Selection Statement

Chul Min Yeum
Assistant Professor
Civil and Environmental Engineering
University of Waterloo, Canada

AE121: Computational Method

UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING

Last updated: 2019-05-03

Recall Relational Expressions

- The relational operators in MATLAB are:

>	greater than
<	less than
>=	greater than or equals
<=	less than or equals
==	equality
~=	inequality

- The resulting type is logical 1 for true or 0 for false
- The logical operators are:

||	or for scalars (and)
&&	and for scalars (or)
~	not

- Also, **xor** function which returns logical true if only one of the arguments is true

Example: Relational Expressions

```
is_true01 = (3> 3)
is_true02 = (3>= 3)
is_true03 = (3 == 3)
is_true04 = ('a' == 3)
is_true05 = ('a' == 97)
is_true06 = or((3>4), (4>3))
is_true07 = ((3>4)||(4>3))
is_true08 = and((3>4), (4>3))
is_true09 = ((3>4)&&(4>3))
is_true10 = xor((3>4), (4>3))
```

```
is_true01 = logical
   0
is_true02 = logical
   1
is_true03 = logical
   1
is_true04 = logical
   0
is_true05 = logical
   1
is_true06 = logical
   1
is_true07 = logical
   1
is_true08 = logical
   0
is_true09 = logical
   0
is_true10 = logical
   1
```

The “is” Functions

- There are many “is” functions in MATLAB that essentially ask a true/false question, and return logical 1 for true or 0 for false
- **isscalar** returns logical 1 (true) if size(A) returns [1 1], and logical 0 (false) otherwise.
- **isvector** returns logical 1 (true) if size(A) returns [1 n] or [n 1] with a nonnegative integer value n, and logical 0 (false) otherwise.
- **ismatrix** returns logical 1 (true) if size(V) returns [m n] with nonnegative integer values m and n, and logical 0 (false) otherwise.
- **isrow** returns logical 1 (true) if size(V) returns [1 n] with a nonnegative integer value n, and logical 0 (false) otherwise.
- **iscolumn** returns logical 1 (true) if size(V) returns [n 1] with a nonnegative integer value n, and logical 0 (false) otherwise.

The “is” Functions (Continue)

- **isempty** returns 1 if the variable argument is empty, or 0 if not.
- **ischar** returns logical 1 (true) if A is a character array and logical 0 (false) otherwise.
- **isspace** returns a logical array TF. If A is a character array or string scalar, then the elements of TF are logical 1 (true) where corresponding characters in A are space characters, and logical 0 (false) elsewhere. isspace recognizes all Unicode® whitespace characters.
- **isletter** returns 1 or 0 for every character in a string – whether it is a letter of the alphabet or not.

Example: isscalar, isvector, and ismatrix

```
is_scalar1 = isscalar(3)
is_scalar2 = isscalar([1 2 ])
is_scalar3 = isscalar([1 2;3 4])
is_scalar4 = isscalar('ab')

unknown_val = 3;
is_scalar5 = (numel(unknown_val)==1)

is_vector1 = isvector(3)
is_vector2 = isvector([1 2 ])
is_vector3 = isvector([1 2;3 4])
is_vector4 = isvector('ab')

is_matrix1 = ismatrix(3)
is_matrix2 = ismatrix([1 2 ])
is_matrix3 = ismatrix([1 2;3 4])
is_matrix4 = ismatrix('ab')
```

```
is_scalar1 = logical
   1
is_scalar2 = logical
   0
is_scalar3 = logical
   0
is_scalar4 = logical
   0
is_scalar5 = logical
   1
is_vector1 = logical
   1
is_vector2 = logical
   1
is_vector3 = logical
   0
is_vector4 = logical
   1
```

```
is_matrix1 = logical
   1
is_matrix2 = logical
   1
is_matrix3 = logical
   1
is_matrix4 = logical
   1
```

- A scalar is considered to be a 1 x 1 vector or 1 x 1 matrix.
- A n x 1 vector is considered to be a n x 1 matrix.

