KAPLAN HIGHER EDUCATION ACADEMY

KAPLAN HIGHER EDUCATION INSTITUTE

Study Guide

Diploma Programme

Computational Mathematics and Computer Architecture

v2.0

Message to Student

"I encourage you to be an active learner, questioning and extending what you know in order to become a contributing participant in the modern global economy."

Dear Kaplan Student,

Thank you for choosing Kaplan Singapore and entrusting your educational investment with us.

It is Kaplan Singapore's and the School of Diploma Studies' mission as your 'private education institution of choice' to commit to providing a fulfilling student journey and learning experience for your development. The underlying goal of Kaplan Singapore is to ensure that our students are able to respond the demands of the globalised economy set against a highly uncertain and complex environment.

In the 21st Century it is important that your educational experience provides with you with the skills and competencies on HOW to think and not just WHAT to think.

As such, the Kaplan Diplomas, which have a long history of being recognised as equivalent to the first year or more of undergraduate studies in high-quality international universities, aim to prepare you for work, further study, and lifelong learning.

Your role as a Kaplan student is to participate, think deeply on a topic and work with other students and your Lecturers to broaden your knowledge. I encourage you to be an active learner, questioning and extending what you know in order to become a contributing participant in the modern global economy.

However, only you can embody these attributes. You are the major determinant of your own academic success and as adult learners you will receive both the freedom and responsibility this entails. With that, I wish you well in your learning and look forward to seeing you succeed.

Rhys Johnson
Chief Operating Officer and Provost, Kaplan Singapore

Kaplan Desired Graduate Attributes

Through the reading of this module, Kaplan Singapore intends to:

- Instill in students the value of lifelong and self-directed learning by stimulating intellectual curiosity, creative and critical thinking and an awareness of cultural diversity;
- Assist students in developing professional attributes, ethical values, social skills and strategies that will nurture success in both their professional and personal lives;
- Foster integrity, commitment, responsibility and a sense of service to the community;
- Prepare students to meet the ever-changing needs of their communities both now and in the future; and
- Promote innovative and effective teaching.

Culminating from these institutional values and educational goals, Kaplan Singapore's Desired Graduate Attributes are:

Inquiry and criticality: Graduates will be able to critically collect, evaluate and apply information and data in order to make decisions in a wide variety of professional situations. This attribute is demonstrated when students:

- Undertake, evaluate and apply appropriate research, theories, concepts and tools to investigate problems and find solutions;
- Exercise critical thinking and independent judgement to assess situations and determine solutions; and
- Have an informed respect for the principles, methods, values and boundaries of their profession and the capacity to question these.

Ethicality and discernment: Graduates will be able to assess situations and respond in an ethically, socially and professionally responsible manner. This attributed is demonstrated when students:

- Act responsibly, ethically and with integrity in their profession;
- Hold personal values and beliefs and participate in the broad discussion of these values and beliefs while respecting the views of others;
- Understand the broad local and global economic, political, social and environmental systems and their impact as appropriate to their discipline and profession; and
- Acknowledge personal responsibility for their own judgments and behaviour

Ability to communicate well: Graduates will recognise the importance and value of communication in the learning and professional environment. This attributed is demonstrated when students:

- Create and present knowledge, arguments and ideas confidently and effectively using a variety of methods and technologies;
- Recognise the wide range of possible audiences for information and respond with communication strategies appropriate to those audiences; and
- Work collaboratively with people from diverse backgrounds and be aware of the different roles of team members and to function within that team.

Independent and reflective practitioner

- Graduates will be able to work independently and be self-directed learners with the capacity and motivation for continued professional learning and development; and
- They will be able to critically reflect on their own practice and evaluate and understand current capacity and further development needs

Embedded within the desired graduate attributes are the following skills:

- Conduct research.
- Analyse, organise and present data and information.
- Think and read critically.
- Make an oral presentation.
- Intellectual curiosity and awareness of culture and diversity.
- Develop professional ethos and practice that will foster success in career and life.
- Meet the ever changing needs of communities now and in the future.

Table of Contents

About this module

This subject aims to equip students with the fundamentals of problem solving and computer arithmetic. Students will also be introduced the features of MATLAB and how to apply MATLAB to find solutions to mathematics problems. Emphasis is placed on identifying the solutions through IT efficiency.

Module Learning Outcomes

Upon successful completion of this module, the student should be able to:

- Analyse the input, output systems used in computers;
- Analyse the internal architecture of computer systems, including the operating system and other crucial hardware and software;
- Describe number representation in computer systems and its relationship to computer architecture;
- Demonstrate the mathematical skills required for basic problem solving in computers systems;
- Be able to use MATLAB to solve mathematical problems;
- Be able to create and use MATLAB script files and interactive plotting tools.

Overview of Learning Resources

Other sources:

See Proquest and Newslink databases linked to your Elearn LMS homepage. The National Library Board on North Bridge Road (databases are for Singaporean/PR only)

Scheme of Work

LESSON	TOPICS
1	01 Computer Architecture • What is Computer Architecture? • What is a Computer? • Brief History of Computers • The Central Processing Unit • Registers • Buses
2	02 Memory Hierarchy • What is Computer Memory? • Computer Memory Key Characteristics • Computer Memory Goals • Computer Memory Hierarchy • Locality of Reference • Cache Memory Principles • Cache Operation
3	03 Base Conversions I • How does computers interpret numbers • Decimal Numbering System • Binary Numbering System • What is an Octal Number? • What is a Hexadecimal Number?
4	04 Base Conversions II • Adding Binary Numbers • Subtracting Binary Numbers • Multiplying Binary Numbers • Adding Octal Numbers • Adding Hexadecimal Numbers
5	05 Matrix Algebra • What is a Matrix • Equal Matrix • Matrix Addition • Matrix Subtraction • Scalar Product • Matrix Multiplication Revision (Part-time) • Review Topics 1-5 • Practice Questions • Assignment Consultation
6	06 Boolean Algebra • Purpose of Boolean Algebra • Boolean Algebra and basic operations • NOT, AND, OR Gates • NAND, NOR, XOR Gates • Truth Tables • Logic Circuit / Boolean Expression

7	Revision (Full-time) • Review Topics 1-5 • Practice Questions • Assignment Consultation
8	07 Boolean Algebra II • Laws of Boolean Algebra • Associative Laws, Commutative Laws, Distributive Laws, Identity Laws, Idempotent Laws, Complement Laws, Annulment Laws • De Morgan's Theorems 1 and 2 • Simplification of Expressions
9	08 MATLAB Introduction & Environment & Functions • Uses of MATLAB and Structured Problem Solving • Understanding The MATLAB environment • Command Window, Workspace Window & Variables Window, Command History Window,Current Folder Window & Edit Window, Plot Gallery Window • MATLAB Variable, Naming rules and Matrix • What is a Function? • Example of a MATLAB Function • MATLAB Basic Mathematical Functions, Trigonometric Functions, Statistical and Data Analysis Functions • MATLAB Uniform and Gaussian Random Matrices • MATLAB Limits and Pre-defined Values
10	Assignment Draft and Consultation (Full-time) Presentation Draft and Discussion (Full-time)
11	09 MATLAB Plotting • MATLAB Types of Charts • MATLAB 2D Plots • MATLAB Subplots
12	10 MATLAB Programming • Control Structures • Sequence • Selection Statements • Repetition Statements Revision (Part-time) • Review Topics 6-10 • Practice Questions
13	Assignment Presentation and Consultation (Part-time) Revision (Full-time) • Review Topics 6-10 • Practice Questions
14	Assingment Presentation and Consultation (Full-time)

Assessment Matters

Assessment Overview

Assessment 1: CA Quiz

Weightage: 20% (20 marks)
Date: To be confirmeed
Duration: 10 minutes per quiz
Test Format: 5 MCQs per topic
•

Assessment 2: Individual Assignment

Weightage: 40% (100 marks)
Date: Lesson 11 (FT) / Lesson 6 (PT)
Citation Format: APA
References: Module Specific

Important Policies

Penalties for Plagiarism

Plagiarism in any form is not tolerated by Kaplan Singapore. That said, direct quotations and general similarities of common terms and language mean the E-Learn LMS will often pick up every small similarity so the likelihood of a Turnitin Similarity report recording a result of 0% is unrealistic. After all, no technology is perfect and there is the need for some direct quotation (provided you reference using APA guidelines, of course) and to use commonly accepted terms and language.

By way of quality control, Kaplan Singapore has imposed an absolute cut-off of 25% in proprietary programmes (a zero grade for the assessment). In all cases, the lecturer is the true determinant of any result. To be certain you should always reference according to the APA style required by the institution.

TOP TIP:
The surest way to succeed is to ensure all work is correctly referenced. Keep a copy of the Kaplan Singapore Academic Works and APA Guide handy when you are typing your assignments and use it to guide you as to correct referencing, citation and other aspects of academic writing.

APA GUIDE: http://bit.ly/igd-apa

Penalties for late submissions

Kaplan Singapore prepares students for the realities of the workforce and further education by requiring students to meet deadlines and submit all work on time. As such, students are required to seek approval and penalties will be imposed on late assignment submissions in accordance with the table below and cited in the Programme Handbook:

No of days late	Penalty
1 – 5 days	10% deduction per day from the marks attained by students.
After 5 days	Assignments that are submitted more than 5 days after the due date will not be accepted and it will be deemed as "No Submission". Student will be required to re-module.

Source: Kaplan School of Diploma Studies Student Handbook (V4.5, 2017), Section 9

Assignments and Kaplan Learning Management System

Kaplan Singapore School of Diploma Studies requires you to submit Assignments through the Learning Management System (E-Learn LMS). When submitted, your assignment is checked for plagiarism by software called Turnitin linked to the E-Learn LMS. The software is intended to provide one more tool to improve the quality of academic writing and as such will be compulsory for use. It is important to note that this is merely one of many tools available to you and that final decisions about the quality of your work rest with your lecturer.

Assigment Submission: How to Use E-Learn LMS for Assignment Submission

1. You will be enrolled by the School of Diploma Studies Programme Management into the E-Learn LMS system only after your fee payment is confirmed.
2. You will be sent your USER NAME and PASSWORD via email.
3. Reset your password as prompted.
4. Enter the site at the following address: https://elearn-diploma.kaplan.com.sg
5. To submit assignment please refer to the LMS Manual

Please refer to your Student Handbook for more details on Penalties for Plagiarism, Misconduct, Examinations Rules and Regulations. Should you have any queries, please contact diploma.sg@kaplan.com

This page intentionally left blank

Study Guide

Topic 01 – Computer Architecture

Learning Objectives

1. Explain the basics of the Central Processing Unit
2. Explain how data flows between CPU and main memory

Guiding Questions

1. What is a Central Processing Unit?
2. What role does a Central Processing Unit play in a computer system?
3. How does data flow from a CPU to main memory?

What is Computer Architecture?

Computer architecture refers to those attributes of a system visible to a programmer or, put another way, those attributes that have a direct impact on the logical execution of a programme. (Stallings, 2016).

We will look at computer architecture in this course to understand how the components in a computer work with each other, which in turn provides us with the insights on how programmes would work in a computer system.

What is a Computer?

In the simplest sense, a computer is a system that takes in inputs, computes or processes in your inputs and produces an output(s). A computer does the bidding of the user and reads and executes instructions or commands issued to it. The commands can be simple or complex however, it must be in a language that the computer can understand.

In today's context, a computer is your laptop, your desktop and interestingly unknown or taken for granted by many, they are in the hands of many users – the Smart phone, which is a compact computer capable of many chores and tasks.

The rise of technology is a long-term trend. Computers are needed everywhere, in areas of security, artificial intelligence, machine learning, data mining, finance, healthcare, building and more. According to MCG figures, the global data centre industry will achieve strong growth over the next five years, reaching a compound annual growth rate (CAGR) of 10% to 15%. (DataCenterNews, 2020). Hence from this outlook, the use of computers will only grow.

Brief History of Computers

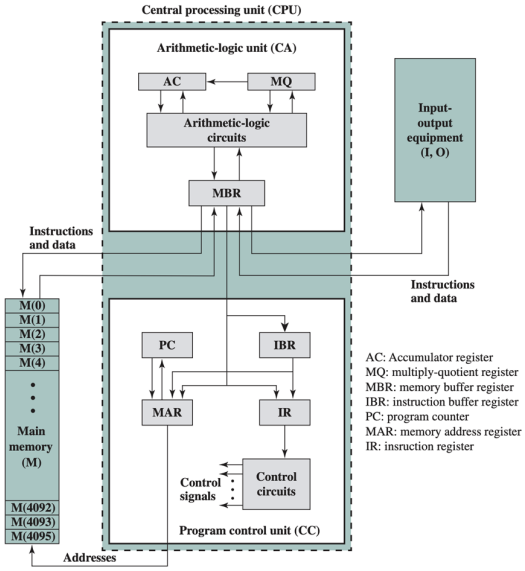
The origins of computers date way back to 1800s where a loom was first created by Joseph Marie in France to automate weaving using punch cards. It was an amazing feat considering that the idea of automation was on the minds of people in that era.
The invention led to another and to another and eventually there were questions on how to store data. This brought us the first generation of computers utilizing 18,000 vacuum tubes

on a 20 by 40-foot room and perhaps an even more well-known computer called the IAS (Institute of Advanced Studies).

The IAS computer featured the **stored memory concept**, conceived by John von Neumann in 1946 but only to be completed in 1952.

An IAS computer contains a **main memory unit**, an **arithmetic and logic unit**, a **control unit,** and an **input-output** equipment. The simplicity of the design can be seen in a conceptual view as seen in the following diagram.

The architecture of IAS computer (also known as the von Neumann machine) became the standard reference for most of today's computers.



IAS Structure (Stallings, 2016) Figure-1a

The Central Processing Unit

From the diagram of IAS computer structure, we can see that the Central Processing Unit (CPU) is critical to the computer system. It is suitably called the brain of the system as it is controlling the operation of the computer and performs its data processing functions. (Stallings, 2016).

It contains components that would process the instructions, hold the data, and send the data back to the main memory. To appreciate how this works in a CPU, we must look further into the different parts of a CPU.

What are the components of a Central Processing Unit?

A CPU would contain several components that are common across general types of CPU and they are:

- Control Unit - Controls the computer's operation
- ALU- Performs data processing functions such as add, divide etc
- Registers- Holds temporary storage in the CPU
- CPU interconnection - To allow communication among the above components

Registers

What are registers and what is their purpose? You can find registers in both the control unit and ALU for the purpose of holding data and addresses. These registers can be categorised as follows:

Memory buffer register (MBR)
-Holds the data to be transferred to or from the main memory or I/O device

Memory address register (MAR)
-Contains the address in the main memory which the data should be stored in or read from into the MBR

Instruction register (IR)
-Stores the current operation or instruction being executed

Instruction buffer register (IR)
-Used to hold temporary right-hand instruction

Programme counter (PC)
-Contains the address of the next instruction to be executed

Accumulator (AC) and multiplier quotient (MQ)
-Stores temporary operands or results of arithmetic operations

Of these, two notable registers are instruction register and the programme counter. The instruction register holds the op-code instruction being executed while the programme counter holds the address of the next instruction pair to be retrieved from the memory.

Communication between CPU and devices – Interconnection structures

So how does the CPU communicate with other devices or components in the computer system such as the main memory?

The answer is using interconnection structures.

A computer consists of a set of components or modules of three basic types (processor, memory, Input and Output devices collectively known as I/O devices) that communicate with each other. In effect, a computer is a network of basic modules. Thus, there must be paths for connecting the modules.

The collection of paths connecting the various modules is called the interconnection structure. The design of this structure will depend on the exchanges that must be made among modules (Stallings, 2016).

To achieve this goal, other considerations must be considered such as allow transfers between memory to CPU, CPU to memory, I/O devices to CPU, CPU to I/O and I/O to memory and memory to I/O.

After several trials, the bus and other bus-like structures become the most popular and commonly used for such communication. The bus interconnection scheme addresses the goal of transferring data between different parts of the system as seen in Figure 1-b

Buses

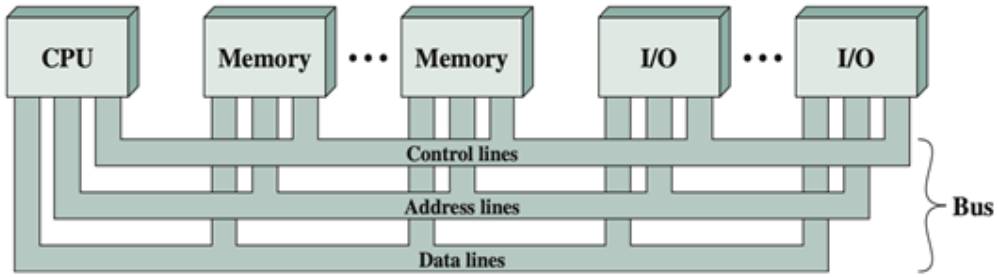
A bus consists of multiple communication lines. A bus that connects major components is known as a system bus with multiple lines.

The lines that form part of the system bus are:

- Data Lines / Data Bus
  Lines that carry data are known as data lines
  Together they are known as the data bus

- Address Lines / Address Bus
  Address lines determine the maximum possible memory capacity of the system
  Together they are known as Address bus

- Control lines
  Control the access to and use of the data and address lines



Bus Interconnection Scheme Figure 1-b

Quick summary

To understand how computers work, the IAS structure must be appreciated. The Central Processing Unit is one of the most important components in a computer. Deeper analysis into the CPU show that the CPU harnesses various components such as the control unit, the ALU, registers and CPU interconnection to process instructions and move data. To facilitate the communication, buses are utilised and become the most popular means to move data.

References

Stallings, W. (2016). Computer organization and architecture: Designing for performance. Hoboken, N.J: Pearson.

Stallings, W. (2018). Operating Systems: Internals and Design Principles. 9th edition: Pearson.

Live Science (2017) History of Computers: A Brief Timeline retrieved from: https://www.livescience.com/20718-computer-history.html

DataCenterNews (2020) Data centre market holds strong amid wider economic outlook retrieved from: https://datacenternews.asia/story/data-centre-market-holds-strong-amid-wider-economic-outlook

Topic 02 – Memory Hierarchy

Learning Objectives

1. List the characteristics of the computer memory systems
2. Describe the use of a memory hierarchy
3. Describe the basic concepts and intent of cache memory

Guiding Questions

1. What is computer memory?
2. What are the key characteristics of computer memory?
3. What is a computer memory hierarchy?
4. What is the locality of a reference?
5. What are the principles of memory cache?
6. What constitutes a cache design?

What is Computer Memory?

Computer memory is a device that stores data or programs on a permanent (non-volatile) or temporary basis. Non-volatile memory allows data to be stored for a longer term even when the power is switched off. Volatile memory, though faster, loses data when there is no power.

Some types of computer memory are designed to be fast whereby the CPU can access data stored in there quickly, while others are designed to be of lower cost so that large amount of data can be stored more cost effectively.

What is Computer Memory?

A computer memory system can be categorised by the following key characteristics.

Location

Location refers to the position of the memory device in a computer system, whether it is internal or external. Internal (or main) memory such as Read Access Memory (RAM), Read Only Memory (ROM) or cache can directly be accessed by the CPU. External (or secondary) memory comprises of peripheral storage devices such as hard disks, solid state drives (SSDs) and magnetic tapes, and accessible to the CPU via device controllers.

Capacity

Capacity for internal memory is typically expressed in terms of bytes (1 byte = 8 bits) or words. Commonly used word sizes are 1 byte (8 bits), 2 bytes (16 bits) and 4 bytes (32 bits). External memory capacity is usually expressed in terms of bytes (e.g. 2TB, 500GB).

Unit of Transfer

The unit of transfer is equivalent to the number of bits that can be read out of or written into memory at a time. For internal memory this is usually equates to a word size. The size of a word generally refers to the number of bits used to represent an integer.
In the case of external memory, unit of transfer is not limited to the word size. It is often larger and is referred to as blocks.

Method of Access

Another distinct characteristic is the method of access which is based on the hardware implementation of a storage device, namely:

- Sequential Access
  Memory is accessed in a specific linear sequence. The time to access a memory record may vary as it depends on the location of the data and its previous location. Some media examples whereby memory is accessed in a sequential manner are magnetic tapes and CD-ROMs.

- Random Access
  In this method, any location of the memory can be accessed randomly. Hence the time to access a given location is independent of the sequence of prior accesses and is constant. Some applications of such random memory access are RAM and ROM.

- Direct Access
  A particular location of the memory can be accessed directly. This method is a combination of the sequential and random access methods, whereby access is conducted in the manner of jumping to some vicinity and then performing a sequential search. As a result, access time is variable. Disk units are examples of direct access.

- Associative Access
  A random-access type of memory. In this memory, a word is accessed based on a portion of its contents rather than its address. Retrieval time is constant independent of location or prior access patterns. An application of this memory access is cache memory.

Performance

It is a vital characteristic of the memory system and matters most to users. The following parameters need to be considered for performance:

- Transfer Rate - the rate at which data is transmitted into or out of a memory unit. It is different for different types of memory.

- Memory Cycle Time – the time required for accessing a block as well as the period prior to the start of the second access. It is mostly required for random access memory and related with the system bus but not the processor.

- Access Time - this is different for different types of memory. In random access memory, it is the time taken to perform a read or write operation. In non-random access memory, it is the time required to place the read-write mechanism at the desired location.

Physical Type

A variety of physical types of memory are available, the most common ones are semiconductor (e.g. RAM) and magnetic surface (e.g. hard disks) memory.

Physical Characteristics

The nature of memory which determines if the memory is volatile, non-volatile, erasable or non-erasable.

- Volatile – persistence of data is not ensured when the power is switched off
- Non-volatile - a memory device continues to hold data even if the power is switched off
- Erasable – allows one to modify data in memory e.g. RAM
- Non-Erasable – data in memory cannot be changed, except by destroying the storage unit e.g. ROM

Organization

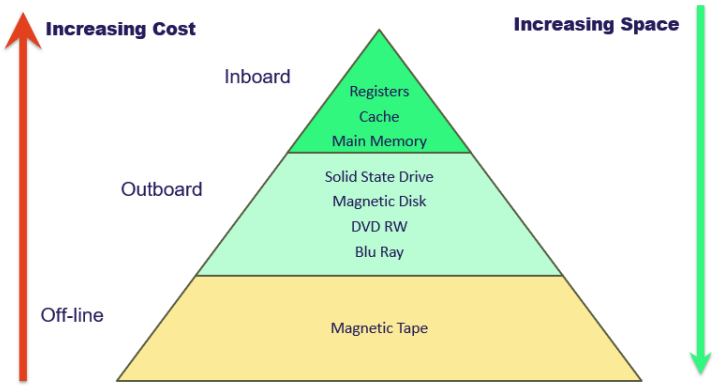
This is a physical arrangement of bits to form words. This is a vital design consideration for random access memory.

Computer Memory Hierarchy

The goals of computer memory are to provide adequate storage to hold data, provision a memory system which is affordable and yet able to achieve an optimal performance. More often than not, these goals come with a trade-off among three key characteristics of memory - namely access time, capacity and cost.

- Decrease in access time → increase in cost per bit
- Increase in capacity → decrease in cost per bit
- Increase in capacity → decrease in access time

A memory hierarchy aims to address these consideration, as they affect the system performance in computer architectural design. Designing for high performance needs to factor in the constraints of the memory hierarchy - size and capabilities of each characteristics. One way to improve system performance is to reduce how far down the memory hierarchy one has to go to access data and also to minimise the frequency of accessing data. To help decrease frequency of access, we would need to understand the **Principle of Locality**.



Memory Hierarchy Figure-2a

Locality of Reference

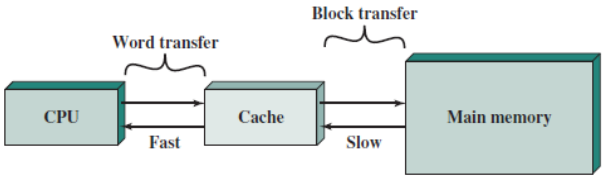
Locality of reference - also known as the Principle of Locality - refers to the tendency of a computer processor to access the same set of memory locations repeatedly over a short period of time. From observations, computer programs tend to reuse data and instructions used recently. 90% of execution happens in 10% of code. We could predict the instructions and data based on past data

Systems that display strong locality of reference are good candidates for performance optimization through the use of techniques such as caching.

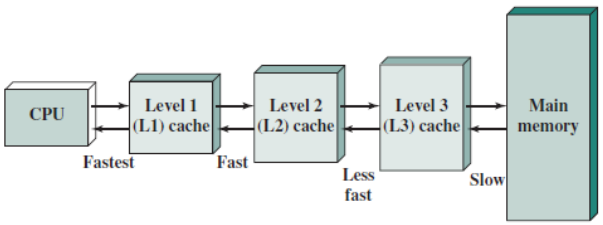
Principles of Cache Memory

Cache is a small volume of memory which is a part of the CPU. It is used to store instructions and data that the CPU is likely to reuse on a short term basis. However cache in a CPU is too small to hold much data, while a the capacity of a main memory board is much larger but too slow in speed. To leverage on the advantages of both types of memory, a plausible solution is to place a faster memory type in between the CPU and main memory. This memory is known as Cache Memory.

Cache memory is designed to leverage on a combination of the memory access time of expensive and high-speed memory, and the large memory size of less expensive and lower-speed memory.



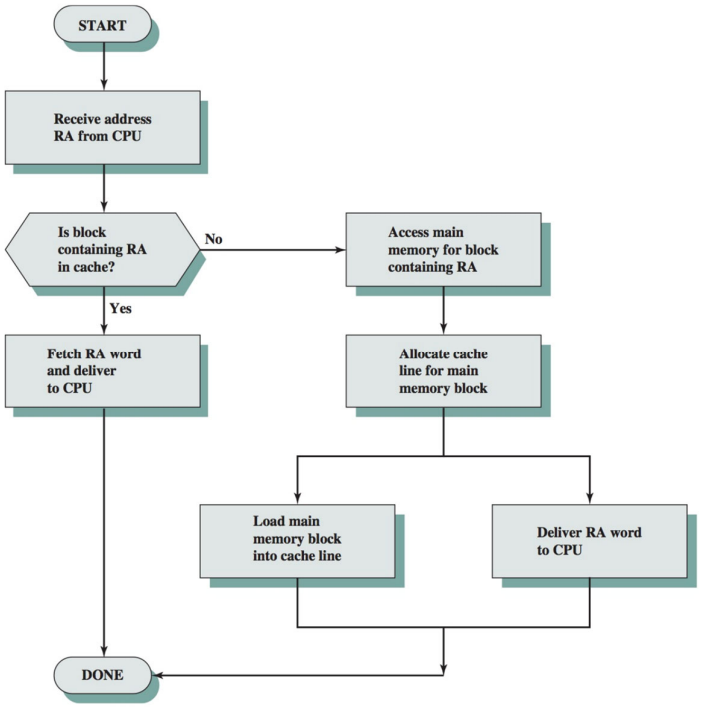
Single Cache Figure-2b,
IAS Structure (Stallings, 2016)



Three-level Cache Figure-2c,
IAS Structure (Stallings, 2016)

Cache Read Operation

In a typical cache operation, the CPU requests for a content in memory location via a Read Address (RA). If the content is found in the cache it will be sent to the CPU, else the required content block from the main memory will be loaded into the cache and sent to the CPU.



Cache Read Operation Figure-2d,
IAS Structure (Stallings, 2016)

Elements of Cache Design

- Cache Addresses
  Virtual memory, which is a means to enable computer programs to address memory from a logical point of view and disregarding the amount of physical main memory available, is supported by most processors. When virtual memory is used, the address fields of machine instructions contain virtual addresses. A memory management unit (MMU) converts virtual addresses of reads and writes instructions into physical addresses in the main memory.

- Cache Size
  The size of the cache should ideally be small enough so that the overall average cost per bit is almost the same as that of the main memory, and yet big enough so that the overall average access time is similar to that of the cache.

- Mapping Function
  As there are less cache lines than main memory blocks, an algorithm is required to map the main memory blocks into cache lines. In addition, a method is necessary to ascertain which main memory block currently occupies a cache line. The mapping function adopted dictates how the cache is organized. Three approaches which are available are:

  i. Direct Mapping – maps each block of main memory into only one possible cache line
  ii. Associative Mapping – enables each main memory block to be loaded into any line of cache
  iii. Set-Associative Mapping – leverages on the strengths of direct and associative mappings by enabling blocks to be mapped into any of the lines of set.

- Line Size
  Cache lines are the chunks of memory managed by the cache. The size of the memory chunks is the cache line size. A cache can only contain a limited number of lines determined by the cache size, and typical cache line sizes are 32, 64 and 128 bytes. If line size is too small, hit ratio will be low. However if line size is too big, data is overwritten very soon after it is fetched.

  A hit ratio is a calculation of cache hits against the total content requests received. A high cache hit ratio of 90% and above means most of the requests are satisfied by the cache, or simply put it the caching is efficient.

- Replacement Algorithms
  A cache that is full eventually will require a new block of data to be brought in. How the data is replaced can be based on one of the following algorithms:

  - First In First Out (FIFO) - Replace the block in the set that has been in the cache longest. FIFO is easily implemented as a round-robin way.

FIFO

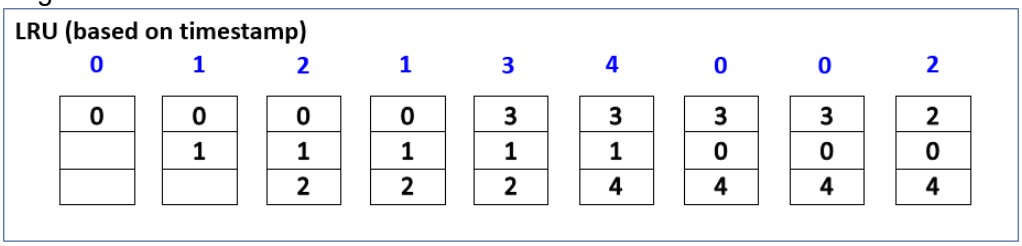
0	1	2	1	3	4	0	0	2
0	0	0	0	3	3	3	3	2
	1	1	1	1	4	4	4	4
		2	2	2	2	0	0	0

  - Least Frequently Used (LFU) - Replace the block in the set that has encountered the fewest references. LFU could be implemented by correlating a counter with each line.

LFU (based on counter)

0	1	2	1	3	4	0	0	2
0	0	0	0	3	3	0	0	0
	1	1	1	1	1	1	1	1
		2	2	2	4	4	4	2

- Least Recently Used (LRU) – Replace the block in the set that has been in the cache longest with no reference to it.



- Write Policy
  When a system writes data to cache, it must at some point write that data to the main memory. The timing of this write is controlled by a write policy which adopts 2 types of writing methods:

  - Write-through – data is written to both the cache and main memory synchronously. This technique is more reliable as the data in the main memory is always valid. However it is slower as substantial memory traffic is generated.
  - Write-back – data is only written to the cache initially. The write to the main memory is postponed until the modified content is about to be replaced by another cache block. This method is faster but less reliable as portions of main memory are invalid.

- Number of Caches
  The use of multiple caches is common among systems in current times. Two considerations to be factored in when designing cache are the number of levels of caches and the use of unified versus split caches.

  - Multilevel cache - more effective than a single level cache, e.g. L1 – internal cache and L2 – external cache.
  - Unified cache vs split cache – a unified cache holds both instructions and data in the same cache and has a higher hit rate. A split cache consists of two physically separate parts - one is dedicated to holding instructions, the other is dedicated to holding data. A split cache is able to scale better.

Quick summary

Computer memory displays a wide range of type, technology, organization, performance, and cost of any feature of a computer system. No one technology is ideal in meeting the memory requirements for a computer system. Hence the typical computer system consists of a hierarchy of memory subsystems. To appreciate how memory cache is designed, the various elements involved such as cache address, cache size and mapping function would need to be considered.

References

Stallings, W. (2016). Computer organization and architecture: Designing for performance. Hoboken, N.J: Pearson.

Stallings, W. (2018). Operating Systems: Internals and Design Principles. 9th edition: Pearson.

Topic 03 – Base Conversions I

Learning Objectives

1. Explain how to convert from binary to decimal and vice versa.
2. Explain how to convert from hexadecimal to decimal and vice versa.

Guiding Questions

1. What is the Decimal Number System?
2. What is a Bit?
3. What are the differences between Binary Number System, Hexadecimal Number System and Octal Number System?
4. How to convert an Integer from Base b to Decimal?
5. How to convert a Decimal Integer into Base b?

The Decimal Number System

The Decimal Number System is used for counting in our daily lives. We use digits 0 to 9 to represent numbers. Consider the number 68.

The number 68 is made up of 6 tens and 8 ones or equivalent to 6 x 10 + 8 x 1.
68
$= 6 \times 10 + 8$
$= 6 \times 10^1 + 8 \times 10^0$

Let us consider another number 5678. We can rewrite 5678 as
5678
$= 5 \times 1000 + 6 \times 100 + 7 \times 10 + 8$
$= 5 \times 10^3 + 6 \times 10^2 + 7 \times 10^1 + 8 \times 10^0$

Explanation: The decimal number system uses 10 as a base. The position of the number is equivalent to the index of the base. The position starts from **0** right the right hand side and increases by 1 for every shift to the left hand side.

Hence the far left most digit which has the biggest value is known as the **most significant digit**. The far right most digit is known as the **least significant digit**.

Let's take a look at 228095. In a table, it can be placed as:

	2	2	8	0	9	5
Position	5	4	3	2	1	0
Value of base 10	10^5	10^4	10^3	10^2	10^1	10^0

What is a Bit?

A Bit is also known as a **B**inary dig**it**. In a Binary system, instead of 10 numbers like the Decimal Numbering system, there are only 2 numbers, 0 and 1.
A Binary number is denoted with the subscript 2. The principle of the numbering system is the same as the Decimal Numbering system and follows the position in the digit.

Let us consider the number 1001_2
1001_2
$= 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$
$= 1 \times 8 + 1$
$= 9$

We can see the numbering system is very similar to that of the Decimal Numbering System. The main difference is the use of Base 2 instead of Base 10.
Let us consider another example $1110\ 0101_2$ with a table.

	1	1	1	0	0	1	0	1
Position	7	6	5	4	3	2	1	0
Value of Base 2	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

$1110\ 0101_2$
$= 1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$
$= 128 + 64 + 32 + 0 + 0 + 4 + 0 + 1$
$= 229$

Converting from Decimal to Binary Number

In the previous section, we saw how to convert from a Binary number to a Decimal number. In this section, we see how to convert from a Decimal number to a Binary number.
The steps to convert from a Decimal number to a Binary number can be detailed as follows:

1. Divide the decimal number by 2.
2. Use the integer quotient for the next repetition.
3. List down all the remainders for each division in order.
4. Repeat the steps until the quotient is equal to 0.
5. Reverse the remainders and you will have the binary number equivalent.

Let us see a simple example to perform the conversion.

Let us use number 39 for Binary conversion.
39 ÷ 2 = 19 remainder **1**
19 ÷ 2 = 9 remainder **1**
9 ÷ 2 = 4 remainder **1**
4 ÷ 2 = 2 remainder **0**
2 ÷ 2 = 1 remainder **0**
1 ÷ 2 = 0 remainder **1**

Once the quotient reaches 0, we can stop the conversion and look at the remainder from bottom up. The remainders are 1, 0, 0, 1, 1, 1. Hence the binary representation is $0010\ 0111_2$. We can verify the result by converting the binary representation back to decimal number. This will show that number is indeed 39.

Let us look at another example using the number 237.
237 ÷ 2 = 118 remainder **1**
118 ÷ 2 = 59 remainder **0**
59 ÷ 2 = 29 remainder **1**
29 ÷ 2 = 14 remainder **1**

14 ÷ 2 = 7 remainder **0**
7 ÷ 2 = 3 remainder **1**
3 ÷ 2 = 1 remainder **1**
1 ÷ 2 = 0 remainder **1**

Hence the binary representation is $1110\ 1101_2$.

What is an Octal Number?

An Octal number as the name implies is a numbering system with 8 numbers. Hence it has Base 8 instead of Base 10 and the numbers go from 0 to 7. An Octal number can be converted into a Decimal number following the same conversion that a Binary undergo to become a Decimal number.

Let us consider the Octal number 671_8.

	6	7	1
Position	2	1	0
Value of Base 8	8^2	8^1	8^0

671_8
$= 6 \times 8^2 + 7 \times 8^1 + 1 \times 8^0$
$= 6 \times 64 + 7 \times 8 + 1 \times 1$
$= 441$

The Octal Number can be converted to Binary Numbers easily as an Octal digit can be represented by 3 bits.

We can see from the table below how to convert an Octal Digit to 3 bits.

Octal Number	**Equivalent Binary Digit**
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Hence the Octal number 271_8 can be simply mapped to $010\ 111\ 001_2$ and then rewritten as $1011\ 1001_2$.

Converting from Decimal Number to Octal Number

The steps to convert from a Decimal number to an Octal number is very similar to the steps needed to convert from a Decimal number to a Binary number:

1. Divide the decimal number by 8.
2. Use the integer quotient for the next repetition.
3. List down all the remainders for each division in order.
4. Repeat the steps until the quotient is equal to 0.
5. Reverse the remainders and you will have the Octal number equivalent.

Let us look at an example.

Let us look at another example using the number 629.

629 ÷ 8 = 78 remainder **5**
78 ÷ 8 = 9 remainder **6**
9 ÷ 8 = 1 remainder **1**
1 ÷ 8 = 0 remainder **1**

Hence the Octal number representation is 1165_8.

What is a Hexadecimal Number?

A hexadecimal numbering system has 16 numbers and starts from 0 and ends with F (F being 15). The use of hexadecimal numbers is convenient and compact for humans compared to the use of binary numbers which is long and cumbersome. The notation can be seen in the table below for the binary equivalent (4 bits long).

Hexadecimal Number	Decimal Number	Binary Digit Equivalent
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Now let us consider the hexadecimal number ACE_{16} or 0xACE.

	A	C	E
Position	2	1	0
Value of Base 8	16^2	16^1	16^0

ACE_{16}
$= 10 \times 16^2 + 12 \times 16^1 + 14 \times 16^0$
$= 10 \times 256 + 7 \times 16 + 14 \times 1$
$= 2560 + 102 + 14$
$= 2676$

Let us look at another example the hexadecimal number $BEE5_{16}$ or 0xBEE5.

	B	E	E	5
Position	3	2	1	0
Value of Base 8	16^3	16^2	16^1	16^0

$BEE5_{16}$
$= 11 \times 16^3 + 14 \times 16^2 + 14 \times 16^1 + 5 \times 16^0$
$= 11 \times 4096 + 14 \times 256 + 14 \times 16 + 5 \times 1$
$= 45056 + 3584 + 224 + 5$
$= 48869$

To convert a Hexadecimal to a Binary number is relatively easy compared to Octal number. Each hexadecimal equates to 4 binary digits. Since 8 bits is a byte, hence 2 hexadecimals represent a byte.

Let us look at an example on how to convert a hexadecimal to the binary format. Since 1 hexadecimal is equivalent to 4 bits, 0xBEE5 can be mapped to $1011\ 1110\ 1110\ 0101_2$.

Converting from Decimal Number to Hexadecimal Number

Like converting Decimal numbers to Binary format and Decimal numbers to Octal format, the steps to convert from a Decimal number to an Hexadecimal number is very similar:

1. Divide the decimal number by 16.
2. Use the integer quotient for the next repetition.
3. List down all the remainders for each division in order.
4. Repeat the steps until the quotient is equal to 0.
5. Reverse the remainders and you will have the Hexadecimal number equivalent.

Let us look at an example using the number 731.

731 ÷ 16 = 45 remainder 11
45 ÷ 16 = 2 remainder **13**
2 ÷ 16 = 0 remainder **2**
As 13 is equivalent to D and 11 is equivalent to B, therefore the Hexadecimal number representation is $2DB_{16}$.

Let us look at another example using the number 3417.

3417 ÷ 16 = 213 remainder 9
213 ÷ 16 = 13 remainder **5**
13 ÷ 16 = 0 remainder **13**
As 13 is equivalent to D and, therefore the Hexadecimal number representation is $D59_{16}$.

Summary

The natural counting system differs from the binary representation in computers. Hence there is a need to convert from decimal number system to binary system for computers to process the numbers.However the binary numbering system is cumbersome and difficult for humans to remember or process. Hence the Octal numbering system and Hexadecimal number system are introduced. It is natural that eventually the Hexadecimal becomes the choice of computer system developers as it is easier to convert Hexadecimal numbers to binary representation.

References

Stallings, W. (2016). Computer organization and architecture: Designing for performance. Hoboken, N.J: Pearson.

Topic 04– Base Conversions II

Topic 04 – Learning Objectives

1. Explain how to add binary numbers.
2. Explain how to add hexadecimal numbers.

Guiding Questions

1. How to add and minus Binary numbers?
2. How to add and minus Octal numbers?
3. How to add and minus Hexadecimal numbers?
4. How to perform Binary subtraction?
5. How to multiply Binary numbers?
6. How to do Binary division?

