```
% There are many possible solutions to this weeks lab
% The simplest is something like this

%---------beginning of runmean------------
% inputs x, winlen, winlen_med
% output z, zm

N=length(x);

% In this soln I replace the first and last w terms with NaN, basically by
% just not overwriting the default values I set here in the preallocation
z=NaN(1,N);

w=(winlen-1)/2;
for i=w+1:N-w,

  sm=0;
  for k=-w:w,
    sm=sm+x(i+k)
  end;
  z(i)=sm/winlen;

end;

wm=(winlen_med-1)/2;
zm=NaN(1,N);
for i=wm+1:N-wm,
  zm(i)=median(x(i+[-wm:wm]));
end;
%----------end of runmean---------

% - Note that I try to keep the 'i+k' form of the original mathematical formula,
%   and I use the trick of filling the z and zm arrays with NaN beforehand.
%
% - In fact it  is a GOOD IDEA to pre-allocate large  arrays (otherwise things
%   start to run MUCH slower  with large N).
%
% - If you find yourself using (winlen-1)/2 a lot - save it to a variable and
%   then use *that* variable! (the same with 'length(x)')
%
% - Also, in *this* lab I didn't want you to use the 'sum' or 'mean' functions,
%   but in future you can make things a bit simpler with them.  Just remember
%   that you are really replacing a structure of the form:

  sm=0;
  for k=-w:w,
    sm=sm+x(i+k)
  end;
  z(i)=sm/winlen;

% with

  z(i)=mean(x(i+[-w:w]));

% which you did anyway for the median.
%
%---------------------------------------------------
% There are lots of OTHER ways to handle the
% end effects
% 1) replace with the original (unsmoothed) data (can do this above by starting
%    with 'z=x')
% 2) take the first value you can calculate, and copy that back to the previous
%    locations
% 3) do a 'circular' procedure, where going off the RHS involves taking points
%    from the LHS (and vice versa), a bit like going around the  world in the
%    slope lab  - useful for, e.g., things measured as a function of angle
%    around a circle (also in fourier transforms)
% 4) fiddle with the window parameters in one of several ways, for the left edge
```



```
%    you could use windows of
%      i)   1:i+w, divide by winlen (ignore missing
%                  points, but get a biased result)
%      ii)  1:i+w, divide by i+w (ignore missing
%                  points, get unbiased result)
%      iii) 1:i+(i-1), divide by 2i-1 (shrink
%                  window so it doesn't need the missing points)
%
%  The last (shrinking the width of the centered window) can be done with
%  something like

for i=1:N
  if i<wm,
    wl=i-1;  % length 0 for first point, 3 for 2nd point, etc.
  elseif i>N-wm,
    wl=N-i;
  else
    wl=wm;
  end;
  % Now use 'wl' for window length
  % ...etc., e.g. for the median:
  z(i)=median(x(i+[-wl:wl]));
end;

% but IF you can recognize that the if block is just computing minimums,
% it CAN be replaced with a single line:

for i=1:N,
  wl=min([i-1;wm;N-i]);
  z(i)=median(x(i+[-wl:wl]));
end;

%
% Note also that you have the question of whether to handle end effects (i.e.
% the if/else/block) INSIDE the main for loop, or whether you put the main
% for loop INSIDE the if/else/block.
%
% In this case its probably better to put it inside, but this is not always
% true.

%----------------------------------------------
%
% The next trick with this is to make sure you are getting the right answer. It
% helps a lot to visualize things. For example, if you have a time series with
% a 'spike' in the middle you can easily see if the centering is off:

z=[0 0 0 0 0 0 0 0 10 0 0 0 0 0 0 0];

% the running mean should replace the spike with
% a sequence of values - say, 5 2's (10/5=2) with
% a 5 point window for the mean.
%

% Here's what I used to mark the lab:
y=[-1:.025:1];
invec=-y.^2;    % Smooth shape (picks out errors more easily)
invec(40)=1;    % A single spike - median should remove this
invec([55:60])=1; % A high reagion - median does NOT remove this
invec(1)=0;     % A weird boundary lets me see how end effects
                % are handled.
x=invec;
winlen=9;
winlen_med=11;

% RP - 17/Oct
```