

Boolean algebra, conditional statements, loops.

Eugeniy E. Mikhailov

The College of William & Mary



Lecture 03

Boolean algebra

Variable of boolean type can have only two values

- true
- false

Boolean algebra

Variable of boolean type can have only two values

- true (Matlab use `1` to indicate it, actually everything but zero)
- false

Boolean algebra

Variable of boolean type can have only two values

- true (Matlab use `1` to indicate it, actually everything but zero)
- false (Matlab uses `0`)

Boolean algebra

Variable of boolean type can have only two values

- true (Matlab use `1` to indicate it, actually everything but zero)
- false (Matlab uses `0`)

There are three logical operators which are used in boolean algebra

Boolean algebra

Variable of boolean type can have only two values

- true (Matlab use `1` to indicate it, actually everything but zero)
- false (Matlab uses `0`)

There are three logical operators which are used in boolean algebra

- \neg - logic **not**, Matlab `~`

\neg true = false

\neg false = true

Boolean algebra

Variable of boolean type can have only two values

- true (Matlab use `1` to indicate it, actually everything but zero)
- false (Matlab uses `0`)

There are three logical operators which are used in boolean algebra

- \neg - logic **not**, Matlab `~`

$$\neg \text{true} = \text{false}$$

$$\neg \text{false} = \text{true}$$

- \wedge - logic **and**, Matlab `&`

$$A \wedge B = \begin{cases} \text{true, if } A=\text{true and } B=\text{true,} \\ \text{false, otherwise} \end{cases}$$

Boolean algebra

Variable of boolean type can have only two values

- true (Matlab use `1` to indicate it, actually everything but zero)
- false (Matlab uses `0`)

There are three logical operators which are used in boolean algebra

- \neg - logic **not**, Matlab `~`

$$\neg \text{true} = \text{false}$$

$$\neg \text{false} = \text{true}$$

- \wedge - logic **and**, Matlab `&`

$$A \wedge B = \begin{cases} \text{true, if } A=\text{true and } B=\text{true,} \\ \text{false, otherwise} \end{cases}$$

- \vee - logic **or**, Matlab `|`

$$A \vee B = \begin{cases} \text{false, if } A=\text{false and } B=\text{false,} \\ \text{true, otherwise} \end{cases}$$

Boolean operators precedence in Matlab

If $A = \text{false}$, $B = \text{true}$, $C = \text{true}$

$$A | \sim B \& C$$

Boolean operators precedence in Matlab

If $A = \text{false}$, $B = \text{true}$, $C = \text{true}$

$$A|\sim B\&C$$

\sim has highest precedence, then $\&$, and then $|$

Boolean operators precedence in Matlab

If $A = \text{false}$, $B = \text{true}$, $C = \text{true}$

$$A | \sim B \& C$$

\sim has highest precedence, then $\&$, and then $|$

$$A | ((\sim B) \& C)$$

Boolean operators precedence in Matlab

If $A = \text{false}$, $B = \text{true}$, $C = \text{true}$

$$A|\sim B\&C$$

\sim has highest precedence, then $\&$, and then $|$

$$A|((\sim B)\&C)$$

Thus

$$A|\sim B\&C = \text{false}$$

Boolean operators precedence in Matlab

If $A = \text{false}$, $B = \text{true}$, $C = \text{true}$

$$A|\sim B\&C$$

\sim has highest precedence, then $\&$, and then $|$

$$A|((\sim B)\&C)$$

Thus

$$A|\sim B\&C = \text{false}$$

“Cat is an animal and cat is not an animal”

