MATH2221
Mathematics Laboratory II

Lecture 4: More on Conditional Statements

Gary Choi

February 4, 2025

Recall: Writing your own MATLAB function

- You can create your own MATLAB function by writing a script in the form:

```
function [y1,...,yn] = myfun(x1,...,xm)
      Statement
      …
end
```

Example:

```
function [x1,x2] = quadratic_formula(a,b,c)
    x1 = (-b + sqrt(b^2-4*a*c))/(2*a);
    x2 = (-b - sqrt(b^2-4*a*c))/(2*a);
end
```

- You can also create a function handle using @

Example:

```
f = @(x,y) 2*x+3*y;
```

Recall: Relational and logical operators

- Relational operators
  - A == B (is equal to)
  - a ~= b (is not equal to)
  - c > d (is greater than)
  - e <= f (is less than or equal to)
  - …
- Logical operators
  - A | B (or), A || B (short-circuit or)
  - (2-1 == 1) & (3^2 == 9) (and),
    (a==1) && (b~=2) (short-circuit and)
  - …
- Returning an array of logical values (0 or 1)

Limitation of short-circuit and/or:
Each expression must evaluate to a scalar logical result.
Example:

```
>> ([1,2] == 1) | ([3,4]~=0)  % OK
ans =
  1×2 logical array
   1   1
>> ([1,2] == 1) || ([3,4]~=0)  % not OK
```

Operands to the logical and (&&) and or (||) operators must be convertible to logical scalar values.
See also:
https://www.mathworks.com/help/matlab/ref/shortcircuitor.html
https://www.mathworks.com/help/matlab/ref/shortcircuitand.html

Recall: *if/elseif/else* statements

- **if-end statement:**

```
if expression
    statements                    % do this if the expression is true
end
```

- Example:

```
a = 3*4;
b = 2*6;
if a == b
    disp('a and b are equal');    % disp: display text/string/array
end
```

Recall: *if/elseif/else* statements

- **if-else-end statement:**

```
if expression
    statements1              % do this if the expression is true
else
    statements2              % do this if the expression is false
end
```

- Example:

```
a = 10;
b = 7;
if a > b
   disp('a is greater than b');
else
   disp('a is not greater than b');
end
```

Recall: *if/elseif/else* statements

- **if-elseif-else-end statement:**

```
if expression1
    statements1              % do this if expression1 is true

elseif expression2
    statements2              % do this if expression1 is false
                                 but expression2 is true
elseif expression3
    statement3               % do this if expression1 and expression 2 are false
                                 but expression3 is true
else
    statementsN               % do this if all the above expressions are false

end
```

Example of *if/elseif/else* statements

- Example: HK income tax

  Consider the following HK income tax scheme.

Net chargeable income	Progressive rate
on the first $50,000	2%
on the next $50,000	6%
on the next $50,000	10%
on the next $50,000	14%
on the remainder	17%

  Exercise: How to write a function income_tax(N) to compute the income tax for any given net chargeable income N?

Example of *if/elseif/else* statements

- Solution:

```
function tax = income_tax(N)

if N <= 50000
    tax = N*0.02;
elseif N <= 50000*2
    tax = 50000*0.02 + (N-50000)*0.06;
elseif N <= 50000*3
    tax = 50000*0.02 + 50000*0.06 + (N-2*50000)*0.10;
elseif N <= 50000*4
    tax = 50000*0.02 + 50000*0.06 + 50000*0.10 + (N-3*50000)*0.14;
else
    tax = 50000*0.02 + 50000*0.06 + 50000*0.10 + 50000*0.14 + (N-4*50000)*0.17;
end

end
```

switch/case/otherwise statements

- **switch/case/otherwise statement:**

```
switch switch_expression

   case case_expression1
      statement1

   case case_expression2
      statement2
       ...
   otherwise
      statementN

end
```

```
% Similar to if-elseif-else-end:

% if switch_expression == case_expression1 (for scalar)
% (or if strcmp(switch_expression,case_expression1) (for string))
%       do statement1

% elseif switch_expression == case_expression2 (for scalar)
% (or elseif strcmp(switch_expression,case_expression2) (for string))
%       do statement2

% else
%         statementN
%
% end
```

Example of *switch/case/otherwise* statements

- Example:

```
n = input('Enter a number: ');

switch n
    case -1
        disp('negative one')
    case 0
        disp('zero')
    case 1
        disp('positive one')
    otherwise
        disp('other value')
end
```

Equivalent to:

```
if n == -1
    …
elseif n == 0
    …
elseif n == 1
    …
else
    …
end
```

Example of *switch/case/otherwise* statements

- Example:

```
day = 'Tue';

switch day
    case 'Tue'
        disp('MATH2221 Lecture');

    case 'Thu'
        disp('MATH2221 Lab');

    otherwise
        disp('Missing MATH2221');

end
```

