VG101 — Intoduction to Computers & Programming

MATLAB Midterm Review

Liu Yihao

UM-JI (Summer 2018)

June 8, 2018

Number base conversion

- Humans use decimal (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
- Computers work internally using binary (0,1)
- Human-friendly way to represent binary: hexadecimal

From base b into decimal: evaluate the polynomial:
$(11111101)_2 = 1\cdot 2^7+1\cdot 2^6+1\cdot 2^5+1\cdot 2^4+1\cdot 2^3+1\cdot 2^2+0\cdot 2^1+1\cdot 2^0 = 253$

$(\mathrm{FD})_{16} = \mathrm{F}\cdot 16^1 + \mathrm{D}\cdot 16^0 = 15\cdot 16^1 + 13\cdot 16^0 = 253$

From decimal into base b: repeatedly divide n by b until the quotient is 0. The remainders are the numbers from right to left:
rem(253,2)=1, rem(126,2)=0, rem(63,2)=1, rem(31,2)=1, rem(15,2)=1, rem(7,2)=1, rem(3,2)=1, rem(1,2)=1

rem(253,16)=13=D, rem(15,16)=15=F

Number base conversion

Exercise:

1. Convert into hexadecimal: 1675, 321, $(100011)_2$, $(10111011)_2$
2. Convert into binary: 654, 2049, ACE, 5F3EC6
3. Convert into decimal: $(111110)_2$, $(10101)_2$, $(12345)_16$, 12C3C

Solution:

1. 1675 = 68B, 321 = $(141)_{16}$, $(100011)2 = (23)_{16}$
2. 654 = $(1010001110)_2$, 2049 = $(100000000001)_2$, ACE = $(101011001110)_2$, 5F3EC6 = $(10111110011111011000110)_2$
3. $(111110)_2 = 62$, $(10101)_2 = 21$, $(12345)_{16} = 74565$, 12C3C = 76860

Algorithm and Program

Algorithm $\longrightarrow$ Programming Language $\longrightarrow$ Interpreter / Compiler $\longrightarrow$ Machine Code

Algorithm: recipe telling the computer how to solve a problem.
Programming Language: a formal language that specifies a set of instructions that can be used to produce various kinds of output.
Interpreter: a computer program that directly executes a programming or scripting language.
Compiler: computer software that transforms computer code written in one programming language (the source language) into another programming language (the target language).
Program: a technical setting stored in the memory of a machine or piece of hardware to be executed, including computers.
Machine code: a set of instructions executed directly by a computer's central processing unit (CPU).

Source Code and Pseudocode

Source Code: a sequence of statements and/or declarations written in some human-readable (usually as a text) computer programming language

```
1  function M = density_sun(r, c, G, T)
2      V = 4 / 3 * pi * (c / (2 * pi)) ^ 3
3      M = 4 * pi ^ 2 * r ^ 3 / (G * T ^ 2)
4  end
```

Pseudocode: an informal high-level description of the operating principle of a computer program or other algorithm.

Input: r, c, G, T
Output: Density of the Sun

1: **function** DENSITY SUN(r, c, G, T)
2: $V \leftarrow \frac{4}{3}\pi\left(\frac{c}{2\pi}\right)^3$
3: $M \leftarrow \frac{4\pi^2 r^3}{GT^2}$
4: **return** M
5: **end function**

Variables and Builtin Variables

Variables: start with a letter, case sensitive.
e.g. a=1+2; A=3+2; a123_=4+5;

Some builtin variables:

- pi $= \pi$
- i $= \sqrt{-1}$
- j $= \sqrt{-1}$
- Inf $=$ Infinity
- NaN: Not a Number

Simple operations

- Addition: +
- Subtraction: -
- Multiplication: *
- Power: ^
- (Right) division: a / b $= \frac{a}{b}$
- Left division: a \b $= \frac{b}{a}$

Arrays and Matrices

- Generate a sequence of numbers: `a:b` or `a:b:c`
- Concatenate (join) elements: `[ ]`
- Generate a 1-dimensional array: `[a:b]` or `[a:b:c]`
- Generate a 2-dimensional array: `[a b c; d e f;]`
- Generate a list between a and b, with n elements: `linspace(a, b, n)`
- `zeros(a,b)`
- `ones(a,b)`

Operations of Arrays and Matrices

Calculate element by element:

- .*
- ./
- .\
- .^

Operations of Matrix:

- Complex conjugate transpose: '
- Nonconjugate transpose: *
- Determinant: `det`
- Inverse matrix: `inv`
- Eigen vector and values: `eig`