Addition of Decimal Numbers

Let us take a look at addition of decimal numbers in the Decimal Number System. The base of the numbers is 10. Hence any addition of numbers above 9 will institute a carry over.
In the following example, we see how to add 2 decimal numbers. After that we will apply the same principle to other number formats.

```
    1 1
    478
+   149
------------
     627
```

Column 1 from the right: The carry over happens when 8 and 9 adds up to 17. This gives a carry over to column 2 from the right.

Column 2 from the right: 1 + 7 + 4 = 12 and this gives another carry over of 1.
Column 3 from the right: 1 + 4 + 1 = 6

The addition of decimal numbers sets a foundation that can be applied to other number formats as well. As long as the addition exceeds the number base – 1, there will be a need to carry over to the next position.

Binary Addition

Similar to Decimal numbers, Binary numbers can be added except that the Base is 2, not Base 10. This makes adding easy since all you need to worry about is that if the number exceeds 1, it does a carry over. There can only be 0 and 1 in the system.

In our examples we will not be considering any negative numbers. Instead we will just focus on positive numbers in this chapter for simplicity. The numbers will be listed in multiple of 4s or 8s or 16s.

Let us consider 2 binary numbers 0011_2 and 0101_2

$$
\begin{array}{r}
1 \\
0011_2 \\
+\ 0101_2 \\
\hline
0_2 \\
\hline
\end{array}
$$

Since 1 and 1 adds up to 2, the result exceeds 1 and there needs to be a carry over. The carry over is indicated at the top of the second column from the right.
After the carry over, the addition carries on as usual.

$$
\begin{array}{r}
11 \\
0011_2 \\
+\ 0101_2 \\
\hline
00_2 \\
\hline
\end{array}
$$

The carry-over is 1 in the second column from the right and adds to 1 and 0. The result is now 2 and this triggers another carry over with the remainder as 0. Hence there is another carry over indicated on top of the 3rd column from the right.

$$
\begin{array}{r}
111 \\
0011_2 \\
+\ 0101_2 \\
\hline
000_2 \\
\hline
\end{array}
$$

The carry over adds to 0 and 1 in column 3 from the right like column 2 (from the right). This adds to 2 again and trigger a final carry over to the 4th column from the right.

$$
\begin{array}{r}
111 \\
0011_2 \\
+\ 0101_2 \\
\hline
1000_2 \\
\hline
\end{array}
$$

Finally, the 4th column from the right adds 1 from carry over and 0 and 0 from the other 2 numbers. The final result is 1000_2.
You can verify by adding the decimal number representation of the binary numbers and check against the result.

Let's look at another example numbers $0011\ 0101_2$ and $0100\ 1101_2$ without the intermediate steps.

$$
\begin{array}{r}
111\ 1\ \ 1 \\
0011\ 0101_2 \\
+\ 0100\ 1101_2 \\
\hline
1000\ 0010_2 \\
\hline
\end{array}
$$

Double Checking our answers:
For verification, $0011\ 0101_2$ is equivalent to decimal number 53 and $0100\ 1101_2$ is equivalent to decimal number 77. The total is 53 + 77 = 130 which is equivalent to $1000\ 0010_2$.

Octal Number Addition

Like Decimal numbers and Binary numbers, addition is done the same way except now the Base is 8. Again we will not consider any negative numbers. In Octal numbers, the carry-over is done whenever the result for one column exceeds 7.

Let us consider 2 Octal numbers 073_8 and 764_8

$$
\begin{array}{rr}
 & 073_8 \\
+ & 764_8 \\
\hline
 & 7_8 \\
\hline
\end{array}
$$

Since 3 and 4 adds up to 7, there is no carry-over. The addition for the second column carries as usual.

$$
\begin{array}{rr}
 & 1 \\
 & 073_8 \\
+ & 764_8 \\
\hline
 & 57_8 \\
\hline
\end{array}
$$

For the second column from the right, the total is 13 which means a carry over of 1 (13 – 8 = 5) and a “remainder” of 5.

$$
\begin{array}{rr}
 & 11 \\
 & 073_8 \\
+ & 764_8 \\
\hline
 & 057_8 \\
\hline
\end{array}
$$

The addition of the carry of 1 and 7 in column 3 from the right will give a value of 8. Anything that exceeds 7 will generate a carry over.

$$
\begin{array}{rr}
 & 11 \\
 & 073_8 \\
+ & 764_8 \\
\hline
 & 1057_8 \\
\hline
\end{array}
$$

The carry-over adds a digit to the front as there is an overflow. Hence it is just a 1 to complete the addition.

We shall see another quick example to reinforce our understanding on Octal addition.

$$\begin{array}{r} \ 111 \\ \ 6543_8 \\ +\ 2345_8 \\ \hline \ 11110_8 \\ \hline \end{array}$$

In this example, there are multiple carry-overs. The result is an Octal number which looks like a Binary number. Verify your results by converting the 2 Octal numbers to Decimal numbers and then add to get the Decimal result. Compare that result with this Octal result and you will find the results to be the same.

Hexadecimal Addition

Hexadecimal Addition is the last stop before we focus on the other operations on Binary numbers. As the principle is mostly the same, we will start with examples. The carry-over happens when the addition is more than 15.

Let us consider 2 Hexadecimal numbers ACE_{16} and 321_{16}

$$\begin{array}{r} \ ACE_{16} \\ +\ 321_{16} \\ \hline \ DEF_{16} \\ \hline \end{array}$$

The example is a simple one, all the numbers add to a maximum of 15. Hence there are no carry-overs. For example, E + 1 gives 15 which is F. C + 2 is 14 which is E and A + 3 is 13 which gives D.

Let us look at an example with carry-overs.

$$\begin{array}{r} \ 1 \\ \ BEAD_{16} \\ +\ 1234_{16} \\ \hline \ 1_{16} \\ \hline \end{array}$$

Column 1 from the right: D + 4 = 13 + 4 = 17, hence there is a carry-over of 1 with remainder of 1

$$\begin{array}{r} \ 1 \\ \ BEAD_{16} \\ +\ 1234_{16} \\ \hline \ E1_{16} \\ \hline \end{array}$$

Column 2 from the right: Carry-over of 1 + A + 3 = 1 + 10 + 3 = 14 = E. There is no carry-over in this case.

```
      1   1
    BEAD16
+   1234 16
-------------
     0E1 16
-------------
```

Column 3 from the right: E + 2 = 14 + 2 = 16. There will be a carry-over of 1 with 0 remainder.

```
      1   1
    BEAD16
+   1234 16
-------------
    D0E1 16
-------------
```

Column 4 from the right: Carry-over of 1 + B + 1 = 1 + 11 + 1 = 13. This equates to D with no carry-over.

Binary Subtraction

For the remaining chapter, we will look at operations on binary numbers such as subtraction, division and multiplication. The reader can apply the same principle for Octal as well as Hexadecimal numbers. Again for our demonstration, only positive results are considered in the examples.

```
    10112
-   01012
-------------
       102
-------------
```

The last 2 digits are simple subtraction of $11_2 - 01_2$ which yields 10_2

```
      1
    ~~1~~011 2
-   0101 2
-------------
      1102
-------------
```

As there is insufficient amount to be deducted in the 3rd column (from right), there is a need to borrow from the 4th column (from right). Hence the subtraction is $10_2 - 01_2$ which yields 01_2
As there is nothing else to deduct in the 4th column (from right), the result is 0110_2

Binary Division

Binary Division is the same as normal division using the long division method. For binary division, there could be remainder as well.

Consider $1011_2 \div 11_2$

Using long division:

```
          11
11 ) 1001
        - 11
       ------
         011
         - 11
        ------
             0
        ------
```

The remainder is 0 in the case. We can verify that 1001_2 is 9 and 9 divided by 3 (11_2) is 3 (11_2)

Consider another example: $1100\ 1000_2 \div 101_2$

Using long division:

```
            10 1000
101 ) 1100 1000
        - 101
       ------
         00101
         -   101
        -----------
             0000
        -----------
```

Again, the remainder is 0 in the case. We can also verify that $1100\ 1000_2$ is 200 and 200 divided by 5 (101_2) is 40 ($10\ 1000_2$)

Summary

Binary numbers like decimal numbers can work with operators such as addition, subtraction, multiplication or division. To add, subtract, multiply and divide binary numbers, one has to take care as the numbers is of Base 2.

Addition can also be applied to octal numbers and hexadecimal numbers.

References

Stallings, W. (2016). Computer organization and architecture: Designing for performance. Hoboken, N.J: Pearson.

Topic 05 – Matrix Algebra

Learning Objectives

1. Explain a Matrix

Guiding Questions

1. What is a vector?
2. What is a matrix?
3. How is a matrix described?
4. How is scalar product performed?
5. What is transposition / transformation of matrices?
6. How is matrix addition and subtraction performed?
7. How is matrix multiplication performed?
8. What are identity matrices?

What is a Vector?

In mathematics, a vector is an object that has both magnitude and a direction. The history of vectors were never truly pinned down and may have been attributed to as far back as 384B.C in the lost work of Aristotle or perhaps seen in vectorial representations in Newton's laws which include velocities and force. (Both velocities require magnitude and direction).

Yet today, in computing, vectors proved to be extremely useful in representing various collections of data. Vectors can be seen as holding a collection of numbers, a collection of Strings, a collection of objects and unlimited possibilities.

For integer n >= 0, a n-vector is a set or array of n numbers known as elements or scalars. The number of elements in a vector is also known as the length of the vector or size of the vector. Hence a vector of size n can be written as such:

$v = \{ x_1, x_2, x_3 \dots x_n \}$

Explanation: The vector is v in this case where the elements are x_1 to x_n where the x_1 is the first element (also known as the 0^{th} element in computing) and x_n is the last element in the vector. For the purpose of explanation in the guide, we will use 1^{st} element as the standard mathematical notation.

What is a Matrix?

A matrix is a two-dimensional array or a rectangular array of numbers. In another way, you could think of a matrix as a collection of vectors.

A matrix has multiple rows and multiple columns. Hence a matrix with m rows and n columns is known as an m x n matrix.

The rows and columns can be thought of as vectors, therefore you can think of a matrix as "m" number of vectors of "n" vectors.

How do you describe a Matrix?

A matrix would be usually represented by a name in upper case letter. Therefore, a matrix could be simply named as A, B or C etc. The elements in the matrix would be simply the name of the matrix in lowercase followed by subscript to indicate the row and column the element's position in the matrix.

We will use the notation a_{ij} to represent an element where "i" indicates the row and "j" refers the column of the matrix.

The m x n matrix can now be written in the following manner:

$$X = \begin{bmatrix} x_{11} & \cdots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{mn} \end{bmatrix}$$

From the above, we can read x_{11} as element in the first row, first column. The x_{mn} element is in the m^{th} row, n^{th} column.

The matrix can also be written as:

$X = (x_{ij})$

Let us run through some examples of matrices:

$$A = \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 2 & 4 \\ 1 & 3 & 5 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

D = [1 2 3 4 5]

From the above matrices,
A is a 2x2 matrix (2 rows and 2 columns). Element 5 is in row 2, column 2.
B is a 2x3 matrix (2 rows and 3 columns). Element 5 is in row 2, column 3.
C is a 3x1 matrix (3 rows and 1 column). Element 2 is in row 2, column 1.
D is a 1x5 matrix (1 row and 5 columns). Element 3 is in row 1, column 3.
Both matrices C and D are 1 dimensional matrices and hence are also vectors.
We can see that matrices and be represented in any number of rows and columns.

What is a Scalar Product of a Matrix?

A scalar product of the matrix is the product of a real number and a matrix. Each matrix element will be multiplied by the real number and the result will be stored as the element in the same position.

In the simplest sense, the scalar product of real number k and matrix A will be given as:
k * A = kA

and can be written as:

$$k * A = k\begin{bmatrix} x_{11} & \cdots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{mn} \end{bmatrix} = \begin{bmatrix} kx_{11} & \cdots & kx_{1n} \\ \vdots & \ddots & \vdots \\ kx_{m1} & \cdots & kx_{mn} \end{bmatrix}$$

An example of a scalar product of a matrix A is given below:

$$A = \begin{bmatrix} 0 & 2 & 4 \\ 1 & 3 & 5 \end{bmatrix}$$

Then 5A is given as

$$5A = 5\begin{bmatrix} 0 & 2 & 4 \\ 1 & 3 & 5 \end{bmatrix} = \begin{bmatrix} 0 & 10 & 20 \\ 5 & 15 & 25 \end{bmatrix}$$

Transposing a Matrix (Transformation)

The act of transposing a matrix is in layman terms flipping a matrix over its diagonal. In a formal manner, the i^{th} row would now become the i^{th} column of the matrix and the j^{th} column is now the j^{th} row in the new matrix. The transposed matrix A will simply be denoted by A'.

The transposition process is also known as transformation of matrix A as it manipulates and changes the rows to columns and columns to rows.

Let the original matrix be X as follows:

$$X = \begin{bmatrix} x_{11} & \cdots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{mn} \end{bmatrix}$$

After transposing,

$$X' = \begin{bmatrix} x_{11} & \cdots & x_{m1} \\ \vdots & \ddots & \vdots \\ x_{1n} & \cdots & x_{mn} \end{bmatrix}$$

Let us look at an example of how to transpose A to A':

$$A = \begin{bmatrix} 0 & 2 & 4 \\ 1 & 3 & 5 \end{bmatrix}$$

$$A' = \begin{bmatrix} 0 & 1 \\ 2 & 3 \\ 4 & 5 \end{bmatrix}$$

A double transposition of A will return the matrix back to its original form.

$$A'' = \begin{bmatrix} 0 & 2 & 4 \\ 1 & 3 & 5 \end{bmatrix} = A$$

Matrix Addition

Two matrices can be added if their dimensions are the same. The addition is simply denoted by the addition sign. To add up the matrices, the corresponding elements in each matrix are added and the result is stored in the same position in the final matrix.

Let X and Y be matrices of the same dimensions (m x n):

$$X = \begin{bmatrix} x_{11} & \cdots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{mn} \end{bmatrix}$$

$$Y = \begin{bmatrix} y_{11} & \cdots & y_{1n} \\ \vdots & \ddots & \vdots \\ y_{m1} & \cdots & y_{mn} \end{bmatrix}$$

Hence X + Y will be given as follows:

$$X + Y = \begin{bmatrix} x_{11} + y_{11} & \cdots & x_{1n} + y_{1n} \\ \vdots & \ddots & \vdots \\ x_{m1} + y_{m1} & \cdots & x_{mn} + y_{mn} \end{bmatrix}$$

Let us look at an example of matrix addition:

$$A = \begin{bmatrix} 0 & 1 & 3 \\ 1 & -1 & 5 \end{bmatrix}$$

$$B = \begin{bmatrix} 7 & 2 & 4 \\ 1 & 3 & 5 \end{bmatrix}$$

$$A + B = \begin{bmatrix} 0+7 & 1+2 & 3+4 \\ 1+1 & 3-1 & 5+5 \end{bmatrix} = \begin{bmatrix} 7 & 3 & 7 \\ 2 & 2 & 10 \end{bmatrix}$$

Matrix Subtraction

Similarly, the difference of two matrices can be performed if their dimensions are the same. The addition is denoted by the subtraction sign. To find the difference of the two matrices, subtraction of the corresponding elements in each matrix are performed and the result is stored in the same position in the final matrix.

Let X and Y be matrices of the same dimensions (m x n):

$$X = \begin{bmatrix} x_{11} & \cdots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{mn} \end{bmatrix}$$

$$Y = \begin{bmatrix} y_{11} & \cdots & y_{1n} \\ \vdots & \ddots & \vdots \\ y_{m1} & \cdots & y_{mn} \end{bmatrix}$$

Hence X - Y will be given as follows:

$$X - Y = \begin{bmatrix} x_{11} - y_{11} & \cdots & x_{1n} - y_{1n} \\ \vdots & \ddots & \vdots \\ x_{m1} - y_{m1} & \cdots & x_{mn} - y_{mn} \end{bmatrix}$$

Let us look at an example of matrix subtraction:

$$A = \begin{bmatrix} 0 & 1 & 3 \\ 1 & -1 & 5 \end{bmatrix}$$
$$B = \begin{bmatrix} 7 & 2 & 4 \\ 1 & 3 & 5 \end{bmatrix}$$

$$A - B = \begin{bmatrix} 0-7 & 1-2 & 3-4 \\ 1-1 & 3-(-1) & 5-5 \end{bmatrix} = \begin{bmatrix} -7 & -1 & -1 \\ 0 & 4 & 0 \end{bmatrix}$$

Product of Matrices

Unlike addition and subtraction of matrices, the dimensions of the matrices matter in a different manner. The inner columns of the first matrix must match the number of rows of the second matrix. Hence for matrices A . B to be possible, A must have dimensions of m x n and B must have dimensions n x p where m, n and p are integers >= 1.

The result of A . B will yield a matrix whose dimensions are m x p.

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}$$
$$B = \begin{bmatrix} b_{11} & \cdots & b_{1p} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{np} \end{bmatrix}$$
$$A.B = \begin{bmatrix} r_{11}. & \cdots & r_{1p} \\ \vdots & \ddots & \vdots \\ r_{m1} & \cdots & r_{mp} \end{bmatrix}$$

Where r is a result obtained by summing up the product of each element in each row of A to each corresponding element of each column of B.

A better illustration can be seen in the example below:

$$P = \begin{bmatrix} 1 & 3 & 4 \\ 2 & 5 & 7 \end{bmatrix}$$
$$Q = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

In this case P is a 2 x 3 matrix and Q is a 3 x 1 matrix. The resultant matrix should be a 2 x 1 matrix.

$$P.Q = \begin{bmatrix} 1 & 3 & 4 \\ 2 & 5 & 7 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$
$$P.Q = \begin{bmatrix} 19 & 33 \end{bmatrix}$$

The first column is obtained by multiplying row 1 of P to column 1 of Q where the element p_{11} will multiply by q_{11} and p_{12} will multiply by q_{21} and p_{13} will multiply by q_{31}. Hence row 1 multiply by column 1 will give element$_{11}$

The working is given as 1*1 + 3*2 + 4*3 = 1 + 6 + 12 = 19.

The second column is obtained by multiplying row 2 of P to column 1 of Q where the element p_{21} will multiply by q_{21} and p_{12} will multiply by q_{21} and p_{23} will multiply by q_{31}. Hence row 2 multiply by column 1 will give element$_{12}$

The working is seen as 2*1 + 5*2 + 7*3 = 2 + 10 + 21 = 33.

Identity Matrix

In matrices, there are identity matrices and is worth mentioning in the case of multiplication. The idea is to search for a matrix such that when A x I or I x A where I is the Identity matrix, the result will still be A.

The identity matrix works only for matrices of dimensions m x m where m is any integer > 0. An identity matrix for a 2 x 2 matrix is:

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Similarly an identity matrix for a 3 x 3 matrix can be written as:

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

For our example we will multiply A by I and see that it gives a result of A.

$$A = \begin{bmatrix} 3 & 6 \\ 4 & 7 \end{bmatrix}$$

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$A \cdot I = \begin{bmatrix} 3 & 6 \\ 4 & 7 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 6 \\ 4 & 7 \end{bmatrix}$$

	Column 1	Column 2
Row 1	3 * 1 + 6 * 0 = 3	3 * 0 + 6 * 1 = 6
Row 2	4 * 1 + 7 * 0 = 4	4 * 0 + 7 * 1 = 7

Quick summary

In mathematics, a matrix is a rectangular array or two-dimensional array of numbers stored in rows and columns. A matrix is particularly useful in the context of computing as it can be thought of as an vector of vectors for storing items.
The operations that can be performed on matrices are addition, subtraction and multiplication. Other operations include transposition and scalar product on a matrix.

References

Gentle, J. E. (2017). Matrix Algebra: Theory, Computations and Applications in Statistics, N.Y: Springer.

Topic 06 – Boolean Algebra

Topic 06 – Learning Objectives

1. Describe Boolean Algebra
2. Describe Boolean Function
3. Describe Boolean Expression

Guiding Questions

1. What is Boolean Algebra?
2. What are the basic operations of Boolean Algebra?
3. What is the difference between AND gate, OR gate, and NOT gate?
4. What is a NAND gate, a NOR gate and a XOR gate?
5. What is a Truth Table?
6. What is a logic circuit?
7. What is a Boolean expression?
8. What is a Boolean Function?

What is Boolean Algebra?

First introduced by George Boole in 1854, Boolean Algebra is a mathematical discipline and a branch of algebra in which mathematical variables hold either the value of True or False (1 or 0). The Boolean Algebra was detailed in George Boole's treatise, An Investigation of the Laws of Thought on Which to Found the Mathematical Theories of Logic and Probabilities. In 1938, Claude Shannon, a research assistant in MIT, observed that Boolean Algebra could be useful in solving relay-switching circuit design.

Boolean algebra can be utilized in two areas of circuitry:

Analysis: It is an economical way of describing the function of digital circuitry.

Design: Given a desired function, Boolean algebra can be applied to develop a simplified implementation of that function. (Stallings, 2016).

Basic Logical Operations

Like any form of mathematical algebra, there are bound to be basic operations. In the Basic Logical Operations of Boolean Algebras, they are:

AND operation

A **AND** B = A . B or A ∧ B

In the AND operation, the result of A . B will yield a TRUE value if and only if both A and B are TRUE. If either or both of A and are FALSE, the result will be FALSE. (eg. If A is TRUE and B is FALSE, the result is FALSE. If A and B are TRUE, result is TRUE)

OR operation

A **OR** B = A + B or A ∨ B

Another operation known as the OR operation will give a TRUE result if A or B or both A and B are TRUE. The operation will only give a FALSE value of both A and B are FALSE. (eg. If A is TRUE and B is FALSE, the operation will give a TRUE value)

NOT operation

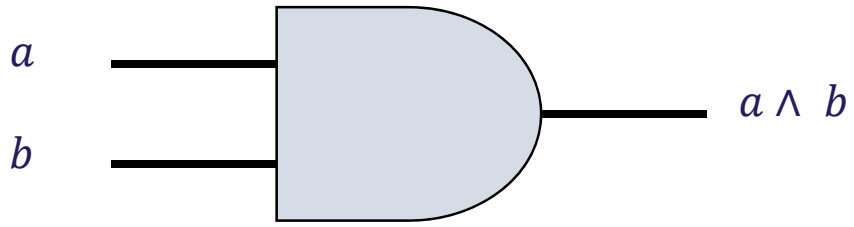
NOT A = $\bar{A}$

The unary operator also known as the Negation operator inverts the value of A. If A is TRUE, NOT A will yield FALSE. If A is FALSE, NOT A will yield TRUE.

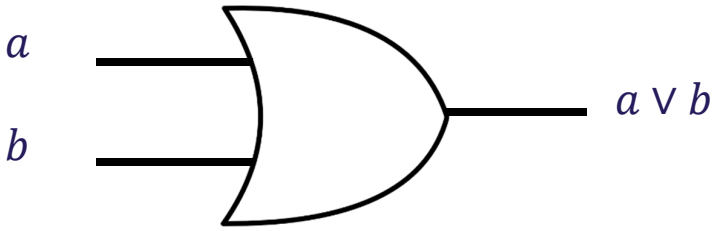
AND, OR and NOT Gates

The logical operators for the above could be implemented by the following gates:

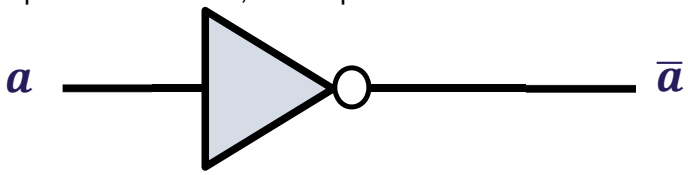
- AND Gate
  Both inputs a and b from the left hand side must be TRUE or 1 for the result to be TRUE or 1.



- OR Gate
  In an OR Gate, either a or b must be 1 to get a result of 1. The result will be 0 if and only if both a and b are 0.



- NOT Gate
  The signal of input “a” will be flipped. Hence if the input a has a value of 1, the output will be 0. If the input value of a is 0, the output will be 1.



Other Logical Operations

Besides the Basic Logical Operations of Boolean Algebras, other Logical Operators include NAND, NOR and XOR. The following operations show how NAND, NOR and XOR perform in a logical circuit.

NAND operation

A N**AND** B = $\bar{A}$ + $\bar{B}$ or $\bar{A} \vee \bar{B}$

We can see that NAND is an inversion of AND. The result of NAND will yield a TRUE in all circumstances except when both A and B are TRUE.

NOR operation

A **NOR** B = $\bar{A}$. $\bar{B}$ or $\bar{A} \wedge \bar{B}$

Like NAND, NOR is an inversion of the OR gate. The NOR gate will only give a result of TRUE when both A and B are FALSE

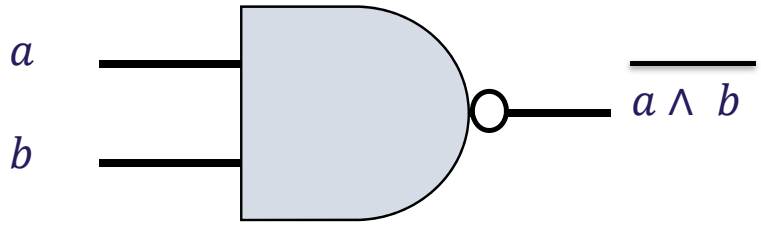
XOR operation

A **XOR** B = A $\oplus$ B

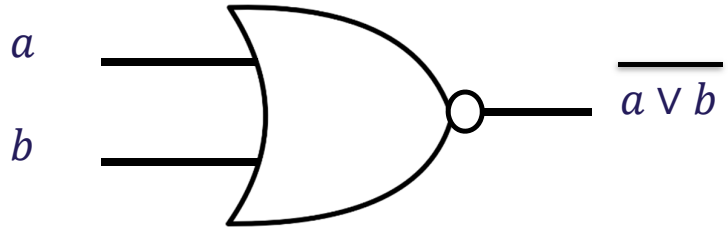
The eXclusive OR gate is a special gate that will yield a TRUE if and only if one of the inputs is TRUE. If both inputs are FALSE, the result will be FALSE. Similarly if both inputs are TRUE, the result will be FALSE as well.

NAND, NOR and XOR Gates
Besides the basic logical operators, the other logical operators above could be implemented by the following gates:

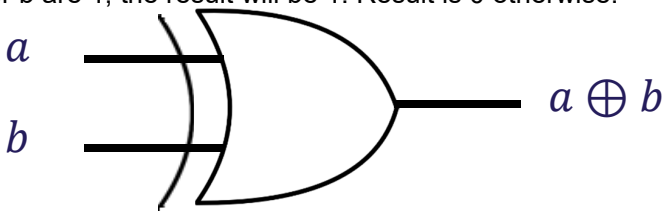
- NAND Gate
  Also known as NOT-a-NAND gate, the result will be 1 in all cases except when both a and b are 1.



- NOR Gate
  Like NAND, this is known as NOT-a-OR gate. The result will be 1 in all cases except when both A and B are 0.



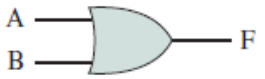
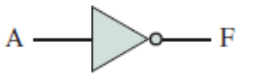
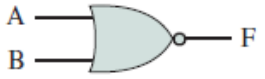
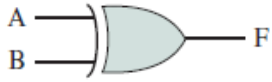
- XOR Gate
  One of the special gates known as the eXclusive-OR gate. If and only if 1 of the inputs a or b are 1, the result will be 1. Result is 0 otherwise.



Truth Tables

The goal of truth tables is to discover whether a propositional expression is true for all logically valid and legitimate input values. A truth table will have one column for every input variable (eg. 3 columns for inputs A, B and C). From the input columns there could be interim columns such as A v B and after deduction, there will be a final column to show all the possible results of the logical expression.

The following figure shows 2 inputs A and B with various operations (Stallings, 2016):

Name	Graphical Symbol	Algebraic Function	Truth Table
AND		$F = A \bullet B$ or $F = AB$	A B | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$F = A + B$	A B | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
NOT		$F = \overline{A}$ or $F = A'$	A | F 0 | 1 1 | 0
NAND		$F = \overline{AB}$	A B | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		$F = \overline{A + B}$	A B | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
XOR		$F = A \oplus B$	A B | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0

It is very often useful to have interim columns and then the interim results can then be combined with multiple inputs with the final result F.

The table for the above logic gates could be written in another manner as such:

(a) Boolean Operators of Two Input Variables

P	Q	NOT P ($\overline{P}$)	P AND Q ($P \cdot Q$)	P OR Q ($P + Q$)	P NAND Q ($\overline{P \cdot Q}$)	P NOR Q ($\overline{P + Q}$)	P XOR Q ($P \oplus Q$)
0	0	1	0	0	1	1	0
0	1	1	0	1	1	0	1
1	0	0	0	1	1	0	1
1	1	0	1	1	0	0	0

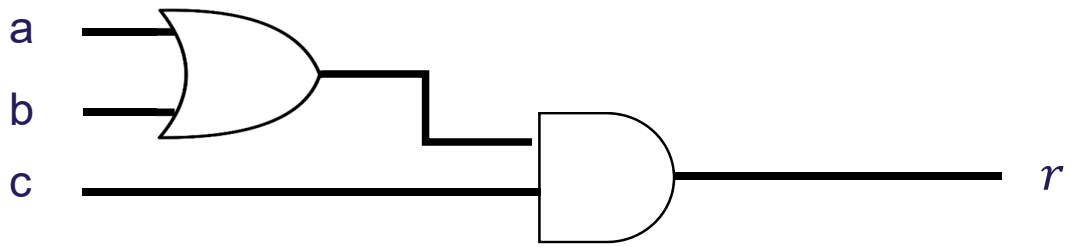
Boolean Operators

Digital Logic (Stallings, 2016)

Logic Circuits

A logic circuit is a combination of logic gates and devices such as multiplexers and registers. A combinational circuit is an interconnected set of gates whose output at any time is a function only of the inputs at that time. (Stallings, 2016). To provide a state to the logic circuits, sequential circuits are needed to be implemented by providing a feedback to itself. For the purpose of this guide, we will focus on combinational circuits.

A logic circuit can be seen (combining a OR and a AND gate) below:



The example shows that 3 inputs can be represented by 2 gates (ie. A and B represented by an OR gate and the C input as part of the inputs of a AND gate).

The layout shows the possibility in other combinations as well. To extend the circuits to have other inputs, one just have to put in more logic gates to combine to form any combinations of a logic circuit. This means a limitless possibility of circuit combinations!

Boolean Functions and Expressions

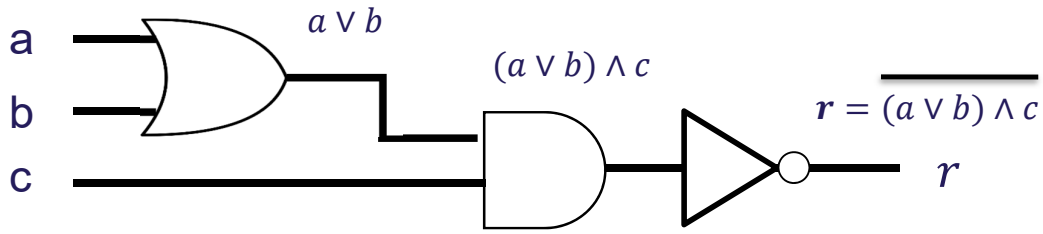
Certainly a logic circuit can be built into a complex circuit with many logic gates combined. The question remained on how to represent the logic circuit in a simplest form such that when the values of the inputs are supplied, we can get a logical output. One method is to write the truth table for such a circuit to determine all the possible outcomes for any combination. Truth tables can be very big if there were multiple inputs. Therefore, a better way must be sought to represent the circuitry better.

Indeed in logic circuitry instead of truth tables, there is another convenient way which is to represent the combination of inputs with a single Boolean function.

What is a Boolean Function? A Boolean function is a mathematical function which maps Boolean inputs to derive the Boolean output. To represent a Boolean function, a Boolean Expression will be required.

What is a Boolean Expression? A Boolean Expression is a mathematical expression that will produce a Boolean value (TRUE or FALSE) when input values are known.

Using the same example with the OR and AND gate, the circuit could be rewritten as follows:



Basic Identities of Boolean Algebra

Deriving from the truth tables, the basic identities of Boolean Algebra can now be formed. The following table depicts the basic postulates as well as other identities. The identities can be proven with reconstructing the truth tables to verify the outputs against the inputs.

Basic Postulates		
$A \cdot B = B \cdot A$	$A + B = B + A$	Commutative Laws
$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$	$A + (B \cdot C) = (A + B) \cdot (A + C)$	Distributive Laws
$1 \cdot A = A$	$0 + A = A$	Identity Elements
$A \cdot \overline{A} = 0$	$A + \overline{A} = 1$	Inverse Elements
Other Identities		
$0 \cdot A = 0$	$1 + A = 1$	
$A \cdot A = A$	$A + A = A$	
$A \cdot (B \cdot C) = (A \cdot B) \cdot C$	$A + (B + C) = (A + B) + C$	Associative Laws
$\overline{A \cdot B} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A} \cdot \overline{B}$	DeMorgan's Theorem

Identities of Boolean Algebra

Digital Logic (Stallings, 2016)

Quick summary

A branch of algebra, known as the Boolean Algebra is created with values focused on only TRUE or FALSE (1 or 0). It is later found that Boolean Algebra has its use in logic circuits and is still used today in simplify logical circuits. The Basic operations of Boolean Algebra are AND, OR and NOT and other notable operations are NAND, NOR and XOR.
With these operations, logical circuits can be simplified with Boolean functions and expressions which in turn can be proven to with truth tables.

References

Stallings, W. (2016). Computer organization and architecture: Designing for performance. Hoboken, N.J: Pearson.

Stallings, W. (2018). Operating Systems: Internals and Design Principles. 9th edition: Pearson.

Topic 07 – Boolean Algebra II

Learning Objectives

1. Describe Boolean Expression

Guiding Questions

1. What are the Basic Laws of Boolean Algebra?
2. What are De Morgan's Theorems 1 and 2?
3. How do we apply the Laws of Boolean Algebra?
4. How do we verify the Laws of Boolean Algebra?

What are the Basic Law of Boolean Algebra?

The purpose of Boolean Algebra is to simplify logical circuits and define the operation of a digital circuit. To simplify logical circuits, a set of Laws was developed by George Boole. George Boole conceptualized the Laws to represent logic by using a form of algebra. With algebra, one can simplify the logical gates by simplifying the Boolean expression using Math-like operations.

The Laws of Boolean Algebra can be categorized into the following categories:

- Associative Laws
- Commutative Laws
- Distributive Laws
- Identity Laws
- Idempotent Laws
- Complement Laws
- Annulment Laws
- Absorption Laws

Other Laws: There are TWO other theorems that came later into the picture. These laws are notable and they are known as De Morgan's Theorems 1 and 2.

Associative Laws

The use of Associative Laws is to attempt to remove brackets of from the Boolean expression and try to regroup the variables in the bid to simplify the expression.

- $(a \vee b) \vee c = a \vee (b \vee c)$
    - **Also written as (A + B) + C = A + (B + C)**
    - **(A OR B) OR C = A OR (B OR C)**
- $(a \wedge b) \wedge c = a \wedge (b \wedge c)$
    - **Also written as (A . B) . C = A . (B . C)**
    - **(A AND B) AND C = A AND (B AND C)**

Commutative Laws

The use of Commutative Laws is to attempt to reorder the variables if there is a need to simplify the expression.

- $a \vee b = b \vee a$
    - **Also written as A + B = B + A**
    - **A OR B = B OR A**
- $a \wedge b = b \wedge a$
    - **Also written as A . B = B . A**
    - **A AND B = B AND A**

Distributive Laws

The use of Distributive Laws is to allow the multiplication of an expression or to factor the expression.

- $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$
    - **Also written as A . (B + C) = (A . B) + (A . C)**
    - **A AND (B OR C) = (A AND B) OR (A AND C)**
- $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$

- **Also written as A + (B . C) = (A + B) . (A + C)**
- **A OR (B AND C) = (A OR B) AND (A OR C)**

Identity Laws

Identity laws when combined with “OR” or “AND” will always return the value of itself.

- $a \vee 0 = a$
  - **Also written as A + 0 = A**
  - **A OR 0 = A**
- $a \wedge 1 = a$
  - **Also written as A . 1 = A**
  - **A AND 1 = A**

Idempotent Laws

Like Identity laws when combined with “OR” or “AND” with itself, the result will always be the value of itself. These Laws are useful for removing duplicates to simplify the expression. These laws may also be useful for adding duplicates to combine with distributive laws/ commutative laws.

- $a \vee a = a$
  - **Also written as A + A = A**
  - **A OR A = A**
- $a \wedge a = a$
  - **Also written as A . A = A**
  - **A AND A = A**

Complement Laws

Complement Laws are useful when combining a complement of itself to produce either 1 or 0.

- $a \vee a' = 1$
    - **Also written as A + A' = 1**
    - **A OR NOT A = 1**
- $a \wedge a' = 0$
    - **Also written A . A' = 0**
    - **A AND NOT A = 0**

Annulment Laws

The results of Annulment Laws are useful when faced with combination with 1 or 0 which will produce 1 and 0 as well depending on "OR" or "AND".

- $a \vee 1 = 1$
    - **Also written as A + 1 = 1**
    - **A OR 1 = 1**
- $a \wedge 0 = 0$
    - **Also written as A . 0 = 0**
    - **A AND 0 = 0**

Absorption Laws

The results of Annulment Laws are useful when faced with combination with 1 or 0 which will produce 1 and 0 as well depending on "OR" or "AND".

- $a \vee (a \wedge b) = a$
    - **Also written as A + (A . B) = A**

- **A OR (A AND B) = A**

- $\boldsymbol{a} \wedge (\boldsymbol{a} \vee \boldsymbol{b}) = \boldsymbol{a}$
  - **Also written as A . (A + B) = A**
  - **A AND (A OR B) = A**

What are De Morgan Theorems?

There are 2 theorems which are particularly solving inversed expressions to simplify Boolean expressions. They are:

- $\neg(\boldsymbol{a} \vee \boldsymbol{b}) = \neg\boldsymbol{a} \wedge \neg\boldsymbol{b}$
  - **Also written as (A + B)' = A' . B'**
  - **NOT (A OR B) = NOT A AND NOT B**
- $\neg(\boldsymbol{a} \wedge \boldsymbol{b}) = \neg\boldsymbol{a} \vee \neg\boldsymbol{b}$
  - **Also written as (A . B)' = A' + B'**
  - **NOT (A AND B) = NOT A OR NOT B**

Boolean Expression Example 1

Prove that the Absorption Law is true.

Prove $\boldsymbol{X} \vee (\boldsymbol{X} \wedge \boldsymbol{Y}) = \boldsymbol{X}$

- **LHS**
- **=** $\boldsymbol{X} \vee (\boldsymbol{X} \wedge \boldsymbol{Y})$
- = $(\boldsymbol{X} \wedge \boldsymbol{1}) \vee (\boldsymbol{X} \wedge \boldsymbol{Y})$ **(Identity Law)**
- = $\boldsymbol{X} \wedge (\boldsymbol{1} \vee \boldsymbol{Y})$ **(Distributive Law)**
- **=** $\boldsymbol{X} \wedge \boldsymbol{1}$ **(Identity Law)**
- **=** $\boldsymbol{X}$

Verify the expression is true using the Truth Table.

