Introduction to Applied Scientific Computing using MATLAB

Mohsen Jenadeleh

In this lecture, slides from Mathworks, MIT, Waterloo and Rutgers Universities are used
Our Guiding Principles

“The purpose of computing is insight, not numbers”
Richard Hamming

“I hear and I forget,
I see and I remember,
I do and I understand.”
Confucious
Main Features of MATLAB

• Easy and efficient programming in a high-level language, with an interactive interface for rapid development.

• Vectorized computations for efficient programming, and automatic memory allocation.

• Built-in support for state-of-the-art numerical computing methods.

• Has variety of modern data structures and data types, including complex numbers.

• High-quality graphics and visualization.
• Symbolic math toolbox for algebraic and calculus operations, and solutions of differential equations.

• Portable program files across platforms.

• Large number of add-on toolboxes for applications and simulations.

• Huge database of user-contributed files & toolboxes, including a large number of available tutorials & demos.

• Allows extensions based on other languages, such as C/C++, supports Java and object-oriented programming.
• Parallel Computing (2)
• Math, Statistics, and Optimization (8)
• Control System Design and Analysis (6)
• Signal Processing and Communications (7)
• Image Processing and Computer Vision (4)
• Test and Measurement, Data Acquisition (5)
• Computational Finance, Datafeeds (7)
• Computational Biology (2)
• Code Generation and Application Deployment (11)
• Database Connectivity (2)

(54 toolboxes + 35 simulink products)
Web Resources

- Getting Started with MATLAB (HTML)
- Getting Started with MATLAB (PDF)
- MATLAB Examples
- MATLAB Online Tutorials and Videos
- MATLAB Interactive Tutorials
- MATLAB Toolbox Reference Manuals
- MATLAB Interactive CD
- Newsletters

- MATLAB User Community
- Other MATLAB Online Resources
- comp.soft-sys.matlab newsgroup

- Octave – a free look-alike version of MATLAB
- FreeMat – another free look-alike version

- NIST – Digital Library of Mathematical Functions
- NIST – Physical Constants
These should be enough to get you started. We will explore them further, as well as other topics, in the rest of the course.
You can select what is on your desktop by Clicking on Layout. Go down to Command History and select docked.
2. MATLAB Desktop

MATLAB Editor for writing Script Files or Functions

workspace window
Several ways of getting help:

1) help menu item on MATLAB desktop opens up searchable help browser window

2) from the following commands:

```matlab
>> helpdesk % open help browser
>> help topic % e.g., help log10
>> doc topic % e.g., doc plot
>> help % get list of all help topics
>> help dir % get help on entire directory
>> help syntax % get help on MATLAB syntax
>> help / % operators & special characters
>> docsearch text % search HTML browser for 'text'
>> lookfor topic % e.g., lookfor acos
```
4. Variables, Constants, Keywords

Variables require no special declarations of type or storage. Examples:

```matlab
>> x = 3; % simple scalar
>> y = [4, 5, 6]; % row vector of length 3
>> z = [4; 5; 6]; % column vector of length 3
>> A = [1, 2, 3; 4, 5, 6]; % 2x3 matrix
>> s = 'abcd efg'; % string
>> C = {'abc', 'defg', '123-456'}; % 1x3 cell array
```

The functions `class` and `size` tell you the type and dimensions of the defined object, e.g.,

```matlab
>> class(C)
>> size(C)
```
Several things happen with this simple MATLAB command:

A variable, `x`, of type double is created
A memory location for the variable `x` is assigned
The value 3 is stored in that memory location called `x`.

```
>> x = 3
x =
    3

>> y = [4, 5, 6]
y =
    4     5     6

>> z = [4; 5; 6]      % note, z = y'
z =
    4     5     6

>> A = [1 2 3; 4 5 6]
A =
    1     2     3
    4     5     6
```
What are your variables? How to clear them?
Use workspace window, or the commands:

   who, whos, clear, clc, close

>> who
Your variables are:
A  y  z

>> whos
    Name      Size     Bytes  Class     Attributes
   A         2x3         48  double
   y         1x3         24  double
   z         3x1         24  double

