# Computers and programming languages introduction

Eugeniy E. Mikhailov

The College of William & Mary

Lecture 01

# Class goals and structure

Primary purpose

- learn to to specify a problem
- break it up into algorithmic pieces
- implement a program to execute these pieces
    - learn Matlab

# Class goals and structure

Primary purpose

- learn to to specify a problem
- break it up into algorithmic pieces
- implement a program to execute these pieces
    - learn Matlab

Structure

- first we learn basics of Matlab as programming language (couple weeks)
- then learn numerical analysis basics while keep mastering Matlab

# Class goals and structure

Primary purpose

- learn to to specify a problem
- break it up into algorithmic pieces
- implement a program to execute these pieces
  - learn Matlab

Structure

- first we learn basics of Matlab as programming language (couple weeks)
- then learn numerical analysis basics while keep mastering Matlab

Weekly schedule

- Monday, Wednesday: normal lecture hours
- Friday: short lecture, lab, hands on

# Building blocks

# Building blocks

To learn a language we need to practice and use this language

- a lot of weight on homeworks and projects

# Building blocks

To learn a language we need to practice and use this language

- a lot of weight on homeworks and projects

No final exam

- Final project defense instead
- December 6 at 14:00 in Small Hall 111

# Building blocks

To learn a language we need to practice and use this language

- a lot of weight on homeworks and projects

No final exam

- Final project defense instead
- December 6 at 14:00 in Small Hall 111

Grades contribution

- Homeworks: 15%
- Midterm projects: 60%
- Final project: 25%

# Building blocks

To learn a language we need to practice and use this language

- a lot of weight on homeworks and projects

No final exam

- Final project defense instead
- December 6 at 14:00 in Small Hall 111

Grades contribution

- Homeworks: 15%
- Midterm projects: 60%
- Final project: 25%

Assignments and lecture notes will be posted on my homepage

- http://physics.wm.edu/∼evmik/

# Homeworks and midterm project deadlines

- **due date**: corresponding Monday at 1:00pm for email submission
- report to be submitted via email as well as a carbon copy to be collected at the beginning of the Monday class

If there is no listings and no algorithms/data files, you will get zero points.

**Late submission penalties**

For each consequent day after the due date there will be a penalty (10% out of maximum possible score). Even if submission happens 1 minute after the due date, it holds 1 day penalty.

**Projects homework preparation recommendation**

Do not wait till the last day to finish your exercise. Programs almost never work at the first try and require quite a lot of time to debug.

# Collaboration and grading scale

- Collaborations are not permitted for homeworks.
- Projects to be done in group of 2 or 3 persons. This is the time to actively discuss and cooperate. Only one report per such group is needed.
  - But everyone is expected to have a full understanding of the project.
  - Be ready to answer questions related to the project without your group support.

Grading scale

| **Grade** | **percentage** | **Grade** | **percentage** | **Grade** | **percentage** |
|---|---|---|---|---|---|
| | | A | 94-100 | A- | 90-94 |
| B+ | 87-90 | B | 84-87 | B- | 80-84 |
| C+ | 77-80 | C | 74-77 | C- | 70-74 |
| D+ | 67-70 | D | 64-67 | D- | 60-64 |
| F | $<60$ | | | | |

# Recommended reading

Everything required for this class will be provided during lecture times.
Two **optional** books for your own references.
A short Matlab reference book: "Getting Started with MATLAB: A Quick Introduction for Scientists and Engineers" bu Rudra Pratap

- ISBN-10: 0199731241
- ISBN-13: 978-0199731244

A more extended treatment of numerical algorithm with Matlab: "Numerical Methods in Engineering with MATLAB" by Jaan Kiusalaas

- ISBN-10: 0521191335
- ISBN-13: 978-0521191333

# Early history of computing

# Early history of computing

Computers use to be humans

# Early history of computing

Computers use to be humans
Computing aids - no programing possible

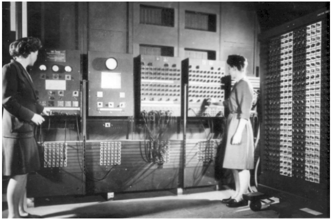
- abacus
- sliding ruler
- pre-calculated tables of function (logarithm, trigonometry . . . )
- mechanical calculators

