Table of Contents

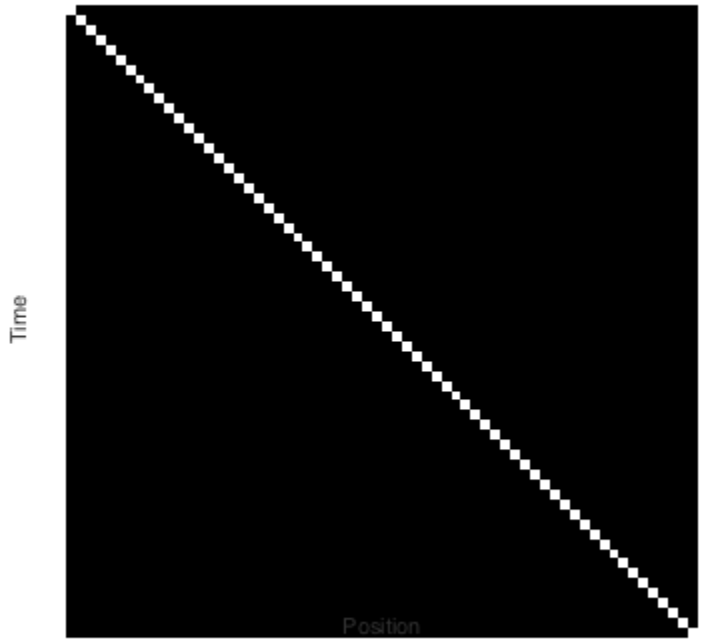

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  File: motion1D.m
%  Author: Eero Simoncelli
%  Spring, 1998.  Upgraded, 6/02.
%
%  Partially based on earlier Cold Spring Harbor Tutorials written by
%  Eero Simoncelli and Geoff Boynton.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% This tutorial presents some concepts for representing and analyzing
% visual motion. The examples are done in a single spatial dimension, to
% keep things simple. We first describe an implementation of a "motion
% energy" model, as described by Adelson & Bergen in 'Spatiotemporal Energy
% Models for the Perception of Motion' (JOSA-A,2:284-299). We show how
% linear separable filters can be added and subtracted to form space-time
% oriented linear filters. These oriented filters, which are selective to
% direction of motion, can be combined nonlinearly to produce filters that
% are not only motion selective, but are also insensitive to phase. Such
% response properties are typical of V1 complex cells.

% We also describe the relationship of this construction to the
% gradient-based algorithms widely used in Computer Vision for estimating
% optical flow.

% Finally, we'll look at these solutions in the Fourier domain.

% To run the tutorial, you need to have matlabPyrTools in your matlab path.
% The matlabPyrTools are available at:
%
% http://www.cns.nyu.edu/pub/eero/matlabPyrTools.tar.gz

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Motion in space-time and spatiotemporal linera filters

```
% Motion corresponds to orientation in space-time. Consider a spot of light
% one pixel wide, translating to the right at one pixel per frame. We can
% view the sequence of signals in the X (space) and T (time) dimensions,
```

```
% with amplitude corresponding to intensity:
stim = eye(64);
figure(1);  showIm(stim);
xlabel('Position');  ylabel('Time');
set(get(gca,'Ylabel'),'Visible','on')

% The horizontial axis specifies spatial position, and vertical axis
% corresponds to time (up=earlier, down=later). Each pixel gives the
% intensity of the signal at a different spatial position and time. The
% motion of the spot corresponds to the fact that its position changes over
% time. This is just the (inverse of) the slope of the line traced out by
% the spot in space-time.

% In order to represent or estimate local velocity, one needs to make
% measurements of this space-time orientation.  We'll do this using a set
% of spatio-temporal filters.
```

```
WARNING: You should compile the MEX version of "range2.c",
         found in the MEX subdirectory of matlabPyrTools, and put it in your matla
```

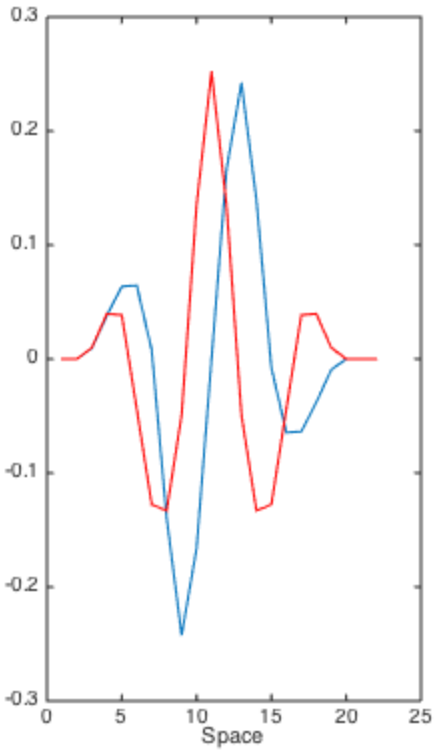


First, we construct two (one-dimensional) spatial filters:

```
sfilt = upBlur([0 0 0.107517 0.074893 -0.469550 0 ...
      0.469550 -0.074893 -0.107517 0 0]);
sdfilt = upBlur([0 0 0.201624 -0.424658 -0.252747 0.940351 ...
      -0.252747 -0.424658 0.201624 0 0]/1.8);
figure(1)
```