>> clear all % clear all variables from memory
>> clc % clear command window
>> close all % close all open figures
Operating system commands:

```plaintext
>> path             % display search path
>> pathtool % modify search path
>> addpath dir % add directory to path

>> cd dir % change directory
>> pwd % print working directory

>> dir % list all files in current dir
>> what % list MATLAB files only
>> which file % display location of file
>> edit file % invoke MATLAB editor
>> help % command provides information about a function

>> help sin %This only works if you know the name of the function you want help with.

>> quit % quit MATLAB
>> exit % quit MATLAB
```
### Some MATLAB® Math Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>MATLAB®</th>
<th>Function</th>
<th>MATLAB®</th>
</tr>
</thead>
<tbody>
<tr>
<td>cosine</td>
<td>cos or cosd</td>
<td>square root</td>
<td>sqrt</td>
</tr>
<tr>
<td>sine</td>
<td>sin or sind</td>
<td>exponential</td>
<td>exp</td>
</tr>
<tr>
<td>tangent</td>
<td>tan or tand</td>
<td>logarithm (base 10)</td>
<td>log10</td>
</tr>
<tr>
<td>cotangent</td>
<td>cot or cotd</td>
<td>natural log (base e)</td>
<td>log</td>
</tr>
<tr>
<td>arc cosine</td>
<td>acos or acosd</td>
<td>round to nearest integer</td>
<td>round</td>
</tr>
<tr>
<td>arc sine</td>
<td>asin or asind</td>
<td>round down to integer</td>
<td>floor</td>
</tr>
<tr>
<td>arc tangent</td>
<td>atan or atand</td>
<td>round up to integer</td>
<td>ceil</td>
</tr>
<tr>
<td>arc cotangent</td>
<td>acot or acotd</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note:**

- \( \cos(\alpha) \) assumes \( \alpha \) in radians; whereas, \( \cosd(\alpha) \) assumes \( \alpha \) in degrees.
- \( \cos(x) \) returns the angle in radians; whereas, \( \cosd(x) \) returns the angle in degrees.

\[ \pi \text{ radians} = 180 \text{ degrees} \]
1. Variable names must begin with a letter
2. Names can include any combinations of letters, numbers, and underscores
3. Maximum length for a variable name is 63 characters
4. MATLAB® is case sensitive. The variable name A is different than the variable name a.
5. Avoid the following names: i, j, pi, and all built-in MATLAB® function names such as length, char, size, plot, break, cos, log, ...
6. It is good programming practice to name your variables to reflect their function in a program rather than using generic x, y, z variables.
Special built-in math constants that should not (though they can) be re-defined as variables:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eps</td>
<td>% machine epsilon - floating-point accuracy</td>
</tr>
<tr>
<td>i,j</td>
<td>% imaginary unit, i.e., sqrt(-1)</td>
</tr>
<tr>
<td>Inf,inf</td>
<td>% infinity</td>
</tr>
<tr>
<td>intmax</td>
<td>% largest value of specified integer type</td>
</tr>
<tr>
<td>intmin</td>
<td>% smallest value of specified integer type</td>
</tr>
<tr>
<td>NaN,nan</td>
<td>% not-a-number, e.g., 0/0, inf/inf</td>
</tr>
<tr>
<td>pi</td>
<td>% pi</td>
</tr>
<tr>
<td>realmax</td>
<td>% largest positive floating-point number</td>
</tr>
<tr>
<td>realmin</td>
<td>% smallest positive floating-point number</td>
</tr>
</tbody>
</table>

Note: i, j are commonly used for array and matrix indices. If you’re dealing with complex-valued data, avoid redefining both i, j.
Values of special constants:

\[
\text{>> eps} \quad \text{\texttt{ans =}} \quad 2.2204e-016 \\
\text{\texttt{>> intmax} \quad \text{\texttt{ans =}} \quad 2147483647} \\
\text{\texttt{>> intmin} \quad \text{\texttt{ans =}} \quad -2147483648} \\
\text{\texttt{>> realmax} \quad \text{\texttt{ans =}} \quad 1.7977e+308} \\
\text{\texttt{>> realmin} \quad \text{\texttt{ans =}} \quad 2.2251e-308}
\]
Special keywords that cannot be used as variable names:

>> iskeyword

ans =
    'break'
    'case'
    'catch'
    'classdef'
    'continue'
    'else'
    'elseif'
    'end'
    'for'
    'function'
    'global'
    'if'
    'otherwise'
    'parfor'
    'persistent'
    'return'
    'switch'
    'try'
    'while'
    'true'
    'false'
Computers store all data (numbers, letters, instructions, …) as strings of 1s and 0s (bits).