# Early history of computing

Computers use to be humans
Computing aids - no programing possible

- abacus
- sliding ruler
- pre-calculated tables of function (logarithm, trigonometry ...)
- mechanical calculators

Modern computers appear at 1946 -ENIAC (Electronic Numerical Integrator And Computer)



- weight: 30 tons
- cost: $500,000 ($6,000,000 adjusted)
- power consumption: 150 kW

# ENIAC vs modern PC

Speed measured in operations per second

ENIAC

- 5000 additions
- 357 multiplications
- 38 divisions

# ENIAC vs modern PC

Speed measured in operations per second

ENIAC

- 5000 additions
- 357 multiplications
- 38 divisions

Athlon 3000+ (2GHz)

- 70,000,000 additions
- 70,000,000 multiplications
- 50,000,000 divisions
- 15,000,000 sin operations

# Common features of modern computer

- Central Processing Unit (CPU)
- memory
  - holds data and executable code
- data input and output
- same hardware can do different calculation sequences
- usually use binary system
- programmable for any general task

# Common features of modern computer

- Central Processing Unit (CPU)
- memory
  - holds data and executable code
- data input and output
- same hardware can do different calculation sequences
- usually use binary system
- programmable for any general task

Speed measured in FLOPS (the number of floating point operations per second) which usually proportional to the clock frequency.

# Common features of modern computer

- Central Processing Unit (CPU)
- memory
  - holds data and executable code
- data input and output
- same hardware can do different calculation sequences
- usually use binary system
- programmable for any general task

Speed measured in FLOPS (the number of floating point operations per second) which usually proportional to the clock frequency.

Different computer architectures (AMD, Mac, Intel, ARM . . . ) have different proportionality coefficient.

# Common features of modern computer

- Central Processing Unit (CPU)
- memory
  - holds data and executable code
- data input and output
- same hardware can do different calculation sequences
- usually use binary system
- programmable for any general task

Speed measured in FLOPS (the number of floating point operations per second) which usually proportional to the clock frequency.

Different computer architectures (AMD, Mac, Intel, ARM . . . ) have different proportionality coefficient.

My 2 GHz AMD PC can do about 50 MegaFLOPS

# Computers . . .

*Computers are incredibly fast,*

# Computers . . .

*Computers are incredibly fast, accurate, and*

# Computers . . .

*Computers are incredibly fast, accurate, and stupid.*

# Computers . . .

*Computers are incredibly fast, accurate, and stupid. Humans beings are incredibly slow,*

# Computers . . .

*Computers are incredibly fast, accurate, and stupid. Humans beings are incredibly slow, inaccurate,*

# Computers . . .

*Computers are incredibly fast, accurate, and stupid. Humans beings are incredibly slow, inaccurate, and brilliant.*

# Computers . . .

*Computers are incredibly fast, accurate, and stupid. Humans beings are incredibly slow, inaccurate, and brilliant. Together they are powerful beyond imagination.*

*Leo Cherne (1969)*

*Computers are incredibly fast, accurate, and stupid. Humans beings are incredibly slow, inaccurate, and brilliant. Together they are powerful beyond imagination.*

*Leo Cherne (1969)*

Thus

Computer is not a substitute for a brain

# Programming languages overview

There are hundreds programming languages.

# Programming languages overview

There are hundreds programming languages.

- Super low-level language
  - binary code
    - the only thing which computers understand
    - each instruction looks like a number
    - usually it is not human readable

# Programming languages overview

There are hundreds programming languages.

- Super low-level language
  - binary code
    - the only thing which computers understand
    - each instruction looks like a number
    - usually it is not human readable
- low-level languages
  - assembler (human readable binary code translation)
  - Fortran, LISP, C, C++, Forth

# Programming languages overview

There are hundreds programming languages.

- Super low-level language
  - binary code
    - the only thing which computers understand
    - each instruction looks like a number
    - usually it is not human readable
- low-level languages
  - assembler (human readable binary code translation)
  - Fortran, LISP, C, C++, Forth
- higher-level languages
  - Tcl, Java, JavaScript, PHP, Perl, Python

# Programming languages overview

There are hundreds programming languages.

