Introduction to MATLAB and its data model

Disclaimer: *These abbreviated notes DO NOT substitute the textbook for this class. They should be used IN CONJUNCTION with the textbook and the material presented in class. If there is a discrepancy between these notes and the textbook, ALWAYS consider the textbook to be correct. Report such a discrepancy to the instructor so that he resolves it. These notes are only distributed to the students taking this class with A. Gerbessiotis in Fall 2011 ; distribution outside this group of students is NOT allowed.*

(c) Copyright A. Gerbessiotis.

As we mentioned in the previous chapter, programs and data are stored in the main memory of a computer. Both program and data are stored as sequences of a collections of binary digits i.e. as a sequence of bytes. The fundamental aggregate is thus the **byte**, a collection of 8 **bits**.

B1. Machine code. If one wants to program a computer in the absence of helpful tools, he needs to manipulate bits and bytes at a very low level according to a specific computer-dependent format that is known as the **machine code** or **machine language**. This is tedious, complicated, error-prone, confusing and buggy.

For the sake of an example, think of the pretty tedious task of writing a simple integer, say `1` into memory. An integer nowadays is represented by 32 bits (i.e. a word) in two's-complement representation, and thus following the notation and discussion of the previous chapter its binary representation might look like the bit sequence `00000000 00000000 00000000 00000001` with spaces added every eights bits for readability and to highlight the bytes in this 32-bit word. Suppose we intend to store this integer in main memory and let memory locations 10,11,12,13 be used for this purpose. How do we store those 4 bytes in those 4 memory locations, one byte per location? Depending on the CPU or microprocessor, one or the other byte-order shown below might be used. A microprocessor using the former byte-order is called a big-endian processor, as opposed to the alternative; Intel processors are little-endian.

```
Big-Endian                              Little-endian
10       11       12       13          10       11       12       13
00000000 00000000 00000000 00000001    00000001 00000000 00000000 00000000

Byte order for storing 32-bit integer 1 = 00000000 00000000 00000000 00000001
```

B2. Assembler (Assembly language). If machine code is too complicated and too bit- or byte-oriented, one can avoid to use such low-level manipulation/programming by programming at a higher level of abstraction by using an **assembly language**. An assembly language is a user-friendlier (but not very user-friendly) version of machine code/language in which the user (the programmer) uses mnemonic codes to execute machine code instructions. In other words, the input to an assembly language is a higher-level language (eg. mnemonics) converted into the output language which is machine code. For example instead of using machine code `10101011`, one could use a mnemonic code `STORE A` that writes back into memory the contents of processor register `A` (a register is a small piece of memory located on the CPU that can be accessed very quickly and efficiently).

(c) Copyright A. Gerbessiotis. All rights reserved.

B3. Compiler. Even assembly language is too low-level for most programmers and end-users. It is also machine-dependent. A user can use instead a **compiler** that translates code written in a higher-level programming language that is mostly machine independent (for example one can write a piece of code in `C, C++, Java` that is identical on a Windows PC, a Sun Workstation, a Linux PC, an Apple OS-X laptop) into a lower-evel language that can then be executed by the target platform (Sun workstation or Windows PC). The lower-level language can be assembly (and then needs to be converted by an assembler into machine code) or even plain machine code that can just be executed directly.

Thus when using a compiler a user performs two independent tasks:

(a) the user first writes a program in a higher-level programming language that is saved into a file (or more than one files), and then compiles the file(s) using a compiler, with the compiler converting these file(s) into assembly or machine code,

(b) the object code (generic for machine code or assembly that is almost ready for execution) thus generated by the compiler can then be separately executed by the CPU.

Thus in executing a program with the use of a compiler there are at least two distinct steps: compilation (step (a)) and execution (step (b)). In several instances, compilation might require more steps because for example the compiler sometimes does not translate higher-level language code into assembly code directly but to some intermediate code, that is then translated into some machine-independent assembly and so on. And if someone made a mistake in the higher-level language program, both steps need to be repeated.

For example, a C program `testprog.c` can be compiled by a command `gcc testprog.c -o testprog` that compiles the source code `testprog.c` that contains the source program into an executable file named `testprog`, and then the executable file is run by issuing a `./testprog`.

B4. Interpreter. If the two steps of a compiler-based program execution seem complicated, an interpreter can be used that can execute directly (or indirectly) higher-level programming language statements with out the requirement for separate compilation and execution. **MATLAB** (initials for **MAT**rix **LAB**oratory), Python, Perl, Ruby, csh/tcsh/bash, Lisp, are interpreted programming languages that is, programs written in them can be executed directly by an interpreter of the language. The interpreter for **MATLAB** is slightly more complex and intelligent. It executes several precompiled pieces of code that are available to it, and compiled by a compiler that is part of **MATLAB** itself.

