Penn
UNIVERSITY of PENNSYLVANIA

# ENGR 105: Introduction to Scientific Computing

## Some Simple Linear Algebra, Interfacing with MATLAB, Numerical Accuracy

Dr. Graham. E. Wabiszewski

## Announcements

Canvas / Piazza

- **Anyone not registered with Canvas or Piazza?**
- **If not - email me**
- Canvas access usually takes ~1 day from official course registration
- Take the "Initial Assessment Quiz" if you haven't done so already

## Homework

### HW #1

- Currently posted to Canvas (demo on how to access)
- Due by midnight on 9/11
- Upload to Canvas as a word document - include answers, code, plots
- Make sure the assignment is formatted reasonably

### Reading HW #1

- Finish by 9/11
- Fair game for lab quiz
- Demo on how to access

## First lab quiz

Lab quiz #1

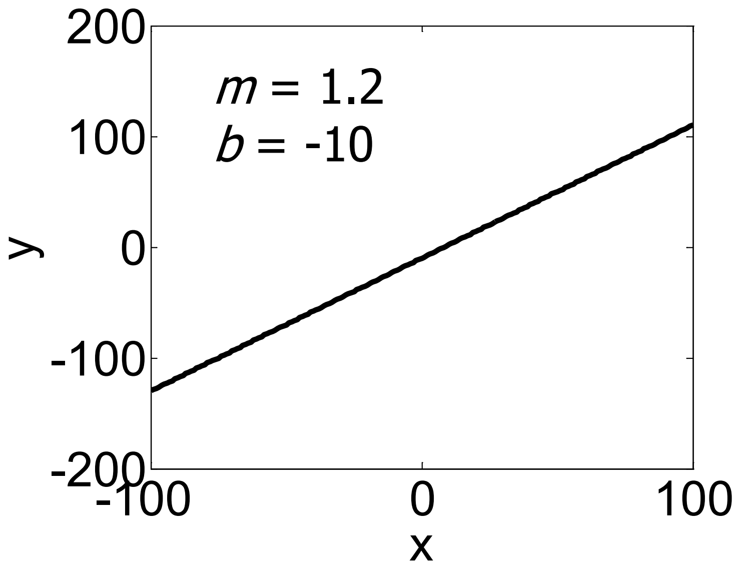
- Wednesday 9/11
- No Googling, looking up answers, or using Matlab (only have Canvas open in the browser window)
- Fair game: lecture slides up to 9/9, Ch.1 and Ch.2 of Essential Matlab, article "All I really need to know about pair programming I learned in kindergarten"

## Linear algebra primer

You are likely familiar with canonical linear equations of the form:

$$ax + by = c$$

$$y = mx + b$$



## What is the solution to two linear equations with two independent variables?

$$a_1 x + b_1 y = c_1 \quad (1)$$

$$a_2 x + b_2 y = c_2 \quad (2)$$

## Linear algebra primer

For a simple system you can use substitution or elimination!

$$x = \frac{c_1 - b_1 y}{a_1}$$

(3) solving for *x* in (1)

$$a_2\left(\frac{c_1 - b_1 y}{a_1}\right) + b_2 y = c_2$$

(4) substitute (3) into (2)

## Linear algebra primer

$$y = \frac{c_2 - \dfrac{a_2 c_1}{a_1}}{b_2 - \dfrac{a_2 b_1}{a_1}}$$

(5) solve for y

## Linear algebra primer

$$a_1 x + b_1 \left( \frac{c_2 - \dfrac{a_2 c_1}{a_1}}{b_2 - \dfrac{a_2 b_1}{a_1}} \right) = c_1$$

(6) substitute (5) into (1) or (2)

$$x = \frac{1}{a_1} \left[ c_1 - b_1 \left( \frac{c_2 - \dfrac{a_2 c_1}{a_1}}{b_2 - \dfrac{a_2 b_1}{a_1}} \right) \right]$$

(7) solve for x

## Linear algebra primer

There is a simpler way - matrices....

$$a_1 x + b_1 y = c_1$$
$$a_2 x + b_2 y = c_2$$

...can be represented as...

$$\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$$

## Linear algebra primer

More generally, for any linear system of *n* variables with *n* equations

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n} &= b_1 \\
a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n} &= b_2 \\
&\vdots \\
a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn} &= b_n
\end{aligned}
$$

## Linear algebra primer

...can be represented as...

