ENGR151 Recitation Class 2

week 4

Wu JiaXi

UM-SJTU joint institute

nina_nhk@sjtu.edu.cn

Computer & Programming, MatLab-scripting

October 11, 2024

Presentation Overview

1. Playbook Review
   - C2
   - C3
2. Worksheet Review
   - W3
3. Referencing

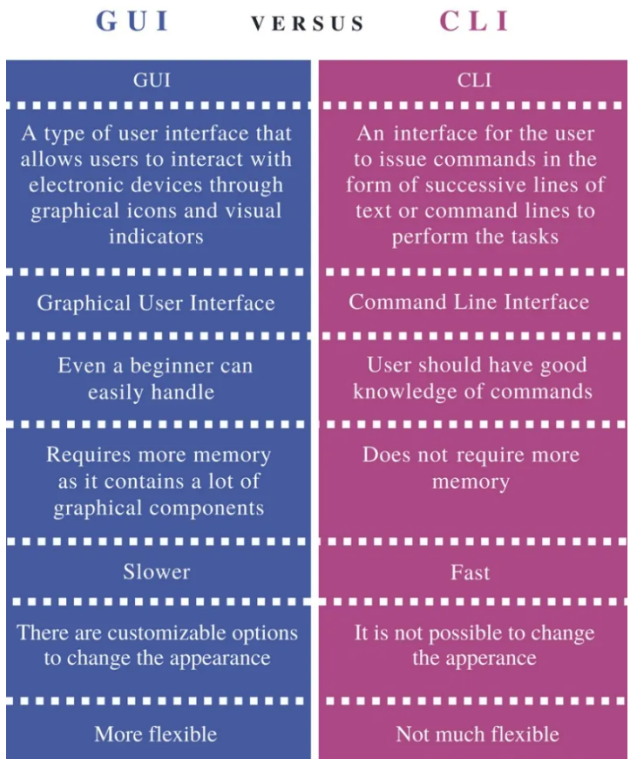
Disclaimer:
The answers provided here are not guranteed to be correct. Please use them as a references only and verify with reliable source.

Note:
only go through some questions that we think they are necessary.

2.1.1 What is the terminal mode of MATLAB?

In terminal mode, MATLAB operates entirely through the **command line interface (CLI)**

This mode allows you to execute commands and scripts directly from the terminal or command line.



2.1.4What is the difference between clc and clearvars?

- `clc`: Clears the **command window**. It erases all text and output currently visible in the command window, but it **does not affect the workspace variables or any of the data in memory**.
- `clearvars`: Clears **variables from the workspace**.It removes the specified variables from memory, freeing up the workspace.

Usecases

- `clc`: to clean up the command window when it becomes cluttered with output.
- `clearvars`: to free up memory or reset the workspace by removing variables you no longer need.

Note: use `help` command for quick search of function.

2.1.6 What is the difference between an array and a matrix?

Aspect	Array in MATLAB	Matrix in MATLAB
Definition	General n-dimensional data structure (can be 1D, 2D, or higher).	A two-dimensional array with rows and columns.
Dimensions	Can have more than two dimensions (e.g., 1D, 3D, etc.).	Always two-dimensional (rows and columns).
Creation	Created with functions like `zeros()`, `ones()`, `rand()`, etc. for any dimension.	Created with functions like `zeros()`, `ones()`, `rand()` but constrained to 2D.
Element Access	Accessed using multiple indices, e.g., $A(i, j, k)$ for 3D arrays.	Accessed using two indices, $M(i, j)$ for rows and columns.
Supported Operations	Element-wise operations like `.*`, `./`, `.^`.	Matrix-specific operations like matrix multiplication ($M * N$), inversion, etc.
Use Case	Can represent multi-dimensional data, such as 3D grids or higher.	Used primarily for linear algebra operations (e.g., solving systems of equations, transformations).