(c) Copyright A. Gerbessiotis. All rights reserved.

Algorithms and Problem Specification

In this course we will learn how to write small programs that implement efficiently computer-based solutions of computational problems.

B5. Algorithm. An **algorithm** is a well-defined (i.e. correct) sequence of computational steps that takes some value or set of values as input and generates some value or set of values as an output.

An algorithm is thus a method/tool for solving well-specified computational problems. A (computational) *problem* defines an input-output relationship. An **algorithm** describes a specific procedure for achieving this relationship, i.e. for each problem we may have more than one algorithms that solve it.

Before one embarks on solving a computational problem with a computer (and MATLAB) one needs to understand the problem itself i.e the input and output relationship of the problem. Computers are inherently precise and error-prone (buggy) and human beings are inherently imprecise. Combining (or coordinating) the two can have very disastrous consequences.

For example some time ago a piece of code contained the code `DO 10 I=1.10` written in programming language FORTRAN. The intent was to write `DO 10 I=1,10`. This piece of code was an orbit computation program that was used in NASA's Mercury flights in the early 60s. However when a user started running this program with known input and output data, the output the program was generating was close to but not identical to the known output data. Only because of this discrepancy was the missing comma (or incorrectly appearing period) discovered, and disaster was averted.

Consider for example the very simple problem of addition of numbers. One could describe the input output relationship for this problem as follows.

```
Input:  Two numbers a and b
Output:  a+b
```

Every NJIT student can understand what the problem is. Finding an algorithm that realizes a solution to that problem is also easy (it probably requires elementary school background). However the description of the problem is imprecise and writing code that solves it might be a complicated task even for a Computer Science student.

(c) Copyright A. Gerbessiotis. All rights reserved.

Question 1. Why is it so?

Answer. The user never specified explicitly how long the numbers can be.

- The user never specified whether a, b are natural (positive integer) numbers or
- arbitrarily sized integer numbers (say, `13213239219321` or `-23123210321020210210`),
- in decimal or binary or hexadecimal notation or some other notation,
- or fractionals (say, `1232132/1323223`),
- or real numbers (say, `232321.213213223`),
- or a mix of all of these,
- or even a complex number (i.e. $21323.23 + i2132$),
- or something else.

The same applies to the output.

Question 2. Why specifying the input form is important.

Answer. Computers are not as flexible as human beings. For example, computers are susceptible to attacks (by criminals) known as **buffer overflow attacks**. Because the input (and output) specification is imprecise, a reckless (or not very careful) programmer might make some generic assumptions on how large the input can be. A careless or intentionally devious user might supply a non-conforming input that would crash the program or force it to behave unpredictably and harmfully.

When we write in math $f(x) = x^2 - 2x + 1$, we define a function on an unknown variable x (sometimes we call x the indeterminate). This specific function is called a polynomial. The degree of the polynomial is two, and sometimes we call $f(x)$ a quadratic function. Human beings are quite intelligent and can easily fill-in details. Computers are dummy unless designed by extremely intelligent humans; the presence of many bugs (i.e. irregular instructions that cause problems in a computer code/program) indicates that we still have to accept that computers are not very intelligent or human beings do not yet know how to program them intelligently.

Human intelligence and inherent lack of precision

For a human $f(x) = x^2 - 2x + 1$ means $f(x) = x^2 - 2 \cdot x + 1$. (A $\cdot$ multiplication operator is implied between the 2 and the x.) When a human is asked `How much is` $f(0.5)$?, that human understands that the polynomial must be evaluated at $x = 0.5$, i.e. the human evaluates $f(0.5) = 0.5^2 - 2 \cdot 0.5 + 1$ and responds with the answer of `0.25`.

Computer precision and lack of intelligence

A computer on the other-hand needs guidance. It needs to be told what is expected to do.

- It needs to be told that it ought to evaluate a function.
- It needs to be told what the name of the function is going to be i.e. `f`.
- It needs to be told how the input is going to be made available i.e. `through a parameter to the function`.
- It needs to be **precisely** told what the range of values for the input parameter will be.
- It needs to be told what to do with the computed value i.e. `return it to the user who invoked the function` and what the range of values for the computed function value is going to be.
- It needs to be given precisely the input function (i.e. $\cdot$) or be told otherwise that $2x$ means $2 \cdot x$.
- It needs to be allocated in advance the scratch space where extra calculations can be performed.

(c) Copyright A. Gerbessiotis. All rights reserved.