Boolean operators precedence in Matlab

If $A = \text{false}$, $B = \text{true}$, $C = \text{true}$

$$A | \sim B \& C$$

\sim has highest precedence, then $\&$, and then $|$

$$A | ((\sim B) \& C)$$

Thus

$$A | \sim B \& C = \text{false}$$

“Cat is an animal and cat is not an animal”
is false statement

Boolean operators precedence in Matlab

If $A = \text{false}$, $B = \text{true}$, $C = \text{true}$

$$A | \sim B \& C$$

\sim has highest precedence, then $\&$, and then $|$

$$A | ((\sim B) \& C)$$

Thus

$$A | \sim B \& C = \text{false}$$

“Cat is an animal and cat is not an animal”
is false statement

$$\sim Z \& Z =$$

Boolean operators precedence in Matlab

If $A = \text{false}$, $B = \text{true}$, $C = \text{true}$

$$A | \sim B \& C$$

\sim has highest precedence, then $\&$, and then $|$

$$A | ((\sim B) \& C)$$

Thus

$$A | \sim B \& C = \text{false}$$

“Cat is an animal and cat is not an animal”
is false statement

$$\sim Z \& Z = \text{false}$$

Boolean logic examples

There is an island, which is populated by two kind of people: liars and truthlovers.

- Liars always lie and never speak a word of truth.
- Truthlovers always speak only truth.

Suppose, you are landed on this island and met a person. What will be the answer to your question “Who are you?”

Boolean logic examples

There is an island, which is populated by two kind of people: liars and truthlovers.

- Liars always lie and never speak a word of truth.
- Truthlovers always speak only truth.

Suppose, you are landed on this island and met a person. What will be the answer to your question “Who are you?”

- The answer always will be “Truthlover”.

Boolean logic examples

There is an island, which is populated by two kind of people: liars and truthlovers.

- Liars always lie and never speak a word of truth.
- Truthlovers always speak only truth.

Suppose, you are landed on this island and met a person. What will be the answer to your question “Who are you?”

- The answer always will be “Truthlover”.

Now you see a person who answers to your question. “I am a liar.”
Is it possible?

Boolean logic examples

There is an island, which is populated by two kind of people: liars and truthlovers.

- Liars always lie and never speak a word of truth.
- Truthlovers always speak only truth.

Suppose, you are landed on this island and met a person. What will be the answer to your question “Who are you?”

- The answer always will be “Truthlover”.

Now you see a person who answers to your question. “I am a liar.”
Is it possible?

- This makes a paradox and should not ever happen on this island.

Matlab boolean logic examples

- `123.3 & 12=`

Matlab boolean logic examples

- `123.3 & 12 = 1`
- `~ 1232e-6 =`

Matlab boolean logic examples

- $123.3 \& 12 = 1$
- $\sim 1232e-6 = 0$

Matlab boolean logic examples

- $123.3 \ \& \ 12 = 1$
- $\sim 1232e-6 = 0$

```
>> B=[1.22312, 0; 34.343, 12]
```

```
B =
```

```
1.2231      0
```

```
34.3430    12.0000
```


Matlab boolean logic examples

- $123.3 \ \& \ 12 = 1$
- $\sim 1232e-6 = 0$

```
>> B=[1.22312, 0; 34.343, 12]
```

```
B =
```

```
1.2231      0  
34.3430    12.0000
```

```
~B
```

Matlab boolean logic examples

- $123.3 \ \& \ 12 = 1$
- $\sim 1232e-6 = 0$

```
>> B=[1.22312, 0; 34.343, 12]
```

```
B =  
1.2231      0  
34.3430    12.0000
```

```
~B
```

```
ans =
```

```
0      1  
0      0
```

Matlab boolean logic examples

- $123.3 \ \& \ 12 = 1$
- $\sim 1232e-6 = 0$

```
>> B=[1.22312, 0; 34.343, 12]
```

```
B =  
1.2231      0  
34.3430    12.0000
```

```
~B
```

```
ans =
```

```
0      1  
0      0
```

```
B | ~B
```

Matlab boolean logic examples

- $123.3 \ \& \ 12 = 1$
- $\sim 1232e-6 = 0$

```
>> B=[1.22312, 0; 34.343, 12]
```

```
B =  
1.2231    0  
34.3430   12.0000
```

```
~B
```

```
ans =
```

```
0    1  
0    0
```

```
B | ~B
```

“To be or not to be”
The answer is to be

```
ans =
```

```
1    1  
1    1
```

Matlab boolean logic examples

```
>> B=[1.22312, 0; 34.343, 12]
```

```
B =
```

```
1.2231    0  
34.3430   12.0000
```

```
>> A=[56, 655; 0, 24.4]
```

```
A =
```

```
56.0000   655.0000  
0         24.4000
```

