Application of Matlab for Finance

Week 2

Shi, Yining

September 7, 2017

Today's Class

- Logical Expressions and Operators
- Flow control
- Loop structure
- User-defined functions

Logical Expressions

- A logical expression tests a conditional statement among arguments by using relational operator or logical functions
- There are six basic relational operators:
  - less than: $<$
  - less than or equal to: $<=$
  - greater than: $>$
  - greater than or equal to: $>=$
  - equal to: $==$
  - not equal to: $\sim=$

- Logical function: `isempty`, `isnan`, `all(.)`, `any(.)` etc
- Logical expressions have only two possible outcomes:
  - TRUE with value of 1
  - FALSE with value of 0

Logical Operators

The logical operators operate between simple expressions in order to create compound expressions.

- AND (A & B): returns logical 1 (true) if both A & B are true, logical 0 (false) otherwise.
- OR (A | B): returns logical 1 (true) if either A or B, or both, evaluate to true, and logical 0 (false) if neither of them are true.
- NOT (~A): returns logical 1 (true) if A evaluates to false, and logical 0 (false) if it evaluates to true.

Logical Indexing


```
1 A = 100;
2 B = 10
3 res = A > B % 1 as true
```

- `res` is a logical output of 1 as true.


```
1 A = [1,-5,5,2];
2 B = A > 2 % [0,0,1,0]
3 A(B) % 5, the 3rd element
4 A(B) = 10 % replaces the 3rd element with 10
```


- `B = [0,0,1,0]` is a matrix of logical output for `A>2` operation on each element of A.
- `A(B)` indexes the matrix A with the logical matrix B, returning a vector of elements where `A>2` is true. $\rightarrow$ element A(3)=5.
- `A(B)=10` reassigns the true element with new value 10. It changes matrix A into `A = [1,-5,10,2]`.

Logical Indexing


```
A = [1 -5 5 2];
I = find(A>2) % returns 3
A(I) % returns 5
A(I) = 10; % replaces with value 10
```


- `find(.)` performs similar function but returns directly the index number of the element in matrix A that satisfied the logical condition.
- `I = find(A>2)` returns the index value 3, implying that the 3rd element of matrix A is true for the judgement `A>2`;
- `A(I)` index the matrix `A` with `I` to read `A(3)`, which is 5;
- `A(I)=10` reassigns the true element with new value 10, `A(3)=10`. It changes matrix A into `A = [1,-5,10,2]`.

Logical Indexing


```
A = [1,-5,5,2];
B = A>0 & A<10; % B = [1,0,1,1]
A(B) % returns [1,5,2]
A(B) = NaN % change A = [NaN,-5,NaN,NaN]
isnan(A) % as [1,0,1,1]
A(isnan(A)) = [] % remove NaN elements A = -5
% A = A(~isnan(A)): similar operation
```


- Now it is a compounded logical expression for `0<A<10`
- `A(B)=NaN` replaces the logical true elements with NaN, Not a Number. NaN is often seen for missing data.

- `isnan(A)` return logical output 1 if any element of A is NaN and zero if it contains a number.
- `A(isnan(A))=[]` deletes all NaN element, leading A = -5
- `A = A(~isnan(A))` does the same thing by reassigning A as the element(s) that is not NaN.

Flow Control

- The results of logical operation are often used for decision making under flow controls.

- Eg. if A > B, do operation 1; otherwise, do operation 2.
- Main flow control structures:

  - `if`
  - `switch` (Appendix)

If structure

- The if structure evaluates a logical expression and executes a group of statements based on the value of the expression.


```
1  if logical_expression
2      statements
3  end
4
5  x = -5;
6  if x < 0
7      disp('x is negative');
8  end
```


- If the logical expression is TRUE, then all the statements between the `if` and `end` lines are EXECUTED.
- If the logical expression is FALSE, then all the statements between the `if` and `end` lines are SKIPPED.

else and elseif

- **if..else** ⇒ There is no logical expression behind `else`. The statements associated with `else` are executed only if the preceding logical expression behind `if` is false
- **if..elseif** ⇒ There is a logical expression behind the `elseif`, and will be executed if the preceding logical expression behind `if` is false.
- You can have multiple `elseif` statements within an `if` block, but only one `else` statement.


```
1 if logical_expression1
2     statement1
3 elseif logical_expression2
4     statement2
5 else
6     statement3
7 end
```

else and elseif examples

- **`x = input(.)`** asks the user to input a value via a message printed in the command window (`'Enter a number: '`)
- The value of your input is assigned to variable `x` for the following operation
- Note: Interactive operations cannot be operated in live editors.


```
1  x = input('Enter a number:');
2  if x < 0
3      disp('x is negative')
4  else
5      disp('x is not negative');
6  end
```



```
1  x = input('Enter a number');
2  if x < 0
3     disp('x is negative');
4  elseif x > 0
5     disp('x is positive');
6  else
7     disp('x is zero');
8  end
```

