Engineering Analysis ENG 3420 Fall 2009

Dan C. Marinescu
Office: HEC 439 B
Office hours: Tu-Th 11:00-12:00
Lecture 5

- Last time:
  - Applications of Laplace Transform for solving differential equations.
  - Example.

- Today:
  - Internal representations of numbers and characters in a computer.
  - Arrays in Matlab
  - Matlab program for solving a quadratic equation

- Next Time
  - Roundoff and truncation errors
  - More on Matlab
Insight not numbers!!

- Engineering requires a quantitative analysis (ultimately some numbers) but the purpose of engineering analysis is to gain insight not numbers!!
- Engineering analysis is based on concepts from several areas of math and statistics:
  - Calculus (integration, differentiation, series)
  - Complex analysis
  - Differential equations
  - Linear algebra
  - Probability and statistics
- The purpose of this class is not to teach you Matlab, but to teach you how to use Matlab for solving engineering problems.
- Matlab is just a tool, an instrument. What good is to have a piano without being taught how to play…. 
- One must be familiar with the software tools but understand the math behind each method.
A 32-bit word can be used to represent:

1. a signed integer
2. a floating point number
3. a string of characters
4. a machine instruction

**1. Signed Integer**

- **bit 31** is the sign bit:
  - 0 → positive integer
  - 1 → negative integer

**Magnitude of the signed integer**

\[ b_{30} \times 2^{30} + b_{29} \times 2^{29} + \ldots + b_2 \times 2^2 + b_1 \times 2^1 + b_0 \times 2^0 \]

**2. Floating Point Number**

- **bit 31** is the sign bit:
  - 0 → positive integer
  - 1 → negative integer

**Floating point number**

\[ (-1)^{\text{sign}} \times \text{significant} \times 2^{\text{exponent}} \]

**3. ASCII Characters**

- 8 bit per character

**4. Machine Instruction**

- **Opcode** 8 bit
- **Addressing Information** 24 bit
Two’s complement representation of signed integers

- The 2’s complement representation of a
  - positive integer is the binary representation of the integer. For example: \((0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010)_2 = 2^{10}\)
  - negative integer is obtained by subtracting the binary representation of the integer from a large power of two (specifically, from \(2^N\) for an \(N\)-bit two's complement) and the adding 1

- Example: \(N=32\) ➔ the 2’s complement of \((-2)\)

  \[
  \begin{align*}
  (1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111)_{2} & - (0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010)_{2} \\
  = & (1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101)_{2}
  \end{align*}
  \]

  \[
  \begin{align*}
  (1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101)_{2} & + (0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001)_{2} \\
  = & (1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110)_{2}
  \end{align*}
  \]
Two’s complement representations of integers

(0000 0000 0000 0000 0000 0000 0000 0000)₂ = 0₁₀  
(0000 0000 0000 0000 0000 0000 0000 0001)₂ = 1₁₀  
(0000 0000 0000 0000 0000 0000 0000 0010)₂ = 2₁₀  

...........................................

(0111 1111 1111 1111 1111 1111 1111 1110)₂ = (2,147,483,646)₁₀  
(0111 1111 1111 1111 1111 1111 1111 1111)₂ = (2,147,483,647)₁₀  

(1000 0000 0000 0000 0000 0000 0000 0000)₂ = -(2,147,483,648)₁₀  
(1000 0000 0000 0000 0000 0000 0000 0001)₂ = -(2,147,483,647)₁₀  
(1000 0000 0000 0000 0000 0000 0000 0010)₂ = -(2,147,483,646)₁₀  

...........................................

(1111 1111 1111 1111 1111 1111 1111 1110)₂ = -(2)₁₀  
(1111 1111 1111 1111 1111 1111 1111 1111)₂ = -(1)₁₀
2’s complement of signed integers

- The advantages of two-s complement representation:
  - All negative numbers have the leftmost bit equal to 1 thus the hardware needs only to check one bit to determine if a number is positive or negative.

- When 32 bits are used to represent signed integers the range is: from \(-(2,147,483,648)_{10}\) to \((2,147,483,648)_{10}\)
Floating point numbers

- The range is limited. For example, with 32 bit the single precision representation allows us to represent floating point numbers in the range:
  
  \[
  2.0 \times 10^{38} - 2.0 \times 10^{-38}
  \]
  
  About 6 to 7 significant digits.