Matlab boolean logic examples

```
>> B=[1.22312, 0; 34.343, 12]
```

```
B =
```

```
1.2231    0  
34.3430  12.0000
```

```
>> A=[56, 655; 0, 24.4]
```

```
A =
```

```
56.0000  655.0000  
0         24.4000
```

B&A

Matlab boolean logic examples

```
>> B=[1.22312, 0; 34.343, 12]
```

```
B =
```

```
1.2231    0
34.3430   12.0000
```

```
>> A=[56, 655; 0, 24.4]
```

```
A =
```

```
56.0000   655.0000
0         24.4000
```

```
B&A
```

```
ans =
```

```
1    0
0    1
```

Matlab boolean logic examples

```
>> B=[1.22312, 0; 34.343, 12]
```

```
B =
```

```
1.2231    0
34.3430   12.0000
```

```
>> A=[56, 655; 0, 24.4]
```

```
A =
```

```
56.0000   655.0000
0         24.4000
```

B&A

A | ~B

```
ans =
```

```
1    0
0    1
```


Matlab boolean logic examples

```
>> B=[1.22312, 0; 34.343, 12]
```

```
B =
```

```
1.2231    0
34.3430   12.0000
```

```
>> A=[56, 655; 0, 24.4]
```

```
A =
```

```
56.0000   655.0000
0          24.4000
```

B&A

```
ans =
```

```
1    0
0    1
```

A|~B

```
ans =
```

```
1    1
0    1
```

Comparison operators

Math	Matlab
$=$	<code>==</code> double equal sign!
\neq	<code>~=</code>
$<$	<code><</code>
\leq	<code><=</code>
$>$	<code>></code>
\geq	<code>>=</code>

Comparison operators

Math	Matlab
=	== double equal sign!
\neq	~=
<	<
\leq	<=
>	>
\geq	>=

```
x=[1, 2, 3, 4, 5]
```

```
x =
```

```
1     2     3     4     5
```

Comparison operators

Math	Matlab
=	== double equal sign!
\neq	~=
<	<
\leq	<=
>	>
\geq	>=

```
x=[1, 2, 3, 4, 5]
```

```
x =
```

```
1     2     3     4     5
```

```
x >= 3
```

Comparison operators

Math	Matlab
$=$	<code>==</code> double equal sign!
\neq	<code>~=</code>
$<$	<code><</code>
\leq	<code><=</code>
$>$	<code>></code>
\geq	<code>>=</code>

```
x=[1,2,3,4,5]
```

```
x =
```

```
1     2     3     4     5
```

```
x >= 3
```

```
ans =
```

```
0     0     1     1     1
```

Comparison operators

Math	Matlab
=	== double equal sign!
\neq	~=
<	<
\leq	<=
>	>
\geq	>=

```
x=[1,2,3,4,5]
```

```
x =  
    1     2     3     4     5
```

```
x >= 3
```

```
ans =  
    0     0     1     1     1
```

```
% chose such 'x' where x>=3  
x(x >= 3)
```

Comparison operators

Math	Matlab
=	== double equal sign!
\neq	~=
<	<
\leq	<=
>	>
\geq	>=

```
x=[1,2,3,4,5]
```

```
x =  
    1     2     3     4     5
```

```
x >= 3
```

```
ans =  
    0     0     1     1     1
```

```
% chose such 'x' where x>=3
```

```
x(x >= 3)
```

```
ans =  
     3     4     5
```

Comparison with matrices

```
>> A=[1,2;3,4]
```

```
A =
```

```
1     2
```

```
3     4
```

```
>> B=[33,11;53,42]
```

```
B =
```

```
33     11
```

```
53     42
```


Comparison with matrices

```
>> A=[1,2;3,4]
```

```
A =
```

```
1     2
```

```
3     4
```

```
>> B=[33,11;53,42]
```

```
B =
```

```
33     11
```

```
53     42
```

```
A>=2
```

Comparison with matrices

```
>> A=[1,2;3,4]
```

```
A =
```

```
1     2
3     4
```

```
>> B=[33,11;53,42]
```

```
B =
```

```
33     11
53     42
```

```
A>=2
```

```
ans =
```

```
0     1
1     1
```

