MATLAB matrices (including vectors) and arrays

This is Chapter 2 of the textbook

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 2013 ; distribution outside this group of students is NOT allowed.*

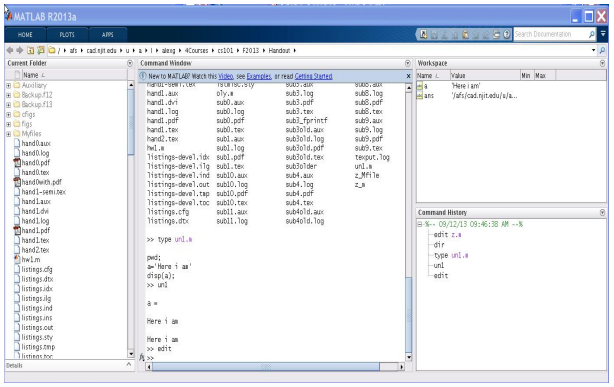


Figure 1: MATLAB Window view

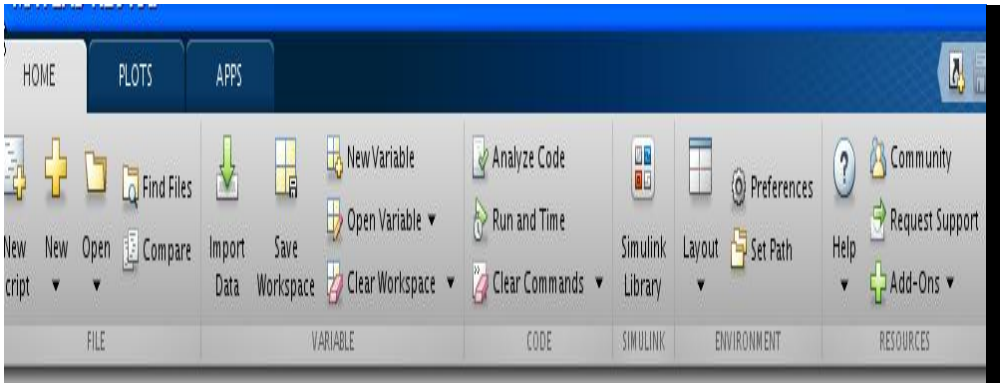


Figure 2: Toolstrip

3.1 Window view of MATLAB. A view of a MATLAB window for Matlab R2013a is shown in Figure 1. Behavior (default or otherwise) might be different in other recent versions of MATLAB and is definitely different from older versions of it. The Top pane of MATLAB contains three tabs, `HOME`, `PLOTS` and `APPS`. If you click once or twice on HOME the toolstrip (Figure 2 will disappear or re-emerge. Sometimes the `Toolstrip` appears attached to the `HOME` tab. If it is not shown click on `HOME` once to make it appear or once more to make it disapper. Somewhere in the middle to the right of the toolstrip you may find a button called `Layout`: this determines among others what windows are open (eg Command History, Workspace, etc) or permanently/by-default displayed (eg toolstrip). Note that such behavior or layout might be MATLAB version-dependent. Next to Layout you can find `Preferences`.

The MATLAB `Command Window` is centrally located and it is the widest one. On the left of the command window there usually is a vertical window called `Current Folder` that lists the contents of the current working directory/folder. On the right of the command window there are two vertical windows usually names `Workspace` and `Command history`. The `Workspace` is the user's workspace and can also be retrieved by typing `who` or `whos` in the command window. The `Command History` includes all the commands the user has entered in MATLAB's command window, not only in the current session but since the last time the user cleared it. If you want to clear the command history on the right of the title area there is a circular button with an arrow pointing downwards. Click on it.

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

3.2 MATLAB file types: An M-file. MATLAB recognizes and treats special filenames in a specific way. One such case is a filename ending with the suffix `.m` (`mm` is also a suffix). This is known in MATLAB as an M-file. For example `file.m` is such a file. For this case `file` is the prefix, `m` or `.m` is a suffix. Sometimes `m` is called the file extension and thus a period separates the prefix from the extension. In MATLAB an M-file can contain a sequence or collection of MATLAB functions/commands. One can call such an M-file from within MATLAB by typing in the name of the prefix only, that is by typing `file` without the `.m`. Then the sequence of MATLAB functions in the M-file will be executed in the command window and when this completes control will pass back to the MATLAB command window and the prompt will appear.

Reminder. When calling an M-file such as `myfile.m` , do not type `.m`, just `myfile`

3.3 MATLAB special M-files: startup.m and finish.m. There are two special M-files in MATLAB. The M-file `startup.m` is executed as soon as MATLAB loads up (and the command window appears), and `finish.m` that is invoked when the user is about to exit the command window by say, typing the function `quit`. Note that those two files will be executed as long as they reside in the directory from which MATLAB will be invoked or from which MATLAB will be terminated. One way to see whether the execution of `startup.m` or `finish.m` has occurred or is to occur is to type in the command window the operating system-related command `dir` to get the directory listing of the current directory within the command window and confirm the existence of one or the other files there. The typing of the comand `type startup.m` or `type finish.m` will show the contents (i.e. MATLAB commands) of those two files.

3.4 dir,ls, type, pwd, cd

The function `dir` described in 3.3 is a MSDOS/Windows command prompt relic equivalent to the UNIX command `ls`; both can be invoked from within the command window and both list the contents of the current working directory of MATLAB (the word folder is equivalent and a synonym to the word directory and will be used interchangeably). What this current directory is can be revealed by typing `pwd` (which is the initials of `print working directory`). You can view the contents of a file say `finish.m` by typing the MSDOS/Windows-style `type finish.m` command with a space separating the two. If you want to change directory, the `cd` command can be used appropriately (UNIX or Windows syntax as required). If you want to navigate to the top of your directory space type `cd` $\sim$.

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

3.5 Editing a file from within the Command Window or through Home/New. Note that in Matlab R2013a or in other variants the current working directory appears on a long line over the Command Window window (and its title/label). In fact on the left of the command window there usually is a vertical window calls `Current Folder` that lists the contents of the current working directory/folder. A user can edit (i.e. create, save) a MATLAB M-file by typing the word `edit` in the Command Window. If no argument follows the user will be given the opportunity to save the work in a file (M-file) later on through the opened window related to the Editor that has just been invoked. Of course one can type `edit myfile.m` and if `myfile.m` exists it will be opened, or a new file will be opened for writing. One can edit a file from outside the command window. MATLAB's toolstrip attached to the `HOME` tab (top of the window) has a button (second from left, with a huge plus sign) that is called `New`. One can create a New M-file by clicking on button `New` attached to tab `HOME` of the MATLAB window and then choose the first option or `Script` if it happens not to be the first option. An M-file is a script file that contains MATLAB commands. One can edit a file by clicking on `New` (it is the button with the huge plus sign second from left in the `Toolstrip`).

