



# Logical Vectors



## Week 5

Loosely follows Chapters 5



# Logical Vectors

---

When comparing vectors, a logical vector results

Essentially an element-wise comparison operation

```
a = 1:5;
```

```
b = [0 2 3 5 6];
```

```
a == b
```

```
ans =
```

```
 [0 1 1 0 0]
```

# Avoid Division By Zero

---

- With limits, can encounter zeros
- We want to “simulate” a zero
- MATLAB will give NaN

# Avoid Division By Zero Example

---

```
x = -4*pi : pi / 20 : 4*pi;
```

```
y = sin(x) ./ x;
```

Where  $x=0$  you'll receive NaN in y

```
logicalVector = x == 0;
```

The fix:

```
x = x + logicalVector*eps;
```

OR

```
x(logicalVector)=eps;
```

# Avoid Infinity

---

- With limits, encounter near infinity
- We want to simulate “infinity”
- This can throw plots off drastically

# Avoid Infinity Example

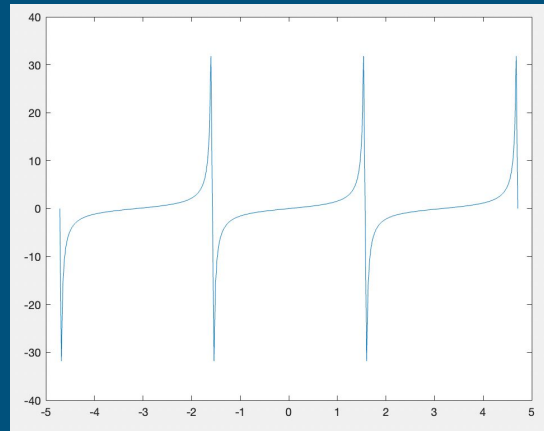
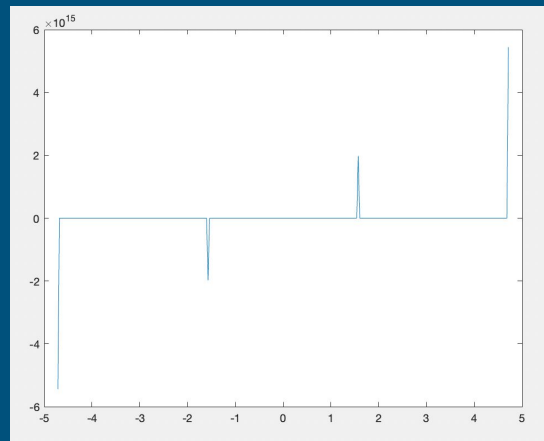
```
x = -3/2*pi : pi/100 : 3/2*pi;
```

```
y = tan(x);
```

```
plot(x, y)
```

The fix:

```
y = y .* (abs(y) < 1e10);
```



# Counting and Comparing Matches Example

```
r = rand(1,7)
```

```
r =
```

```
0.8147 0.9058 0.1270 0.9134 0.6324 0.0975 0.2785
```

```
sum(r < 0.5)
```

```
ans =
```

```
3
```

Behind the scenes:

```
r < 0.5
```

```
ans =
```

1×7 logical array

```
0 0 1 0 0 1 1
```

# Counting and Comparing Matches

---

`all()`

`any()`

`sum()`



# Relational Operations

---

- Perform element-wise comparisons between two arrays
- They return a logical array of the same size
  - Element values will be 1 or 0 for true or false respectively

# Relational Operators for Logical Operations

---

- < Less than
- > Greater than
- == Equal to (logical equality)
- ~ Not
- <= Less than or equal to ( $\leq$ )
- >= Greater than or equal to ( $\geq$ )
- ~= Not equal to ( $\neq$ )
- xor() Exclusive OR

# Logical Data Type

- The logical data type is another MATLAB data type.
- It is either equal to 1 (true) or 0 (false)

<u>Input</u>	<u>Output (ans =)</u>	<u>Data Type</u>
4 < 5	1	Logical Array
4 <= 5	1	Logical Array
4 > 5	0	Logical Array
4 >= 5	0	Logical Array
4 ~= 5	1	Logical Array

# Round-off Error

---

- The == and ~= commands can produce confusing results due to round-off error.
- Example

```
a = 0;  
b = sin(pi);  
a == b
```

```
ans =  
0
```

# Round-off Error

---

What happened?

