Lab 11

- Global variables
- Function handles

Background in Chapter 7.3

Global Variables

- Allows information to be shared among several functions without passing the information as an argument.
- Must be declared as a global variable in all the functions that needs to access it.
- Should be used when the alternative becomes too tedious.

The global keyword

To make a variable global, initialize it in the workspace:

```
global myvariable
myvariable = …;
```

All functions that want to use this variable must declare

```
global myvariable
```

Example of global variable

We want to define functions for converting between nautical miles and km so we easily translate data from on system to another:

```
distance_km = nmiles2km (60);
distance_nm = km2nmiles (1000.0);
```

Example of a global variable

In order to make the right conversions we need to set up a scaling factor that relates nautical miles and kilometers. How long is a nautical mile????

```
global NM2KM
NM2KM = 1.852;      % km in one nautical mile
```

Now we can make functions nmiles2km and km2nmiles that use this global scaling factor.

Example of a global variable

```
function km = nmiles2km (nm)
% NMILES2KM   Converting lengths in km to nautical miles
% km = nmiles2km (nm)
% Input:    nm, distance in nautical miles
% Output: km, the same distance in kilometer
```

```
function nm = km2nmiles (km)
% KM2NMILES   Converting lengths in nautical miles to km
% nm = km2nmiles (km)
% Input:    km, distance in kilometer
% Output: nm, the same distance in nautical miles
```

Example of a global variable

```
function km = nmiles2km (nm)
% NMILES2KM   Converting lengths in km to nautical miles
% km = nmiles2km (nm)
% Input:    nm, distance in nautical miles
% Output: km, the same distance in kilometer
global NM2KM
km = nm .* NM2KM;
```

```
function nm = km2nmiles (km)
% KM2NMILES   Converting lengths in nautical miles to km
% nm = km2nmiles (km)
% Input:    km, distance in kilometer
% Output: nm, the same distance in nautical miles
global NM2KM
nm = km ./ NM2KM;
```

Function handles

- A function handle is a reference (or pointer) to a defined function
- You create a handle by using the @ symbol before the function name

```
trig_handle = @cosd;
```

- To run the function referenced by the handle, use feval

```
cos_30 = feval (trig_handle, 30);
```

Why use function handles?

- Function handles can be passed as arguments to other functions
  - Such functions are called *function functions*
- The extra layer of abstraction allows many types of algorithms to be vastly simplified (see today's lab).
  - Initial if/switch testing to assign a handle to a certain function call
  - Repeated calls to that handle without further tests speeds up execution and simplify the code

Function calls w/ or w/o handle

- Traditional call:

  [out1, out2, …] = functionname (in1, in2, …);

- Call via handle:
  - fhandle = @functionname;
  - …
  - [out1, out2, …] = feval (fhandle, in1, in2, …);

Matlab function functions

- fplot, for plotting a function
  - fplot (handle, limits), e.g.

    fplot (@cosd, [0 360]);
- quad, for integrating a function
  - quad (handle, a, b), i.e.,

    quad (@sqrt, 0, 1)