3.6 Diary keeping with `diary, diary on, diary off`. If you want to keep track of the activity in the command prompt (both the MATLAB commands you type, but also MATLAB's output), you can use the command `diary`. For example `diary('myd.txt')` will use file `myd.txt` to store information. When you type in MATLAB this command it also initiates a `diary on`. All the commands you type, and MATLAB's output are to be saved into the designated file. However you need to type a `diary off` for this to be committed/save into the file. You can then view the contents of it with `type myd.txt`. You can then issue `diary on` to go on saving additional information, as needed (also requiring a `diary off` eventually). The new material is added to (appended) the existing contents.

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

MATLAB M-file and diary example

```
%   let a file is name  myfile.m
%  Filename is        myfile.m
%  Prefix of it is    myfile
%  Suffix of it is    .m  or sometimes m (aka extension)
%  Extension of it is m

%  M-file is a file with suffix .m i.e. extension m
%  Special M-files are  startup.m and finish.p

>> ls ~/startup.m ~/finish.m
% If they exist they serve a special purpose
% All the MATLAB command in startup.m are executed before the command windows returns to the user the
% promptt >>
%
% When the user exits the command window, all the MATLAB commands in finish.m are executed
% before the windows closes (with a slight delay)
% The instructor startup.m file contains
   disp('HELLO. This is startup.m');
   diary(['CS101_alexg_' , datestr(now,'yyyymmdd_HHMMSS') ]);
   pause(2);
% The middle line is not obvious: it creates a diary file that keeps track of MATLAB commands
% typed in the command window
% If you don't have one type in a MATLAB command window
% diary('fname.txt');   create a diary with name  fname,txt (also does diary on)
% diary on              start it or restart it
%  diary off            stop and commit changes

>> edit ~/startup.m  % to create or edit one
>> edit ~/finish.m
% You can execute M-file myfile.m from within a command window by typing its prefix
>> myfile
```

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

3.7 MATLAB MAT (.mat) data files.

Whereas an M-file has a suffix `.m`, a MATLAB MAT file has a suffix of `.mat` in its filename. Thus `file.m` denotes an M-file but `file.mat` would denote a MATLAB MAT file. As we have already mentioned an M-file contains a sequence of MATLAB commands that operate on MATLAB variables and more. In MAT files MATLAB stores NOT COMMANDS but plain data such as variables (their names and their values) in binary format.

Think about storing in an MAT files the contents of the Workspace only but in an M-file the contents of the command history of a MATLAB environment. The data stored into an M-file are in binary format: it is very difficult for a human to read them. Whereas one can edit an M-file with an editor, this cannot happen with a MAT file.

Thus if one uses MATLAB as a calculator and has some intermediate results but wants to take a break and continue the calculations and did not maintain an M-file, the that person can save the values of all data generated into an MAT file, exit MATLAB, and then start MATLAB again and recover the variables and their values but not the commands that generated the variables and their values. The latter could be retrieved either through the use of a diary or an M-file. Because of this it is always a good idea to use an M-file for your calculations rather than working directly in a command window.

3.8 MATLAB load, save.

All variables defined in a MATLAB command window can be saved into a MATLAB file say `mydata.mat` by typing `save mydata` (the suffix will be created automatically). One can then issue a `clear` command in the command window to clear the variables (along with their values). The data however are still available in `mydata.mat`. Subsequently one can load them by using `load mydata` (the suffix is added as needed). A user can selectively load a specific variable (and its value) by typing instead `load mydata y`, or `load mydata x y`, where the list of variables to be loaded follows the name of the MAT file. This applies also to the `save` command. A `save mydata x y` could have also been used for the same effect as `save mydata`. The `append` option of save can be used to add one more variable into `mydata` without modifying `x,y`. This can be effected by typing `save -append mydata z`.

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

MATLAB MAT `.mat` file example

```
>> x=1;
>> y=2;
>> whos
  x         1x1                         8  double array
  y         1x1                         8  double array
>> save mydata
>> dir
 mydata.mat
>> clear
>> whos
>>
>> load mydata
>> whos
  x         1x1                         8  double array
  y         1x1                         8  double array
>> z=4;
>> save -append mydata z
>> clear
>> whos
>>load mydata
>>whos
  x         1x1                         8  double array
  y         1x1                         8  double array
  z         1x1                         8  double array
```

```
%  A .mat file  (mat from MATRIX) contains variables and their respective values copied from workspace
%  save fname.mat   % saves all existing variables from the workspace into file fname.mat
%                       % variables still exist in workspace and if modified, changes
%                       % WILL NOT affect values at fname.mat
%                       % if fname.mat exists all prior contents WILL BE ERASED first.
%      save fname.mat a b c  % stores only a,b,c but not other variables
%                       % all contents of fname.mat will be erased before the save
%      load fname.mat   % loads all variables from fname.mat
%      load fname.mat a % loads  only a from fname.mat
%
%       save fname.mat  d  % Erases all previous contents of fname.mat and stores d only
%
%       To maintain old variables + values!
%   To maintain old variables + values!
%
%      save fname.mat -append d  % Existing variables are maintained
```

(c) Copyright A. Gerbessiotis.

3.9 MATLAB anonymous functions. In MATLAB a very easy way to define a very simple function is by creating on-the-fly an "anonymous" function whose presence does not require an M-file (most MATLAB functions are required to reside in an M-file whose name should coincide with the name of the function to be defined). Thus an anonymous function $f(x) = x^2$ is defined by typing `f=@(x) (x*x)`. We note that x, the indeterminate, is the parameter of f in this context whereas $2, 5$ are the arguments for which the function f evaluates to 4 and 25 respectively.

```
>> f= @(x) (x^2)          % x described the function: it is the parameter aka indeterminate
  f= @ (x)(x^2)           % the value of f. In MATLAB terminology f is a function_handle
>> whos
  f         1x1               32  function_handle
>> f(5)
 ans = 25
>> x =10                   % this x is a variable; it has nothing to do with the x of the function_handle
>> whos
  ans       1x1                8  double
  f         1x1               32  function_handle
  x         1x1                8  double
>> f(x)                    % x has a value (100)
ans = 100                 % MATLAB first evaluates x to 10
                          % MATLAB then  uses the 10 to substitute for the parameter x in the
                          % functions handle  f= @ (x)(x^2)  i.e
                          %  f= @(10  (10^2)
                          % The value computed for (10^2) ie. 100 is returned as the value for f(x)
```

3.10 MATLAB is a weakly-typed language. MATLAB is an interpreted weakly-typed language. This means that the same variable x can be assigned an integer value, a real value, a string value, yet MATLAB will never complain about this. However it is the responsibility of the user and/or programmer to remember how a variable is being used and treated.
3.11. MATLAB abstraction: matrix. MATLAB's data model revolves around the matrix object. Every object is MATLAB is a matrix and stores information. Every variable in MATLAB is a matrix. A variable storing a single value (integer or real or something else) in MATLAB, is called a **scalar variable**, and is a 1×1 matrix. MATLAB matrixes consist of elements of arbitrary data types. We mentioned in Subject 2 some of the available data types in MATLAB such as `logical, char, double, single, int8, int16, int32, int64, uint8, uint16, uint32, uint64`. These are **built-in or primitive** data types supported by MATLAB. A user can define his/her own **composite** data types as well.