B6. Parameter and Argument. Going back to the $f(x)$ example, x is called the **parameter** of the function. When we ask to evaluate f at point $x = 0.5$ by writing $f(0.5)$ we pass the value 0.5 as an **argument** to the function f. So the unknown variable is called the parameter, but its assigned value when an evaluation takes place is called the argument. In general in computer programs every parameter needs to be defined or declared explicitly (in compiled languages) or implicitly (in several interpreted languages).

B7. Variable. In computer programs every parameter or argument whose value changes becomes a **variable**. Every variable has a **data-type**. What is the data-type of a variable?

B8. Constant. If the value of a variable can never get modified, then it's called a constant.

B9. Data-type. In a programming language, every variable has a data-type, which is the set of values the variable takes and the operations defined on the data-type.

In mathematics, an integer or a natural number is implicitly defined to be of arbitrary precision. Computers and computer languages have **built-in** (called **primitive**) data-types for integers of finite precision. These primitive integer data-types can represent integers with 8-, 16-, 32- or (in some cases) 64-bits. An integer data-type of higher precision is only available not as a primitive data-type but as a **composite** data-type through aggregation and composition and built on top of primitive data-types. In languages that are **strongly-typed** a variable needs to be fully defined (i.e. specify the range of values) before it is used; in **weakly-typed** languages the first time a variable is used becomes also a declaration point and definition.

In computing we use the term **definition** of a variable to signify where space is allocated for it, and **declaration** of a variable to signify our intend to use it in a piece of a program and there is no prior declaration or definition of it (it might have been defined elsewhere). For a variable there can be only ONE definition but multiple declarations.

MATLAB. The nice thing with MATLAB is that it is a weakly-typed language and thus we do not need to bother with a variable's definition or declarations. A variable is defined (and also declared) the first time it is being used. MATLAB supports 8-,16-,32-, and 64-bit signed (two's complement) and unsigned integers. Also single (32-bit) and double (64-bit) precision floating-point numbers. These data types are called `int8, int16, int32, int64, uint8, uint16, uint32, uint64, single, double.` The default data type for a number in MATLAB is `double`. One can convert a number to a specific data type by using the name of the data type as a function i.e. `int32(8)` converts 8 into an 32-bit signed (two's complement) integer. Characters are represented in MATLAB as Unicode characters (2bytes).

(c) Copyright A. Gerbessiotis. All rights reserved.

B10. What is a data model? A data model is an abstraction that describes how data are represented and used.

MATLAB data model. The data model of MATLAB is the concept of a **matrix** i.e. elements arranges into a give number of rows and a given number of columns. The number of rows and columns are the dimensions of the matrix. A matrix with n rows and m columns is denoted by its geometry $n \times m$. Matrix elements are of the same data type. Thus the MATLAB data model supports characters and strings (collections of characters), integers of different sizes (eg. 32-bit or 64-bit etc), real numbers in the form of floating-point numbers of different sizes, complex numbers, and includes not only these (primitive) data-types but also methods that allow aggregation and composition of these data-types (e.g. in the form of matrices and vectors).

MATLAB internal representation. MATLAB implements the abstraction of a matrix by using a data structure called **array**. Array elements are random-access indexed. Thus for array a its elements $a(1)$ and $a(1000000)$ can be accessed in the same amount of time no matter what the indexes are (first and millionth). An array stores a collection of elements of the same data type.

B11. What is an Abstract Data Type (ADT)? An abstract data-type (ADT) is a mathematical model and a collection of operations defined on this model.
For example, we might define an ADT called **dictionary** D for which we define two operations: (a) Insert(D,x), for inserting x into the dictionary, and (b) LookUp(D,x), for looking-up object x into dictionary D. Note that at this abstraction level it is not necessary to define whether x is a string of characters (i.e. a word) or some other data-type. The important part of the definition is the collection of the operations and their semantics.

BM0. Accounts You can run MATLAB on a PC or an AFS system. If you already have an NJIT PC (laptop or otherwise) with MATLAB preinstalled you can skip most of this discussion. Otherwise you might have to install MATLAB on you PC. For this you need to have a UCID account (and its password). If you don't have such an account, go to `http://ist.njit.edu/accounts/index.php` and obtain a UCID account and set its password. More details are available in Subject 0. The following are useful URLs that you might need to use to complete the steps outlined below or to find software discussed in the next few pages.