X	Y	$\boldsymbol{X} \wedge \boldsymbol{Y}$	$\boldsymbol{X} \vee (\boldsymbol{X} \wedge \boldsymbol{Y})$
1	1	1	1
1	0	0	1
0	1	0	0
0	0	0	0

We can see the first column X is equal to the last column and verify that the expression is indeed equal. (LHS = RHS)

$\boldsymbol{X} \vee (\boldsymbol{X} \wedge \boldsymbol{Y}) = \boldsymbol{X}$ **(Proved)**

Boolean Expression Example 2

Prove $\boldsymbol{X} \vee (\neg \boldsymbol{X} \wedge \boldsymbol{Y}) = \boldsymbol{X} \vee \boldsymbol{Y}$

- **LHS**
- **=** $\boldsymbol{X} \vee (\neg \boldsymbol{X} \wedge \boldsymbol{Y})$
- = $\big(\boldsymbol{X} \vee (\boldsymbol{X} \wedge \boldsymbol{Y})\big) \vee (\neg \boldsymbol{X} \wedge \boldsymbol{Y})$ **(Absorption Law)**
- **=** $((\boldsymbol{X} \wedge \boldsymbol{X}) \vee (\boldsymbol{X} \wedge \boldsymbol{Y})) \vee (\neg \boldsymbol{X} \wedge \boldsymbol{Y})$ **(Idempotent Law)**
- **=** $(\boldsymbol{X} \wedge \boldsymbol{X}) \vee (\boldsymbol{X} \wedge \boldsymbol{Y}) \vee (\boldsymbol{X} \wedge \neg \boldsymbol{X}) \vee (\neg \boldsymbol{X} \wedge \boldsymbol{Y})$ **(Complement Law)**
- **=** $\big((\boldsymbol{X} \wedge \boldsymbol{X}) \vee (\boldsymbol{X} \wedge \neg \boldsymbol{X})\big) \vee \big((\boldsymbol{X} \wedge \boldsymbol{Y}) \vee (\neg \boldsymbol{X} \wedge \boldsymbol{Y})\big)$ **(Commutative Law)**
- **=** $\big(\boldsymbol{X} \wedge (\boldsymbol{X} \vee \neg \boldsymbol{X})\big) \vee \big(\boldsymbol{Y} \wedge (\boldsymbol{X} \vee \neg \boldsymbol{X})\big)$ **(Distributive Law)**
- **=** $(\boldsymbol{X} \wedge \boldsymbol{1}) \vee (\boldsymbol{Y} \wedge \boldsymbol{1})$ **(Complement Law)**
- **=** $\boldsymbol{X} \vee \boldsymbol{Y}$ **(Proved)**

Verify that the Boolean expression is true using the Truth Table as shown below:

X	Y	$\boldsymbol{X} \vee \boldsymbol{Y}$	$\neg \boldsymbol{X}$	$\neg \boldsymbol{X} \wedge \boldsymbol{Y}$	$\boldsymbol{X} \vee (\neg \boldsymbol{X} \wedge \boldsymbol{Y})$
1	1	1	0	0	1
1	0	1	0	0	1
0	1	1	1	1	1
0	0	0	1	0	0

From the table, we observe that column 3 has the same values as the last column. This means that the LHS = RHS and indeed the expression is true.
$\boldsymbol{X} \vee (\neg \boldsymbol{X} \wedge \boldsymbol{Y}) = \boldsymbol{X} \vee \boldsymbol{Y}$ **(Proved)**

Basic Identities of Boolean Algebra

The following table shows the summary of all the laws created by George Boole and other laws such as DeMorgan's Theorem. These theorems are very useful in solving Boolean Expressions and simplifying them. With Boolean Algebra, you can simplify a longer Boolean expression to a shorter expression which means less logic gates and circuits can be used.

Basic Postulates		
$A \cdot B = B \cdot A$	$A + B = B + A$	Commutative Laws
$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$	$A + (B \cdot C) = (A + B) \cdot (A + C)$	Distributive Laws
$1 \cdot A = A$	$0 + A = A$	Identity Elements
$A \cdot \overline{A} = 0$	$A + \overline{A} = 1$	Inverse Elements
Other Identities		
$0 \cdot A = 0$	$1 + A = 1$	
$A \cdot A = A$	$A + A = A$	
$A \cdot (B \cdot C) = (A \cdot B) \cdot C$	$A + (B + C) = (A + B) + C$	Associative Laws
$\overline{A \cdot B} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A} \cdot \overline{B}$	DeMorgan's Theorem

Identities of Boolean Algebra

Digital Logic (Stallings, 2016)

All these laws shown in the table can be easily proven by writing the Truth Tables and showing that the LHS = RHS like how you would prove in a normal mathematical equation.

Quick summary

Boolean Algebra has basic laws such as associative laws, complement laws, distributive laws, identity laws, annulment laws, idempotent laws and absorption laws. On top of that, there are also De Morgan's theorems 1 and 2. All these laws and theorems can help in logic circuits and is still used today in simplify logical circuits.

The theorems can be proven with the use of Truth tables to prove that the LHS = RHS.

References List

Stallings, W. (2016). Computer organization and architecture: Designing for performance. Hoboken, N.J: Pearson.

Stallings, W. (2018). Operating Systems: Internals and Design Principles. 9th edition: Pearson.

Topic 08 – MATLAB Introduction, Environment and Functions

Learning Objectives

1. Describe what MATLAB is
2. Describe why MATLAB is widely used in engineering and science
3. List the advantages and limitations of MATLAB
4. Explain how to formulate problem using a structured problem solving approach
5. Explain how to launch the MATLAB program
6. Explain the use of various MATLAB windows
7. Explain the use of matrices
8. Explain the use of variables
9. Describe the various common mathematical functions
10. Describe the use of trigonometric functions in MATLAB
11. Explain how to compute and use statistical and data analysis functions
12. Generate uniform and Gaussian random-number matrices
13. List the computational limits of MATLAB
14. Explain how to use the special values and functions built into MATLAB

Guiding Questions

1. What is MATLAB?
2. Why is MATLAB popular in engineering and science?
3. What are the advantages and limitations of MATLAB?
4. How do we solve a problem using MATLAB?
5. What types of windows are available in MATLAB?
6. What are variables?
7. What are matrices?
8. What is a Function?
9. What are the basic mathematical functions in MATLAB?
10. Which trigonometric functions are supported in MATLAB?
11. Which functions are used for statistical and data analysis in MATLAB?
12. How do we generate matrices with uniform OR Gaussian random numbers?
13. What are the computational limits of MATLAB?
14. What are special values or functions in MATLAB?

What is MATLAB?

MATLAB is a programming platform and mathematical computing software specially designed for engineers and scientists. MATLAB is trusted by millions of scientists and engineers due its focus on mathematical computing and solutions created for that purpose.

MATLAB is also known as Matrix Laboratory. The name is evident that the language is focused on matrix computation.

Why is MATLAB popular in engineering and science?

MATLAB provides many libraries and functions for the work of engineers and scientists such as:

Data Analytics

Data Analytics is a field that allows scientists to explore different datasets and relate them together to make sense of them. MATLAB makes data analytics simple with ample tools to access and process data.

Data can be easily stored in flat file formats, databases and cloud storage. MATLAB allows easy management and cleaning of data using its processing capabilities.

Simulations

Simulations are often used in various sectors to achieve a similar process in the real world and this allows scientists, engineers or even financiers to be confident in their assumptions that their models or creations are correct. As such Simulink is used for prototyping before actual production.

With Simulink as the for modelling, simulations are easily achieved with programming. With the sim() function, you can use the sim to enable timeouts, catch simulation errors and exceptions and access metadata.

Machine Learning

MATLAB provides the needed processing for data which is the basis for machine learning. Machine learning is the study of computer algorithms to forecast the future based on past actions. Machine language is part of artificial intelligence.

MATLAB uses AutoML to enable scientists to create optimized models with a simplified workflow. Without machine learning expertise, MATLAB allows one to build models in just three steps.

Deep Learning

Deep Learning is a key technology in driverless vehicles. It is key to voice enabled technologies. Deep Learning imitates the workings of the human brain in processing data and hence seeks to use the patterns for decision making.

Deep learning is especially suited for image recognition, which is important for solving problems such as facial recognition, motion detection, and many advanced driver assistance technologies such as autonomous driving, lane detection, pedestrian detection, and autonomous parking. (Mathworks)

MATLAB utilizes Deep Learning Toolbox to allow usage of simple MATLAB commands to create and interconnect networks. MATLAB enables deep learning by transfer learning approach.

Robotics

Robotics deals with creation, design and operation of robots. Robotics is used traditionally in industry as industrial robots and this has changed over the years as robots become cheaper to build and put together. Applications in everyday use is apparent in the global robotic vacuum cleaner which has seen exponential growth over the last 5 years. The advantages are obvious as people seek to make better use of their time and leave the cleaning to the robots and hence decreased cleaning time. With MATLAB and Simulink you can:

- Connect and control the robot with applications you created
- Customise and connect to various sensors and actuators to send control signals or analyze many types of data.
- Reduce coding on developer's end as MATLAB and Simulink can generate code for embedded chips
- Work with Arduino and Raspberry Pi to reduce cost using support packages
- Continue to use legacy code and integrate with existing systems.

Image and Video Processing

MATLAB has tools and algorithms to analyze images and videos. As mentioned, MATLAB has the ability to track face and motion and hence recognize facial features.

Signal Processing

MATLAB allows sampling of data from signals and has functions that aid in analyzing, preprocessing, filtering, resampling smoothing, detrending, finding peaks and signal patterns and many more. MATLAB has the Signal Analyzer app and the Filter Design App to preprocess signals and design digital filters respectively and these 2 apps generate MATLAB code.

MATLAB Advantages and Limitations

Every tool has its pros and cons and MATLAB will certainly have its strengths and weaknesses. Let us take a look at the advantages of MATLAB first.

Advantages

- Easy to learn
  - With simple constructs, it is one of the easiest languages to pick up
- It is a scripting language
  - Scripting languages have one major advantage over compiled languages: There is no need to compile.
  - The good thing about non-compilation is that errors are generated usually at the point where there is a mistake. It gives the coder a chance to see part of the program run
- It has many libraries for engineering purposes
  - With a focused language meant for engineering, it means that the engineers need not hunt for other packages to run the programs they need
- MATLAB supports modelling
- Cross platform capability
  - MATLAB is capable of running on different platforms such as Windows, macOS and Linux.
  - When you write MATLAB code, you can easily transfer to another platform without rewriting the code

Limitations

- High Level Language
  - MATLAB is a high level scripting language
  - Unable to support low level programming that deals with drivers
- Not free
  - MATLAB is not free. It costs for every license
  - For additional modules, you will need to purchase them.
- Slower than compiled language
  - MATLAB is a scripting language which means it is an interpreted language. Interpreted languages are run on the fly, hence the speed is definitely slower compared with compiled languages

How do we solve a problem using MATLAB?

Programs are meant to solve problems and to solve problems using a programming platform or language such as MATLAB is to following a 5 step approach:
Step 1: Define the Problem
Step 2: Identify Inputs and Outputs
Step 3: Create an Algorithm
Step 4: Implement the Solution (Using MATLAB)
Step 5: Test the Solution (Iterate if required)

Step 1: Define the Problem

- Assess the situation and makes sense of what are facts
  - Read up the problem statement
- Look at source of the problem
- Determine in which process the problem lies

Step 2: Identify Inputs and Outputs

- Determine inputs
  - Inputs are incoming data to the program.
  - Inputs can be seen as inputs via keyboard, mouse, sound, images
  - Inputs are then processed by the program to produce output(s)
- Determine the output(s)
  - Look at the desired outcome of the program
  - Output(s) is/are a good place to look at if you have some trouble understanding the problem. By looking at the output, you can figure out what the problem is.

Step 3: Create an Algorithm

- An algorithm is the step by step method to solve a problem
- Creating an algorithm will help to detect errors early on before implementing the solution itself.
- By determining whether an algorithm will work, one can save time and cost compared with finding errors in the actual solution.

Step 4: Implement the Solution

- At this stage, the physical product or virtual product should be created.
- In MATLAB, it is the program that is capable of running and solving the problem
- The program should follow the algorithm closely and implement the steps in the algorithm

Step 5: Test the Solution (Iterate if required)

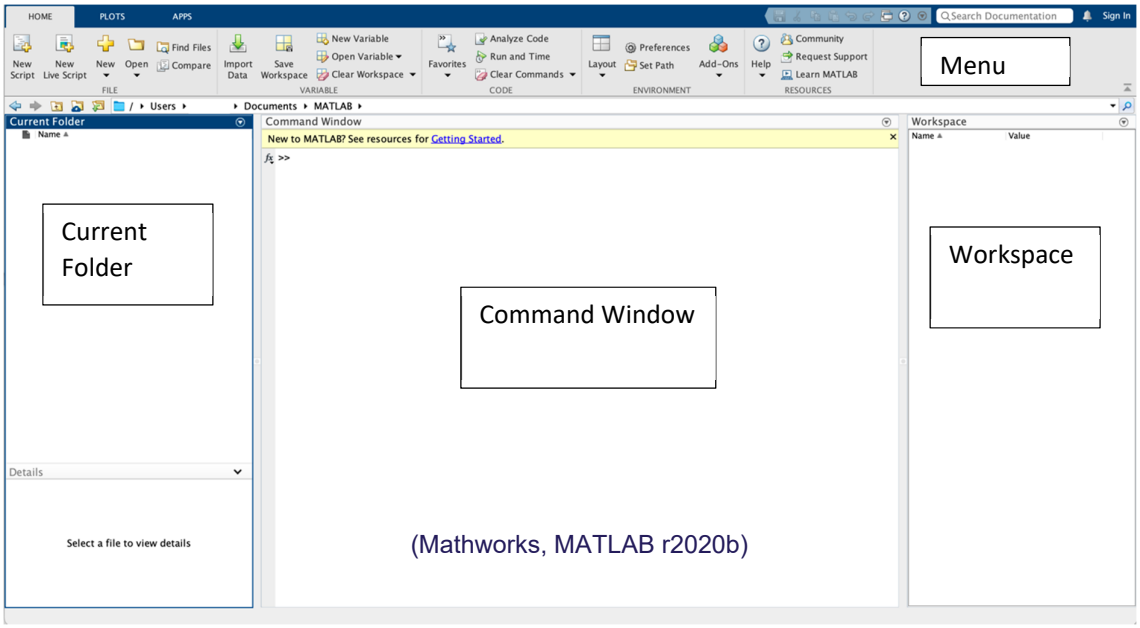
- Test the solution
  - A product is not really complete until you test the solution
  - A product must be error free before launch
  - If a problems or errors are detected, the right thing must be done. Iterate through the previous stages to correct the issue(s).

What types of windows are there in MATLAB?

MATLAB has a number of notable windows such as:

- Command Window
- Workspace Window
- Workspace Variables Window

- Command History Window
- Current Folder Window
- Edit Window
- Plot Gallery Window



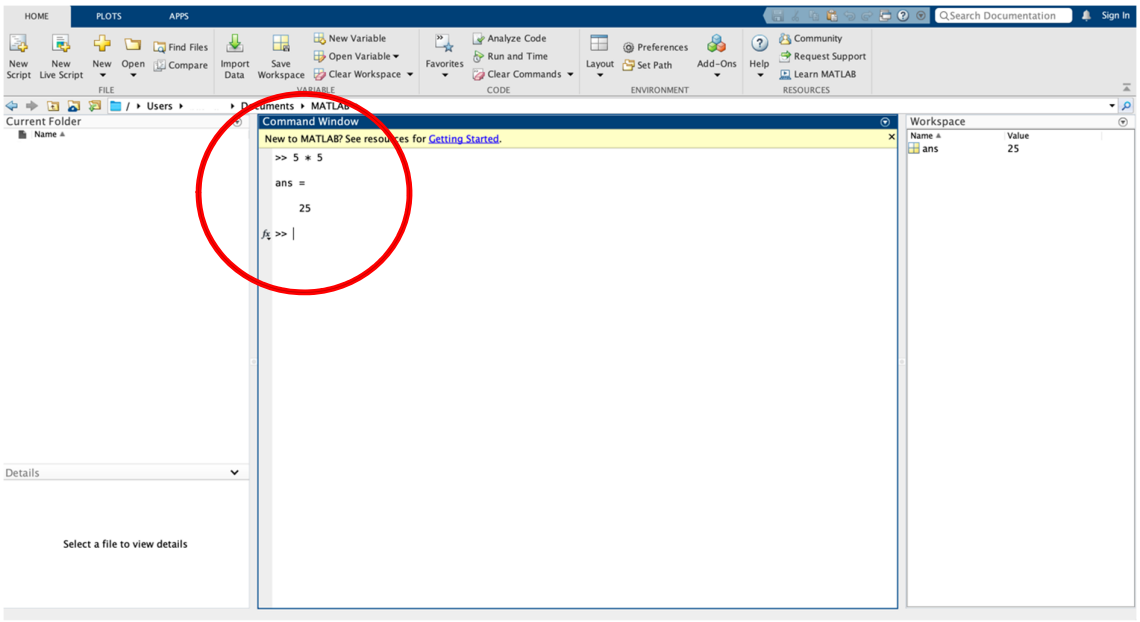
MATLAB Command Window

The Command Window is the window where you can enter commands to run. It is the window where the action is. Executing simple arithmetic calculation like product, difference, addition and subtraction is done easily simply by entering the mathematical formula.

The Command Window is also the place to see the output when a MATLAB script is run.

Examples of commands to try out

- 5 * 5
- 5 ^3
- 6 * (3 + 2^2) – 3

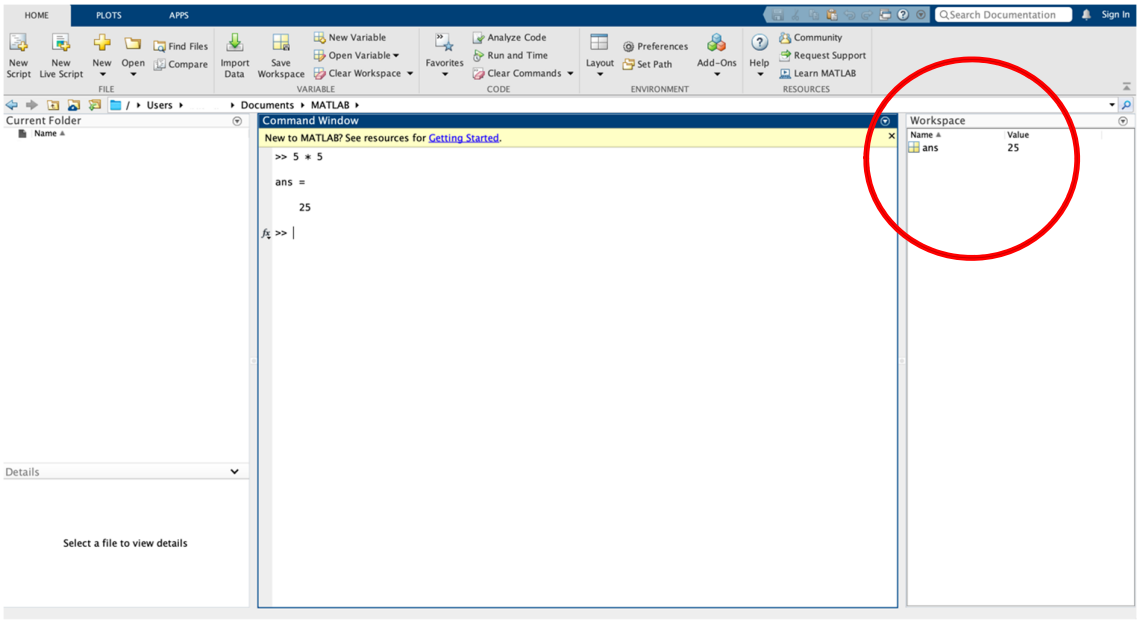


(Mathworks, MATLAB r2020b)

MATLAB Workspace Window

MATLAB has a Workspace Window which stores variables and their values whenever the user runs a program or some commands in the command window. The default value is stored in a variable called “ans”

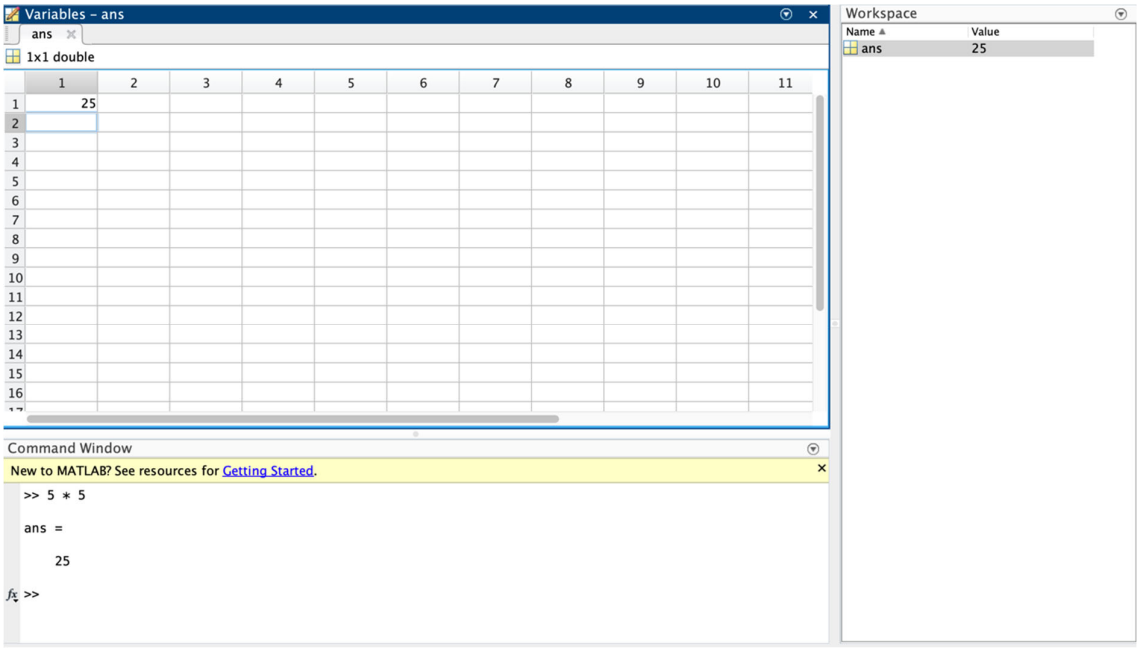
The value of “ans” is seen in the workspace window. You can change the value of “ans” by simply entering another command or double clicking the variable to view the contents in the Workspace Variable Window.



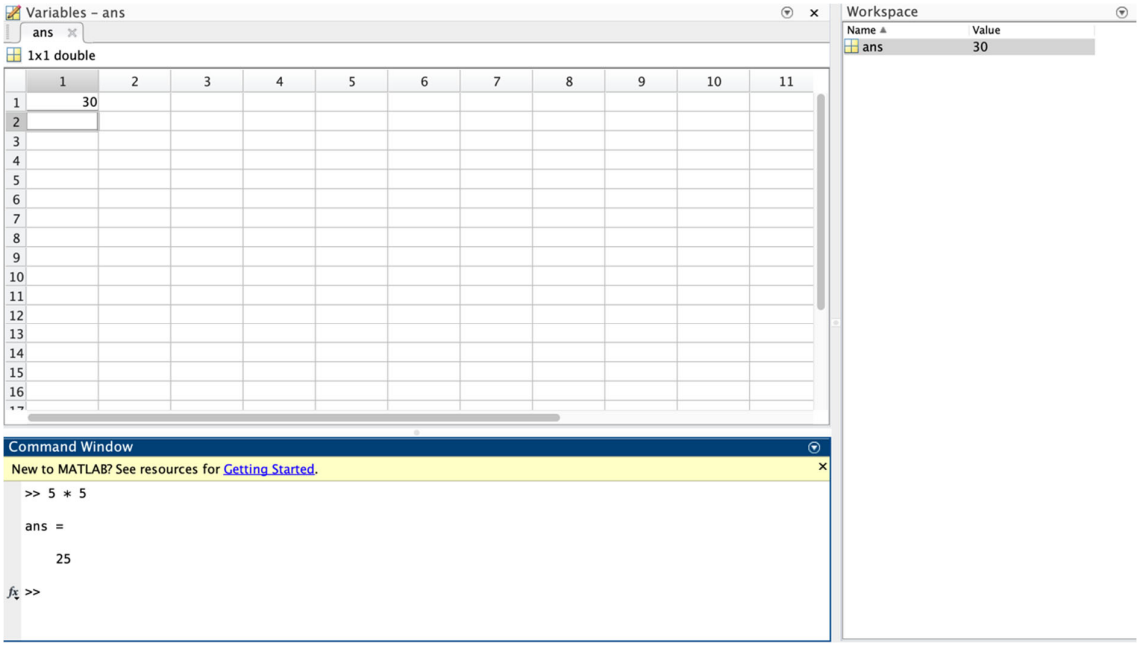
MATLAB Workspace Variables Window

The MATLAB Workspace Variables Window allows you to modify or add values to the variables already in the workspace.

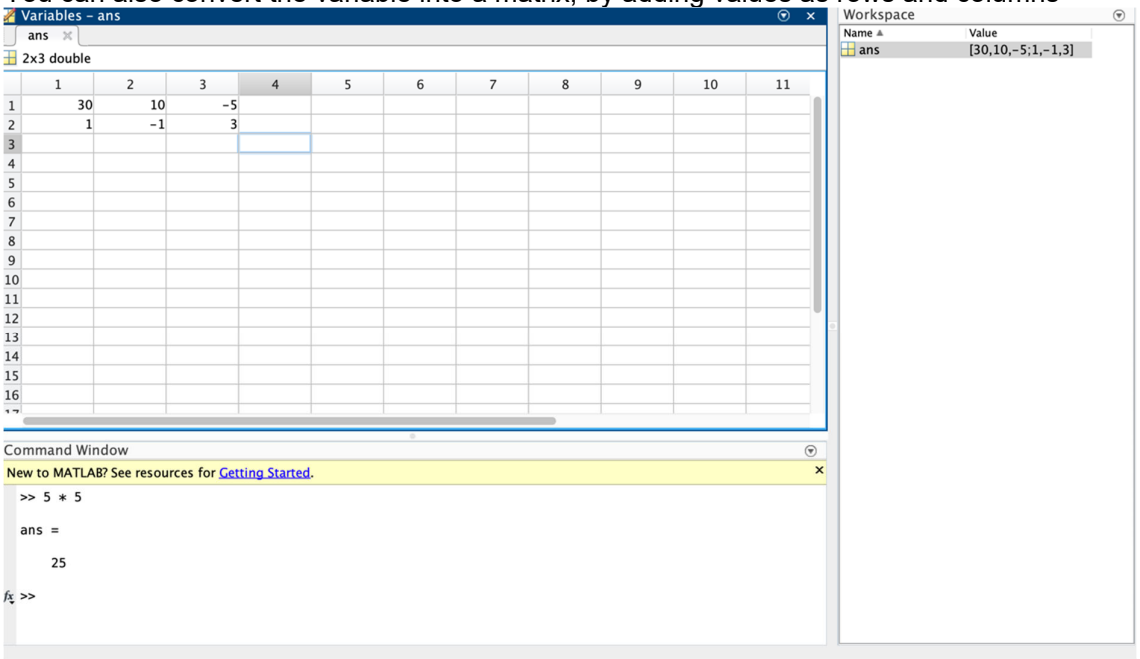
To reach the Workspace Variables Window, double click the variable in the Workspace Window. The Workspace Variables Window will now appear and look similar to a spreadsheet.



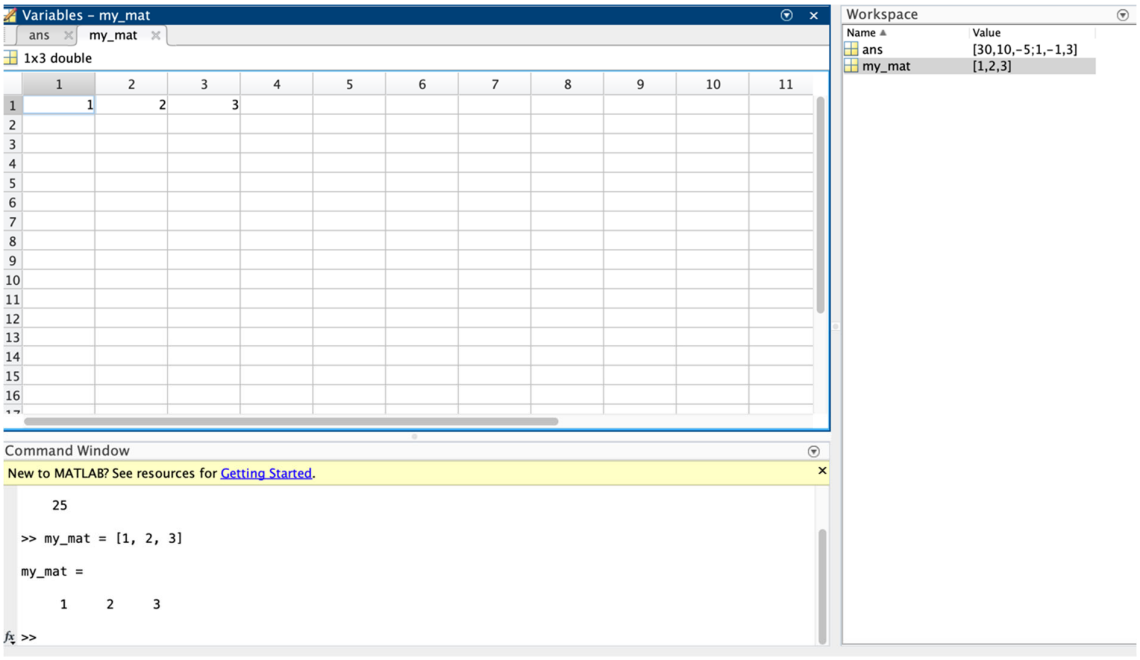
To change the value of the variable, overwrite the value in the Workspace Variables Window.



You can also convert the variable into a matrix, by adding values as rows and columns



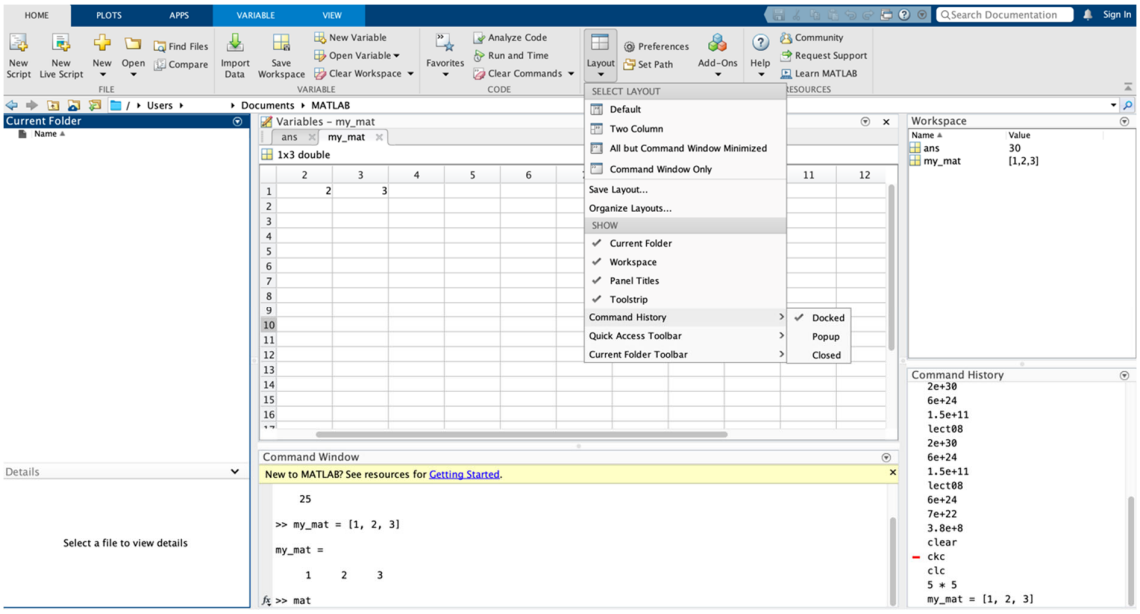
You can also add variables of your own instead of using the default “ans” variable. You can key in my_mat = [1, 2, 3] as an example in the Command Window and press “Enter”. You will now see the new variable in the Workspace. Open up my_mat by double clicking on the variable name and you will see it as a single row matrix in the Workspace Variables Window.



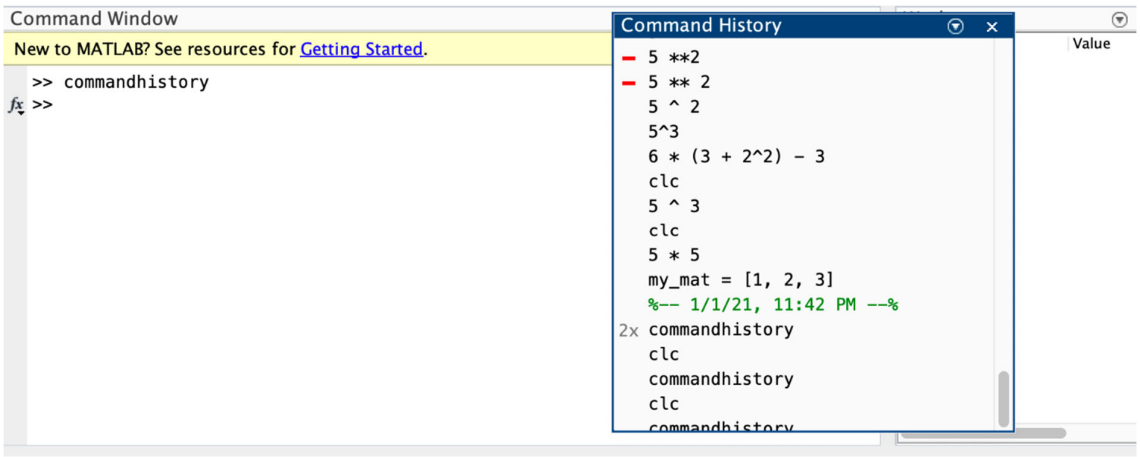
MATLAB Command History Window

The MATLAB Command History Window allows one to review his/her past actions/commands. By default, the Command History can store 25,000 statements! That is very useful indeed. The Command History Window also allows the user repeat a past action by clicking on the previous command to execute the command again.

The default setting for Command History Window is pop up window. To open the Command History Window, go to Layout in the Menu and select Command History -> Docked. This action will dock the Command History Window just below the Workspace Window and show the list of commands you have previously entered as seen in the screenshot.

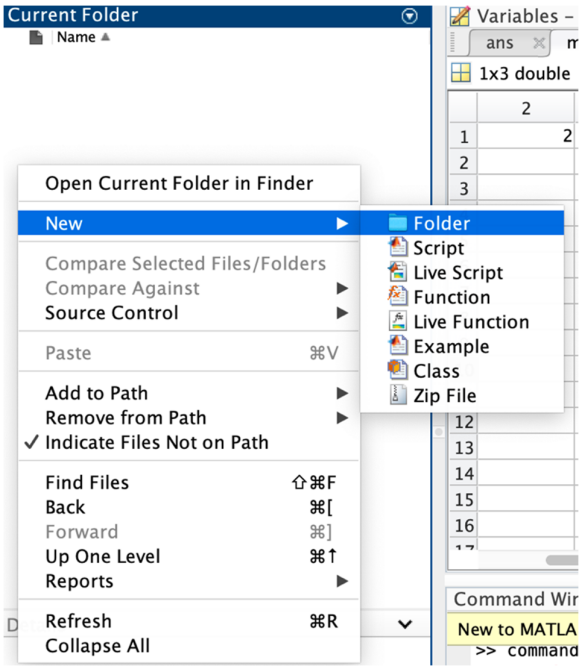


Alternatively you can type “commandhistory” in the Command Window to display the previous commands

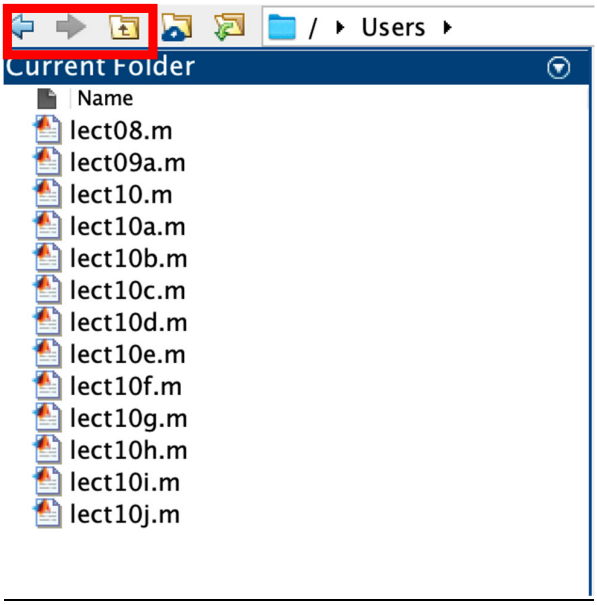


MATLAB Current Folder Window

MATLAB has a Current Folder Window to show the current directory or path that you are in. You can create files/folders in there or navigate the source files from there.



If you wish to navigate to the parent directory, you can click on the folder icon. The arrow icons can allow you to move forward or to previous folder like a history command.

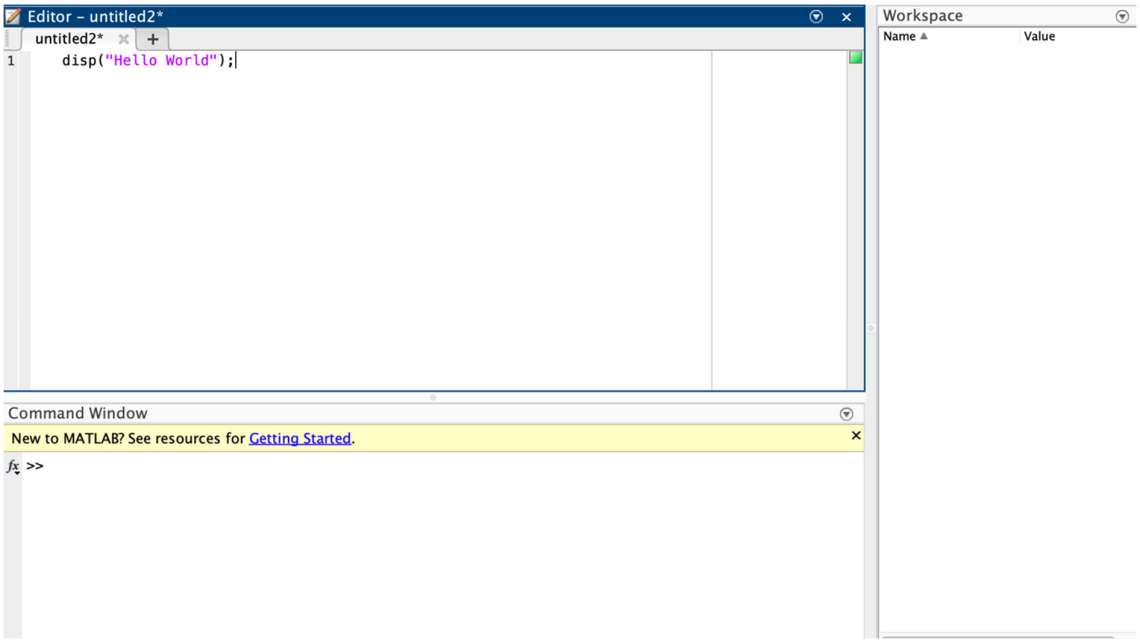


While MATLAB allows you to key in command after command in the Command Window, it is not useful if you cannot save these commands. MATLAB has the facility to allow you to save commands in files and allow you to modify the commands in the Edit Window.

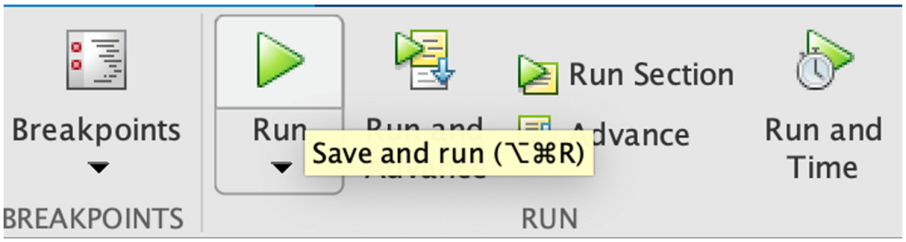
To create the script, you can choose to:

- right click in the Current Window and select New -> Script OR;
- you can choose New Script in the menu.

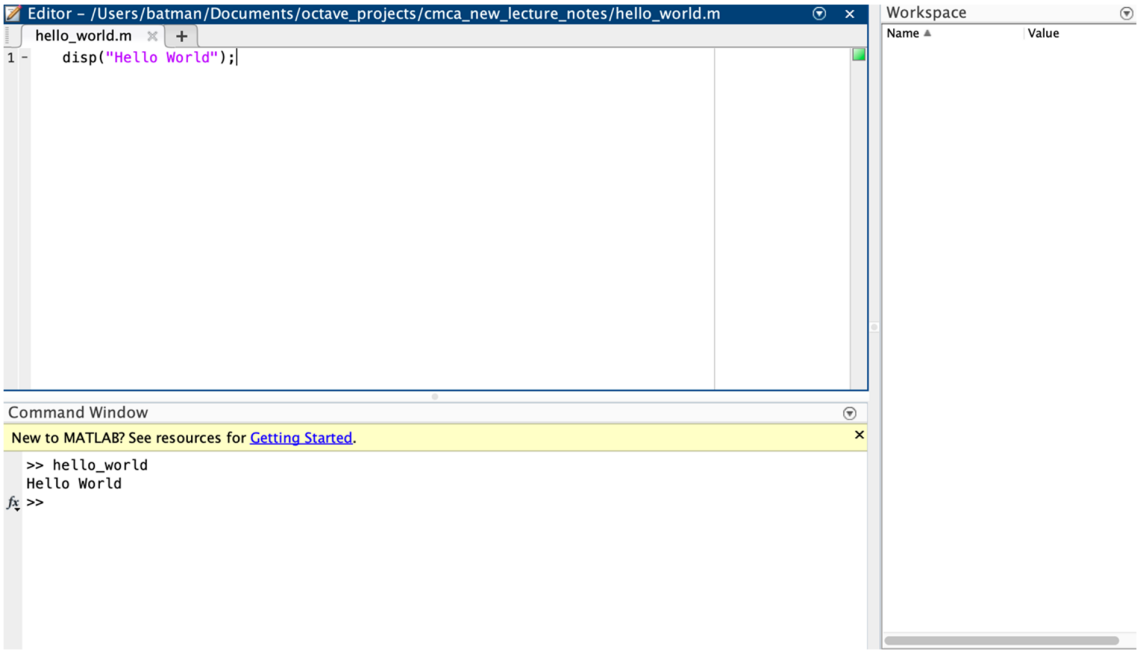
The Edit Window is where you will be writing your programs in the future.