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \cdots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}$$

...or...

$$AX = B$$

## Linear algebra primer

$$A^{-1}AX = A^{-1}B$$

Multiply by the inverse of *A*

$$IX = A^{-1}B$$

Resolve to *X*

$$X = A^{-1}B$$

Solution to *X*

Finding solutions to linear sets of equations is "easy" in Matlab

## Opening a new script / M-file

Command line method

- ex. `edit newScript`

From the toolbar

- Demo

## Running a script / interrupting a script

How to execute scripts / M-files

- F5 (demo)
- Run the script from the command window (demo)
- Left click (demo)

Interrupt a loop

- “ctrl+c” (demo)

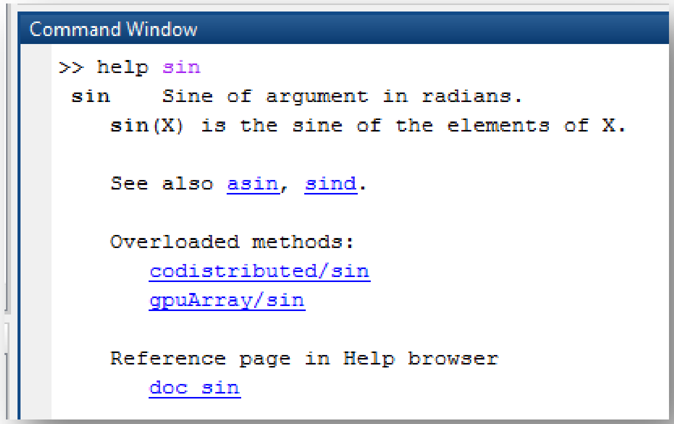
## Accessing MATLAB help

Methods previously mentioned

- Accessing MATLAB help from "view product documentation"
- The Google method

Alternative methods

- Help from the command line (demo)
- Highlight and press "F1" (demo)



```
>> help sin
 sin    Sine of argument in radians.
    sin(X) is the sine of the elements of X.

    See also asin, sind.

    Overloaded methods:
       codistributed/sin
       gpuArray/sin

    Reference page in Help browser
       doc sin
```

## Opening existing or user defined functions

Several methods are available to run a script:

- “ctrl+d” (demo)
- Be wary of modifying existing scripts/functions - unless intentional

## A detailed look at memory

- Each of the memory locations in the computer can be thought of as storing a finite sequence of binary digits or bits.
- Each of these storage locations has a particular size say 8 bits or 16 bits or 32 or 64 bits.
- A group of 8 bits is referred to as a byte.

| Address | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 1071 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| 1072 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 1073 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1074 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |

## Representing numbers in binary

- We can represent any non-negative integer we want in binary using standard place value convention where the exponents is now 2 instead of 10 as it is for regular decimal numbers

| Address | $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
|---|---|---|---|---|---|---|---|---|
| 1071 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| 1072 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 1073 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1074 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |

$$1011_2 = 8 + 2 + 1 = 11_{10}$$

$$110010_2 = 32 + 16 + 2 = 50_{10}$$

## Overflow

- Note that in a computer system the size of a memory location is fixed when it is allocated at 8, 16, 32, 64 etc. bits
- This limits the range of values that can be stored in that location. For example an 8 bit storage location can only store unsigned integers between 0 and 255.
- If you try to store a larger or smaller value you will run into the limits

```
X = uint8(78); Y = uint8(190);
Z = X + Y; overflow problem value will
 be clipped to 255
```

## Storage limits

8 bit unsigned values

  0 – 255

16 bit unsigned values

  0 – 65535

32 bit unsigned values

  0 – approximately 4 billion
  ($2^{32}$ – 1)

## Computer arithmetic

- Since arithmetic operations are carried out in the computer using finite storage locations the results can be limited by range or imprecise
- It is the programmers job to make sure that she allocates appropriate storage locations for the task at hand and uses those variables appropriately