A **bit** is short for **binary digit**. It has only two possible values: On (1) or Off (0).

A **byte** is simply a string of 8 bits.

A **kilobyte** (kB) is 1000 bytes (commercial), kilobyte is traditionally used to denote $1024$ ($2^{10}$) bytes.

A **megabyte** (MB) is 1,000,000 bytes

A **gigabyte** (GB) is 1,000,000,000 bytes

For a sense of size, click on link below:

http://highscalability.com/blog/2012/9/11/how-big-is-a-petabyte-exabyte-zettabyte-or-a-yottabyte.html
MATLAB by default uses double-precision (64-bit) floating-point numbers following the IEEE floating-point standard. You may find more information on this standard in:

**Representation of Floating-Point Numbers**

C. Moler, "Floating Points," MATLAB News and Notes, Fall, 1996 (PDF file)

\[
x = (-1)^s \cdot (1+f) \cdot 2^{(e-1023)}
\]

- 1 bit: sign
- 52 bits: mantissa
- 11 bits: exponent
- \(1 \leq e \leq 2046\), \(e=0\), \(e=2047\)

- \(0 \leq f < 1\)
- \(f_{\text{min}} = \epsilon = 2^{-52}\)

**machine epsilon**
MATLAB can also use single-precision (32-bit) floating point numbers if so desired.

There are also several integer data types that are useful in certain applications, such as image processing or programming DSP chips. The integer data types have 8, 16, 32, or 64 bits and are signed or unsigned:

$$\text{int8, int16, int32, int64}$$
$$\text{uint8, uint16, uint32, uint64}$$

These data types work for integers as long as the integers don’t exceed the range for the data type chosen.

They take up less memory space than doubles.
They don’t work for non-integers. If you create a variable that is an int8 and try to assign it a value of 14.8, that variable will be assigned a value of 15 instead (closest integer within the range).

One common application for integer data types is image data (jpeg, png, …)

For more information do:

```plaintext
>> help datatypes
>> help class % determine datatype
>> help int32 % example
```
## Numeric Data Types

MATLAB has several different options for storing numbers as bits. Unless you specify otherwise, all numbers in MATLAB are stored as doubles.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>double</td>
<td>64 bit floating point</td>
<td>$\pm2.23\times10^{-308}$ to $\pm1.80\times10^{308}$</td>
</tr>
<tr>
<td>single</td>
<td>32 bit floating point</td>
<td>$\pm1.18\times10^{-38}$ to $\pm3.4\times10^{38}$</td>
</tr>
<tr>
<td>uint8</td>
<td>8 bit unsigned integer</td>
<td>Integers from 0 to 255</td>
</tr>
<tr>
<td>int8</td>
<td>8 bit signed integer</td>
<td>Integers from $-128$ to 127</td>
</tr>
<tr>
<td>uint16</td>
<td>16 bit unsigned integer</td>
<td>Integers from 0 to 65535</td>
</tr>
<tr>
<td>int16</td>
<td>16 bit signed integer</td>
<td>Integers from $-32768$ to 32767</td>
</tr>
<tr>
<td>uint32</td>
<td>32 bit unsigned integer</td>
<td>Integers from 0 to 4294967295</td>
</tr>
<tr>
<td>int32</td>
<td>32 bit signed integer</td>
<td>Integers from $-2147483648$ to 2147483647</td>
</tr>
</tbody>
</table>
Why should I care how data is stored in a computer?

Example: Perform each of the following calculations in your head.

\[
a = \frac{4}{3}
b = a - 1
c = 3b
e = 1 - c
\]

What does MATLAB get?
Why should I care how data is stored in a computer?

What does MATLAB get?

\[ a = \frac{4}{3} = 1.3333 \]
\[ b = a - 1 = 0.3333 \]
\[ c = 3 \times b = 1.0000 \]
\[ e = 1 - c = 2.2204 \times 10^{-16} \]

It is not possible to perfectly represent all real numbers using a finite string of 1s and 0s.
Not all numbers can be represented exactly even using 64 bit doubles.

If we do many, many calculations with numbers which are just a tiny bit off, that error can grow very, very large depending on the type of computations being performed.