Equivalent to:

```
if strcmp(day, 'Tue')
    …

elseif strcmp(day, 'Thu')
    …

else
    …

end
```

if/elseif/else statements vs *switch/case/otherwise* statements

- if/elseif/else statements:
  - Easier to write with mixed classes or complicated condition tests, e.g.

```
if a == 1
    …
elseif (b >= 3) || (c < 1)
    …
end
```

- *switch/case/otherwise* statements:
  - More organized when handling multiple conditions with the same class, e.g.

```
switch n
case 1      % i.e. if n == 1
    …
case -1     % i.e. elseif n == -1
    …
end
```

for statements (*for* loop)

- **for loop: loop a set of statements repeatedly**

```
for i = (a given vector)
    statements;          % Do this for every element i in the vector
    …
end
```

- Most common for loops:

```
for i = 1:n              % index = start:end
    statements;          % every statement here is repeated n times
    …                    % i.e. do them for i = 1, then for i = 2, …, for i = n
end

for j = m:s:n            % index = start:increment:end
    statements;          % every statement here is repeated for j = m to j = n
    …                    % with an increment of s
end
```

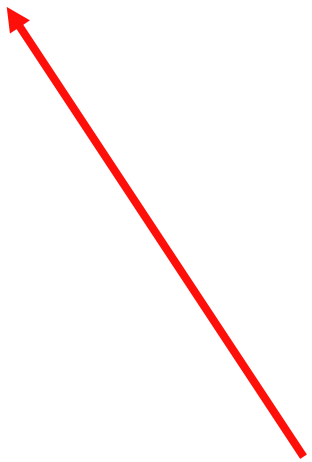
Examples of *for* loop

- Example: Compute
  $1 + 2 + \cdots + 100$

```
s = 0;                          % A variable for storing the sum

for i = 1:100
    s = s + i;                  % Add i to s for i = 1, 2, …, 100
end

s                               % The final result
```



Caution: Remember to put the semicolon ";" here if you don't want to display the value of s. Otherwise, it will mess up your command window!

Examples of *for* loop

- Note: The vector in the for loop condition does not need to be consecutive

- Example: If v = [2, 9, 1, 4, 2], compute the sum of all elements of v.

```
v = [2, 9, 1, 4, 2];

s = 0;                          % A variable for storing the sum
for i = v
    s = s + i;                  % Add i to s for each element i in v
end
s                               % The final result
```

Examples of *for* loop (summations)

- Example: Compute

$$1^2 + 2^2 + \cdots + 100^2$$

```
s = 0;                          % A variable for storing the sum
for i = 1:100
    s = s + i^2;                % Add i^2 to s for i = 1, 2, …, 100
end
s                               % The final result
```

- Example: Compute

$$1^2 + 3^2 + 5^2 \ldots + 99^2$$

```
s = 0;                          % A variable for storing the sum
for i = 1:2:99
    s = s + i^2;                % Add i^2 to s for i = 1, 3, 5, …, 99
end
s                               % The final result
```

Examples of *for* loop (products)

- Example: Compute

$$1^{1} \cdot 2^{1/2} \cdot 3^{1/3} \cdot \ldots \cdot 100^{1/100}$$

```
s = 1;                          % A variable for storing the product

for n = 1:100
    s = s*n^(1/n);              % Multiply s by n^(1/n)
end

s                               % The final result
```

Examples of *for* loop (recurrence equations)

- Example: If $F_1 = 1$, $F_2 = 1$ and $F_n = F_{n-1} + F_{n-2}$ for all $n \geq 3$, find F_{20}.

```
F = zeros(20,1);          % Define an array F to store all values of F_n
F(1) = 1;                 % Assign the initial values of F_1 and F_2
F(2) = 1;

for n = 3:20
    F(n) = F(n-1)+F(n-2); % Obtain F_n and store it in the array
end

F(20)                     % The final result
```

Examples of *for* loop (recurrence equations)

- Example: If $F_1 = 1$, $F_2 = 1$ and $F_n = F_{n-1} + F_{n-2}$ for all $n \geq 3$, find F_{20}.