## Floating point representations

"God created the integers, all the rest is the work of man"
Leopold Kronecker

- In addition to integer values (positive and negative) scientific computation also involves rational numbers like
  0.335 and -27.890
- Irrational numbers like pi and e are typically approximated by rational values
- To do this we appeal to scientific notation and represent these numbers in a canonical form
  $2.1345 \times 10^{17}$

## Scientific notation

- Note that numbers in scientific notation have the following components
  - A sign – positive or negative
  - A mantissa – ex. 2.1345
  - An exponent – ex. 17
- All of these pieces can be represented as integers
  - This is how rational numbers are represented in the computer

## Binary floating point numbers

- Binary floating point numbers consist of a sign bit, a fractional field and an exponent field. Note that the exponent in this case is base 2 not base 10.
- IEEE single precision numbers require 32 bits

  1 bit sign 8 bits exponent 23 bits mantissa
- IEEE double precision numbers require 64 bits

  1 bit sign 11 bits exponent 52 bits mantissa
- Double precision numbers have a larger range and precision than single precision numbers as the name implies so by default MATLAB represents numbers in this format

  ex. $-1.010110101 \times 2^{78}$

## Issues with floating point arithmetic

- Note that once again floating point numbers are restricted to a fixed number of bits and this has important consequences
- Consider the decimal equivalent of having a representation where you were restricted to 2 digits after the decimal point

$$\begin{array}{r} 5.43 \times 10^6 \\ +\ 7.43 \times 10^4 \end{array} \qquad \begin{array}{r} 5.43 \times 10^6 \\ +\ 0.0743 \times 10^6 \\ =\ 5.5043 \times 10^6 \end{array}$$

In this case when the result is stored the last two digits in red will be lost after rounding
(Rounding error DEMO)

## Issues with floating point arithmetic

- It is important to remember that because floating point numbers are restricted in size every arithmetic operation involving floating point numbers, addition, subtraction, multiplication, division etc. has the potential to lose information
- Hence, the results of most arithmetic operations involving fractions on the computer are to be viewed as inherently approximate
- **Take home message : Computers are not as good at arithmetic as you may think they are!!!**

## Numerical analysis

- In order to get reliable results out of computations it is important to design ones code very carefully to avoid or minimize the impact of roundoff error and other imprecisions.
- Good numerical codes are designed by specialists known as numerical analysts who do this

## MATLAB number types

- You can use the `whos` function in MATLAB to show how various variables are stored
- The MATLAB types indicate the binary storage format

  uint8 – unsigned 8 bit storage location

  int64 – signed 64 bit integer storage location

  double – double precision floating point number

  single – single precision floating point number
- There are also associated functions you can use to specify the storage type of a variable

  `x = int32(67)`; creates a 32 bit integer variable
- Note that there is a tradeoff between size in bytes and the capacity of the numerical format

## Arrays and numerical types

- When an array variable is created all of the numeric values in that array share the same numeric type. For example we can talk about an array of uint16s or an array of doubles.
  - `x = uint16([1 3 4 5 89]);` make an array of 5 uint16s
  - `x = single(0:0.1:100);` make an array of single precision numbers

## It’s all bits

- All information in the computer is represented in the form of bits
- While we have been talking about numbers, every other type of data, text, images, sound etc. is eventually broken down and represented as a series of numbers

## Representing text

- The ASCII code associates a number with every letter and symbol you may care to type
- It also encodes symbols for things like newline and carriage return that don’t print but affect text
- In MATLAB we can think of a string of text as an array of integral numbers - MATLAB represents each character as a 16 bit unicode value

## Representing sound

- Audio can be represented as an array of numbers representing sound samples over time
- Since sound is a transverse wave we can think of these numbers as representing the displacement of the wave over time

## Images

- Images are actually 2 dimensional arrays of numeric values
- Color images actually store 3 numbers at each pixel for the red green and blue channels of the image

# A few useful MATLAB commands

`whos`

Lists all of the variables currently in your workspace and shows you their types

`clear`

Clears all of the variables in your workspace – you can also use this to clear specific variables

`clc`

Just clears the command window – has no effect on the workspace