Relational operators and Logical Flow
UC Berkeley
Fall 2004, E77

http://jagger.me.berkeley.edu/~pack/e77

Copyright 2005, Andy Packard. This work is licensed under the Creative Commons Attribution-ShareAlike License. To view a copy of this license, visit http://creativecommons.org/licenses/by-sa/2.0/ or send a letter to Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.

Relational Operators (pg 151-154)

Relational operators are used to compare variables.

There are 6 comparisons

- "equal to", using **==**
- "not equal to", using **~=**
- "less than", using **<**
- "less than or equal to", using **<=**
- "greater than", using **>**
- "greater than or equal to", using **>=**

The result of a comparison is either TRUE (1) or FALSE (0)

Array comparisons

Suppose **`A`** and **`B`** are double arrays of the same size. Let **`op`** be any of the 6 relational operators (**`==`**, **`~=`**, **`<`**, **`<=`**, **`>`**, **`>=`**)

Then the expression

```
A op B
```

is a *logical* array of the same size. The relational operator is applied elementwise, comparing **`A(i,j)`** to **`B(i,j)`**.

Example

```
>> A = rand(2,4);
>> B = 0.5*ones(2,4);
>> A<B
```

LOGICAL arrays

The result of a relational operation is a *logical* array

- A logical array contains only 0’s and 1’s.
- It cannot contain any other numerical values
- Internal representation in MATLAB is different than for double arrays.

You can use a *logical* array in any numerical calculation as though it is a *double* array –the 0’s and 1’s behave normally.

```
>> A = [1 0 1 1];
>> B = logical(A);
>> whos
>> A==B
>> isequal(A,B)
```

Indexing with LOGICAL arrays

In a typical row/column reference,

`M(RowIndex,ColIndex)`

both **`RowIndex`** and **`ColIndex`** are *double* arrays, whose positive, integer values specify which rows and columns of the array **`M`** are being referenced.

If **`RowIndex`** and **`ColIndex`** are *logical* arrays, the locations of the 1's specify which rows and columns of the array **`M`** are being referenced

```
>> M = rand(4,5);
>> Ridx = logical([1 0 0 1]);
>> Cidx = logical([0 0 1 1 1]);
>> M(Ridx,Cidx)   %same as M([2 4],[3 4 5])
```

Scalar/Array comparisons

Suppose **`A`** is a scalar, and **`B`** is a double array. Let **`op`** be any of the 6 relational operators (**`==`**, **`~=`**, **`<`**, **`<=`**, **`>`**, **`>=`**)

Then the expression

```
A op B
```

is an array of the same size as **`B`**. The relational operator is applied comparing the scalar **`A`** to each element of **`B`**.

Example

```
>> A = 2.5;
>> B = [0 3 4;-1 -2 1;6 2.5 2.4];
>> C = A<=B;
```

Array/Scalar comparisons

Suppose **`A`** is a double array, and **`B`** is a scalar. Let **`op`** be any of the 6 relational operators (**`==`**, **`~=`**, **`<`**, **`<=`**, **`>`**, **`>=`**)

Then the expression

```
A op B
```

is an array of the same size as **`A`**. The relational operator is applied comparing each element of **`A`** to the scalar **`B`**.

Example

```
>> A = sin(linspace(0,pi,20));
>> B = 0.5
>> C = A>B;
```

find

The command **find** returns the indices of the nonzero entries.

```
>> m = rand(6,1);
>> m(find(m<0.5)) = 0;
```

logical

But logical indexing also work, so you can just do

```
>> m = rand(6,1);
>> m(m<0.5) = 0;
```

For arrays, find returns the indices in a single-index form, using the well-defined ordering for the elements in an array.

```
>> m = rand(4,5);
>> idx = m<0.5;
>> m(idx) = -m(idx);
```

Care in using == on numeric data

In finite precision arithmetic (MATLAB has about 17 digits of precision), it is not true that

(a+b)+c is equal to *a+(b+c)*

What happens

- in computing *a+b*, some roundoff error may occur, and then in computing the additional sum with *c*, additional roundoff occurs.
- in computing *b+c*, some different roundoff error may occur, and then in computing the additional sum with *a*, additional roundoff occurs.

Imagine 2-digit arithmetic

$$\underbrace{1.2 + .74}_{1.9} + .24 \;\rightarrow\; \underbrace{1.9 + .24}_{2.1}$$

$$1.2 + \underbrace{.74 + .24}_{.98} \;\rightarrow\; \underbrace{1.2 + .98}_{2.2}$$

Logical Operators (pg 155-156)

Logical operators are used to combine variables.