- MATLAB computes `sin(pi)` to within `eps`. (A number approximately zero)
- The `==` operator properly concludes that 0 and a number which is approximately zero are different.
- The best way to fix this is to show things to be approximately equal.

```
abs(a-b) < 1.0e-14
```

```
ans =
```

```
1
```

# Logical Operators

---

- Logical, or Boolean, operators produce a logical result

<u>Operator</u>	<u>Operation</u>
&	Logical AND
	Logical OR
~	Logical NOT

# Logical Operators

---

- Example
  - `a = true;`
  - `b = false;`
  - `c = true;`
- What are the results of the following
  - `a & b`
  - `b & c`
  - `a & c`
  - `a | b`
  - `b | c`
  - `a | c`
  - `c | a`
  - `b | b`

# Hierarchy of Operations

---

- Logical operators are evaluated **after** arithmetic and relational operations.

## Order of Operations

1. Arithmetic
2. Relational Operators
3. All  $\sim$  (NOT) operators
4. All  $\&$  (AND) operators
5. All  $|$  (OR) operators



# Example

---

You need the **full** condition for each condition you wish to meet.

`60 <= finalExam < 70`

`(finalExam >= 60) & (finalExam < 70)`

-----

`(a ~= 0) | (b ~= 0) | (c ~= 0)`

`~((a == 0) & (b == 0) & (c == 0))`

# Logical Functions

---

- Functions in MATLAB that implement logical operations
- For example
  - The `find()` function returns the index numbers of the matrix that meet the logical statement given.

```
v = [1, 3, 5, 6, 3, 4; 6, 36, 6, 7, 843, 5];
```

```
[row, col] = find(v == 3)
```

```
row =
```

```
1
```

```
1
```

```
col =
```

```
2
```

```
5
```

# find()

---

We can use functions to manipulate data more efficiently.

Get a subscript of all non-zero values in a

`find(a)`

Make a vector containing only non-zero values in a

`a = a(find(a))`

# Logical Functions

---

- Other examples

- `ischar()` returns 1 if ( ) contains character data
- `isinf()` returns 1 if ( ) contains infinity (inf)
- `isnan()` returns 1 if ( ) contains not a number (nan)
- `isnumeric()` returns 1 if ( ) contains numeric data
- `isempty()` returns 1 if ( ) is empty ( `x = []` )
- `any()` returns 1 if any element in ( ) is non-zero
- `all()` returns 1 if all elements in ( ) are non-zero
- `exist()` returns 1 if ( ) exists as a workspace variable

# any() and all()

---

Can be useful in if statements

```
if all(a >= 1)
```

```
    % do something
```

```
end
```



# Matrices



## Week 5

Loosely follows Chapters 6



# Subscripts

---

- Elements within a matrix are referenced with two subscripts.
- `matrix(3)` Returns third row, first column (Different for vectors)
- `matrix(3,1)` Returns third row, first column
- `matrix(1:3, 4:6)` First through third row of fourth through sixth column
- `matrix(:, 4:6)` All rows in fourth through sixth column

# Subscripts

---

- Elements referenced in subscripts can also be written to using the assignment operator.
- `matrix(1:2, 2:3) = ones(2)` sets values to ones
- The keyword `end` can be used to reference the last element in a row or column.
- `matrix(:, 2:end)` Retrieves all elements but the first column.