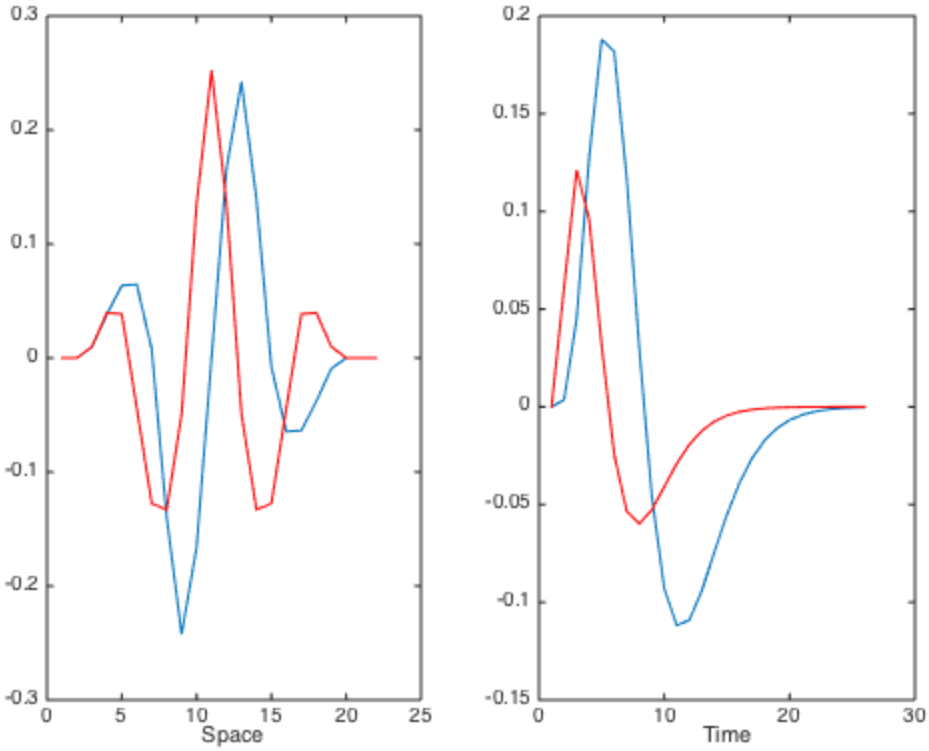
```
subplot(1,2,1); plot(sfilt); hold on; plot(sdfilt,'r'); hold off
xlabel('Space');
```

```
WARNING: You should compile the MEX version of "upConv.c",
         found in the MEX subdirectory of matlabPyrTools, and put it in your matla

stop =

     1    22

WARNING: You should compile the MEX version of "upConv.c",
         found in the MEX subdirectory of matlabPyrTools, and put it in your matla

stop =

     1    22
```



Next, create two causal temporal filters, one that contains more ripples than the other:

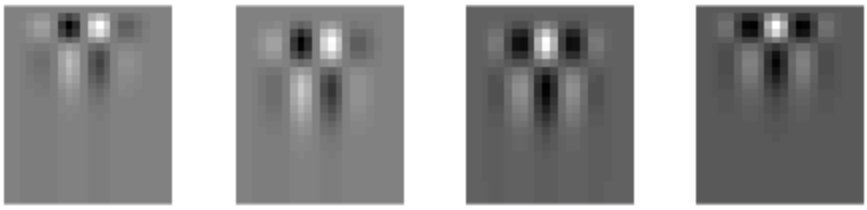
```
tsz = 25;
tfilt = 2*temp_imp_resp(5,22,[0:tsz]'/tsz);
tdfilt = temp_imp_resp(2.5,22,[0:tsz]'/tsz)/2.5;
subplot(1,2,2); plot(tfilt); hold on; plot(tdfilt,'r'); hold off
xlabel('Time');
```



And now we create separable filters by taking the outer products of the four combinations of spatial and temporal filters:

```
even_slow = tfilt * sdfilt;
even_fast = tdfilt * sdfilt ;
odd_slow = tfilt * sfilt ;
odd_fast = tdfilt * sfilt ;

figure(2)
subplot(2,4,1);showIm(odd_fast,'auto','auto',0);
subplot(2,4,2);showIm(odd_slow,'auto','auto',0);
subplot(2,4,3);showIm(even_slow,'auto','auto',0);
subplot(2,4,4);showIm(even_fast,'auto','auto',0);
```

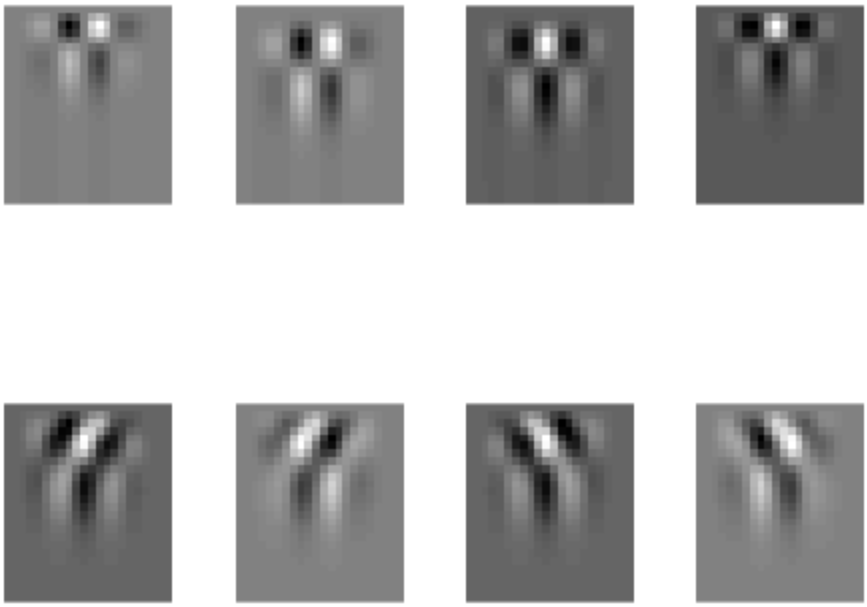
```
WARNING: You should compile the MEX version of "range2.c",
         found in the MEX subdirectory of matlabPyrTools, and put it in your matla
WARNING: You should compile the MEX version of "range2.c",
         found in the MEX subdirectory of matlabPyrTools, and put it in your matla
WARNING: You should compile the MEX version of "range2.c",
         found in the MEX subdirectory of matlabPyrTools, and put it in your matla
WARNING: You should compile the MEX version of "range2.c",
         found in the MEX subdirectory of matlabPyrTools, and put it in your matla
```



