QF627 Programming and Computational Finance

S0102: MATLAB Basics

Learning Outcomes:

1. (DNT) Assessment:
   - Class Participation: 20 %
     - Attendance: 16%
     - Peer Evaluation: 4%
   - In-Class Exercises: 15%
   - Homework Assignments: 15%
   - Final Exam: 50%, ☑ closed-book / ☐ open-book, ☑ 2 hours / ☐ 3 hours
2. ☑ (DNT) We prefer to run MATLAB programs in the MATLAB IDE, though we can also run a script through the "Command Prompt" or "Terminal".
3. (DNT) MATLAB Desktop has four commonly used windows:
   - Editor (creating, editing, debugging and running scripts)
   - Command Window (running commands line by line)
   - Current Folder (the nearest folder that MATLAB programs can be accessed)
   - Workspace (list of variables that have been defined)
4. ☑ (DNT) Use of MATLAB Debugger + Workspace
5. ☑ (DNT) MATLAB "Change Current Folder" or "Add to Path".
6. ☑ (DNT) MATLAB also has "Live Editor" to create, edit and run programs.
7. MATLAB uses ☐ brackets / ☑ braces for indexing and ☐ brackets / ☑ braces for a function call.
8. ☑ True / ☐ False MATLAB command `clear` removes items from workspace, freeing up system memory.
9. ☑ True / ☐ False MATLAB command `clc` clears Command Window.
10. ☑ True / ☐ False In MATLAB, values/expressions separated by commas form a comma-separated list. Such as list, by itself, is not very useful. But when used with large and more complex data structures like MATLAB structures and cell arrays, the comma-separated list can enable you to simplify your MATLAB code.
11. ☑ True / ☐ False MATLAB uses semicolon (`;`) to suppress output in the Command Window.
12. ☑ True / ☐ False MATLAB also uses semicolon (`;`) to signify end of row in 2D arrays.
13. ☑ True / ☐ False Elements in the row (of a 2D array) can be separated by using either commas or spaces.
14. ☑ (DNT) Review of the formulation of the "HDB Loan Calculator" application

15. MATLAB Variables Names
    - starts with a ☑ letter / ☐ digit / ☐ underscore
    - followed by ☑ letters / ☑ digits / ☑ underscores
16. MATLAB is case ☑ sensitive / ☐ insensitive, so **A** and **a** ☐ are / ☑ are not the same variable.
17. (DNT) The maximum length of a variable name is the value that namelengthmax command returns. The value is 63 on my computer.
18. ☐ True / ☑ False MATLAB can use **_a** as a variable name.
19. MATLAB arithmetic operators:
    - addition: +
    - subtraction: -
    - multiplication: *
    - division: /
    - power: ^
20. Precedence of the above 5 arithmetic operators and parentheses.
    - + (☐ < / ☑ =) - (☑ < / ☐ =)* (☐ < / ☑ =)/ (☑ < / ☐ =)^
21. MATLAB expression to compute 4^{3^2}: ☐ `4^3^2` / ☑ `4^(3^2)`
22. `**` (Python) ⇔ ^ (MATLAB)
23. **`print(P)`** (Python) ⇔ disp(P) (MATLAB)
24. ☑ True / ☐ False In MATLAB, **`disp(1, 2)`** causes an **`Error`**.
25. ☑ (DNT) A temporary solution: (Python) `print(1, 2)` ⇔ (MATLAB) pythonprint(1，2)
26. MATLAB code converted from the following Python code:

Python	MATLAB

`PV=800000`
`t=25`
`r=2.6/100`
`P=(r/12*PV)/(1-(1+r/12)**(-12*t))`
`print(P)`
	`%2345678901234567890123456789012345`
	`clc`
	`PV=800000;`
	`t=25;`
	`r=2.6/100;`
	`P=(r/12*PV)/(1-(1+r)/12)^(-12*t));`
	`disp(P)`

27. (DNT) Default format of the output in MATLAB Command Window is format short
28. (DNT) GUI programming: **`objectName`** (PyQt5) ⇔ Tag (MATLAB GUIDE)
29. ☑ (DNT) Demonstration of the main steps in building the HDB Loan Calculator MATLAB GUI.
30. MATLAB function str2num converts a string to a number.
31. MATLAB function num2str converts a number to a string.

