CSCI 130
Introduction to Engineering Computing
Class Meeting #26

Engineering Computing and Problem Solving with Matlab

Logical functions – ***find, any*** and ***all***
Vector calculations
Anonymous functions

Logical Functions – *find*

A useful technique to locate and select items from an array according to a given criterion

Example:

Create an example matrix of random numbers

```
>> X = randn(10,3);
>> X

X =

    0.5377   -1.3499    0.6715
    1.8339    3.0349   -1.2075
   -2.2588    0.7254    0.7172
    0.8622   -0.0631    1.6302
    0.3188    0.7147    0.4889
   -1.3077   -0.2050    1.0347
   -0.4336   -0.1241    0.7269
    0.3426    1.4897   -0.3034
    3.5784    1.4090    0.2939
    2.7694    1.4172   -0.7873
```

Logical Functions – *find*

Use *find* to determine where the negative numbers are in the X matrix

```
>> find(X<0)

ans =

     3
     6
     7
    11
    14
    16
    17
    22
    28
    30
```

```
>> X = randn(10,3);
>> X

X =

    0.5377   -1.3499    0.6715
    1.8339    3.0349   -1.2075
   -2.2588    0.7254    0.7172
    0.8622   -0.0631    1.6302
    0.3188    0.7147    0.4889
   -1.3077   -0.2050    1.0347
   -0.4336   -0.1241    0.7269
    0.3426    1.4897   -0.3034
    3.5784    1.4090    0.2939
    2.7694    1.4172   -0.7873
```

ans **contains locations counting down first column, then down second column, etc. How to make these into row-column indices?**

Logical Functions – *find*

Determine row/column indices

```
>> col = ceil(locations/N);
>> row = locations - N*(col-1);
>> [ row col ]

ans =

     3     1
     6     1
     7     1
     1     2
     4     2
     6     2
     7     2
     2     3
     8     3
    10     3
```

When the find is applied to a one-dimensional array, the locations are directly the subscripts of the array elements

Use these to extract X values

```
>> k = length(row);
>> for i = 1:k
      selectX(i) = X(row(i),col(i));
   end
>> [ row col selectX' ]

ans =

    3.0000    1.0000   -2.2588
    6.0000    1.0000   -1.3077
    7.0000    1.0000   -0.4336
    1.0000    2.0000   -1.3499
    4.0000    2.0000   -0.0631
    6.0000    2.0000   -0.2050
    7.0000    2.0000   -0.1241
    2.0000    3.0000   -1.2075
    8.0000    3.0000   -0.3034
   10.0000    3.0000   -0.7873
```

Logical Functions – *find*

A more complicated selection criterion:

```
>> find(X<-0.5 & X>-1 | X>0.5 & X<1)

ans =

     1
     4
    13
    15
    21
    23
    27
    30
```

find all the values in X that are either between -0.5 and -1 or between 0.5 and 1

Note the use of the logical *and* (&) and *or* (|) operators

Logical Functions – *all* and *any*

Results are true(1)/false(0)

```
>> all(X > -3 & X < 3)

ans =

     0     0     1
```

Are all the elements between -3 and 3?
No, in columns 1 and 2
Yes, in column 3

```
>> X = randn(10,3);
>> X

X =

    0.5377   -1.3499    0.6715
    1.8339    3.0349   -1.2075
   -2.2588    0.7254    0.7172
    0.8622   -0.0631    1.6302
    0.3188    0.7147    0.4889
   -1.3077   -0.2050    1.0347
   -0.4336   -0.1241    0.7269
    0.3426    1.4897   -0.3034
    3.5784    1.4090    0.2939
    2.7694    1.4172   -0.7873
```

```
>> any(X>3)

ans =

     1     1     0
```

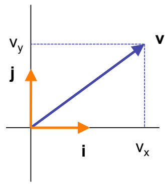
Are there any elements > 3 ?