These separable filters can be added or subtracted in pairs to produce filters that are space-time oriented, roughly analogous to the receptive field properties of V1 simple cells. For example, adding the odd_fast filter to the even_slow filter results in a linear filter that is selective for leftward motion. We can make four such filters:

```
leftward_1=odd_fast+even_slow;
leftward_2=-odd_slow+even_fast;
rightward_1=-odd_fast+even_slow;
rightward_2=odd_slow+even_fast;

subplot(2,4,5);showIm(leftward_1,'auto','auto',0);
subplot(2,4,6);showIm(leftward_2,'auto','auto',0);
subplot(2,4,7);showIm(rightward_1,'auto','auto',0);
subplot(2,4,8);showIm(rightward_2,'auto','auto',0);
```

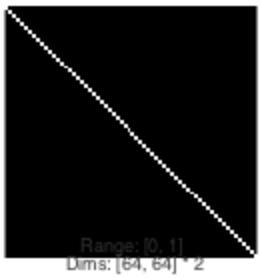
```
WARNING: You should compile the MEX version of "range2.c",
         found in the MEX subdirectory of matlabPyrTools, and put it in your matla
WARNING: You should compile the MEX version of "range2.c",
         found in the MEX subdirectory of matlabPyrTools, and put it in your matla
WARNING: You should compile the MEX version of "range2.c",
         found in the MEX subdirectory of matlabPyrTools, and put it in your matla
WARNING: You should compile the MEX version of "range2.c",
         found in the MEX subdirectory of matlabPyrTools, and put it in your matla
```

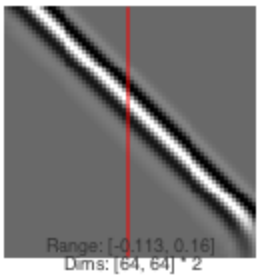


To compute a filter's response to such a stimulus, we simply perform a 2-d convolution of the stimulus with the filter. For example:

```
resp1 = conv2(stim,rightward_1,'same');
figure(3)
subplot(1,2,1); showIm(stim);
subplot(1,2,2); showIm(resp1);

% The way to interpret the response image is to consider each column of the
% image as the the time-course of the response of a filter (cell) centered
% at that location. The filter responds when the impulse passes by. For
% example, if we consider the response of a filter centered on the center
% column of the response image:
column=round(size(stim,2)/2); %center column
line([column,column],[1,size(stim,2)],'Color','r');
```

```
WARNING: You should compile the MEX version of "range2.c",
         found in the MEX subdirectory of matlabPyrTools, and put it in your matla
WARNING: You should compile the MEX version of "range2.c",
         found in the MEX subdirectory of matlabPyrTools, and put it in your matla
```

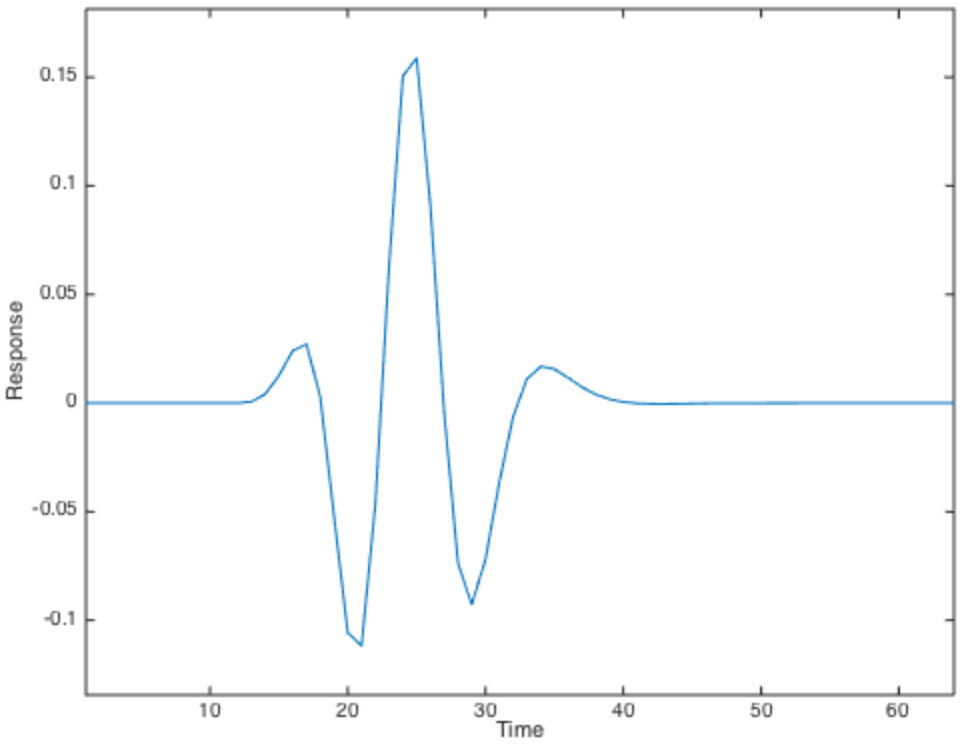




Now look at the time course of this response:

```
figure(4); clf
showIm(resp1(:,column)); ylabel('Response'); xlabel('Time')
```

```
WARNING: You should compile the MEX version of "range2.c",
         found in the MEX subdirectory of matlabPyrTools, and put it in your matla
```