Note: More here [Link toward documentation]

2.1.6 What is the conjugate of a real number?

The conjugate of a real number is the same as the real number itself, because real numbers have no imaginary part. For complex numbers, the conjugate is obtained by changing the sign of the imaginary part.

```
Use conjvar to find conjugation of a number
```

2.1.6 What should you test to see the difference between A' and A.'?

1. randomly create a simple matrix
2. test it out in Matlab
3. observe the output

e.g

```
A = [1+2i, 2-3i; 3+4i, 4-5i];
```

C2 Playbook Review

$A = [1 + 2i, 2 - 3i; 3 + 4i, 4 - 5i];$

- A':

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

  transpose and conjugate matrix

- $A.'$:

$$A.' = \begin{bmatrix} 1 + 2i & 3 + 4i \\ 2 - 3i & 4 - 5i \end{bmatrix}$$

  only transpose matrix

2.1.6 Benefit of using help rather than the documentation

- quick and efficient
- text-base and light-weight
- shows you the most relevant, concise details about a specific function

More comparision:

Aspect	help	Documentation
Access	Command line within MATLAB	External web page or help browser
Speed	Quick, text-only output	Slower, with more detailed information
Detail Level	Concise description of a specific function	Comprehensive guide, including examples, explanations, and visuals
Usage	Best for quick reference or syntax check	Best for in-depth learning or troubleshooting
Offline Availability	Available offline	Requires internet connection (for full documentation)
Format	Text output in the MATLAB terminal	Full HTML or PDF documentation with images, links, and more

2.1.8 In MATLAB when scanning a matrix, in what order are elements read?

In MATLAB, elements are read **column-wise by default**.

- About row and column indices: $A(i, j)$, where i is the row index and j is the column index.
- It iterates over columns first, then rows.

For example,

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

It is read in the order: $A(1,1) = 1$, $A(2,1) = 4$, $A(1,2) = 2$, $A(2,2) = 5$, $A(1,3) = 3$, $A(2,3) = 6$.

2.2.2 What is the difference between *A*&*B* and *A*&&*B*?

For *A*&*B*:

- Checks each condition.
- *A* and *B* can work with arrays or matrices but must be of the same size.
- Slower due to element-wise checks.

For *A*&&*B*:

- If the first expression is false, the second expression is not evaluated (short-circuit evaluation).
- *A* and *B* must be scalar conditions (evaluate to single true/false values).
- Faster and more efficient.

2.2.3 Explain when to prefer switch over if:

- Use **switch** when dealing with multiple specific cases related to a single variable.
- Use **if** when conditions are varied, involve comparisons, or require logical operations.

C2 Playbook Review

discuss the code in C2.
(you should be able to answer the following questions)

```
1  exist('./file') & load('./file')
2  exist('./file') && load('./file')
3  k=input('Press a key: ','s');
4  if k>='0' && k<='9'
5    disp('Digit')
6  else
7    disp('Not a digit')
8  end
```

- **The difference between lines 1 and 2.**
- **Why is k compared to '0' between quotes and not 0'?**
- **Why is the first script returning an error when more than one key is pressed?**

2.3.2 If str2num() is replaced by str2double() is the script still working?

```
i=1; o=input('Input a basic arithmetic operation: ','s');
while (o(i) >= '0' && o(i) <= '9') i = i+1; end
n1=str2num(o(1:i-1)); n=o(i); n2=str2num(o(i+1:end));
switch n
  case '+', n1+n2
  case '-', n1-n2
  case '*', n1*n2
  case '/', n1/n2
  otherwise, disp('Not a basic arithmetic operation')
end
```

C2 Playbook Review

2.3.6 How useful are the continue and break commands?

- `continue`
  - **Purpose**: It skips the remaining code in the current iteration of a loop and proceeds to the next iteration.
  - **Use Case**: helpful when skipping certain iterations based on a condition without terminating the entire loop.
- `break`
  - **Purpose**: It terminates the loop entirely and transfers control to the statement following the loop.
  - **Use Case**: Useful when you want to exit a loop prematurely based on a condition.