After typing in the program, click on Run button.



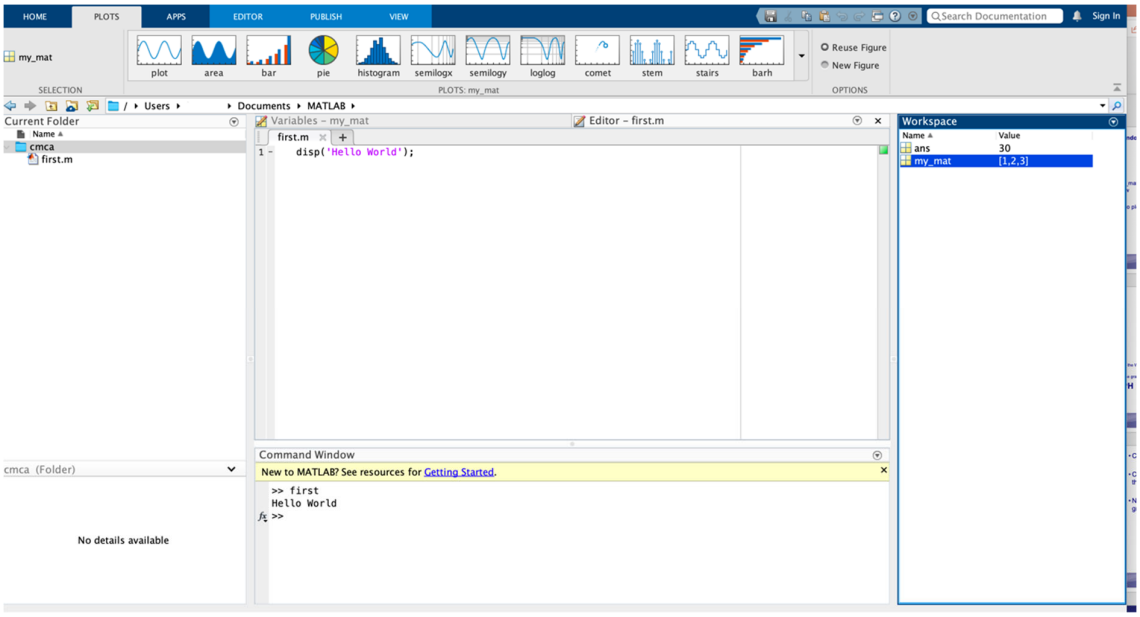
You will be prompted to save the file. Give it a filename and save the file. You will see the commands / program's output in the Command Window

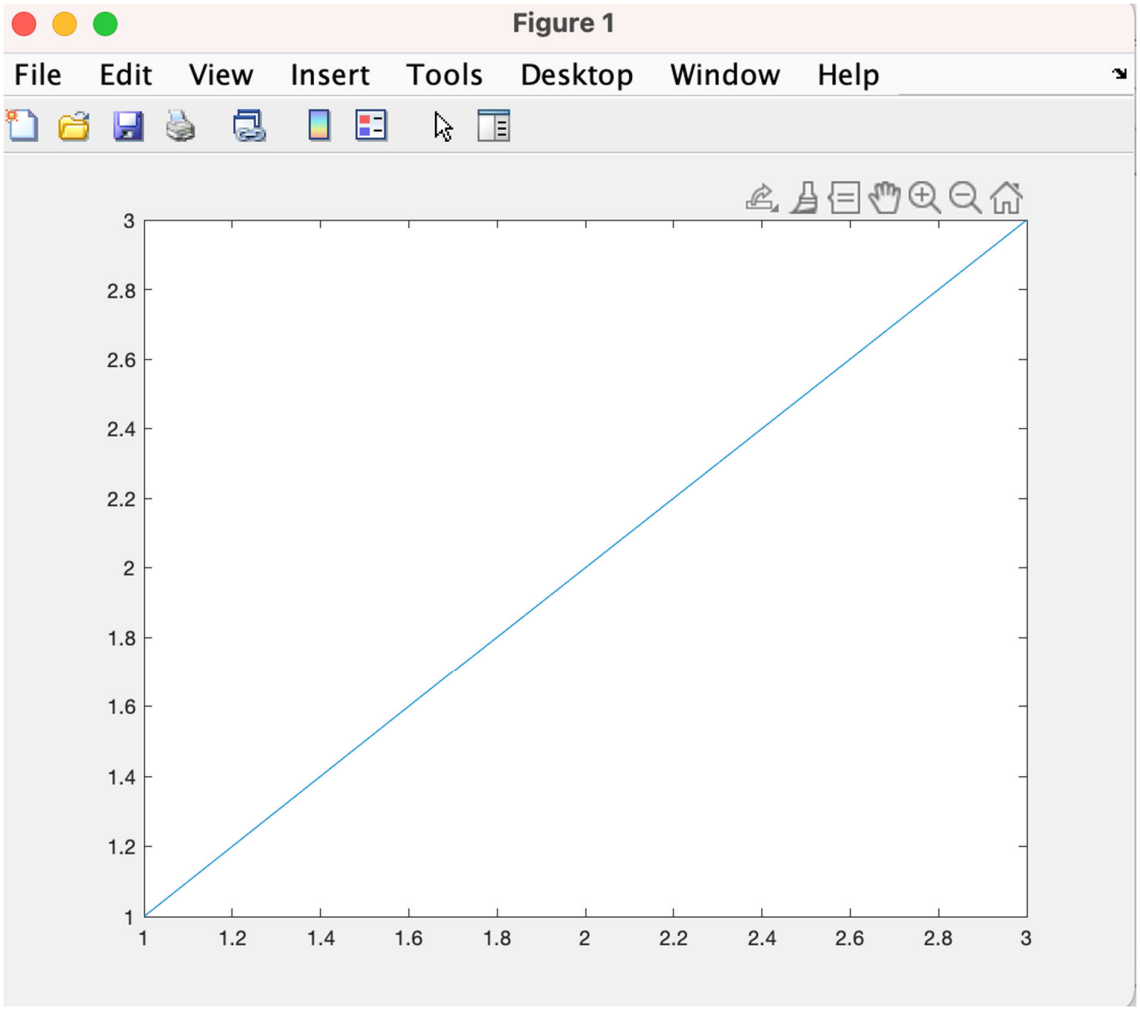


MATLAB Plot Gallery Window

The MATLAB Plot Gallery Window provides you with examples on different types of graphical representations of data.

To reach the Plot Gallery Window, choose the Plot Tab. If you do not have any matrix or vector in the Workspace Window, create one (eg. my_mat = [1, 2, 3]). Choose plot chart to plot the graph.





MATLAB Variables

In every programming platform / language, there is a need to store data temporarily. In programming, data is stored using variables. In MATLAB, there are rules to follow.

Variable naming rules

- Starts with a letter, followed by letters, digits or underscores
- Case sensitive
- Maximum length of the variable can be discovered using command “namelengthmax”
- Cannot be a keyword

An example of creating a variable to store information:
height_in_metres = 1.8

MATLAB Matrix

MATLAB uses matrices frequently and that is the selling point of MATLAB. A single dimensional matrix is known as a vector. Every matrix has rows and columns. The name for a matrix follows the MATLAB variables.

An example of creating a variable to store information:
my_vec = [2, 3, 4]; % creates a vector
my_mat = [1, 2, 3, 4; 5, 6, 7, 8]; % creates a 2 x 4 matrix

What is a Function?

In programming, there will be blocks of code that are repetitive. These blocks of codes are usually organized into functions for readability and reusability. Hence a function, also known as a subroutine, is used to organize code such that:

- A function is a self-contained module
- It takes in inputs and gives an output or return values
- A function should do one thing and do it well.

What are basic mathematical functions in MATLAB?

Basic Mathematical functions in include Logarithms, Exponentials, Rounding and Discrete Mathematics. MATLAB provides support for each of these types with various functions. We shall see that in the following tables.

Basic functions

abs(x)	Return absolute value of x. Absolute value refers to the positive values of x.
	
	abs(-2) will return 2
	abs(2) will return 2
	
	Example:
	x = -5;
	y = abs(x);
	disp(y); % output is 5

exp(x)	Return the value of e to the power x where e is Euler's number which is a constant approximately equal to 2.71828. exp(2) gives 7.3891 exp(3) gives 20.0855 Example: x = 5; y = exp(x); % e^5 fprintf('y is %f\n', y); % output is 148.413159

log(x)	Return the natural logarithm of x. Also known as logarithm to the base of e
	
	log(100) gives 4.6052
	log(exp(1)) gives 1
	
	Example:
	x = 6;
	y = log(x);
	fprintf('y is %f\n', y); % output is 1.791759

log10(x)	Return the logarithm value of x base 10.
	
	log10(100) gives 2
	log10(1000) gives 3
	
	Example:
	x = 10000;
	y = log(x);
	fprintf('y is %f\n', y); % output is 4

nthroot(x, n)	Return the nth root value of x nthroot(9, 2) gives 3 nthroot(256, 4) gives 4 Example: y = nthroot(1024, 10); fprintf('y is %f\n', y); % output is 2.0
power(x, n)	Return the x to the power of n. If n is negative, it is 1 divided by (x to the power of n). power(2, 3) gives 8 power(4, -2) gives 0.0625 Example: y = power(2, 10); fprintf('y is %f\n', y); % output is 1024
rem(x, y)	Return remainder of x divided by y rem(13, 4) gives 1 rem(17, -3) gives 2 rem(-17, 3) gives -2 Example: x = 5;

	y = rem(23, x); fprintf('y is %f\n', y); % output is 3

sign(x)	Y = sign(x) returns an array Y the same size as x, where each element of Y is:
	
	1 if the corresponding element of x > 0.
	0 if the corresponding element of x == 0.
	-1 if the corresponding element of x < 0.
	
	sign(-18) gives -1
	sign([4, 0, 3, -7]) gives [1 0 1 -1]
	
	Example:
	mat = [-3, 0, 4, 2; 2, -4, 0 7];
	all_signs = sign(mat);
	disp(all_signs);

sqrt(x)	Return the value of square of x. If x is negative, 0 is returned.
	
	sqrt(9) gives 3
	sqrt(121) gives 11
	
	Example:
	x = 169;
	y = sqrt(x);
	fprintf('y is %f\n', y);

sum(x)	Return the sum of all values in a vector or along the first array dimension of a matrix. sum([1, 4, 5]) gives 10 sum([1, 2, 3; 11, 13, 15]) gives 12 15 18

Rounding functions

ceil(x)	Return the value of x rounded toward +ve infinity ceil(2.1) gives 3 ceil(2.9) gives 3 ceil(-2.1) gives -2 ceil(-2.9) gives -2 Example: x = -5.9;

	y = ceil(x); fprintf('y is %f\n', y); % output is -5

fix(x)	Return the value of x rounded toward zero
	
	fix(2.1) gives 2
	fix(2.9) gives 2
	fix(-2.1) gives -2
	fix(-2.9) gives -2
	
	Example:
	x = 5.8;
	y = fix(x);
	fprintf('y is %f\n', y); % output is 5

floor(x)	Return the value of x rounded toward -ve infinity
	
	floor(2.1) gives 2
	floor(2.9) gives 2
	floor(-2.1) gives -3
	floor(-2.9) gives -3
	
	Example:
	x = -5.9;
	y = floor(x);
	fprintf('y is %f\n', y); % output is -6

round(x)	Return the value of x rounded to nearest integer
	
	round(2.1) gives 2
	round(2.9) gives 3
	round(-2.1) gives -2
	round(-2.9) gives -3
	
	Example:
	x = 5.8;
	y = round(x);
	fprintf('y is %f\n', y); % output is 6

Discrete Mathematics

factor(x)	Return prime factors of x. Do note that it DOES NOT return all factors of x. factor(20) will return 2, 2, 5 factor(32) will return 2, 2, 2, 2, 2

	Example:
	x = 30;
	y = factor(x);
	disp(y); % output is 2, 3, 5
	
	% note that other factors like 6, 10, 15
	% are not displayed

factorial(x)	Return the value of factorial of x. factorial(3) gives 6 factorial(6) gives 720 Example: x = 4; y = factorial(x); % 1 * 2 * 3 * 4 fprintf('y is %f\n', y); % output is 24

gcd(x, y)	Find the greatest common denominator of x and y
	
	gcd(20, 30) gives 10
	gcd(48, 80) gives 16
	
	Example:
	x = 24;
	y = 32;
	z = gcd(x, y);
	fprintf('z is %f\n', z); % output is 8

lcm(x, y)	Find the least common multiple of x and y
	
	lcm(20, 30) gives 60
	lcm(48, 80) gives 240
	
	Example:
	x = 24;
	y = 32;
	z = lcm(x, y);
	fprintf('z is %f\n', z); % output is 96

isprime(x)	Return 1 if x is a prime number, 0 otherwise. Note that x must be a nonnegative number. isprime(17) gives 1 isprime(20) gives 0 Example: x = [17, 12, 5];

	y = isprime(x); disp(y); % output is 1 0 1
primes(x)	Return all prime numbers less than or equal to x primes(20) gives 2, 3, 5, 7, 11, 13, 17, 19

perms(x)	Return all permutations of vector x.
	
	Example:
	vec = [1 3 5];
	permutations = perms(vec);
	disp(permutations);
	
	Output:
	5 3 1
	5 1 3
	3 5 1
	3 1 5
	1 5 3
	1 3 5

rats(x)	Return the fraction representation of x
	
	rats(2.25) gives 9/4
	rats(1.333333) gives 4/3
	
	Example:
	mat = [2.333333, 1.51515151, 3.75];
	fractions = rats(mat);
	disp(fractions);

Which trigonometric functions are supported in MATLAB?

MATLAB supports a number of trigonometric functions in mathematics. Basic trigonometric functions such as sine, cosine, tangent are supported and their inverse functions as well hyperbolic functions. We would not cover all the functions but just 3 of them.

Basic Trigonometric Functions

sin(x)	Return the sine value of x (x in radians). x can be a matrix sin(pi) will return 0 Example: x = pi/2;

	y = sin(x); disp(y); % output is 1

cos(x)	Return the cosine value of x (x in radians). x can be a matrix
	
	cos(pi) will return -1
	
	Example:
	x = pi/2;
	y = cos(x);
	disp(y); % output is close to 0

tan(x)	Return the tangent value of x (x in radians). x can be a matrix
	
	tan(pi/4) will return 1
	
	Example:
	x = pi/8;
	y = tan(x);
	disp(y); % output is 0.4142

Other functions supported include:

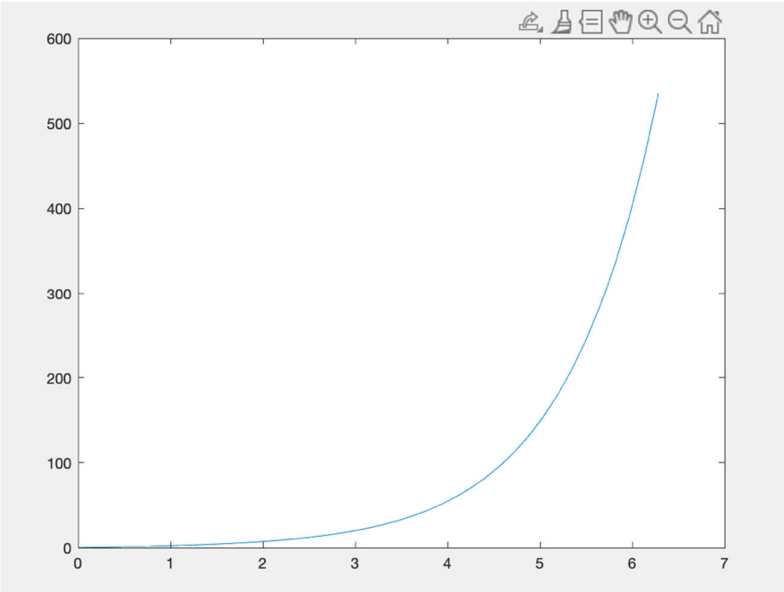
- Inverse functions (asin, acos, atan, atan2)
- Hyperbolic functions (sinh, cosh, tanh)
- Inverse Hyperbolic functions (asinh, acosh, atanh)

Example Problem

Plot a chart of hyperbolic sine function y = 2sinh(x) with x = 0 to x = 2 pi radians.

Suggested Solution:

```
x = 0: pi/20 : 2*pi;
y = 2 * sinh(x);
plot(x,y);
```



Explanation:

- Firstly, x is declared over a range of 0 to 2 pi radians in steps of 2.
- Next sinh() is applied on x and multiplied by 2
- Lastly, the plot() function takes x and y vectors and plot the chart accordingly.

MATLAB Statistical and Data Analysis Functions

MATLAB offers a rich library to deal with statistics and data analysis. The list of functions can be seen in the following tables. The argument passed in can either be vectors or matrices. The following tables will show basic functions for statistics, sorting and getting the sizes of the matrices.

MATLAB offers a rich library to deal with statistics and data analysis. The list of functions can be seen in the following tables. The argument passed in can either be vectors or matrices. The following tables will show basic functions for statistics, sorting and getting the sizes of the matrices.

Basic Statistical Functions

max(x)	Return biggest value in each column in matrix x. When 2 values are returned, the first value contains the biggest values for each column, the second value contains the positions of the biggest values for each column max([2, 3, 5, 4]) gives 5 max([1, 3, 5; 6, 4, 2]) gives 6 4 5 **Example:** max([1, 3, 5; 6, 4, 2]); [val, pos] = max([1, 3, 5; 6, 4, 2]);

	disp(val); % 6 4 5 disp(pos); % 2 2 1
min(x)	Return smallest value in each column in matrix x. When 2 values are returned, the first value contains the smallest values for each column, the second value contains the positions of the smallest values for each column min([2, 3, 5, 4]) gives 2 min([1, 3, 5; 6, 4, 2]) gives 1 3 2 **Example:** min([1, 3, 5; 6, 4, 2]); [val, pos] = min([1, 3, 5; 6, 4, 2]); disp(val); % 1 3 2 disp(pos); % 1 1 2

mean(x)	Return average value in each column in matrix x
	
	mean([2, 3, 5, 4]) gives 3.5
	mean([3, 1, 5; 6, 4, 2]) gives 4.5000 2.5000 3.5000
	
	Example:
	mat = [21, 18, 25; 23, 20, 35];
	averages = mean(mat);
	disp(averages); % 22 19 30

median(x)	Return middle value in each column in matrix x if number of elements is odd or average of 2 middle values if number of elements is even
	
	median([2, 3, 1, 5]) gives 2.5
	median([3, 1, 5; 6, 4, 2; 3, 4, 1]) gives 3 4 2
	
	Example:
	my_mat = [12, 30; 14, 20; 18, 25; 20, 23];
	medians = median(my_mat);
	disp(medians); % 16 24

std(x)	Return standard deviation value in each column in matrix x std([2, 3, 5, 4]) gives 1.2910 std([6, 5, 4; 1, 2, 3]) gives 3.5355 2.1213 0.7071 **Example:** my_mat = [12, 30; 14, 20; 18, 25; 20, 23];

	standard_deviations = std(my_mat); disp(standard_deviations); % 3.6515 4.2032

var(x)	Return variance value in each column in matrix x
	
	var([2, 3, 5, 4]) gives 1.6667
	var([6, 5, 4; 1, 2, 3]) gives 12.5000 4.5000 0.5000
	
	Example:
	my_mat = [12, 30; 14, 20; 18, 25; 20, 23];
	variances = var(my_mat);
	disp(variances); % 13.3333 17.6667

Sorting Functions

sort(x)	Sort the elements of a matrix x in ascending order. sort([2, 1, 5, 4]) gives 1 2 4 5 sort([6, 2, 5; 1, 3, 4]) gives 1 2 4 6 3 5
sortrows(x)	Sort the rows of matrix x sortrows([6, 2, 5; 1, 3, 4]) gives 1 3 4 6 2 5

Size Functions

size(x)	Return the dimensions of matrix x (rows and columns)
	
	size([2, 1, 5, 4]) gives 1 4
	size([6, 2, 5; 1, 3, 4]) gives 2 3
	
	Example:
	my_mat = [12, 30; 14, 20; 18, 25; 20, 23];
	[rows, columns] = size(my_mat);
	fprintf('Rows: %d, Columns: %d\n', rows, columns);
	% Rows: 4, Columns: 2

length(x)	Return the largest dimension of matrix x (the larger of rows and columns) length([1, 2, 3, 4, 5]) gives 5 length([6, 2, 5; 1, 3, 4]) gives 3

Generating Uniform and Gaussian Random Matrices

To aid in simulation, MATLAB has the ability to generate random values for matrices through 2 methods namely Uniform Random Numbers and Gaussian Random Numbers. The difference between the 2 types of generation is that Gaussian Random Numbers follow a Normal Distribution.

Uniform Random Numbers

rand(n)	Return n by n matrix of values between 0 and 1 rand(2) may give 0.2920 0.0155 0.4317 0.9841 rand(3) may give 0.8147 0.9134 0.2785 0.9058 0.6324 0.5469 0.1270 0.0975 0.9575
rand(m, n)	Return m by n matrix of values between 0 and 1 rand(2, 3) gives 0.9203 0.7379 0.4228 0.0527 0.2691 0.5479
randi(n)	Return an array containing integer values drawn from the discrete uniform distribution starting from 1 to n randi(3) may give 1, 2 or 3 randi(10) may give a number from 1 to 10
randi([x y], m, n)	Return a matrix of dimensions m by n containing integer values drawn from the discrete uniform distribution starting from x to y randi([2 9], 2, 3) may give 7 9 8 3 4 3

Gaussian Random Numbers

randn(n)	Return a n by n matrix of each value being Gaussian (or normal) random number with a mean of 0 and a variance of 1. randi(3) may give 0.1202 -0.9870 -0.6039

	0.5712 0.7596 0.1769 0.4128 -0.6572 -0.3075
randn(m, n)	Return a m by n matrix of each value being a Gaussian (or normal) random number with a mean of 0 and a variance of 1. randn(2, 3) may give -0.1318 1.0468 0.3277 0.5954 -0.1980 -0.2383

What are the computational limits of MARLA

Every programming language or system has a computational limit. MATLAB has its own computational limits such as the largest or smallest integers or real numbers. These are very useful numbers if you want to get the biggest number in the MATLAB universe. To get the values of these numbers simply enter the commands:

- realmax
  - largest float, 1.7977e+308
- realmin
  - smallest float, 2.2251e-308
- intmax
  - largest integer, 2147483647
- intmin
  - smallest integer, -2147483648

What are special values or functions in MATLAB?

Most systems have pre-defined values. You have seen pi which is a pre-defined value. The same pi is used as a pre-defined value in other languages like Java or Python. MATLAB has its own pre-defined values which are useful to the user.

Please note that pre-defined values and functions in MATLAB should not be replaced by the user's own variables. Replacing them may have unintended consequences. Let us look at the table below for MATLAB's pre-defined values.

Predefined values

pi	Mathematical constant Pi of value 3.1416
i, j	Imaginary numbers
Inf	Infinity, occurs when there is an error such as number divided by zero
NaN	Not a number, occurs when the result of a calculation is undefined
clock	Current time in array format
date	Current date in string format
eps	The distance between 1 and next larger double precision floating point number

Quick summary

MATLAB is a programming platform specially designed for engineers and scientists for various purposes. The reasons behind its popularity with engineers and scientists are its simplicity of usage and focus on engineering needs.

MATLAB has lots of libraries that allow one to perform data analytics, simulations, machine learning, robotics and more. Its environment is simple to use and has intuitive windows for various purposes such as the Command Window, the Workspace Window and the Editor.

MATLAB is a programming platform specially designed for engineers and scientists and has various mathematical functions and functions that deal with statistics and data analysis to make engineering and machine learning easier.

MATLAB is also capable of generating matrices with random values to perform simulations which are important. Like most programming languages, there are useful predefined values in MATLAB.

References List

Stallings, W. (2016). Computer organization and architecture: Designing for performance. Hoboken, N.J: Pearson.

Stallings, W. (2018). Operating Systems: Internals and Design Principles. 9th edition: Pearson.

MathWorks (n.d) MATLAB https://www.mathworks.com/products/matlab.html

Topic 09 – MATLAB Plotting

Learning Objectives

1. Describe the use of Plotting in MATLAB
2. Describe how to generate XY plot
3. Understand and generate different types of plots
4. Create 3D Plots
5. Use Sub plots

Guiding Questions

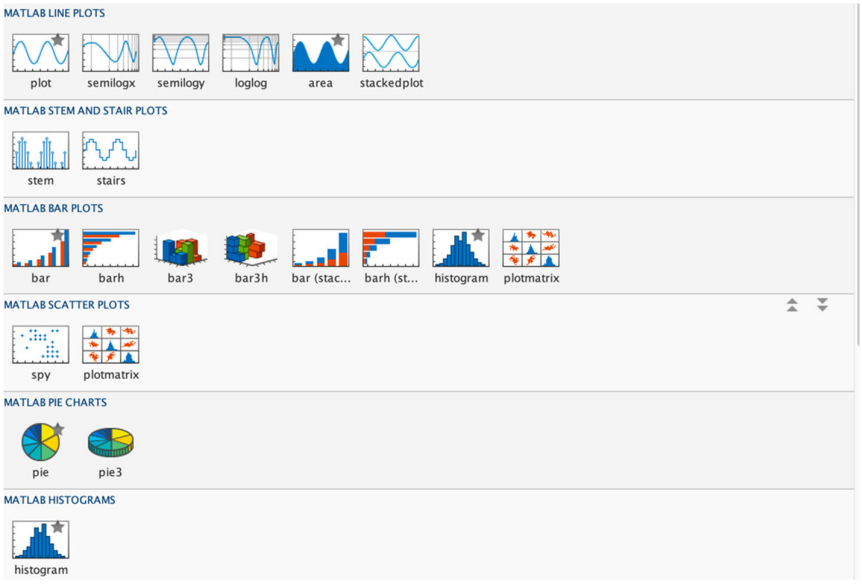
1. Why and how do we perform plotting in MATLAB?

Why use MATLAB Chart Plots

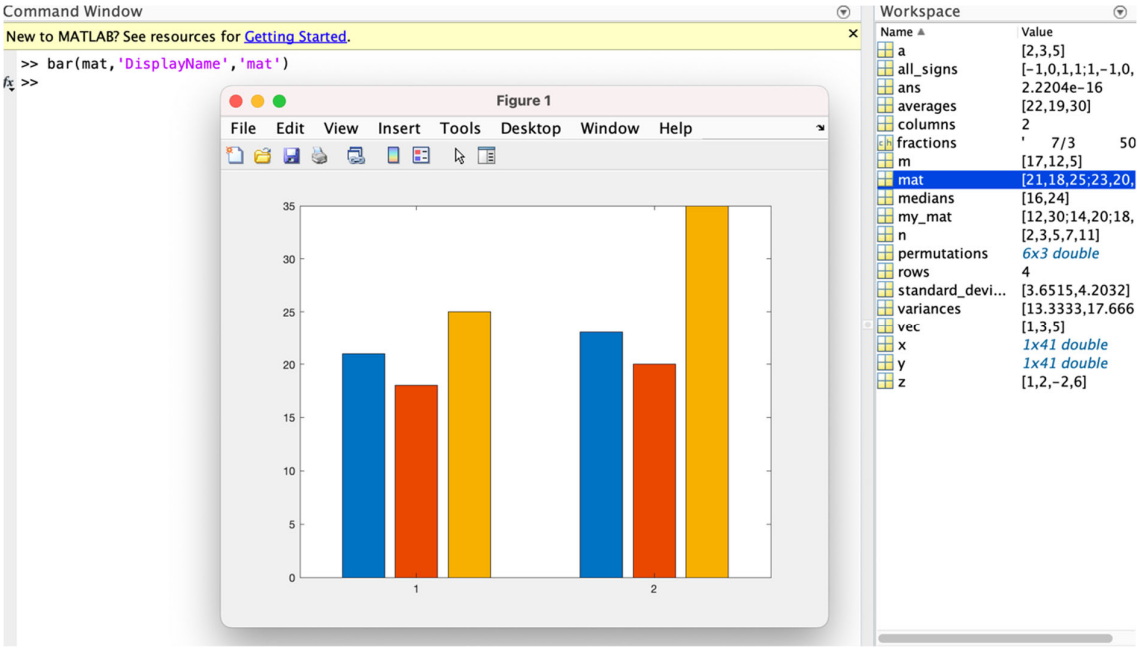
MATLAB has a wide selection of charts to select from to display data graphically. In this chapter we will be looking at some of the charts such as:

- Line 2D Plot
- Histogram
- Pie Chart
- Scatter Plot
- Sub Plots
- Line 3D Plot

To start off a simple plot without code, you can select the Plots Tab and then select a matrix in the Workspace Window, followed by a chart to plot.



In this quick example, the matrix “t” is selected, followed by selecting the bar chart. The output is as follows… without writing code!



You can see how convenient it is to use MATLAB to plot charts without having to write much code. In the following example, we will look at how to plot the graph using a simple Line 2D plot as our basic chart plot.

Using Line 2D Plot in MATLAB

The simplest and most commonly used chart plot is the Line 2D Plot in MATLAB. The selection of x and y axis usually uses the following as a guide:

- x-axis which is the horizontal axis is an independent variable such as time
- y-axis which is the vertical axis is a dependent variable

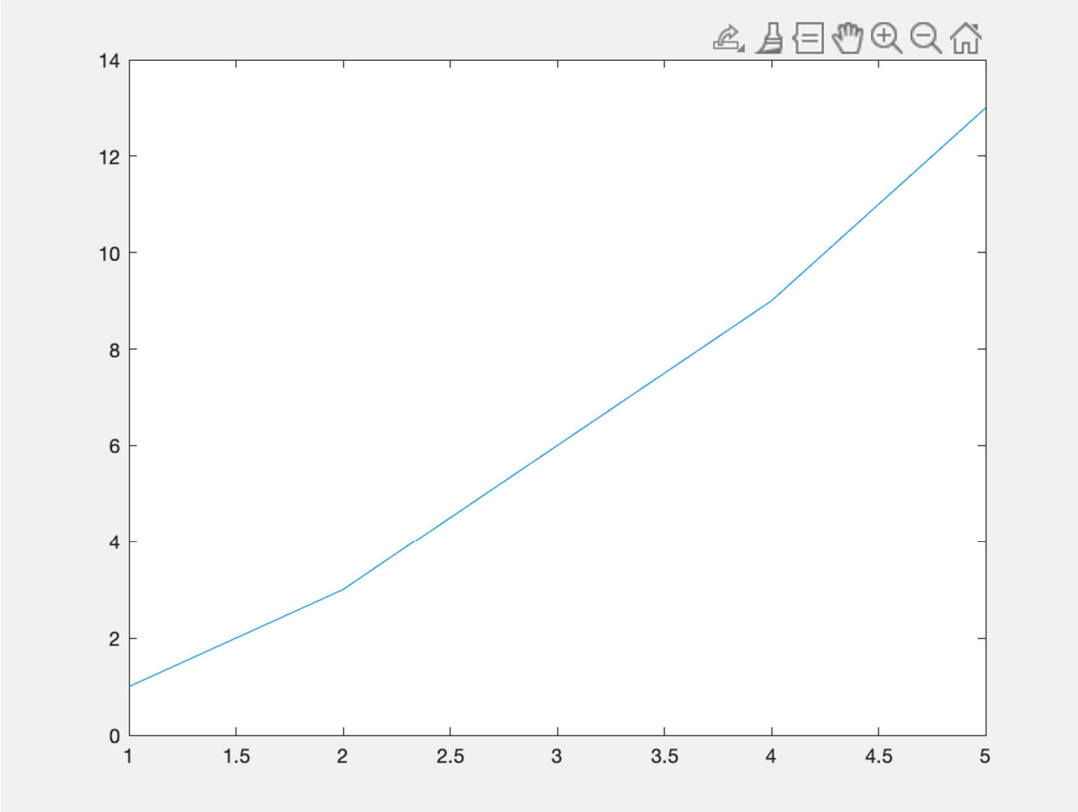
After determining the x-axis and y-axis, we can proceed to use the plot() function in MATLAB to create the chart.

Example 1:

Plot a simple chart with only 1 set of values (a single vector)

Code:

```
y = [1, 3, 6, 9, 13];
plot(y);
```



Explanation:

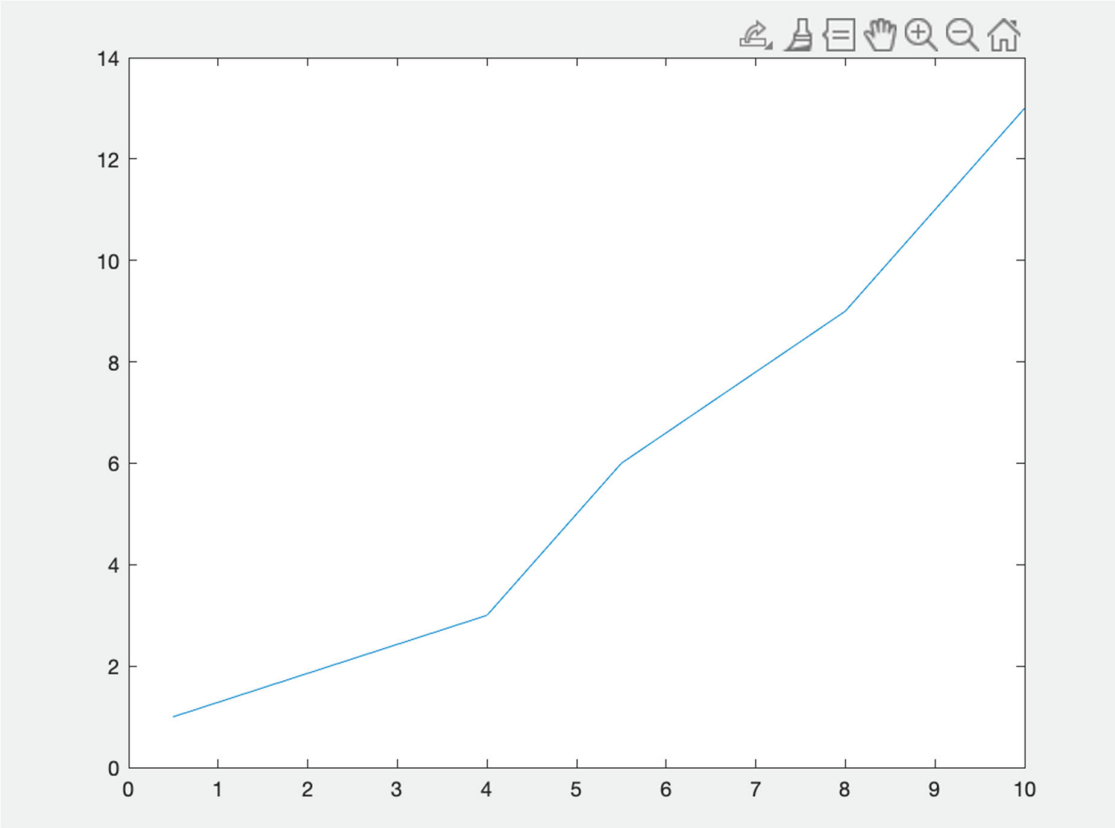
With a single vector, MATLAB automatically provides default values for the x-axis starting from 1, so in the coordinates plotted will be (1, 1), (2, 3), (3, 6), (4, 9) and (5, 13)

Example 2:

Plot a simple chart with only 2 set of values (2 vectors of equal size)

Code:

```
x = [0.5, 4, 5.5, 8, 10];
y = [1, 3, 6, 9, 13];
plot(x, y);
```



Explanation:

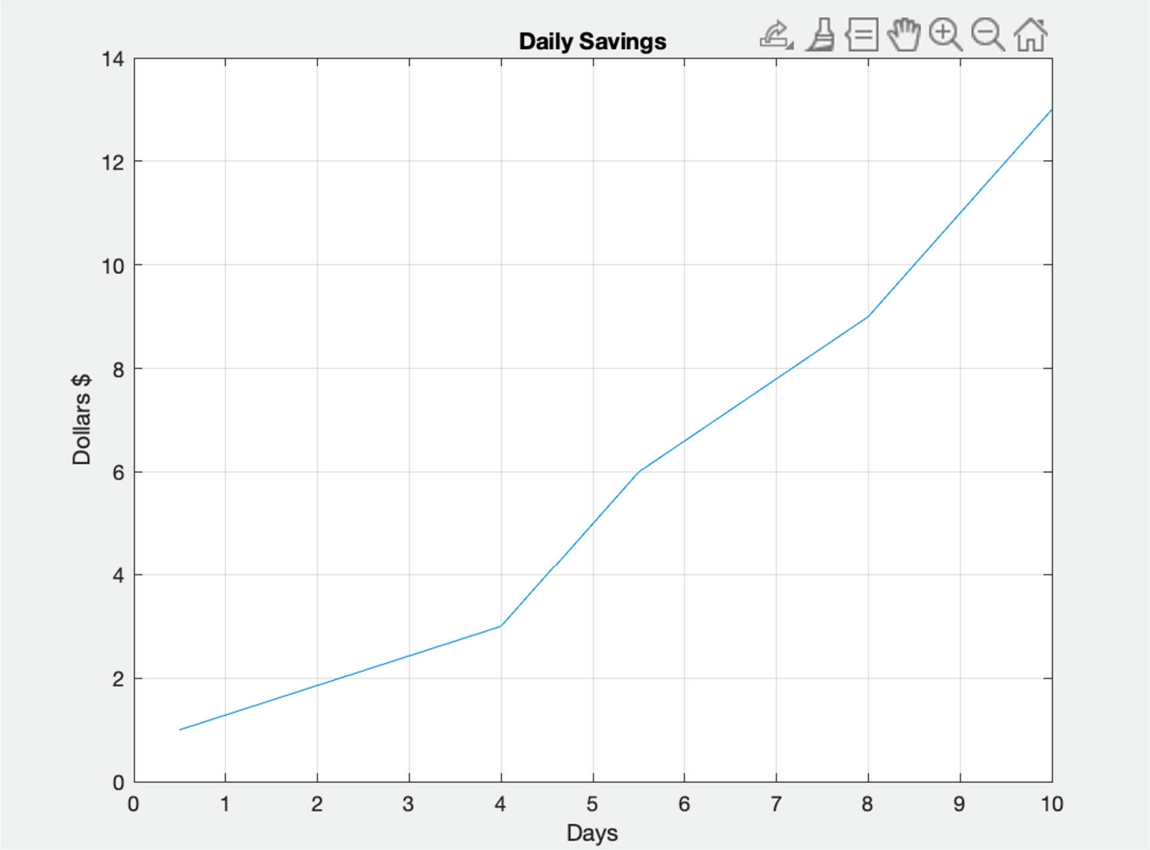
With 2 vectors, MATLAB will check whether the vectors are of equal size before plotting the corresponding values, so in the coordinates plotted will be (0.5, 1), (4, 3), (5.5, 6), (8, 9) and (10, 13). Do note that the vectors must be equal size or there will be an error.

Example 3:

Add title, labels for x and y axes and show the grid for the chart.

Code:

```
x = [0.5, 4, 5.5, 8, 10];
y = [1, 3, 6, 9, 13];
plot(x, y);

grid on;
title('Daily Savings');
xlabel('Days');
ylabel('Dollars $');
```



Note:

You can only add a grid, title, xlabel or ylabel after the chart is plotted.

