ENGR1510J Recitation Class

Week 5

Su Qijian

UM-SJTU Joint Institute

Advanced Matlab, Plotting & Structures

October 14, 2024

Presentation Overview

1. Reminders
   - Reminders
   - Recording
2. Playbook Review
   - c4
3. Worksheets Review
   - w4
4. Reference
   - Reference
   - Q&A

Reminders

- Please remember to release the project! Tomorrow is the deadline!

Recording

- Not Available Yet!

Q1: How to draw basic shapes such as circle, square, rectangle?

Review: c4

Circle:

```
theta = linspace(0, 2*pi, 100); % Angle values from
0 to 2*pi
x = cos(theta); % X coordinates
y = sin(theta); % Y coordinates
plot(x, y);
axis equal; % Equal scaling for both axes
```

Square:

```
rectangle('Position', [0, 0, 1, 1], 'EdgeColor',
'b');
```

Rectangle:

```
rectangle('Position', [0, 0, 2, 1], 'EdgeColor',
'r');
```

Review: c4

Q2: How to draw a large dot or a cross as a marker?

Large Dot Marker:

```
plot(x, y, 'o', 'MarkerSize', 20, 'MarkerEdgeColor',
'b', 'MarkerFaceColor', 'b');
title('Large Dot');
axis equal;
```

Large Cross Marker:

```
plot(x, y, 'x', 'MarkerSize', 20, 'MarkerEdgeColor',
'r');
title('Large Cross');
axis equal;
```

Q3: Explain why data type is important

In programming, a **data type** is a classification of data that tells the compiler or interpreter how the programmer intends to use the data.

Memory Usage: Different data types allocate different amounts of memory. For instance, an `int` may use 4 bytes, while a `float` uses 8 bytes. Efficient selection of data types can reduce memory usage.

Operations: The operations that can be performed on data depending on its type. For example, you can perform arithmetic operations on integers and floating-point numbers, but not on strings or booleans.

Review: c4

Q4: Clearly explain what a data type is

Review: c4

In programming, a **data type** refers to a specific classification that defines what type of value a variable can hold and what operations can be performed on that value.

Review: c4

Q5: Fully understand how 2-complement work

Review: c4

The 2's complement system uses a fixed number of bits (e.g., 8-bit, 16-bit, 32-bit) to represent both positive and negative integers. The most significant bit (MSB) is used as the sign bit, where 0 represents positive numbers, and 1 represents negative numbers.

1. **Invert the digits of the binary number (1's complement).**
   - Change all 0s to 1s and all 1s to 0s.
   - This process is called finding the 1's complement.
2. **Add 1 to the result.**
   - Adding 1 to the inverted number gives the 2's complement.

This resulting binary number is the 2's complement, which represents the negative of the original number.

Review: c4

Let's take the number -5 and represent it in 8-bit 2's complement.

1. Start with the positive binary representation of 5: `00000101` (in 8-bit).
2. Find the 1's complement by inverting the digits: `11111010`.
3. Add 1 to the result: `11111010 + 1 = 11111011`.

So, the 2's complement representation of -5 in 8-bit binary is `11111011`.

Q6: Can decimal numbers be encoded the same way as integers? Explain.

No, decimal numbers (floating-point numbers) cannot be encoded in the same way as integers.
Decimal numbers have fractional components, which integers cannot represent. To handle decimal values, we use floating-point representation, which involves encoding a number in the form of a *mantissa*, *exponent*, and *sign* (using scientific notation in binary).

Review: c4

The most common standard for representing floating-point numbers is the IEEE 754 standard. This standard defines a floating-point number as:

$$N = (-1)^s \times M \times 2^E$$

where:

- s is the sign bit.
- M is the mantissa (fractional part).
- E is the exponent.

For example, the decimal number 5.75 can be represented in 32-bit IEEE 754 format as:

`01000000101110000000000000000000`

This format is very different from how integers are encoded, as it accounts for both the fractional part (mantissa) and the scale (exponent).

Q7: Explain the up and down sides of using arrays of doubles by default in MATLAB.

Review: c4

Upsides:

- **Precision:** MATLAB uses double-precision floating-point numbers (64-bit) by default, which provides high precision for numerical calculations.
- **Convenience:** The default usage of double precision allows users to perform a wide range of calculations without worrying about specifying data types manually.

Downsides:

- **Memory Usage:** Using arrays of doubles by default can consume more memory than necessary, especially when dealing with large datasets or when high precision is not required.
- **Performance Overhead:** Double precision can introduce performance overhead in scenarios where calculations could be performed using lower precision (e.g., single precision or integers).