64 bit doubles have a huge, but still limited range.

What happens if we exceed it? Try the following:

\[
\begin{align*}
>> & \quad a = 373^{1500} \\
>> & \quad b = \text{factorial}(172)
\end{align*}
\]
By default, MATLAB treats all numbers and expressions as complex (even if they are real).

No special declarations are needed to handle complex-number operations. Examples:

```matlab
>> z = 3+4i; % or, 3+4j, 3+4*i, 3+4*j
>> x = real(z); % real part of z
>> y = imag(z); % imaginary part of z
>> R = abs(z); % absolute value of z
>> theta = angle(z); % phase angle of z in radians
>> w = conj(z); % complex conjugate, w=3-4i
>> isreal(z); % test if z is real or complex
```

\[ z = x + jy = Re^{j\theta}, \quad R = |z| = \sqrt{x^2 + y^2}, \quad \theta = \arctan \frac{y}{x} \]

**cartesian & polar forms**

**math notation:** \[ \theta = \text{Arg}(z) \]
>> z = 3+4j
z =
3.0000 + 4.0000i

>> x = real(z)
x =
3

>> y = imag(z)
y =
4

>> R = abs(z)
R =
5

>> theta = angle(z) % in radians
theta =
0.9273

>> abs(z - R*exp(j*theta)) + abs(z-x-j*y) % test
ans =
6.2804e-016

equivalent definitions:

z = 3+4*j
z = 3+4i
z = 3+4*i
z = complex(3,4)
### Display Formats

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&gt;&gt; format</code></td>
<td>% default - 4 decimal places</td>
</tr>
<tr>
<td><code>&gt;&gt; format short</code></td>
<td>% same as the default</td>
</tr>
<tr>
<td><code>&gt;&gt; format long</code></td>
<td>% 15 decimal places</td>
</tr>
<tr>
<td><code>&gt;&gt; format short e</code></td>
<td>% 4 decimal - exponential format</td>
</tr>
<tr>
<td><code>&gt;&gt; format short g</code></td>
<td>% 4 decimals - exponential or fixed</td>
</tr>
<tr>
<td><code>&gt;&gt; format long e</code></td>
<td>% 15 decimals - exponential</td>
</tr>
<tr>
<td><code>&gt;&gt; format long g</code></td>
<td>% exponential or fixed</td>
</tr>
<tr>
<td><code>&gt;&gt; format shorteng</code></td>
<td>% 4 decimals, engineering</td>
</tr>
<tr>
<td><code>&gt;&gt; format longeng</code></td>
<td>% 15 decimals, engineering</td>
</tr>
<tr>
<td><code>&gt;&gt; format hex</code></td>
<td>% hexadecimal</td>
</tr>
<tr>
<td><code>&gt;&gt; format rat</code></td>
<td>% rational approximation</td>
</tr>
<tr>
<td><code>&gt;&gt; format compact</code></td>
<td>% conserve vertical spacing</td>
</tr>
<tr>
<td><code>&gt;&gt; format loose</code></td>
<td>% default vertical spacing</td>
</tr>
<tr>
<td><code>&gt;&gt; vpa(x,digits)</code></td>
<td>% variable-precision-arithmetic</td>
</tr>
</tbody>
</table>

These affect only the display format – internally all computations are done with full (double) precision
Example - displayed value of \( 10\pi \) in different formats:

\[
\begin{align*}
31.4159 & \quad \% \text{ format, or format short} \\
31.415926535897931 & \quad \% \text{ format long} \\
3.1416e+001 & \quad \% \text{ format short e} \\
31.416 & \quad \% \text{ format short g} \\
3.141592653589793e+001 & \quad \% \text{ format long e} \\
31.4159265358979 & \quad \% \text{ format long g} \\
31.4159e+000 & \quad \% \text{ format shorteng} \\
31.4159265358979e+000 & \quad \% \text{ format longeng}
\end{align*}
\]

\[
\begin{align*}
>> \text{vpa}(10\pi) & \quad \% \text{ symbolic toolbox} \\
ans = & \\
31.415926535897932384626433832795 \\

>> \text{vpa}(10\pi,20) & \quad \% \text{ specify number of digits} \\
ans = & \\
31.415926535897932385
\end{align*}
\]

\[
\begin{align*}
>> \text{help format} \\
>> \text{help vpa} \\
>> \text{help digits}
\end{align*}
\]
When you press a key on your computer keyboard, the key that you press is translated to a binary code.