Example: isrow, iscolumn, isempty

```
vec1 = ones(3,1);

is_row1 = isrow(vec1)
is_row2 = isrow(vec1')
is_row3 = isrow(2)
is_row4 = (size(vec1,1) == 1)

is_column1 = iscolumn(vec1)
is_column2 = iscolumn(vec1')
is_column3 = iscolumn(3)
is_column4 = (size(vec1,2) == 1)
```

```
is_row1 = logical
   0
is_row2 = logical
   1
is_row3 = logical
   1
is_row4 = logical
   0
is_column1 = logical
   1
is_column2 = logical
   0
is_column3 = logical
   1
is_column4 = logical
   1
```

```
vec1 = [1 2 3];
is_empty1 = isempty([]);

vec2 = vec1;
vec2(1:end) = [];
is_empty2 = isempty(vec2);

vec3 = find(vec1 == 4);
is_empty3 = isempty(vec3);

is_empty1
is_empty2
is_empty3
```

```
is_empty1 = logical
   1
is_empty2 = logical
   1
is_empty3 = logical
   1
```

Example: ischar, isletter, and isspace

```
is_char1 = ischar('a')
is_char2 = ischar(97)
is_char3 = ischar(char(97))

is_letter1 = isletter('a')
is_letter2 = isletter('ab')
is_letter3 = isletter('a2b')

char_val1 = 'a2 b';
is_letter4 = isletter(char_val1)
is_letter5 = (int8(char_val1) ~= int8(' ')) %


val = int8(' ')
is_space1 = isspace('a')
is_space2 = isspace('a b')
is_space3 = isspace('ab ')

is_space4 = isspace(char_val1)
is_space5 = (int8(char_val1) == int8(' ')) %
```

```
is_char1 = logical
   1
is_char2 = logical
   0
is_char3 = logical
   1
is_letter1 = logical
   1
is_letter2 = 1×2 logical array
     1   1

is_letter3 = 1×3 logical array
     1   0   1

is_letter4 = 1×4 logical array
     1   0   0   1

is_letter5 = 1×4 logical array
     1   1   0   1
```

```
val = int8
    32
is_space1 = logical
   0
is_space2 = 1×3 logical array
     0   1   0

is_space3 = 1×3 logical array
     0   0   1

is_space4 = 1×4 logical array
     0   0   1   0

is_space5 = 1×4 logical array
     0   0   1   0
```

If Statement

- The **if** statement is used to determine whether or not a statement or group of statements is to be executed
- General form:

```
if condition
    action
end
```

- the *condition* is any relational expression
- the *action* is any number of valid statements (including, possibly, just one)
- if the condition is true, the action is executed – otherwise, it is skipped entirely

Example: Simple Examples

```
% example 1: isscalar
x = 10;
is_scalar1 = isscalar(x);

is_scalar2 = false;
if numel(x)==1
    is_scalar2 = true;
end

is_scalar3 = (numel(x)==1);
```

```
is_scalar1
is_scalar2
is_scalar3
```

```
is_scalar1 = logical
   1
is_scalar2 = logical
   1
is_scalar3 = logical
   1
```

```
% example 2: quiz
true_answer = 10;
your_answer = 11;

is_correct1 = false;
if your_answer == true_answer
    is_correct = true;
end

is_correct2 = (your_answer == true_answer);
```

```
is_correct1
is_correct2
```

```
is_correct1 = logical
   0
is_correct2 = logical
   0
```

Representing True/False Concepts

- Note: to represent the concept of false, `0` is used. To represent the concept of true, any nonzero value can be used – so expressions like 5 or ‘x’ result in logical true
- This can lead to some common logical errors
- For example, the following expressions are always true (because the “relational expressions” on the right, 6 and ‘N’, are nonzero so they are true; therefore, it does not matter what the results of the others are):

```
(number < 5) || 6
(letter == ‘n’) || ‘N’
```

You always check your answers.

Revisit: Operator Precedence

How to make an expression of ? in other words, how to write a code to check if x lies in between 5 and 10. If yes, 1 and otherwise 0.

```
x = 1;
disp('x is 1.')
5< x <10
(5<x)<10
5<(x<10)
(5<x) & (x<10)
```

```
x is 1.
ans = logical
   1
ans = logical
   1
ans = logical
   0
ans = logical
   0
```

If-else Statements

- The **if-else** statement chooses between two actions
- General form:

```
if condition
    action1
else
    action2
end
```

- One and only one action is executed; which one depends on the value of the condition (action1 if it is logical true or action2 if it is false)

Revisit: Simple Example

```
x = 10;
is_scalar1 = isscalar(x);
is_scalar2 = [];
if numel(x) == 1
    is_scalar2 = true;
    disp('x is a scalar.');
else
    is_scalar2 = false;
    disp('x is not a scalar.');
end
```

```
x is a scalar.
```

```
is_scalar1 = logical
   1
is_scalar2 = logical
   1
```

If-else Statements are not Always Necessary!