Yes, in columns 1 and 2
No, in column 3

Vector Calculations

Two-dimensional vectors

[vector concepts, math and calculations arise frequently in physics and engineering]



i and j are unit vectors in the x and y direction

v is represented by

$$\mathbf{v} = v_x\mathbf{i} + v_y\mathbf{j}$$

the length or magnitude of v is

$$|\mathbf{v}| = \sqrt{v_x^2 + v_y^2}$$

and the angle v makes with the horizontal (x) axis is

$$\angle\mathbf{v} = \tan^{-1}\left(\frac{v_y}{v_x}\right)$$

In Matlab, a two-dimensional vector is represented by

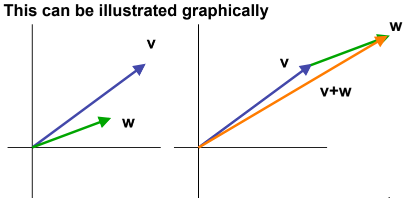
```
>> [ vx   vy ]
```

The same approach is used with three-dimensional vectors with three unit vectors, i, j and k

Vector Calculations

Vector addition

This can be illustrated graphically



and mathematically $\mathbf{v} + \mathbf{w} = (v_x + w_x)\mathbf{i} + (v_y + w_y)\mathbf{j}$

In Matlab

```
>> v = [ 1.9 1.2 ];
>> w = [ 1.3 0.4 ];
>> v+w

ans =

    3.2000    1.6000
```

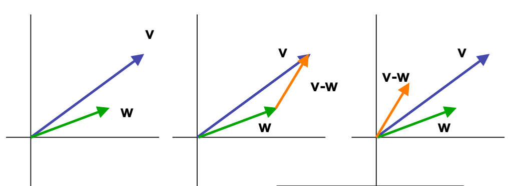
Vector Calculations

Vector subtraction

u = v - w   or   v = w + u

What vector would have to be added to w to yield v?

This can be illustrated graphically



and, in Matlab, by simple subtraction of vectors

```
>> u = v - w

u =

    0.6000    0.8000
```

Vector Calculations

Vector dot product

The dot product, a scalar quantity, is defined by

$$\mathbf{v} = v_x\mathbf{i} + v_y\mathbf{j} \qquad \mathbf{w} = w_x\mathbf{i} + w_y\mathbf{j}$$

$$\mathbf{v}\cdot\mathbf{w} = v_x \cdot w_x + v_y \cdot w_y$$

In Matlab, the dot product can be computed using the ***dot*** function or the inner product

```
>> dot(v,w)

ans =

    2.9500
```

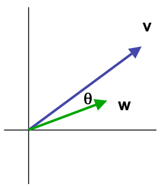
```
>> v*w'

ans =

    2.9500
```

Vector Calculations

Vector dot product



An alternate definition is

$$\mathbf{v}\cdot\mathbf{w} = |\mathbf{v}|\cdot|\mathbf{w}|\cdot\cos(\theta)$$

and the graphical interpretation is to project the w vector onto the v vector and multiply the projected magnitude times the magnitude of v (or vice versa).

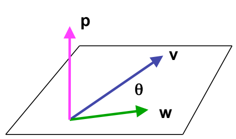
Vector Calculations

Vector cross product

If vectors v and w are defined in three-dimensional space and the plane common to them is illustrated as below, the cross product between v and w is defined as

$$\mathbf{p} = \mathbf{w}\times\mathbf{v}$$

where $|\mathbf{p}| = |\mathbf{w}|\cdot|\mathbf{v}|\cdot\sin(\theta)$



and the angle of p is orthogonal to the plane common to v and w and its direction is given by the ***right-hand rule***

<u>right-hand rule</u>: the direction of p is such that an observer at its tip will observe as counterclockwise the rotation through θ which brings the vector w in line with the vector v

Vector Calculations

Vector cross product

For $\mathbf{w} = w_x\mathbf{i} + w_y\mathbf{j} + w_z\mathbf{k}$ and $\mathbf{v} = v_x\mathbf{i} + v_y\mathbf{j} + v_z\mathbf{k}$

$$\mathbf{p} = \mathbf{w} \times \mathbf{v} = \left(w_y v_z - w_z v_y\right)\mathbf{i} + \left(w_z v_x - w_x v_z\right)\mathbf{j} + \left(w_x v_y - w_y v_x\right)\mathbf{k}$$

In Matlab, the *cross* function performs this calculation

```
>> w = [ w 0 ];
>> v = [ v 0 ];
>> cross(w,v)

ans =

         0         0    0.8000
```

Notice here that the w and v vectors are expanded from two to three dimensions.