A = 1000001     (Decimal = 65)
a = 1100001      (Decimal = 97)
0 = 0110000     (Decimal = 48)
<table>
<thead>
<tr>
<th>Dec</th>
<th>Hex</th>
<th>Char</th>
<th>Dec</th>
<th>Hex</th>
<th>Char</th>
<th>Dec</th>
<th>Hex</th>
<th>Char</th>
<th>Dec</th>
<th>Hex</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>Null</td>
<td>32</td>
<td>20</td>
<td>Space</td>
<td>64</td>
<td>40</td>
<td>$</td>
<td>96</td>
<td>60</td>
<td>‘</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>Start of heading</td>
<td>33</td>
<td>21</td>
<td>!</td>
<td>65</td>
<td>41</td>
<td>A</td>
<td>97</td>
<td>61</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>Start of text</td>
<td>34</td>
<td>22</td>
<td>“</td>
<td>66</td>
<td>42</td>
<td>B</td>
<td>98</td>
<td>62</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td>03</td>
<td>End of text</td>
<td>35</td>
<td>23</td>
<td>#</td>
<td>67</td>
<td>43</td>
<td>C</td>
<td>99</td>
<td>63</td>
<td>c</td>
</tr>
<tr>
<td>4</td>
<td>04</td>
<td>End of transmit</td>
<td>36</td>
<td>24</td>
<td>$</td>
<td>68</td>
<td>44</td>
<td>D</td>
<td>100</td>
<td>64</td>
<td>d</td>
</tr>
<tr>
<td>5</td>
<td>05</td>
<td>Enquiry</td>
<td>37</td>
<td>25</td>
<td>%</td>
<td>69</td>
<td>45</td>
<td>E</td>
<td>101</td>
<td>65</td>
<td>e</td>
</tr>
<tr>
<td>6</td>
<td>06</td>
<td>Acknowledge</td>
<td>38</td>
<td>26</td>
<td>&amp;</td>
<td>70</td>
<td>46</td>
<td>F</td>
<td>102</td>
<td>66</td>
<td>f</td>
</tr>
<tr>
<td>7</td>
<td>07</td>
<td>Ascii bell</td>
<td>39</td>
<td>27</td>
<td>’</td>
<td>71</td>
<td>47</td>
<td>G</td>
<td>103</td>
<td>67</td>
<td>g</td>
</tr>
<tr>
<td>8</td>
<td>08</td>
<td>Backspace</td>
<td>40</td>
<td>28</td>
<td>(</td>
<td>72</td>
<td>48</td>
<td>H</td>
<td>104</td>
<td>68</td>
<td>h</td>
</tr>
<tr>
<td>9</td>
<td>09</td>
<td>Horizontal tab</td>
<td>41</td>
<td>29</td>
<td>)</td>
<td>73</td>
<td>49</td>
<td>I</td>
<td>105</td>
<td>69</td>
<td>i</td>
</tr>
<tr>
<td>10</td>
<td>0A</td>
<td>Line feed</td>
<td>42</td>
<td>2A</td>
<td>*</td>
<td>74</td>
<td>4A</td>
<td>J</td>
<td>106</td>
<td>6A</td>
<td>j</td>
</tr>
<tr>
<td>11</td>
<td>0B</td>
<td>Vertical tab</td>
<td>43</td>
<td>2B</td>
<td>+</td>
<td>75</td>
<td>4B</td>
<td>K</td>
<td>107</td>
<td>6B</td>
<td>k</td>
</tr>
<tr>
<td>12</td>
<td>0C</td>
<td>Form feed</td>
<td>44</td>
<td>2C</td>
<td>,</td>
<td>76</td>
<td>4C</td>
<td>L</td>
<td>108</td>
<td>6C</td>
<td>l</td>
</tr>
<tr>
<td>13</td>
<td>0D</td>
<td>Carriage return</td>
<td>45</td>
<td>2D</td>
<td>-</td>
<td>77</td>
<td>4D</td>
<td>M</td>
<td>109</td>
<td>6D</td>
<td>m</td>
</tr>
<tr>
<td>14</td>
<td>0E</td>
<td>Shift out</td>
<td>46</td>
<td>2E</td>
<td>.</td>
<td>78</td>
<td>4E</td>
<td>N</td>
<td>110</td>
<td>6E</td>
<td>n</td>
</tr>
<tr>
<td>15</td>
<td>0F</td>
<td>Shift in</td>
<td>47</td>
<td>2F</td>