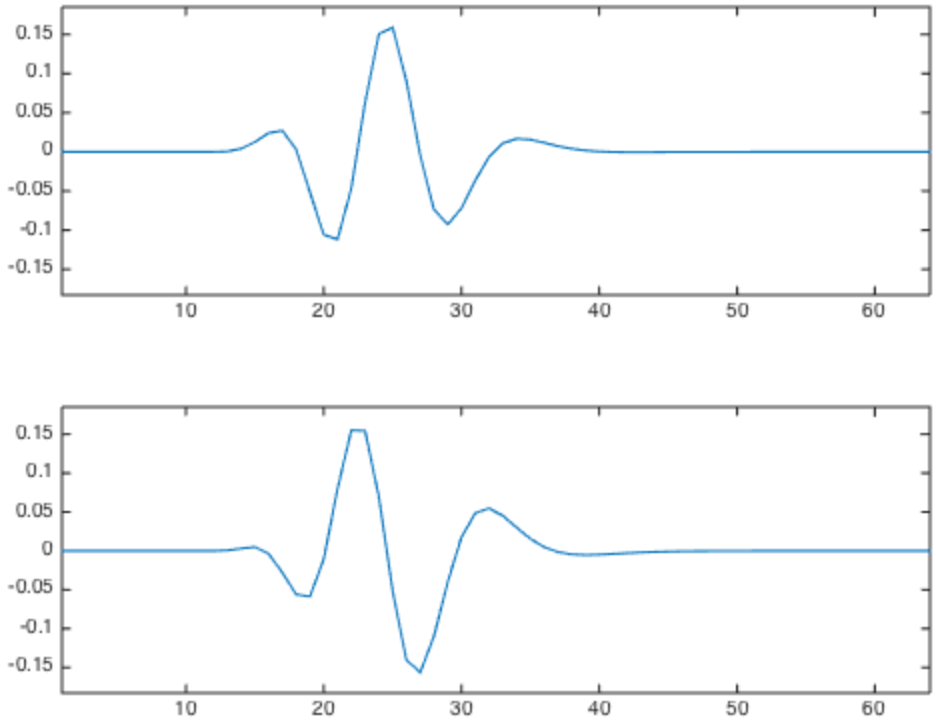


Similarly, consider the response of the rightward_2 filter:

```
resp2 = conv2(stim,rightward_2,'same');
subplot(1,2,2); showIm(resp2);
column=round(size(stim,2)/2); %center column
line([column,column],[1,size(stim,2)],'Color','r');

figure(4);
showIm(resp1(:,column)+j*resp2(:,column));
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

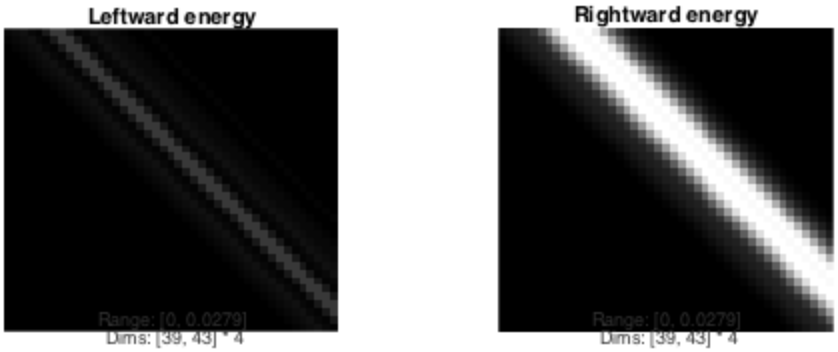
```
WARNING: You should compile the MEX version of "range2.c",
         found in the MEX subdirectory of matlabPyrTools, and put it in your matla
WARNING: You should compile the MEX version of "range2.c",
         found in the MEX subdirectory of matlabPyrTools, and put it in your matla
WARNING: You should compile the MEX version of "range2.c",
         found in the MEX subdirectory of matlabPyrTools, and put it in your matla
```



Motion Energy

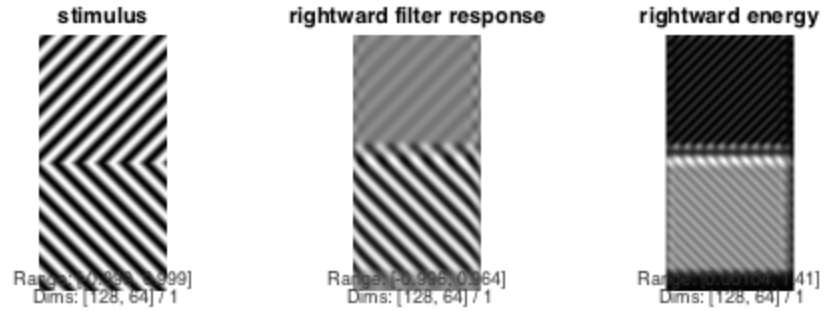
```
% Unlike the linear filter responses shown above, V1 complex cell responses
% are "phase-insensitive": they respond to drifting gratings with a
% more-or-less constant level of activity. Adelson/Bergen modeled this
% using "spatiotemporal energy": the sum of squared responses of filters
% with different phases. For example, we can compute rightward and
% leftwared energy, by adding together the squared responses:

Eright = conv2(stim,rightward_1,'valid').^2+conv2(stim,rightward_2,'valid').^2;
Eleft = conv2(stim,leftward_1,'valid').^2+conv2(stim,leftward_2,'valid').^2;
subplot(1,2,2); rg=showIm(Eright,'auto','auto','Rightward energy')
subplot(1,2,1); showIm(Eleft,rg,'auto','Leftward energy')
```