- **NJIT Account Setup:** `http://ist.njit.edu/accounts/index.php`
- **NJIT Password Reset:** `http://afspassword.njit.edu/`
- **AFS resources/help:** `http://web.njit.edu/`
- **Software(NJIT):** `http://ist.njit.edu/software/download.php`

BM1. Find MATLAB. One way to run matlab is at home/dorm-room using your own Windows-based PC. The alternative is to run the free already installed version that is available on afs machines. For this you need an AFS login-name (`AFS-login` from now on) and a password (`AFS-password` from now on). You also need to pick an afs machine of your choice. Say we pick `afs12.njit.edu`. You can replace 12 by any number from around 1 to around 30. No matter what you pick the system will log give access to one of `afsconnect1.njit.edu` or `afsconnect2.njit.edu` which are the only two machines accessible from OUTSIDE NJIT. If you are inside NJIT, you can connect to one of 35 machines named `sunlab1.njit.edu` to `sunlab35.njit.edu`. If you login from home or from some other machine you need to use a program named `Secure Shell` to connect to an AFS machine. If you don't have it, NJIT has it for you; download and install it on your PC if it's not there already.

BM2. Connect to an AFS machine with Secure Shell. Click on a Secure-Shell icon, and then click on the pane(area) `QuickConnect` to get a window that looks like the one in Figure 1. Type in the field `Host Name` the afs machine you will be connected to (say `afs12.njit.edu`) and in the `UserName` field, type in your `AFS-login` (whatever it is for you), and be prepared to provide the corresponding password, after you click the `Connect` button. The first time you connect to `afs12.njit.edu`, the window in Figure 2 might pop-up, or otherwise you get the Password prompt in Figure 3.

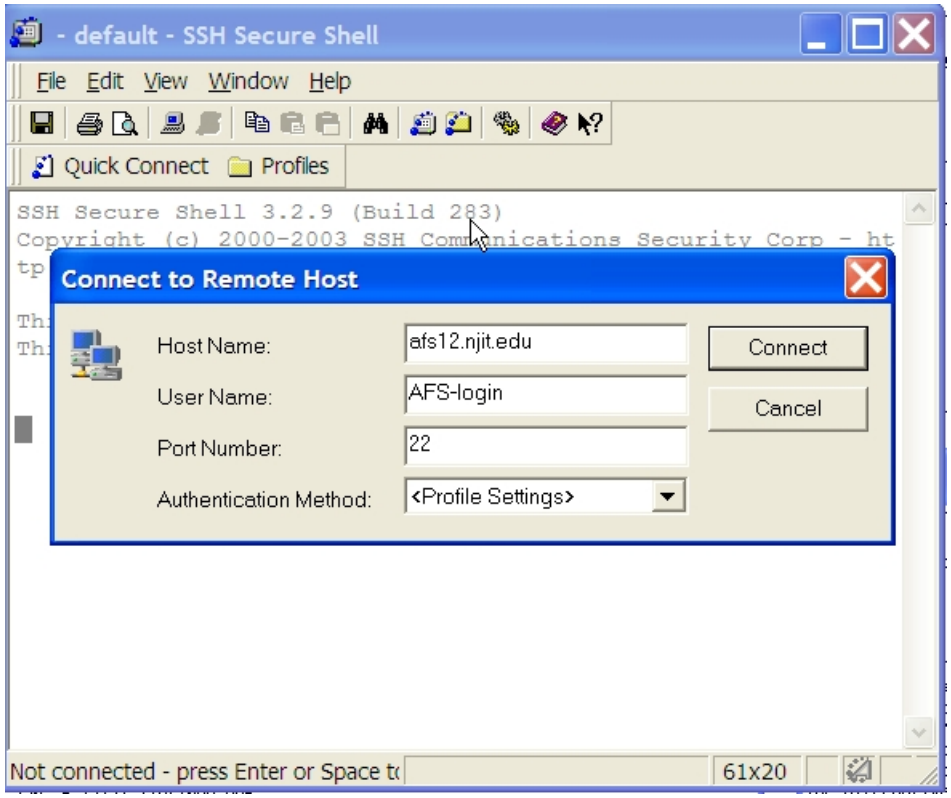


Figure 1: Secure Shell Window and Connect to Remote Host prompt.