Exercises a.1

- Create a simple `if` structure that returns the square root of a random number x.
  - use function `randn()` to generate the random number x
  - the calculation result is stored in a variable called `res`
  - it allows an error message if x is negative


```
1  x = randn(1) %generate a random number
2
3  if x >=0
4      res = sqrt(x)
5  else % x<0
6      error('Negative number has no square root.')
7  end
```

Exercises a.2

- For a random number x, let the user choose 1 of the following 3 calculations on x: (1) the absolute value; (2) the square; (3)the square root.
- User's choice of calculation is stored in a variable called `control` (use `input()` function)
- display error messages if:
  - x is negative in the square root calculation
  - the user input a control value is not 1, 2 or 3
- display the value of `control, x, res`

Exercises a.2


```
control = input('Please choose your calculation: 1 for
    abs, 2 for square, 3 for square root:  ');
x = randn(1);
if control == 1
    res = abs(x);
elseif control == 2
    res = x^2;
elseif control == 3
    if x > 0
        res = sqrt(x);
    else
        error('Negative number has no square root.')
    end
else
    error('Invalid control variable.')
end
```

Display Results

- `disp(.)` prints contents in the command window. The content can either be
  - a variable or a matrix `disp(X)`
  - or a text message with quotation `disp('Hello World')`

- To display a combination of text and numbers, using `disp([ 'X=' ,num2str(X), 'Y=' ,num2str(Y) ])`
  - `[]` combines various inputs together
  - `num2str(.)` converts numbers to strings in order to be display together with texts inside `' '`.


```
clc % clear commond window
disp(['Your choice of calculation is ', num2str(control)])
disp(['x is ', num2str(x), ' and the result is ' ...
    ,num2str(res)])
```

Display Results

- `fprintf(formatSpec, A1, A2, ...)` also displays results `A1, A2, ...`, but with a format defined in `formatSpec`

```
1 formatSpec = ' Your choice of calculation is %d \n x ...
      is %.2f and the result is %.2f.\n';
2 fprintf(formatSpec, control, x, res)
```


- `'%'` starts the formatting operator.
- 3 `'%'` marks mean 3 variables inputs to display.
- `'%d'`, `'%.2f'` and `'%.2f'` print the first value as integer, the second and third values in the output as a floating `'f'` number with 2 digits `'.2'`.
- `'\n'` starts a new line.

Loop structure

- Loops are used to repeat the same calculations on a variable.
- Main loop structure
  - `for`
  - `while`

The for loop


```
1 for index = start_val:increment:end_val
2     statements % condition on the value of index
3 end
```


- The `for` loop structure executes repeatedly a statement or group of statements for a predetermined number of times
- The counter variable `index`
  - `index` starts at value `start_val`, increases by `increment` each time until reach the `end_val`, when the loop is terminated
  - `increment` can be omitted if increase step = 1


```
1 for index = start_val:end_val
2     statements
3 end
```

The for loop

- Create a 100×1 matrix A, whose row element is its row number plus 5.

```
1  A = zeros(100,1) % empty matrix stores the results
2  for index = 1:100
3      A(index,1) = index + 5;
4  end
```


- `index = 1, 2..., 100`
- for `index=1`, do `A(index,1)=index+5` ⇒ `A(1,1)=1+5=6`

- move to `index=2`, do `A(index,1)=index+5` ⇒ `A(2,1)=2+5=7`
- move to `index=3`, do `A(index,1)=index+5` ⇒ `A(3,1)=3+5=8`
- repeat the same calculation for each value of `index` until it reaches `100`

The while loop

The while loop structure executes a statement or group of statements repeatedly, as long as the controlling logical expression is true (that is, if it evaluates to logical 1).


```
1  while logical_expression
2      statements
3  end
4
5  i = 0; % initial value of i
6  while (i < 10)
7      i = i + 1; %overwrite i = i+1
8      disp(i)
9  end
```


As long as `i<10`, repeat operation `i=i+1` until `i=10`

Exercises b.1

- Generate a 1000×1 vector of a random variable `x`.
- Use the logic indexing to find elements of x that is positive and store them in a variable called `price`.
- Use a `for` loop to calculate the natural log of the price.
  - always create an empty matrix beforehand to store the results;
  - function `log()` calculate the natural logarithm (ln);
- With the calculated ln price, calculate the log return
  - log return in finance as $r_t = ln(P_t) - ln(P_{t-1})$
  - return of the starting date is zero $r_0 = 0$.