- Double precision representation ➔ the Mantissa (significant) uses 32+24 = 56 bits. About 14 significant digits.
Arrays

- In addition to scalars, we can use vectors (one-dimensional arrays), or matrices (two-dimensional arrays). Examples:
  
  ```
  >> a = [1 2 3 4 5]   % specify the elements of the array individually separated by spaces or commas
         a =
            1   2   3   4   5
  >> a =1:2:10     % specify, the first element, the increment, and the # of elements
         a =
            1   3   5   7   9
  >> a=[1 3 5 7 9]     % specify the elements separated by commas or spaces
         a =
            1   3   5   7   9
  ```

- Arrays are stored column wise (column 1, followed by column 2, etc.). A semicolon marks the end of a row.
  
  ```
  >> a=[1;2;3]
         a =
            1
            2
            5
  ```
The transpose of an array

- The transpose (‘) operator transforms rows into columns and columns into rows;

```matlab
>> b = [2 9 3 ;4 16 5 ;6 13 7 ;8 19 9;10 11 11]
b =
   2   9   3
   4  16   5
   6  13   7
   8  19   9
  10  11  11

>> b'  % The transpose of matrix b
ans =
   2   4   6   8  10
   9  16  13  19  11
   3   5   7   9  11

>> c = [2 9 3 4 16; 5 6 13 7 8; 19 9 10 11 11]
c =
   2   9   3   4  16
   5   6  13   7   8
  19   9  10  11   1
```
Element by element array operations

- Both arrays must have the same number of elements.
  1. multiplication (".*")
  2. division ("./") and
  3. exponentiation (notice the "." before "+", "/", or "^").

```matlab
>> b.* b
ans =
   4   81    9
  16  256   25
  36  169   49
  64  361   81
 100  121  121
>> b ./ b
ans =
   1   1   1
   1   1   1
   1   1   1
   1   1   1
   1   1   1
   1   1   1
```
How to identify an element of an array

>> b(4,2)   % element in row 4 column 2 of matrix b
ans =
    19
>> b(9)    % the 9-th element of b (recall that elements are stored column wise)
ans =
    19
>> b(5)    % the 5-th element of b (recall that elements are stored column wise)
ans =
    10
>> b(17)
??? Attempted to access b(17); index out of bounds because numel(b)=15.

>> whos
    Name      Size             Bytes  Class       Attributes
    b         5x3               120  double
Array Creation – with Built In Functions

- `zeros(r,c)` ➔ create an $r$ row by $c$ column matrix of zeros
- `zeros(n)` ➔ create an $n$ by $n$ matrix of zeros
- `ones(r,c)` ➔ create an $r$ row by $c$ column matrix of ones
- `ones(n)` ➔ create an $n$ by $n$ matrix of ones
- `help elmat` ➔ gives a list of the elementary matrices
Colon Operator

- Create a linearly spaced array of points:
  \[ \text{start:diff\text{valu}}:\text{limit} \]
  - \text{Start} \rightarrow \text{first value in the array,}
  - \text{diff\text{valu}} \rightarrow \text{difference between successive values in the array, and}
  - \text{limit} \rightarrow \text{boundary for the last value}

- Example
  \[ \gg 1:0.6:3 \]
  \[ \text{ans} = \]
  \[
  \begin{array}{cccc}
  1.0000 & 1.6000 & 2.2000 & 2.8000 \\
  \end{array}
  \]
Colon Operator (cont’d)

- **If diffval** is omitted, the default value is 1:
  >> 3:6
  ans =
  3  4  5  6

- **To create a decreasing series, diffval must be negative:**
  >> 5:-1.2:2
  ans =
  5.0000  3.8000  2.6000

- **If start+diffval>limit** for an increasing series or
  **start+diffval<limit** for a decreasing series, an empty matrix is returned:
  >> 5:2
  ans =
  Empty matrix: 1-by-0

- **To create a column, transpose the output of the colon operator, not the limit value; that is, (3:6)’ not 3:6’**
Array Creation - `linspace`

- `linspace(x1, x2, n)` ➔ create a linearly spaced row vector of `n` points between `x1` and `x2`

  Example
  
  ```matlab
  >> linspace(0, 1, 6)
  ans =
       0    0.2000    0.4000    0.6000    0.8000    1.0000
  ```

- If `n` is omitted, 100 points are created.
- To generate a column, transpose the output of the `linspace` command.
Array Creation - \texttt{logspace}

- \texttt{logspace(x1, x2, n)} \rightarrow create a logarithmically spaced row vector of \textit{n} points between $10^{x1}$ and $10^{x2}$

- Example:
  \begin{verbatim}
  >> logspace(-1, 2, 4)
  \end{verbatim}
  \begin{verbatim}
  ans =
  0.1000  1.0000  10.0000  100.0000
  \end{verbatim}

- If \textit{n} is omitted, 100 points are created.