Example 4:

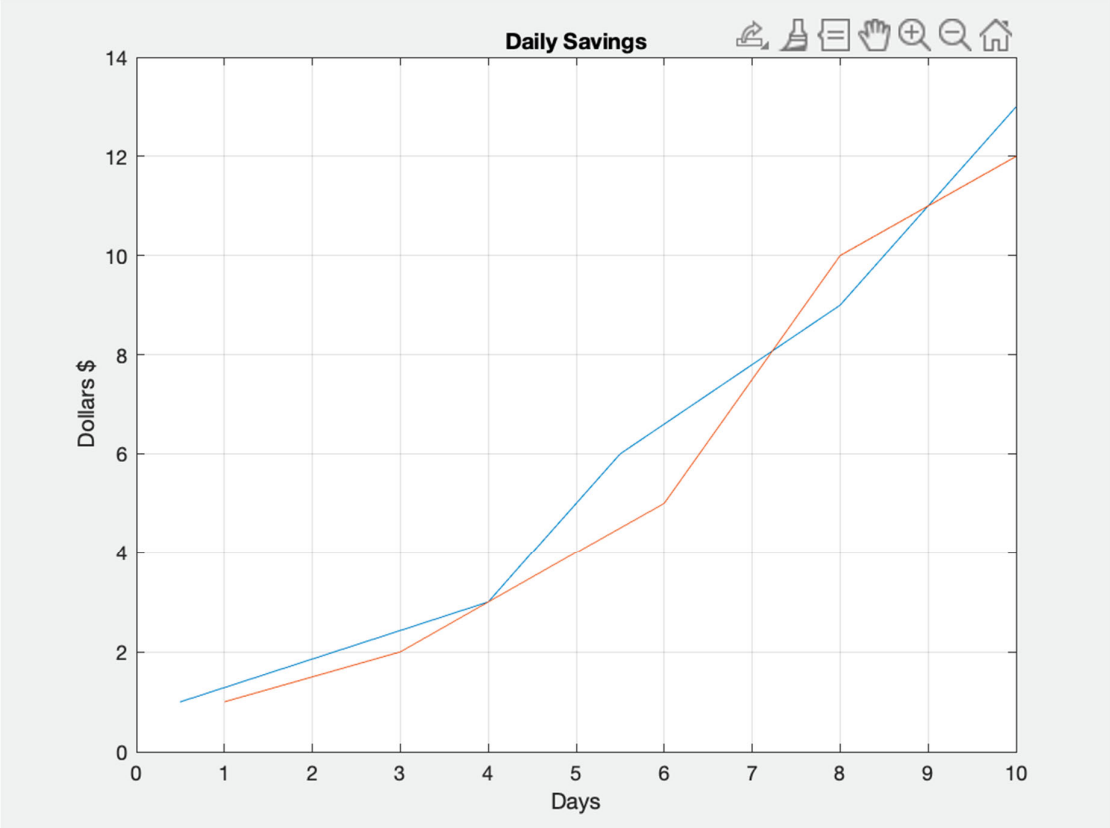
Add another graph to be plotted on the same scale.

```
Code                                    Sample                                  1:
x           =       [0.5,       4,      5.5,        8,          10];
y           =       [1,         3,      6,          9,          13];
plot(x,                                                                         y);

hold                                                                            on;
x2          =       [1,         3,      6,          8,          10];
y2          =       [1,         2,      5,          10,         12];

grid                                                                            on;
title('Daily                                                                    Savings');
xlabel('Days');
ylabel('Dollars                                                                 $');
```

OR

Code Sample 2:

```
x = [0.5, 4, 5.5, 8, 10];
y = [1, 3, 6, 9, 13];

x2 = [1, 3, 6, 8, 10];
y2 = [1, 2, 5, 10, 12];

plot(x, y, x2, y2);

grid on;
title('Daily Savings');
xlabel('Days');
ylabel('Dollars $');
```



Explanation:

In the first method, you will need the line “hold on” to wait for another graph to be plotted in order to show both charts on the same scale. Otherwise, only the most recent chart plotted will be shown.

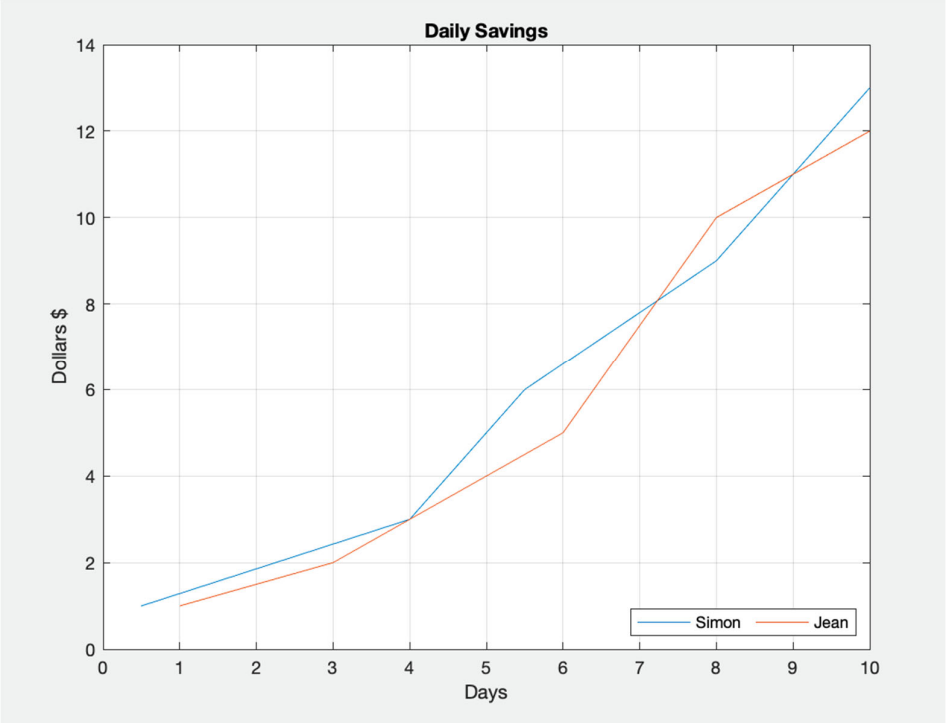
In the second method, you do not need to have the “hold on” code. Both lines of code will produce the same output as seen in the above chart.

Example 5:

Add a legend to the plot to show the legend in the bottom in a horizontal manner.

Code:

```
x = [0.5, 4, 5.5, 8, 10];
y = [1, 3, 6, 9, 13];

x2 = [1, 3, 6, 8, 10];
y2 = [1, 2, 5, 10, 12];

plot(x, y, x2, y2);

grid on;
title('Daily Savings');
xlabel('Days');
ylabel('Dollars $');

legend({'Simon', 'Jean'}, 'Location', 'southeast', 'NumColumns', 2);
```



Explanation:

legend() method allows you to specify the location of the legend (using 'Location') and whether to lay the items in a vertical orientation or horizontal orientation through the use of the property 'NumColumns'

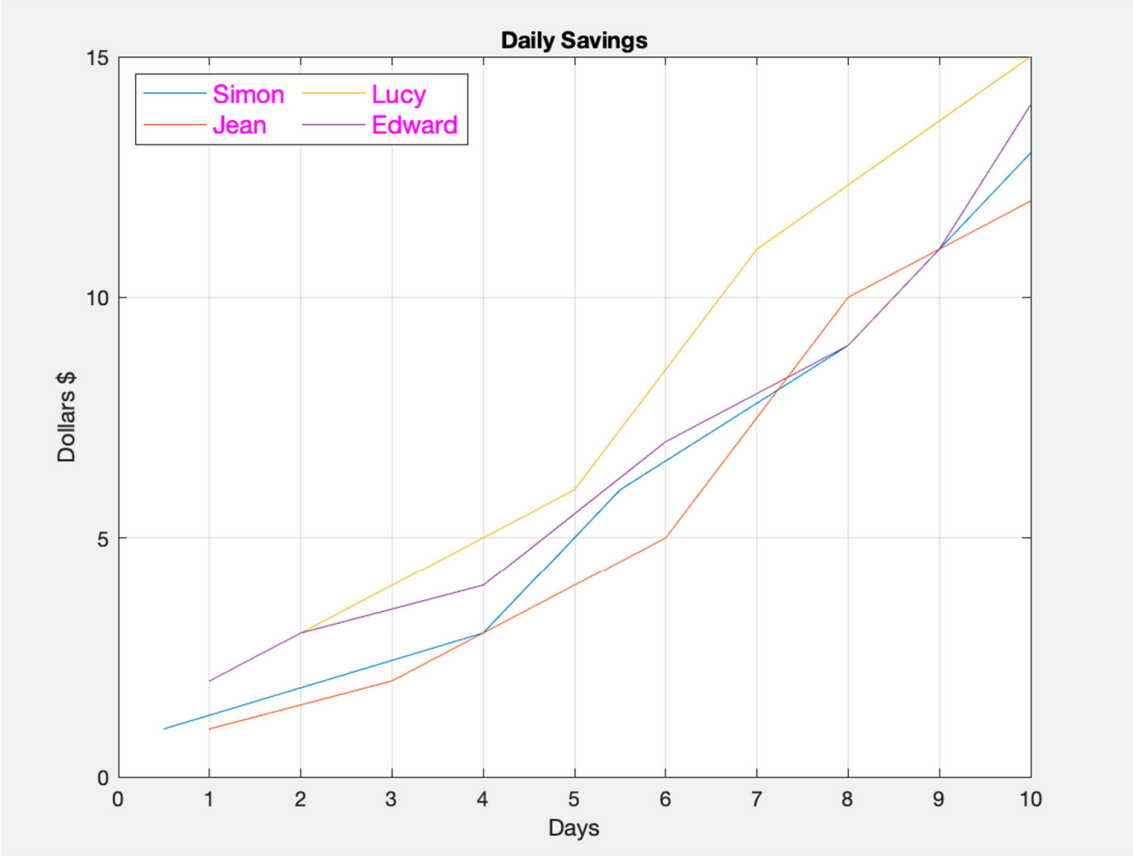
Example 6:

Now let us add 2 more graphs, and set the location, text colour, font size, columns of the legend using the legend object.

Code:

```
x = [0.5, 4, 5.5, 8, 10];
y = [1, 3, 6, 9, 13];

x2 = [1, 3, 6, 8, 10];
y2 = [1, 2, 5, 10, 12];

x3 = [1, 2, 5, 7, 10];
y3 = [2, 3, 6, 11, 15];

x4 = [1, 2, 4, 6, 8, 9, 10];
y4 = [2, 3, 4, 7, 9, 11, 14];

plot(x, y, x2, y2, x3, y3, x4, y4);
```

```
% grid on, title, labels for x and y axes
grid on;
title('Daily Savings');
xlabel('Days');
ylabel('Dollars $');

% lgd is now the legend object
lgd = legend('Simon', 'Jean', 'Lucy', 'Edward');
lgd.Location = 'northwest';
lgd.NumColumns = 2;
lgd.FontSize = 12;
lgd.TextColor = 'magenta';
```



Explanation:

lgd is the legend object we assigned to when we used the legend() function. Now you can easily manipulate the properties of the legend object simply by using the dot '.' operator. You can see the property FontSize can be accessed and modified using

lgd.FontSize = 12;

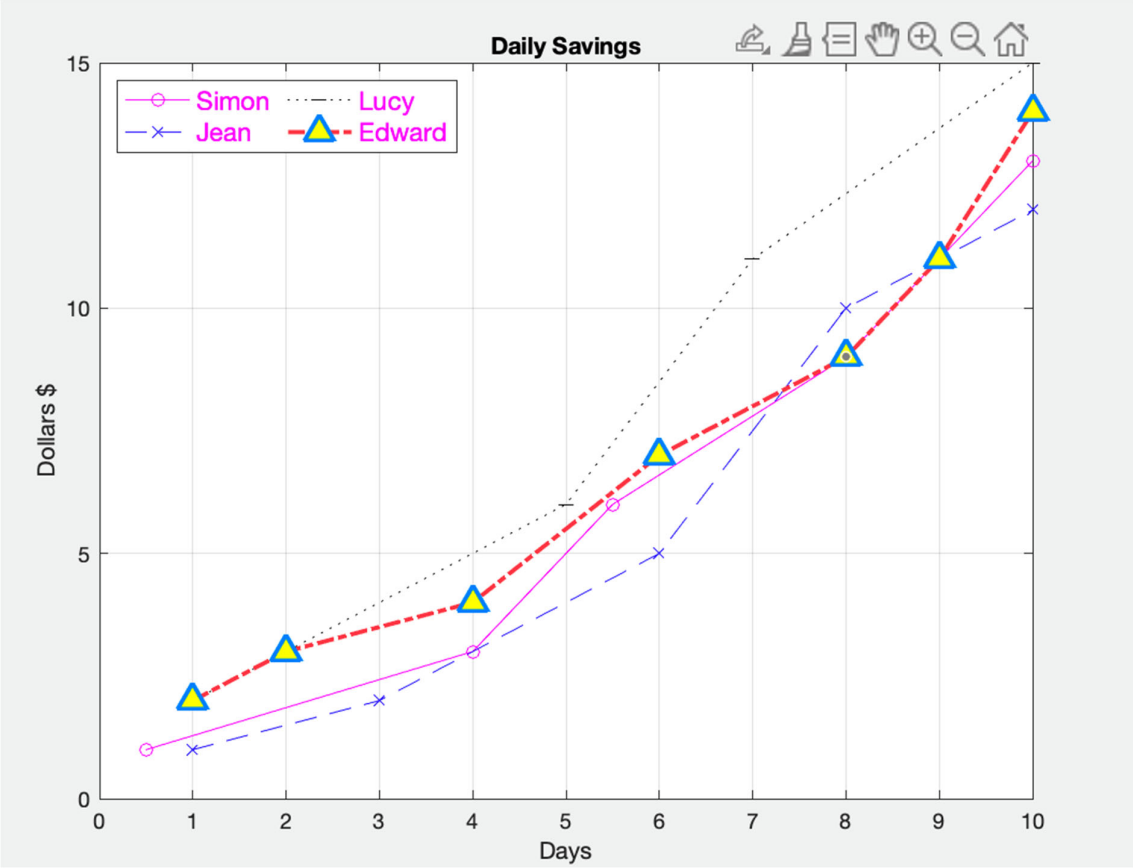
There are other properties you can change as well, such as the background colour of the legend, whether the font should be bold or italic.

Example 7:

In this example, we will change the colours, markers, line appearances of the line graphs.

Code:

```
x = [0.5, 4, 5.5, 8, 10];
y = [1, 3, 6, 9, 13];
plot(x, y, '-mo');
hold on;

x2 = [1, 3, 6, 8, 10];
y2 = [1, 2, 5, 10, 12];
plot(x2, y2, '--bx');
hold on;

x3 = [1, 2, 5, 7, 10];
y3 = [2, 3, 6, 11, 15];
plot(x3, y3, ':_k');
hold on;

x4 = [1, 2, 4, 6, 8, 9, 10];
y4 = [2, 3, 4, 7, 9, 11, 14];
p4 = plot(x4, y4, '-.^');
p4.Color = [1 0.3 .3];
p4.LineWidth = 2;
p4.MarkerSize = 12;
p4.MarkerFaceColor = [ 1 1 0 ];
p4.MarkerEdgeColor = [ 0 0.5 1 ];

grid on;
title('Daily Savings');
xlabel('Days');
ylabel('Dollars $');

lgd = legend('Simon', 'Jean', 'Lucy', 'Edward');
lgd.Location = 'northwest';
lgd.NumColumns = 2;
lgd.FontSize = 12;
lgd.TextColor = 'magenta';
```



Explanation:

The code is refactored to plot individually with “hold on” instead of plotting altogether on a single line. The two reasons for doing so are that the code is now easier to read and secondly, we use the properties of plot 4 to customize the graph.

The first 3 plots use the in-built properties. If you wish to customize the graph, look at plot 4.

Plot 4 uses the same principles as the object seen in Example 6 where the legend object is lgd. You can customize the marker face colour, marker edge colour, line width and marker size.

Refer to the table for Line 2D plot choices below (from the link: https://www.mathworks.com/help/matlab/ref/linespec.html)

Line 2D Plot Choices

Line Style	Description
-	Solid line
--	Dashed line
:	Dotted line
-.	Dash-dot line

Color	Description
y	yellow
m	magenta
c	cyan
r	red
g	green
b	blue
w	white
k	black

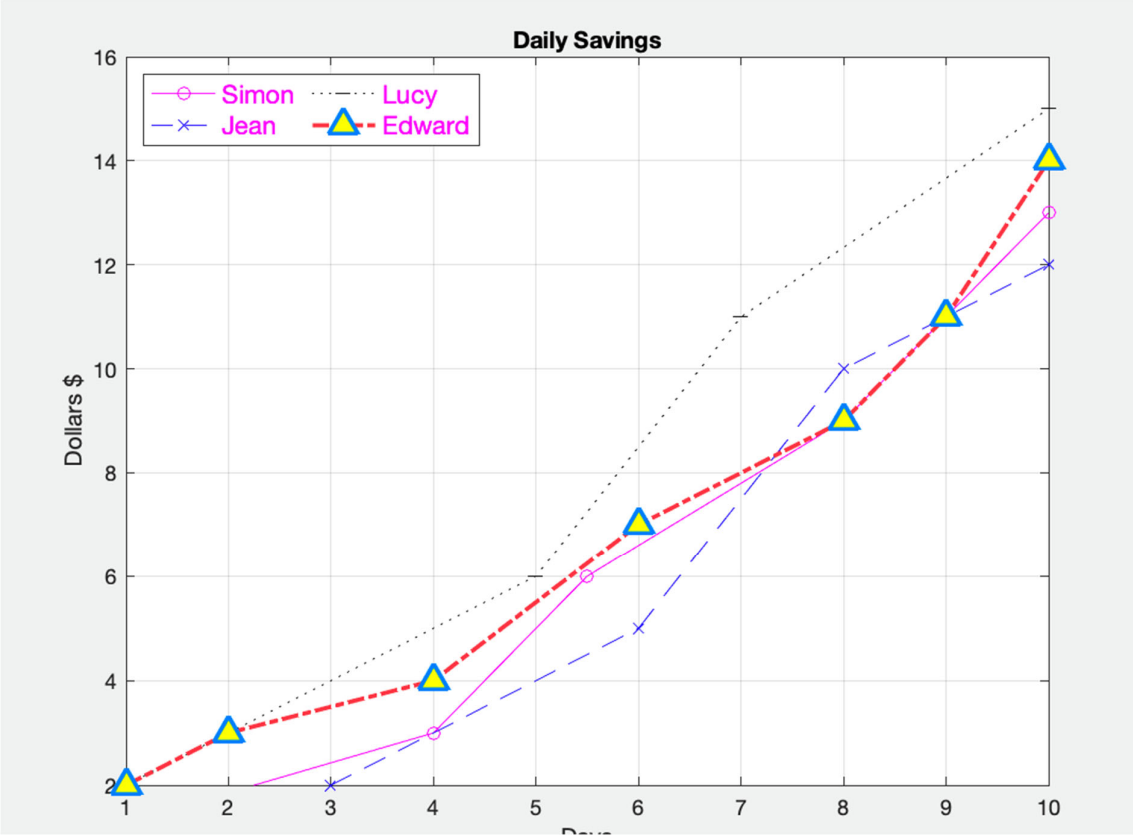
Marker	Description
'o'	Circle
'+'	Plus sign
'*'	Asterisk
'.'	Point
'x'	Cross
'_'	Horizontal line
'|'	Vertical line
's'	Square
'd'	Diamond
'^'	Upward-pointing triangle
'v'	Downward-pointing triangle
'>'	Right-pointing triangle
'<'	Left-pointing triangle
'p'	Pentagram
'h'	Hexagram

Example 8:

In this example, we can adjust the axis with the axis() function.

Code:

```
% code as in Example 7
axis([1 10 2 16]);
```



Explanation:

The axis() function allows us to adjust the values for x and y axes. In this example, the x-axis is adjusted to start from 1 to 10. The y-axis is adjusted to start from 2 to 16.

What is MATLAB subplot?

MATLAB supports multiple plots and uses subplot() to achieve this need to display multiple charts together in a single screen on separate windows. The code to be used is

subplot(m, n, p)

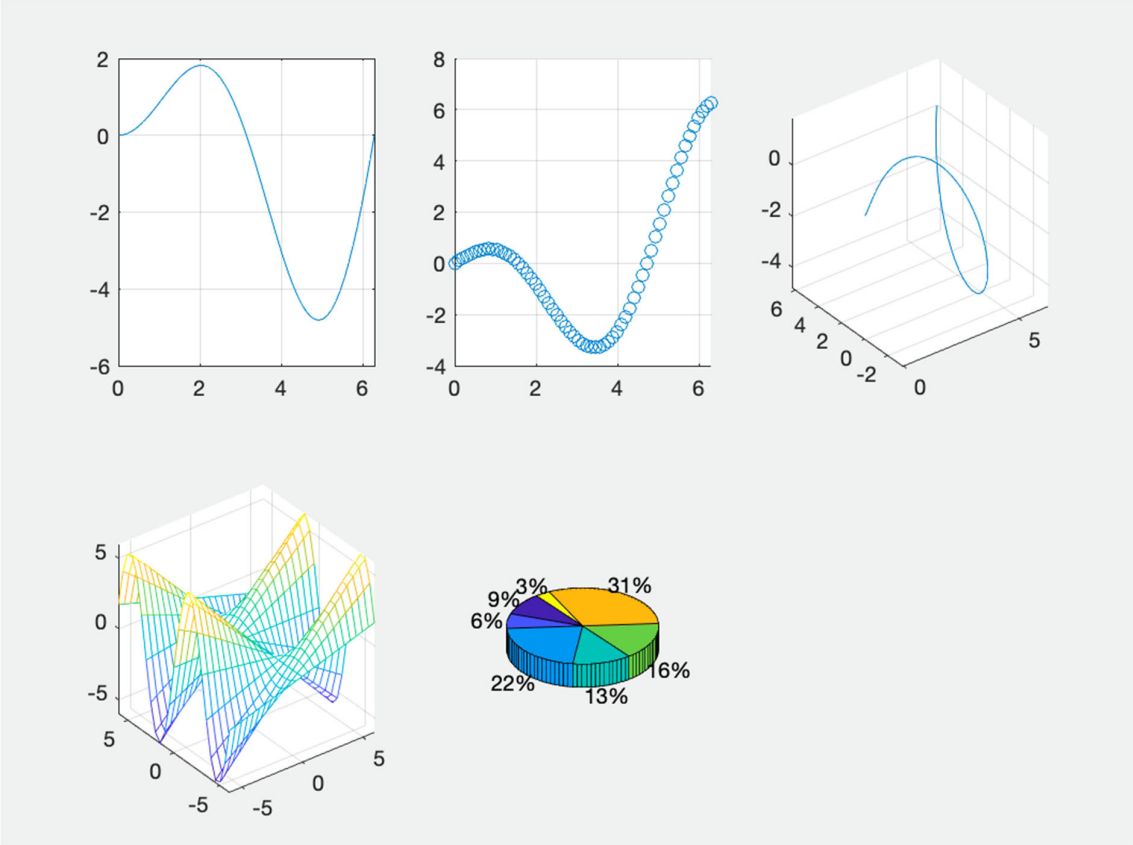
where
m is the number of rows,
n is the number of columns,
p is the position of the plot

Example:

Plot a variety of charts using different chart types.

Code:

```
x = 0:pi/30:2*pi;
y = x .* sin(x);
z = x .* cos(x);

sales = [3, 2, 7, 4, 5, 10, 1];
[X,Y] = meshgrid(-6:.5:6);
Z = X .* sin(Y);

subplot(2, 3, 1);
plot(x, y);
grid on;

subplot(2, 3, 2);
scatter(x, z);
grid on;

subplot(2, 3, 3);
plot3(x, z, y);
grid on;

subplot(2, 3, 4);
mesh(X, Y, Z);

subplot(2, 3, 5);
pie3(sales);
```



Explanation:

- 5 charts are plotted using subplot(2, 3, p) where number of rows is 2 and number of columns is 3 and p is the position.
    - Eg. subplot(2, 3, 5) puts the pie chart into position 5
- The positions are calculated row by row, from left to right. 2 by 3 means there are maximum 6 positions with the first row having positions 1, 2 and 3 and second row having positions 4, 5, 6.
- Each plot uses a different type of charts
    - Position 1: Line 2D
    - Position 2: Scatter chart
    - Position 3: Line 3D
    - Position 4: Mesh plot
    - Position 5: Pie 3D

Quick summary

MATLAB has graph plot functions to allow creating of graphical data representation to allow the users to visualize better. Graphs are useful for trend spotting (line charts) and groupings (scatter plots).

References List

MathWorks (n.d) MATLAB https://www.mathworks.com/products/matlab.html

Topic 10 – MATLAB Programming

Learning Outcomes

1. Describe the use of control structures
2. Describe the use of sequence statements
3. Describe how to use selection statements
4. Describe how to use repetition statements

Guiding Questions

1. What are control structures?
2. What are sequence statements?
3. What are the various selection statements?
4. When and how to use the for loop?
5. When and how to use the while loop?

What are control structures in MATLAB?

A control structure is a code body that can direct program flow in a different manner using one of the following structures

- Sequence statements
    - The default mode of MATLAB and other programming languages, the interpreter processes the statements or commands one by one in sequence

- Selection statements
    - The selection statements encourage different flows due to different processing needs. These statements include:
        - If – else statements
        - Switch statements

- Repetition statements
    - The name says it all. It performs repetition to repeat a certain action or actions. Such statements include:
        - for loop
        - while loop

Sequence Statements

Sequence statements are really just step by step commands that are executed without conditions. They have no special keywords.

Example 1:

Write a program to calculate the force between the Earth and the Sun.

Code:

```
%  Given the formula
%              m_a * m_b
%     F   =  G ----------
%                 r^2
%  where m_a and m_b are masses of 2 bodies
%        R is the distance between m_a and m_b
%        G is 6.674 30 x 10^-11
%
%  Find F, force between the Sun and Earth
%  given
%     mass of sun is      1.989 √ó 10^30 kg
%     mass of earth is   5.97219 √ó 10^24 kg
%     distance btween them is 147 million kilometres

mass_sun = 1.989 * 10^30;
mass_earth = 5.97219 * 10^24;
distance = 147 * 10^6 * 10^3;

G = 6.67430 * 10^-11;
```

Example 2:

Write a simple program to calculate the Body Mass Index (bmi) of a person who is 60kg and stands at 1.75m tall.
Code:

```
weight = 60;
height = 1.75;
bmi = weight / (height * height);

fprintf("BMI is %.2f\n", bmi);
```

What are selection statements in MATLAB?

A MATLAB program uses the following structures for Selection:

- if statement
- if … else statement
- if … elseif … else statement
- switch … case

if Statement

A single if statement is used if a condition applies and if nothing else will be applied if the condition is not met:

Example 1:

A store sells some retail goods. If a customer is a member and he/she is entitled to 10% discount. If the person is not a member, then the discount does not apply.

Code:
```
price = 100;
is_member = true;
if is_member
   price = price * 0.9;
end
fprintf('Price to pay is %.2f\n', price);
```

Example 2:

A program is required to calculate the amount of government funding given based on the age of a newly hired worker. Create the program that takes in an age input and print the grant given by the government. You can assume that the government funding for companies hiring workers aged 65 and above is $5000.

Code:
```
age = input('Enter age of new hire: ');
grant = 0;
if age >= 65
    grant = 5000;
end
fprintf('Grant is: %d\n', grant);
```

if – else Statement

An if-else statement is used if a condition applies, an action must be performed and if the condition does not apply, another action must be performed:

Example 1:

Determine whether a person is eligible to start working in Singapore based on age. If the person is of legal age, inform the person that he/she is ready to work, otherwise inform the person that he/she cannot work due to his/her age.

Code:
```
age = 16;
if age > 16
    fprintf('You are %d and ready to work!\n', age);
else
    fprintf('No work for you as you are only %d\n', age);
end
```

Example 2:

In a driving school, students are required to take a test. Upon taking a test, he/she will be given demerit points during the test. A program is required to inform the test taker whether he/she has passed or failed. Create the program that takes in an input based on demerit

points and print the “Passed” or “Failed”. The passing mark is below 20 demerit points.

Code:
```
demerit_points = input('Enter number of demerit points attained: ');
if demerit_points < 20
   fprintf('Passed!\n');
else
   fprintf('Failed!\n');
end
```

if – elseif statement

An if-elseif statement is used if there could be multiple conditions to be tested but only 1 set of actions can be executed.

Example 1:

Determine the grade of a Special paper taken in ‘Advanced Levels’.

Code:
```
score = 75;
if score > 75 && score <= 100
   fprintf('Grade is Distinction!\n');
elseif score >= 50 && score <= 75
   fprintf('Grade is Merit\n');
elseif score >= 0 && score < 50
   fprintf('Grade is Fail\n');
else
   fprintf('Invalid score!\n');
```

Example 2:

Roxy Games Store sells all sorts of games like computer games, online game cards and board games. Roxy has 3 tiers of membership which will entitle members to different amount of discount. For Silver members, the discount is 10%. For Gold members, the discount is 15% and for Platinum members, the discount is 25%. Write a program to ask the user for total price of items and the membership tier (S – Silver, G – Gold, P – Platinum, N – None) and print the price to be paid after discount.

Code:
```
total_amount = input('Enter total amount purchased: $');
member_type = input('Enter member type (S, G, P) or N for none: ', 's');
discount = 0;

if strcmp(member_type, 'S')
   discount = 10;
elseif strcmp(member_type, 'G')
   discount = 15;
elseif strcmp(member_type, 'P')
```

```
    discount = 25;
end

total_amount = total_amount * (100 - discount) / 100;
fprintf('Amount to be paid: $%.2f after %d percent discount.\n', total_amount, discount);
```

Switch – case statement

A switch-case statement is sometimes used over if-else statements due to readability. It would be intuitive to use switch-case if you are testing a single variable and it has discrete values.:

Example 1:

Determine the discount given based on a number of rides purchased for a fun fair Ferris Wheel ride.

Code:

```
num_rides = 3;
discount;
switch (num_rides)
    case 1
        discount = 10;
    case { 2, 3 }
        discount = 20;
    case 4
        discount = 30;
    otherwise
        discount = 5;
end
disp(discount);
```

Example 2:

Roxy Games Store sells all sorts of games like computer games, online game cards and board games. Roxy has 3 tiers of membership which will entitle members to different amount of discount. For Silver members, the discount is 10%. For Gold members, the discount is 15% and for Platinum members, the discount is 25%. Write a program to ask the user for total price of items and the membership tier (S – Silver, G – Gold, P – Platinum, N – None) and print the price to be paid after discount. Implement the solution using a switch-case statement that caters to upper and lower cases of the inputs for (S, G, P)

Code:

```
total_amount = input('Enter total amount purchased: $');
member_type = input('Enter member type (S, G, P) or N for none: ', 's');
discount = 0;

switch member_type
    case {'S', 's'}
        discount = 10;
    case {'G', 'g'}
        discount = 15;
```

```
    case {'P', 'p'}
        discount = 25;
end

total_amount = total_amount * (100 - discount) / 100;
fprintf('Amount to be paid: $%.2f after %d percent discount.\n', total_amount, discount);
```

Explanation:

The switch case statements allow you to key in several outcomes for the value to be tested. You can see in this case, it is better than the if-elseif statements as it makes the code more readable and easier to maintain.

What are repetition statements in MATLAB?

A MATLAB program can repeat statements using one of the following repetition structures:

- for … end
- while … end

for – loop Statement

A for loop is used in situations where the number of repetitions is known:

Example 1:

Use a for-loop to print all integers from 1 to 5

Code:

```
for n=1:5
    fprintf('%d \n', n);    % 1 2 3 4 5
end
```

Example 2:

Use a for-loop to print all integers from 1 to 20, skipping 2 numbers for each iteration

Code:

```
for n=1:3:20
    fprintf('%d \n', n);    % 1 4 7 10 13 16 19
end
```

Example 3:

Use a for-loop to items in a vector

Code:

```
vec = [1, 3, 8, 5];
for n=1:length(vec)
```

```
   fprintf('%d \n', vec(n)); % 1 3 8 5
end
```

Example 4:

Use a for-loop to items in a matrix

Code:

```
my_mat = [3, 6, 9; 2, 4, 6];
[rows, cols] = size(my_mat);
for row=1:rows
   for col=1:cols
      fprintf('(%d) ', my_mat(row, col));
   end
   fprintf('\n');
end
% (3) (6) (9)
% (2) (4) (6)
```

Example 5:

Use a for-loop to print only ODD numbers greater than 5 in the matrix

Code:

```
my_mat = [3, 6, 9; 2, 5, 7; 11, 10, 7];
[rows, cols] = size(my_mat);
for row=1:rows
   for col=1:cols
      item = my_mat(row, col);
      if item > 5 && rem(item, 2) == 1
         fprintf('%d in position (%d, %d)\n', ...
            my_mat(row, col), row, col);
      end
   end
end

% 9 in position (1, 3)
% 7 in position (2, 3)
% 11 in position (3, 1)
% 7 in position (3, 3)
```

while – loop Statement

A while loop is typically used in situations where the number of repetitions is typically unknown:

Example 1:

Request the user to key in numbers and count how many numbers entered. Allow the user to enter 0 to quit

Code:

```
count = 0;
num = input('Enter any number or 0 to exit: ');
while num ~=0
   count = count + 1;
   num = input('Enter any number or 0 to exit: ');
end
fprintf('You entered %d numbers\n', count);
```

Example 2:

Allow the user to key in marks from 0 to 10. Count the number of passes (score 5 or more) and number of failures. Print the number of passes, failures and number of students. Allow the user to key in -1 to exit.

Code:

```
pass_count = 0;
fail_count = 0;
score = input('Enter a score between 0 to 10 (-1 to quit): ');
while score ~= -1
   if score >= 5 && score <= 10
      pass_count = pass_count + 1;
   elseif score >= 0 && score < 5
      fail_count = fail_count + 1;
   else
      fprintf('Invalid score! 0 to 10 only!\n');
   end
   score = input('Enter a score between 0 to 10 (-1 to quit): ');
end
fprintf('Number of passes: %d\n', pass_count);
fprintf('Number of failures: %d\n', fail_count);
fprintf('Number of students in total: %d\n', (pass_count + fail_count));
```

Example 3:

A system accepts an input as form of measurement and performs a conversion based on the units (cm to ft and inch) and vice versa.
The system will accept an input until NO further input is given and the program will end.

Code:

```
%  Example run:
%    Enter number:  100
%    Convert to (cm/feet)? feet
%    Converted: 3 ft 4 in
%    Enter number:  4
%    Convert to (cm/feet)? cm
%    Converted: 120 cm
%    Enter number:
%    Thank you and have a great day!
%

fprintf("Welcome to length conversion program!\n");
```

```
number = input("Enter a number: ");

while isempty(number) == false   % if user enters an empty string
  convert_type = input("Convert to cm or feet? ", "s");

  if strcmp(convert_type, "cm")   % user chooses cm
     fprintf("Converting feet to cm\n");
     converted_length = number * 30;   % 1 foot = 30 cm

     for index = 1:length(number)
        fprintf("%.2f feet = %.2f cm\n", number(index), converted_length(index));
     end
  elseif strcmp(convert_type, "feet")
     fprintf("Converting cm to feet\n");
  else   % user chooses anything other than cm or feet
     fprintf("Choice must be cm or feet only!\n");
  end

  number = input("Enter a number: ");
end
fprintf("Thank you and have a great day!\n");
```

Quick summary

MATLAB utilizes sequential code as default to run a series of commands in a program. Like most programming languages, MATLAB is also capable of allowing selection and repetition of statements. This capability of MATLAB shows that MATLAB is a full programming language that have these features to help engineers or scientists customise their solutions for their engineering needs or research purposes.

References List

MathWorks (n.d) MATLAB https://www.mathworks.com/products/matlab.html

Kaplan City Campus @ Wilkie Edge | 8 Wilkie Road, Level 2, Singapore 228095

Kaplan City Campus @ PoMo | 1 Selegie Rd, Level 7, Singapore 188306

6733 1877 info.sg@kaplan.com

KaplanSingapore

KAPLAN

Computational Mathematics and Computer Architecture

Topic 01
Computer Architecture

Learning Outcomes

Upon successful completion of this topic, students should be able to:

- Explain the basics of the Central Processing Unit (CPU)
- Explain how data flows between CPU and main memory

WHAT IS A COMPUTER?

- An electronic device that stores and processes data and information.
- A system that receives inputs, processes the inputs and produces outputs.
- A complex system containing millions of electronic components.

KAPLAN

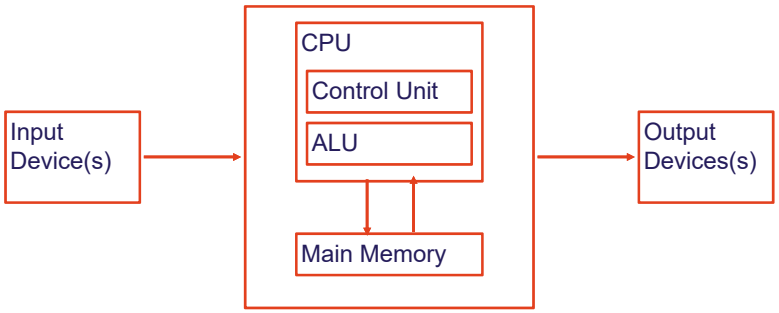
TYPES OF COMPUTERS

iMac

Oppo Phone

KAPLAN

SIMPLIFIED DIAGRAM OF A COMPUTER SYSTEM



5

KAPLAN

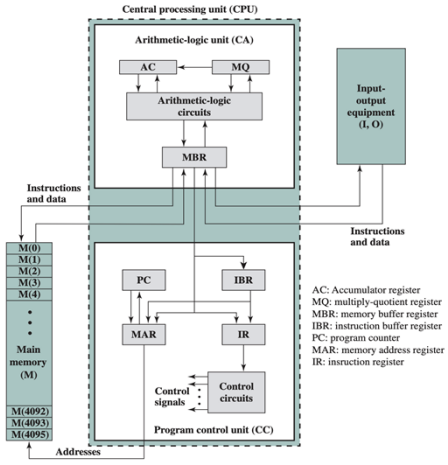
QUICK BRIEF HISTORY: IAS STRUCTURE

- The IAS Computer is first stored programme computer which forms the base of most of all modern computers.
- Attributed to John von Neumann, it is also known as the von Neumann Architecture.

6

KAPLAN

IAS STRUCTURE



IAS Structure

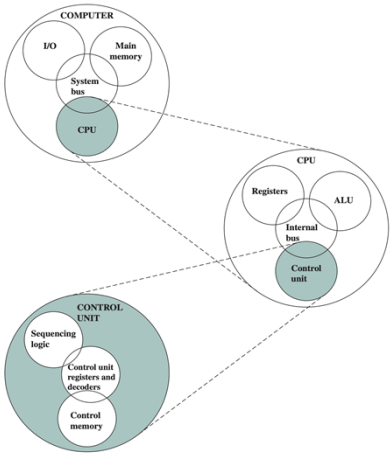
KAPLAN

MODERN COMPUTERS

- Hence a typical modern computer follows the von Neumann architecture
- A typical modern computer is interested in providing functions such as data processing, data storage, data movement and control
- To provide the functionalities, a modern single processor computer would usually consist of:
  - A Central Processing Unit
  - Main Memory
  - I/O
  - System interconnection

KAPLAN

HIERARCHICAL VIEW OF SINGLE PROCESSOR COMPUTER



View of Single Processor Computer

9

KAPLAN

WHAT IS A CENTRAL PROCESSING UNIT?

- A component of the computer that:
  - controls the operation of the computer;
  - performs its data processing functions.

10

KAPLAN

COMPONENTS OF CPU

1. Control Unit
   - Controls the computer's operation
2. ALU
   - Performs data processing functions such as add, divide, etc.
3. Registers
   - Holds temporary storage in the CPU
4. CPU Interconnection
   - To allow communication among the above components

11

KAPLAN

TYPES OF REGISTERS

- Memory Buffer Register
- Memory Address Register
- Instruction Register
- Instruction Buffer Register
- Programme Counter
- Accumulator

12

KAPLAN

OTHER TYPES OF COMPUTER ARCHITECTURE

- Harvard Architecture
- ARM Architecture

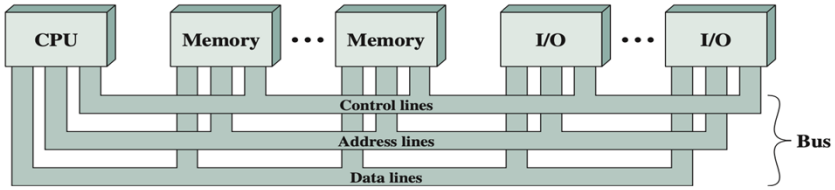
KAPLAN

HOW DATA FLOW FROM CPU TO MAIN MEMORY & OTHER DEVICES?

- To allow CPU, main memory and other devices to communicate, there must be paths and connections between them.
- To allow such communication, there must be an interconnection structure.
- Hence buses are commonly used as a means of communication in the computer system.

KAPLAN

BUS



Bus Interconnection Scheme

15

KAPLAN

SYSTEM BUS

A system bus is a bus that connects major computer components with multiple lines

- Processor
- Memory
- IO

16

KAPLAN

SYSTEM BUS – LINES

- Data Lines / Data Bus
  - Lines that carry data are known as data lines
  - Together they are known as the data bus
- Address Lines / Address Bus
  - Address lines determine the maximum possible memory capacity of the system
  - Together they are known as Address bus
- Control lines
  - Control the access to and use of the data and address lines

17

KAPLAN

SUMMARY

- CPU is the Central Processing Unit in a computer system and it is the unit that executes instructions and convert the relevant instructions into actions or outputs.
- To communicate with memory and other devices, buses are used in the computer system to facilitate such communication.

18

KAPLAN

References

- Stallings, W. (2016). Computer organization and architecture: Designing for performance. Hoboken, N.J: Pearson.

KAPLAN

KAPLAN

Computational Mathematics and Computer Architecture

Topic 02
Memory Hierarchy

Learning Outcomes

Upon successful completion of this topic, students should be able to:

- List the characteristics of the computer memory systems.
- Describe the use of a memory hierarchy.
- Describe the basic concepts and intent of cache memory.