- Super low-level language
  - binary code
    - the only thing which computers understand
    - each instruction looks like a number
    - usually it is not human readable
- low-level languages
  - assembler (human readable binary code translation)
  - Fortran, LISP, C, C++, Forth
- higher-level languages
  - Tcl, Java, JavaScript, PHP, Perl, Python
- Unfortunately none of them serves all needs.

# Programming languages implementations

**Compiled**

- generate computers binary code
  - it takes time
- faster execution time
- a bit harder to debug
- if you find and fixed an error (bug) you need to recompile
- Examples: Assembler, C, C++, Fortran

# Programming languages implementations

**Compiled**

- generate computers binary code
  - it takes time
- faster execution time
- a bit harder to debug
- if you find and fixed an error (bug) you need to recompile
- Examples: Assembler, C, C++, Fortran

**Interpreted**

- No compilation
- interpretation to machine code per instruction
- slow (since you have to interpret same instruction over and over)
- cross-platform code
- Examples: Perl, JavaScript, Lua, Php, Tcl, Shells, Matlab

# Programming languages implementations

**Compiled**

- generate computers binary code
  - it takes time
- faster execution time
- a bit harder to debug
- if you find and fixed an error (bug) you need to recompile
- Examples: Assembler, C, C++, Fortran

**just-in-time compilation**

- middle ground
- compile once to bytecode
- cross-platform
- Examples: Java, Python

**Interpreted**

- No compilation
- interpretation to machine code per instruction
- slow (since you have to interpret same instruction over and over)
- cross-platform code
- Examples: Perl, JavaScript, Lua, Php, Tcl, Shells, Matlab

# Matlab as a language of choice

Matlab (matrix laboratory)

# Matlab as a language of choice

Matlab (matrix laboratory)

**Pro**

- interpreted
  - easy to use and debug
- quite fast if done right, since main functions are compiled
- large selection of scientific related functions
- built in graphics/plotting
- Turing complete (you can do with it everything which computer is capable)
- designed to do numerical calculations

# Matlab as a language of choice

Matlab (matrix laboratory)

**Pro**

- interpreted
  - easy to use and debug
- quite fast if done right, since main functions are compiled
- large selection of scientific related functions
- built in graphics/plotting
- Turing complete (you can do with it everything which computer is capable)
- designed to do numerical calculations

**Contra**

- interpreted
  - could be slow if programmed inefficiently
- Not free to modify internals
- quite fast since for main functions it calls a compiled code
- rudimentary symbolic calculations

# Matlab: where to get

- Free for W&M students
- available for Mac and Windows
- visit `http://www.wm.edu/offices/it/a-z/software/`
- go to ‘Licensed software”
- choose appropriate “Math & Statistics” Software section
- download Matlab

# Matlab: where to get

- Free for W&M students
- available for Mac and Windows
- visit `http://www.wm.edu/offices/it/a-z/software/`
- go to 'Licensed software"
- choose appropriate "Math & Statistics" Software section
- download Matlab

Please, do it before this Friday class, also do not forget to bring your notebooks/laptops with you for Friday classes.

# Discretization - The main weakness of computers

- coming from resources limitation

For example:

$$1/6 = 0.1666666666666666\cdots$$

But computer has limited amount of memory. Thus it cannot hold infinite amount of digits and has to truncate somewhere.
Let's say it can hold only 4 significant digits.

$$1/6 = 0.1667_c$$

This called round off error due to truncation/rounding. Then for computer

$$1/6 = 1/5.999$$

or

$$\mathbf{0.1667}123 = \mathbf{0.1667}321 = \mathbf{0.1667}222 = \mathbf{0.1667}111$$

or even more interesting

$$20 \times (1/6) - 20/6 = 20 \times 0.1667 - 3.333 = 3.334 - 3.333 = 10^{-4}$$

# Binary representation - why PHYS 256

Modern general purpose computers use binary representation

- bit is a smallest unit of information
- bit value is either 0 or 1

Bit is too small so we use byte

- byte = 8 bits stitched together
- byte can represent values in the range $-128 \cdots 0 \cdots 127$
- the major (the left most) bit usually holds the sign ($s$) of the number
  - 0: means positive
  - 1: means negative
- $01001010_2$
- decimal representation $01001010_2$ = $(-1)^0 \times (0 \times 2^0 + 1 \times 2^1 + 0 \times 2^2 + 1 \times 2^3 + 0 \times 2^4 + 0 \times 2^5 + 1 \times 2^6)$ = $2 + 8 + 64$ =74