<td>/</td>
<td>79</td>
<td>4F</td>
<td>O</td>
<td>111</td>
<td>6F</td>
<td>o</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>Data link escape</td>
<td>48</td>
<td>30</td>
<td>0</td>
<td>80</td>
<td>50</td>
<td>P</td>
<td>112</td>
<td>70</td>
<td>p</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>Device control 1</td>
<td>49</td>
<td>31</td>
<td>1</td>
<td>81</td>
<td>51</td>
<td>Q</td>
<td>113</td>
<td>71</td>
<td>q</td>
</tr>
<tr>
<td>18</td>
<td>12</td>
<td>Device control 2</td>
<td>50</td>
<td>32</td>
<td>2</td>
<td>82</td>
<td>52</td>
<td>R</td>
<td>114</td>
<td>72</td>
<td>r</td>
</tr>
<tr>
<td>19</td>
<td>13</td>
<td>Device control 3</td>
<td>51</td>
<td>33</td>
<td>3</td>
<td>83</td>
<td>53</td>
<td>S</td>
<td>115</td>
<td>73</td>
<td>s</td>
</tr>
<tr>
<td>20</td>
<td>14</td>
<td>Device control 4</td>
<td>52</td>
<td>34</td>
<td>4</td>
<td>84</td>
<td>54</td>
<td>T</td>
<td>116</td>
<td>74</td>
<td>t</td>
</tr>
<tr>
<td>21</td>
<td>15</td>
<td>Neg acknowledge</td>
<td>53</td>
<td>35</td>
<td>5</td>
<td>85</td>
<td>55</td>
<td>U</td>
<td>117</td>
<td>75</td>
<td>u</td>
</tr>
<tr>
<td>22</td>
<td>16</td>
<td>Synchronous idle</td>
<td>54</td>
<td>36</td>
<td>6</td>
<td>86</td>
<td>56</td>
<td>V</td>
<td>118</td>
<td>76</td>
<td>v</td>
</tr>
<tr>
<td>23</td>
<td>17</td>
<td>End trans. block</td>
<td>55</td>
<td>37</td>
<td>7</td>
<td>87</td>
<td>57</td>
<td>W</td>
<td>119</td>
<td>77</td>
<td>w</td>
</tr>
<tr>
<td>24</td>
<td>18</td>
<td>Cancel</td>
<td>56</td>
<td>38</td>
<td>8</td>
<td>88</td>
<td>58</td>
<td>X</td>
<td>120</td>
<td>78</td>
<td>x</td>
</tr>
<tr>
<td>25</td>
<td>19</td>
<td>End of medium</td>
<td>57</td>
<td>39</td>
<td>9</td>
<td>89</td>
<td>59</td>
<td>Y</td>
<td>121</td>
<td>79</td>
<td>y</td>
</tr>
<tr>
<td>26</td>
<td>1A</td>
<td>Substitution</td>
<td>58</td>
<td>3A</td>
<td>:</td>
<td>90</td>
<td>5A</td>
<td>Z</td>
<td>122</td>
<td>7A</td>
<td>z</td>
</tr>
<tr>
<td>27</td>
<td>1B</td>
<td>Escape</td>
<td>59</td>
<td>3B</td>
<td>;</td>
<td>91</td>
<td>5B</td>
<td>]</td>
<td>123</td>
<td>7B</td>
<td>{</td>
</tr>
<tr>
<td>28</td>
<td>1C</td>
<td>File separator</td>
<td>60</td>
<td>3C</td>
<td>&lt;</td>
<td>92</td>
<td>5C</td>
<td>\</td>
<td>124</td>
<td>7C</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>1D</td>
<td>Group separator</td>
<td>61</td>
<td>3D</td>
<td>=</td>
<td>93</td>
<td>5D</td>
<td>]</td>
<td>125</td>
<td>7D</td>
<td>}</td>
</tr>
<tr>
<td>30</td>
<td>1E</td>
<td>Record separator</td>
<td>62</td>
<td>3E</td>
<td>&gt;</td>
<td>94</td>
<td>5E</td>
<td>^</td>
<td>126</td>
<td>7E</td>
<td>_</td>
</tr>
<tr>
<td>31</td>
<td>1F</td>
<td>Unit separator</td>
<td>63</td>
<td>3F</td>
<td>?</td>
<td>95</td>
<td>5F</td>
<td>`</td>
<td>127</td>
<td>7F</td>
<td>`</td>
</tr>
</tbody>
</table>
Strings in MATLAB