3.12 A MATLAB array implements a MATLAB matrix object. A matrix object is represented internally in MATLAB by a data structure known as an array which is a linear segment (sequence) of space that holds multiple number of elements of the same data type.

3.13 A MATLAB array stores elements in column-major form. An array stores a MATLAB matrix in **column-major** form (see also 3.38).

3.14. Matrix INPUT in MATLAB: row-major form. In MATLAB matrices are defined (i.e. typed) in row-major order.

3.15. Matrix OUTPUT in MATLAB: column-major form. However MATLAB prints matrices in column-major order. (In the remainder we use the term "order", "form" interchangeably.)

3.16. Default data type for a MATRIX object/MATLAB array. Real numbers in MATLAB are of `double` data type. We discussed briefly in Subjects 1-2 that real numbers are represented as floating-point numbers in MATLAB. The most common representation is in the form of a `double` which is 8-bytes long. Thus reals in the range from 10^{-308} to 10^{308} or so can be represented with up to 15 or 16 significant digits in a `double`. Unless a programmer explicitly asks for another data type, the default data type of a number object (scalar or otherwise) is a double. Thus `a=1` will use a double to store 1 as the value of variable `a`.

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

3.17 Variable names. MATLAB variables have names as introduced in Subject 2. Avoid variable names in MATLAB with more than 63 characters (or the current value of MATLAB's `namelengthmax`), they must start with a lower-case or capital case (English alphabet) letter (A to Z or a to z) but the remaining characters can be those alphabetic characters or digits 0 to 9 or the underscore sumbol _. Avoid spaces or punctuation marks. They cannot appear in a variable's name. (The underscore symbol is not the dash symbol − used for the subtraction operator.)

3.18 Assigning a value to a variable. The operation such as `x=10` assigns the value 10 to variable x. If x does not exist, it is created; its default data type would be `double`. One can write in MATLAB `y= 12.34` or `y=0.1234e2` for the exponential notation. The latter means $0.1234 \cdot 10^2$.

3.19 Special variable names. We have already mentioned several special variable in MATLAB. Sometimes we call keywords but if they can only be used exclusively by MATLAB we call them reserved words. Some them are `pi` and `i, j` which are all keywords. Even if special variables have special meaning and values, their value can be overidden. For example `pi=3` will create a local variable in the user's WORKSPACE named `pi` whose value is 3. The special MATLAB variable (the 3.1415...) still exists, but the local variable has PRECEDENCE of appearance and hides the MATLAB variable/constant. Thus a clear will only destroy the local copy of `pi` and will restore access to the more desirable value/variable. The same of course applies to i, j as well. To remind you $i * i = j * j = -1$ and i, j are complex numbers and the square root of -1 (positive one!). Try to do a `whos=10`. Afterward a `whos` returns that values and not information about all variables defined in the user's WORKSPACE. One need to do a `clear whos` to restored the default behavior. Two other special variables or functions in MATLAB are `nan` and `inf` that hold the values `NaN` and `Inf` respectively. Be very careful with the capitalization. `Nan` and `Inf` is a specific bit sequence used by `single, double` to represent the sate `Not a Number` and `Infinity`. The former is the result of an operation such a `0/0` , `Inf-Inf` or `Inf/Inf` or `Inf*0` or `0 * NaN`. (Think about NaN being possible to be plus or minus infinity or 0.) Comparing a NaN to a Nan is comparing apples to oranges (they are both fruits i.e. NaN but different) and give a false i.e. a NaN is not equal to a NaN. One can query a variable or a number whether it produces an `Inf, NaN`. Thus `isnan(a)` or `isinf(a)` return a true or false (1 or 0) depending on whether the variable a has a `NaN` (`Inf`) value or not respectively. Note also that while `NaN` is not equal to `NaN`, this might not be so for $5/0$ and $1/0$. Of course $5/0 + 3/0$ and $4/0$ are both equal to each other

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

3.20 Significant number of digits of real numbers in storage and on display. Although in Mathematics real numbers can have infinite precision (think of π or 1/3), in MATLAB a real number is stored as a 4B or 8B single or double respectively only 2^{32} or 2^{64} values can be represented respectively. As already mentioned `double` variables can maintain 15-17 significant digits, and `single` fewer. Yet you can print the value of a number on a screen in MATLAB's command window what you see is only few of those digits., The default display of a real number **format short** shows only 4 decimal digits. A more accommodating display is `format long` that can print 15 decimal digits for a `double` but only 7 for a `single`. Yet the internal representation of a double or single can maintain more significant digits. Note also that calculations involving `pi` and finite precision irrational or trancendental numbers can give unexpected results. For example we expect in Math that $sin(\pi)$ and $cos(\pi/2)$ both give 0 as an answer. Yet MATLAB's `sin(pi)` and `cos(pi/2)` give 1.2246e-16 and 6.1232e-17 respectively, i.e. different results that are small but non-zero! Comparing numbers such as 1.00000000000000001 and 1 can give unpredictable (or not) results.

3.21 ans, dieresis(...), semicolon(;), clc, errors and warnings, percent(%). The simplest use of MATLAB is as a rudimentary calculator. Every time a calculation such as `1+2` is typed and the ENTER/Return key is pressed, the result is computed and stored into a default variable named `ans` for answer. The default data type of `ans` is a `double` that uses 8B. In addition the assignment of the result to `ans` is shown onscreen. This display can be suppressed by typing a semicolon at the end of the expression as in `1+2;`. The semicolon can be to separate multiple expressions such as `1+2;20+30`. The evaluation of the last expression (righmost) determines the value that will be assigned to `ans`. If an expression is long and needs to be split over two (or more lines) one must type a dieresis i.e. three periods one after the other to continue the writing of the expression on the following line. Otherwise an error will be reported. MATLAB errors are in red while warnings (softer errors) are not in red. If the screen contains too many expressions, you can clear the screen of the command window by typing `clc`. If you want the contents of a line or fraction of to be ignored type the comment symbol of matlab which is the percent sign. Everything on that line starting the percent symbol will be ignored by MATLAB's evaluator (interpreter).