32. MATLAB mathematical functions:

Maths function	MATLAB function	Maths function	MATLAB function
$\ln(x)$	**`log(x)`**	$\tan(x)$	**`tan(x)`**
$\log_{10}(x)$	**`log10(x)`**	$\cot(x)$	**`cot(x)`**
$\log_2(x)$	**`log2(x)`**	$\sec(x)$	**`sec(x)`**
e^x	**`exp(x)`**	$\text{asin}(x)$	**`asin(x)`**
$\sin(x)$	**`sin(x)`**	$\text{acos}(x)$	**`acos(x)`**
$\cos(x)$	**`cos(x)`**	$\lfloor x \rfloor$	**`floor(x)`**
$\lvert x \rvert$	**`abs(x)`**	$\lceil x \rceil$	**`ceil(x)`**

33. [✓ True] / ☐ False MATLAB can use `007` as a numeric literal.
34. ☐ True / [✓ False] MATLAB can use `7_7` as a numeric literal.
35. ☐ True / [✓ False] MATLAB can use `0xdeadbeaf` as a numeric literal.
36. [✓ True] / ☐ False MATLAB can use `1e5` as a numeric literal.
37. MATLAB **character vector** is a sequence of characters enclosed in [✓ single] / ☐ double quotation marks.
38. ☑ (DNT) MATLAB **character array** is a 2D array that stores character vectors (of the same length) as rows.
39. Some MATLAB special characters in character vectors include
    - single quotation mark: `''`
    - single percent sign: `%%`
    - single backslash: `//`
    - new line: `\n`
40. ☐ True / [✓ False] MATLAB can use triple-quotes to create strings as Python does.
41. ☐ True / [✓ False] MATLAB can concatenate two adjacent string literals.
42. ☐ True / [✓ False] MATLAB can use addition (**+**) to concatenate two string literals.
43. [✓ True] / ☐ False MATLAB can use the **`[]`** operator to concatenate two strings.
44. [✓ True] / ☐ False MATLAB can use function **`strcat`** to concatenate two strings.
45. [✓ True] / ☐ False MATLAB can use function **`strcmp`** to compare equality of two strings.
46. [✓ True] / ☐ False MATLAB can use function **`strfind`** to find all occurrence of one string in another. If not found, the function returns **`[]`**.
47. [✓ True] / ☐ False MATLAB indexing and slicing use braces.
48. [✓ True] / ☐ False e In MATLAB, the index starts from 1.
49. [✓ True] / ☐ False In MATLAB, negative index is not allowed. Subscript indices must either be real positive integers or logicals.
50. [✓ True] / ☐ False In MATLAB, slicing **`start:end`** includes **`end`**.

51. ☑ True / ☐ False In MATLAB, extended slicing's format is **`start:step:end`**.
52. ☑ True / ☐ False In MATLAB, **`x(1:1)`** is not empty.
53. ☑ True / ☐ False In MATLAB, **`x(3:1)`** is empty.
54. ☑ True / ☐ False In MATLAB, **`1:3`** creates the array **`[1, 2, 3]`**.
55. ☑ True / ☐ False In MATLAB, **`x(1:3)`** is equivalent to **`x([1,2,3])`**.
56. Output of the following MATLAB code is: `'accb'`

```
clear;
clc;
x='abcdef';
x([1,3,3,2])
```

57. Output of the following MATLAB code is: `'ollellleeo'`

```
clear;
clc;
x='Hello';
x([5,3,3,2,3,4,4,2,2,5])
```

58. Remainder: **`5%2`** (Python) ⬄ `mod(5,2)` (MATLAB)
59. Integer Division: **`5//2`** (Python) ⬄ `floor(5/2)` (MATLAB)
60. ☑ True / ☐ False MATLAB uses `...` for (explicit) line joining.
61. ☑ True / ☐ False MATLAB does not have implicit line joining as Python has.
62. ☑ True / ☐ False MATLAB uses `%` for comments.
63. ☑ True / ☐ False MATLAB does not require any indentation for a code block.
64. ☑ True / ☐ False MATLAB needs to save the function definition in an M-file.
65. ☑ True / ☐ False In an M-file, only the first function (i.e. the main function) is visible to others. Additional functions defined are for internal use only.
66. The MATLAB function to compute monthly installment of the HDB Loan on slide 74 is

```
%234567890123456789012345678901234567890123456789012345678901234567890
function [P]=funP(PV, r, t)
P=(r/12*PV)/(1-(1+r/12)^(-12*t));
end
```

67. ☑ True / ☐ False MATLAB does not need to **`import`** anything as Python does.