2.3.6 Why is it important not to abuse those commands?

- lower code readiblity as the logic become more complicated
- Easily causing logic error in which functions are not working as intended
- use condition statement for better control

2.3.4 What does it mean for the code to be indented?
It refers to the practice of adding spaces or tabs at the beginning of lines to visually represent the structure and hierarchy of the code.Which then helps increases code readability. Having good practice of indented code is important.

2.3.10 What is a logical mask? A logical mask in MATLAB is an array of logical values (true or false) that can be used to index or filter elements in another array. It allows you to selectively access or manipulate parts of an array based on certain conditions.
for example

```
A = [1, 2, 3, 4, 5];
mask = A > 3; % This will result in [false, false,
false, true, true]
```

C2 Playbook Review

```
1 A=magic(5); B=A >10;A(B)=0
2 a=input('Vector: ')
3 b=(mod(a,2)==0);
4 c=a.^2;
5 c(~b)=a(~b).^3
```

2.3.10 What is the mod() function doing?
2.3.10 Given a 5×5 matrix A, use the mod() function to determine the row of A(14).

3.1.1 Clearly explain the difference between script and function

Aspect	Script	Function
Definition	Sequence of commands	Block of code with input/output
Scope of Variables	Global (base workspace)	Local to the function
Input	No inputs	Takes input arguments
Output	No direct outputs	Returns one or more outputs
Usage	Simple tasks, batch commands	Modular code, reusable
Calling	By typing script name	By using function name with parentheses

3.1.3 Clearly explain what is a ”sub-function”?
In MatLab, sub-functions are written after the main function in the same file, and they can only be called by the main function or other sub-functions within the file.
3.1.3Why are sub-functions very useful, and should be use as much as possible?

- As sub-functions **are not visible or accessible from outside the file**, making them useful for organizing code that is meant to be used internally.
- It allows breaking down complex tasks into smaller, manageable pieces.
- It can be reused multiple times within the same file, reducing code duplication.
- Easy debugging.

3.1.3 How to catch more than one output?

In MatLab, we can catch more than one output by using brackets,namely **[]**.

for example,

```
function [sumVal, prodVal] = addAndMultiply(x, y)
    sumVal = x + y;
    prodVal = x * y;
end
```

3.2.1 Explain in simple terms what is recursion.

Recursion is when a function calls itself to solve a problem.

1. The function keeps calling itself.
2. Each call gets closer to a base case, which is a simple scenario where the function doesn't call itself anymore.
3. Once the base case is reached, the function returns a result, and all the previous calls can finish.

- Base case: The stopping point where the function doesn't call itself.
- Recursive step: The part where the function calls itself with a simpler problem.

3.2.2 Explain in simple terms the basic difference between iteration and recursion.

Feature	Iteration	Recursion
Definition	Repeats a block of code using loops.	Calls a function within itself.
Structure	Uses `for`, `while`, or similar loops.	Uses function calls with base and recursive cases.
Termination	Stops when a loop condition fails.	Stops when a base case is reached.
Memory Usage	Generally uses less memory (only one frame on the stack).	Can use more memory due to multiple stack frames.
Performance	Usually faster and more efficient.	Can be slower for deep recursion due to overhead.
Simplicity	Can be simpler for straightforward tasks.	Often simpler for problems that naturally fit recursive definitions.
State Management	Maintains state using loop variables.	Maintains state through function parameters and local variables.
Debugging	Easier to trace through loop iterations.	Can be harder to debug due to multiple calls.

3.2.8 Explain why recursion can cause some memory issues?

- **Stack Overflow**: Too many recursive calls can exhaust the stack memory, leading to a crash.
- **Memory Inefficiency**: Recursion can lead to inefficient use of memory, especially in cases of redundant calculations, because each recursive call maintains its own state on the stack.