Accessing elements in a matrix

- Coordinates: using their (row, column) position
- Use ":" to refer to the whole row / column
- "1" is the top / left index
- "end" is the bottom / right index
- Use `size(A)` to get the number of rows and columns in A
- use `length(A)` to get `max(size(A))`
- use an expression of A (i.e., `A==x`) to generate a logical array depending on some condition
- let B be the generated logical array above, then use `A(B)` to generate a new array with elements in A that satisfy this condition

Expressions

Relational operators:

- $<$ less than
- $<=$ less than or equal to
- $>$ greater than
- $>=$ greater than or equal to
- $==$ equal to
- $\sim=$ not equal to

Logical operators:

- & and, A && B evaluates expression B only if A is True
- | or, A || B evaluates expression B only if A is False
- $\sim$ not
- `xor(A,B)` exclusive or of A and B

Conditional Statements

`if` statement:

```
1  if expression1
2      statements1
3  elseif expression2
4      statements2
5  else
6      statements
7  end
```

`switch` statement:

```
1  switch variable
2      case value1
3          statements1
4      case value2
5          statements2
6      otherwise
7          statements
8  end
```

Loop Statements

The `while` loop:

```
while expression
    statements
end
```

The `for` loop:

```
for i=start:increment:end
    statements
end
```

The `continue` and `break` commands:

- `continue`: skip the remaining statements in the loop to go to the next iteration
- `break`: exit the loop and execute the next statements outside the loop

Efficiency in multilevel loops: MATLAB uses the "column-major order", so column should be in the outer loop

Data types

- Different numbers (integer, real, complex...)
- Different range (short, long...)
- Different precision (single, double...)
- Different types $\Rightarrow$ different memory usage, performance

Numeric types:

- int: int8, int16, int32 and int64
- uint: unit8, uint16, uint32 and uint64
- 32bits; `realmax('single')`, `realmin('single')`
- 64 bits; `realmax`, `realmin`
- Two methods for numeric conversions:
  e.g. `cast(a,'uint8')` or `unit8(a)`

Char type: array of Unicode characters, specified by placing data inside a pair of single quotes (e.g. `a='test'; whos a`)

Structures

Structures are array with "named data containers" called fields. Fields can be any kind of data.

1. Initializing the structure

```
student(1)= struct('name','iris num', 'gender',...
'female', 'marks', [30 65 42]);
student(2)= struct('name','jessica wen',...
'gender', 'female', 'marks', [98 87 73]);
student(3)= struct('name','paul wallace',...
'gender', 'male','marks', [65 72 68]);
```

2. Using the structure

```
student(3).gender
mean([student(1:3).marks])
```

3. To go further: who got the best mark?

```
[m,i]=max([student(1:3).marks]);
student(ceil(i/3)).name
```

Script vs Function

Script:

- Sequence of MATLAB statements
- No input/output arguments
- Operates on data on the workspace

Function:

- Sequence of MATLAB statements
- Accepts input/output arguments
- Variable are not created on the workspace