68. Convert the following Python code to MATLAB, using the same function name.

Python

```
from scipy.stats import norm
from math import *
def BS_EuroCallPut(S,K,r,q,sigma,T,t):
   d1=(log(S/K)+(r-q+sigma**2/2.)*(T-t))/(sigma*sqrt(T-t))
   d2=d1-sigma*sqrt(T-t)
   c=S*exp(-q*(T-t))*norm.cdf(d1)-K*exp(-r*(T-t))*norm.cdf(d2)
   p=K*exp(-r*(T-t))*norm.cdf(-d2)-S*exp(-q*(T-t))*norm.cdf(-d1)
   return [c, p]

r=BS_EuroCallPut(50,50,0.04,0.01,0.4,0.5,0)
print(r)
```

MATLAB Function

```
%234567890123456789012345678901234567890123456789012345678901234567890
function [c,p]=BS_EuroCallPut(S,K,r,q,sigma,T,t)
d1=(log(S/K)+(r-q+sigma^2/2)*(T-t))/(sigma*sqrt(T-t));
d2=d1-sigma*sqrt(T-t);
c=S*exp(-q*(T-t))*normcdf(d1)-K*exp(-r*(T-t))*normcdf(d2);
p=K*exp(-r*(T-t))*normcdf(-d2)-S*exp(-q*(T-t))*normcdf(-d1);
end
```

MATLAB Script

```
clear;
clc;
[c, p]=BS_EuroCallPut(50,50,0.04,0.01,0.4,0.5,0)
```

69. What is the output on slide 81, when we run the following command:

```
>> r=BS_EuroCallPut(50,50,0.04,0.01,0.4,0.5,0)
r =
 5.9316
```

70. ☑ True / ☐ False MATLAB does not have keyword arguments.
71. ☑ True / ☐ False MATLAB does not have keyword-only parameters.
72. Convert the following Python `lambda` expression (for an anonymous function) to a MATLAB anonymous function with the same function name.

Python	MATLAB
`f=lambda x,y: x+y`	`f=@(x,y) x+y;`

73. The MATLAB command to evaluate the first function in the following function handle cell array at `x=1` is: `f{1}(1)`

```
f={@(x) x+1, @(x,y) x+y};
```

74. ☑ True / ☐ False In MATLAB `['ab', 'cde']` results `'abcde'`.
75. ☑ True / ☐ False In MATLAB `[1, 2]` results an array of 2 numbers.

76. ☐ True / ☑ False In MATLAB **`x=[1, 'ab']`** results an array of 2 items, where **`x(1)`** is **`{1}`** and **`x(2)`** is **`{'ab'}`**.
77. ☑ True / ☐ False In MATLAB **`x={1, 'ab'}`** results a cell array of 2 items, where **`x{1}`** is **`1`** and **`x{2}`** is **`'ab'`**.
78. ☑ True / ☐ False MATLAB does not have nested arrays. **`[[1,2],[3,4]]`** reduces to **`[1,2,3,4]`**.
79. ☑ True / ☐ False MATLAB has nested cell arrays, e.g. **`{{1,2},{3,4}}`**.
80. In MATLAB, if **`x={{1,2},{3,4}}`**, the values for the following notations are

Notation	Value	Notation	Value
`x(1)`	☐ Error / ☑ {1x2 cell}	**`x{1}`**	☐ Error / ☑ {[1]}{[2]}
`x(1)(1)`	☑ Error / ☐ ____	**`x(1){1}`**	☑ Error / ☐ ____
`x{1}{1}`	☐ Error / ☑ 1	**`x{1}(1)`**	☐ Error / ☑ {[1]}

81. In MATLAB, if **`x={1,2,3}`**, after the following assignment, values of **`a`**, **`b`** are

Assignment	Values of **`a`**, **`b`**	Assignment	Values of **`a`**, **`b`**
`[a,b]=x{1:2}`	☐ Error / ☑ **a=**1 **b=**2	**`[a,b]=x{:}`**	☐ Error / ☑ **a=**1 **b=**2
`[a,b]=deal(x{1:2})`	☐ Error / ☑ **a=**1 **b=**2	**`[a,b]=deal(x{:})`**	☑ Error / ☐ **a=**____ **b=**____
`[a,b]=1, 2`	☑ Error / ☐ **a=**____ **b=**____	**`[a,b]=deal(1,2)`**	☐ Error / ☑ **a=**1 **b=**2

82. ☑ True / ☐ False Extracting multiple elements from a cell array yields a comma-separated list.
83. ☑ True / ☐ False MATLAB **`deal`** function requires the number of outputs to match the number of inputs.
84. ☐ True / ☑ False In the output, character vectors do not show single-quotation marks.
85. ☐ True / ☑ False In the output, cell arrays do not show curly braces. Character vectors in a cell array are shown with single-quotation marks.
86. ☑ True / ☐ False In the output, arrays of numbers do not show square brackets.
87. ☐ True / ☑ False In the output, cell arrays do not show curly braces. Arrays of numbers in a cell array are shown with square brackets.
88. ☑ True / ☐ False In the output, arrays and cell arrays in a cell array are not shown in details, but a summary respectively in square brackets and curly braces, e.g. `[1x2 double]` or `{1x2 cell}`.