- Simplify this statement:

```
if num < 0
   num = 0;
else
   num = num;
end
```

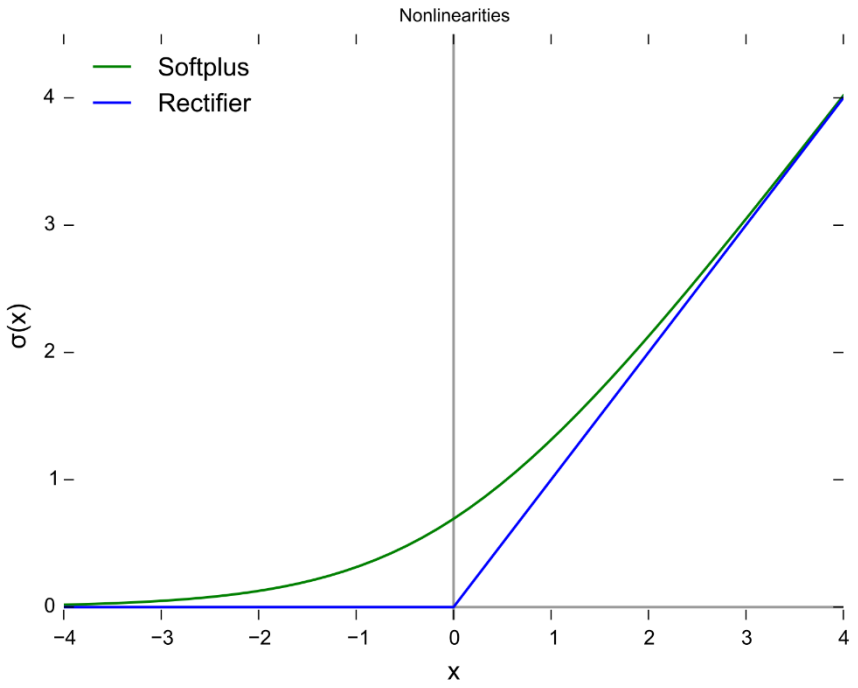
- Answer:

```
if num < 0
   num = 0;
end
```

- The point is that the **else** clause does not accomplish anything, so it is not necessary … sometimes just an **if** statement is all you need!

Example: Rectifier

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



https://en.wikipedia.org/wiki/Rectifier_(neural_networks)

```
x = 10;
if x<0
    fx1 = 0;
else
    fx1 = x;
end

fx2 = x*(x>0); % one line

x = -10;
if x<0
    fx3 = 0;
else
    fx3 = x;
end

fx4 = x*(x>0); % one line

fx1
fx2
fx3
fx4
```

```
fx1 = 10
fx2 = 10
fx3 = 0
fx4 = 0
```

Example: Check a Dimension of Your Input Vector

```
x = rand(5,1);
% x = rand(1,5);

check_error1 = 0;
if ~isrow(x) || (numel(x)~=5)
    check_error1 = 1;
end

check_error2 = 0;
if ~((size(x,1)==1) && (size(x,2) == 5))
    check_error2 = 1;
end

check_error3 = 0;
if ~and(isrow(x), (numel(x)==5))
    check_error3 = 1;
end

check_error4 = 0;
if ~isequal(size(x), [1 5])
    check_error4 = 1;
end
```

```
check_error1 = 1
check_error2 = 1
check_error3 = 1
check_error4 = 1
```

```
% x = rand(5,1);
x = rand(1,5);

check_error1 = 0;
if ~isrow(x) || (numel(x)~=5)
    check_error1 = 1;
end

check_error2 = 0;
if ~((size(x,1)==1) && (size(x,2) == 5))
    check_error2 = 1;
end

check_error3 = 0;
if ~and(isrow(x), (numel(x)==5))
    check_error3 = 1;
end

check_error4 = 0;
if ~isequal(size(x), [1 5])
    check_error4 = 1;
end
```

```
check_error1 = 0
check_error2 = 0
check_error3 = 0
check_error4 = 0
```

Throwing an Error

- MATLAB has an **error** function that can be used to display an error message in red, similar to the error messages generated by MATLAB

```
if radius <= 0
    error('Sorry; %.2f is not a valid radius\n', radius)
else
    % carry on
end
```

- When an error is thrown in a script, the script stops executing

Example: Throwing an Error

```
x = rand(1,5);
if ~isequal(size(x), [1 5])
    error('incorrect dimension of x')
end
assert(isequal(size(x), [1 5]), 'incorrect dimension of x');

x = rand(5,1);
% if ~isequal(size(x), [1 5])
%     error('incorrect dimension of x')
% end
assert(isequal(size(x), [1 5]), 'incorrect dimension of x');
```