There are 3 binary operations

- "logical AND", using **`&`**
- "logical OR", using **`|`**
- "logical exclusive OR", using **`xor`**

Along with unary negation

- "logical NOT", using ~

For arrays, the operators are applied elementwise, and the results have logical values of TRUE (1) or FALSE (0)

Logical Operators

If **A** and **B** are scalars (*double* or *logical*), then

- **A&B** is TRUE (1) if **A** and **B** are both nonzero, otherwise it is FALSE (0)
- **A|B** is TRUE (1) if either **A** or **B** are nonzero, otherwise it is FALSE (0)
- **xor(A,B)** is TRUE (1) if one argument is 0 and the other is nonzero, otherwise it is FALSE (0)
- **~A** is TRUE if **A** is 0, and FALSE if **A** is nonzero.

For arrays, the operations are applied elementwise, so **A** and **B** must be the same size, or one must be a scalar.

`if`, `end` (page 168-171)

To conditionally control the execution of statements, you can use

`expression` should be a numeric or logical array.

```
if expression
    statements
end
```

From now on, refer to this as:

"**`expression`** is TRUE"

If the ~~real part of all of the entries of **`expression`** are nonzero~~, then the statements between the **`if`** and **`end`** will be executed. Otherwise they will not be.

Execution continues with any statements after the **`end`**.

`if, else, end`

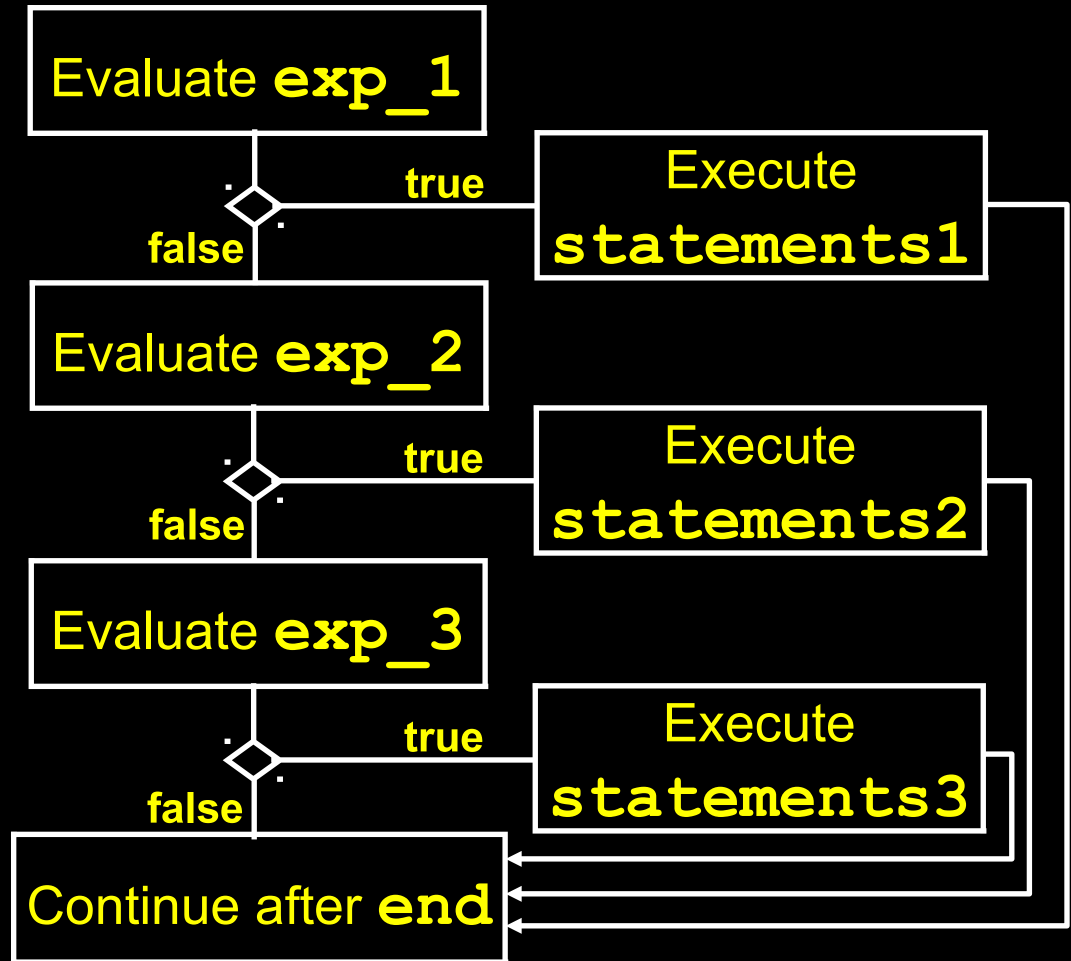
```
if exp_1
    statements1
else
    statements2
end
```

