],[2,3])+eye(2,2)
A =
     1 3 3
     4 5 7
     7 8 9
```

Exercise Write a program in the shell that computes the solution x of

$$\begin{aligned} Ax &= b \\ b &= (1, 8, 21, 40, 65, 96, 42) \\ \mathbb{R}^{7\times 7} \in A &= \begin{cases} a_{ii} = i & \forall i \\ a_{i\,i+1} = 2 * (i-1) + 1 & \forall i \\ 0 & \text{else} \end{cases} \end{aligned}$$

3 Functions

For complex or repeatedly used calculations, you can define functions. A function is either a block at the end of an m-file, or an m-file itself represents a function if it begins with a declaration of a function. A function is defined as follows:

```
function [output1,output2,...]=FUNCTIONNAME(input1,input2,...)

    [some Calculation to define the values of the output variables]
```

If the function is at the end of a m-file, it can be called from within the m-file with its `FUNCTIONNAME`. If the whole m-file represents a function, then the file-name is the name of the function, regardless of what you wrote in the declaration of the function as `FUNCTIONNAME`.

Exercise Create a function which computes: $f(x_1, x_2) = 3\sin(x_1) * \cos(4x_2)$. This function should be in an independent m-file.

4 Graphical outputs

Commands which create graphics out of your data are for example `plot`,`mesh` and `surf`. Please look at the Octave online manual, the Matlab help or type `help` followed by the command on the prompt to inform yourself how to use these commands.

Exercise Plot the function from the latter exercise as a 2D-surface over the plane $[0, 20]^2$.

5 Further useful functions and commands

- `round`: for rounding real numbers
- `sqrt`: for the square root of a real number
- `sign`: for the signum function
- `imag`: for the imaginary part of a complex number
- `real`: for the real part of a complex number
- `save`: for saving variables to disk
- `load`: for loading data from disk
- `Ctrl` and `c`: for stopping the current calculation
- `clear`: for deleting variables
- `close`: for closing figure windows