Q8: What is type casting?

Type casting refers to the process of converting a variable from one data type to another. In programming, type casting allows developers to change the data type of a variable explicitly or implicitly, depending on the situation.

1. Implicit Type Casting

For example:

- In many programming languages, when you combine an `int` with a `float`, the `int` is automatically promoted to a `float`.

```
a = 5;
b = 3.14;
c = a + b;
% MATLAB automatically converts 'a' to float
```

2. Explicit Type Casting It requires the programmer to specify the target data type.
For example:

- Converting a floating-point number to an integer will result in the loss of the fractional part, and the programmer needs to explicitly tell the program to perform this conversion.

```
a = 3.75;
b = int32(a);
% Manually cast 'a' to an integer (result is 3)
```

Review: c4

Q9: Why shouldn't you use functions such as `isreal` to exit a loop?

Review: c4

Functions like `isreal` are used to check whether a given variable or result is a real number, and they return a logical value (true or false).

`isreal` checks whether the result is real, but due to floating-point precision errors, the loop may continue running indefinitely.

Example:

- Consider an iterative algorithm performing repeated calculations. Even if the expected result is real, slight inaccuracies due to floating-point precision (e.g., $1.0 + 1e^{-15}i$), making `isreal` return `false`.

Review: c4

Q10: Detail a good way to use `isletter`.

The function `isletter` in MATLAB is used to determine whether the characters in a string are alphabetic letters. It returns a logical array where each element corresponds to whether a character in the string is a letter (true) or not (false).
A good way to use `isletter` is when you need to process or filter out alphabetic characters from a string, such as in text parsing, data cleaning, or string validation tasks **Refer to `rc1.pdf` w2 ex8**.

Review: c4

Q11: In `strcmpi` what is the meaning of the i?

Review: c4

The `i` in `strcmpi` stands for **case-insensitive** comparison. This means that when using `strcmpi` in MATLAB, the function compares two strings while ignoring the case (upper or lower) of the letters. In contrast, `strcmp` performs a case-sensitive comparison, where the case of each letter must match exactly.

Example: `strcmp('Hello', 'hello')` % returns 0 (false)
`strcmpi('Hello', 'hello')` % returns 1 (true)

In the first example, `strcmp` returns `false` because 'H' and 'h' are different due to case sensitivity. However, `strcmpi` returns `true` because it ignores case differences.

Q12: What is difference between strfind and findstr'. Give two examples showing how to best use each of them.

Review: c4

`strfind` and `findstr` are both used in MATLAB to locate substrings within a string, but they differ in how they handle their inputs and outputs:

- `strfind` searches for the occurrence of a substring within a string and returns the starting indices of the matches.
- `findstr` is a legacy function that works similarly but is more limited. It finds the starting index of one string in another and returns a vector of indices.

Review: c4

Example 1: Using `strfind`

```
str = 'ENGR1510J is fun';
idx = strfind(str, 'fun');
disp(idx); % Output: 14
```

Example 2: Using `findstr`

```
str1 = 'MATLAB';
str2 = 'LA';
idx = findstr(str2, str1);
disp(idx); % Output: 3
```

Q13: What is the difference between an ascii and a binary file?

Review: c4

ASCII File:

- An `ASCII` (American Standard Code for Information Interchange) file is a plain text file where data is represented in a human-readable form.
- ASCII files can be opened and edited with standard text editors (e.g., Notepad, Vim) because the content consists of readable characters.
- Example: A text file containing the string ”Hello World” would store the ASCII values for each letter.

Binary File:

- A `binary` file, on the other hand, stores data in a format that is not human-readable. The data is stored as raw binary data (1s and 0s).
- Binary files are typically opened and processed by specific programs that understand the file format (e.g., image viewers, compilers).
- Example: An image file such as a `.jpg` or an executable file `.exe` is stored in binary format.

Review: c4

Q14: What are the benefits of binary files over ascii ones? What about the other way around?

Review: c4

Benefits of Binary Files over ASCII Files

- **Efficiency:** Binary files store data more efficiently, taking up less space on disk for the same amount of information.
- **Accurate Representation of Complex Data:** Binary files can store complex data structures such as floating-point numbers or machine code exactly as they are represented in memory.

Benefits of ASCII Files over Binary Files

- **Human Readability:** ASCII files are easy to read and edit with standard text editors.
- **Cross-platform Compatibility:** ASCII files are generally more portable across different platforms and systems because they only contain human-readable characters, and almost all systems can handle ASCII text.

