An introduction to MATLAB in

COMPUTATIONAL SCIENCE, 11 SEPTEMBER 2017

1 Introduction

MATLAB is a programming language (or, more accurately, a scripting language, with core functions that are pre-compiled) for the *numerical* solution of numerical mathematical problems (problems that arise in exact sciences and technical sciences). It relies on very powerful, reliable, and robust standard commands[1] and powerful subroutines. MATLAB is user friendly and allows easy and quick programming with transparant coding. MATLAB is very suitable for quickly testing ideas for new algorithms. Nevertheless, MATLAB's results are very reliable: its core functions are based on subroutines (from LAPACK) that have been written by leading computational scientists and incorporate the 'state of the art' of Computational Science. If improved algorithms become available, you can be sure that they will be incorporated in the next version of MATLAB. The same can be said (be it in somewhat less degree) about the subroutines (function subroutines) that are provided with the MATLAB package. Moreover, the core function have been optimised for the processor on which MATLAB is running.

User friendliness will be somewhat at the costs of efficiency. An algorithm implemented in C++ or FORTRAN will be faster than an implementation in MATLAB. Moreover, the factor by which MATLAB is slower depends on the type of command. Nevertheless, MATLAB is very efficient, and a great number of research institutes that have a lot of computational work on huge data sets, use MATLAB, even for their routine computations.

In contrast to programming languages as C++ and FORTRAN, there is no need in MATLAB to specify the type of quantities (booleans, integers, reals, complex numbers) that you are using and, the sizes of arrays need not to be assigned in advance: MATLAB will detect these things during computation. This is one of the reasons why MATLAB is so user friendly and at the same time it also the reason why MATLAB is less efficient.

MATLAB is based on manipulations with arrays of numbers. An $n \times k$ matrix is a `n` by `k` array, an n-vector is a `n` by `1` array. The product $\mathbf{AB}$ of an $n \times k$ matrix $\mathbf{A}$ an a $k \times \ell$ matrix $\mathbf{B}$ is an $n \times \ell$ matrix and is obtained in MATLAB by running the command `A*B`.[2] `A'` is the matrix $\mathbf{A}^* \equiv \mathbf{A}^{\mathrm{H}} \equiv \bar{\mathbf{A}}^{\mathrm{T}}$, the $k \times n$ complex conjugated transposed of $\mathbf{A}$. If $\mathbf{x} = (x_1, \ldots, x_n)^{\mathrm{T}}$ and $\mathbf{y} = (y_1, \ldots, y_n)^{\mathrm{T}}$ are column vectors of dimension n and $(\mathbf{x}, \mathbf{y}) \equiv \sum_{j=1}^{n} x_j \bar{y}_j$ is the (complex) inner product of $\mathbf{x}$ and $\mathbf{y}$, and $\mathbf{x}$ and $\mathbf{y}$ have been entered in MATLAB as `x`, and `y` respectively (as `n` by `1` arrays) then the inner product can be computed in MATLAB by the command `y'*x`. (Note that the inner product can be obtained as the product of matrices. In particular, note that the sizes in the command `y'*x` are consistent with the rules of matrix multiplication.[3] Note also that `x'*y;` may lead to another value in case of complex entries.)

The assignments below are meant as an illustration of the above remarks and also serve as an easy introduction to MATLAB programming. They provide hints to speed up subroutines for problems from Numerical Linear Algebra.

If this is your first introduction to MATLAB, then, please, read the tutorial first. Do not hesitate to use MATLAB's `help` options to obtain information on individual commands. The `type` option may be helpful as well. For instance, running `help lu` explains how the L and U factors of a so-called LU-decomposition of a matrix can be obtained using MATLAB. It also explains what an LU-decomposition is. With `type pcg`, you not only learn what PCG is and what is does, but is also displays the PCG code. Test this: run the commands `help lu`, `help pcg` and `type pcg`. What do you obtain if you run `type lu`?

Assignment 1.

Enter the matrix $\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 10 \end{bmatrix}$ and the vector $\mathbf{b} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$. Use MATLAB to solve $\mathbf{x}$ from the equation $\mathbf{Ax} = \mathbf{b}$. Replace the $(3,3)$ coefficient of $\mathbf{A}$ by 9 and again solve $\mathbf{x}$ from $\mathbf{Ax} = \mathbf{b}$. Can you understand MATLAB's result and message? Now, run the command `lu(A)`, as well as the first three commands that are mentioned in "`help lu`". Can you understand

[1] If, for instance, `A` is a numerical $n \times n$ matrix and `b` is a numerical n-vector then the MATLAB command `x=A\b` returns the numerical solution of the matrix-vector equation $\mathbf{Ax} = \mathbf{b}$: this simple looking command incorporates a complete, robust, and reliable solution procedure.

[2] Here, $\mathbf{A}$ and $\mathbf{B}$ have been entered in MATLAB as `A` and `B` respectively.

[3] In a language as C++, 0 is the lowest array-index, in MATLAB that is 1. For instance, with the command `x=0:0.1:1`, `x` turns into an 1 by 11 array with `x(1)=0`, `x(2)=0.1`, `x(3)=0.2`, ..., that is, `x(i)=(i-1)*0.1` for `i=1,2,...,11`.

the results? Now, replace $\mathbf{A}$ by a 3×2 matrix that consists of the first and second column of the original $\mathbf{A}$. What do you expect from the command `x=A\b`? Are the results in line with your expectation? What do you expect from the command `x=A*b`?