```
WARNING: You should compile the MEX version of "range2.c",
         found in the MEX subdirectory of matlabPyrTools, and put it in your matla

rg =

         0    0.0279


ans =

         0    0.0279
```



To see the phase-insensitivity, we make a new stimulus that contains a leftward- and rightward-moving sinusoidal grating:

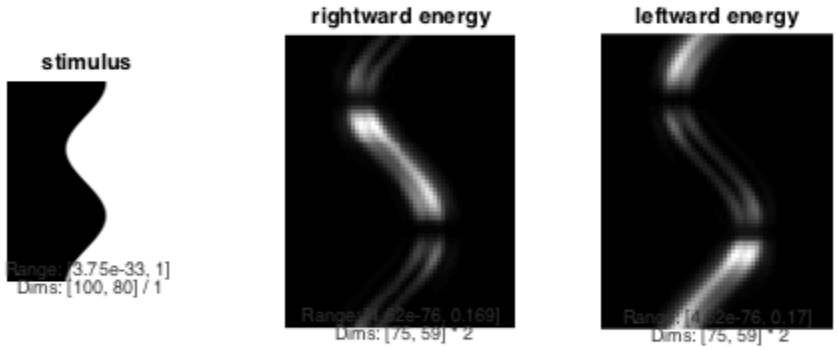
```
stim = [mkSine(64,8,pi/4); mkSine(64,8,-pi/4)];
figure(3)
subplot(1,3,1); showIm(stim); title('stimulus')

resp1 = conv2(stim,rightward_1,'same');
subplot(1,3,2); showIm(resp1);  title('rightward filter response')

resp2 = conv2(stim,rightward_2,'same');
Eright = resp1.^2+resp2.^2;
subplot(1,3,3); showIm(Eright); title('rightward energy')

WARNING: You should compile the MEX version of "range2.c",
         found in the MEX subdirectory of matlabPyrTools, and put it in your matla
WARNING: You should compile the MEX version of "range2.c",
         found in the MEX subdirectory of matlabPyrTools, and put it in your matla
WARNING: You should compile the MEX version of "range2.c",
         found in the MEX subdirectory of matlabPyrTools, and put it in your matla
```



Now try this on the Adelson/Bergen stimulus of figure 15:

```
dims = [100,80];
t_profile = dims(2)/2 + (dims(2)/8)*cos(3*pi*[0:dims(1)-1]'/dims(1));
[lutX,lutY] = rcosFn(2);
stim = pointOp((ones(dims(1),1)*[1:dims(2)])-(t_profile*ones(1,dims(2))), ...
    lutY, lutX(1), lutX(2)-lutX(1),0);

figure(4); clf
subplot(1,3,1); showIm(stim); title('stimulus')
Eright = conv2(stim,rightward_1,'valid').^2+conv2(stim,rightward_2,'valid').^2;
Eleft = conv2(stim,leftward_1,'valid').^2+conv2(stim,leftward_2,'valid').^2;
Estatic = conv2(stim,odd_slow,'valid').^2+conv2(stim,odd_fast,'valid').^2+...
     conv2(stim,even_slow,'valid').^2+conv2(stim,even_fast,'valid').^2;
subplot(1,3,2); showIm(Eright); title('rightward energy');
subplot(1,3,3); showIm(Eleft); title('leftward energy');

% The energy responses are smooth and are selective for their
% corresponding motions.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

WARNING: You should compile the MEX version of "pointOp.c",
         found in the MEX subdirectory of matlabPyrTools, and put it in your matla
WARNING: You should compile the MEX version of "range2.c",
         found in the MEX subdirectory of matlabPyrTools, and put it in your matla
WARNING: You should compile the MEX version of "range2.c",
```

```
        found in the MEX subdirectory of matlabPyrTools, and put it in your matla
WARNING: You should compile the MEX version of "range2.c",
        found in the MEX subdirectory of matlabPyrTools, and put it in your matla
```

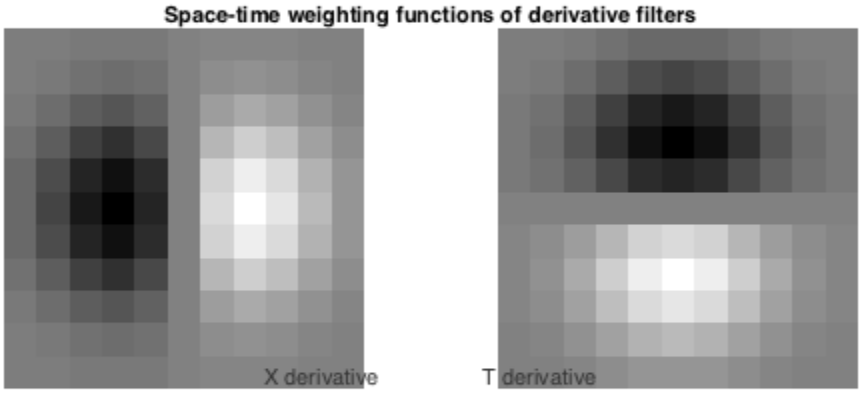


Gradient Methods for Velocity Computation

```
% As described above, motion is orientation in space-time. Computationally,
% we'd like to describe a method for estimating this orientation. This
% requires that we specify the relationship between the local velocity and
% the image intensity. The standard method of doing this is to assume that
% the image is translating (but not changing in any other way) over time:
%
%   I(x, t + Dt) = I(x - v.Dt, t)
%
% If we assume the time interval Dt  is small, we can expand both sides of
% the equation in a first-order Taylor series:
%
%  I(x, t) + Dt * dI/dt = I(x, t) - v * Dt * dI/dx
%
% After eliminating common terms, we get:
%
%  dI/dt  + v dI/dx = 0
%
% This is known as the "differential brightness constancy constraint".

