matPIV

matPIV is a simplified PIV (Particle Image Velocimetry) library for MATLAB.
Local image displacements are calculated using image cross-correlation.
The algorithms included in this library are modeled after the methods presented in:
Hart, D.P. "Super-resolution PIV by Recursive Local-correlation". Journal Visualization , Vol 3. No. 2 (2000) pp. 187-194 [DOI: 10.1007/BF03182411]

Usage:

Recursive PIV:

```
[vx,vy,x,y,vx_nan,vy_nan] = piv_rec(im1,im2,...)
% Recursively calculate PIV vectors
% Inputs:
%   im1, im2:   input images (grayscale matricies)
% Parameters:
%   'startW',w  starting width of correlation window (default: 64)
%   'startH',h  starting height of correlation window (default: 64)
%   'endW',w    ending width of correlation window (default: 16)
%   'endH',h    ending height of correlation window (default: 16)
%   'InitScale',s factor by which to rescale initial image during
%                   calculation of image drift
% Output:
%   vx,vy   difference vectors at locations specified by x,y
%   x,y     position of difference vectors (e.g. x=[0.5,16.5,32.5...])
%   vx_nan,vy_nan
%           difference vectors with NaNs at the location where correlation
%           did not return a useable result.  Values of vx,vy at those
%           locations are interpolated from the previous itteration.
%*************************************************************************
```

Recursively calculate PIV vectors. The algorithm starts by finding the whole-scale displacement using the cross-correlation of the first image with the second. It then recursively scans smaller windows to generate the image-difference vector field. The vector corresponding to a given sampling window is used to inform the location of the window sampled from the second image in subsequent steps. The algorithm returns after reaching a minimum window size, specified with the options 'endW' and 'endH'.

Non-recursive PIV:

Calculate PIV displacement field between two images.

```
[vx2,vy2,x2,y2,vx_nan,vy_nan] = piv(im1,im2,dW,dH,varargin)
% PIV image tracking function
% Uses image cross-correlation to determine "flow-field" difference between
% two images.
% Inputs:
%   im1 first image
%   im2 second image
%   dW width (in pixels) of the cross-correlation window
```

```
%   dH height (in pixels) of the cross-correlation window
%       Without specifying any additional arguments, the flow field will be
%       calculated with windows which overlap by 50%.
% Optional inputs:
%   piv(...,x,y)    row vectors specifying the locations to use
%                   when compute correlation. 0.5 increments are
%                   interpreted as the space between two pixels.
%   piv(...,x,y,vx,vy)  specify initial guess for flow field
%               x,y are row vectors specifying locations of flow points
%               vx,vy are matricies of the flow vectors in x and y
%               directions, respectively.  The vectors should coorespond to
%               the locations given by [XX,YY] = meshgrid(x,y)
%   piv(...,x,y,vx,vy,'interp')
%               Use the specified flow field as above, but interpolate a
%               new mesh that is twice the density of the original.
%               Example:
%                   x=[1,2,3] => [1,1.5,2,2.5,3]
%   piv(...,x,y,vx,vy,'interp',method)
%               method: interpolation method to use [see interp2()]
%                   'linear' (default)
%                   'nearest'
%                   'cubic'
%                   'spline'
% Outputs:
%   vx2: array of flow vectors in x direction
%   vy2: array of flow vectors in the y direction
%   x2: vector listing x-location of vx2
%   y2: vector listing y-location of vy2
%   vx_nan: array of x-vectors, with NaNs indicating locations which
%           could not be computed because the cross-correlation did not
%           yield a peak of stron enough signal
%   vy_nan: array of y-vectors with NaNs...
% ************************************************************************
```

Pre-calculate Vector-field Locations:

```
[x,y] = locPIV(imsz,sW,sH,eW,eH)
% Returns the location for velocity matrix returned by:
%   [...,x,y,...]=piv_rec(...,'startW,sW,'endW',eW,'startH',sH,'endH',eH)
%  Default:
%       startW = 64, startH = 64
%       endW = 16, endH = 16
% ========================================================================
```

Use this function to calculate the vector-field locations that will be returned by piv_rec().
This is very useful when trying to pre-allocate memory.

Filtering Results to eliminate errors:

Vector-fields generated with PIV may contain erroneous vectors which don't correspond with neighboring vectors. residfilt() can be used to identify and replace these vectors. The algorithm is based

on the methods present in: Wester & Scarano “Universal outlier detection for PIC data”, Experiments in Fluids (2005) 39 pp. 1096-110 [DOI: 10.1007/s00348-005-0016-6]

```
[qxf,qyf,qxn,qyn,Res] = residfilt(qx,qy,resth,varargin)
% Filter vector field <qx,qy> using Westerweel & Scarano outlier detection
% Inputs:
%   qx: x-components of the vector field
%   qy: y-compnenets of the vector field
% Optional Input:
%   resth:  residual threshold value, above which a vector is deemed an
%           outlier and replaced (default 3.0)
% Parameters:
%   'radius',r  number of neighboring vectors to use in determing if a
%               vector is an outlier (default: 1)
%   'error',er  Approximate estimate of data variability (default: 0.1)
%   'method',str:
%       'median' (default) use median vector value to compute residual and
%                replacement vectors
%       'linear' Fit data in neighborhood to a least-squares approximation
%                of a plane (v=ax+by+c).  Use the expected value to
%                computer residuals
% Outputs:
%   qxf, qyf: The filtered vector field
%   qxn, qyn: The vector field with identified outlier points replaced with
%             NaN
%   Res:    The residual value computed at each location.
%==========================================================================
```

In the case where the is very little motion, the PIV algorithm will produce vectors field with low levels of noise, due to inaccuracies in the sub-pixel peak detection algorithms. These can be smoothed using an adaptive noise filter such as the Wiener filter. MATLAB includes an Wiener filter with the “Image Processing Toolbox”. Its use is very simple

```
qx = wiener2(qx,[5,5]);
qy = wiener2(qy,[5,5]);
```

Included sub-functions:

Find local maxima:

Find the local maxima by searching a 3x3 neighborhood for the pixel of highest intensity.

```
[out,ind]=pkfnd(im,th,sz)
```

Find the local maxima by dilating the image and looking for pixels which don’t change

```
 [PKS,IND] = pkfnd2(im,sz,thresh)
```

Sub-pixel local-maxima:

```
XY = pkfnd_subpixel(im,PKS,method) % Find subpixel pixel peak
% Input:
%   im grayscale image
%   PKS [X,Y] local maxima locations (e.g. use pkfnd or pkfnd2 to get these)
%   method: 'centroid' use 3-point weighted average (default)
```

```
%           'parabolic' estimate peak with parabola
%           'gaussian' use gaussian 3-point fitting
%       Note:   All calculation are currently using 3-point fitting. We might
%               want to change this in the future.
% Output:
%   XY [X,Y] locations of the centroids associated with each peak in PKS
% ========================================================================
```

Get highest peak:

```
[XY,SNR] = imfindpeak(C)
% Finds the highest peak in an image and returns the subpixel location of
% that peak using a 3-point gaussian fitting.
% Input:
%   C input image
% Output:
%   XY = [X,Y] the sub-pixel location of the highest peak
%   SNR     ratio of value of largest peak to second largest peak, or average
%           value of C if no other peaks are detected
%************************************************************************
% Requirements:
%   pkfnd(...) by Dufresnes et al. (needs DTK modification).
%   pkfnd_subpixel(...) by D. Kovari
%========================================================================
```