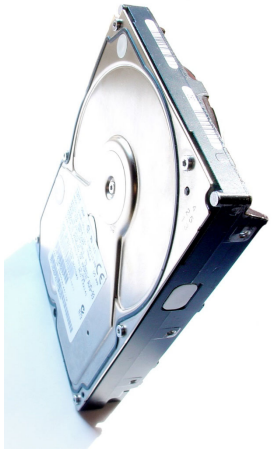
KAPLAN

WHAT IS COMPUTER MEMORY?

- A storage space in the computer.
- Used to store data to be processed or saved.
- Hence memory can be categorized as permanent or temporary (volatile).
- Different memory devices are grouped under the **Memory Hierarchy.**

KAPLAN

EXAMPLES OF MEMORY



KAPLAN

COMPUTER MEMORY KEY CHARACTERISTICS

- Location
  - -Internal memory / External memory
  - -E.g. RAM / Hard disk
- Capacity
  - -Number of bytes, words
  - -E.g. 2 TB, 500 GB

KAPLAN

COMPUTER MEMORY KEY CHARACTERISTICS

- Unit of Transfer
  - -Word generally refers number of bits to represent an integer
  - -For RAM, the number of bits read from / written into
- Method of Access
  - -Sequential
  - -Direct Access
  - -Random Access
  - -Associative Access

KAPLAN

COMPUTER MEMORY KEY CHARACTERISTICS

- Performance
  - -Matters most to the users
  - -Metrics include
    - -Access Time,
    - -Memory Cycle Time,
    - -Transfer Rate
- Physical Type
  - -Semi conductor, Magnetic etc

KAPLAN

COMPUTER MEMORY KEY CHARACTERISTICS

- Physical Characteristics
  - -Volatile
  - -Non-volatile
  - -Erasable
  - -Non-Erasable
- Organization
  - -Arrangement of bits to form words
  - -Matters to random access memory

KAPLAN

COMPUTER MEMORY GOALS

- **Goals** of Computer Memory can be summarized as follows:
  - To provide enough storage to hold data
  - To provide a memory system that is affordable
  - To provide a speed almost as fast as the fastest level
- **Question**: How do we provide a system that has ample memory so cheap yet so fast?

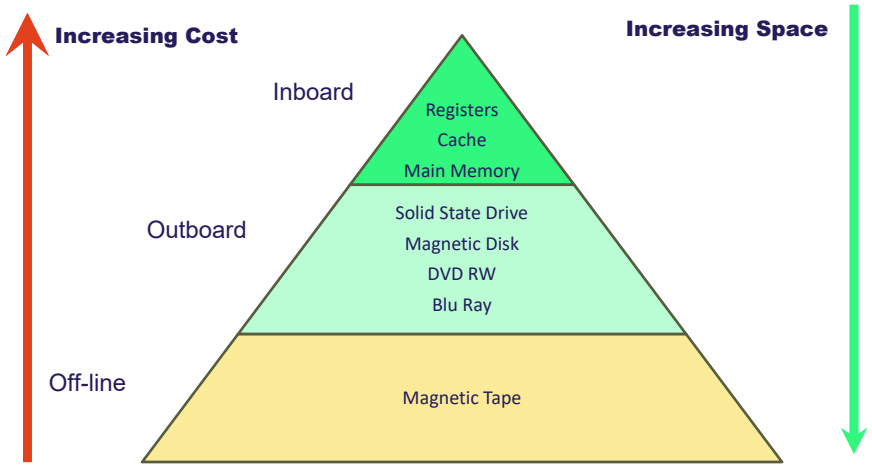
KAPLAN

COMPUTER MEMORY HIERARCHY

- **Answer**: Use a Memory Hierarchy instead of a single memory component.
- The relationships generally hold true:
  - -Speed of access **UP**, Cost per bit **UP**
  - -Capacity **UP**, Cost per bit **DOWN**
  - -Capacity **UP**, Speed of access **DOWN**

KAPLAN

MEMORY HIERARCHY



KAPLAN

ACHIEVING MEMORY GOALS

How do we achieve the goals?

ANSWER: **Decrease frequency of access**.

To help decrease frequency of access, we will need to understand **Locality of Reference Principle**

KAPLAN

LOCALITY OF REFERENCE

- Also known as Principle of Locality
- From observations, programs tend to reuse data and instructions they have used recently.
- **Important Observation**: 90% of execution happens in 10% of code
- **Important Realization**: We can predict the instructions and data based on past data

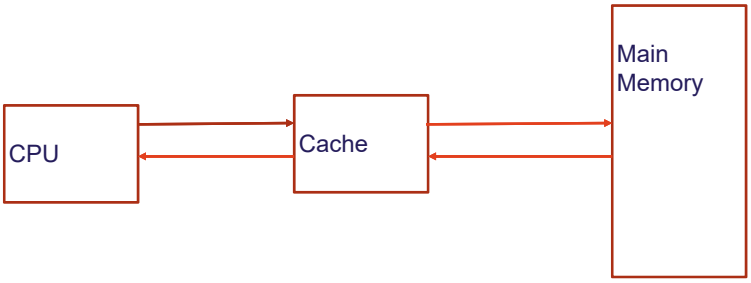
KAPLAN

CACHE MEMORY PRINCIPLES

- Even with Principle of Locality, the registers on the CPU are too small to hold the data
- Main Memory onboard are much larger but far too slow
- The Solution:
  - Put a faster memory type in between the CPU and Main Memory
  - This memory is known as Cache Memory

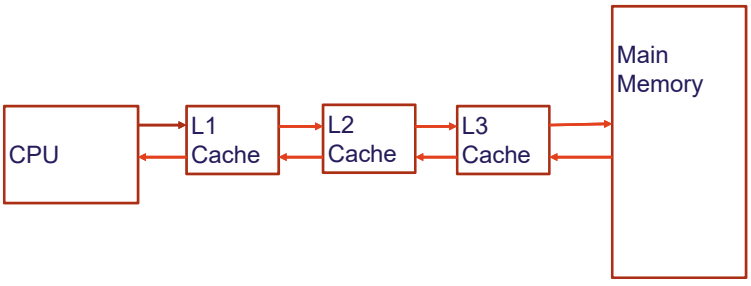
KAPLAN

CACHE MEMORY DIAGRAM – SINGLE CACHE



KAPLAN

CACHE MEMORY DIAGRAM – 3 LEVEL CACHE



KAPLAN

CACHE ANALOGY

- Caching can be explained with the following analogy:
  - You are researching a topic based on a particular article on a book found in the Library.
  - You can either go to the visit the library every time you need to read up the article again (No Cache) OR;
  - You can borrow the book temporarily unless you do not need to use the book again (Cache).

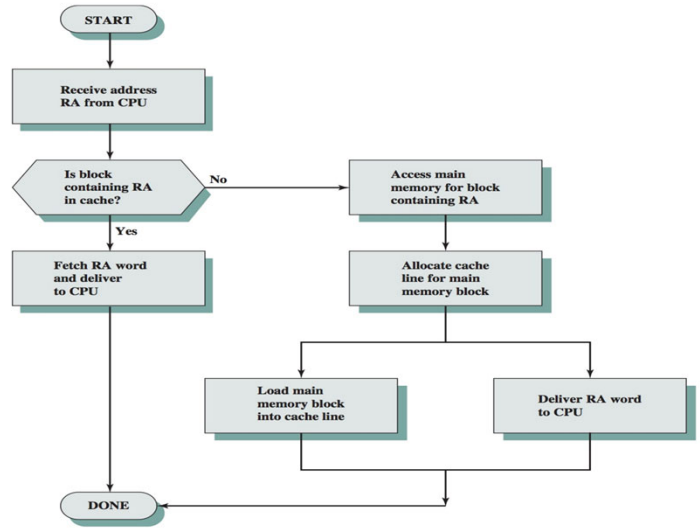
KAPLAN

CACHE READ OPERATION

- Let us examine how a typical Cache Read Operation looks like:
  - Step 1: Read address of word to be read generated by Processor
  - Step 2: Found in cache?
    - If Found, deliver to Processor
    - If NOT Found, Cache Miss (Step 3)
  - Step 3: Cache Miss (Optional)
    - Address loaded onto system bus
    - Deliver data to both cache and processor

KAPLAN

CACHE READ OPERATION



KAPLAN

ELEMENTS OF CACHE DESIGN

• There are several aspects of Cache Design, let us focus on the following:

- Cache Addresses
- Cache Size
- Mapping Function
- Line Size
- Replacement Algorithms
- Write Policy
- Number of Caches

KAPLAN

CACHE ADDRESSES

- Cache Addresses:
  - Logical also known as Virtual Address
    - Stores data using Virtual Addresses
  - Physical
    - Stores data using main memory physical addresses

KAPLAN

CACHE SIZE

- Ideally small enough such that cost per bit is approximately cost of main memory
- Ideally big enough so that the access time is close to the cache alone.

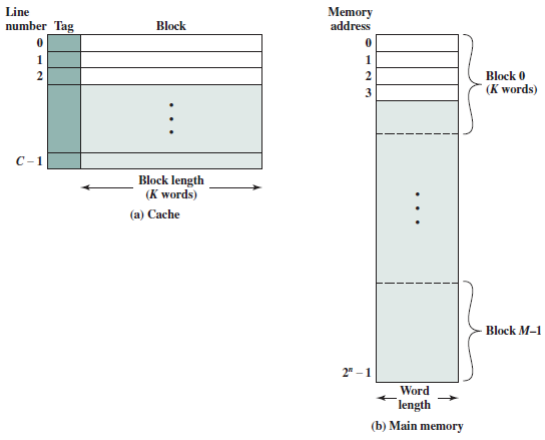
KAPLAN

CACHE – MAPPING FUNCTION AND LINES

- To understand how caching will work, we will have to observe the mapping function of the memory to the cache and vice versa.
- The cache will contain m blocks and each block will contain K words.
- Each block is also known as **lines**. The number of lines will be much less than the number of blocks in the Main memory (m <= M)
- The implication is the Principle of Locality must be utilized to ensure less access to the Main Memory.

KAPLAN

CACHE / MAIN MEMORY STRUCTURE



KAPLAN

LINE SIZE

- No optimum line size.
- Approximately between 8 bytes to 64 bytes.
- If line size is too small, hit ratio will be low.
- If line size is too big, data is overwritten very soon after it is fetched.

KAPLAN

REPLACEMENT ALGORITHMS

- A cache that is full eventually will require a new block of data to be brought in.

- How the data is replaced can be based on one of the following algorithms:
  - First In First Out (FIFO)
  - Least Frequently Used (LFU)
  - Least Recently Used (LRU)

KAPLAN

WRITE POLICY

- When writing to the cache, there are 2 techniques:
  - **Write Through**
    - Writes to both cache and main memory
    - Slower
    - More reliable
  - **Write back**
    - Writes on to cache
    - Faster
    - Less reliable

KAPLAN

NUMBER OF CACHES

- When writing to the cache, there are 2 techniques:
  - **Multilevel cache**
    - More efficient than a single level cache
    - E.g. L1 – internal cache and L2 – external cache
  - **Unified cache vs Split Cache**
    - Unified Cache
      - Higher hit rate
    - Split Cache
      - One dedicated to instructions, the other dedicated to data
      - More potential for scalability

KAPLAN

SUMMARY

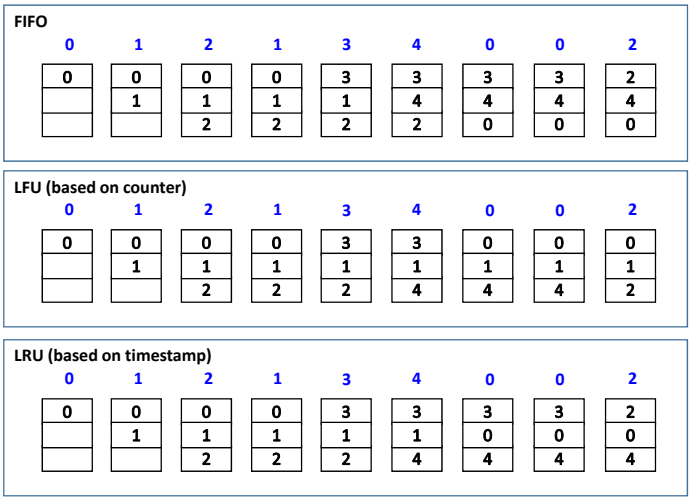
- Computer Memory is used for storing data permanently or temporarily.
- To achieve the goals of having Computer Memory that is affordable, fast and big lies in the utilization of Principle of Locality.
- An example of Principle of Locality in use is the utilization of Cache.

KAPLAN

References

- Stallings, W. (2016). Computer organization and architecture: Designing for performance. Hoboken, N.J: Pearson.
- Hennessy J and Patterson D (2019). Computer Architecture: A Quantitative Approach, Morgan Kaufmann.

KAPLAN


FIFO
0 1 2 1 3 4 0 0 2
0 0 0 0 3 3 3 3 2
1 1 1 1 4 4 4 4
2 2 2 2 0 0 0
LFU (based on counter)
0 1 2 1 3 4 0 0 2
0 0 0 0 3 3 0 0 0
1 1 1 1 1 1 1 1
2 2 2 2 4 4 4 2
LRU (based on timestamp)
0 1 2 1 3 4 0 0 2
0 0 0 0 3 3 3 3 2
1 1 1 1 1 0 0 0
2 2 2 2 4 4 4 4

KAPLAN

Computational Mathematics and Computer Architecture

Topic 03
Base Conversions I

Learning Outcomes

Upon successful completion of this topic, students should be able to:

1. Explain how to convert from binary to decimal and vice versa.
2. Explain how to convert from hexadecimal to decimal and vice versa.
3. Explain how to add binary numbers.
4. Explain how to add hexadecimal numbers.

HOW DOES COMPUTERS INTERPRET NUMBERS?

- Do computers know how to interpret numbers?
- The short answer is yes.
- However only through conversion of human readable numbers to **binary numbers** .
- A binary number is a numbering system using bits.

KAPLAN

WHAT IS A BIT? WHAT IS A BYTE?

- Bit – **B**inary Dig**it**.
  - Small unit in a computer system
  - Takes on either 0 or 1 (True or False)
- Byte – 8 bits
- To understand how binary numbers work, let's look at the Decimal numbering system followed by the Binary numbering system.

KAPLAN

DECIMAL NUMBERING SYSTEM EXAMPLE

2 3 5

5

KAPLAN

DECIMAL NUMBERING SYSTEM EXAMPLE (2)

2 3 5

0 The first place is always 0 from the right hand side

6

KAPLAN

DECIMAL NUMBERING SYSTEM EXAMPLE (3)

2 3 5
1 0

The second place is incremented by 1

7

KAPLAN

DECIMAL NUMBERING SYSTEM EXAMPLE (4)

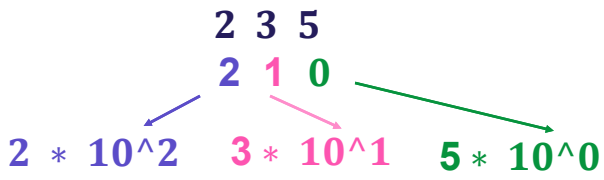
2 3 5
2 1 0

The most significant number has position 2

8

KAPLAN

DECIMAL NUMBERING SYSTEM EXAMPLE (5)



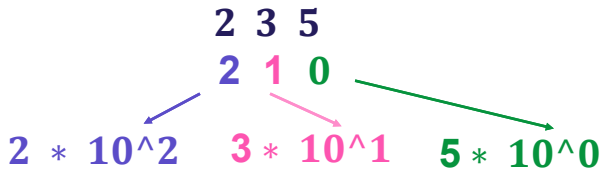
As they are base of 10, the number means the indices
Or to the power of

9

KAPLAN

Decimal Numbering System Example (6)

Now add them up



$$200 + 30 + 5$$

$$= 235$$

10

KAPLAN

BINARY NUMBERING SYSTEM EXAMPLE

1 0 1 0

11

KAPLAN

BINARY NUMBERING SYSTEM EXAMPLE (2)

1 0 1 0
0

Like the decimal numbering system, the first place is always 0 from the right hand side

12

KAPLAN

BINARY NUMBERING SYSTEM EXAMPLE (3)

1 0 1 0
1 0

Like the previous example,
the second place is incremented by 1

13

KAPLAN

BINARY NUMBERING SYSTEM EXAMPLE (4)

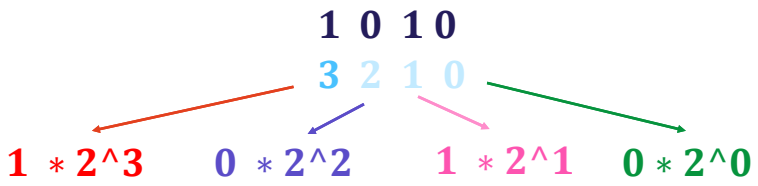
1 0 1 0
3 2 1 0

The most significant number
has position 3

14

KAPLAN

BINARY NUMBERING SYSTEM EXAMPLE (5)



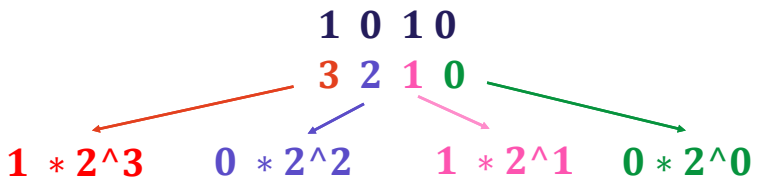
In this case they are base of 2, hence the numbers are multiplied By base 2 to the power of 2

15

KAPLAN

BINARY NUMBERING SYSTEM EXAMPLE (6)

Again, sum them all up



$$8 + 0 + 2 + 0$$

$$= 10$$

16

KAPLAN

CONVERT FROM BINARY NUMBERS TO DECIMAL NUMBERS

- In the previous example, we have successful converted a binary number into a decimal number.
- Now let us look at how to convert from a decimal number into a binary number instead.

KAPLAN

CONVERT FROM DECIMAL NUMBERS TO BINARY NUMBERS

One way to convert from Decimal numbers to Binary numbers is to follow the steps below:

- Divide the decimal number by 2.
- Use the integer quotient for the next repetition.
- List down all the remainders for each division in order.
- Repeat the steps until the quotient is equal to 0.
- Reverse the remainders and you will have the binary number equivalent.

KAPLAN

CONVERT FROM DECIMAL NUMBERS TO BINARY NUMBERS

Divisor	Quotient	Remainder
2	39	

19

KAPLAN

CONVERT FROM DECIMAL NUMBERS TO BINARY NUMBERS

Divisor	Quotient	Remainder
2	39	1
2	19	

20

KAPLAN

CONVERT FROM DECIMAL NUMBERS TO BINARY NUMBERS

Divisor	Quotient	Remainder
2	39	1
2	19	1
2	9	

21

KAPLAN

CONVERT FROM DECIMAL NUMBERS TO BINARY NUMBERS

Divisor	Quotient	Remainder
2	39	1
2	19	1
2	9	1
2	4	

22

KAPLAN

CONVERT FROM DECIMAL NUMBERS TO BINARY NUMBERS

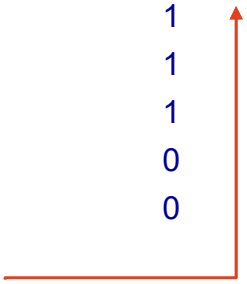
Divisor	Quotient	Remainder
2	39	1
2	19	1
2	9	1
2	4	0
2	2	

23

KAPLAN

CONVERT FROM DECIMAL NUMBERS TO BINARY NUMBERS

Divisor	Quotient	Remainder
2	39	1
2	19	1
2	9	1
2	4	0
2	2	0
	1	



Answer: 0 0 1 0 0 1 1 1

24

KAPLAN

TRY CONVERTING THESE

- 0101 1011 to decimal number
- 1010 1100 to decimal number

- 328 to binary number
- 733 to binary number

KAPLAN

WHAT IS AN OCTAL NUMBER?

- An Octal Number is a number of base 8

- Goes from 0 to 8

KAPLAN

CONVERT FROM OCTAL NUMBERS TO DECIMAL NUMBERS

Number the positions as usual

$$\begin{matrix} 2 & 3 & 7 \\ 2 & 1 & 0 \end{matrix}$$

27

KAPLAN

CONVERT FROM OCTAL NUMBERS TO DECIMAL NUMBERS (2)

Multiply base 8 and add them

$$\begin{matrix} 2 & 3 & 7 \\ 2 & 1 & 0 \end{matrix}$$

$$2 * 8^2 \qquad 3 * 8^1 \qquad 7 * 8^0$$

$$128 + 24 + 7$$

$$= 159$$

28

KAPLAN

CONVERT FROM DECIMAL NUMBERS TO OCTAL NUMBERS

Divisor	Quotient	Remainder
8	189	

29

KAPLAN

CONVERT FROM DECIMAL NUMBERS TO OCTAL NUMBERS

Divisor	Quotient	Remainder
8	189	5
8	23	

30

KAPLAN

CONVERT FROM DECIMAL NUMBERS TO OCTAL NUMBERS

Divisor	Quotient	Remainder
8	189	5
8	23	7
	2	

Answer: 275

31

KAPLAN

CONVERT OCTAL TO BINARY

- An Octal number can also be converted into a Binary number easily
- Each Octal number is equivalent to 3 bits

32

KAPLAN

CONVERT FROM OCTAL NUMBERS TO BINARY NUMBERS

Each number is equivalent to 3 bits

2 3 7

1 1 1

33

KAPLAN

CONVERT FROM OCTAL NUMBERS TO BINARY NUMBERS (2)

2 3 7

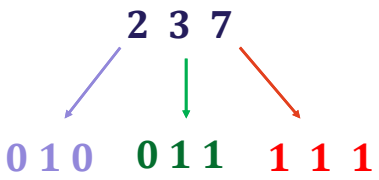
0 1 1 1 1 1

34

KAPLAN

CONVERT FROM OCTAL NUMBERS TO BINARY NUMBERS (3)

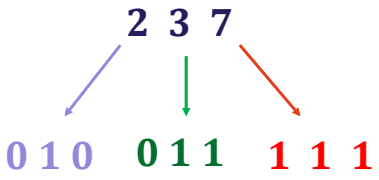
Number the numbers in multiples of 8



35

KAPLAN

CONVERT FROM OCTAL NUMBERS TO BINARY NUMBERS (3)



Rewrite as 0000 0000 1001 1111

36

KAPLAN

WHAT IS A HEXADECIMAL NUMBER?

- A Hexadecimal Number is a number of base 16
- Goes from 0 to 9, A to F
- A is 10, B is 11 … F is 15

37

KAPLAN

USES OF HEXADECIMAL NUMBERS

- Hexadecimals are used to represent IPv6 addresses.
- They are used to represent hardware addresses.

38

KAPLAN

CONVERT FROM HEXADECIMAL NUMBERS TO DECIMAL NUMBERS

Number the positions as usual

$$1\ A\ C$$
$$2\ 1\ 0$$

39

KAPLAN

CONVERT FROM HEXADECIMAL NUMBERS TO DECIMAL NUMBERS (2)

Multiply base 16 and add them

$$1\ A\ C$$
$$2\ 1\ 0$$

$$1 * 16\text{^}2 \qquad 10 * 16\text{^}1 \qquad 12 * 16\text{^}0$$

$$256 + 160 + 12$$

$$= 428$$

40

KAPLAN

CONVERT FROM DECIMAL NUMBERS TO HEXADECIMAL NUMBERS

Divisor	Quotient	Remainder
16	335	

41

KAPLAN

CONVERT FROM DECIMAL NUMBERS TO HEXADECIMAL NUMBERS

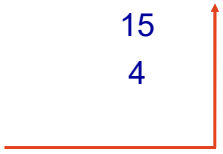
Divisor	Quotient	Remainder
16	335	15
16	20	

42

KAPLAN

CONVERT FROM DECIMAL NUMBERS TO HEXADECIMAL NUMBERS

Divisor	Quotient	Remainder
16	335	15
16	20	4
	1	



Answer: 0x14F

43

KAPLAN

CONVERT FROM HEXADECIMAL NUMBERS TO BINARY NUMBERS

Each number is 4 bits

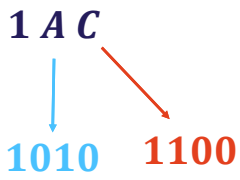
$1\ A\ C$

1100

44

KAPLAN

CONVERT FROM HEXADECIMAL NUMBERS TO BINARY NUMBERS (2)

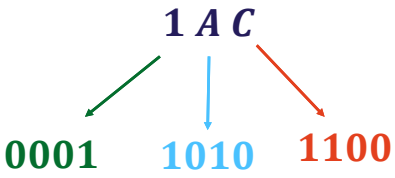


45

KAPLAN

CONVERT FROM HEXADECIMAL NUMBERS TO BINARY NUMBERS (3)

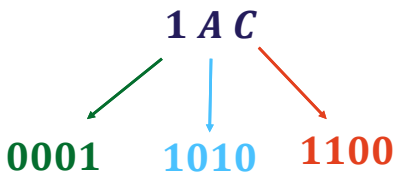
The final number should be in multiple of 8



46

KAPLAN

CONVERT FROM HEXADECIMAL NUMBERS TO BINARY NUMBERS (4)



Answer: 0000 0001 1010 1100

47

KAPLAN

POINT TO PONDER

- Why are Hexadecimal numbers more popular than Octal numbers?

48

KAPLAN

SUMMARY

- Computers understand binary numbers. Numbers have to be converted to binary numbers before they can be processed by computers.
- Hexadecimals are used in IPv6 and is a shorter representation than binary numbers.

KAPLAN

References

- https://www.geeksforgeeks.org/program-binary-decimal-conversion/

KAPLAN

KAPLAN

Computational Mathematics and Computer Architecture

Topic 04
Base Conversions II

Learning Outcomes

Upon successful completion of this topic, students should be able to;

1. Explain how to convert from binary to decimal and vice versa.
2. Explain how to convert from hexadecimal to decimal and vice versa.
3. Explain how to add binary numbers.
4. Explain how to add hexadecimal numbers.

ADDING BINARY NUMBERS

- Adding binary numbers is like adding decimal numbers except that the numbers are of base 2

0 1 0 1 + 0 0 1 1

```
   1
 0 1 0 1
+0 0 1 1
---------
       0
```

3

KAPLAN

ADDING BINARY NUMBERS (2)

- If the numbers add to 2 or 3, then there is a carry over

0 1 0 1 + 0 0 1 1

```
   1
 0 1 0 1
+0 0 1 1
---------
     0 0
```

4

KAPLAN

ADDING BINARY NUMBERS (3)

- Normally the numbers are in multiples of 8

0 1 0 1 + 0 0 1 1

```
  1
  0 1 0 1
+ 0 0 1 1
-------
    0 0 0
```

5

KAPLAN

ADDING BINARY NUMBERS (4)

- For simplicity, we keep the numbers to 4 bits here.

0 1 0 1 + 0 0 1 1

```
  1
  0 1 0 1
+ 0 0 1 1
-------
  1 0 0 0
```

6

KAPLAN

SUBTRACTING BINARY NUMBERS

- Subtracting binary numbers is similar to adding

0 1 0 1 − 0 0 1 1

```
  0 1 0 1
− 0 0 1 1
---------
        0
```

7

KAPLAN

SUBTRACTING BINARY NUMBERS (2)

- Like decimal subtraction, there is a need to borrow if the minuend is not enough to be deducted from

0 1 0 1 − 0 0 1 1

```
    1 0
  0 ~~1~~ 0 1
− 0 0 1 1
---------
        0
```

8

KAPLAN

SUBTRACTING BINARY NUMBERS (3)

0 1 0 1 - 0 0 1 1

```
  0 1 0 1
- 0 0 1 1
---------
    1 0
```

9

KAPLAN

MULTIPLYING BINARY NUMBERS

- Multiplying binary numbers is similar to multiplication of decimal numbers

0 1 0 1 * 0 0 1 1

```
  0 1 0 1
* 0 0 1 1
---------
  0 1 0 1
```

10

KAPLAN

MULTIPLYING BINARY NUMBERS (2)

0 1 0 1 * 0 0 1 1

```
    0 1 0 1
*   0 0 1 1
-----------
    0 1 0 1
  0 1 0 1 0
```

11

KAPLAN

MULTIPLYING BINARY NUMBERS (3)

0 1 0 1 * 0 0 1 1

```
    0 1 0 1
*   0 0 1 1
-----------
    0 1 0 1
  0 1 0 1 0
-----------
  0 1 1 1 1
```

12

KAPLAN

ADDING OCTAL NUMBERS

- Adding octal numbers is similar as well

1 3 5 + 2 6

```
    1
  1 3 5
+   2 6
-------
      3
```

13

KAPLAN

ADDING OCTAL NUMBERS (2)

- When numbers add to above 8, there is a carry over

1 3 5 + 2 6

```
  1 3 5
+   2 6
-------
  1 6 3
```

14

KAPLAN

ADDING HEXADECIMAL NUMBERS

- Adding hexadecimals numbers is similar as well

1 A + 2 C

```
  1
  1 A
+ 2 C
-----
    6
```

15

KAPLAN

ADDING HEXADECIMAL NUMBERS (2)

- When numbers add to 16 and above, there is a carry over.

1 A + 2 C

```
  1 A
+ 2 C
-----
  4 6
```

16

KAPLAN

TRY ADDING THESE

- Binary numbers: 0101 1011 and 0010 1100
- Octal numbers: 555 and 657
- Hexadecimal numbers: 12A and ABC

KAPLAN

SUMMARY

- Addition, subtraction, multiplication and division can be performed on binary numbers.
- Similarly other forms such as octal and hexadecimal numbers can do the same thing.

KAPLAN

References

- https://www.geeksforgeeks.org/program-binary-decimal-conversion/

KAPLAN

KAPLAN

Computational Mathematics and Computer Architecture

Topic 05
Matrix Algebra

Learning Outcomes

Upon successful completion of this topic, students should be able to;

1. Explain what a matrix is.

KAPLAN

WHAT IS A MATRIX?

- A collection of numbers
- Consists of rows and columns
- Each item in a matrix is known as an element

3

KAPLAN

APPLICATION OF MATRICES TODAY

- Computer Graphics
- Encoding and decoding messages
- Study of quantum mechanics
- Physical System Modelling

And many more …

4

KAPLAN

A MATRIX WITH M ROWS AND N COLUMNS

$$\begin{pmatrix} a11 & a12 & \dots & a1n \\ a21 & a22 & \dots & a2n \\ \dots & \dots & \dots & \dots \\ am1 & am2 & \dots & amn \end{pmatrix}$$

m x n matrix

5

KAPLAN

EQUAL MATRICES

$$\boldsymbol{A = B}$$

$$\begin{pmatrix} \boldsymbol{a} & \boldsymbol{b} \\ \boldsymbol{c} & \boldsymbol{d} \end{pmatrix} = \begin{pmatrix} \boldsymbol{w} & \boldsymbol{x} \\ \boldsymbol{y} & \boldsymbol{z} \end{pmatrix}$$

Matrix A is said to be equal to Matrix B
IF and only IF they are of the same dimension
And corresponding elements are the same

In this case, a=w, b=x, c=y and d=z

6

KAPLAN

EQUAL MATRICES EXAMPLE

$$\begin{pmatrix} a & 3 \\ c & -1 \end{pmatrix} = \begin{pmatrix} 3 & x-2 \\ x+2 & z+1 \end{pmatrix}$$

Solve the unknown variables

a = 3

x – 2 = 3
x = 5

c = x + 2
= 5 + 2
= 7

z + 1 = -1
z = -2

7

KAPLAN

MATRIX ADDITION

Matrix Addition can only take place if the dimensions of both matrices are the same

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} + \begin{pmatrix} w & x \\ y & z \end{pmatrix} = \begin{pmatrix} a+w & b+x \\ c+y & d+z \end{pmatrix}$$

8

KAPLAN

MATRIX ADDITION EXAMPLE

$$\begin{pmatrix} 3 & 1 \\ -1 & 4 \end{pmatrix} + \begin{pmatrix} 5 & 6 \\ 8 & 0 \end{pmatrix} = \begin{pmatrix} 3+5 & 1+6 \\ -1+8 & 4+0 \end{pmatrix}$$

$$= \begin{pmatrix} 8 & 7 \\ 7 & 4 \end{pmatrix}$$

9

KAPLAN

MATRIX SUBTRACTION

Matrix Subtraction like Matrix Addition takes place if dimensions of both matrices are the same.

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} - \begin{pmatrix} w & x \\ y & z \end{pmatrix} = \begin{pmatrix} a-w & b-x \\ c-y & d-z \end{pmatrix}$$

10

KAPLAN

MATRIX ADDITION / SUBTRACTION EXERCISES

$$\begin{pmatrix} 2 & 3 \\ -1 & 5 \end{pmatrix} + \begin{pmatrix} 4 & 5 \\ -1 & 8 \end{pmatrix} = ?$$

$$\begin{pmatrix} 4 & 5 & 3 \\ 7 & -2 & 0 \end{pmatrix} - \begin{pmatrix} -2 & 3 & 4 \\ -5 & 6 & 7 \end{pmatrix} = ?$$

11

KAPLAN

SCALAR PRODUCT

- Scalar Product is implemented simply by multiplying a multiplier to the matrix elements.

$$m\begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} ma & mb \\ mc & md \end{pmatrix}$$

12

KAPLAN

SCALAR PRODUCT EXAMPLE

- Find the resultant matrix

$$2\begin{pmatrix} 3 & 1 \\ 5 & 7 \end{pmatrix} = \begin{pmatrix} 2*3 & 2*1 \\ 2*5 & 2*7 \end{pmatrix}$$

$$= \begin{pmatrix} 6 & 2 \\ 10 & 14 \end{pmatrix}$$

13

KAPLAN

MATRIX MULTIPLICATION

- To multiply the matrix A by the matrix B, the number of columns in A must be equal to the number of rows in B.
- If A is a $m\,x\,k$ matrix, and B is a $k\,x\,n$ matrix, then the resultant matrix of multiplication of matrix A by matrix B is a $m\ by\ n$ matrix.

14

KAPLAN

MATRIX MULTIPLICATION

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} w & x \\ y & z \end{pmatrix} = \begin{pmatrix} aw + by & ax + bz \\ cw + dy & cx + dz \end{pmatrix}$$

15

KAPLAN

MATRIX MULTIPLICATION EXAMPLE

$$A = \begin{pmatrix} 3 & 5 \\ 7 & 2 \\ -1 & 8 \end{pmatrix}, \qquad B = \begin{pmatrix} -1 & 3 & 2 \\ 4 & 5 & 6 \end{pmatrix},$$

$$AB = \begin{pmatrix} 3*-1+5*4 & 3*3+5*5 & 3*2+5*6 \\ 7*-1+2*4 & 7*3+2*5 & 7*2+2*6 \\ -1*-1+8*4 & -1*3+8*5 & -1*2+8*6 \end{pmatrix}$$

$$= \begin{pmatrix} 17 & 34 & 36 \\ 1 & 31 & 26 \\ 33 & 37 & 46 \end{pmatrix}$$

16

KAPLAN

MATRIX MULTIPLICATION EXERCISE

$$A = \begin{pmatrix} \mathbf{5} & \mathbf{9} \\ \mathbf{3} & \mathbf{2} \end{pmatrix}, \quad B = \begin{pmatrix} 1 & 3 & 5 \\ 4 & 7 & 2 \end{pmatrix},$$

$$AB = ?$$

17

KAPLAN

SUMMARY

- Matrices are used in various fields such as graphics and modelling.
- Matrix operations include addition, multiplication and scalar product.

18

KAPLAN

References

- https://www.geeksforgeeks.org/program-binary-decimal-conversion/

KAPLAN

KAPLAN

Computational Mathematics and Computer Architecture

Topic 06
Boolean Algebra

Learning Outcomes

Upon successful completion of this topic, students should be able to;

1. Describe Boolean Algebra.
2. Describe Boolean Function.
3. Describe Boolean Expression.

WHAT IS THE PURPOSE OF BOOLEAN ALGEBRA?

- Also known as Logical Algebra.
- Used to simplify the logic circuits.
- Uses 0 and 1.

3

KAPLAN

BOOLEAN ALGEBRA AND BASIC OPERATIONS

- Boolean Algebra has a number of operations;
  - -NOT
  - -AND
  - -OR
- To understand the operations better, let's take a look at the different gates.

4

KAPLAN

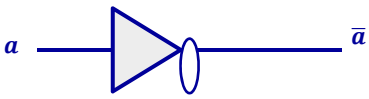
THE NOT GATE

- Also known as an inverter, the NOT Gate takes a single input and flips the it as an output.
- Hence a 0 as an input will be output as 1 and vice versa.

KAPLAN

DIAGRAM OF NOT GATE

- The input a from the left hand side of the diagram will be flipped to output as $\overline{a}$.



KAPLAN

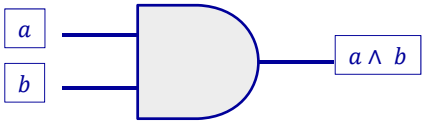
THE AND GATE

- The AND Gate takes in 2 or more inputs and checks whether all inputs are 1.
- Hence if all inputs are 1, the output will be 1. In any other cases, the output will be 0.

KAPLAN

DIAGRAM OF AND GATE

- The input ***a*** and ***b*** from the left hand side of the diagram must be 1 for the result to be 1.



KAPLAN

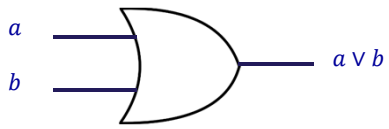
THE OR GATE

- The OR Gate takes in 2 or more inputs and checks whether at least one of inputs are 1.
- If at least one of the inputs are 1, the output will be 1. In any the case of all inputs are 0, the output will be 0.

KAPLAN

DIAGRAM OF OR GATE

- Either input ***a*** or ***b*** from the left hand side of the diagram must be 1 for the result to be 1.



KAPLAN

BOOLEAN ALGEBRA AND OTHER OPERATIONS

- Other Boolean Algebra operations are:
  - -NAND
  - -NOR
  - -XOR
- Let's take a look at such gates

KAPLAN

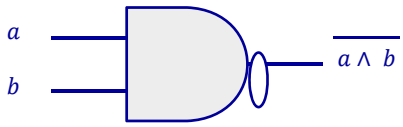
THE NAND GATE

- The NAND Gate is known as NOT-AND gate.
- It is the logical opposite of an AND gate and will produce a 0 output when all inputs are 1.

KAPLAN

DIAGRAM OF NAND GATE

- The input a and b from the left hand side of the diagram must NOT be 1 at the same time for the result to be 1.



13

KAPLAN

THE NOR GATE

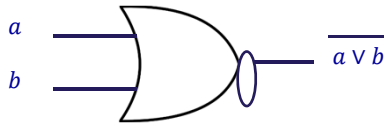
- Like the NAND Gate, the NOR Gate is a NOT-OR Gate.
- Hence only if both of the inputs are 0, the output will be 1. In any other cases, the output is 0.

14

KAPLAN

DIAGRAM OF NOR GATE

- Both inputs a and b from the left hand side of the diagram must be 0 for the result to be 1.



15

KAPLAN

THE XOR GATE

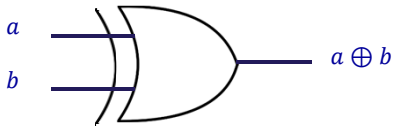
- A special gate known as a XOR Gate (Exclusive OR Gate) gives 1 if and only if one of the inputs is 1.
- If both inputs are 0 or both inputs are 1, then XOR will produce 0.

16

KAPLAN

DIAGRAM OF XOR GATE

- Only ONE of the inputs $\boldsymbol{a}$ and $\boldsymbol{b}$ from the left hand side of the diagram must be 1 for the result to be 1.



17

KAPLAN

TRUTH TABLES

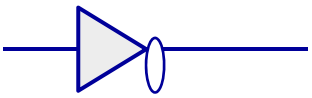
- After learning about the different logic gates, let's take a look at the truth tables for some of them.
- A Truth Table is a map for all possible scenarios in a logic gate or circuit.
- Utilizes 0 or 1 and in some cases T or F.

18

KAPLAN

NOT GATE TRUTH TABLE

a	$\overline{a}$
1	0
0	1

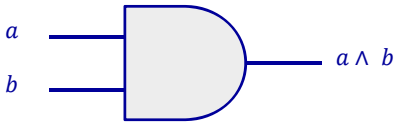


19

KAPLAN

AND GATE TRUTH TABLE

a	b	$a \wedge b$
1	1	1
1	0	0
0	1	0
0	0	0

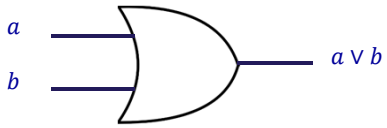


20

KAPLAN

OR GATE TRUTH TABLE

a	b	$a \vee b$
1	1	1
1	0	1
0	1	1
0	0	0

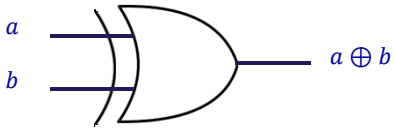


21

KAPLAN

XOR GATE TRUTH TABLE

a	b	$a \oplus b$
1	1	0
1	0	1
0	1	1
0	0	0



22

KAPLAN

TRUTH TABLES FOR NAND, NOR?