Functions and Sub-functions

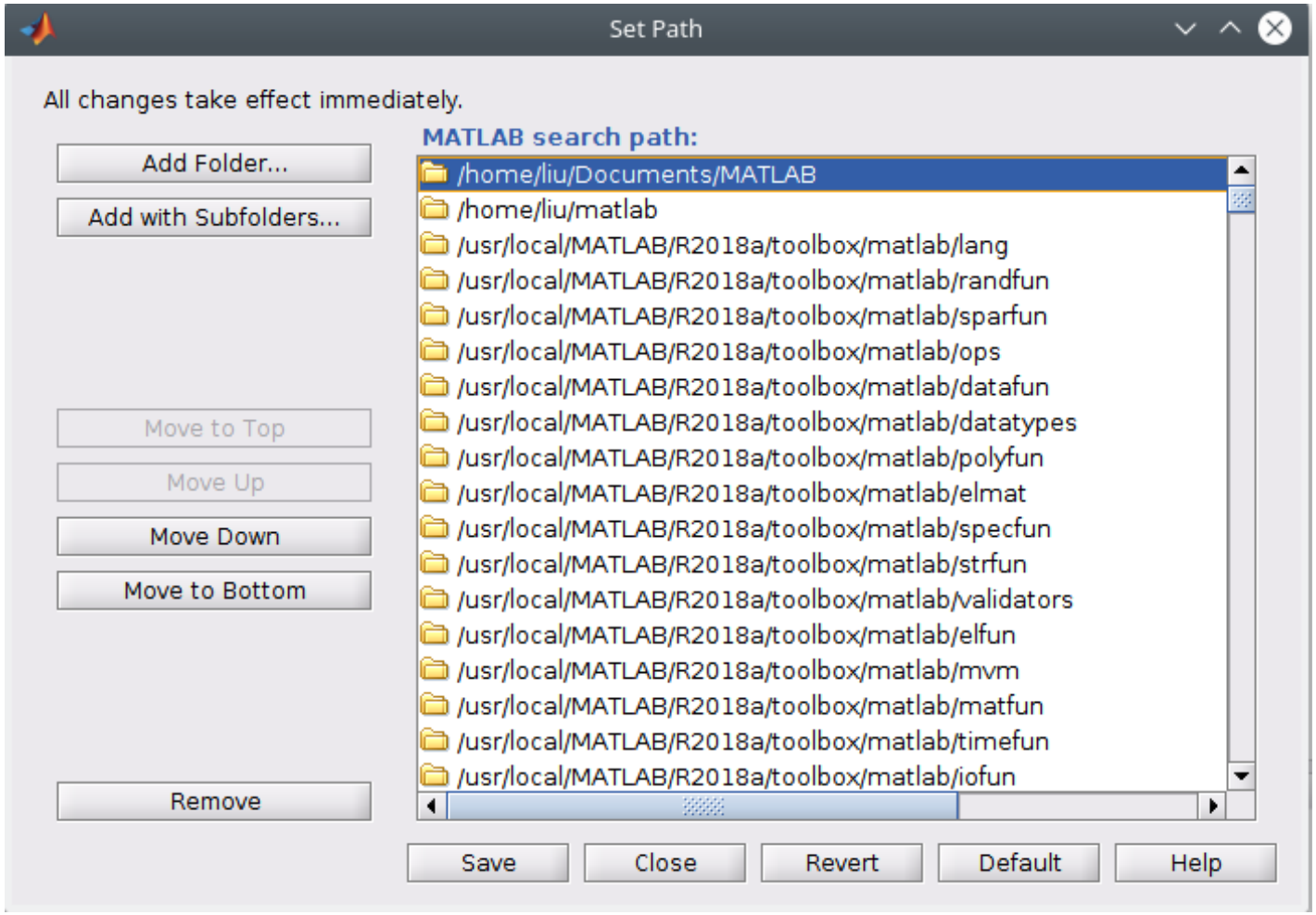
- Function saved in a .m file
- The .m file must be in the `Path`
- The function name must be the same as the filename
- `function [output1, output2,...] = Functionname(input1,input2,...)`
- The function can be called from another .m file or from the workspace

A .m file can contain:

- A "main function"
- Several sub-functions, only visible to other functions in the same file

Path

`Path` consists of the MATLAB search path and the current folder.



Mathematical Functions

- Defining an anonymous function: `f=@(x) x^2-1`
- Integral: `syms z; int(z^2+1), int(z^2+1,0,1)`
- Differentiation: `syms t; diff(sin(t^2))`
- Limit: `limit(sin(t)/t,0)`
- Finding a root of a continuous function: `fzero(f,0.5)`
- Square root: `sqrt(9)`
- Nth root: `nthroot(4, 3)`
- Rounding: `mod(a, b)`, `floor(a)`, `ceil(a)`, `round(x, n)`

File I/O Functions

`save` and `load`:

- `save('filename','var1','var2',..,'format')`
- `load('filename','format')`
- format can be -mat (default, binary) or -ascii (text)

`fid` with `fopen`:

- `fid = fopen('filename', 'permission')`
- `fgetl(fid)`: returns one line
- `fread(fid, count, precision)`: read count elements of type precision
- `fwrite(fid, A, precision)`: write A as elements of type precision
- `ftell(fid)`: offset in bytes of the file position indicator
- `fseek(fid, offset, origin)`: go to position offset starting at origin, where origin can be 'bof', 'cof' or 'eof' (i.e. beginning, current, end)
- `fclose(fid)`

File Permissions

'r'	open file for reading
'w'	open file for writing; discard existing contents
'a'	open or create file for writing; append data to end of file
'r+'	open (do not create) file for reading and writing
'w+'	open or create file for reading and writing; discard existing contents
'a+'	open or create file for reading and writing; append data to end of file

Other Utilities and Functions

- Clean up: `clc`, `clf`, `clear all`
- Time: `tic`, `toc`
- Numeric: `whos`, `isnumeric`, `isreal`, `isnan`, `isinf`, `isfinite`
- String: `isletter`, `isspace`,`strfind`, `findstr`, `strcmp`, `strncmp`, `strrep`, `strsplit`, `strjoin`, `num2str`, `str2num`
- Random: `rand`, `randn`, `randperm`, `random`