MATLAB stores strings as an array of characters using the ASCII code.

Each letter in a string takes up two bytes (16 bits) and the two bytes are the binary representation of the decimal number listed in the ASCII table.

Try the following in MATLAB:

```matlab
>> month = 'August'
>> double(month)
```
input/output functions: `disp`, `input`

```matlab
>> x = 10; disp('the value of x is:'); disp(x);
the value of x is:
    10

>> x = input('enter x: ')  % numerical input
    enter x: 100         % 100 entered by user
    x =
        100

>> y = input('enter string: ', 's');  % string input
    enter string: abcd efg
>> y = input('enter string: ')       % string input
    enter string: 'abcd efg'
    y =
        abcd efg
```

`disp`, `input`
arrays and matrices are the most important data objects in MATLAB

We discuss briefly:

a) row and column vectors
b) transposition operator, '

c) colon operator, :
d) equally-spaced elements, linspace
e) accessing array elements
f) dynamic allocation & de-allocation
g) pre-allocation
The key to efficient MATLAB programming can be summarized in three words:

vectorize, vectorize, vectorize

and avoid all loops

Compare the two alternative computations:

\[
x = [2, -3, 4, 1, 5, 8]; \\
y = zeros(size(x)); \\
for n = 1:length(x) \\
    y(n) = x(n)^2; \\
end
\]

\[
x = [2, -3, 4, 1, 5, 8]; \\
y = x.^2;
\]

\[
\text{element-wise exponentiation } .^\wedge \\
\text{ordinary exponentiation } ^\wedge
\]

answer: \[y = [4, 9, 16, 1, 25, 64]\]
>> x = [0 1 2 3 4 5]          % row vector
x =
0 1 2 3 4 5

>> x = 0:5                    % row vector
x =
0 1 2 3 4 5

>> x = [0 1 2 3 4 5]'
% column vector, (0:5)'
x =
0
1
2
3
4
5

the prime operator, ', or transpose, turns row vectors into column vectors, and vice versa

caveat: ' is actually conjugate transpose, use dot-prime, .', for transpose w/o conjugation
\[
\text{>> } z = [i; 1+2i; 1-i] \quad \text{\% column vector}
\]
\[
z = \\
\begin{align*}
0 &+ 1.0000i \\
1.0000 &+ 2.0000i \\
1.0000 &- 1.0000i \\
\end{align*}
\]
\[
\text{>> z.' \quad \text{\% transpose without conjugation}}
\]
\[
\text{ans = }
\begin{align*}
0 &+ 1.0000i & &1.0000 &+ 2.0000i & &1.0000 &- 1.0000i \\
\end{align*}
\]
\[
\text{>> z' \quad \text{\% transpose with conjugation}}
\]
\[
\text{ans = }
\begin{align*}
0 &- 1.0000i & &1.0000 &- 2.0000i & &1.0000 &+ 1.0000i \\
\end{align*}
\]
\[
\text{>> (z.').'} \quad \text{\% same as (z').'} \quad \text{or, \ conj(z)}
\]
\[
\text{ans = }
\begin{align*}
0 &- 1.0000i \\
1.0000 &- 2.0000i \\
1.0000 &+ 1.0000i \\
\end{align*}
\]
about linspace:

\[ x = \text{linspace}(a, b, N+1); \]

is equivalent to:

\[ x = a : (b-a)/N : b; \]

i.e., \(N+1\) equally-spaced points in the interval \([a,b]\)

or, dividing \([a,b]\) into \(N\) equal sub-intervals

\[ x(n) = a + \left(\frac{b-a}{N}\right)(n-1), \quad n = 1, 2, \ldots, N+1 \]

\[
\begin{align*}
\gg x &= 0 : 0.2 : 1 & \text{\% in general, } x = a:s:b \\
\gg x &= \text{linspace}(0,1,6) & \text{\% see also logspace}
\end{align*}
\]

\[ x = \]

\[
\begin{bmatrix}
0 & 0.2000 & 0.4000 & 0.6000 & 0.8000 & 1.0000 \\
\end{bmatrix}
\]