3.2.8 Implement the generation of Fibonacci numbers using (i) iteration and (ii) recursion.

- iteration
  1. Input a non-negative integer n.
  2. if n == 0, return 0 / If n == 1, return 1./ iterates f = f0 + f1
  3. sum the result and output the n-th Fibonacci number.
- recursion
  1. Input a non-negative integer n.
  2. if n == 0, return 0 / If n == 1,return 1/ return fibonacci(n-1) + fibonacci(n-2)
  3. output result

C3 Playbook review

iteration

```
function f = fibonacci_iter(n)
   if n == 0
     f = 0;
   else if n == 1
     f = 1;
   else
     f0 = 0;
     f1 = 1;
     for i = 2:n
       f = f0 + f1;
       f0 = f1;
       f1 = f;
     end
   end
end
```

recursion

```
function f = fibonacci_rec(n)
   if n == 0
     f = 0;
   elseif n == 1
     f = 1;
   else
     f = fibonacci_rec(n-1) + fibonacci_rec(n-2);
   end
end
```

3.2.8 What is a ”seed”?

A seed is an initial value used to start a random number generator, which is like the starting point for a sequence. By using a seed, you can control the sequence of ”random” numbers that are produced.

3.2.8 What is happening if the same seed is used to generated random numbers?

If you use the same seed each time, the random number generator will produce the same sequence of numbers every time. This can be useful when you want to reproduce results (for debugging or testing), but it also means the numbers aren’t truly random.

3.3.3 What does it mean to p̈rint in a string?̈

P̈rinting in a stringr̈efers to inserting values into a string using special formatting symbols. Instead of just printing plain text, you can include variables or values inside a string. This is done using format specifiers like `%d` (for integers) or `%f` (for floats).

3.2.8 How to know what flags (e.g., `%d`, `%g`) or special characters (e.g., `\n`, `\t`) should be used? - Read the documentation for `sprintf` or `fprintf`.

Format specifiers are used to format different data types:

- `%d`: For integers (decimal).
- `%f`: For floating-point numbers (decimals).
- `%s`: For string.
- `%c`: For char.
- `%g`: For the most compact representation of a floating-point number (either `%f` or scientific notation, `%e`).

Special characters are used to control how the output looks:

- `\n`: Newline (moves to the next line).

C3 Playbook review

3.3.5 What are the dangers of not closing a file?

- The system keeps the file open in memory, which uses resources unnecessarily and can slow down the program.
- The file might remain locked, preventing other programs or users from accessing or modifying it.
- Data might not be written or saved properly, leading to file corruption.

3.3.5 Why should files be opened as for ”as little time as possible”?

- Opening a file uses system resources (memory and file handles), and the sooner you close it, the more resources are freed for other tasks.
- The longer a file is open, the greater the chance of issues like crashes, file corruption, or conflicts with other programs.
- Closing a file ensures that all data is written and saved properly, reducing the risk of data loss or corruption.

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

Ex 1

Ensure you can answer all the questions in C3 sildes

go through the slide yourself and **make sure you are familiar with all concepts mentioned.**

Ex2

Read Matlab documentation for the following functions:

functions

save, load, ($s|f$) printf, rand, fopen,fclose,randperm

Note:
use "$help + function\_name$" to search for documentation effectively.

Ex2

save

- Purpose: Saves workspace variables to a file.
- Syntax:
  - `save(filename)`: Saves all variables in the workspace to the file filename.
  - `save(filename, variables)`: Saves the specified variables to the file filename.
  - `save(filename, variables, '-append')`: Appends variables to an existing file.

load

- Purpose: Loads variables from a file into the workspace.
- Syntax:
  - `load(filename)`: Loads all variables from the file filename.
  - `load(filename, variables)`: Loads only the specified variables from the file.