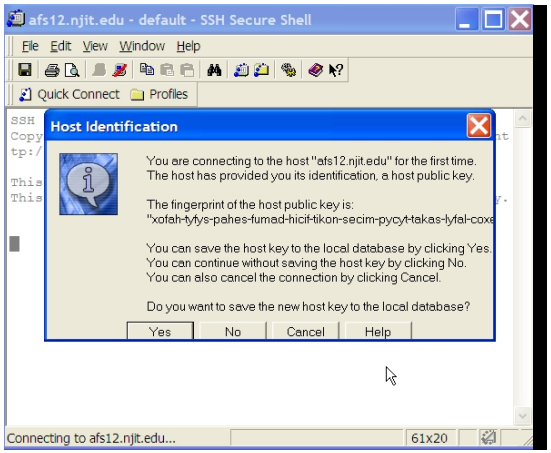


Figure 2: First time window

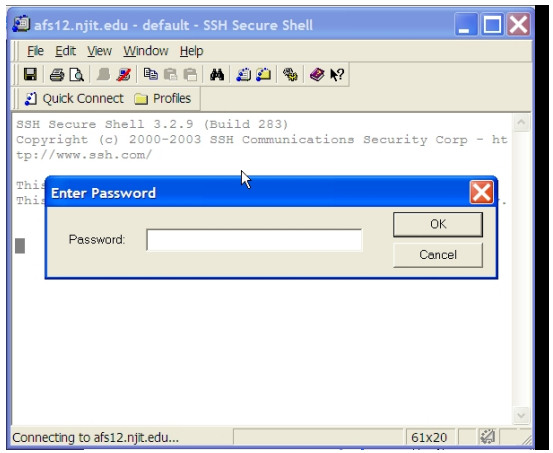


Figure 3: Password Prompt

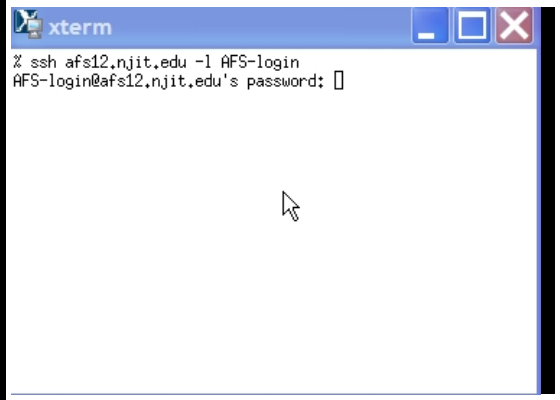


Figure 4: Secure Shell Invocation from a terminal.

BM2a. Alternative command-line connection. If you are on a UNIX (or LINUX or even Apple OSX) system, you can connect from the command prompt (or Terminal in Apple OSX) by typing in the command window something like `ssh afs12.njit.edu -l AFS-login`. Here the `AFS-login` is typed after `-l` which is the minus symbol (located on the keyboard next to the zero key) followed immediately by the ell letter (the letter after k but before m in the alphabet). Spaces are important where shown; case (lower case vs upper case) is important and all typing is sensitive to it. After pressing the ENTER symbol on the keyboard the Password prompt appears in the following line as shown in Figure 4. You type in the correct `AFS-password` and you enter `afs12.njit.edu` or whatever other machine you chose.

Note. If you are inside NJIT, use a `sunlab` machine; from outside NJIT you will end up using an `afsconnect` machine. The latter ones are not for CPU intensive applications, so multiple MATLAB sessions will experience a slowdown.

BM3. MATLAB invocation (text-based). If you are still using your Windows machine, find out how to call the MATLAB program: an icon on the Desktop might be available or you might have to have to click on `Start, All Program` and then select from the list that follows. Or if you are in `afs12.njit.edu` you can invoke MATLAB by just typing in `matlab` in lower-case and then pressing ENTER. (From now on every time we type in a command, we will assume that we press ENTER, and thus not repeat the "and then pressing ENTER".) MATLAB gives you the option of running one of four versions. Choose the latest by typing in 4 that corresponds to `MATLAB 7.0 R14.3`. If you want to quit press `ctrl-C`. This is done by pressing the Control key `Ctrl` and while pressing that key also pressing the `C` key as well. After you typed 4, MATLAB will take some time to load itself. After some time you get the `>>` MATLAB prompt. Note that MATLAB is an interpreter as we have already discussed. Therefore you are ready to execute MATLAB commands, functions or programs. If you want to quit right now (at the command prompt of MATLAB), you have the option of a graceful exit by typing in `quit`.