# Duplicating Rows and Columns

---

- Sometimes referred to as tiling
- `repmat()` allows you to repeat a matrix

```
vectorA = [ 1, 2, 3 ];
```

```
repmat(vectorA, 3, 1)
```

```
% Or repmat(vectorA, [3, 1])
```

```
ans =
```

```
1    2    3
```

```
1    2    3
```

```
1    2    3
```

# Extracting Rows and Columns

---

- Logical arrays can be used to extract rows or columns
- `a(:, logical([1 0 1]))`      Extracts the first and third columns
- Positional values in an array can also be used
- `a(:, [1 3])`      Performs the same as above

# Elementary Matrices

---

- There are a set of functions for generating elementary matrices
  - `ones()`
  - `zeros()`
  - `rand()`
  - `eye()`
- See more with the `elmat` command

# Special Matrices

---

- There are a set of functions for generating special matrices
  - `pascal()`
  - `magic()`
  - `gallery()`
  - `hankel()`
- For more use the `help elmat` command

# Matrices and Functions

---

- Most functions perform actions by column
  - `all()` returns 1 for each column where all elements are non-zero
  - `any()` returns 1 for each column where any elements are non-zero
  - `sum()` returns a vector with the sum of each column of elements
  - `mean()` returns a vector with the mean of each column of elements
- To perform a function on a matrix and obtain a scalar functions must be nested or performed in steps.

```
colSum = sum(matrix)
totalSum = sum(colSum)
```

```
totalSum = sum(sum(matrix))
```

# Manipulating Matrices

---

- There are many functions for easily transforming matrices.
  - `diag()`
  - `flipud()`
  - `fliplr()`
  - `rot90()`
  - `tril()`
- For more see help
- Think of ways you can achieve different transformations by calling multiple functions.
- `triu() == rot90(rot90(tril(rot90(rot90(matrix))))))`

# Matrix Multiplication

- Remember `matrixA * matrixB` is different from `matrixA .* matrixB`
- Matrix Multiplication steps
  - Multiply each column vector by each row vector element-wise and sum the results

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \times \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} = 1 \times 4 + 2 \times 5 + 3 \times 6 = 22$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \times \begin{bmatrix} 1 & 3 & 5 & 7 \\ 2 & 4 & 6 & 8 \end{bmatrix} = \begin{bmatrix} 1*1+2*2 & 1*3+2*4 & 1*5+2*6 & 1*7+2*8 \\ 3*1+4*2 & 3*3+4*4 & 3*5+4*6 & 3*7+4*8 \\ 5*1+6*2 & 5*3+6*4 & 5*5+6*6 & 5*7+6*8 \end{bmatrix}$$
$$= \begin{bmatrix} 1+4 & 3+8 & 5+12 & 7+16 \\ 3+8 & 9+16 & 15+24 & 21+32 \\ 5+12 & 15+24 & 25+36 & 35+48 \end{bmatrix} = \begin{bmatrix} 5 & 11 & 17 & 23 \\ 11 & 25 & 39 & 53 \\ 17 & 39 & 61 & 83 \end{bmatrix}$$

# Matrix Multiplication Rules

---

- Matrix multiplication is not commutative ( $A*B \neq B*A$ )
- Columns of matrix A must match rows of matrix B
  - Resulting matrix is rowsXcols (the remaining values)
- To multiply a matrix by a vector in that order, the vector must be a column vector
  - Remember, cols of matrix A must match rows of matrix B



# Matrix Exponentiation

---

- Matrix operation  $A^2 == A * A$
- The matrix must be a square matrix
  - Think about why

# Key Takeaways

---

- Keywords/characters
  - :
  - end
- Matrix Functions
  - repmat()
  - ones()
  - zeros()
  - rand()
- Matrix Multiplication
  - Rules
  - Process

# Key Takeaways

---

- Logical Data Types
- Relational Operators
  - < Less than
  - > Greater than
  - == Logically equal to
- Logical Operators
  - & And
  - | Or
  - ~ Not
- Logical functions
  - ischar(x)
  - isinf(x)
  - ...