Review: c4

Q15: What is a data structure?

Review: c4

A **data structure** is a way of organizing, managing, and storing data in a computer so that it can be accessed and modified efficiently.

- **Organization:** Data structures allow the systematic arrangement of data for better access and management.
- **Efficiency:** Using appropriate data structures can significantly improve the efficiency of algorithms, especially in terms of time complexity and space complexity.
- **Flexibility:** Data structures allow for flexible storage and manipulation of different types of data, from simple integers to more complex objects.

Example in MATLAB: `complexNum = struct('real', 3, 'imaginary', 4);`

Q16: What are the benefits of using data structures?

Review: c4

1. Efficient Data Management: They **systematically organize data**, making it easier to store, access, and modify large datasets. Proper data structures can **reduce the complexity** of operations like searching and sorting.

2. Optimized Algorithms: The right data structure **improves algorithm performance**, optimizing time and space complexity. For example, **hash tables** can reduce search time from $O(n)$ to $O(1)$.

3. Faster Access to Data: Arrays, trees, and hash tables allow quick access to data, improving the efficiency of operations like lookups and inserts.

Q17: How to use the struct keyword?

Review: c4

In MATLAB, the `struct` keyword is used to create structures. A structure can contain different fields, and each field can hold various types of data such as numbers, strings, arrays, or even other structures.

A structure can be created using the `struct` keyword followed by field names and their corresponding values.

Example in MATLAB:

```
person = struct('Name', 'HorseCow', 'Age', 30,
'Occupation', 'Professor?');
```

This creates a structure called `person` with three fields: `Name`, `Age`, and `Occupation`.

Q18: How to take advantage of vectorizing loops to access several elements at once?

Vectorization in MATLAB allows you to perform operations on entire arrays or vectors without explicitly using loops.

Using a Loop:

```
arr = [1, 2, 3, 4, 5];
for i = 1:length(arr)
    arr(i) = arr(i).*2;
end
```

Q19: How to know that max returns two values?

Review: c4

It can be found by referring to MATLAB's documentation or by inspecting the output of the `max` function when called with two output arguments.

When calling `max` with two outputs, it returns:

- The first output is the maximum value in the array.
- The second output is the index (position) of the maximum value.

```
arr = [10, 20, 5, 40, 15];
[maxValue, maxIndex] = max(arr);
disp(maxValue); % Output:  40
disp(maxIndex); % Output:  4
```

Q20: Why is the ceil function used in the code?

Review: c4

The `ceil` function in MATLAB is used to round numbers up to the nearest integer. This means that any fractional part of a number is discarded, and the number is rounded up, regardless of how small the fractional part is.

```
items = 23;
batchSize = 5;
numBatches = ceil(items / batchSize); % Round up to
the next integer
```

Q21: Explain the reasoning applied to discover the name of the student with the highest score.

Review: c4

To discover the name of the student with the highest score.
First, you need to have the data organized in two arrays or structures:

- One array (or structure field) contains the names of the students.
- Another array (or structure field) contains their corresponding scores.

Review: c4

```
scores = [85, 92, 78, 90, 95];
[maxScore, index] = max(scores);
```

In this example, `maxScore` will contain the value `95`, and `index` will store `5`, the position of the highest score in the `scores` array.

Once you have the index of the highest score, you can use it to retrieve the corresponding student's name from the array of student names.

```
names = {'Alice', 'Bob', 'Charlie', 'David', 'Eva'};
highestScorer = names(index);
```

Here, `highestScorer` will contain `'Eva'`, since she has the highest score.

Note: For some of the questions, we won't directly provide the entire source code. Instead, we will offer ideas, a piece of code, or pseudo-code to help guide you on the worksheet questions.

Use the function plot to draw basic geometrical shapes in MATLAB.