Ex2

(s|f)printf

- Purpose: Formats data into a string (sprintf) or writes formatted data to a file or command window (fprintf).
- Syntax:
  - `sprintf(formatSpec, A1, ..., An)`: Formats the data according to formatSpec and returns it as a string.
  - `fprintf(formatSpec, A1, ..., An)`: Writes the formatted data to the screen or a file.

rand

- Purpose: Returns a scalar from a uniform distribution in the interval (0,1).
- Syntax:
  - `rand`: Returns a scalar from a uniform distribution in the interval (0,1).
  - `rand(n)`: Returns an n-by-n matrix of random numbers.
  - `rand(m,n)`: Returns an m-by-n matrix of random numbers.

Ex2

fopen

- Purpose: Opens a file for reading, writing, or appending.
- Syntax:
  - `file_id = fopen(filename, mode)`: Opens the file and returns a file identifier (file_id).
  - Modes:
    - `r`: Read (file must exist).
    - `w`: Write (creates a new file or overwrites an existing one).
    - `a`: Append (writes to the end of the file).

fclose

- Purpose: Closes an open file.
- Syntax:
  - `fclose(file_id)`: Closes the file associated with the file identifier `file_id`.
  - `fclose('all')`: Closes all open files.
- **Remember to close the file every time you open one!**

Ex2

randperm

- Purpose: Returns a row vector containing a random permutation of integers.
- Syntax:
  - `randperm(n)`: Returns a random permutation of the integers from 1 to n.
  - `randperm(n, k)`: Returns k unique integers selected randomly from 1 to n.

Ex3

Solve the problem (3.22—3.103) using an alternative algorithm and different Matlab functions.

Ex3

Objective: Solve the problem using different algorithms and MATLAB functions.

- Explore alternative string splitting functions.
- Investigate other functions to randomly select lines from a file.
- Propose a different algorithm to solve the problem.

Ex3

1. **Alternative functions for splitting strings:**
   - `strsplit(string)`
   - `regexp(string, pattern, 'split')`
2. **Functions for picking random lines:**
   - `randi([1, file_total_length])`
3. **Alternative algorithm:**
   - Read the file, identify the target line, and extract the required string. (Note: This might be redundant but could work in specific cases.)

Ex4

Dicuss the following recursion

A child couldn't sleep, so her mother told her a story about a little frog, who couldn't sleep, so the frog's mother told her a story about a little bear, who couldn't sleep, so the bear's mother told her a story about a little weasel who fell asleep. . . . and the little bear fell asleep; . . . and the little frog fell asleep; . . . and the child fell asleep.

For an automated information service a telephone company needs the digits of phone numbers to be read digit by digit. Design an algorithm allowing to rewrite an sequence of digits into words, with a space between each word, but no space at the beginning and at the end.

what are the base cases?

Try to Implement the second algorithm

```m
function result = number_to_words(number)
    % Special case: When number is 0, we return an empty string
    if number == 0
        result = '';
        return;
    end

    % Extract the last digit
    digit = mod(number, 10);
    rest_of_number = floor(number / 10);

    % Convert the last digit to its word equivalent
    word = digit_to_word(digit);

    % Base case
    if rest_of_number == 0
        result = word;
    else
        % Recursive case
        result = [number_to_words(rest_of_number), ' ', word];
    end
end

```

Ex5

For any integer n, we define its successor $S(n) = n + 1$. For any $x, y \in \mathbb{N}$, let f be the function defined by:

$$f(x, 0) = 0$$

$$f(x, S(y)) = g(f(x, y), x),$$

where g is such that

$$g(x, 0) = x$$

$$g(x, S(y)) = S(g(x, y)).$$

Manually calculate f (0,0), f (1,0), f (1,1), f (2,1), f (2,2), and f (2,3). Using words explain what you guess the function f is doing.

Function $\mathtt{f}$ is recursively multiplying $\mathtt{x}$ and $\mathtt{y}$. With the behavior of the helper function $\mathtt{g}$, it builds up the result by adding $\mathtt{x}$ to the previous result $\mathtt{y}$ times.