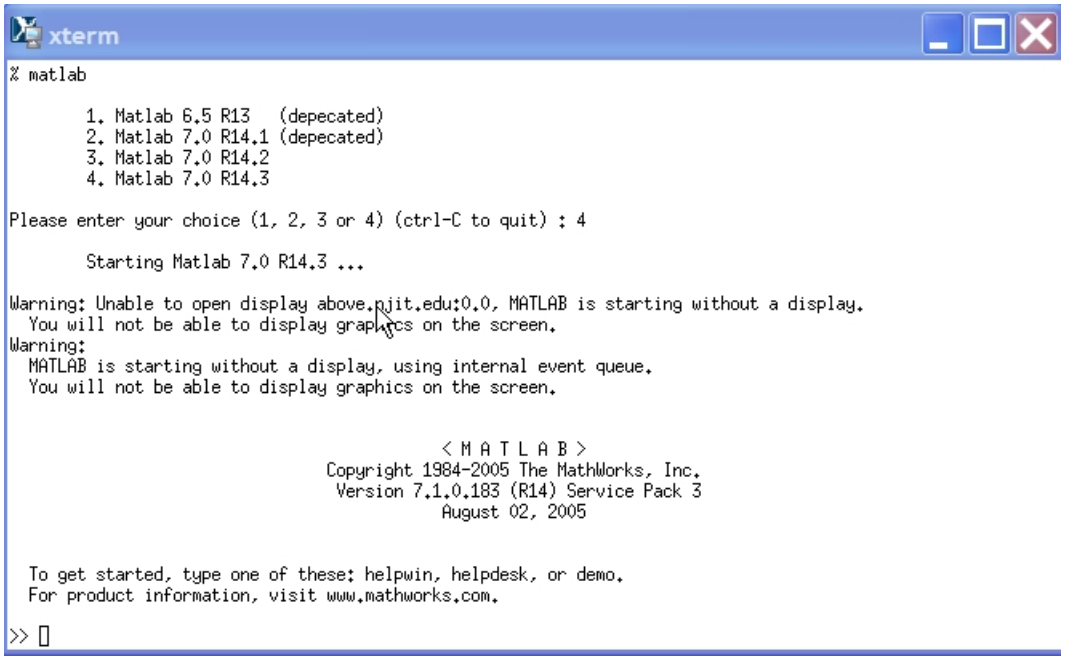


Figure 5: Text-based MATLAB invocation.

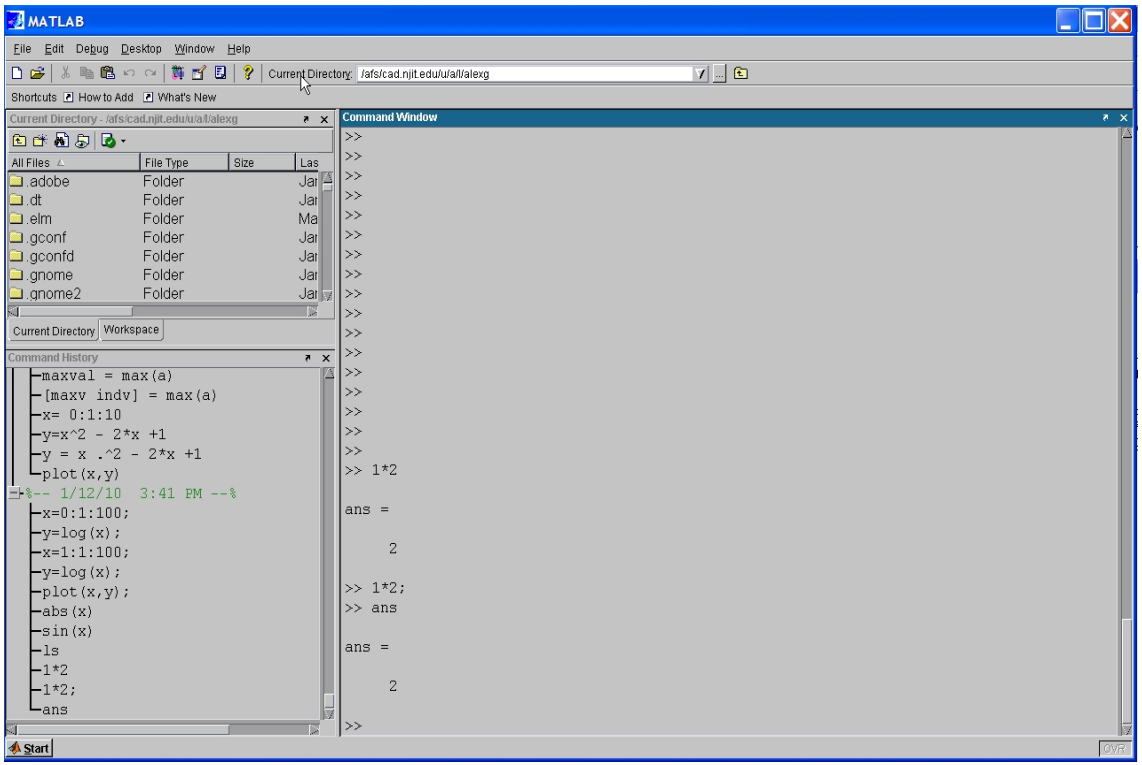


Figure 6: Graphical MATLAB invocation.