The M-files Assignment2.m ... Assignment4.m can be used for the following three assignments.[4]

Assignment 2.

In contrast to programming languages as C++ and FORTRAN, there is no need to assign memory location in MATLAB; MATLAB even does not have an "official" way for doing that. [5] In the M-files Assignment2.m it can be observed that assigning memory can lead to dramatic differences in computational time. For instance, consecutively take the values `10^4`, `5*10^4`, `10^6`, and so on, for `N`. What is the effect of the command `clear x` in this file? Is there a difference between $\mathbf{y}$ being a vector of integers, or a vector of real number (multiply, for instance, $\mathbf{y}$ by π, `y=y*pi;`), or of complex numbers (add the complex number i to $\mathbf{y}$: `ii=sqrt(-1); y=y+ii;`)?

Assignment 3.

Use Assignment3.m to check whether the claim that MATLAB is faster by performing vector operations as $\mathbf{x} = \mathbf{y} + \mathbf{z}$ as vector operations, that is, as

```
x=y+z;
```

than by loping:

```
for j=1:n, x(j)=y(j)+z(j); end
```

Assignment 4.

An array of function-values can be computed in MATLAB in different ways. For instance, in order to plot the graph of $x \rightsquigarrow \sin^2(x)$ for $x \in [0, 2\pi]$, you can proceed as follows.

Compute an array

```
x=0:0.01:1; x=2*x*pi;
```

of x-values and compute the associated array `f` of sine^2-values by looping

```
for j=1:length(x)
    y(j)=sin(x(j)); f(j)=y(j).*y(j);
end
```

Or, as an alternative, use array manipulations

```
y=sin(x); f=y.*y;
```

Finally, a *function* subroutine (that allows multiplication of arrays) can be used.

For functions of two variables, say x and y, it often is useful to generate a 2-dimensional array of x-values and a 2-dimensional array of y-values ($n \times n$ arrays). For this, the MATLAB commands `meshgrid` and `ndgrid` can be used.

- Investigate the efficiency of computing an array of function values for a function of two variables (as $f(x, y) = \sin(x)\sin(2y)$ for $(x, y) \in [0, \pi] \times [0, \pi]$) for different ways of defining a function.
- Discuss the advantages and disadvantages of these different ways.
- Why would it be useful to define a function as a function subroutine (or as an so-called inline function, or as a string) rather than "ad hoc" in the command line in which you want to compute the function values?

In this course, we prefer to define functions (in particular, matrix-vector multiplications) as function subroutines (that take arrays as input variables).

Matrices can be viewed as two dimensional arrays of numbers. There are commands in MATLAB for graphically displaying arrays of one, two or three dimensions. MATLAB assigns a color value to each number (in the range of the array). The color value is determined not only by the number but also by the maximum and minimum value of the array of interest and the `colormap` that is be used. A digital 'black and white' picture can be viewed as the graphical representation of a 2-dimensional array: a black and white picture consists of pixels with a certain intensities of gray. Intensities of gray in a black and white picture correspond to (integer) numbers between 0 (black) and 255 (white).[6] In other words, a digital black and white picture can be viewed as a matrix (and vice-versa, possible with some scaling, a real matrix can be viewed as a digital picture).

Assignment 5.

Note that, in Assignment5.m, a picture is being entered and identified with a matrix.

An $n \times k$ matrix $\mathbf{A}$ can be transformed into an nk-vector by lining up the columns of the matrix. There are several ways to do this in MATLAB. Investigate the efficiency of these approaches.

Assignment 6.

The SVD (Singular Value Decomposition, `svd` in MATLAB) of an $n \times k$ matrix $\mathbf{A}$ consists of a unitary $n \times n$

[4] You can run an M-files as Assignment2.m in MATLAB by entering the command `Assignment2` (that is, without the extension .m).

[5] "Unofficially", commands as `x=zero(200,300);` can be used to assign an array, in this case an array of 200 by 300 zeros.

[6] or, depending on the convention that is being used, to real numbers between 0 and 1.

matrix $\mathbf{U}$ (that is, $\mathbf{U}^{\mathrm{H}}\mathbf{U} = \mathbf{I}_n$, where $\mathbf{I}_n$ is the $n \times n$ identity matrix), a unitary $k \times k$ matrix $\mathbf{V}$ and an $n \times k$ diagonal matrix Σ such that $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^{\mathrm{H}}$. In addition, the diagonal entries Σ_{ii} of Σ are non-negative reals and being ordered from large to small ($\Sigma_{ii} \geq \Sigma_{jj} \geq 0$ for all $i < j \leq \min(n, k)$).

- Enter a black and white picture as a matrix,
- determine its SVD,
- plot the diagonal of Σ (on log10-scale),
- for $j = 1 : k$, compute the matrix $\mathbf{A}_j$ that arises by multiplying the matrices $\mathbf{U}_j$, Σ_j and $\mathbf{V}_j^{\mathrm{H}}$. Here, $\mathbf{U}_j$ is the $n \times j$ matrix that consists of the first j columns of $\mathbf{U}$, $\mathbf{V}_j$ consists of the first j columns of $\mathbf{V}$ and Σ_j is the left $j \times j$ upper block of Σ,
- for $j = 1 : k$, represent $\mathbf{A}_j$ as a picture.

Inhoudsopgave