Formatted I/O Functions

Write formatted data into a string: `sprintf(format, variables)`
Writing in a file: `fprintf(fid, format, variables)`

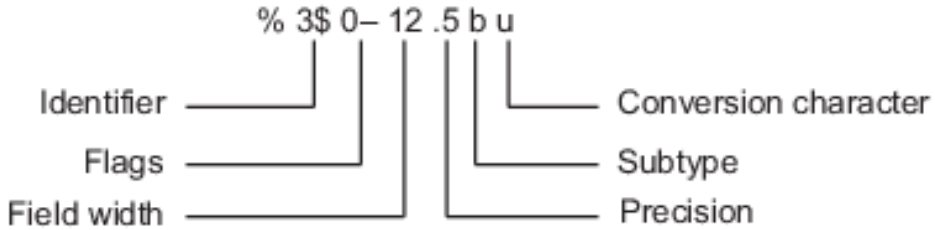
- Format: string with special flag, replaced by value of the variables
- Special characters: carriage return/tab/backspace

Reading from a file: `fscanf(fid, format, size)`

- Reads a file
- Converts into the specified format
- Only reads size elements if size is specified
- Returns a matrix containing the read elements
- Returns an optional parameter: number of elements successfully read

Formatting Operator

A formatting operator starts with a percent sign, %, and ends with a conversion character. The conversion character is required. Optionally, you can specify identifier, flags, field width, precision, and subtype operators between % and the conversion character.



Identifier and subtype operators are not often used, so we will skip them.

Conversion Character

Value Type	Conversion	Details
Integer, signed	%d or %i	Base 10
Integer, unsigned	%u	Base 10
	%o	Base 8 (octal)
	%x	Base 16 (hexadecimal), lowercase letters a-f
	%X	Same as %x, uppercase letters A-F
Floating point number	%f	Fixed-point notation
	%e or %E	Exponential notation, such as 3.141593e+00, 3.141593E+00
	%g or %G	The more compact of %e or %f (%E or %f for %G), with no trailing zeros
Characters or strings	%c	Single character
	%s	Character vector or string array.

Flags

- '': Left-justify. **Example**: %-5.2f, %-10s
- '+': Always print a sign character (+ or) for any numeric value. **Example**: %+5.2f; Right-justify text. **Example**: %+10s
- ' ' Insert a space before the value. **Example**: % 5.2f
- '0' Pad to field width with zeros before the value. Example: %05.2f
- '#' Modify selected numeric conversions:
  - For %o, %x, or %X, print 0, 0x, or 0X prefix.
  - For %f, %e, or %E, print decimal point even when precision is 0.
  - For %g or %G, do not remove trailing zeros or decimal point.

  Example: %#5.0f

Field Width and Precision

Field Width: Minimum number of characters to print. The field width operator can be a number, or an asterisk (*) to refer to an input argument.
Example: The input arguments `('%12d', intmax)` are equivalent to `('%*d', 12, intmax)`.
The function pads to field width with spaces before the value unless otherwise specified by flags.

Precision:

- For %f, %e, or %E, Number of digits to the right of the decimal point
  Example: %.4f prints pi as 3.1416
- For %g or %G, Number of significant digits
  Example: %.4g prints pi as 3.142

Recursion

Recursive acronyms:

- GNU: GNU's Not Unix
- LAME: LAME Ain't an MP3 Encoder
- WINE: WINE Is Not an Emulator
- PHP: PHP Hypertext Preprocessor
- JOJ: JOJ Online Judge

Recursion: repeating items in a self-similar way
Strategy: a function calling itself
The most common algorithm that using recursion is called divide and conquer, which means split a complex problem into similar but smaller problems. The splitting process will repeat until an initial condition, where the answer can be obviously given, is reached. Then the process goes back to solve the original problem layer by layer.

Use Recursion to solve mathematical problems

For example, when we try to find the factorial of n

```
1  function f = fact(n)
2      if n == 0 % initial condition (stop the recursion)
3          f = 1;
4      else
5          f = n * fact(n - 1);
6      end
7  end
```

When you are solving the mathematical problems using recursion, the most important part is finding the **recursive expression** of the answer.

Example: Josephus Problem

A group of n people are standing in a circle, and are numbered from 1 to n in the clockwise order. Now a count-off by m process begins from the No.1 person and goes in the clockwise order. Every time a person is counted as m, he should go out of the circle. The process is repeated until there is only one person left. So what is the number of the last man?

Please implement a function `josephus(n,m)` which returns the number of the last person.

The problem is easy to solve using recursion.