Circle:

```
theta = linspace(0, 2*pi, 100); % Angle to 2*pi
r = 1; % Radius of the circle
x = r * cos(theta); % X-coordinates
y = r * sin(theta); % Y-coordinates
plot(x, y, 'b', 'LineWidth', 2); % Plot the circle
axis equal; % Ensure scaling is equal
```

Square:

```
x = [-1, 1, 1, -1, -1]; % square's vertices
y = [-1, -1, 1, 1, -1]; % square's vertices
plot(x, y, 'r', 'LineWidth', 2); % Plot the square
axis equal; % Ensure scaling is equal
```

Review: w4

Rectangle:

```
x = [0, 2, 2, 0, 0]; % rectangle's vertices
y = [0, 0, 1, 1, 0]; % rectangle's vertices
plot(x, y, 'g', 'LineWidth', 2); % Plot the
rectangle
axis equal; % Ensure scaling is equal
```

Triangle:

```
x = [0, 1, 2, 0]; % triangle's vertices
y = [0, 2, 0, 0]; % triangle's vertices
plot(x, y, 'm', 'LineWidth', 2); % Plot the triangle
axis equal; % Ensure scaling is equal
```

Review: w4

Write an algorithm that prompts the user for an integer and returns its 2-complement.

- **Step 1:** Prompt the user for an integer input.
- **Step 2:** Check if the number is positive or negative.
  - If positive, return the binary representation.
  - If negative, proceed with calculating the 2's complement.
- **Step 3:** For negative numbers:
  - Convert the number to its positive equivalent.
  - Get the binary representation.
  - Invert the bits (1's complement).
  - Add 1 to the inverted binary number (2's complement).
- **Step 4:** Display the result.

Example 1: Positive Integer Input

- **Input:** 5
- **Output:** Binary representation: 101

Review: w4 (10 mins)

Implement the previous algorithm in Matlab.

1. `dec2bin(x, bits)`: Converts a decimal number `x` into its binary string representation, with an optional argument to specify the number of bits. **Refer to RC1 for implementation.**

- `bits` (optional): The number of bits for the binary representation.

2. `abs(x)`: Returns the absolute value of the number `x`, which is useful when converting negative numbers to positive.

3. `bitcmp(x, 'uint8')`: Returns the bitwise complement (1's complement) of an integer `x`, treating it as an unsigned integer of the specified bit length.

- `'uint8'` (optional): Specifies the unsigned integer type (e.g., `'uint8'`,`'uint16'`,`'uint32'`). This determines the number of bits used for the complement.

Review: w4 (10 mins)

- First, take the absolute value: `abs(-5)` gives `5`.
- Convert to binary: `dec2bin(5, 8)` gives `'00000101'`.
- Then, invert the bits: `'bitcmp(00000101, 'uint8')'` becomes `'11111010'` (1's complement).
- Finally, add 1: `11111010 + 1` gives `11111011`, which is the 2's complement.

Review: w4

Define a structure that contains the chapter number, the title of the chapter, and the number of slides in this chapter.

```
chapterInfo = struct( ...
'ChapterNumber', 1, ...
'Title', 'Introduction to MATLAB', ...
'NumSlides', 25 ...
);
```

The structure contains:

- **ChapterNumber:** The chapter number.
- **Title:** The title of the chapter, stored as a string.
- **NumSlides:** The number of slides in the chapter.

Return the title of the longest and shortest chapters in the course as well as how many slides compose each of them.

- Input: A structure array containing chapter information.
- Initialize: Variables for the longest and shortest chapters.
- Loop through the structure array:
  - Compare the number of slides in each chapter with the current longest and shortest.
- Output: Titles and number of slides for both the longest and shortest chapters.

Review: w4 (10 mins)

```
chapters(1) = struct('ChapterNumber', 1, 'Title',
'Introduction', 'NumSlides', 20);
chapters(2) = struct('ChapterNumber', 2, 'Title',
'Loops and Conditions', 'NumSlides', 35);
chapters(3) = struct('ChapterNumber', 3, 'Title',
'Functions', 'NumSlides', 10);
chapters(4) = struct('ChapterNumber', 4, 'Title',
'Data Structures', 'NumSlides', 50);

% Initialize variables for the longest and shortest
chapters
longestChapter = chapters(1);
shortestChapter = chapters(1);
```

Review: w4 (10 mins)

```
for i = 2:length(chapters)
  if chapters(i).NumSlides >
longestChapter.NumSlides
    longestChapter = chapters(i);
  end
  if chapters(i).NumSlides <
shortestChapter.NumSlides
    shortestChapter = chapters(i);
  end
end
```

```
disp(['Longest Chapter:  ', longestChapter.Title, ',
Slides:  ', ...
num2str(longestChapter.NumSlides)]);
disp(['Shortest Chapter:  ', shortestChapter.Title,
', Slides:  ', ...
num2str(shortestChapter.NumSlides)]);
```

References

- Manuel. *c4.pdf*. JI Canvas, 2024. [Link].
- Manuel. *w4.pdf*. JI Canvas, 2024. [Link].

The End

Questions? Comments?