3.22 variables, clear, who, whos. The value of a MATLAB expression can be assigned to a variable.
Thus `x=10` or `x = 7+3` does the following in MATLAB. It evaluates the expression on the right of the equal sign (in the former case it is a constant 10 whose value is 10, in the latter case is the sum of 7 and 3 which is also 10), and the assigns this value to a variable named x. If x does not already exist, MATLAB creates x; its default data type for number is a `double`. Note that a `7+3` is thus equivalent to an implied `ans= 7+3`. MATLAB keeps track of all defined variables in its Workspace. The user can detect them by typing `who` where minimal information is provided, or `whos` where size information is also reported. If you want to get convinced that every MATLAB object is a matrix perusal of the output of `whos` will convince you. If one issues a `clear` this function will erase all variable already defined and its associated values. A `clc` on the other hand will only clear the display but not the MATLAB workspace. A `clear` can accept arguments separated with spaces: only those variables listed would be erased.
3.23 help. Anytime in MATLAB a question arises about the semantics of a command, one can just type `help` followed by a specific command to retrieve useful information about it. Thus `help transpose` provides some minimal information about the transpose function of MATLAB.

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

```
>> 1;                    % suppress output ; 1 gets stored into default variable ans (for answer)
>> whos                  % <<< this percent sign is the MATLAB comment symbol
  ans  1x1  8 double     % default data type in MATLAB is a double for numerical values
>> ans                   % retrieve the value of a variable by typing in its name
 ans =1
>> 1                     % does not suppress output
 ans =1
>> x=1;
>> whos
  ans 1x1 8 double
  x   1x1 8 double
>> x
x = 1
>> aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa = 10
 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa = 10
>> aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa = 10
Warning: 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'
exceeds the MATLAB maximum name length of 63 characters and will be truncated to
'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'.
>> y = 0.1234e2
  y=12.3400
>> x = 12.34
  x=12.3400
>> i*i
 ans=-1
>> j*j
 ans=-1
>> 5/0
 ans = Inf
>> 0/0
 ans = NaN
>> 5/0 == 4/0
 ans = 1
>> Inf == Inf
 ans =1   % true
>> NaN == NaN
 ans = 0  % false
>> isnan(0/0)
 ans=1
```

```
 >> sin(pi)
 ans = 1.2246e-16
 >> cos(pi/2)
 ans = 6.1232e-17
 >> cos(pi/2) == sin(pi)
 ans = 0
 >> 1.000000000000000001 == 1
ans =1
 >> 1.000000000000001 == 1
ans =0
 >> >> 1+
 1+
   |
Error: Expression or statement is incomplete or incorrect.
 >> 1+ ...
    2
 ans =3
 >> clc   % clears screen
 >> clear % cleans MATLAB WORKSPACE
 >> a=10; b=20;
 >> who
  a b
 >> clear
 >> who
 >>                % No variables defined in Workspace anymore.
 >> 1+2; 10+20;     % separate multiple expressions with ;
 >> ans             % the last one gives its value to ans!
  ans=30
```

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

```
 >> a = [ 1 2 3 4]        % a row- vector is a matrix with one row
  a =  1 2 3 4
>>  b = [ 1, 2, 3, 4]
  b =  1 2 3 4
>>  c = [ 1; 2; 3; 4]     % a vector (column vector) is a matrix with one column
 c = 1
     2
     3
     4
>> d = transpose(c)       % transpose(c) or c.'
 d = [ 1 2 3 4]
>> d = c.'
>> d = c'
 d = [ 1 2 3 4]
>> e = [ 1  2 3  4; 5 6  7 8; 9 10 11 12]
 e =  1  2  3  4
      5  6  7  8
      9 10 11 12
>> whos
  a         1x4                32  double
  b         1x4                32  double
  c         4x1                32  double
  d         1x4                32  double
  e         3x4                96  double
>> f = a
  f = [ 1 2 3  4]
>> [1 2 3 4]
ans = [1 2 3 4]
>> g= ans
  g = [ 1 2 3 4]
>> i
  ans = 0.000  + 1.0000i
>> i'
  ans = 0.0000 - 1.0000i
```

3.24 Writing a matrix in MATLAB: Space or comma for column separation of matrices but semicolon for row separation. We can initialize scalar variables and matrices the same way. Thus in the example to follow variable a becomes a 4-element `row-vector`. The left bracket [starts a matrix and thus also a row vector and a right bracket] concludes it. The elements are listed one after the other separated by spaces; one could have used commas as well to separate the elements of the single row but of multiple columns (column separator). b is the same matrix (vector as a). If a `column-vector` is to be initialized such as vector c, we need to separate its multiple row elements with a semicolon instead (row separator). The two vectors a and b are both of the same structure/geometry i.e. 1×4. A non vector matrix e is also created in the example above. In our examples a matrix/vector was directly assigned to a variable such as a, b, c, d, e, f. This does not need to be so. We can create an unnamed matrix that will be maintained as the value of `ans` and then we can assign it to a variable such as g. Note that e is input in row major order/form. The first row was input first, then its second row with the two separated by a semicolon.

3.25 Rows becoming columns (and columns becoming rows): Transpose operator. Column vector c is row vector a or b rotated (the columns of a become rows of c). Operator **transpose** in MATLAB is also the period followed by the quotation symbol `.'` and both can affect this, that is for matrix X, transpose(X) or `X.'` turns the rows of X into columns (and equivalently the columns of X into rows). If the numbers inside the matrix are real the same can be effected by the conjugate transpose which is just the quotation mark `'`. Thus $d = c'$ creates a row vector d from c which is equal to a and b. The only problem with the latter is if there are complex numbers among the elements of a matrix or a vector. See what happens when the matrix is scalar i in the example of the previous page!

3.26 `ones,ones(n,m),ones(n), zeros,zeros(n,m),zeros(n), eye,eye(n,m),eye(n), rand, rand(n,m),rand(n)` . Each one of ones, zeros, eye, rand accepts no, one, two or even more arguments. For a no argument call a 1×1 scalar will result, otherwise a $n \times m$ or $n \times n$ matrix will be generated. Function `ones` generate a matrix filled with one, zeros filled with zeros, and rand filled with random real numbers in the range (0,1). Function eye creates a matrix filled with zeros except along the main diagonal that is filled with ones (main diagonal is the one where the column index coincides with the row index). By default the matrices are `double`. Other than function rand the other functions can be instructed with a last argument to generate the elements in another data type; thus 'int8' presented as a string like this generates the zeros or ones as needed as 8-bit integers in two's complement representation. For rand the choice is between 'single' and 'double' only.