Okay, now it is your turn to map the tables for NAND and NOR

KAPLAN

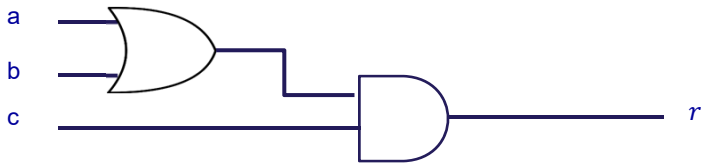
LOGIC CIRCUIT

- A logic circuit is a combination of logic gates and devices such as multiplexers, registers.
- For our study, we will look at combination of logic gates to form a Boolean expression.

KAPLAN

LOGIC CIRCUIT EXAMPLE

- A logic circuit is seen here combining a OR gate with a AND gate.



25

KAPLAN

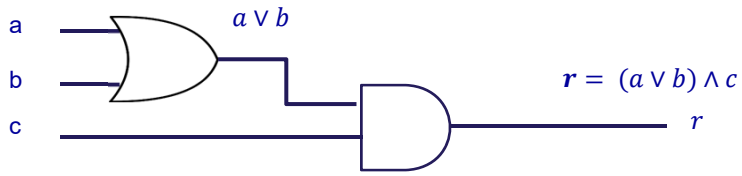
WHAT IS A BOOLEAN EXPRESSION?

- An expression that will produce a Boolean value (True or False).
- In a logic circuit, the combination of the devices can be written as a Boolean expression as well.

KAPLAN

LOGIC CIRCUIT EXAMPLE – BOOLEAN EXPRESSION

• The circuit can be written as a Boolean Expression

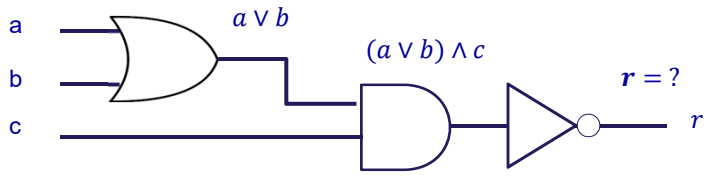


27

KAPLAN

LOGIC CIRCUIT EXAMPLE – BOOLEAN EXPRESSION (2)

• Let us add another NOT gate. What is the resulting Boolean expression?

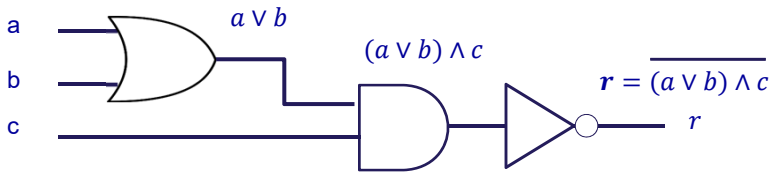


28

KAPLAN

LOGIC CIRCUIT EXAMPLE – BOOLEAN EXPRESSION (3)

- Let us add another NOT gate. What is the resulting Boolean expression?



29

KAPLAN

SUMMARY

- Boolean Algebra has basic operations such as NOT, AND and OR.
- Other logic gates are NAND, NOR and XOR.
- Logic gates can be combined to give a more complex circuit.
- Each logic circuit can be written as a Boolean Expression.

30

KAPLAN

References

- Stallings, W. (2016). Computer organization and architecture: Designing for performance. Hoboken, N.J: Pearson.

KAPLAN

KAPLAN

Computational Mathematics and Computer Architecture

Topic 07

Boolean Algebra II

Learning Outcomes

Upon successful completion of this topic, students should be able to:

- Describe Boolean Expression

WHAT ARE LAWS OF BOOLEAN ALGEBRA

- Laws of Boolean Algebra are sets of rules or laws that are used to define Boolean expressions
- Laws of Boolean Algebra also help to reduce Boolean expressions to simply logic circuits.

KAPLAN

BOOLEAN ALGEBRA LAWS

- Boolean Algebra has a number of laws
  - Associative Laws
  - Commutative Laws
  - Distributive Laws
  - Identity Laws
  - Idempotent Laws
  - Complement Laws
  - Annulment Laws
- Boolean Algebra also has notable theorems
  - De Morgan's Theorems 1 and 2

KAPLAN

ASSOCIATIVE LAWS

- $(a \vee b) \vee c = a \vee (b \vee c)$
- $(a \wedge b) \wedge c = a \wedge (b \wedge c)$

KAPLAN

COMMUTATIVE LAWS

- $a \vee b = b \vee a$
- $a \wedge b = b \wedge a$

KAPLAN

DISTRIBUTIVE LAWS

- $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$
- $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$

KAPLAN

IDENTITY LAWS

- $a \vee 0 = a$
- $a \wedge 1 = a$

KAPLAN

IDEMPOTENT LAWS

- $a \vee a = a$
- $a \wedge a = a$

KAPLAN

COMPLEMENT LAWS

- $a \vee a' = 1$
- $a \wedge a' = 0$

KAPLAN

ANNULMENT LAWS

- $\boldsymbol{a} \vee 1 = \mathbf{1}$
- $\boldsymbol{a} \wedge 0 = \mathbf{0}$

KAPLAN

DE MORGAN THEOREM 1

- $\neg(\boldsymbol{a} \vee \boldsymbol{b}) = \neg\boldsymbol{a} \wedge \neg\boldsymbol{b}$

KAPLAN

DE MORGAN THEOREM 2

- $\neg(a \wedge b) = \neg a \vee \neg b$

KAPLAN

SIMPLIFICATION OF EXPRESSION: EXAMPLE 1

- Prove $X \vee (X \wedge Y) = X$
- $X \vee (X \wedge Y)$

KAPLAN

SIMPLIFICATION OF EXPRESSION: EXAMPLE 1 (2)

- Prove $X \vee (X \wedge Y) = X$

- $X \vee (X \wedge Y)$
- $= (X \wedge 1) \vee (X \wedge Y)$ **(Identity Law)**

15

KAPLAN

SIMPLIFICATION OF EXPRESSION: EXAMPLE 1 (3)

- Prove $X \vee (X \wedge Y) = X$

- $X \vee (X \wedge Y)$
- $= (X \wedge 1) \vee (X \wedge Y)$ **(Identity Law)**
- $= X \wedge (1 \vee Y)$ **(distributive law)**

16

KAPLAN

SIMPLIFICATION OF EXPRESSION: EXAMPLE 1 (4)

- Prove $X \vee (X \wedge Y) = X$

- $X \vee (X \wedge Y)$
- $= (X \wedge 1) \vee (X \wedge Y)$ $(\textbf{Identity Law})$
- $= X \wedge (1 \vee Y)$ **(distributive law)**
- $= X \wedge 1$ **(Identity Law)**

17

KAPLAN

SIMPLIFICATION OF EXPRESSION: EXAMPLE 1 (5)

- Prove $X \vee (X \wedge Y) = X$

- $X \vee (X \wedge Y)$
- $= (X \wedge 1) \vee (X \wedge Y)$ $(\textbf{Identity Law})$
- $= X \wedge (1 \vee Y)$ **(distributive law)**
- $= X \wedge 1$ **(Identity Law)**
- $= X$

$(\textit{This is known as the Absorption Law})$

18

KAPLAN

SIMPLIFICATION OF EXPRESSION: EXAMPLE 2

- Prove $X \vee (\neg X \wedge Y) = X \vee Y$
- $X \vee (\neg X \wedge Y)$

KAPLAN

SIMPLIFICATION OF EXPRESSION: EXAMPLE 2

- Prove $X \vee (\neg X \wedge Y) = X \vee Y$
- $X \vee (\neg X \wedge Y)$
- $= \big(X \vee (X \wedge Y)\big) \vee (\neg X \wedge Y)$ $\quad (Absorption\ Law)$

KAPLAN

SIMPLIFICATION OF EXPRESSION: EXAMPLE 2

- Prove $X \vee (\neg X \wedge Y) = X \vee Y$

- $X \vee (\neg X \wedge Y)$
- $= \big(X \vee (X \wedge Y)\big) \vee (\neg X \wedge Y)$ $(Absorption\ Law)$
- $=\big((X \wedge X) \vee (X \wedge Y)\big) \vee (\neg X \wedge Y)$ $(Indempotent\ Law)$

KAPLAN

SIMPLIFICATION OF EXPRESSION: EXAMPLE 2

- Prove $X \vee (\neg X \wedge Y) = X \vee Y$

- $X \vee (\neg X \wedge Y)$
- $= \big(X \vee (X \wedge Y)\big) \vee (\neg X \wedge Y)$ $(Absorption\ Law)$
- $=((X \wedge X) \vee (X \wedge Y)) \vee (\neg X \wedge Y)$ $(Indempotent\ Law)$
- $=(X \wedge X) \vee (X \wedge Y) \vee (X \wedge \neg X) \vee (\neg X \wedge Y)$ $(Complement\ Law)$

KAPLAN

SIMPLIFICATION OF EXPRESSION: EXAMPLE 2

- Prove $X \vee (\neg X \wedge Y) = X \vee Y$

- $X \vee (\neg X \wedge Y)$
- $= \big(X \vee (X \wedge Y)\big) \vee (\neg X \wedge Y)$ $(Absorption\ Law)$
- $=((X \wedge X) \vee (X \wedge Y)) \vee (\neg X \wedge Y)$ $(Indempotent\ Law)$
- $=(X \wedge X) \vee (X \wedge Y) \vee (X \wedge \neg X) \vee (\neg X \wedge Y)$ $(Complement\ Law)$
- $= \big((X \wedge X) \vee (X \wedge \neg X)\big) \vee \big((X \wedge Y) \vee (\neg X \wedge Y)\big)$ **(Commutative)**

KAPLAN

SIMPLIFICATION OF EXPRESSION : EXAMPLE 2

- Prove $X \vee (\neg X \wedge Y) = X \vee Y$

- $X \vee (\neg X \wedge Y)$
- $= \big(X \vee (X \wedge Y)\big) \vee (\neg X \wedge Y)$ $(Absorption\ Law)$
- $=((X \wedge X) \vee (X \wedge Y)) \vee (\neg X \wedge Y)$ $(Indempotent\ Law)$
- $=(X \wedge X) \vee (X \wedge Y) \vee (X \wedge \neg X) \vee (\neg X \wedge Y)$ $(Complement\ Law)$
- $= \big((X \wedge X) \vee (X \wedge \neg X)\big) \vee \big((X \wedge Y) \vee (\neg X \wedge Y)\big)$ **(Commutative)**
- $= \big(X \wedge (X \vee \neg X)\big) \vee \big(Y \wedge (X \vee \neg X)\big)$ **(distributive law)**

KAPLAN

SIMPLIFICATION OF EXPRESSION: EXAMPLE 2

- Prove $X \vee (\neg X \wedge Y) = X \vee Y$

- $X \vee (\neg X \wedge Y)$
- $= \big(X \vee (X \wedge Y)\big) \vee (\neg X \wedge Y)$ $(Absorption\ Law)$
- $=((X \wedge X) \vee (X \wedge Y)) \vee (\neg X \wedge Y)$ $(Indempotent\ Law)$
- $=(X \wedge X) \vee (X \wedge Y) \vee (X \wedge \neg X) \vee (\neg X \wedge Y)$ $(Complement\ Law)$
- $= \big((X \wedge X) \vee (X \wedge \neg X)\big) \vee \big((X \wedge Y) \vee (\neg X \wedge Y)\big)$ **(Commutative)**
- $= \big(X \wedge (X \vee \neg X)\big) \vee \big(Y \wedge (X \vee \neg X)\big)$ **(distributive law)**
- $= (X \wedge 1) \vee (Y \wedge 1)$ $(\mathbf{Complement\ Law})$

25

KAPLAN

SIMPLIFICATION OF EXPRESSION: EXAMPLE 2

- Prove $X \vee (\neg X \wedge Y) = X \vee Y$

- $X \vee (\neg X \wedge Y)$
- $= \big(X \vee (X \wedge Y)\big) \vee (\neg X \wedge Y)$ $(Absorption\ Law)$
- $=((X \wedge X) \vee (X \wedge Y)) \vee (\neg X \wedge Y)$ $(Indempotent\ Law)$
- $=(X \wedge X) \vee (X \wedge Y) \vee (X \wedge \neg X) \vee (\neg X \wedge Y)$ $(Complement\ Law)$
- $= \big((X \wedge X) \vee (X \wedge \neg X)\big) \vee \big((X \wedge Y) \vee (\neg X \wedge Y)\big)$ **(Commutative)**
- $= \big(X \wedge (X \vee \neg X)\big) \vee \big(Y \wedge (X \vee \neg X)\big)$ **(distributive law)**
- $= (X \wedge 1) \vee (Y \wedge 1)$ $(\mathbf{Complement\ Law})$
- $= X \vee Y$

26

KAPLAN

SUMMARY

- Boolean Algebra has basic laws such as associative laws, complement laws, distributive laws, identity laws commutative laws.
- There are 2 other important results known as De Morgan's laws or Theorems 1 and 2

KAPLAN

References

- Stallings, W. (2016). Computer organization and architecture: Designing for performance. Hoboken, N.J: Pearson.

KAPLAN

KAPLAN

Computational Mathematics and Computer Architecture

Topic 08
MATLAB Introduction, Environment and Functions

Learning Outcomes

Upon successful completion of this topic, students should be able to:

- Describe what MATLAB is
- Describe why MATLAB is widely used in engineering and science
- List the advantages and limitations of MATLAB
- Explain how to formulate problem using a structured program solving approach
- Explain how to launch the MATLAB program
- Explain the use of various MATLAB windows
- Explain the use of matrices and variables
- Describe the various common mathematical functions
- Describe the use of trigonometric functions in MATLAB
- Explain how to compute and use statistical and data analysis functions
- Generate uniform and Gaussian random-number matrices
- List the computational limits of MATLAB
- Explain how to use the special values and functions built into MATLAB

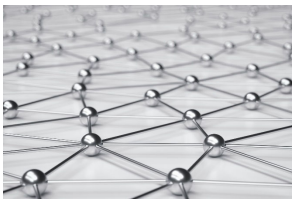
WHAT IS MATLAB?

- MATLAB is a programming platform specially designed for engineers and scientists for various purposes.
- MATLAB as the name goes, Matrix Laboratory means that the language is heavily based on matrix and makes it easy for expression of computational mathematics.

KAPLAN

USES OF MATLAB

- With MATLAB, one can perform:
  - Data Analytics
  - Simulations
  - Machine Learning
  - Deep Learning
  - Image and Video Processing
  - And many more



KAPLAN

USES OF MATLAB – DATA ANALYTICS

- Data Analytics and MATLAB:
  - Data Analytics is a field that allows scientists to explore different datasets and relate them together to make sense of them
  - According to LinkedIn, one of the fastest growing jobs with 5X job growth in 2017 for data scientists
  - Data Analytics are used in various ways such as to predict the efficacy of medicines and survival rates in the field of Medicine

5

KAPLAN

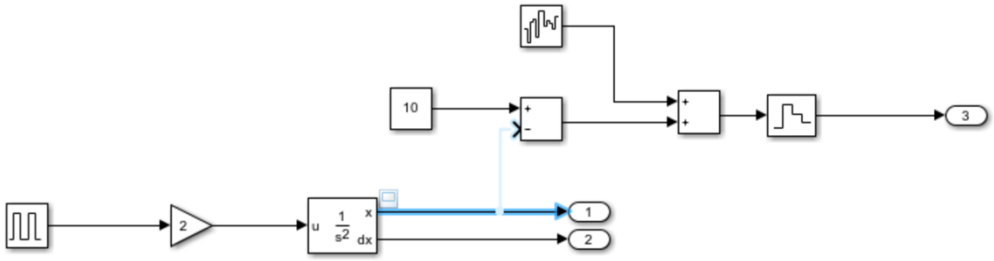
USES OF MATLAB – SIMULATIONS

- Simulations, Simulink and MATLAB:
  - Simulink is a graphical programming environment for modeling, simulating and analyzing multidomain dynamical systems.
  - Simulink is used in various areas for modelling especially in finance, medicine, pharmaceutical for validating and verifying system design
  - Simulink is also used in prototyping by performing simulations before actually building the real item.

6

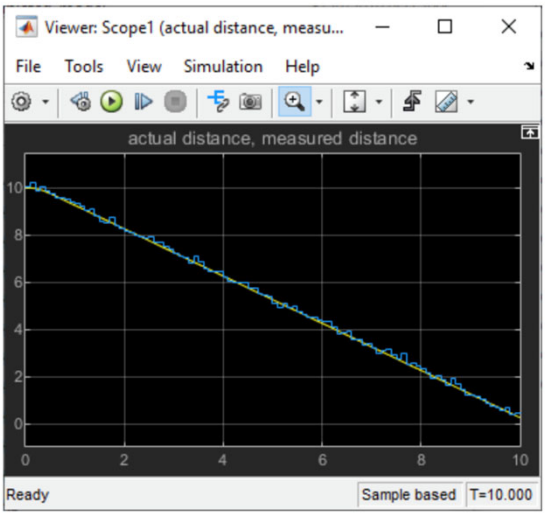
KAPLAN

MATLAB – SIMULINK



KAPLAN

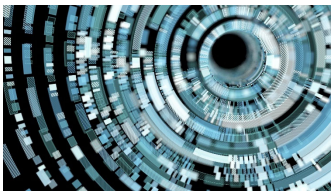
MATLAB – SIMULINK RESULT



KAPLAN

USES OF MATLAB – MACHINE LEARNING

- Machine Learning with MATLAB:
  - Taking data analytics further, machine learning takes data and then learn from it and uses algorithms to forecast the future based on past actions.
  - To facilitate machine learning, there are many algorithms in this area.
  - To harness the power of machine learning, MATLAB provided AutoML to enable scientists to create optimized models



9

KAPLAN

USES OF MATLAB – DEEP LEARNING

- Deep Learning and MATLAB:
  - Deep Learning is a technique in machine learning that teaches computers to learn by example
  - Deep Learning is used in driverless cars / buses / trucks
  - The key technology behind voice enabled technologies
  - In MATLAB, deep learning is enabled by transfer learning approach

10

KAPLAN

MACHINE LEARNING Vs. DEEP LEARNING

Figure 3. Comparing a machine learning approach to categorizing vehicles (left) with deep learning (right).

KAPLAN

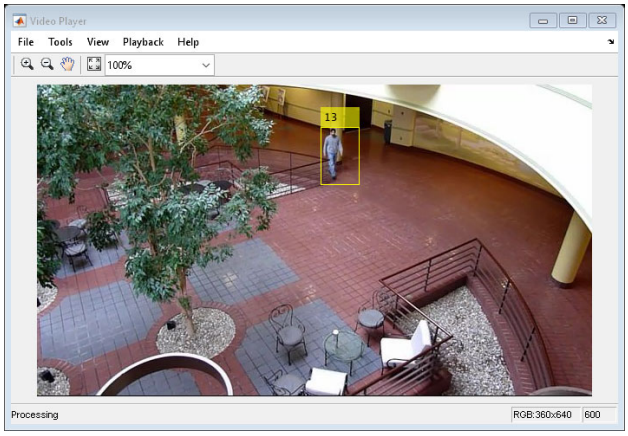
USES OF MATLAB – IMAGE AND VIDEO PROCESSING

- Image and Video Processing MATLAB:
  - In MATLAB, there are tools and algorithms to analyze videos or even write videos
  - Ability to track face and motion
  - The ability to recognize facial features

12

KAPLAN

VIDEO PROCESSING – OBJECT RECOGNITION



KAPLAN

USES OF MATLAB – ADVANTAGES

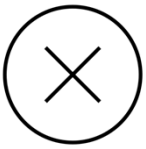
- Advantages of using MATLAB:
  - Easy to learn
  - It is a scripting language, compilation is not needed
  - It has many libraries for engineering purposes
  - It supports modelling
  - It is capable of running across different platforms as long as the system has MATLAB runtime

14

KAPLAN

USES OF MATLAB – LIMITATIONS

- Limitations of MATLAB:
  - MATLAB being a scripting language will not be able to support low level programming such as drivers
  - MATLAB is not free, it costs for every licence
  - MATLAB is slower than a compiled language like every other scripting language



KAPLAN

MATLAB

STRUCTURED PROBLEM SOLVING

KAPLAN

STRUCTURED PROBLEM SOLVING

- A typical problem solving method has 5 steps (D-I-C-I-T):
  - Step 1: Define the Problem
  - Step 2: Identify Inputs and Outputs
  - Step 3: Create an Algorithm
  - Step 4: Implement the Solution
  - Step 5: Test the Solution (Iterate if required)

STRUCTURED PROBLEM SOLVING – DEFINE THE PROBLEM (STEP 1)

- In the “Define the Problem” step,
  - Assess the situation and make sense of what are facts
  - Look at source of the problem
  - Determine in which process the problem lies

STRUCTURED PROBLEM SOLVING – IDENTIFY INPUTS & OUTPUTS (STEP 2)

- In the “Identify Inputs and Outputs” step,
  - Sometimes the problem may not be so clear, and it may be good to examine what inputs are given and what outputs are desired.
  - Inputs are given as data incoming such as keying in username and passwords or image to be processed.
  - Outputs are the desired outcomes such as successful login or detection of a potential crime due to detection of sudden movement of people in the video and heightened sounds in the video.

19

KAPLAN

STRUCTURED PROBLEM SOLVING – CREATE AN ALGORITHM (STEP 3)

- The next step is to Create an Algorithm,
  - -An algorithm is also known as the methodology to solve a problem
  - -While it is not the actual solution itself, creating an algorithm allows errors to be detected early before the creation of the actual solution itself.
  - -Errors detected early in this stage can save a programmer lots of heartache and time from being wasted, and to the corporation a decent amount of money saved due to early detection

20

KAPLAN

STRUCTURED PROBLEM SOLVING – IMPLEMENT THE SOLUTION (STEP 4)

- In this stage ”Implement the Solution”,
  - -The solution could be a physical product or a virtual product.
  - -In this course, the product will be implemented using MATLAB programming language.
  - -Follow the algorithm closely and implement each step stated in the algorithm.

KAPLAN

STRUCTURED PROBLEM SOLVING – TEST THE SOLUTION (STEP 5)

- In this ”final” stage you need to ”Test the Solution”,
  - - It is the final yet may not be the final stage as you need to test the solution actively to determine any problems.
  - - Once problems are detected, iterate through the previous stage(s).
  - - The solution is considered complete only when all tests are error free.

KAPLAN

STRUCTURED PROBLEM SOLVING – AN EXAMPLE!

- We shall look at one example and use the 5-stage approach to solve the problem.
- Problem:
  - Assume that there are only 2 celestial bodies (for example: Earth and the Sun), find the gravitational force F between any 2 celestial bodies.

KAPLAN

STEP 1: DEFINE THE PROBLEM

- In this stage, we note that there are only 2 celestial bodies.
- The problem is to resolve the gravitational force between the 2 celestial bodies.
- After some research, we found that the formula that we are seeking is:

$$F = G\frac{m_1 m_2}{d^2}$$

KAPLAN

STEP 2: IDENTIFY THE INPUTS AND OUTPUTS

$$F = G\frac{m_1 m_2}{d^2}$$

- We can see that output we want is F where F is the gravitational Force between the 2 celestial bodies.
- We know that G has the value of 6.674 30 x 10^{-11} m^3 kg^{-1} s^{-2}
- Hence we need the inputs of the 2 masses m1 and m2 and the distance between the 2 masses (ie. d)

25

KAPLAN

STEP 3: CREATE AN ALGORITHM

- Ask the user for the masses of the 2 celestial bodies in kg
- Ask the user for the distance between the 2 celestial bodies in m.
- Apply the formula and find F

$$F = G\frac{m_1 m_2}{d^2}$$

- Print the result F to the screen

26

KAPLAN

STEP 4: IMPLEMENT THE SOLUTION

```
% Ask for the masses of the 2 celestial bodies in kg
m1 = input('Mass of first body: ');
m2 = input('Mass of second body: ');
% Ask for the distance between the 2 celestial bodies in m.
d = input('Distance between them: ');
G = 6.67430 * 10^-11;
% Apply the formula and find F
F = G*m1*m2/(d^2);

% Print the result F to the screen
disp(F);
```

27

KAPLAN

STEP 5: TEST THE SOLUTION

```
>> lect08
Mass of first body: 2e+30
Mass of second body: 6e+24
Distance between them: 1.5e11
       3.5596e+22
```

Test case 1: Sun and Earth

```
>> lect08
Mass of first body: 6e+24
Mass of second body: 7e+22
Distance between them: 3.8e+8
       1.941278e+20
```

Test case 2: Earth and Moon

28

KAPLAN

STRUCTURED PROBLEM SOLVING – PROBLEM SOLVED

- We have solved the problem!
- Let us visit the MATLAB environment to apply this in a real MATLAB program

KAPLAN

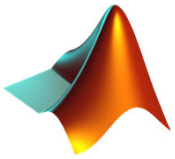
MATLAB

UNDERSTANDING THE MATLAB ENVIRONMENT

KAPLAN

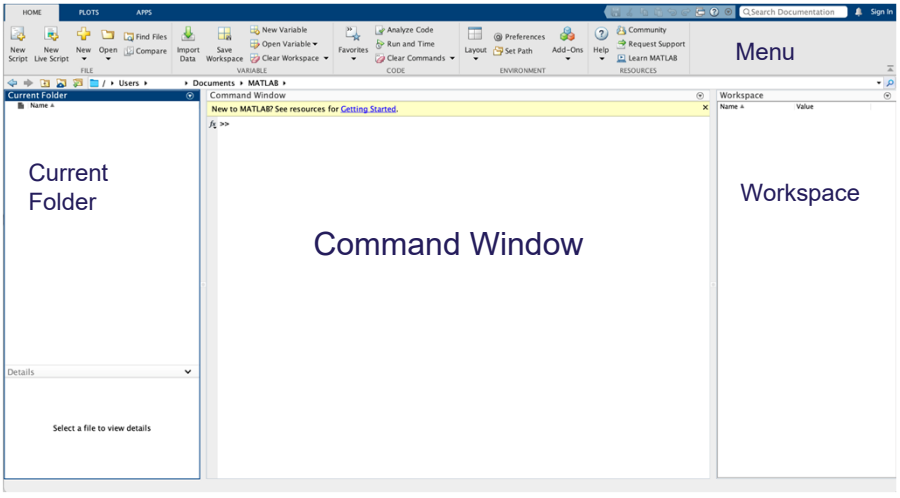
HOW TO LAUNCH MATLAB

- Depending on your system, you can search for ”MATLAB” via the start menu or search
- Alternatively, you can look for the MATLAB icon and click on it to run



KAPLAN

MATLAB R2020b WINDOW



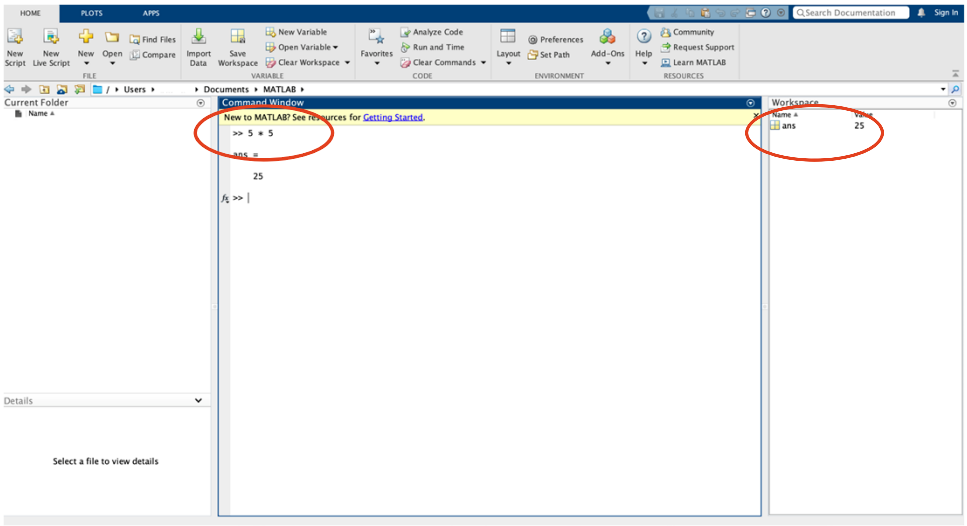
KAPLAN

MATLAB COMMAND WINDOW

- The Command Window allows you to enter commands to run
- It can be as simple as running an arithmetic calculation like 5*5
- When you run a script, the output of the commands will be shown as well
- Let us run 5 * 5 and see the output.

KAPLAN

MATLAB COMMAND WINDOW OUTPUT



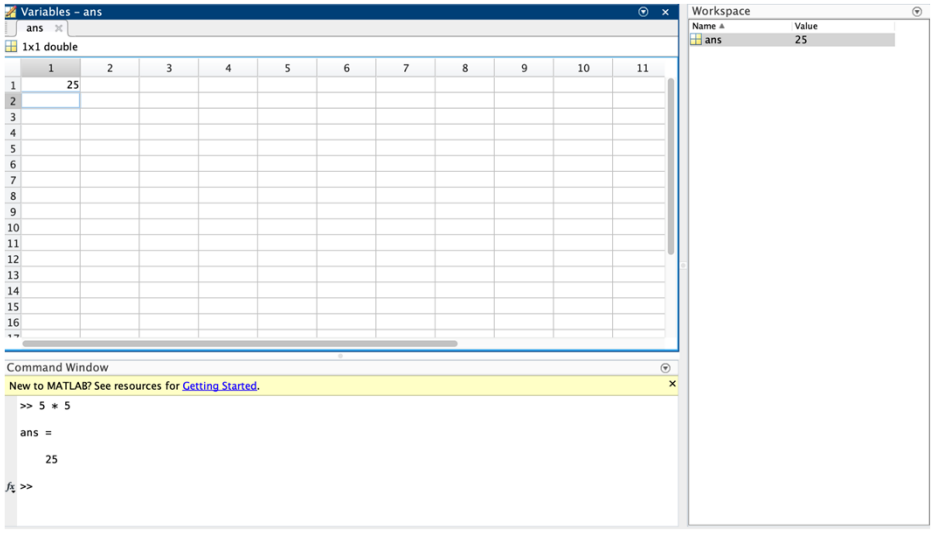
KAPLAN

MATLAB WORKSPACE WINDOW

- Notice that after you run the Command there is a variable “ans”
- The value of ”ans” is the value you obtained from 5 * 5 and is stored generically in the workspace window.
- This is how MATLAB keeps track of the values of your variables.
- Now double click on the variable “ans”

KAPLAN

MATLAB WORKSPACE VARIABLES WINDOW



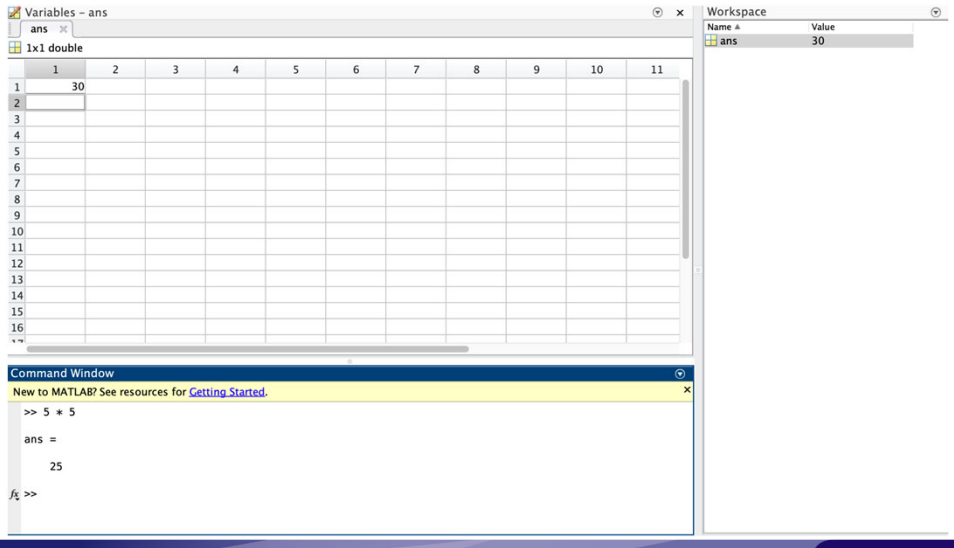
KAPLAN

MATLAB WORKSPACE VARIABLES WINDOW (Previously Known as Document Window)

- You will now reach the Workspace Variables Window (Previously known as Document Window)
- You can alter the value of 25 to 30 and press “Enter” and notice the change

KAPLAN

MATLAB WORKSPACE VARIABLES WINDOW



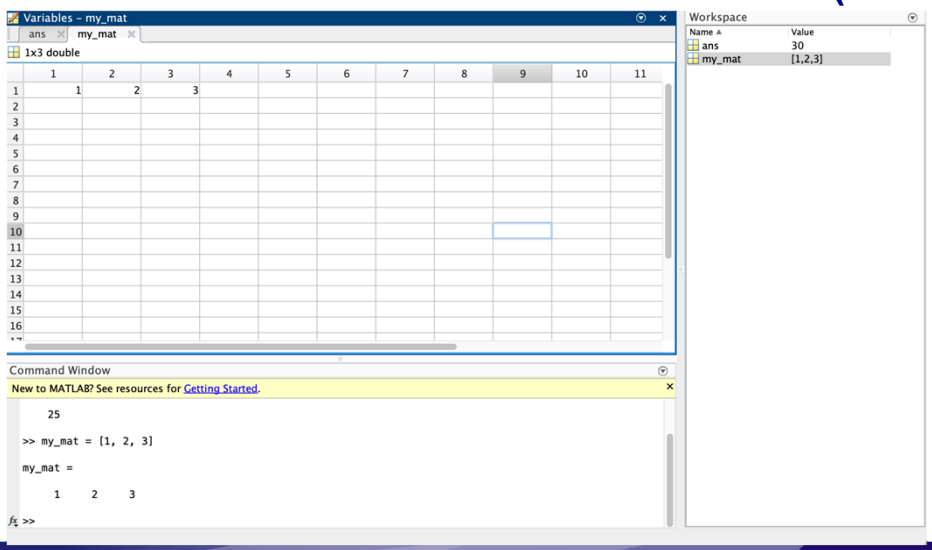
KAPLAN

MATLAB WORKSPACE VARIABLES WINDOW

- Now add another variable by typing my_mat = [1, 2, 3]
- Press ”Enter” and see the variable in the Workspace Window
- Double click on “my_mat” to obtain the same window
- What you just did was to input a variable called ”my_mat” which is a matrix type that can contain multiple values (in this case 3 values)

KAPLAN

MATLAB WORKSPACE VARIABLES WINDOW (MATRIX)



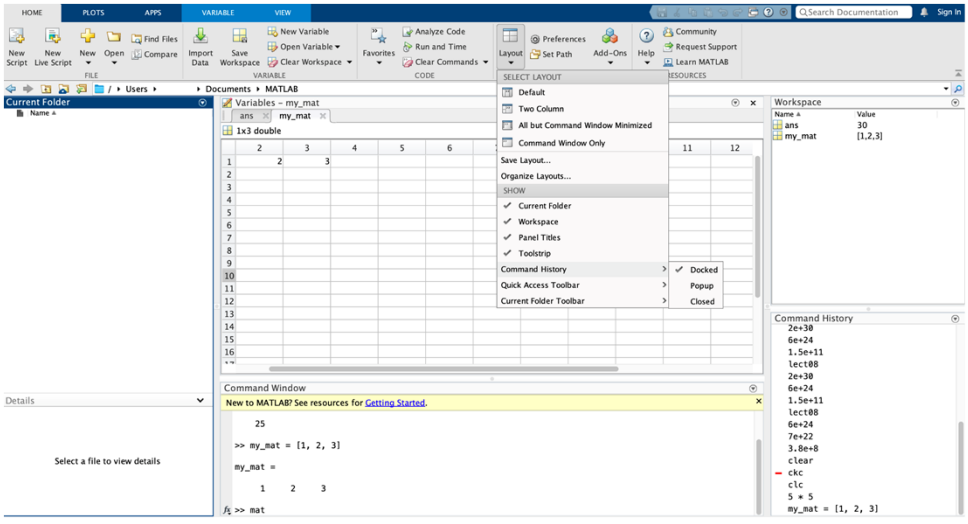
KAPLAN

MATLAB COMMAND HISTORY WINDOW

- Now that I have done what I need to do, I need to refer back to my previous history. How do I do that?
- In MATLAB, you can do that by referring to Command History Window.
- In MATLAB, the Command History Window is pop up window by default. You will need to go to Layout and select Docked
- OR alternatively you could type "commandhistory" in Command Window to show the previous commands

KAPLAN

MATLAB COMMAND HISTORY WINDOW



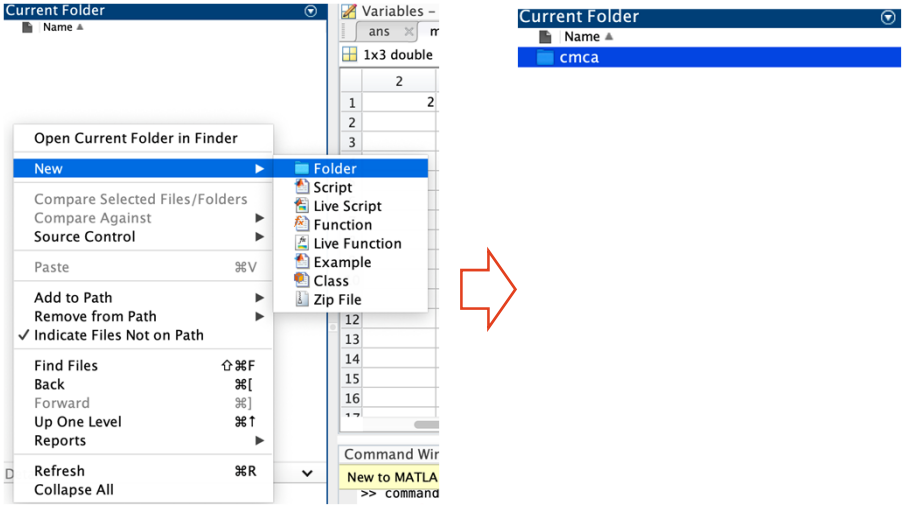
KAPLAN

MATLAB CURRENT FOLDER WINDOW

- The Current Folder shows all the files and folders in the current directory. At the moment it is empty.
- For this example, go to Current Folder Window and right click and create a new Folder “cmca”

KAPLAN

MATLAB CURRENT FOLDER WINDOW



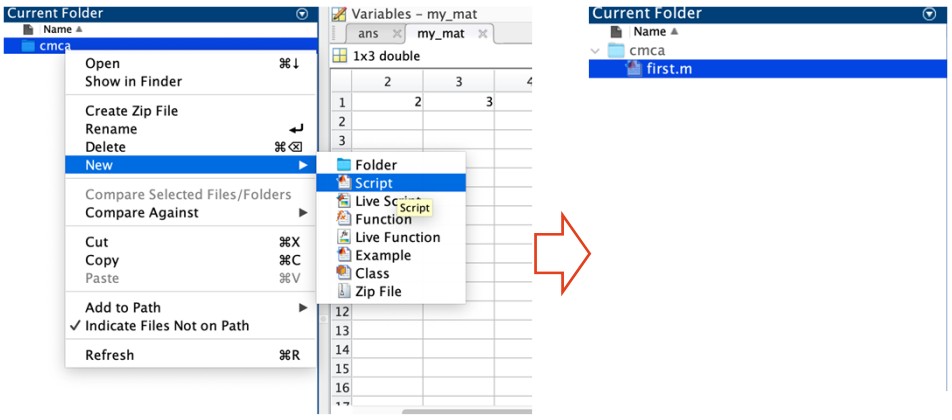
KAPLAN

MATLAB EDIT WINDOW

- In the folder “cmca”, right click and select “New” -> “Script”
- Key in a name eg. “first.m”
- Double click on ”first.m” and you will reach the Edit Window
- This is where you will be writing most of your programs in the future.
- Type in the command to print ”Hello World” in the next few slides and click and run

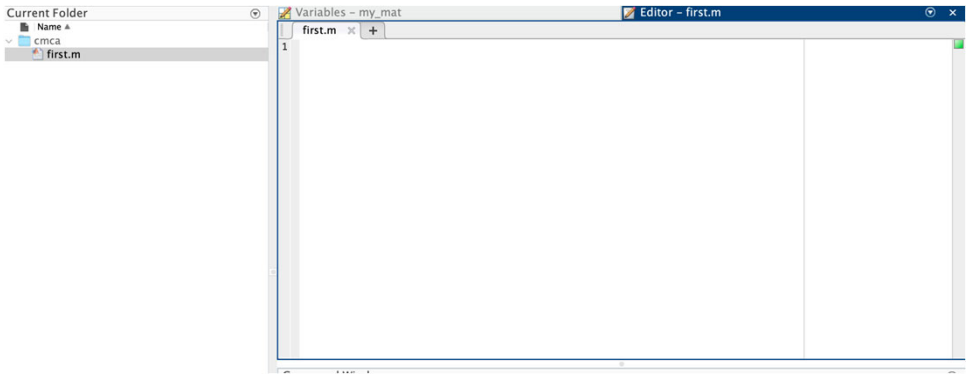
KAPLAN

MATLAB EDIT WINDOW



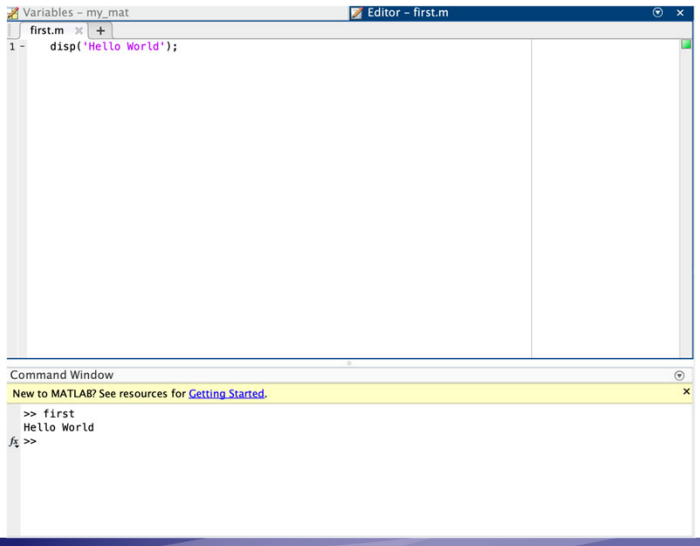
KAPLAN

MATLAB EDIT WINDOW



KAPLAN

MATLAB EDIT WINDOW



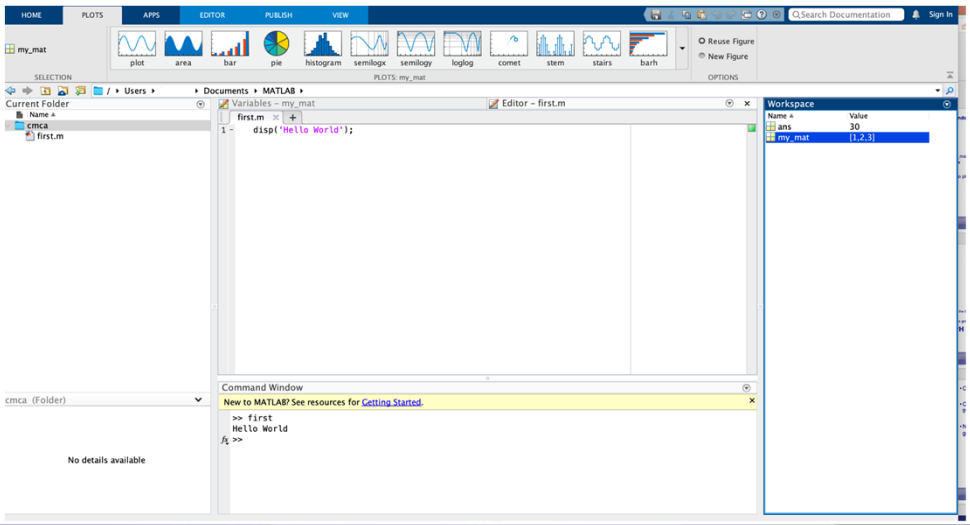
KAPLAN

MATLAB PLOT GALLERY WINDOW

- Choose the Plot Tab
- Choose the variable “my_mat” in the Workspace Window
- Now choose ”plot” chart to plot the graph

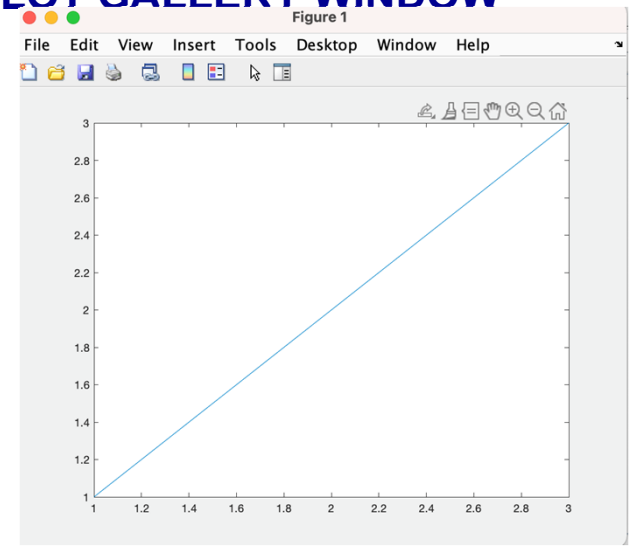
KAPLAN

MATLAB PLOT GALLERY WINDOW



KAPLAN

MATLAB PLOT GALLERY WINDOW



KAPLAN

MATLAB

MATLAB VARIABLES AND MATRIX

KAPLAN

MATLAB VARIABLE AND NAMING RULES

- We have seen variables in action. Let us take a look at what variables are.
- A variable is used to store value such as numbers, Boolean or strings
- Starts with a letter, followed by letters, digits, or underscores.
- Case sensitive
- To find out the maximum length use the ”namelengthmax” command.
- Cannot use keywords

KAPLAN

MATLAB MATRIX

- Previously we also looked at a single dimensional matrix in the previous slides.
- A single dimensional matrix is called a vector.
- Each element in the matrix has an index, starting with 1
- Example of vector:

```
my_vec = [2, 3, 4];
disp (my_vec(3));    % will display 4
```

KAPLAN

MATLAB MATRIX

- A 2 dimensional matrix is defined by rows and columns.
- To access each element, get row followed by the index
- Example of matrix:

```
my_mat = [ 1, 2, 3; 4, 5, 6 ];    % defines 2 rows 3 columns
disp(my_mat(2, 2));               % prints 5
```

KAPLAN

MATLAB

MATLAB FUNCTIONS

KAPLAN

WHAT IS A FUNCTION?

