Unit 1: MATLAB Basics
As a Programming Language

Variables / Arrays

- First of all: **Every variable in MATLAB is an array!**
  - An array has two dimensions by default (so it's like a matrix), but it is easy to create arrays of more dimensions.
- The first two dimensions represent **rows** and **columns**.
- A **scalar** is just an array with a single element.
- A **vector** is an array with at most one **non-singleton** dimension.
- An array element can be
  - A single value ("normal" arrays)
  - Another array (more complicated; to be discussed later)

Variables / Arrays

- MATLAB variables are created and deleted dynamically.
- Variable names are case-sensitive.
- To create a variable: Just assign something to it.
- List initialization / specification of an array:
  - Enclose the element values in `[...]`, using semicolon (`;`) to separate rows.
- Specify an empty array using `[]`.
- To free the memory used by an array:
  - Setting it to an empty array; the variable name remains valid.
  - Statement **`clear`**; the variable name is deleted.

Data Types

- Basic numerical data types: **`double`** (the default), **`single`**, **`int8`**, **`uint8`**, **`int16`**, **`uint16`**, **`int32`**, **`uint32`**
- Complex numbers (**`complex`**)
- Logical data type (**`logical`**)
- Character data type (**`char`**; to be discussed later)
- Note: All the data types are actually **classes** (like in OOP).
- User-defined data types (to be discussed later)
- Functions related to data types: **`class`**, **`isa`**, **`logical`**, **`islogical`**
- Type casting (conversion between data types)

Allocation and Initialization

- Some functions for allocating arrays: **`zeros`**, **`ones`**, **`eye`**, **`true`**, **`false`**, **`rand`**
  - Specification of sizes / dimensions / data type (when applicable)
- Functions for getting array dimensions and sizes: **`size`**, **`length`**, **`numel`**, **`isempty`**
- Initializing vectors with evenly-spaced values:
  - **`a:b`** and **`a:c:b`** expressions
  - Function **`linspace`**

Array Element Indexing

- **Important: Array indices are 1-based in MATLAB.**
- Access and assignment of a sub-array
  - Format: `A(v1,v2,v3,...)`
  - Each dimension is specified by a vector (which can be a scalar) that represents the **subscripts**. for that dimension.
  - The use of a single colon (`:`) to index a dimension:
  - The use of keyword `end` in array indices:

Array Element Indexing

```
A = { 15 23 8; 7 11 14 }
```

A MATLAB array in the memory:

Header	15	7	23	11	8	14
Subscript #1 (row):	1	2	1	2	1	2
Subscript #2 (column):	1	1	2	2	3	3
Linear index:	1	2	3	4	5	6

- **Linear index** corresponds to the arrangement of array elements in the memory.
- When the first two dimensions are concerned, MATLAB uses "**column-major**", while C uses "**row-major**".

Array Element Indexing

- Use **`A(:)`** to convert an array to a **column vector**.
- Conversion between subscripts and linear indices: Functions **`sub2ind`**, **`ind2sub`**
- Related functions that change "array shapes" while preserving the array data:
  - Function **`squeeze`**
  - Function **`reshape`**
- **`A(B)`** type array access (both **`A`** and **`B`** are arrays), with **`B`** containing positive integers that are **linear indices** into **`A`**.

Adding and Deleting Rows/Columns

- Adding rows/columns:
- Deleting rows/columns:
- Concatenation
  - Along columns and rows
  - Beyond the first two dimensions: Function **`cat`**
- Repeating an array: Function **`repmat`**

Text Output

- Just type a variable name or an expression:
  - Or the variable name followed by assignment to it.
  - Just give an expression without a variable name: Assignment is made to variable **`ans`**
  - End the statement with a semicolon (`;`) to suppress such text outputs.
- Function **`disp`** shows the value of an expression without the variable name, and is more compact.
- C-style formatted output: function **`fprintf`**
  - **`fprintf('Size of A is %dx%d\n', size(A));`**
  - C-style escape sequences ok, (**`'\n'`**, **`'\t'`**, etc.)
  - The format string is repeated if there are more values to print out.

Basic Operators

scalar operator scalar

- Arithmetic operations: `+`, `-`, `*`, `/`, `^`; function: `mod`
- Relational operators: `>`, `>=`, `<`, `<=`, `==`, `~=`
- Logical operators: `~`(unary), `&`, `&&`, `|`, `||`
  - `&&` and `||` are preferred over `&` and `|` for efficiency when the conditions are scalars (skipping unnecessary evaluations as in C).

Basic Operators

array operator scalar

- Result is an array.
- Operation is between each array element and the scalar.
- Arithmetic operations: `+`, `-`, `*`, `/`, `.^`; function: `mod`
  - `^`: This is a matrix operation (only if the array is a square matrix).
  - `.^`: This is for element-wise power computation.
- Relational operators: `>`, `>=`, `<`, `<=`, `==`, `~=`
- Logical operators: `~`(unary), `&`, `|`

Basic Operators

scalar operator array

- Result is an array.
- Operation is between each array element and the scalar.
- Arithmetic operations: `+`, `-`, `*`, `^`, `./`; function: `mod`
  - `/`: Cannot be used in this case. (Use `./` instead.)
- Relational operators: `>`, `>=`, `<`, `<=`, `==`, `~=`
- Logical operators: `~`(unary), `&`, `|`