Alternative approach without storing all values:

```
a = 1;                  % The initial values, where a represents F_{n-2} and
b = 1;                  % b represents F_{n-1}

for n = 3:20
    c = b+a;            % Obtain F_n and store it as c
    a = b;              % Update a using the latest value of b
    b = c;              % Update b using the latest value of c
end                     % (note: the order of the operations is important)!

c                       % The final result
```

while statements (*while* loop)

- ***while* loop: repeat the statements until the condition is NOT satisfied**

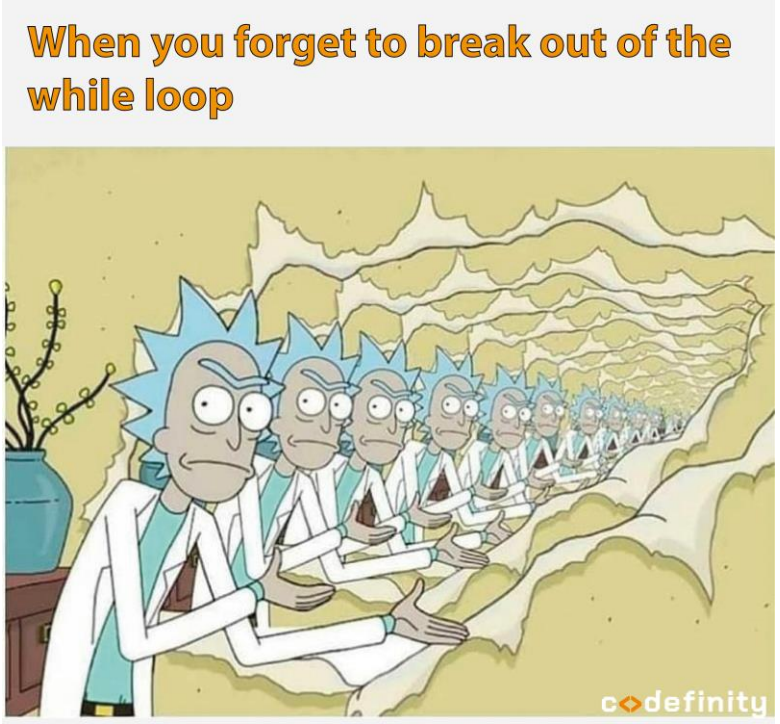
```
while condition
    statements        % repeating the statements while the condition is true (1)
end
```

- Example: Find the smallest n such that $1 + 2 + \cdots + n > 1000$

```
s = 0;                % s: a variable for storing the sum
n = 0;                % n: start with 0 and to be updated in the while loop
while s <= 1000       % while s ≤ 1000, do the following:
    n = n + 1;        % - Update n with n+1
    s = s + n;        % - Add n to the current s
end
n
```

while statements (*while* loop)

- **Caution:**
  - It is easy to get into an endless loop if the *while* loop is set up improperly!
  - Remember to check and see whether you have updated all relevant variables in each *while* loop
  - Use Ctrl+C to stop the iterations if needed



- Example: Sum 100 random number

```
a = rand(1,100);
s = 0;                  % Storing the sum
j=1;                    % Set the counter variable
while j < 101
        s = s+a(j);     % Update the sum
        j = j+1;        % Update the counter variable
                        % If you don't add this step, then it will loop infinitely!
end
s
```

Example of *while* loop (recurrence equations)

- Example: If $F_1 = 1$, $F_2 = 1$ and $F_n = F_{n-1} + F_{n-2}$ for all $n \geq 3$, find the smallest n such that $F_n \geq 1000$.

```
F = [1,1];                  % Create an array with the initial values of F1 and F2
n = 2;

while F(n) < 1000
    n = n + 1;              % Update n with n + 1
    s = F(n-1)+F(n-2);      % Obtain the value of Fn
    F = [F, s];             % Append the value to the current array F
end
n
```

for loops vs *while* loops

- *for* loops:
  - Usually used when you need to repeat some operations for a fixed number of times

    Example:
    - Construct a matrix with some given size
    - Do some update/sampling for 100 times
    - …

- *while* loops:
  - Usually used when you don't know how many times you need to repeat but have a known stopping criterion

    Example:
    - Find the smallest n/smallest number of steps such that …
    - Repeat the operations until the result converges (with error less than a threshold)
    - …

Nested loops

- Note that you can combine *for* loops, *while* loops, *if/elseif/else* statements etc. in your operations

```
for i = 1:m
    statementA;                 % For every i, statementA will be run once
    for j = 1:n
        for k = 1:p
            statementB;         % For every combination of (i,j,k),
             …                      statementB will be run once
        end
        while condition
            statementC;         % For every combination of (i,j), statementC will be run
            …                      multiple times until “condition” is not satisfied
        end
    end
end
```

Examples of nested loops

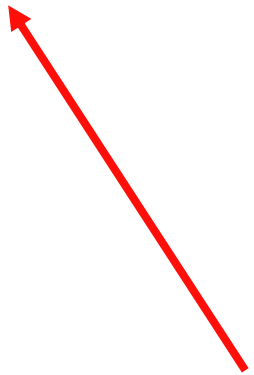
- Example: Compute

$$\sum_{i=1}^{5}\sum_{j=1}^{10} ij$$

```
s = 0;                          % A variable for storing the sum

for i = 1:5                     % In the outer for loop, the process below will be
    for j = 1:10                repeated for i = 1,2,3,4,5
        s = s + i*j;            % In the inner for loop, we have i*1+i*2+…+i*10
    end
end

s                               % The final result
```