- A function is a self contained module
- A function takes in inputs and gives an output (s)
- A function does one thing and does it well

KAPLAN

FUNCTION: A FIRST LOOK AT MATHEMATICAL FUNCTION

- Consider the following mathematical function y = f(x), where

$$f(x) = x + 3$$

- If x = 1, f(1) = 1 + 3 = 4, hence y = 4
- The input here is 1, and after passing into f(x) where x = 1, the output value is 4, and returned to y
- The value of y now is 4.

KAPLAN

FUNCTION: A FIRST LOOK AT MATHEMATICAL FUNCTION (2)

- If x = 2, f(2) = 2 + 3 = 4, hence y = 5
- If x = 5, f(5) = 5 + 3 = 8, hence y = 8
- We can see the ”return values” of the function f(x) vary on the input values of x
- And y will take on different results when x values are different

KAPLAN

FUNCTION: AN EXAMPLE OF MATLAB FUNCITON

- Consider the following MATLAB function, where

$$y = power(5, 2);$$

- The function name is power and takes in 2 inputs (arguments)
- The first input is the base (5), the second input is the index (2)
- The value of y now is 25.

KAPLAN

FUNCTION: AN EXAMPLE OF MATLAB FUNCTION (2)

- Let us change the values of the same function, where

$$y = power(4, 3);$$

- The first input is now 4 (base), the second input now 3 (index)
- The value of y now is 4 to the power of 3 which gives 64.

KAPLAN

FUNCTION: AN EXAMPLE OF MATLAB function (3)

- Instead of having number 4 as the base, let us use a variable instead

```
x = 6;
y = power(x, 2);
```

- The first input is now x (base) which will evaluate to 6, the second input now 2 (index)
- The value of y now is 6 to the power of 2 which gives 36.

KAPLAN

FUNCTION: AN EXAMPLE OF MATLAB FUNCITON (4)

- Now replace the single variable with a single dimensional matrix (vector)

```
x = [1, 2, 3];
y = power(x, 2);
```

- The first input is now a vector
- The value of y now is [1, 4, 9] where the power of 2 is applied to all elements in the vector.

63

KAPLAN

FUNCTION: AN EXAMPLE OF MATLAB FUNCTION (5)

- Now let x be a 2 dimensional matrix instead

```
x = [1, 2, 3; 4, 5, 6];
y = power(x, 2);
```

- The first input is now a matrix
- The value of y now is [1, 4, 9; 16, 25, 36] where the power of 2 is applied to all elements in the 2 by 3 matrix.

64

KAPLAN

MATLAB

BASIC MATHEMATICAL FUNCTIONS

KAPLAN

MATLAB BASIC MATHEMATICAL FUNCTIONS

- Basic Mathematical functions involve the following:
  - Logarithms
  - Exponentials
  - Rounding
  - Discrete Mathematics

- In our discussion we will look at various math functions and their examples

KAPLAN

MATLAB BASIC MATHEMATICAL FUNCTIONS

abs(x)	Return absolute value of x abs(-2) gives 2
exp(x)	Return value of e to the power of x where e is about 2.71828 exp(3) gives approximately 20.0855
log(x)	Get natural logarithm of x (base is e) log(100) gives approximately 4.6052
log10(x)	Get the logarithm of x base 10 log10(100) gives 2

KAPLAN

MATLAB BASIC MATHEMATICAL FUNCTIONS (2)

nthroot(x, n)	Get the nth root value of x nthroot(9, 2) gives 3
rem(x, y)	Return remainder of x divided by y rem(13, 4) gives 1
sign(x)	Y = sign(x) returns an array Y the same size as x, where each element of Y is: 1 if the corresponding element of x > 0. 0 if the corresponding element of x == 0. -1 if the corresponding element of x < 0.
sqrt(x)	Return square root of x sqrt(9) gives 3

KAPLAN

MATLAB BASIC MATHEMATICAL FUNCTIONS – ROUNDING

ceil(x)	Return the value of x rounded toward +ve infinity ceil(2.1) gives 3 ceil(2.9) gives 3 ceil(-2.1) gives -2 ceil(-2.9) gives -2
fix(x)	Return the value of x rounded toward zero fix(2.1) gives 2 fix(2.9) gives 2 fix(-2.1) gives -2 fix(-2.9) gives -2

KAPLAN

MATLAB BASIC MATHEMATICAL FUNCTIONS – ROUNDING (2)

floor(x)	Return the value of x rounded toward -ve infinity floor(2.1) gives 2 floor(2.9) gives 2 floor(-2.1) gives -3 floor(-2.9) gives -3
round(x)	Return the value of x rounded to nearest integer round(2.1) gives 2 round(2.9) gives 3 round(-2.1) gives -2 round(-2.9) gives -3

KAPLAN

MATLAB BASIC MATHEMATICAL FUNCTIONS – DISCRETE

factor(x)	Return prime factors of x factor(20) gives 2, 2, 5
factorial(x)	Return the factorial of x factorial(5) gives 120
gcd(x, y)	Find the greatest common denominator of x and y gcd(20, 30) gives 10
lcm(x, y)	Find the least common multiple of x and y lcm(20, 30) gives 60

KAPLAN

MATLAB BASIC MATHEMATICAL FUNCTIONS – DISCRETE (2)

isprime(x)	Return 1 if x is a prime number, 0 otherwise isprime(17) gives 1
primes(x)	Return all prime numbers less than or equal to x primes(20) gives 2, 3, 5, 7, 11, 13, 17, 19
rats(x)	Return the fraction representation of x rats(2.25) gives 9/4

KAPLAN

MATLAB BASIC MATHEMATICAL FUNCTIONS – EXAMPLE

- Example
  - -Supposed you havAe 17 sweets and you need to put 3 sweets per bag, how many bags will you have? How many sweets are left over?

  - -Suggested Solution:

```
num_sweets = 17;
sweets_per_bag = 3;
num_bags = num_sweets/sweets_per_bag;
disp(floor(num_bags));
num_left_over = rem(17, 3);
disp(num_left_over);
```


KAPLAN

MATLAB

TRIGONOMETRIC FUNCTIONS

KAPLAN

MATLAB TRIGONOMETRIC FUNCTIONS

- Trigonometric Functions:
  - MATLAB has a number of Trigonometric Functions that are intuitively easy to use
  - Basic trigonometric functions such as sine, cosine, tangent or hyperbolic sine, hyperbolic cosine or even inverse sine functions are supported by MATLAB
  - We will look at an example of using a sine function

KAPLAN

MATLAB TRIGONOMETIC FUNCTIONS – SINE

- Using MATLAB Sine Function

-MATLAB Sine Function takes in a single argument in radians and return you the value

-Let's apply sin() onto a single value

-Example:

```
x = pi/4
y = sin(x);
disp(x);    % shows 0.7071
```


KAPLAN

MATLAB TRIGONOMETRIC FUNCTIONS – SINE (2)

- Using MATLAB Sine Function

  -You can also apply the sine function or other trigonometric functions on a matrix

  -Example:

```
x = [0, pi/4, pi/2, 3*pi/4, pi];
y = sin(x);
disp(y);    % 0    0.7071    1.0000    0.7071    0.0000
```

77

KAPLAN

MATLAB TRIGONOETRIC FUNCTIONS TABLE

sin	asin	sinh	asinh	
cos	acos	cosh	acosh	
tan	atan	tanh	atanh	atan2

KAPLAN

MATLAB

STATISTICAL AND DATA ANALYSIS

KAPLAN

MATLAB STATISTICAL AND DATA ANALYSIS FUNCTIONS

- MATLAB offers a rich library with functions to find out the biggest value in a collection, the average of all the values in a collection etc.
- MATLAB does this by allow one to pass in a vector or matrix into the function and the function will return the correct values to the caller.

KAPLAN

MATLAB STATISTICAL AND DATA ANALYSIS FUNCTIONS (2)

max(x)	Return biggest value in each column in matrix x max([2, 3, 5, 4]) gives 5 max([1, 3, 5; 6, 4, 2]) gives 6 4 5 **Example:** max([1, 3, 5; 6, 4, 2]); [val, pos] = max([1, 3, 5; 6, 4, 2]); disp(val); % 6 4 5 disp(pos); % 2 2 1

KAPLAN

MATLAB STATISTICAL AND DATA ANALYSIS FUNCTIONS (3)

min(x)	Return smallest value in each column in matrix x min([2, 3, 5, 4]) gives 2 min([1, 3, 5; 6, 4, 2]) gives 1 3 2 **Example:** min([1, 3, 5; 6, 4, 2]); [val, pos] = min([1, 3, 5; 6, 4, 2]); disp(val); % 1 3 2 disp(pos); % 1 1 2

KAPLAN

MATLAB STATISTICAL AND DATA ANALYSIS FUNCTION (4)

mean(x)	Return average value in each column in matrix x mean([2, 3, 5, 4]) gives 3.5 mean([3, 1, 5; 6, 4, 2]) gives 4.5000 2.5000 3.5000
median(x)	Return middle value in each column in matrix x if number of elements is odd or average of 2 middle values if number of elements is even median([2, 3, 1, 5]) gives 2.5 median([3, 1, 5; 6, 4, 2; 3, 4, 1]) gives 3 4 2

KAPLAN

MATLAB STATISTICAL AND DATA ANALYSIS FUNCTION (5)

std(x)	Return standard deviation value in each column in matrix x std([2, 3, 5, 4]) gives 1.2910 std([6, 5, 4; 1, 2, 3]) gives 3.5355 2.1213 0.7071
var(x)	Return variance value in each column in matrix x var([2, 3, 5, 4]) gives 1.6667 var([6, 5, 4; 1, 2, 3]) gives 12.5000 4.5000 0.5000

KAPLAN

MATLAB SORTING AND SIZE FUNCTIONS

- MATLAB has sorting functions that sorts the matrices by row or by column
- In addition to sorting functions, MATLAB also allows the user to find out the dimensions of its matrices

KAPLAN

MATLAB SORTING AND SIZE FUNCTIONS (2)

`sort(x)`	Sort the elements of a matrix x in ascending order. sort([2, 1, 5, 4]) gives 1 2 4 5 sort([6, 2, 5; 1, 3, 4]) gives 1 2 4 6 3 5
`sortrows(x)`	Sort the rows of matrix x sortrows([6, 2, 5; 1, 3, 4]) gives 1 3 4 6 2 5

KAPLAN

MATLAB SORTING AND SIZE FUNCTIONS (3)

`size(x)`	Return the dimensions of matrix x (rows and columns) `size([2, 1, 5, 4])` gives 1 4 `size([6, 2, 5; 1, 3, 4])` gives 2 3
`length(x)`	Return the largest dimension of matrix x (the larger of rows and columns) `length([6, 2, 5; 1, 3, 4])` gives 3

KAPLAN

MATLAB

GENERATING UNIFORM AND GAUSSIAN RANDOM MATRICES

KAPLAN

MATLAB UNIFORM RANDOM NUMBERS

- Random numbers are useful in simulations
- MATLAB allows creation of matrices using random numbers through 2 methods:
  - -Uniform Random Numbers
  - -Gaussian Random Numbers
- Gaussian Random Numbers have normal distribution

KAPLAN

MATLAB UNIFORM RANDOM NUMBERS

`rand(n)`	`Return n by n matrix of values between 0 and 1`
	
	`rand(2) gives`
	`0.2920  0.0155`
	`0.4317  0.9841`

`rand(m, n)`	`Return m by n matrix of values between 0 and 1`
	
	`rand(2, 3) gives`
	`0.9203  0.7379  0.4228`
	`0.0527  0.2691  0.5479`

KAPLAN

MATLAB UNIFORM RANDOM NUMBERS (2)

`randi(n)`	Return an array containing integer values drawn from the discrete uniform distribution starting from 1 to n randi(3) may give 1, 2 or 3
`randi([x y], m, n)`	Return a matrix of dimensions m by n containing integer values drawn from the discrete uniform distribution starting x to y randi([2 9], 2, 3) may give 7 9 8 3 4 3

KAPLAN

MATLAB GAUSSIAN RANDOM NUMBERS

`randn(n)`	Return a n by n matrix of each value being Gaussian (or normal) random number with a mean of 0 and a variance of 1. randi(3) may give 0.1202 -0.9870 -0.6039 0.5712 0.7596 0.1769 0.4128 -0.6572 -0.3075
`randn(m, n)`	Return a m by n matrix of each value being a Gaussian (or normal) random number with a mean of 0 and a variance of 1. randn(2, 3) may give -0.1318 1.0468 0.3277 0.5954 -0.1980 -0.2383

KAPLAN

MATLAB

LIMITS AND MATLAB PRE-DEFINED VALUES

KAPLAN

MATLAB – LIMITS

- In the system, there are computational limits. In MATLAB, you can ask them to be returned to you using constants

- The constants are:

```
-realmax (largest float, 1.7977e+308)
-realmin (smallest float, 2.2251e-308)
-intmax (largest integer, 2147483647)
-intmin (smallest integer, -2147483648)
```

KAPLAN

MATLAB – PRE-DEFINED VALUES

- In most systems there will be pre-defined values. MATLAB like most systems have pre-defined values which we have seen earlier (pi)
- Pre-defined values and functions in MATLAB can be replaced but should be avoided to have unintended consequences.
- We shall look at the pre-defined values in the following slides

KAPLAN

MATLAB – PRE-DEFINED VALUES

pi	Mathematical constant Pi of value 3.1416
i, j	Imaginary numbers
Inf	Infinity, occurs when there is an error such as number divided by zero
NaN	Not a number, occurs when the result of a calculation is undefined
clock	Current time in array format
date	Current date in string format
eps	The distance between 1 and next larger double precision floating point number

KAPLAN

MATLAB – PRE-DEFINED VALUES EXAMPLE

- Using some MATLAB Pre-defined values

```
x = 0;
y = 0;
z = 3;
disp(x/y);   % NaN
disp(z/y);   % Inf
disp(date);  % 24-12-2020
```


KAPLAN

SUMMARY

- MATLAB is used in many areas of engineering and different fields such as medicine and finance.
- MATLAB is easy to use and has many libraries
- Structured Problem Solving can be used in conjunction with MATLAB to solve daily problems or more sophisticated engineering problems
- MATLAB uses variables and more notably matrices to help track data in applications
- MATLAB has many in-built functions for various mathematical functions
- MATLAB also has functions for statistical and data analysis
- Like other languages MATLAB has pre-defined values that should not be replaced

KAPLAN

References

- Professor Pogge, R. (2007). An Introduction to Solar System Astronomy: The Apple and the Moon. Hoboken. http://www.astronomy.ohio-state.edu/~pogge/Ast161/Unit4/gravity.html

- MathWorks (n.d) MATLAB https://www.mathworks.com/products/matlab.html

KAPLAN

Computational Mathematics and Computer Architecture

Topic 09
Plotting

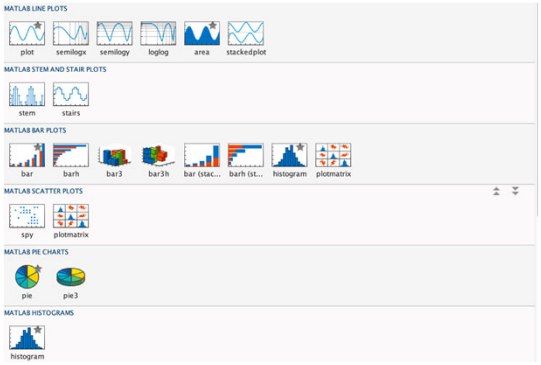
Learning Outcomes

Upon successful completion of this topic, students should be able to:

- Describe the use of Plotting in MATLAB
- Describe how to generate XY plot.
- Understand and generate different types of plots
- Create 3D Plots
- Use Sub plots

WHY USE MATLAB CHART PLOTS?

- MATLAB has a vast gallery to display data graphically
- There are many standard plots to choose from.

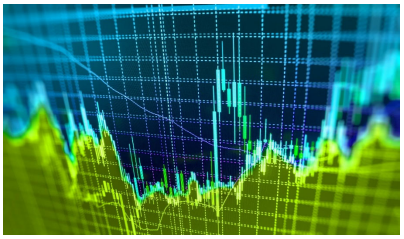


3

KAPLAN

TYPES OF MATLAB CHARTS

- There are many types of MATLAB Charts such as:
  - Line 2D Plot
  - Histogram
  - Pie Chart
  - Scatter Plot
  - Sub Plots
  - Line 3D Plot



4

KAPLAN

MATLAB CHARTS – LINE 2D PLOT

- The simplest and commonly used plot is the 2D Plot:
  - Horizontal axis is usually independent variable such as time
  - Vertical axis is the y-axis, usually the dependent variable
  - To plot simply use plot() function

5

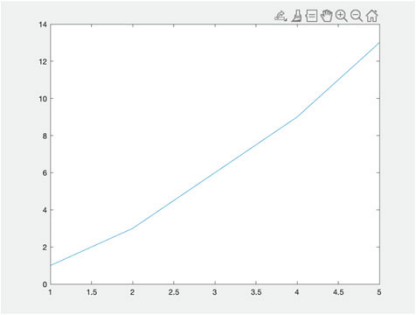
KAPLAN

MATLAB CHARTS – LINE 2D PLOT (2)

- Let us take a look at some simple examples:

```
y = [1, 3, 6, 9, 13];

plot(y);
```

- We can see that with a single vector, the x is defined as 1, 2 ... for each of the elements in y



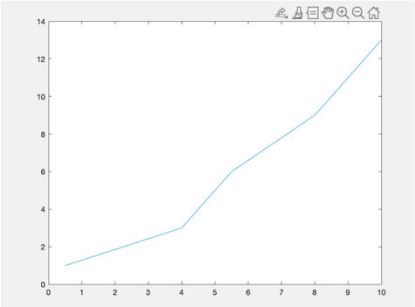
6

KAPLAN

MATLAB CHARTS – LINE 2D PLOT

- We will add a vector for the x-axis:

```
x = [0.5, 4, 5.5, 8, 10];
y = [1, 3, 6, 9, 13];

plot(x, y);
```



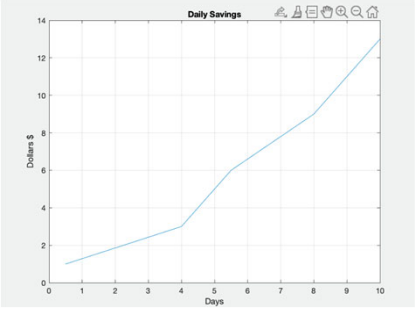
7

KAPLAN

MATLAB CHART – LINE 2D (4)

- Now add a title, x label, y label and a grid:

```
x = [0.5, 4, 5.5, 8, 10];
y = [1, 3, 6, 9, 13];

plot(x, y);

grid on;
title('Daily Savings');
xlabel('Days');
ylabel('Dollars $');
```



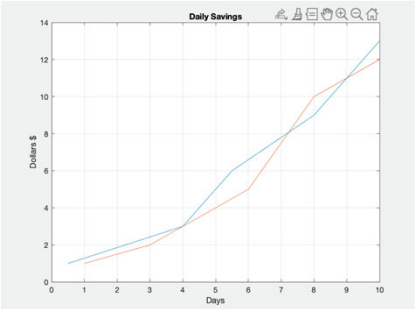
8

KAPLAN

MATLAB CHARTS – LINE 2D PLOT (5)

- Okay we need to compare with the previous chart, let us add another set of values and put hold on to wait for the 2nd chart to be plotted:

```
... % previous code
hold on;
x2 = [1, 3, 6, 8, 10];
y2 = [1, 2, 5, 10, 12];

plot(x2, y2);
```

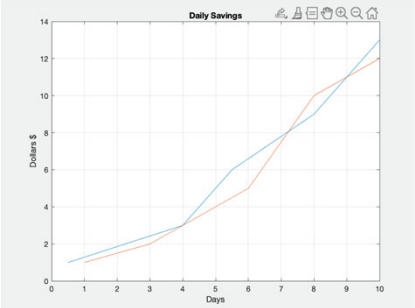


KAPLAN

MATLAB CHARTS – LINE 2D (6)

- Alternatively, we can plot them together without using hold on or another plot statement:

```
x = [0.5, 4, 5.5, 8, 10];
y = [1, 3, 6, 9, 13];
x2 = [1, 3, 6, 8, 10];
y2 = [1, 2, 5, 10, 12];

plot(x, y, x2, y2);
... % other code
```



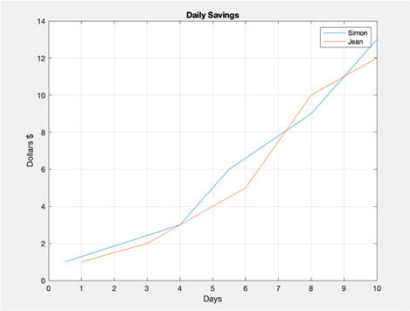
KAPLAN

MATLAB CHARTS – LINE 2D PLOT (7)

- We can see it is difficult to see what the lines mean, so it would be good to add a legend:

```
... % previous code
legend({'Simon', 'Jean'});
```

- We can see that the legend could be covering the chart, so we can shift it.



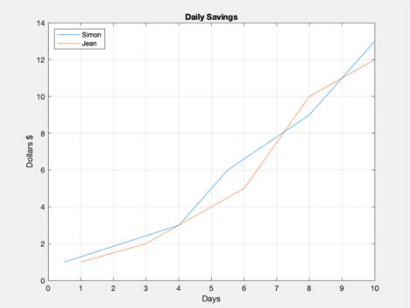
KAPLAN

MATLAB CHARTS – LINE 2D PLOT (8)

- Now change the code to add the location

```
... % previous code
legend({'Simon',
'Jean'},'Location','northwest');
```

- We can see that the legend has shifted to the top left



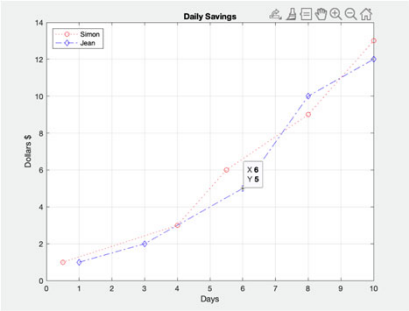
KAPLAN

MATLAB CHARTS – LINE 2D PLOT (9)

- You may want to change the look and colour of the charts plotted:

```
... % previous code
plot(x, y, ':or', x2, y2, '-.db');
```

- The first graph is now red dotted line with circle markers while the second graph is now a blue dash dot line with diamond markers



13

KAPLAN

MATLAB CHARTS – LINE 2D PLOT (10) FULL CODE LISTING

```
x = [0.5, 4, 5.5, 8, 10];
y = [1, 3, 6, 9, 13];

x2 = [1, 3, 6, 8, 10];
y2 = [1, 2, 5, 10, 12];

plot(x, y, ':or', x2, y2, '-.db');

grid on;
title('Daily Savings');
xlabel('Days');
ylabel('Dollars $');

legend({'Simon', 'Jean'},'Location','northwest');
```

14

KAPLAN

MATLAB – LINE 2D PLOT CHOICES

Line Style	Description
`-`	Solid line
`--`	Dashed line
`:`	Dotted line
`-.`	Dash-dot line

Color	Description
`y`	yellow
`m`	magenta
`c`	cyan
`r`	red
`g`	green
`b`	blue
`w`	white
`k`	black

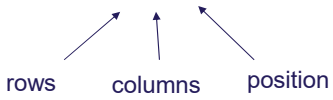
Marker	Description
`'o'`	Circle
`'+'`	Plus sign
`'*'`	Asterisk
`'.'`	Point
`'x'`	Cross
`'_'`	Horizontal line
`'|'`	Vertical line
`'s'`	Square
`'d'`	Diamond
`'^'`	Upward-pointing triangle
`'v'`	Downward-pointing triangle
`'>'`	Right-pointing triangle
`'<'`	Left-pointing triangle
`'p'`	Pentagram
`'h'`	Hexagram

KAPLAN

MATLAB SUBPLOTS

- Sometimes there is a need to display multiple charts together.
- In such a case we use subplots

```
subplot(m, n, p)
```

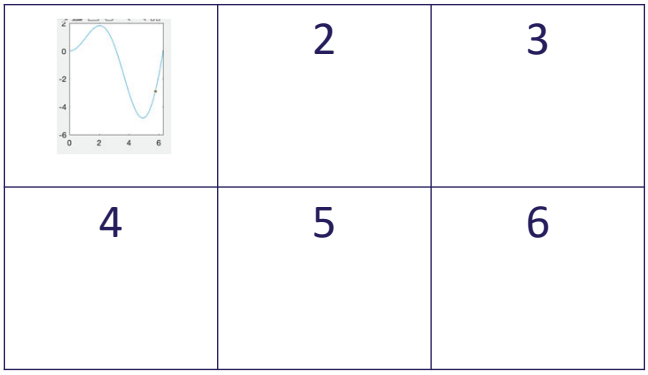


16

KAPLAN

MATLAB SUBPLOTS (2)

- `subplot(2, 3, 1)` – means 2 rows, 3 columns, position 1



KAPLAN

MATLAB SUBPLOTS (3) FULL CODE LISTING PART 1

```
x = 0:pi/30:2*pi;
y = x .* sin(x);
z = x .* cos(x);
sales = [3, 2, 7, 4, 5, 10, 1];
[X,Y] = meshgrid(-6:.5:6);
Z = X .* sin(Y);

subplot(2, 3, 1);
plot(x, y);
grid on;
```

18

KAPLAN

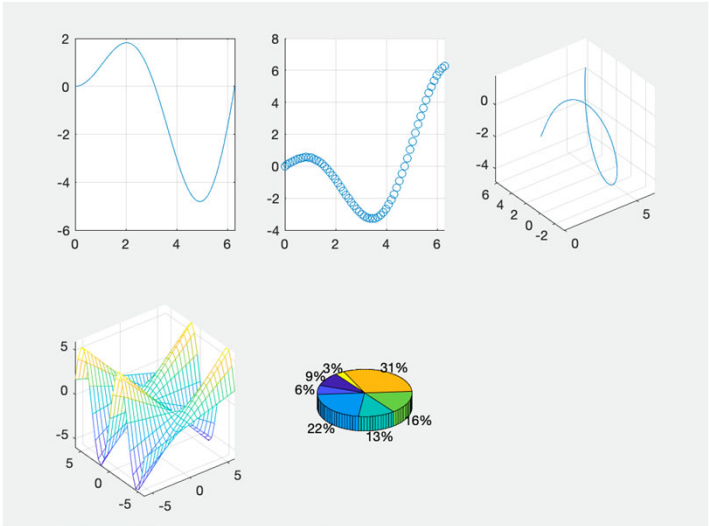
MATLAB SUBPLOTS (4) FULL CODE LISTING PART 2

```
subplot(2, 3, 2);
scatter(x, z);
grid on;

subplot(2, 3, 3);
plot3(x, z, y);
grid on;

subplot(2, 3, 4);
mesh(X, Y, Z);

subplot(2, 3, 5);
pie3(sales);
```

19

KAPLAN

MATLAB SUBPLOTS (5) OUTPUT



20

KAPLAN

SUMMARY

- MATLAB's strength lies not only on its data analytics modules but also on Charting.
- MATLAB has different charts such as XY plots, scatter plots, Histogram, PIE charts and is a very powerful charting tool.

KAPLAN

References

- MathWorks (n.d) MATLAB https://www.mathworks.com/products/matlab.html

KAPLAN

KAPLAN

Computational Mathematics and Computer Architecture

Topic 10
MATLAB Selection and Repetition

Topic 10 – MATLAB Selection and Repetition

Upon successful completion of this topic, students should be able to:

- Describe the use of control structures
- Describe the use of sequence statements
- Describe how to use the selection statements
- Describe how to use repetition statements

MATLAB

CONTROL STRUCTURES

KAPLAN

MATLAB – CONTROL STRUCTURES

- Programs often requires examining conditions to decide on which course of action to take.
- This code base that examines the conditions and therefore changes the flow of the program is known as control structures.

KAPLAN

MATLAB – CONTROL STRUCTURES (2)

- A control structure is implemented to direct the flow of the program using one of the following:

  -Sequence statements
    -Sequential flow

  -Selection statements
    -Using if-else, switch

  -Repetition statements
    -Using while and for

5

KAPLAN

MATLAB

SEQUENCE STATEMENTS

KAPLAN

MATLAB – SEQUENCE STATEMENTS

- Sequence statements are the default mode of operation for any program. It is not implemented with any special keywords.
- Sequence statements can be thought of as a cooking recipe that follows the recipe step by step

-Example:

```
Step 1
Stir sugar, cream, and milk into a saucepan over low heat until sugar has
dissolved.

Step 2
Transfer cream mixture to a pourable container such as a large measuring cup.
```


KAPLAN

MATLAB – SEQUENCE STATEMENTS

- An example of sequence statements in program is often seen as one line being executed after another

-Example:

```
total = 50 + 30 + 45;
average = total / 3;
fprintf("Average of 3 numbers is: %d\n", average);
```


KAPLAN

MATLAB

SELECTION STATEMENTS

KAPLAN

MATLAB – SELECTION STATEMENTS

- A MATLAB program has Selection statements using one of the following structure:

  -if statement
  -if … else statement
  -if … elseif … else statement
  -switch … case

10

KAPLAN

MATLAB – SELECTION STATEMENTS – IF STATEMENT

- A single if statement is used if a condition applies and if nothing else will be applied if the condition is not met:

-An example is when a person is a member and he/she is entitled to say 10% discount. If the person is not a member, then the discount does not apply.

```
price = 100;
is_member = true;
if is_member
   price = price * 0.9;
end
fprintf('Price to pay is %.2f\n', price);
```


KAPLAN

MATLAB – SELECTION STATEMENTS – IF STATEMENTS

- An if-else statement is used if a condition applies, an action must be performed and if the condition does not apply, another action must be performed:

-An example is trying to determine whether the person is eligible to start work in Singapore.

```
age = 16;
if age > 16
    fprintf('You are %d and ready to work!\n', age);
else
    fprintf('No work for you as you are only %d\n', age);
end
```


KAPLAN

MATLAB – SELECTION STATEMENTS – IF – ELSEIF STATEMENT

- An if-elseif statement is used if there could be multiple conditions to be tested but only 1 set of actions can be executed.
  -An example is trying to determine the grade of a Special paper taken in 'Advanced Levels'

```
score = 75;
if score > 75 && score <= 100
    fprintf('Grade is Distinction!\n');
elseif score >= 50 && score <= 75
    fprintf('Grade is Merit\n');
elseif score >= 0 && score < 50
    fprintf('Grade is Fail\n');
else
    fprintf('Invalid score!\n');
end
```

13

KAPLAN

MATLAB – SELECTION STATEMENT – SWITCH – CASE STATEMENT

- A switch-case statement is sometimes used over if-else statements due to readability.
- It would be intuitive to use switch-case if you are testing a single variable and it has discrete values.
  - An example would be trying to determine the discount based on the number of rides to be taken

14

KAPLAN

MATLAB – SELECTION STATEMENTS – SWITCH – CASE STATEMENT (2)

```
num_rides = 3;
discount;
switch (num_rides)
    case 1
        discount = 10;
    case { 2, 3 }
        discount = 20;
    case 4
        discount = 30;
    otherwise
        discount = 5;
end
disp(discount);
```

15

KAPLAN

MATLAB

REPETITION STATEMENTS

KAPLAN

MATLAB – REPETITION STATEMENTS

- A MATLAB program can repeat statements using one of the following repetition structures:

  -for... end

  -while … end

KAPLAN

MATLAB – FOR LOOP – KNOWN NUMBER OF REPETITIONS

- A for loop is used in situations where the number of repetitions is known:
  -One such example is to print all numbers in a range (1 to 5)

```
for n=1:5
    fprintf('%d \n', n);   % 1 2 3 4 5
end
```

-Another such example is to print all numbers in a range (1 to 20), skipping 2 numbers in between

```
for n=1:3:20
    fprintf('%d \n', n);   % 1 4 7 10 13 16 19
end
```


KAPLAN

MATLAB – FOR LOOP – KNOWN NUMBER OF REPETITIONS (2)

- A for loop can also be used with a vector:

  -Example with a vector

```
vec = [1, 3, 8, 5];

for n=1:length(vec)
    fprintf('%d \n', vec(n)); % 1 3 8 5
end
```

19

KAPLAN

MATLAB – FOR LOOP – KNOWN NUMBER OF REPETITIONS (3)

- A for loop can also be used a matrix:

  -Example with a matrix

```
my_mat = [3, 6, 9; 2, 4, 6];
[rows, cols] = size(my_mat);
for row=1:rows
    for col=1:cols
        fprintf('(%d) ', my_mat(row, col));
    end
    fprintf('\n');
end
% (3) (6) (9)
% (2) (4) (6)
```

20

KAPLAN

MATLAB – WHILE LOOP – UNKNOWN NUMBER OF REPETITIONS

- A while loop is typically used in situations where the number of repetitions is typically unknown:

  -Example: Request the user to key in some numbers or 0 to quit

```
count = 0
num = input('Enter any number or 0 to exit: ');
while num ~=0
    count = count + 1;
    num = input('Enter any number or 0 to exit: ');
end
fprintf('You entered %d numbers\n', count);
```

21

KAPLAN

MATLAB – WHILE LOOP – UNKNOWN NUMBER OF REPETITIONS (2)

- A sample output:

```
Enter any number or 0 to exit: 1
Enter any number or 0 to exit: 2
Enter any number or 0 to exit: 3
Enter any number or 0 to exit: 4
Enter any number or 0 to exit: 5
Enter any number or 0 to exit: 6
Enter any number or 0 to exit: 8
Enter any number or 0 to exit: 0
You entered 7 numbers
```

22

KAPLAN

MATLAB

COMBINING SELECTION WITH REPETITION

KAPLAN

MATLAB – COMBINING SELECTION WITH REPETITION

- Write a program that counts odd and even positive numbers from a vector.

```
numbers = [52, -35, 12, -31, 1, 33, 20, 6];
```

Next declare some 2 variables to count the positive odd numbers and positive even numbers

```
positive_odd_count = 0;
positive_even_count = 0;
```


KAPLAN

MATLAB – COMBINING SELECTION WITH REPETITION

- The following program counts odd and even positive numbers from a vector.
  -Example: Request the user to key in some numbers or 0 to quit

```
numbers = [52, -35, 12, -31, 1, 33, 20, 6];
positive_odd_count = 0;
positive_even_count = 0;
for number=numbers
    if number >= 0
        if rem(number, 2) == 0
            positive_odd_count = positive_odd_count + 1;
        else
            positive_even_count = positive_even_count + 1;
        end
    end
end
fprintf("Number of positive odd numbers: %d\n", positive_odd_count);
fprintf("Number of positive even numbers: %d\n", positive_even_count);
```

25

KAPLAN

SUMMARY

- MATLAB, like majority of the programming languages have control structures
- The default mode for MATLAB is sequential execution.
- To cater to alternate flows, MATLAB also provides selection and repetition structures like most programming languages

26

KAPLAN

References

- MathWorks (n.d) MATLAB https://www.mathworks.com/products/matlab.html

KAPLAN