What is the base case and recursive case here?

Ex5

Implement it and test it on larger inputs.

sample function

```
function result = f(x, y)
  if y == 0
    result = 0; % Base case
  else
    result = g(f(x, y-1), x); % Recursive case
  end
end
 function result = g(x, y)
  if y == 0
     result = x; % Base case
  else
     result = g(x, y-1) + 1; % Recursive case
  end
end
```

Ex6

Write a MATLAB program composed of a single command to sum up all integers from 0 to n.

Write a simple iterative MATLAB program to sum up all integers from 0 to n.

Ex6

a single command to sum up all integers from 0 to n.

```
sum = n * (n + 1) / 2;
```

a simple iterative MATLAB program to sum up.

- sum function

```
sum = sum(0:n);
```

- while

```
while i <= n
  sum = sum + i;
   i = i + 1;
end
```

- for

```
for i = 0:n
  sum = sum + i;
end
```

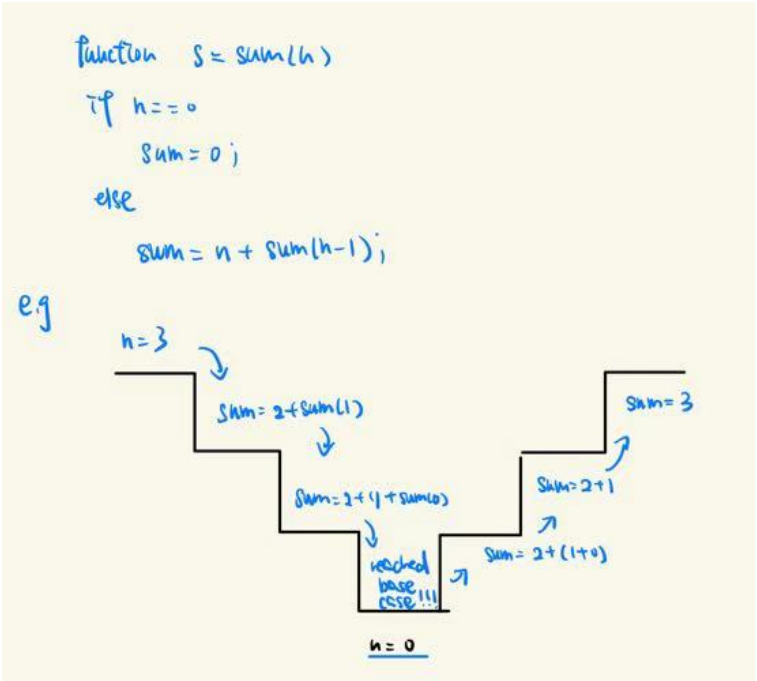
Ex6

Now write a simple recursive MATLAB program to sum up all integers from 0 to n.

- What is the base case?
- What actions occur at each level of the recursion?
- Draw a simple diagram showing the recursion progress.

Ex6

- **What is the base case?**
  For different algorithm, there will be different base cases. In the diagram I provided, the base cases will be **n=0;**
- **What actions occur at each level of the recursion?**
- **Draw a simple diagram showing the recursion progress.**



Ex6

ideas given, try to fill in the blanks!

recsum.m

```
function s = _____(n)
    if n == _____
        s = _____;
    else
        s = _____ + _____;
    end
```

reference code

```
function s= recsum(n) % pay attention to the file
name as this file only contain this function!
  if n == 0
    s = 0;
  else
    s = n + recsum(n - 1);
  end
end
```

References

- Manuel. *c2.pdf*. JI Canvas, 2024. [Link].
- Manuel. *c3.pdf*. JI Canvas, 2024. [Link].
- Manuel. *w3.pdf*. JI Canvas, 2024. [Link].
- Wang, Yuheng. *rc*. FOCS JI Gitea, 2023. [Link].

The End

Questions? Comments?