89. ☑ True / ☐ False In MATLAB, we can use `[]` operator or function **`horzcat`** to add items to a cell array. To add items from every item in an array, we need to use function **`num2cell`**, otherwise the whole array will be added to the cell array as a single item.
90. ☑ True / ☐ False In MATLAB, we can define **`x.a`** directly without having **`x`** defined, e.g. **`x.a=1`**. This creates a structure **`x`**, and **`a`** is a field name. More field names can be added to **`x`** by simple assignments, e.g. **`x.b=2`**. MATLAB function **`fieldnames`** can be used to return all the field names of a structure. We can also create a structure using the **`struct`** function.
91. ☑ True / ☐ False In MATLAB, structures with the same field names can be concatenated (using the `[]` operator) into an array of structures, or a structure array.
92. ☑ True / ☐ False For a structure array, extracting a field of the structure yields a comma-separated list.
93. ☑ True / ☐ False MATLAB does not have membership test operators. We can use the MATLAB function **`ismember`**, but it does not simply return a single value of True or False. The output depends on the inputs and the number of outputs.
94. ☑ True / ☐ False In the output, MATLAB displays **1** for true and **0** for false.
95. ☑ True / ☐ False In MATLAB, function **`ismember`** cannot tell whether one string is a substring of another. Function **`strfind`** can help.
96. ☑ True / ☐ False In MATLAB, a non-zero number represents true.
97. ☑ True / ☐ False In MATLAB, empty array and empty string do not represent true or false. We need to use function **`isempty`** to obtain a truth value.
98. ☑ True / ☐ False MATLAB does not support chained comparison.
99. Output of the MATLAB expression **`1<10<5`**: 1
100. ☑ True / ☐ False In MATLAB, **`Inf`** (or **`inf`**) values are equal to other **`Inf`** (or **`inf`**) values.
101. ☑ True / ☐ False In MATLAB, **`NaN`** (or **`nan`**) values are not equal to any other numerical values, including other **`NaN`** (or **`nan`**) values.
102. ☑ True / ☐ False In MATLAB, **`1==~1`** returns **0** instead of causing a syntax error. This is because logical not (**~**) has a higher precedence than equality relational operator (**==**).
103. ☑ True / ☐ False MATLAB does not have augmented assignment.
104. ☑ True / ☐ False MATLAB has some functions for set operations, e.g. **`intersect`**, **`ismember`**, **`setdiff`**, **`union`** and **`unique`**. These functions work on arrays of numbers, cell arrays of strings or character vectors. These functions do not work on cell arrays of numbers.
105. ☑ True / ☐ False MATLAB has **`return`** statement, which return control to invoking function.

106. Convert the following Python code to MATLAB. Manage to use the same variable names.

Python

```python
x=150_000
if 0<=x<20_000:
    y=0
elif 20_000<=x<30_000:
    y=0+0.02*(x-20_000)
elif 30_000<=x<40_000:
    y=200+0.035*(x-30_000)
elif 40_000<=x<80_000:
    y=550+0.07*(x-40_000)
elif 80_000<=x<120_000:
    y=3_350+0.115*(x-80_000)
elif 120_000<=x<160_000:
    y=7_950+0.15*(x-120_000)
elif 160_000<=x<200_000:
    y=13_950+0.18*(x-160_000)
elif 200_000<=x<240_000:
    y=21_150+0.19*(x-200_000)
elif 240_000<=x<280_000:
    y=28_750+0.195*(x-240_000)
elif 280_000<=x<320_000:
    y=36_550+0.2*(x-280_000)
else:
    y=44_550+0.22*(x-320_000)
print(y)
```

MATLAB

```matlab
%2345678901234567890123456789012345678901234567890123456789012345678
x=150000;
if 0<=x && x<20000
y=0
elseif 20000<=x && x<30000
y=0+0.02*(x-20000)

elseif 30000<=x && x<40000
y=200+0.035*(x-30000)
elseif 40000<=x && x<80000
y=550+0.07*(x-40000)
elseif 80000<=x && x<120000
y=3350+0.115*(x-80000)
elseif 120000<=x && x<160000
y=7950+0.15*(x-120000)
elseif 160000<=x && x<200000
y=12950+0.18*(x-160000)
elseif 200000<=x && x<240000
y=21150+0.19*(x-200000)
elseif 240000<=x && x<280000
y=28750+0.195*(x-240000)
elseif 280000<=x && x<320000
y=36550+0.2*(x-280000)
else
y=44550+0.22*(x-320000)
end
disp(y)
```