3.27 `diag(a)` **and** `diag(diag(a)) , triu(a),tril(a).` Function `diag` isolates the diagonal of a matrix i.e. the elements for which row and column indexes have the same value. However `diag(diag(a))` will create a matrix whose diagonal is the diagonal of a and everything else is 0. The matrix is square with number of rows/columns the elements of `diag(a)`. `triu(a),tril(a)` will extract the upper or lower triangular matrix of a and fill with zero the remaining elements.

```
>> zeros
 ans = 0
>> ones(1,3)
 ans = [1 1 1]
>> eye(2,4)
 ans =   1  0  0  0
         0  1  0  0
>> rand(2,2)
 ans = 0.5450    0.1183
       0.9739    0.6346
>> b=ones(1,3,'int32');
>> whos b
   b  2x3   24 int32
```

(c) Copyright A. Gerbessiotis.

```
>> a = [ 1 2 3 4 ; 5 6 7 8; 9 10 11 12]
 a =  1  2  3  4
      5  6  7  8
      9 10 11 12
>> a(2,3)
 ans = 7
>> a(2,3)=70
a =  1     2     3     4
     5     6    70     8
     9    10    11    12
>> k =4
 k =4
>> k(1,1) =4
 k =4
>> whos a k
 a 3x4   96 double
 k 1x1    8 double
>> a(1,5)=100
a =  1     2     3     4   100
     5     6    70     8     0
     9    10    11    12     0
>> k
  k=4
>> k(1,4)= 400
  k = 4  0  0 400
```

3.28 How to retrieve an element of a matrix. For a matrix a we can retrieve the element of its second row and third column by writing $a(2,3)$ using parentheses. T hus for the matrix a so defined writing $a(2,3)$ returns 7 since the element in the second row and third column is a 7.

3.29 How to change an element of a matrix. For a matrix a we can modify the element of its second row and third column by accessing it as if it was a variable i.e. $a(2,3)$ and then assigning a new value to it. Thus `a(2,3)=70` will change the 7 into 70, and if a semicolon does not follow 70, the whole matrix with the changed value showing up will be returned. For a scalar value such as k in the example above matrix indexing can also be used. A scalar is a 1×1 matrix!

3.30 length(a) For a vector, it returns its length (number of elements), for a matrix it returns the longest/largest of its dimensions.

3.31 size(a), size(a,1),size(a,2) It returns a row vector with two elements, the number of rows and columns of matrix a. One can retrieve them separately, by typing `size(a,1), size(a,2)` respectively.

What is the effect of `size(a)+1`? Food for thought...

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

3.32 MATLAB is a weakly-typed language and matrices are dynamically allocated and reallocated! For a matrix a we can modify an element using matrix indexing. What happens if we write by accident $a(1,5) = 100$? MATLAB is a forgiving language but behaves unexpectedly to an unsuspecting user. MATLAB will assume that user wishes to add a 5th column in a. Towards this it will work as follows: (a) It will create a new 3×5 matrix and will initialize its elements to zeros, (b) it will then copy all elements of a into that new matrix, (c) it will change the element of row 1 and column 5 to have value 100, and (d) then will rename the new matrix a and discard the previous one!

Thus the total amount of work would 3×5 to create the new matrix and fill its elements with zeros, 3×4 to copy the old elements into the new matrix, assume the renaming or deletion of the old matrix is for free, and then one more operation is the assignment $a(1,5) = 100$. Total cost is $3 \times 5 + 3 \times 4 + 1 = 28$.

The side-effect of this is that MATLAB will extend our 3×3 matrix c into a 4×4 matrix and then set the element of the first row and fourth column into a 101. **WATCH-OUT for side-effects!** The creation of the new matrix and its initialization will cost $4 \times 4 = 16$ operations, the copying of the old will cost an additional $3 \times 3 = 9$ and finally $c(1,4) = 101$ will cost one more for a total of 26. Here follows another example and a better solution. Using the previous accounting, the whole sequence requires at least $5 \times 5 = 25$ steps to create a new matrix and initialize it to zeroes, and $2 \times 2 = 4$ steps to copy `a` into the new matrix and one step to set an element to 10. Though for 2×2 and 5×5 matrix this is not very costly, if we had a $10,000 \times 10,000$ matrix b and then set `b(100000,100000)=10` this would become quite a costly operation. Avoid at all costs such a dynamic reallocation of a matrix. If you know in advance that the matrix will increase to a 5×5 just use the last three lines in the MATLAB code above instead!

```
>> a= [1 2; 3 4]
a =  1  2
     3  4
>> a(5,5)= 10
a = 1  2  0  0  0
    3  4  0  0  0
    0  0  0  0  0
    0  0  0  0  0
    0  0  0  0 10
% A more efficient solution is
>> a= zeros(5,5);
>> a([1 2],[1 2])= [1 2;3 4];
>> a(5,5)=10;
```

3.33 Modifying a variable that stores a scalar value. This works not just for general matrices but also for scalars. Note the effect of $k(1,4) = 400$ to the scalar variable k whose value was 4. NEVER FORGET: **the fundamental object of MATLAB is a MATRIX.** MATLAB will extend the 1×1 vector (or matrix) into a full-blown 1×4 row-vector!

3.34 end It is a variable that relates to the size of a matrix a and size(a,1) and size(a,2). If it is used in a row index it holds or represents the number of rows of a, if it is used in a column index, it holds the number of its columns. For a matrix a observe the output of $a(1 : end, 1 : end)$ for example.

3.35 sum, prod, sort For a vector x you can sum its elements by operation $sum(x)$, or multiply them with $prod(x)$. If x contains numbers then $sum(x)/length(x)$ gives the average value of the elements in x. Thus for $x =$ [30 10 20], we get 20. Function $sort(x)$ will return the elements of x sorted in increasing (non-decreasing order) i.e. [10 20 30].

3.36 min,max For a vector x, `min(x), max(x)` return the smallest element/largest element of x. If one wants to know the corresponding index of x then that value can also be retrieved. Thus $y = minx(x)$ for the example above returns `ans=10` but a `[a b] = min(x)` returns $a = 10, b = 2$. Note that `[a b]` indicates that we want `min` to return a row vector of two elements not just one; in that case the first is the minimum value and the latter the index of it in x.

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

3.37 Shortcut expressions. Certain arrays in MATLAB can be initialized using shortcut expressions; this is faster than writing the elements one by one. Members of a sequence of elements can be generated using a short **shortcut** expression. A shortcut (or colon operator) expression has the form

`first:increment:last`

where `first` is the value of the first element in the sequence that will be generated in the form of a row vector,
`increment` is the value that will be added to the previously generated value (or `first`) to generate the next value of the sequence, and
`last` is an upper bound of the last value that will be generated, in the sense that any value generated is NO MORE than `last` (i.e. it can be equal or less than `last`). Thus the arithmetic sequence
`first, first+increment, first+2*increment` will be generated. The last value that will be generated is
`first + k*increment` where k is the largest integer such that $\texttt{first} + \texttt{k} * \texttt{increment} \leq \texttt{last}$.
Instead of writing

`first:1:last`

one can just write

`first:last`

implying that the increment (that is missing) is one.