Comparison with matrices

```
>> A=[1, 2; 3, 4]
```

```
A =
```

```
1     2
3     4
```

```
>> B=[33, 11; 53, 42]
```

```
B =
```

```
33     11
53     42
```

```
A>=2
```

```
A (A>=2)
```

```
ans =
```

```
0     1
1     1
```

Comparison with matrices

```
>> A=[1,2;3,4]
```

```
A =
```

```
1     2
3     4
```

```
>> B=[33,11;53,42]
```

```
B =
```

```
33     11
53     42
```

```
A>=2
```

```
ans =
```

```
0     1
1     1
```

```
A (A>=2)
```

```
ans =
```

```
3
2
4
```

Comparison with matrices

```
>> A=[1, 2; 3, 4]
```

```
A =
```

```
1     2
3     4
```

```
>> B=[33, 11; 53, 42]
```

```
B =
```

```
33     11
53     42
```

```
A>=2
```

```
ans =
```

```
0     1
1     1
```

```
A (A>=2)
```

```
ans =
```

```
3
2
4
```

```
B (A>=2)
```

Chose such
elements of B where
elements of $A \geq 2$

Comparison with matrices

```
>> A=[1, 2; 3, 4]
```

```
A =
```

```
1     2
3     4
```

```
>> B=[33, 11; 53, 42]
```

```
B =
```

```
33     11
53     42
```

```
A>=2
```

```
ans =
```

```
0     1
1     1
```

```
A (A>=2)
```

```
ans =
```

```
3
2
4
```

```
B (A>=2)
```

Chose such elements of B where elements of $A \geq 2$

```
ans =
```

```
53
11
42
```

if-else-end statement

```
if expression  
this part is executed  
only if expression is  
true  
else  
this part is executed  
only if expression is  
false  
end
```

if-else-end statement

`if` *expression*

this part is executed
only if *expression* is
true

`else`

this part is executed
only if *expression* is
false

`end`

`if` *hungry*

buy some food

`else`

keep working

`end`

if-else-end statement

if expression

this part is executed
only if *expression* is
true

else

this part is executed
only if *expression* is
false

end

if hungry

buy some food

else

keep working

end

```
if (x>=0)
    y=sqrt(x);
else
    error('cannot do');
end
```

Common mistake in the 'if' statement

```
if (x=y)
    D=4;
    Z=45;
    C=12;
else
    D=2;
end
```

Common mistake in the 'if' statement

```
if (x=y)
  D=4;
  Z=45;
  C=12;
else
  D=2;
end
```

the value of 'D' is always 4, except the case when $y=0$

Common mistake in the 'if' statement

```
if (x=y)
    D=4;
    Z=45;
    C=12;
else
    D=2;
end
```

the value of 'D' is always 4, except the case when $y=0$

someone used assignment operator (=) instead of comparison (==)

Short form of 'if-end' statement

`if` *expression*
this part is executed
only if *expression* is
true
`end`

Short form of 'if-end' statement

if expression

this part is executed
only if *expression* is
true
end

if won a million
go party
end

Short form of 'if-end' statement

`if` *expression*

this part is executed
only if *expression* is
true
`end`

`if` *won a million*
go party
`end`

```
if (deviation<=0)
    exit;
end
```

The 'while' statement

```
while expression  
this part is executed  
while expression is  
true  
end
```


The 'while' statement

`while` *expression*

this part is executed

`while` *expression* is

true

`end`

`while` *hungry*

keep eating

`end`

The 'while' statement

`while` *expression*

this part is executed

`while` *expression* is

true

`end`

`while` *hungry*

keep eating

`end`

```
i=1;  
while (i<=10)  
    c=a+b;  
    z=c*4+5;  
    i=i+2;  
end
```

The 'while' statement

`while` *expression*

this part is executed

`while` *expression* is

true

`end`

`while` *hungry*

keep eating

`end`

```
i=1;
while (i<=10)
    c=a+b;
    z=c*4+5;
    i=i+2;
end
```

`while` loop is extremely useful but they are not guaranteed to finish. For a bit more complicated conditional statement and loop it is impossible to predict if the loop will finish.