Basic Operators

array operator array

- Result is an array.
- Element-wise operations: The two arrays must have the same size.
  - Numerical operations require that the two arrays to be of the same type (unless one is a scalar double).
  - Arithmetic operations: `+`, `-`, `.*`, `./`, `.^`; function: `mod`
  - Relational operators: `>`, `>=`, `<`, `<=`, `==`, `~=`
  - Logical operators: `~`(unary), `&`, `|`

Some Basic Math Functions

These functions are element-wise when applied to arrays:

- Rounding functions: **`round`**, **`ceil`**, **`floor`**
- The use of **`abs`** (different meanings for real and complex numbers)
- Functions: **`exp`** and **`log`**, (also **`log2`** and **`log10`**)
- Functions: **`sin`**, **`cos`**, **`tan`**, **`asin`**, **`acos`**, **`atan`**, **`atan2`**
  - Degree-based versions by attaching **`d`** to triangular function names (such as **`sind`**)
- A variable with default value: **`pi`**

Basic Vector/Array Operations

- Vector functions: **`sum`** and **`mean`**
- Vector functions: **`cumsum`** and **`cumprod`**
- Using vector functions on multi-dimensional arrays:
  - Specifying the dimension
  - Example: Whole-array sum and mean
- Functions: **`min`** and **`max`**
  - Array-and-scalar or array-and-array (element-wise)
  - Single-array vector-wise operation: **`min(X,[],dim)`**
- Sorting: **`sort`**
  - Getting the index

Logical Indexing

- Logical indexing: Using logical values (true or false) to indicate which array elements to select.
  - Selecting array elements by logical indexing
  - Assignment with logical indexing
  - An example: Thresholding.
- Function: **`find`**
  - linear index output
  - subscripts output

Basic Matrix Operations

- You've got to know some linear algebra …
- Some useful operations:
  - For vectors: functions **`dot`** and **`cross`**
  - For square matrices: functions **`diag`** and **`trace`**
  - transpose: operators **`'`** and **`.'`**; they are the same for real numbers.
  - Functions: **`fliplr`** and **`flipud`**
  - Matrix operations: **`*`**, **`/`**, **`^`**
  - Solving a set of linear equations: Operator **`\`**
  - Also used a lot: functions **`norm`**, **`inv`** and **`eig`**
  - Many more …

Using Matrix Operator '\'

- Solving a set of multi-variable linear equations:
  - The set of equations is expressed as $\boldsymbol{Az=y}$ ($\boldsymbol{A}$ is a square matrix; $\boldsymbol{z}$ and $\boldsymbol{y}$ are column vectors)
  - $\boldsymbol{z}$ (the unknowns) is solved by $\boldsymbol{A\backslash y}$
- Min-squared-error approximation (example: line fitting):
  - To find a line $y=ax+b$ that approximates the points (x_1,y_1), (x_2,y_2), … (x_n,y_n), where $n>2$.
  - Set of equations in the form of $\boldsymbol{Az=y}$:
    - $\boldsymbol{A} = [x_1\ 1;\ x_2\ 1;\ \ldots\ ;\ x_n\ 1]$
    - $\boldsymbol{y} = [y_1;\ y_2;\ \ldots\ y_n]$
    - $\boldsymbol{z} = [a;\ b]$
  - $\boldsymbol{z}$ (the unknowns) is solved by $\boldsymbol{A\backslash y}$

Vectors as Sets

- Set operations: Functions: **`intersect`**, **`union`**, **`setdiff`**
  - Using returned indices
- A related function: **`unique`**

Scripts and m files

- Scripts: collections of statements saved in a file (**m** file)
- Scripts use the global scope (i.e., the workspace) for their variables, so they share variables with interactive statements.
- Using the editor
- Comments (symbol %)

Program Flow Control Overview

- Usage similar to C.
- Conditional branching:
  - `if … elseif … else … end`
  - `switch … case … otherwise … end`
- Loops:
  - `while … end`
  - `for … end`

Program Flow Control

- Using **`if … elseif … else … end`**:

```
if expression
   statements
elseif expression
   statements
…
else
   statements
end
```

- Can be nested.
- Use scalar for the conditions. (Vector conditions are handled with AND.)
- Numerical conditions are treated as true if non-zero and false if zero.

Program Flow Control

- Using **`switch … case … otherwise … end`**:

```
switch switch_expression
case case_expression
   statements
case case_expression
   statements
…
otherwise
   statements
end
```

Program Flow Control

- Using **`switch … case … otherwise … end`**:
  - Only the statements under one **`case`** (or **`otherwise`**) are executed (different from C)
  - **`otherwise`** is optional
  - A case expression can contain multiple choices to be matched to the switch expression.
    - Example: **`case {1, 2}`**
    - That is actually a cell array (to be discussed later).
  - **`switch`** and **`case`** expressions can be strings (to be discussed later)

Program Flow Control

- Using **`while`** … **`end`**:

```
while expression
   statements
end
```

- Same as in C

Program Flow Control

- Using **`for … end`**:

```
for control_variable = values

   statements

end
```

- Here **`control_variable`** is a variable name. (Instead of using **`i`** and **`j`**, I prefer using **`ii`** and **`jj`**.)
- In the k^{th} iteration, index is the k^{th} column (a row vector) of ***`values`***.
  - For normal use, set ***`values`*** to a row vector to avoid confusion.

Program Flow Control

- Using **`break`**: Terminate the current (inner-most) **`while`** or **`for`** loops
- Using **`continue`**: Terminate the current iteration of the current **`while`** or **`for`** loops, and start next iteration.
- Same as in C