# Binary representation (cont.)

Byte is clearly to small to be used for real life computation.
Matlab uses 8 bytes or 64 bits for number representation

- available range $-2,147,483,648 \cdots 0 \cdots 2,147,483,647$
- you can find this range by executing `intmin` and `intmax`
- notice that you cannot use numbers outside of this range
  - $2,147,483,647 + 10 = 2,147,483,647$
  - this is called overflow error

# Float numbers representation

What to do if you need to store a float number?

# Float numbers representation

What to do if you need to store a float number?
For example $-123.765 \times 10^{12}$

# Float numbers representation

What to do if you need to store a float number?
For example $-123.765 \times 10^{12}$

- First convert it to scientific notation
  - $-1.23765 \times 10^{14}$

# Float numbers representation

What to do if you need to store a float number?
For example $-123.765 \times 10^{12}$

- First convert it to scientific notation
  - $-1.23765 \times 10^{14}$
- truncate it to certain number of significant digits
  - let use 4 for example (actually 17 decimals for 64 bits float number)
  - $-1.237 \times 10^{14}$

# Float numbers representation

What to do if you need to store a float number?
For example $-123.765 \times 10^{12}$

- First convert it to scientific notation
  - $-1.23765 \times 10^{14}$
- truncate it to certain number of significant digits
  - let use 4 for example (actually 17 decimals for 64 bits float number)
  - $-1.237 \times 10^{14}$
- resulting number should have a form $(-1)^s \times c \times b^q$
  - where $s$ is a sign bit (1 in our case)
  - c is mantissa or coefficient (1.237)
  - b is the base (10)
  - q is the exponent (14)

# Float numbers representation

What to do if you need to store a float number?
For example $-123.765 \times 10^{12}$

- First convert it to scientific notation
  - $-1.23765 \times 10^{14}$
- truncate it to certain number of significant digits
  - let use 4 for example (actually 17 decimals for 64 bits float number)
  - $-1.237 \times 10^{14}$
- resulting number should have a form $(-1)^s \times c \times b^q$
  - where $s$ is a sign bit (1 in our case)
  - c is mantissa or coefficient (1.237)
  - b is the base (10)
  - q is the exponent (14)

# Float numbers representation

What to do if you need to store a float number?
For example $-123.765 \times 10^{12}$

- First convert it to scientific notation
  - $-1.23765 \times 10^{14}$
- truncate it to certain number of significant digits
  - let use 4 for example (actually 17 decimals for 64 bits float number)
  - $-1.237 \times 10^{14}$
- resulting number should have a form $(-1)^s \times c \times b^q$
  - where $s$ is a sign bit (1 in our case)
  - c is mantissa or coefficient (1.237)
  - b is the base (10)
  - q is the exponent (14)

Computers internally use binary base

- $b = 2$
- 64 bits for full representation
  - 52+1 bits for mantissa (about 17 decimal digits)
  - 11 bits for exponent ($\pm$307)

# Limits of the float representation

- maximum $\pm 1.797693134862316 \times 10^{308}$ (use `realmax` in Matlab)
  - $(1.797693134862316 \times 10^{308}) \times 10 =$ `Inf`
  - overflow error
- minimum $\pm 2.225073858507201 \times 10^{-308}$ (use `realmin` in Matlab)
  - $(2.225073858507201 \times 10^{-308})/10 = 0$
  - underflow problem
- truncation error
  - $1.797693134862316 + 20 = 21.797693134862318$
  - $1.797693134862316 + 100 = 101.797693134862\_\_$
- how to mitigate
  - try to use numbers of the similar magnitude
  - do not rely on the least significant digits