Exercise b.1


```
x = randn(1000,1);
price = x(x>0); % price is elements of x that are positive

T = length(price); % number of loop times
ln_p = zeros(T,1); % creat an empty matrix to store result

% calculate log price
for t = 1: T;
    ln_p(t,1) = log(price(t,1));
end

% calculate log return
ln_r = zeros(T,1);
for t = 2: T; %1st day return = 0
    ln_r(t,1) = ln_p(t)-ln_p(t-1);
end
```

Exercise b.2

- Use a `while` loop to count the number of uniformly distributed realizations (use the `rand(.)`) between 0 and 1 that it takes to add up to 20 (or more).


```
1 %% Exercise b.2
2 my_sum = 0;
3 count = 0;
4 while (my_sum < 20)
5     temp = rand(1);
6     my_sum = my_sum +temp;
7     count = count + 1;
8 end
9 disp(count);
```

User-defined functions


```
1 function [ out1, out2] = my_func( input1, input2 )
2     % function body, do some stuff here
3 end
```


- A separate .m file starts with `function` to declare it's a user-defined function file .
- `[out1, out2]` are the declared outputs of the function
- `my_func` is the name of the function, so as the name of the .m file
- `input1, input2` are the required inputs the function
- The function body performs calculations and produces outputs designed by users.
- The function needs to be located in the current working dictionary to be called.

Exercises c

(c.1) Create a function called `myabs(.)` that returns the absolute value of a input.

(c.2) Create a function called `myfact(.)` that returns the factorial of an integer. Remember that $0! = 1$. Use function `prod(1:n)` to get $n!$.

(c.3) Create 2 functions called `stats_1(.)` and `stats_2(.)` that both output the maximum, minimum and average of a vector input.

- `stats_1(.)` outputs three values respectively;
- `stats_2(.)` outputs one vector that stores the three calculations.

- Note: Distinguish what is the function file and what is the main command.

Exercise c.1

Function

```
1  function [ res ] = myabs( x )
2  % this function computes the absolute value
3  % X: the input can be scalar, vector or matrix
4  % res: the output
5  res = abs(x)
6  end
```

Main Command

```
1  %% Exericse c.1
2  % here is the main command to call the function
3  % call function on input -10
4  myabs(-10)
5
6  % call function on input t and store output y
7  t = -0.8
8  y = myabs(t)
```

Exercise c.2

Function

```
1  function [ res ] = myfact( n )
2  % this function performs factorial calculation
3  % n: the input can be scalar
4  % res: the output
5  res = prod(1:n);
6
7  end
```

Main Command

```
1  % here is the main command to call the function
2  % call function on input 10
3  myfact(10)
4
5  % call function on input n and store output y
6  n = 77
7  y = myfact(n)
```

Exercises c.3

Function

```
1  function [max_val, min_val, avg_val ] = stats_1( vec )
2  max_val = max(vec);
3  min_val = min(vec);
4  avg_val = mean(vec);
5  end
```



```
1  function [ res ] = stats_2( vec )
2  res = [max(vec), min(vec), mean(vec)];
3  end
```


Main Command

```
1  % here is the main command to call the function
2  % stats_1: three outputs for max, min and avg
3  x = [1,2,3,4,5];
4  [a, b, c] = stats_1(x)
5
6  % stats_2: 1 output stores the max, min and avg
7  res = stats_2(x)
```

TakeAway

- Logical expression
- Flow control
- Loop
  - Note: if a calculation can be conducted at the matrix level, no need to use a loop (Next Class);
  - Note: `log(.)` can be perform on the entire matrix, much faster than using the loop.
- Function:
  - It is a separate .m file from your main code file.
  - It has to be shown in the current folder to be called.

Appendix: Switch structure

The `switch` structure executes certain statements based on the value of the `expression`, which takes only the values specified in difference case scenario.


```
switch expression % expression is scalar/string
    case value1
        statement1
    case value2
        statement2
        ...
    case value10
        statement10

    % executes if expression does not match any case
    otherwise
        default_statement
end
```

Appendix: Switch structure


```
1  mynumber = input('Enter a number:');
2
3  switch mynumber %switch among different values of mynumber
4      case -1 % mynumber == -1
5          disp('negative one');
6      case 0  % mynumber == 0
7          disp('zero');
8      case 1  % mynumber == 1
9          disp('positive one');
10     otherwise % mynumber is not -1 or 0 or 1
11         disp('other value');
12 end
```

Appendix: Exercise a.3

Perform the same tasks as in exercise a.2 with a `switch` structure.

- but set `x=0; res=0` if the user gives invalid control input


```
control = input('Please choose 1 for abs, 2 for squared, ...
    3 for square root: ');
x = randn(1);
switch control
    case 1
        res = abs(x);
    case 2
        res = x^2;
    case 3
        if x > 0
            res = sqrt(x)
        else
            error('Negative number has no squared root.')
        end
    otherwise
        error('Invalid control variable.')
        x = 0;
        res = 0;
end
```