The 'while' statement

`while` *expression*

this part is executed

`while` *expression* is

true

`end`

`while` *hungry*

keep eating

`end`

```
i=1;
while (i<=10)
    c=a+b;
    z=c*4+5;
    i=i+2;
end
```

`while` loop is extremely useful but they are not guaranteed to finish. For a bit more complicated conditional statement and loop it is impossible to predict if the loop will finish.

Yet another common mistake is

```
i=1;
while (i<=10)
    c=a+b;
end
```

The 'while' statement

`while` *expression*

this part is executed `while` *hungry*

`while` *expression* is keep eating

true `end`

`end`

```
i=1;
while (i<=10)
    c=a+b;
    z=c*4+5;
    i=i+2;
end
```

`while` loop is extremely useful but they are not guaranteed to finish. For a bit more complicated conditional statement and loop it is impossible to predict if the loop will finish.

Yet another common mistake is

```
i=1;
while (i<=10)
    c=a+b;
end
```

not updating the term leading to fulfillment of the `while` condition

The 'for' statement

```
for variable = expression  
do something  
end
```

In this case variable is assigned
consequently with columns of the
expression, and then statements inside of
the loop are executed

The 'for' statement

```
for variable = expression  
do something  
end
```

In this case variable is assigned
consequently with columns of the
expression, and then statements inside of
the loop are executed

```
sum=0;  
x=[1, 3, 5, 6]  
for v=x  
    sum=sum+v;  
end
```

```
>> sum  
sum =  
    15
```

The 'for' statement

```
for variable = expression  
do something  
end
```

In this case variable is assigned
consequently with columns of the
expression, and then statements inside of
the loop are executed

```
sum=0;  
x=[1, 3, 5, 6]  
for v=x  
    sum=sum+v;  
end
```

```
>> sum  
sum =  
    15
```

`for` loops are guaranteed to complete after predictable number of iterations (the amount of columns in *expression*).

Example

$$S = \sum_{i=1}^{100} i = 1 + 2 + 3 + 4 + \cdots + 99 + 100$$

Example

$$S = \sum_{i=1}^{100} i = 1 + 2 + 3 + 4 + \dots + 99 + 100$$

```
S=0; i=1;
while (i<=100)
    S=S+i;
    i=i+1;
end
```

Example

$$S = \sum_{i=1}^{100} i = 1 + 2 + 3 + 4 + \dots + 99 + 100$$

```
S=0; i=1;  
while (i<=100)  
    S=S+i;  
    i=i+1;  
end
```

```
S=0;  
for i=1:100  
    S=S+i;  
end
```

Example

$$S = \sum_{k=1} a_k$$

While $k \leq 100$ and $a_k \geq 10^{-5}$, where $a_k = k^{-k}$.

Example

$$S = \sum_{k=1} a_k$$

While $k \leq 100$ and $a_k \geq 10^{-5}$, where $a_k = k^{-k}$.

```
S=0; k=1;
while( (k<=100) & (k^-k >= 1e-5) )
    S=S+k^-k;
    k=k+1;
end
```

Example

$$S = \sum_{k=1} a_k$$

While $k \leq 100$ and $a_k \geq 10^{-5}$, where $a_k = k^{-k}$.

```
S=0; k=1;
while( (k<=100) & (k^-k >= 1e-5) )
    S=S+k^-k;
    k=k+1;
end
```

```
>> S
S =
    1.2913
```

Example

$$S = \sum_{k=1} a_k$$

While $k \leq 100$ and $a_k \geq 10^{-5}$, where $a_k = k^{-k}$.

```
S=0; k=1;
while( (k<=100) & (k^-k >= 1e-5) )
    S=S+k^-k;
    k=k+1;
end
```

```
>> S
S =
    1.2913
```

```
S=0; k=1;
while( k<=100 )
    a_k=k^-k;
    if (a_k < 1e-5)
        break;
    end
    S=S+a_k;
    k=k+1;
end
```

Example

$$S = \sum_{k=1} a_k$$

While $k \leq 100$ and $a_k \geq 10^{-5}$, where $a_k = k^{-k}$.

```
S=0; k=1;
while( (k<=100) & (k^-k >= 1e-5) )
    S=S+k^-k;
    k=k+1;
end
```

```
>> S
S =
    1.2913
```

```
S=0; k=1;
while( k<=100 )
    a_k=k^-k;
    if (a_k < 1e-5)
        break;
    end
    S=S+a_k;
    k=k+1;
end
```

```
>> S
S =
    1.2913
```


Same example with 'for' loop and use of matrix ops

$$S = \sum_{k=1} a_k$$

While $k \leq 100$ and $a_k \geq 10^{-5}$, where $a_k = k^{-k}$.