- When $n = 1$, it is obvious that the only man in the circle is the only man left, so in this case the number of the last man is 1
- Suppose that $josephus(n-1, m) = k$, then consider the situation that there are n people. Obviously the first one to get out is No.m (or $mod(m, n)$ if $m > n$)
- So there are only $n-1$ men left now, and it can be seen that the last man should be the kth man counting from No.$m+1$ (or $mod(m+1, n)$), which should be No.$mod((m+k), n)$
- The mod function has a limitation that it will return 0. However what we want is No.n instead of No.0, so a if statement should be written.

```
1  function y = josephus(n,m)
2      if n==1
3          y = 1;
4      else
5          y = mod(josephus(n-1,m)+m,n);
6          if y==0
7              y = n;
8          end
9      end
10 end
```

Using recursion to solve more complex problems

Sometimes a question is not so pure-mathematical and might be a little bit complex. And I think the most crucial part of solving this kind of problems is properly setting the **input arguments** and **return values** of a function.

- As for the input arguments, they should be complete enough to describe a partial procedure in solving a large problem.
- Then what should you do is splitting a whole procedure into several partial procedures.
- Then is to find the initial conditions. Note that in many cases, there might be multiple initial conditions.

Example: Hanoi Tower

A Tower of Hanoi consists of three rods and a number of disks of different sizes, which may slide onto any rod. The system starts with disks on one rod, with disks in ascending order of size, the smallest on the top. You job is to move every all to another rod, under such conditions:

- Only one disk can be moved at a time.
- A disk can only be moved if it is the uppermost disk on a stack.
- No disk may be placed on top of a smaller disk.

Display all the moves.

`https://www.mathsisfun.com/games/towerofhanoi.html`

- The whole process is to move n disks from rod 1 to rod 3. So the definition of the partial procedure should also be move some disks from a certain rod to another rod.
- So I define the subfunction $hanoimove(n, a, b)$, which is the function to print the process of moving n disks from rod a to rod b.
- It is obvious that the initial condition is when $n = 1$, then we just move it from a to b.
- In the case that $n > 1$, it can be seen the disk that should be settled on rod b is the last disk which is the largest. But before that, we should remove all the disks above it, and they should go to the third rod c.
- When all the disks are moved to rod c, then we just move the bottom disk to rod b.
- At last, we should move the disks that has been moved to rod c back to rod b. Then the whole process is over.

So `hanoimove(n,a,b)` includes the three partial procedures: `hanoimove(n-1,a,c)`, `hanoimove(1,a,b)` and `hanoimove(n-1,c,b)`

```
1  function hanoi(n)
2      hanoimove(n,1,3);
3  end
4
5  function hanoimove(n,a,b)
6      if n==1
7          fprintf('%d --> %d\n',a,b);
8      else
9          c = 6-a-b;
10         hanoimove(n-1,a,c);
11         hanoimove(1,a,b);
12         hanoimove(n-1,c,b);
13     end
14 end
```

Exercise: Eight Queens Puzzle

The eight queens puzzle is the problem of placing eight chess queens on an 8x8 chessboard so that no two queens threaten each other. Thus, a solution requires that no two queens share the same row, column, or diagonal.

The eight queens puzzle is an example of the more general n queens problem of placing n non-attacking queens on an nxn chessboard, for which solutions exist for all natural numbers n with the exception of $n = 2$ and $n = 3$.

Write a MATLAB function `queen(n)` to generate all solutions to the n queens puzzle.

2D Plotting

- Plot the columns of x, versus their index: `plot(x)`
- Plot the vector x, versus the vector y : `plot(x,y)`
- Plot function between limits `fplot(f, lim)`
- More than one graph on the figure: `hold`
- Axis properties: `axis`
- Line properties: `linespec`
- Polar graph: `polar(t,r)`
- Bar graph: `bar(x,y)`
- Horizontal bar graph: `barh(x,y)`
- Pie chart: `pie(x)`
- More than one plot: `subplot(mnp)`

3D Plotting

- Plot the vector x, y, t: `plot3(x,y,t)`
- Set up a mesh: `[x, y]=meshgrid(v)`
- Contour: `contour(x,y,z)`
- Color map: `pcolor(x,y,z)`
- 3D view: `surf(x,y,z)`
- 3D bar graph: `bar3(x,y)`
- 3D horizontal bar graph: `bar3h(x,y)`
- 3D pie chart: `pie3(x)`

Curve Fitting

Parametric fitting:

1. Collect data
2. Import data into MATLAB
3. Open curve fitting tool
4. Determine the best fit
5. Extrapolate the data

Applying a fit:

1. Provide a name for the fit
2. Select cdate for “X data”
3. Select pop for “Y data”
4. Test various types with different fit names

Good luck in the Midterm Exam!