Anonymous Functions

The need for the so-called *anonymous function* occurs in Matlab when it is necessary to pass additional arguments through a function to another function. Here is an example to make this clear.

Given a value of the parameter *Re*, solve the following equation for *f*, with an initial estimate for *f*. Then, carry out a case study of *f* versus values of *Re* from 1000 through 1000000, spaced logarithmically.

$$\frac{1}{\sqrt{f}} - 4 \cdot log_{10}\left(Re \cdot \sqrt{f}\right) + 0.4 = 0$$

We can use *fzero* to solve for *f*, given one value of *Re* as follows.

Anonymous Functions

Create a function to compute the equation error [= 0 when the equation is solved]

```
function result=fanning(f)
Re=10000;
result=1/sqrt(f)-4*log10(Re*sqrt(f))+0.4;
```

Use *fzero* to find the solution

```
>> fzero(@fanning,0.05)

ans =

    0.0077
```

But now we want to solve the equation for *f*, and for many values of *Re*. That means we cannot set the value of *Re* in the *fanning* function.

Anonymous Functions

Set up a range of *Re* values spaced logarithmically.

```
npoints=100;
Re=logspace(3,6,npoints);
```

Now, we want to solve the equation for each value of *Re*, in an m-script like

```
for i = 1:npoints
    % need to use fzero to solve for f
    % for each of the Re(i) values
end
```

The problem is: How does the value of *Re(i)* get communicated "through" the built-in *fzero* function to the *fanning* function?

This dilemma is resolved through the use of an anonymous function.

Anonymous Functions

Function *f_anon* is defined, allowing an additional argument, *Re(i)*, to be passed to function *fanning*

```
% m-script for case study
npoints=100;
Re=logspace(3,6,npoints);
fs = zeros(npoints,1);
fstart = 0.05;
for i = 1:npoints
    f_anon = @ (f) fanning(f,Re(i));
    fs(i) = fzero(f_anon,fstart);
    fstart = fs(i);
end
```

and function *fanning* now needs to be modified to accommodate the additional argument

```
function result=fanning(f,Re)
result=1/sqrt(f)-4*log10(Re*sqrt(f))+0.4;
```

Anonymous Functions

Analyzing the m-script:

```
Re=logspace(3,6,npoints);
```

Set up *npoints* (100 typical) equally spaced by the logarithm of *Re* between 1000 (10^3) and 1000000 (10^6)

```
fs = zeros(npoints,1);
fstart = 0.05;
```

set up an *fs* vector for the solutions initially filled with 0's

set an initial guess for *f* at 0.05

```
for i = 1:npoints
```

solve the equation for each value of *Re(i)*

```
f_anon = @ (f) fanning(f,Re(i));
```

define an anonymous function *f_anon* based on the *fanning* function with an identified argument *f* and an extra argument *Re(i)*

Anonymous Functions

Analyzing the m-script:

```
fs(i) = fzero(f_anon,fstart);
```

call *fzero* with the anonymous function as the first argument and *fstart* as the starting guess for the solution – the solution is stored in *fs(i)*

```
fstart = fs(i);
```

use the *fs(i)* value as the starting guess for the next solution of the equation

Create a semilog plot of the *f* solutions vs *Re*

```
>> semilogx(Re,fs,'k');grid
>> xlabel('Re')
>> ylabel('f')
>> title('Case Study of f versus Re')
```

Anonymous Functions