# Goals

- Turn your running median script into a function to compute a running median and implement various checks
- CODE: Function definition and help lines
- CODE: subfunctions
- CODE: function calls from a main script
- CODE: Useful MATLAB functions: `error`, `rem`, `size`

Note: Do NOT use the built-in MATLAB functions for computing running means and running medians in this lab.

# The Lab

From the week 6 lab you should have a script to compute a running median and a running mean. One data set you used was contained in the file `aircraft_gps.mat`. You'll use that file again here. If your code didn't work you can use one of the posted solutions for that lab from the web site. If you use a posted solution convince yourself it works by plotting the raw data as well as the running mean and running median (x-axis should be the variable `gps.mtime` which you can display in hrs, mins using `datetick`).

To make this code easier to use repeatedly, and to check for user input errors, we will a) turn this code into a FUNCTION, and b) add some code that performs INPUT CHECKS to test for inappropriate inputs, embedded in the function itself.

1. Function: `calcmedian.m`

   Turn your `runmean` script into a function called `calcmedian` that takes ONLY two input arguments (`invec` and `winlen`, IN THAT ORDER), computes ONLY the **running median**, and returns ONLY `outvec`, a vector containing the running median. Provide comment lines to explain how you dealt with the ends of your array. Make sure your function contains the H1 line, indicating the function usage, and the input and output arguments.

2. Script: `testmedian.m`

   Write a short script called `testmedian` to (1) load the GPS data file, (2) call the running median function using input data `gps.vel` and a window length of 7, and (3) plot in the same figure the raw velocity data `gps.vel` and your running median (This script should be a lot like `runtest` from the week 6 lab). Check that the running median is working correctly by examining your plot.

   It is fun (and realistic) to use the aircraft GPS data to test your algorithm. However, one problem with this is that rather subtle problems can get missed by a quick visual check because the data is somewhat noisy already. You can (and should, if possible) also always test algorithms with test data that you make up yourself. Here's an example of a test data set that we used to grade your original running median script. Try using it for debugging:

```
y=[-1:.025:1];     % parabolic shape shows issues with window 'centering',
                   % also shows scaling problems
invec=-y.^2;       % Predictable shape at ends makes problems in
                   % end-handling easier to see
invec(1)=0;        % make end different (helps w/ end effects)
invec(40)=1;       % Single spike filtered out by median filter
invec([55:60])=1; % ...but longer step is not (for small window lengths)
                   % single spike and step also have 'nice' properties for
                   % ...running mean
winlen=11;
t=1:length(y);
```

3. Subfunction: `checkinputs`

   Write a subfunction called `checkinputs` that is called by `calcmedian` and does some INPUT CHECKS before starting with the calculations. Input checks are various tests on the inputs to see if they will cause problems in the later calculations. The code for `checkinputs` should be physically written in the same file as `calcmedian` (see example of subfunction in your textbook). The subfunction should print a message indicating the nature of the problem and should stop the program if:

   - `invec` is not a column or row vector.
   - `invec` is shorter than `winlen+1`.
   - `winlen` is even.
   - `winlen` is less than 3.

   *Hint:* The built-in MATLAB functions `size` and `rem` might be useful in logical tests.

   The built-in MATLAB function `error` is key for these kinds of checks, because it can be used to print an informative error message AND stop the program. Type `help error` to understand why. REMEMBER to test your code by trying out "wrong" scenarios – e.g., check that you get the correct error message if you use an even # for `winlen` in a call to `calcmedian`.

# To Hand In

Submit your function `calcmedian.m` via Connect by 4pm on Friday. I will run it using a script that loads a data file and I will check that (1) the running median is implemented properly (I will check this by plotting the raw and running median data), (2) your function returns the appropriate error messages if I attempt to run it with the wrong input (e.g., if I attempt to use an even number for `winlen`).

As usual comment your code and add the following info AS A COMMENT at the bottom of the function header comment:

```
%  partner.name=  'YYYYYY';
%  Time_spent=   XX   hours
```