- To generate a column vector, transpose the output of the \texttt{logspace} command.
Arithmetic operations on scalars and arrays

- The operators, in order of priority:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>^</td>
<td>$4^2 = 16$</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>$-8 = -8$</td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>$2\pi = 6.2832$</td>
<td></td>
</tr>
<tr>
<td>/</td>
<td>$\pi/4 = 0.7854$</td>
<td></td>
</tr>
<tr>
<td>\</td>
<td>$6\div 2 = 0.3333$</td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>$3+5 = 8$</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>$3-5 = -2$</td>
<td></td>
</tr>
</tbody>
</table>
Complex numbers

- Arithmetic operations can be performed on complex numbers:
  \[ x = 2 + i \times 4; \ (\text{or} \ 2 + 4i, \ \text{or} \ 2 + j \times 4, \ \text{or} \ 2 + 4j) \]

\[
\begin{align*}
y &= 16; \\
3 \times x \\
\text{ans} &= \\
&\quad 6.0000 + 12.0000i \\
x + y \\
\text{ans} &= \\
&\quad 18.0000 + 4.0000i \\
x' \\
\text{ans} &= \\
&\quad 2.0000 - 4.0000i
\end{align*}
\]
MATLAB can also perform operations on vectors and matrices.

The * operator for matrices is defined as the *outer product* or what is commonly called “matrix multiplication.”

- The number of columns of the first matrix must match the number of rows in the second matrix.
- The size of the result will have as many rows as the first matrix and as many columns as the second matrix.
- The exception to this is multiplication by a 1x1 matrix, which is actually an array operation.

The ^ operator for matrices results in the matrix being matrix-multiplied by itself a specified number of times.

- Note - in this case, the matrix must be square!
Help built-in function

- **help** ➔ gives information about both what exists and how those functions are used:
  - **help elmat** ➔ list the elementary matrix creation and manipulation functions, including functions to get information about matrices.
  - **help elfun** ➔ list the elementary math functions, including trig, exponential, complex, rounding, and remainder functions.

- **Lookfor** ➔ search help files for occurrences of text (useful if you know a function’s purpose but not its name)
Element-by-Element Calculations

- *Element-by-element* operations ➔ carry out calculations item by item in a matrix or vector.
- Array multiplication (.*), array division operators (./),
- and array exponentiation (.^) (raising each element to a corresponding power in another matrix)
- Both matrices must be the same size or one of the matrices must be 1x
M-files; Script and Function Files

- MATLAB allows to store commands in text files called *M-files*; the files are named with a `.m` extension.
- Two main types of M-files
  - Script files
  - Function files

- A *script file* ➔ set of MATLAB commands saved on a file; when MATLAB runs a script file, it is as if you typed the characters stored in the file on the command window.

- Function files ➔
  - accept input arguments from and return outputs to the command window,
  - variables created and manipulated within the function do not impact the command window.
Function File Syntax

- The general syntax for a function is:
  
  ```matlab
  function outvar = funcname(arglist)
  % helpcomments
  statements
  outvar = value;
  
  where
  
  - `outvar`: output variable name
  - `funcname`: function’s name
  - `arglist`: input argument list - comma-delimited list of what the function calls values passed to it
  - `helpcomments`: text to show with help `funcname`
  - `statements`: MATLAB commands for the function
  ```
Quadratic equations

- $ax^2+bx+ c=0$

- The roots:
  \[ x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \]

- Special case: $a=0$ and $b=0 \implies bx + c=0 \implies x = - c/b$

- Special case: $b=0 \implies ax^2+ c=0
  \[ x_{1,2} = \pm i \sqrt{\frac{c}{a}} \]
Matlab program for solving quadratic equations

Function quadroots(a,b,c) % Input: the coefficients (a,b,c)
% Output: the real and imaginary parts of the solution: %(R1,I1),(R2,I2)
if a==0 % special case
  if b =~0
    x1=-c/b
  else
    error('a and b are zero')
  end
else
  d = b^2 – 4 * a *c;
  if  d > = 0 % real roots
    R1 = (-b + sqrt(d)) / (2*a)
    R2 = (-b - sqrt(d)) / (2*a)
  else % complex roots
    R1 = -b/(2*a)
    l1= sqrt(abs(d)) /(2*a)
    R2 = R1
    l2= -l1
  end
end