One of the sets of statements will be executed

- If **`exp_1`** is TRUE, then **`statements1`** are executed
- If **`exp_1`** is FALSE, then **`statements2`** are executed

if, elseif, end

```
if exp_1
    statements1
elseif exp_2
    statements2
elseif exp_3
    statements3
end
```



Could also have an **else** before the **end**

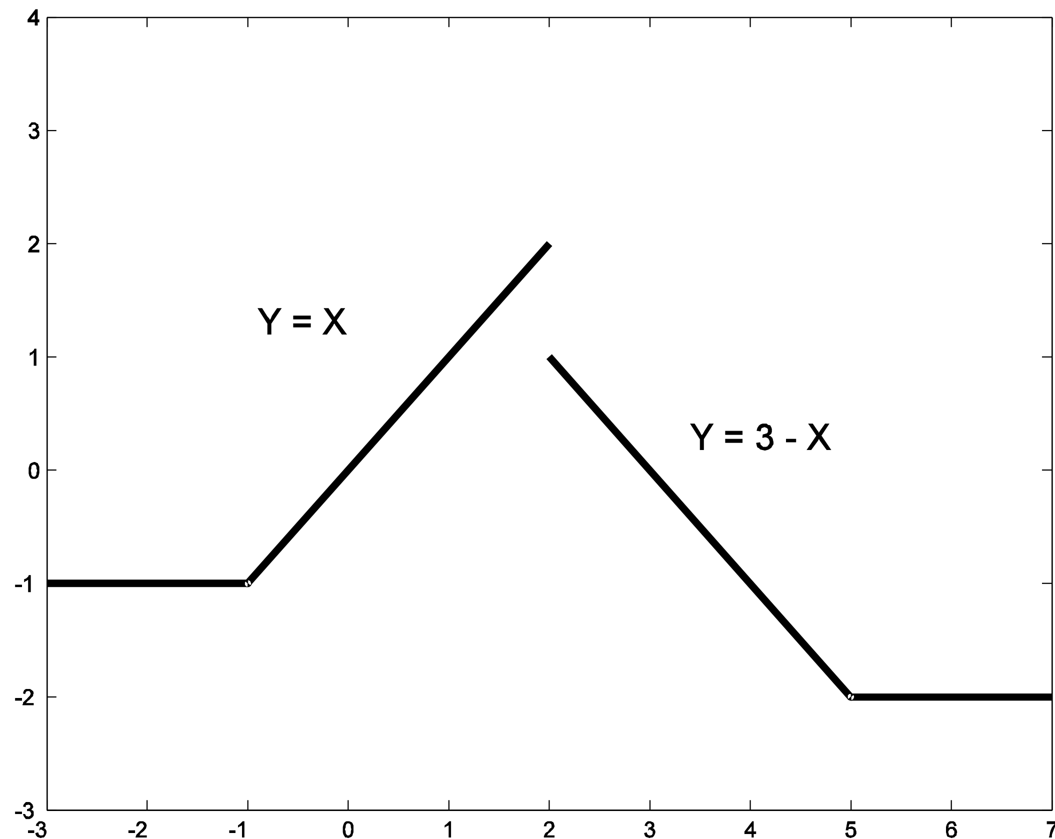
Illegal stuff

```
if exp_1
    statements1
elseif exp_2
    statements2
else
    statements3
elseif exp_4
    statements4
end
```

```
if exp_1
    statements1
else
    statements2
else
    statements3
end
```

Piecewise linear function

TASK: Create a m-file function for the mathematical function Y = F(X) shown below.



Simple example with IF/ELSEIF

```
function y = plinear(x)
if isscalar(x) & isa(x,'double') & isreal(x)
   if x<-1
      y = -1;
   elseif x<2
      y = x;
   elseif x<5
      y = 3-x;
   else
      y = -2;
   end
else
   error('x should be a real scalar');
end
```