For the general case with base a, increment b and limit c

a:b:c

If $b > 0$ and $a > c$ returns an empty vector, and if $b < 0$ and $a < c$ it also returns an empty `1-by-0` vector.

Shortcut-expression initialization (continued)

Examples.
(a). `1:2:10` generates the odd (positive) numbers less than 10, i.e. $1, 3, 5, 7, 9$, since `1:2:10` generates `1, 1+2, 1+2+2, 1+2+2+2, 1+2+2+2+2`.
(b). `1:3` is `1 2 3` and `a= [1:3]'` generates a vector a with `1 2 3` as its elements of rows 1,2,3. Then `b=[a' a']` goes one step further and generates one longer (twice as long) row i.e. a 1×6 array since a is a column-vector, a' becomes a row-vector and thus b become a row-vector of twice as many columns as a'. Finally $c = [a \quad a]$ generates a 3×2 matrix with two identical columns each one being equal to a.
(c). `1:1:3` is another way to write `1:3`.
(d). `1:-1:0` generates `[ 1 0 ]`.
(e). `1:1:0` generates an empty 1-by-0 matrix.
(f). `1:end` might work for row/column vector and matrices as well to fill rows/columns.

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

```
>> a=input('Enter scalar ')
Enter scalar pi
a =
    3.1416
>> b=input('Enter row vector')
Enter row vector[1 2 3 ]
b =
     1     2     3
>> c=input('Enter string','s')
Enter string12345
c =
12345
>> c*2
ans =
    98   100   102   104   106

>> whos c
  Name      Size                    Bytes  Class
  c         1x5                        10  char array
Grand total is 5 elements using 10 bytes
```

3.38 input. We can read an array or matrix through the keyboard interactively. For this we can use the function `input`. Since `input` is a function its first parameter is a string that is printed on the terminal/window. A second parameter may or may not exist. If it is present and is equal `'s'` then the input read is converted into a string and then returned.

Examples. In the example above a requests that a scalar value be read. The user supplies the variable `pi` which is evaluate and its value assigned to a. Note that `'Enter scalar '` has a space before the quote, as opposed to `'Enter row vector'` that requests a vector. Caution should be drawn in the case of variable c. Because of `'s'` we know that the input will be converted into a string. Thus `12345` is converted into a character array that contains 5 characters: 1,2,3,4,5. Thus when one types $c*2$ the result is not a scalar $2*12345 = 24690$ but the doubling of the ASCII values of 1,2,3,4,5 which are respectively 49,50,51,52,53. A `whos c` on c can convince us that this is indeed the case.

```
>> a= [1 2 3 4 ; 5 6 7 8; 9 10 11 12]
a =  1     2     3     4
     5     6     7     8
     9    10    11    12
>> a(4)                     % find the 4th element in column-major order
ans = 2
>> a(9)                     % find the 9th element in column-major order
ans = 11
>> a(1,end)                 % matrix indexing: first row last column
ans=4
>> a(end,1)                 % matrix indexing: first column last row
ans= 9
>> a(end)                   % last element in column-major order ('bottom right')
ans = 12
>> a(1:end,1:end)           % same as a
ans= 1     2     3     4
     5     6     7     8
     9    10    11    12
>> a(1:end)                 % column major order as a row vector!
ans = 1     5     9     2     6    10     3     7    11     4     8    12
>> a(:)     ;               % what does the output look like ? Hint: transpose a(1:end)
```

3.39 Array vs Matrix. What is an array? An array in MATLAB is the internal storage or implementation of the mathematical construct or object known as matrix. The textbook interchangeably uses matrix and its representation as an array to refer to variable a. Operations on arrays and matrixes are slightly different for multiplication and division for example. For example multiplication on arrays is an element-wise operation different from the matrix-multiplication defined previously. Thus the elements of a two dimensional matrix for example can also be accessed through a single indexing rather than using the (row, column) indexing scheme. This way $a(j)$ means the j-th element of matrix a in column major order. Note in the example below that `end` holds three different values depending on the context: number or row, number of columns, or number of elements of a.

3.40 Two-dimensional matrix as an one-dimensional matrix. A two dimensional $n \times m$ matrix a in MATLAB is stored in column major form. To retrieve its element of row i and column j we just write $a(i, j)$ using parentheses. However we can retrieve that element by viewing a as a one-dimensional entity, a long column or vector or elements. Then what is the m for which $a(m)$ retrieves $a(i, j)$? In column-major form, element $a(i, j)$ is the $m = (j - 1) * n + i$-th element.

```
>> a
a = 1.1000    2.2000    3.3000    4.4000    5.5000
>> a([1 4])               % retrieve the first and fourth element
ans = 1.1000    4.4000

>> a(1:2:5)
ans = 1.1000    3.3000    5.5000

>> a(3:end)                % rertieve the elements starting with the third
ans = 3.3000    4.4000    5.5000

>> a(1:2:5) = [10 20 30]   % change the first third and fourth elements to the specified values
a = 10.0000    2.2000   20.0000    4.4000   30.0000

>> a([1 2]) = [100 200]    % do so for the first and second elements
a = 100.0000  200.0000   20.0000    4.4000   30.0000
```

3.41 Subarrays Submatrices/Subvectors. In MATLAB we can retrieve subarrays of an array or a submatrix of a matrix by specifying the boundaries i.e. rows and columns that we are retrieving. We can use shortcut expressions as well to realize this selection of rows/columns.

Example. Consider `a= [ 1.1 2.2 3.3 4.4 5.5]` . Then $a(3)$ retrieves its third element i.e. 3.3 (note a typo in the textbook on page 35 involving an array `arr1`). If we want to retrieve/read more than one element we can groupd the relevant indexes in a row vector. Then $a([1 \quad 4])$ retrieves only the first and fourth column i.e. returns 1.1 4.4 . Note that the column indexes 1 and 4 are grouped into a row vector. And $a(1:2:5)$ retrieves the odd-indexed columns i.e. 1.1 3.3 5.5 in the form of a row vector. Note again that $1:2:5$ is not surrounded by brackets. The shorcut colon expression returns a row vector instead. If $b = a'$, then b is a column vector. For two-dimensional matrices, such as the 3×3 matrix c, we have that $c(1,:)$ retrieves its first row, and $c(1:2:3,:)$ retrieves its odd-indexed rows. If we write $a(3:\texttt{end})$, we get 3.3 4.4 5.5.

3.42 Subarrays on the left side of an assignment. If we use a subarray on the left-hand-side of an assignment we update only some of the values (the matrix dimensions remain the same) of the matrix.