BM3. MATLAB invocation (graphical). In Figure 6 we see a graphical MATLAB window. The screen is split 1/3-2/3, with the right portion of the screen occupied by the **Command Window** similar to the one we obtained with the text-based invocation. The left portion of the screen is split into two subwindows. The top one shows the current directory (from which the invocation was issued) and the bottom-left windows shows a command history of the commands issued in the command window.

BM4. MATLAB Data Model. MATLAB's data model is very simple. Every object in MATLAB is a **matrix** or a special case of a matrix that is called a **vector**. A matrix or a vector are implemented internally by using a computer-based structure known as the **array**. Therefore before we begin talking about MATLAB's capabilities we need to provide definitions for a **matrix** and a **vector**. In the next few pages we review some definitions from matrix theory.

You will be formally introduced to the concepts of a matrix and a vector in a linear algebra class. Informally, a **matrix** is a rectangular array of numbers. For example, matrix A below is how we write a 3×4 matrix, where 3 indicates the row dimension of the matrix, i.e. the number of its rows, and 4 the column dimension i.e. the number of its columns. We thus say that A has three rows and four columns and we call it a 3×4 matrix A. The number of elements in A is thus the product of its two dimensions, in this case 12. We can individually reference elements of the matrix.

For example we can refer to the element of the second row and third column, i.e. 4. For matrix A we sometimes say $A = (a_{ij})$ to indicate how we reference its elements individually (i.e. using a in lower case). If such a definition is not explicitly given, you may assume it is present or just reference A by using the capital case letter instead. Thus we refer to an element of row i and column j of A by writing A_{ij}, or a_{ij}. Sometimes one might write $A_{i,j}, a_{i,j}$ for clarity i.e. insert a comma between the indices (indexes). If a subscript becomes messy or unreadable one can write $A(i,j)$ or $a(i,j)$. Thus $A(2,3), a(2,3), a_{23}, A_{23}$ all refer to an element of the second row and third column of A that is the element whose value is 4. Note that row and column indices start from 1; in some programming languages that implement matrices indexing starts from 0. MATLAB adheres to the start from 1 indexing.

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \\ 1 & 3 & 9 & 27 \end{pmatrix}$$

A generic (i.e. general) matrix say B may be denoted as an $n \times m$ matrix, i.e. it has n rows and m columns.

BM5.1 Square matrix. A special case of a matrix is a **square matrix** in which the number of rows and columns is the same. Thus if $n = m$ we say that then B is a square $n \times n$ (although one could also write that B is an $m \times m$) matrix. Several computational problems can be described through square matrices. Matrix B on the following page is a 4×4 matrix.

$$B = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix} \quad C = \begin{pmatrix} 1 & 5 & 9 & 13 \\ 2 & 6 & 10 & 14 \\ 3 & 7 & 11 & 15 \\ 4 & 8 & 12 & 16 \end{pmatrix} \quad D = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 0 & 1 & 7 & 8 \\ 0 & 0 & 1 & 12 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad E = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 5 & 6 & 0 & 0 \\ 9 & 10 & 0 & 0 \\ 13 & 14 & 15 & 16 \end{pmatrix}$$

BM5.2 Row-major and column-major fill-in of a matrix. Matrices B and C are both square of row/column dimension 4. Numbers $1 - 16$ are used to fill both matrices. You may have observed that the fill-in pattern is **row-major** for matrix B in the sense that the numbers fill-in the first row first, then the second and so one left-to-right top-to-bottom. On the other hand the fill-in pattern is **column-major** for matrix C since the elements first fill the first column, then the second and so on top-to-bottom left-to-right.

BM5.3 (Main) Diagonal of square matrix. The diagonal of a square matrix is the elements whose row and column indices are the same, i.e. all elements a_{ii}. We can describe B's diagonal by writing $1, 6, 11, 16$ top-left to bottom-right. The elements above the main diagonal form the upper triangle and the elements below form the lower triangle.

BM5.4 Upper and Lower triangular matrices. An upper triangular matrix is a matrix whose lower triangle is all zeroes i.e. this is roughly equivalent to saying its upper triangle and the diagonal are NON zeroes. A lower triangular matrix is a matrix whose upper triangle is all zeroes i.e. roughly speaking, the lower triangle and the diagonal are NON zeroes. Matrix D is an upper triangular and E is a lower triangular matrix. If the diagonal elements are zero then a matrix can be further called **strictly upper** or **strictly lower** triangular.

BM5.5 Diagonal Matrix. A diagonal matrix is a matrix whose non-diagonal elements are all zero.