107. Convert the following Python code to MATLAB. Manage to use the same variable names.

Python

```
bi=[0, 200, 550, 3350, 7950, 13950, 21150, 28750, 36550, 44550]
mi=[2.0, 3.5, 7.0, 11.5, 15.0, 18.0, 19.0, 19.5, 20.0, 22.0]
xi=[20000, 30000, 40000, 80000, 120000, 160000, 200000, 240000,
    280000, 320000]
x=400_000
if x>xi[-1]:
    i=len(xi)-1
else:
    i=next(filter(lambda w: w[1]>x, enumerate(xi)))[0]-1

if i==-1:
    y=0
else:
    y=bi[i]+mi[i]/100*(x-xi[i])
print(y)
```

MATLAB

```
%234567890123456789012345678901234567890123456789012345678901234567890
bi=[0, 200, 550, 3350, 7950, 13950, 21150, 28750, 36550, 44550];
mi=[2.0, 3.5, 7.0, 11.5, 15.0, 18.0, 19.0, 19.5, 20.0, 22.0];
xi=[20000, 30000, 40000, 80000, 120000, 160000, 200000, 240000, 280000, 320000];
x=400000;
if x>xi(end)
i=length(xi);
else
c=find(xi>x);
i=c(1)-1;
end
if i==0
y=0;
else
y=bi(i)+mi(i)/100*(x-xi(i));
end
disp(y)
```

108. [✓ True] / ☐ False The `switch` block tests each case until one of the case expressions is true.

109. Complete the following table:

A	isscalar(A)	isvector(A)	ismatrix(A)	isempty(A)
2	1	1	1	0
[2, 2]	0	1	1	0
[2; 2]	0	1	1	0
[2 2;2 2]	0	0	1	0
[]	0	0	1	1
'a'	1	1	1	0
'abc'	0	1	1	0
''	0	0	1	1

110. [✓ True] / ☐ False MATLAB has functions **`arrayfun`**, **`cellfun`** and **`strctfun`** to apply a simple function to each element of an array, each cell of a cell array and each field of a scalar structure, respectively.

111. What is the output the following MATLAB code?

MATLAB code		

`for i=1:3`
`   disp(i)`
`end`
	`x=1:3`
	`for i=x`
	`   disp(i)`
	`end`
		`x=[1, 2, 3]`
		`for i=x`
		`   disp(i)`
		`end`

Output		

`%2345678901234567890`
`1`
`2`
`3`
	`%2345678901234567890`
	`1  2  3`
	`1`
	`2`
	`3`
		`%2345678901234567890`
		`1  2  3`
		`1`
		`2`
		`3`

112. What is the output the following MATLAB code?

MATLAB code		

`x=1:3`
`for i=x`
`   x(3)=4;`
`   disp(i)`
`end`
	`x=1:3`
	`for i=x`
	`   i=5;`
	`   disp(i)`
	`end`
		`for i='abc'`
		`   disp(i)`
		`   i=2;`
		`end`

Output		

`%2345678901234567890`
`1  2  3`
`1`
`2`
`3`
	`%2345678901234567890`
	`1  2  3`
	`5`
	`5`
	`5`
		`%2345678901234567890`
		`a`
		`b`
		`c`

113. What is the output the following MATLAB code?

MATLAB code	Output

`for i={'abc', 1, 'abc'}`
`   disp(i)`
`end`
	`%2345678901234567890`
	`'abc'`
	`[1]`
	`'abc'`

114. ☑ True / ☐ False MATLAB **`for`** loop and **`while`** loop do not have the "**`else`**" clause.
115. ☑ True / ☐ False (MATLAB) **`continue`** passes control to next iteration of the **`for`** or **`while`** loop. **`break`** terminates execution of **`for`** or **`while`** loop.
116. ☑ True / ☐ False MATLAB, as all other programming languages, allows recursive function definition.
117. Convert the following Python class definition and its simple application to MATLAB.

Python code

```
import math
class A:
   value=1
   def __init__(self, val):
      self.value=val
   def roundOff(self):
      return round(self.value, 2)

a=A(math.pi)
a.roundOff()
```

MATLAB class definition

```
%234567890123456789012345678901234567890123456789012345678901234567890
classdef A
   proprties
     value
   end
   methods
      function obj=A(val)
         obj.value=val;
      end
      function r=roundoff(obj)
           r=round(obj.value,2);
      end
   end
end
```

MATLAB script

```
a=A(pi)
a=
  A with properties:
  value:3.1416
a.roundoff()
ans=
   3.1400
```