Example. Consider `a(1:2:5)` . It modifies the 1st,3rd,5th elements to the values specified by the array on the right-hand side. And `a([1 2])` changes the values of the first and second columns respectively.

```
>> a= [ 1 2 3 4 ; 5 6 7 8 ; 9 10 11 12];
>> a(1:2,1:2) = 100
a =
   100   100     3   4
   100   100     7   8
     9    10     11 12
```

3.43 Assigning scalar values to arrays/subarrays. If a scalar value (right hand side of equality sign) is applied to a matrix range (submatrix), then all the affected entries are modified to that value.

Example. For the example above, a $a(1:2, 1:2) = 100$ means that the top-left subarray of a defined by the top-two rows and columns is going to have its values modified to 100.

3.44 Deleting a row or a column. An operation such as **a(:,3)=[ ]** would delete the third column, whereas **a(3,:)=[ ]** would delete the third row. We are then left with a 2×3 matrix containing the values `1,2,4,5,6,8` in row major fillin/order.

3.45 Adding a row. We can reconstruct a by adding a new row, containing `9, 10, 12`, the previously deleted row. This can be done with a statement `a = [ a ; 9 10 12];` in MATLAB.

3.46 Adding a column. We can add a column by first transposing a to reduce this problem into one that adds a row. Thus `a = a'` is the first step. We need to add next a row that contains `[ 3 7 11 ]`. We do this by typing first $a = [a(1:2,:); 3, 7, 11; a(3:end,:)]$. First we form the first two rows of a, then we form the new one, and then we form the remaining rows starting with the third. We are almost close to retrieving the original a. What needs to be done is transpose a, i.e. `a = a'`.

3.47 Adding a column directly. We could add a column directly avoiding transpositions. First form the vector that needs to be added as in `b = [ 3 ; 7 ; 11 ]`. Then do in MATLAB `a= [a(:, 1:2) b a(:,3:end)]` , i.e. form the first two columns, then add the new column, and then the remaining columns.

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

3.48 Generating vectors for numerical solution of differential equations. Let us say you want to define a time interval between 0 and $t = 1$ and you want to split this interval into subintervals with step $h = 0.2$. One way to effect this is by typing `t= 0:0.2:1` to store the result in array $t = [0 \quad 0.2 \quad 0.4 \quad 0.6 \quad 0.8 \quad 1.0]$. However in some cases one would like to retrieve index information (i.e. the first, second etc interval.) Then an operation, such as $h = t./0.2$ would achieve this: note we use to array division to divide each element of t by the step 0.2. And also note that even if t is a vector, 0.2 is just a number. MATLAB is intelligent enough to realize what one wants to do i.e. divided each element of t with a real number. The only problem is that h is an array of real numbers. For indexing we need integers. One can then do $h = int32(h)$, or $h = floor(h)$. Note something interesting for h. Although MATLAB array indexes start from 1 the elements of h start from 0. Thus $h(1) = 0, h(2) = 1, h(3) = 2, h(4) = 3, h(5) = 4, h(6) = 5$. If you don't know how many elements there are in t a `length(t)` will give the answer (or `length(h)`).

3.49 Shifting a vector to the left one position. We can cut and paste t at will. Thus $t = t(2 : end)$ will eliminate its first element and shorten t by one. But let's go back to the original t. We can shift/rotate its elements by one by performing the operation $t([2 : end \quad , 1]) = t$. This will form a new t that would include $t(2), t(3), t(4), t(5), t(6), t(1)$ in that order. We used subarrays to achieve this

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

```
>> a= zeros(5,5);
>> a([1 2],[1 2])= [1 2;3 4];
>> a(5,5)=10;
>> a+1 ;
>> a-1 ;
>> a>1 ;
```

3.50 Array operation where second argument is a scalar. When we use an array (binary) operator we expect the two operands to be matrices of the same geometry (same rows and columns). Yet in the example above we perform operations $a+1$, $a-1$ and $a>1$ where the second argument is a scalar. The first two operations $+, -$, are interpreted by MATLAB to be array/matrix operations involving addition and subtraction. Even though the second operand is not a matrix of the same dimensions, MATLAB turns the scalar value into a matrix of the same dimensions as those of a, and whose entries are all equal to the scalar value and then the operations is performed as if it was between two matrices of the same geometry. For the third operation, every element of a is compared to 1 and the results is stored into a 5×5 matrix `ans`. In this latter case the data type of each element of `ans` will be a `logical` one-byte incarnation of true and false (i.e. of zeroes and ones). In the former two case a is a 5×5 matrix of `8-byte double` elements.

Operation	Matrix	Array	Condition
Addition	a + b	a + b	a,b same shape
Subtraction	a − b	a − b	a,b same shape
Multiplication		a .∗ b	a,b same shape
Multiplication	a ∗ b		rows(b) == columns(a)
Division		a ./ b	a,b same shape
Division	$a/b = a *$ inv(b)		square a,b, b nonsingular
Division	$a/b = a *$ inv(b)		a,b vectors of same shape, b nonzero
Back-Division		a .\ b	a,b same shape
Back-Division	$a\backslash b = b *$inv(a)		square a,b, a nonsingular
Back-Division	$a\backslash b = b *$inv(a)		a,b vectors of same shape, a nonzero

Figure 3: Matlab Array and Matrix operators

```
Operation         Array operator       Matrix Operator   Comment

Addition           +  i.e. C=A + B     + i.e. C=A+B      Element-wise addition   ;A,B,C are all nXm
Subtraction        -  i.e. C=A - B     - i.e. C=A-B      Element-wise subtraction;A,B,C are all nXm
Multiplication     .* i.e. C=A .* B    * i.e. D=E*F      .* : element-wise A,B,C are all nXm
                                                          *  : matrix multiply D : nXm E: nXp F: pXm
Division           ./ i.e. C=A ./ B    / i.e. D=E/F      ./, .\ element-wise A,B,C are all nXm
BackDivision       .\ i.e. C=A .\ B    \ i.e. G=E\F      D= E*inv(F) and G=F*inv(E)
                                                         For a square matrix Z nXn, inv(Z) is also nXn
                                                         such that Z * inv(Z) = inv(Z) * Z = I
```

ARRAY and MATRIX operations in MATLAB : Symbols used

3.51 Non-multiplicative array and matrix operations: $+, -$**.** For two MATLAB matrices of the same shape a and b, operations
$+$, $-$ are array and matrix operations and are defined as the element-wise addition and subtraction of the two matrices that is for $a(i,j)$ and $b(i,j)$ the resulting element $c(i,j)$ is one of $a(i,j)+b(i,j)$ and $a(i,j)-b(i,j)$, respectively. The operators $+, -$ signify **array addition/subtraction** and also **matrix addition/subtraction**.