BM5.6 Zero matrix. The zero matrix **O**, is a matrix whose elements are all 0. Thus a zero matrix can be viewed as an extreme example of a diagonal matrix with all elements including the diagonals being zero.

BM5.7 All-one matrix. The all-one matrix **1**, is a matrix whose elements are all 1.

BM5.8 Unit matrix. A unit matrix **I** is different from an all-one matrix. All its elements are zero except for the main diagonal elements that are all 1. That is a unit matrix is a diagonal matrix whose main diagonal elements are all 1.

BM5.9 Matrix transpose. The transpose of matrix A is denoted by A^T or A^t or sometimes simply A'. A^t is A with its rows becoming columns and its columns becoming rows of A^t. Thus if A is an $n \times m$ matrix, then A^t becomes a $m \times n$ matrix.

$$A^t = A^T = A' = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 4 & 9 \\ 1 & 8 & 27 \end{pmatrix} \qquad x = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \qquad y = \begin{pmatrix} 1 & 2 & 3 \end{pmatrix} \qquad x^t = \begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$$

BM5.10 Symmetric Matrix. A square matrix A is symmetric if and only if $A = A^t$. (The all-one, the zero, and the unit matrix are all symmetric.)

BM5.11 (Column) vector x. A (column) vector x is an $n \times 1$ matrix, i.e. a matrix with just one column. When we talk about vector x we implicitly mean column vector x.

BM5.12 Row vector y. A row vector y is an $1 \times n$ matrix, i.e. a matrix with just one row.

BM5.13 Writing a vector. On a piece of paper, it's easier to write a (column) vector as a row vector horizontally rather than using the vertical layout. This way we we write down x^t and read `vector` x, or we can write down y and read `vector` y^t.

BM5.14 Scalar. A scalar (i.e. value) is a 1×1 matrix, or (column) vector, or vector, or row vector. Given the multiple possibilities of calling such an entity (as a matrix or vector), we just call it a scalar.

BM5.15 Other matrices: Tridiagonal. A tridiagonal matrix is a square matrix $A = (a_{ij})$, where $a_{ij} = 0$ for all i, j such that $\mid i - j \mid > 1$.

BM5.16 Other matrices: Permutation. A permutation matrix is a square matrix whose entries are zero except for a single one in every row and column.

BM5.17. Matrix Addition. For two matrices A and B of geometry $n \times m$, matrix C of geometry also $n \times m$ is called the **sum** of A and B and we write $C = A + B$ if and only if $c_{ij} = a_{ij} + b_{ij}$.

BM5.18. Matrix Subtraction. For two matrices A and B of geometry $n \times m$, matrix C of geometry also $n \times m$ is called the **difference** of A and B and we write $C = A - B$ if and only if $c_{ij} = a_{ij} - b_{ij}$.

Addition and Subtraction requires $n \times m = n \cdot m$ operations. For multiplication, Linear Algebra and MATLAB support the traditional matrix multiplication, to be given in BM5.20. In addition MATLAB supports an array multiplication which is defined similar ro matrix addition and matrix subtraction.

BM5.19. Array multiplication. For two matrices A and B of geometry $n \times m$, matrix C of geometry also $n \times m$ is called the **array product** of A and B and we write $C = A.*B$ if and only if $c_{ij} = a_{ij}.*b_{ij}$. (Note that the operator $.*$ is a period immediately followed by an asterisk.) Array division or back-division can also be defined.

BM5.20. Matrix multiplication. For two matrices A and B of geometry $n \times m$, and $m \times p$ respectively, matrix C of geometry $n \times p$ is called the **matrix product** of A and B and we write $C = A \times B$ if and only if $c_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj}$. Similarly to a scalar product definition, the $\times$ may be dropped or even replaced by a $\cdot$ dot; thus $C = AB$ or $C = A \cdot B$ or $C = A \times B$ are all matrix multiplication products. However the $.*$ needs to be explicitly specified for an array product.

Matrix multiplication thus requires mnp scalar multiplications and $np(m-1)$ scalar additions. Thus for matrix multiplication to be defined the right dimension of A must be equal to the left dimension of B. Or we can say the innermost dimensions of the product as they are implied by $A \times B$ are the same.

BM5.21. Matrix inverse. For a **square** matrix A of geometry $n \times n$, under certain conditions, the inverse of matrix A is defined and denoted by A^{-1} rather than $1/A$. The inverse A^{-1} of A is such that $A \times A^{-1} = A^{-1} \times A = I$.

BM5.22. Matrix division. For **square** matrices A, B of geometry $n \times n$, A/B is to mean $A * B^{-1}$ and $A\backslash B$ to mean B/A.

(c) Copyright A. Gerbessiotis. All rights reserved.