NAME

flct - perform local correlation tracking between 2 images

SYNOPSIS

flct in-file out-file deltat deltas sigma [-t thresh -k kr -s N[pP][qQ][i] -pc latmin latmax -h -bc -q] ...

DESCRIPTION

flct finds a 2-D velocity field from which an initial image (image1) evolves into a 2nd image (image2) over a time deltat. The technique "FLCT", was initially desribed in section 3 of Welsch B. T., Fisher G. H.,, Abbett W. P., and Regnier, S. 2004 ApJ 610, 1148. Following this, an updated writeup of the technique reflecting improvements to the code was published by G. H. Fisher and B. T. Welsch in PASP conf. series vol 383, p 373 (2008) (publicly available at http://arxiv.org/abs/0712.4289). The current technique can be summarized as follows:

For each pixel location in each of the 2 images, first form smaller sub-images that are centered on the given pixel location, perform a mean subtraction of each subimage, and then multiply each of these subimages with a 2-D gaussian of width sigma pixels, centered at the given pixel location. Then compute the cross-correlation function between the 2 resulting truncated images, and find the pixel shifts in x and y that maximize that function. The pixel shifts are converted to velocity units by the magnitude of the time separation deltat and the unit of length along the edge of a single pixel, deltas. The cross-correlation function is computed using standard Fourier Transform techniques, employing version 3 of the FFTW (http://www.fftw.org) package. Second-order Taylor expansion of the solution is done to find the sub-pixel location of the peak of the cross-correlation function.

When sigma is set to zero, a special case, the cross-correlation function of the two full images is computed, without multiplying by the gaussian. The net shift in x and y between the two full images is then printed to stdout, and the output file out-file contains single values for the shifts in the output velocity arrays.

If desired, the IDL procedure shift frac2d, which is included in the flct distribution, can then be used to remove this overall shift from image2. shift frac2d works in the same was as the IDL shift function, but allows for non-integer shifts.

The two input images are read from in-file , which is a binary-format file that can be written with the IDL procedure vcimage2out.pro, out-file contains the resulting 2-D arrays of vx, vy, and vm (the x and y components of the velocity, and a mask array set to 0 for those locations where no velocity is computed, 1 where it is, and 0.5 where it is interpolated -- see discussion of interpolation in the section on the "skip" option). The output file can be read with the IDL procedure vcimage3in.pro. The data in in-file and out-file is stored in binary, large-endian byte order, and flct and the IDL I/O procedures to read and write the files should be platform independent.

OPTIONS

-t thresh

Do not compute the velocity at a given pixel if the average absolute value between the 2 images at that location is less than thresh. If thresh is between 0 and 1, thresh is assumed given in units relative to the largest absolute value of the image averages. To force a thresh value between 0 and 1 to be considered in "absolute" units, append an "a" to the numerical value of thresh. If the velocity isn't computed, the mask array is set to 0 at that location (it is 1 otherwise).

-k kr Perform gaussian, low pass filtering on the sub-images that are used to construct the cross-correlation function. The value of kr is expressed in units of the maximum wavenumber (Nyquist frequency) in each direction of the sub-image. Specifically, the complex amplitude of the (kx,ky) Fourier mode of each sub-image is multiplied by exp(-(kx/kxr)^2-(ky/kyr)^2), where kxr = kr*kx_max and kyr=kr*ky_max, and where kx_max, ky_max are the Nyquist frequencies of the sub-image.

This option is most useful when the images contain significant amounts of uncorrelated, pixel-to-pixel noise-like structure. Empirically, values of kr in the range of 0.2 to 0.5 seem to be most useful, with lower values resulting in stronger filtering.

-s N[pP][qQ][i]

Only compute the velocity every N pixels in both the x and y directions. This "skip" option is useful when the images are very large and one does not need the velocity computed at every location in the two images. The sub-options pP qQ and i are each optional, but if present, they must occur in the stated order. The subparameter pP where P is an integer, is the offset for the start of computation in the x-direction of the array, and similarly qQ where Q is an integer, is the offset for the start of computation in the y-direction of the array. The default values of P and Q are zero (no p or q present in the skip string). P and Q must be smaller in absolute value than the skip integer N. If P or Q are negative, they are reset to N-|P| or N-|Q|. The suboption i determines if cubic convolution interpolation is to be done on the points that were skipped, using values from the points that were computed. If interpolation is done on a given gridpoint, then the mask array vm that is written out to out-file contains a value of 0.5 for those points that were interpolated. The skip string argument must not contain any blanks.