6 points, 5 subintervals
\[
x = a : s : b;
\]
the number of subintervals within \([a,b]\) is obtained by rounding \((b-a)/s\), down to the nearest integer,
\[
N = \text{floor}\left((b-a)/s\right);
\]
length\((x)\) is equal to \(N+1\)
\[
x(n) = a + s*(n-1), \quad n = 1, 2, \ldots, N+1
\]

% before rounding, \((b-a)/s\) was in the three cases:
% \(1/0.3 = 3.3333\), \(1/0.4 = 2.5\), \(1/0.7 = 1.4286\)
Note: MATLAB array indices always start with 1 and may not be 0 or negative.

```matlab
>> x = [ 2, 5, -6, 10, 3, 4 ];
x(1), x(2), x(3), x(4), x(5), x(6)
```

Other languages, such as C/C++ and Fortran, allow indices to start at 0. For example, the same array would be declared/defined in C as follows:

```c
double x[6] = { 2, 5, -6, 10, 3, 4 };
x[0], x[1], x[2], x[3], x[4], x[5]
```

exception: logical indexing, discussed later
accessing array entries:

```matlab
>> x = [2, 5, -6, 10, 3, 4]
x =
    2     5    -6    10     3     4

>> length(x)    % length of x, see also size(x)
an =
    6

>> x(1)         % first entry
ans =
    2

>> x(3)         % third entry
ans =
   -6

>> x(end)       % last entry – need not know length
ans =
    4
```
accessing array entries:

>> x(end-3:end)                           % x = [2, 5, -6, 10, 3, 4]
ans =
   -6  10   3        4 % last four

>> x(3:5)                             % list third-to-fifth entries
ans =
   -6  10   3

>> x(1:3:end)                             % every third entry
ans =
     2  10

>> x(1:2:end)                             % every second entry
ans =
     2  -6   3
accessing array entries:

```matlab
>> x = [2, 5, -6, 10, 3, 4];

>> x(end:-1:1)    % list backwards, same as fliplr(x)
an =
    4  3  10  -6  5  2

>> x([3,1,5])       % list [x(3),x(1),x(5)]
an =
    -6  2  3

>> x(end+3) = 8
x =
    2  5  -6  10  3  4  0  0  0  8
```

automatic memory re-allocation
automatic memory allocation and de-allocation:

>> clear x

>> x(3) = -6
x =
    0     0    -6

>> x(6) = 4
x =
    0     0    -6     0     0     4

>> x(end) = []                  % delete last entry
x =
    0     0    -6     0     0

>> x = [2, 5, -6, 10, 3, 4];
>> x(3)=[]                      % delete third entry
x =
    2     5     10     3     4
pre-allocation

>> clear x
>> x = zeros(1,6)          % 1x6 array of zeros
x =
   0   0   0   0   0   0   0   0

>> x = zeros(6,1)          % 6x1 array of zeros
x =
   0   0   0   0   0   0   0   0

>> help zeros
>> help ones
clear x;
for k=[3,7,10]  
    x(k) = 3 + 0.1*k;
    disp(x);
end

0.0  0.0  3.3
0.0  0.0  3.3  0.0  0.0  0.0  3.7
0.0  0.0  3.3  0.0  0.0  3.7  0.0  0.0  4.0

x = zeros(1,10);  
for k=[3,7,10]  
    x(k) = 3 + 0.1*k;
    disp(x);
end

0.0  0.0  3.3  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  3.3  0.0  0.0  3.7  0.0  0.0  0.0  0.0
0.0  0.0  3.3  0.0  0.0  3.7  0.0  0.0  4.0  0.0

% k runs successively through the values of [3,7,10]  
% display current vector x

% pre-allocate x to length 10
First assignment will be posted on the course website on Saturday 28.10.2017

Due date: Wednesday 1.11.2017 11:55 PM

Try to use LaTeX for generating your report and send me the PDF

Recommended: Use the Overleaf online website for generating latex documents: https://www.overleaf.com