Note: In this case, the two *for* loops can be swapped

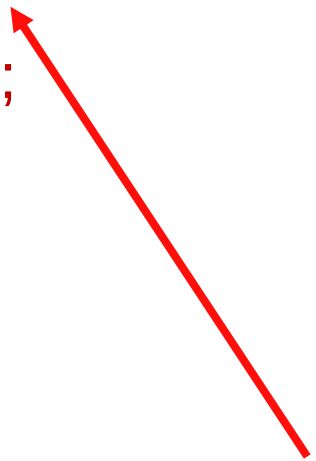
Examples of nested loops

- Example: Compute
  $1 + (1 + 2) + (1 + 2 + 3) + \cdots + (1 + 2 + \cdots + 10)$

```
s = 0;                          % A variable for storing the sum

for m = 1:10                    % In the inner for-loop, we add 1+2+…+m to the
    for n = 1:m                 variable s
        s = s + n;              % In the outer for-loop, we repeat this action
    end                         for m = 1, …, 10
end

s                               % The final result
```



Note: We can put m in the vector 1:m here, as m is defined by the outer *for* loop. However, we CANNOT swap the two *for* loops in this case.

Examples of nested loops

- Example: Compute

$$\sum_{i=1}^{5}\sum_{j=1}^{10} f(i,j)$$

where $f(i,j) = \begin{cases} i^2 + j^2 & \text{if } i+j \text{ is odd} \\ ij & \text{if } i+j \text{ is even} \end{cases}$

```
s = 0;
for i = 1:5
    for j = 1:10
        if mod(i+j,2) == 1                % Combined with a if-else statement here
           s = s + (i^2 + j^2);
        else
           s = s + i*j;
        end
    end
end
s
```

Examples of nested loops

- Example: Compute

$$\sum_{i=1}^{10} \frac{1}{p_i}$$

where p_i is the i-th prime

```
s = 0;                          % s: a variable for storing the sum
i = 1;                          % i: serves as a counter
N = 1;                          % N: a positive integer, to be updated in the while loop
while i <=10                    % While we have NOT added 10 terms, do the following:
    if isprime(N)               • Check whether N is a prime.
        s = s + 1/N;            • If yes, then N is the i-th prime. We add 1/N to the
        i = i + 1;                current sum s and update i with i+1.
    end                         • (Regardless of yes or no) Update N with N+1
    N = N+1;
end
s                               % The final result
```

Some practical considerations about the efficiency

- In many situations, *for* loops/*while* loops can get the job done but may not be the fastest way!

- MATLAB is optimized for operations involving matrices and vectors, so it is a good practice to vectorize your code (i.e. use vector/matrix-based operations and avoid loops) whenever possible
  (See also: https://www.mathworks.com/help/matlab/matlab_prog/vectorization.html)

  Example: Compute $1 + 2 + \cdots + 100$
  sum(1:100)

  Example: Compute $1^2 + 2^2 + \cdots + 100^2$
  sum((1:100).^2)

  Example: Compute $1^1 \cdot 2^{1/2} \cdot 3^{1/3} \cdot \cdots \cdot 100^{1/100}$
  prod((1:100).^(1./(1:100)))

Some practical considerations about the efficiency

- You can check the computational time of your operations using tic and toc:

```
tic;                % start/restart the timer
Statements;         % Your statements
…
toc;                % stop the timer and output the total time taken
```

- Example: Summing 100000 random values

```
a = rand(100000,1);

% Method 1
tic;
s = 0;
for i = 1:100000
    s = s + a(i);
end
toc;
```

Elapsed time is 0.001443 seconds.

```
% Method 2
tic;
s = sum(a);
toc;
```

Much faster!

Elapsed time is 0.000241 seconds.

Reminder: Lab 3 this week

January

Sun	Mon	Tue	Wed	Thu	Fri	Sat
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	[28]	[29]	[30]	[31]	

February

Sun	Mon	Tue	Wed	Thu	Fri	Sat
						[1]
[2]	[3]	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	1

March

Sun	Mon	Tue	Wed	Thu	Fri	Sat
2	[3]	[4]	[5]	[6]	[7]	[8]
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

April

Sun	Mon	Tue	Wed	Thu	Fri	Sat
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17		

Lecture 1- Lecture 13

Lab 1 - Lab 10 (40%)

Test 1 (30%)
Test 2 (30%)

Thank you!

Next time:

- More remarks on loops and conditional statements
- Visualization