Same example with 'for' loop and use of matrix ops

$$S = \sum_{k=1} a_k$$

While $k \leq 100$ and $a_k \geq 10^{-5}$, where $a_k = k^{-k}$.

```
S=0;
for k=1:100
    a_k=k^-k;
    if (a_k < 1e-5)
        break;
    end
    S=S+a_k;
end
```

Same example with 'for' loop and use of matrix ops

$$S = \sum_{k=1} a_k$$

While $k \leq 100$ and $a_k \geq 10^{-5}$, where $a_k = k^{-k}$.

```
S=0;
for k=1:100
    a_k=k^-k;
    if (a_k < 1e-5)
        break;
    end
    S=S+a_k;
end
```

```
>> S
S =
    1.2913
```

Often it is more elegant to use built in Matlab matrix operators

```
>> k=1:100;
>> a_k=k.^-k;
>> S=sum(a_k(a_k>=1e-5))
S =
    1.2913
```

Note

- use of the *choose elements* construct
- built in `sum` function

Interest rate related example

Suppose bank gave you 50% interest rate (let's call it 'x'), and you put one dollar in.

How much would you get at the end of the year?

- one payment at the end of the year

$$M_1 = 1 * (1 + x) = 1 * (1 + .5) = 1.5$$

Interest rate related example

Suppose bank gave you 50% interest rate (let's call it 'x'), and you put one dollar in.

How much would you get at the end of the year?

- one payment at the end of the year

$$M_1 = 1 * (1 + x) = 1 * (1 + .5) = 1.5$$

- interest payment every half a year

$$M_2 = 1 * (1 + x/2) * (1 + x/2) = 1 * (1 + .5/2)^2 = 1.5625$$

Interest rate related example

Suppose bank gave you 50% interest rate (let's call it 'x'), and you put one dollar in.

How much would you get at the end of the year?

- one payment at the end of the year

$$M_1 = 1 * (1 + x) = 1 * (1 + .5) = 1.5$$

- interest payment every half a year

$$M_2 = 1 * (1 + x/2) * (1 + x/2) = 1 * (1 + .5/2)^2 = 1.5625$$

- interest payment every month

$$M_{12} = 1 * (1 + x/12)^{12} = 1.6321$$

Interest rate related example

Now let's find how your return on investment (M_N) depends on the number of payments per year

```
x=.5; N_max=100; N=1:N_max;
M=0*(N); % since N is vector M will be a vector too
for i=N
    M(i)=(1+x/i)^i;
end
plot(N,M,'-'); set(gca,'FontSize',24);
xlabel('N, number of payments per year');
ylabel('M_n, return on investment'); % note M_n use
title('Return on investment vs number of payments');
```

Of course we do not need computer to show that $M_\infty = e^x = 1.6487$ but we need it to calculate something like

$$M_{1001} - M_{1000} = 2.0572 \times 10^{-7}$$

Interest rate related example

Now let's find how your return on investment (M_N) depends on the number of payments per year

```
x=.5; N_max=100; N=1:N_max;
M=0*(N); % since N is vector M will be a vector too
for i=N
    M(i)=(1+x/i)^i;
end
plot(N,M,'-'); set(gca,'FontSize',24);
xlabel('N, number of payments per year');
ylabel('M_n, return on investment'); % note M_n use
title('Return on investment vs number of payments');
```

Of course we do not need computer to show that $M_\infty = e^x = 1.6487$ but we need it to calculate something like

$$M_{1001} - M_{1000} = 2.0572 \times 10^{-7}$$

Bonus question: can you calculate M without use of loops?

Interest rate related example