Nested if-else Statements

- To choose from more than two actions, ***nested*** **if-else** statements can be used (an **if** or **if-else** statement as the action of another)
- General form:

```
if condition1
   action1
else
    if condition2
        action2
    else
        if condition3
            action3
        % etc: there can be many of these
        else
             actionn    % the nth action
         end
     end
end
```

The elseif Clause

- MATLAB also has an **elseif** clause which shortens the code (and cuts down on the number of ends)
- General form:

```
if condition1
    action1
elseif condition2
    action2
elseif condition3
   action3
% etc: there can be many of these
else
    actionn    % the nth action
end
```

The elseif Clause is Not Always Necessary!

- Simplify this statement:

```
if val >= 4
  disp('ok')
elseif val < 4
  disp('smaller')
end
```

- Answer:

```
if val >= 4
  disp('ok')
else
  disp('smaller')
end
```

- The point is that if you get to the else clause, you know that the expression val >= 4 is false – so, val must be less than 4 so there is no need to check that.

Example: Grading

Write a program that gives a grade based on a score. A: >=90, B: 80~90, C:70~80, D: <70

```
score = 61;
garde1 = [];
if score>=90
    grade1 = 'A';
else
    if score>=80
        grade1 = 'B';
    else
        if score >=70
            grade1 = 'C';
        else
            grade1 = 'D';
        end
    end
end
```

```
grade2 = [];
if score>=90
    grade2 = 'A';
elseif and(score>=80, score<90)
    grade2 = 'B';
elseif and(score>=70, score<80)
    grade2 = 'C';
else
    grade2 = 'D';
end
```

```
grade1 = 'D'
grade2 = 'D'
```

Example: Correct Use of elseif Statement

```
score = 85;

% this is a wrong example
if score >= 70
    disp('Your grade is C.');
elseif score > 80
    disp('Your grade is B.');
elseif score > 90
    disp('Your grade is A.');
else
    disp('Your grade is D.');
end
```

```
Your grade is C.
```

```
% correct
if score < 70
    disp('Your grade is D.');
elseif score < 80
    disp('Your grade is C.');
elseif score < 90
    disp('Your grade is B.');
else
    disp('Your grade is A.');
end
```

```
Your grade is B.
```

The Switch Statement

- The **switch** statement can frequently be used in place of a nested **if-else** statement
- General form:

```
switch switch_expression
  case caseexp1
    action1
  case caseexp2
    action2
  case caseexp3
    action3
  % etc: there can be many of these
  otherwise
    actionn
end
```

- this can be used when comparing the switch_expression to see if it is equal to the values on the case labels (the **otherwise** clause handles all other possible values)

Example: Switch Statement

Let's write a code for conducting an opposite action. A program tells you the score range when you type your grade.

```
grade = 'A';

if grade == 'A'
    disp('Your score is in the range of 90-100.');
elseif grade == 'B'
    disp('Your score is in the range of 80-90.');
elseif grade == 'C'
    disp('Your score is in the range of 70-80.');
elseif grade == 'D'
    disp('Your score is below 70.');
else
    disp('We do not have such grade.')
end
```

```
switch grade
    case 'A'
        disp('Your score is in the range of 90-100.');
    case 'B'
        disp('Your score is in the range of 80-90.');
    case 'C'
        disp('Your score is in the range of 70-80.');
    case 'D'
        disp('Your score is below 70.');
    otherwise
        disp('We do not have such grade.');
end
```

```
Your score is in the range of 90-100.
```

Common Pitfalls

- Some common pitfalls have been pointed out already; others include:
  - Using = instead of == for equality in conditions
  - Putting a space in the keyword **elseif**
  - Not using quotes when comparing a string variable to a string, such as

    letter == y

    instead of

    letter == 'y'
  - Writing conditions that are more complicated than necessary, such as

    if (x < 5) == 1 instead of just if (x < 5)

Programming Style Guidelines

- Use indentation to show the structure of a script or function. In particular, the actions in an **if** statement should be indented.
- When the **else** clause isn't needed, use an **if** statement rather than an **if-else** statement

CTRL + I : Smart Indent

Slide Credits and References

- Stormy Attaway, 2018, Matlab: A Practical Introduction to Programming and Problem Solving, 5th edition
- Lecture slides for “Matlab: A Practical Introduction to Programming and Problem Solving”
- Holly Moore, 2018, MATLAB for Engineers, 5th edition