3.52 Multiplicative, array operations: $.*$ $./$ $.\backslash$ $.\hat{}$. All the operations in this group are array operations: the multiplicative symbol is preceded by a period that indicates an array operation. For two MATLAB matrices of the same shape a and b, operations
$.*$, $./$, $.\backslash$, $.\hat{}$ are defined as element-wise operations between the two matrices that is for $a(i,j)$ and $b(i,j)$ the resulting element $c(i,j)$ is one of $a(i,j)*b(i,j)$, $a(i,j)/b(i,j)$, $b(i,j)\backslash a(i,j)$, $a(i,j)\hat{}b(i,j)$, respectively.

3.53 Matrix operation $*$. Since an array is the representation of a matrix in MATLAB, matrix addition and matrix subtraction are identical to array addition and subtraction as noted in 3.38. This was also mentioned in Subject 2. For matrix multiplication, the definition of Subject 2 applies i.e. $c = a*b$ is defined if and only if the number of columns of a is equal to the number of rows of b and then the result $c(i,j)$ is such that $c(i,j) = \sum_k a(i,k)*b(k,j)$.

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

```
 >> a = [ 1 2 3 4]         % a row- vector is a matrix with one row
  a =  1 2 3 4
>>  b = [ 1, 2, 3, 4]
  b =  1 2 3 4
>> a+b
ans = 2   4   6   8         % matrix addition,  array addition      are +  (common operator)
>> a-b
ans = 0   0   0   0         % matrix subtraction, array subtraction are -  (common operator)
>> a .* b
ans = 1   4   9  16         % array multiplication is .*
>> a ./ b
ans = 1   1   1   1         % array division       is ./
>> a .\ b
ans = 1   1   1   1         % array back-division  is .\
>> a * b                    % matrix multiplication not defined
Error using  *
Inner matrix dimensions must agree.
>> a / b                    % a / b for matrices is a * inv(b) where inv(b) is the inverse
                            % of a square matrix
                            % for non square matrix b MATLAB may try to solve a system
                            % of linear equations such as
                            % x b = a whose solution is x = a / b
 ans = 1.0000               % so this answer is not very surprising!
```

3.54 Matrix operations $/, \backslash$ defined for square matrices of the same shape. For two matrices a and b, the quotient a/b could be defined as the matrix product of a and the inverse $1/b$ of b. Thus the inverse of a matrix needs to be defined; for scalars, $1/b$ is the inverse of b if and only if $b(1/b) = (1/b)b = 1$. Similarly for a matrix b the inverse $b^{-1} = \texttt{inv}(b)$ is defined as that matrix so that $bb^{-1} = b^{-1}b = I$, where I is the identity matrix that in MATLAB notation is `eye`. Therefore a/b is defined as $a * \texttt{inv}(b)$ and $a\backslash b$ is defined as $b * \texttt{inv}(a)$ in MATLAB. Thus for / and \ to be defined $\texttt{inv}(b)$ and $\texttt{inv}(a)$ must exist and be defined. There is no equivalent to .^ for matrices. The inverse of a matrix is only defined for square matrices.

3.55 Matrix operations $/, \backslash$ for same shaped vectors. Yet even if division and backdivision should involve square matrices in the example involving row vectors a, b and answer was provided, rathen than an error. This is because for matlab to determin a/b MATLAB attempts to solve equation $xb = a$ whose solution is $x = a/b$ (forget for the moment the case b is singular or zero). For two row vectors b, a the equation $xb = a$ accepts a solution for a scalar x. This is the answer provided in the example above.

3.56 Other builtin functions as array operators. Other builtin operators such as `sin(x), cos(x), ... , sqrt(x)` etc work also on inputs that are vectors and matrices. In that case, the operator applies to each element of the vector/matrix.

3.57 Non-traditional behavior on some inputs: $\sqrt{}$ on matrices with negative values. Functions that by default are undefined on some inputs might have MATLAB equivalents that return surprising results. For example consider `sqrt`, the square root scalar function. It can also be extended to work on vectors and matrices. What if the input is a negative real number or a vector/matrix has elements that are negative numbers? The MATLAB realization of `sqrt` will work just fine. For negative numbers it returns a complex number. In fact the field of complex numbers includes all real numbers and is extended by one extra symbol: an i (sometimes denoted by j) such that $i^2 = j^2 = -1$, i.e. the extra symbol is the $\sqrt{-1}$. Thus for a negative number $-x$, where $x > 0$, the $\sqrt{-x}$ would be simply $\sqrt{-1} * \sqrt{x} = i * \sqrt{x}$, i.e. `sqrt` is well-defined for all real numbers if i (or j) is available, i.e. for complex numbers.

3.58 `date, now, datestr(now), calendar, weekday(now)`. The `date` function returns date information. And `calendar` returns a current month calndar. Variable `now` contains time/date information. Thus `datestr(now)` returns date and time information. A `weekday(now)` would return day of the week information in the form of a integer (1 being a Sunday).

3.59 `clock, cputime, tic, toc`. The `clock` function returns clock time in a single variable. Function `cputime` shows the CPU time since the command window was invoked. You can start a specific timing with a `tic` function call and terminate it with a `toc` function calls. The elapsed time between the `tic` and the `toc` will be shown. A `clear` command will not affect the `tic` time.

```
ones(n)    ones(n,m)       : n x n ,  n x m  all-1 matrix
zeros(n)  zeros(n,m)       : n x n ,  n x m  all-0 matrix
eye(n)     eye (n,m)       : n x n identity matrix, n x m identity matrix (see previous page)
rand(n)   rand(n,m)        : nxn, nxm array of (pseudo)random numbers between 0 and 1.
 ones(size(x)),
zeros(size(x))             : generates all-1 or all-0 of same dimension as matrix x

length(x)                  : return length of vector x or matrix x
size  (x)                  : returns number of rows and columns in x.

input(string,format)       : read from the keyboard the initialization expression
                             if format is present and equal to 's' a string is returned.

whos                       : list all variables and their definitions
whos x                     : list definition of variable x
clear                      : erase all variables
dir, ls, pwd               : directory listing and current working directory
clear                      : clear variables
clc                        : clear screen

 3.1. List of builtin matrix initialization functions

 pi    : the 3.14 stored internally to 15 digts
 i,j   : the square root of (-1) used in conjunction with complex numbers
 inf   : computer's representation of infinity  Inf
 nan   : representation of NaN, i.e. Not-a-Number
 clock : Array containing current date/time
 cputime: cputime in seconds since the start of a session
 date  : String containing current date
 eps   : Epsilon. Two numbers whose absolute value is less than eps are considered
         equal.
 ans   : answer variable for a result not assigned to some other variable.

 3.2 List of special symbols/variables
```

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