Empirically, setting N > sigma / 2 pixels and then interpolating compares poorly with calculations performed at every gridpoint.

If using the skip option while also using the "Plate Carree" option (see below), you must also use the interpolation suboption described above.

-h If this "high resolution" flag is set, cubic convolution interpolation is performed to 0.1 pixel precision before Taylor expansion is done to find the location of the peak.

We have DISABLED this option because it is slower and less accurate than the default method.

-pc latmin latmax
If this "Plate Carree" option is set, then the input images are assumed to be in Plate Carree coordinates (uniformly spaced in longitude and latitude). In this case, the images are interpolated to a Mercator Projection, then FLCT is run, and the resulting velocities are interpolated back to Plate Carree coor-

dinates, with cos(latitude) modulation. The limits latmin and latmax are assumed to be given in radians, unless they end in 'd', in which case they are assumed to be in degrees, and will then be converted to radians. Because of possible artifacts in low-signal regions of the image due to the interpolation algorithm, we strongly recommend using the thresholding option ("-t thresh") when using the Plate Carree option.

If sigma is set to 0 when the Plate Carree option is set, then single values of the x and y shifts are returned, and a single value of 0.5 is returned as the mask value. The x and y shifts are modulated by the cosine of the average latitude, coinciding with the centroid of the image in pixel space.

-bc If this flag is set, an attempt is made to correct the bias that under-estimates velocity amplitudes. This bias is intrinsic to the FLCT algorithm. The bias correction algorithm that is implemented uses the Hessian determinant, and the assumed value of sigma, to adjust the x-and-y velocities. More detail is given in the file bias_correction_in_flct.txt in the FLCT distribution.

-q If this flag is set, no non-error output is sent to stdout.

EXAMPLES

example using a shifted image in an IDL session, with sigma=15:

```
IDL>f1=randomu(seed,101,101)

IDL>f2=shift_frac2d(f1,1.,-1.)

IDL>vcimage2out,f1,f2,'testin.dat'

IDL>$flct testin.dat testout.dat 1. 1. 15.

IDL>vcimage3in,vx,vy,vm,'testout.dat'

IDL>shade_surf,vx

IDL>shade_surf,vy
```

Same as above, but only computing every 5 pixels, and then interpolating:

```
IDL>$flct testin.dat testout.dat 1. 1. 15. -s 5i
```

Same as above, but only computing every 5 pixels, with 1 pixel x and 2 pixel y offsets, and then interpolating:

```
IDL>$flct testin.dat testout.dat 1. 1. 15. -s 5p1q2i
```

Remove a net shift between images f1 and f2, using sigma=0 (result into f3):

```
IDL>vcimage2out,f1,f2,'testin.dat'

IDL>$flct testin.dat testout.dat 1. 1. 0 -k 0.5

IDL>vcimage3in,delx,dely,delm,'testout.dat'

IDL>f3=shift_frac2d(f2,-delx,-dely)
```

Same as 1st example, but using low-pass filtering, run outside of IDL:

```
flct testin.dat testout.dat 1. 1. 15. -k 0.25
```

Same as previous, but only compute vel. for avg abs. image values above 0.5:

```
flct testin.dat testout.dat 1. 1. 15. -t 0.5a
```

Print out short summary of documentation:

```
flct
```

FILES

There are no configuration files.

KNOWN LIMITATIONS

flct is unable to find flows that are normal to image gradients. This is a defect of the LCT concept.

flct cannot determine velocities on scales below the scale size of structures in the images. This is a defect of the LCT concept.

Images that have minimal structure can give nonsensical velocity results.

Results can depend on value of sigma. User must experiment to determine best choice of sigma.

Velocities corresponding to shifts less than 0.1-0.2 pixels are not always detected. It may be necessary to increase the amount of time between images, depending on the noise level in the images. Sometimes using the filtering option helps.

Velocities computed within _sigma_ pixels of the image edges can be unreliable.

Noisy images can result in spurious velocity results unless a suitable threshold value _thresh_ is chosen.

AUTHORS

George H. Fisher, SSL UC Berkeley <fisher at ssl dot berkeley dot edu>
Brian T. Welsch, SSL UC Berkeley <welsch at ssl dot berkeley dot edu>

SEE ALSO

source code of vcimage2out.pro (IDL procedure), source code of vcimage3in.pro (IDL procedure), and source code of shift_frac2d.pro (IDL procedure).