% Derivatives are only properly defined for continuous signals - in order
```

```
% to compute derivatives of sampled signals, one needs to interpolate them.
% For this tutorial, we'll do our interpolating with a circularly-symmetric
% Gaussian, which has the nice property that it is separable.
X=[-5:5];  sig=2;
gFilt = exp(- (X.^2)/(2*sig^2));
dFilt = X .* gFilt;
dxFilt = gFilt' * dFilt;   % separable combinations
dtFilt = dFilt' * gFilt;
clf; showIm(dxFilt + i*dtFilt,'auto','auto',...
       'Space-time weighting functions of derivative filters');
xlabel('X derivative                T derivative');
```

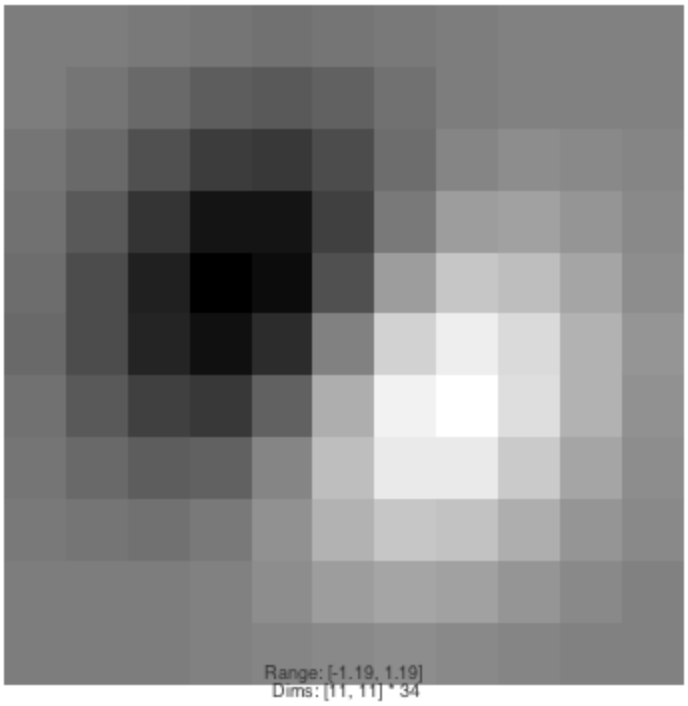
```
WARNING: You should compile the MEX version of "range2.c",
         found in the MEX subdirectory of matlabPyrTools, and put it in your matla
WARNING: You should compile the MEX version of "range2.c",
         found in the MEX subdirectory of matlabPyrTools, and put it in your matla
```



Notice that because these filters are separable, and are NOT selective for a particular direction of motion. But derivatives in two dimensions have a beautiful rotation-invariance property: the derivative in an arbitrary directions is just a linear combination of the derivatives along the two axes. For example:

```
angle = pi/6;
angleFilt = cos(angle) * dxFilt + sin(angle) * dtFilt;
figure; showIm(angleFilt);
```

```
WARNING: You should compile the MEX version of "range2.c",
         found in the MEX subdirectory of matlabPyrTools, and put it in your matla
```

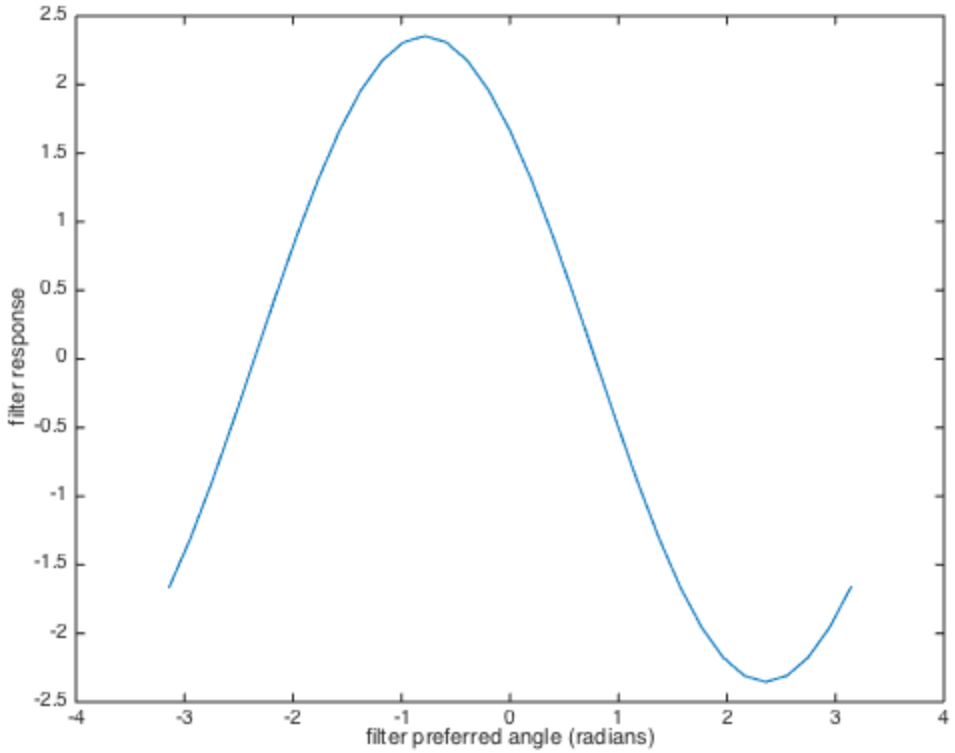


Because of this property, one way to think about computing velocity is to find the angle at which we get a maximal filter response. We first compute the filter responses in the center of the xt image:

```
sz=64;
xtImpulse = eye(sz);
res = cconv2(xtImpulse, dxFilt);
xFiltResp = res(sz/2-1,sz/2);
res = cconv2(xtImpulse, dtFilt);
tFiltResp = res(sz/2-1,sz/2);
```

...and use the rotation-invariance property to compute the responses at a bunch of angles:

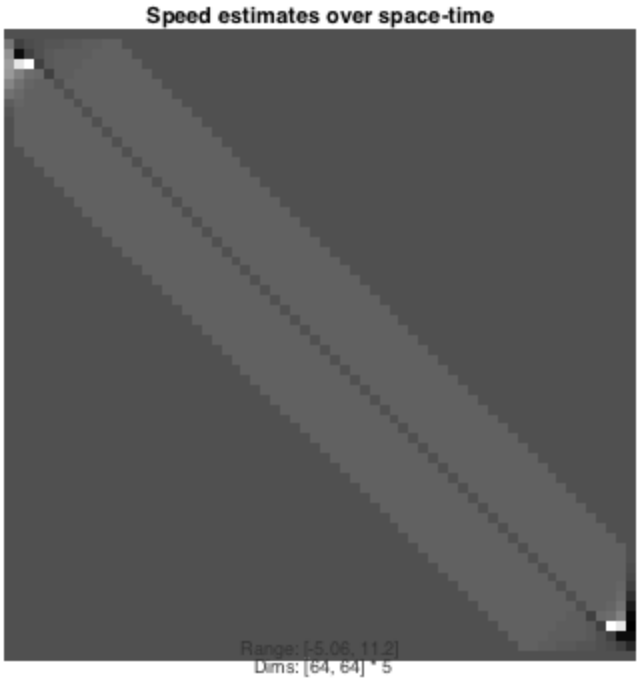
```
angles = (pi) * [-16:16]/16;
responses = cos(angles) * xFiltResp + sin(angles) * tFiltResp;
clf; plot(angles, responses);
xlabel('filter preferred angle (radians)');
ylabel('filter response');
```



The peak response is at -45 degrees, which corresponds to a rightward velocity of 1 pixel/frame. Instead of searching for the filter that gives the largest response, we can compute an estimate of velocity at each point in space and time using the gradient constraint:

```
dxImpulse = corrDn(xtImpulse,dxFilt,'dont-compute');
dtImpulse = corrDn(xtImpulse,dtFilt,'dont-compute');

% We can compute an estimate of velocity at each point in space and time using
% the gradient constraint.
vImpulse = - dtImpulse ./ dxImpulse;

% You should have gotten a "divide by zero" warning, resulting in a
% velocity field with lots of NaN (non-a-number) values, because the
% dxImpulse image has lots of zeros. We can do something a bit arbitrary to
% fix this (we'll fix it more properly in a moment):
dxImpulse = dxImpulse + (abs(dxImpulse)<1e-3);
vImpulse = - dtImpulse ./ dxImpulse;
clf; showIm(vImpulse,'auto','auto','Speed estimates over space-time');
```

```
WARNING: You should compile the MEX version of "corrDn.c",
         found in the MEX subdirectory of matlabPyrTools, and put it in your matla
Warning: Using REFLECT1 edge-handling (use MEX code for other options).
WARNING: You should compile the MEX version of "corrDn.c",
         found in the MEX subdirectory of matlabPyrTools, and put it in your matla
Warning: Using REFLECT1 edge-handling (use MEX code for other options).
WARNING: You should compile the MEX version of "range2.c",
         found in the MEX subdirectory of matlabPyrTools, and put it in your matla
```



What you see is an x-t image of VELOCITY ESTIMATES. The intensity at each point is proportional to estimated speed. The white pixels have a value of one, the correct speed. But, there is an oblique line of zeroes in between the two lines of ones, and the remainder of the image is filled with zeros. The problem is that the spatial derivative dx-impulse is zero at some locations and we cannot compute the velocity at those points.

```
% To "fix" the strip of zeroes between the lines of ones, we can combine
% the derivative measurements over a local neighborhood (by blurring the
% squared filter outputs). This comes from combining the derivative
% constraint over small neighborhoods, in a squared-error fashion:
%
%     E(v) = sum ( Ix*v + It )^2.
%
% The minimal solution for v is:
%
%     v = -sum(Ix*It) / sum(Ix*Ix)

% The local summation can be implemented using a convolution with a lowpass
% filter.  We'll choose a 5-tap binomial coefficient filter:
blurFilt = namedFilter('binom5')*namedFilter('binom5')';

dxImpulse = corrDn(xtImpulse,dxFilt,'dont-compute');
dtImpulse = corrDn(xtImpulse,dtFilt,'dont-compute');
vBlurImpulse = - (corrDn(dtImpulse*dxImpulse,blurFilt,'dont-compute')) ./ ...
                   (1e-4 + (corrDn(dxImpulse*dxImpulse,blurFilt,'dont-compute')));
vBlurImpulse(1:6,1:6)
```

```
clf; showIm(vBlurImpulse,'auto','auto','Speed estimates over space-time');

% The velocity estimate is now correct (v=1) in the vicinity of the
% moving impulse. You still get zero far from the diagonal (also near
% the edges, because we are unable to compute derivatives there).
% This is because these locations are so far away from the moving
% impulse that the neighborhood over which their velocities are
% computed do not contain any motion information. This problem occurs
% in an even more serious way in 2D.

WARNING: You should compile the MEX version of "corrDn.c",
         found in the MEX subdirectory of matlabPyrTools, and put it in your matla
Warning: Using REFLECT1 edge-handling (use MEX code for other options).
WARNING: You should compile the MEX version of "corrDn.c",
         found in the MEX subdirectory of matlabPyrTools, and put it in your matla
Warning: Using REFLECT1 edge-handling (use MEX code for other options).
WARNING: You should compile the MEX version of "corrDn.c",
         found in the MEX subdirectory of matlabPyrTools, and put it in your matla
Warning: Using REFLECT1 edge-handling (use MEX code for other options).
WARNING: You should compile the MEX version of "corrDn.c",
         found in the MEX subdirectory of matlabPyrTools, and put it in your matla
Warning: Using REFLECT1 edge-handling (use MEX code for other options).

ans =

    1.3430    1.6258    2.7509  -55.2018   -1.5453   -0.5347
    1.6912    1.9648    2.8504    8.6877   -3.7890   -1.0296
    2.1372    2.3129    2.7342    3.7090    8.2912   -6.9647
    2.2790    2.3349    2.4412    2.6036    2.9026    3.6844
    2.3137    2.2706    2.1948    2.1031    2.0095    1.9440
    2.5564    2.3582    2.0906    1.8480    1.6451    1.4909

WARNING: You should compile the MEX version of "range2.c",
         found in the MEX subdirectory of matlabPyrTools, and put it in your matla
```

How is this related to the Adelson-Bergen calculation? First we note that:

```
Ix*It = 1/4 * [ (Ix + It)^2 - (Ix - It)^2 ]
```

and thus the solution for v given above may be written:

```
v = -[ sum(Ix + It)^2 - sum(Ix - It)^2 ] / 4*sum(Ix)^2
```

Now the numerator is a difference of sum-of-squared linear measurements. Because of the properties of multi-dimensional derivatives, each of these measurements corresponds to the convolution with a single derivative filter, where the derivative is taken in a direction of +/- 45 degrees. The squaring and blurring is a form of local energy calculation. Under suitable conditions, this can be made formally equivalent to the sum of squares of even and odd filters given in Adelson and Bergen. So the numerator corresponds to rightward minus leftward energy. The denominator is the "static" energy.

The only other issue is the choice of filters. The gradient algorithm described above relies on two filters that are the temporal and spatial derivatives of a common "prefilter". The Adelson-bergen filters approximately satisfy this constraint (and one can design filters satisfying the constraint that are quite similar to the Adelson-Bergen filters). In this case, the prefilter is a separable bandpass filter.

Now we'll compute velocity on a one-dimensional image with more interesting content. We'll make a 1D fractal noise signal:

```
fract = mkFract([1 32],2);
clf; plot(fract);
xlabel('Position')
ylabel('Intensity (arbitrary units)')
```

```
Attempted to access dims(3); index out of bounds because numel(dims)=2.

Error in mkFract (line 17)
tdim = dims(3);

Error in motionTutorial (line 341)
```

And now we make it move leftward at a rate of one pixel per time-step:

```
xtFract  = zeros(16,size(fract,2));
for row = 1:size(xtFract,1)
 xtFract(row,:) = [fract(1,(33-row):32), fract(1,1:(32-row))];
end
showIm(xtFract)
```

And compute the velocity

```
vBlurFract = compute1dFlow(xtFract);
clf; showIm(vBlurFract,'auto','auto','Speed estimates over space-time');

% Now the entire space-time image of speed estimates contains values close
% to 1 because the entire pattern is moving 1 pixel per frame.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Motion in the Frequency Domain

A connection between the gradient and spatio-temporal energy mechanisms

```
% A translating one-dimensional pattern has a fourier spectrum lying on a
% line through the origin. The slope of the line corresponds to the pattern
% velocity. Take a look  at the Fourier amplitude of the moving fractal
% pattern:

fractMag = fftshift(abs(fft2(xtFract,16,16)));
clf; showIm(fractMag,[0 100],'auto','Spatiotemporal Fourier amplitude');

% The Fourier magnitude image is plotted over a range of [-pi, pi] in both
% the wx (spatial frequency) and wt (temporal frequency) directions. Most
% of the energy in this stimulus lies along an oblique line in wx-wt.
% [But why doesn't it lie EXACTLY on a line?]
```

Remember that convolution corresponds to multiplication in the frequency domain. Therefore, the Fourier Transform of the derivative images (dxFract and dtFract) corresponds to the product of the DFT's of the filter and fract. Let's look at the Fourier magnitude of the filters:

```
dtMag = fftshift(abs(fft2(dtFilt,32,32)));
dxMag = fftshift(abs(fft2(dxFilt,32,32)));
clf; showIm(dxMag + i*dtMag,'auto','auto',...
       'Spatiotemporal freq responses of t- and x-deriv filters');
```

We can write the temporal derivative as a sum of two other spatio-temporal filters oriented at +/- 45 degrees (that is, filters that are most sensitive to either rightward or leftward motion):

```
drFilt = (dxFilt+dtFilt)/2;
dlFilt = (dxFilt-dtFilt)/2;
clf; showIm(dlFilt + i*drFilt,'auto','auto',...
       'Space-time weighting functions of right- and left-selective filters');
```

Now look at the FTs of these two direction selective filters. They are rightward and leftward spatiotemporal energy filters.

```
rMag = fftshift(abs(fft2(drFilt,32,32)));
lMag = fftshift(abs(fft2(dlFilt,32,32)));
clf; showIm(rMag + i*lMag,'auto','auto',...
       'Spatiotemporal freq responses of right and left-selective filters');
```

Given this relationship between right-/left-selective filters and the x-/t-derivative filters, we can now rewrite the velocity calculation to look like an Adelson-Bergen spatio-temporal energy calculation:

velocity = (R - L)/S

where R and L are rightward and leftward "energy", and S is static energy (computed with the X derivative).

```
dlFract = corrDn(xtFract,dlFilt,'dont-compute');
drFract = corrDn(xtFract,drFilt,'dont-compute');
dxFract = corrDn(xtFract,dxFilt,'dont-compute');

lEnergy = corrDn(dlFract.*dlFract, blurFilt,'dont-compute');
rEnergy = corrDn(drFract.*drFract, blurFilt,'dont-compute');
sEnergy = corrDn(dxFract.*dxFract, blurFilt,'dont-compute');
sEnergy = sEnergy + 1e-4;  %% avoid divide-by-zero

vBlurSteFract = (lEnergy - rEnergy) ./ sEnergy;
figure;
showIm(vBlurSteFract, 'auto','auto',...
       'Speed, computed with opponent energy mechanism');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Temporal Aliasing

```
% The patterns above were moving only one pixel per frame. Now consider a
% pattern moving three pixels per frame:
speed = 3;
fract2 = mkFract([1 64],2);
xtFract2  = zeros(16,size(fract2,2));
for row = 1:size(xtFract2,1)
 xtFract2(row,:) = [fract2(1,(65-row*speed):64), fract2(1,1:(64-row*speed))];
end
clf; showIm(xtFract2,'auto','auto','Intensity over space-time');
```

Now let's try to estimate the velocity of this stimulus:

```
vBlurFract2 = compute1dFlow(xtFract2);
clf; showIm(vBlurFract2,'auto','auto','Speed estimates over space-time');
```

Look at a histogram of the estimated velocities:

```
clf; hist(vBlurFract2(:),30);
```

The speed estimates have quite a lot of variability. This is because the filters are too small for an image displacement of 2 pixels, so the algorithm is beginning to fail. To see this another way, look at the Fourier transform of the signal:

```
fract2Mag = fftshift(abs(fft2(xtFract2,32,32)));
clf; showIm(fract2Mag,[0,100],'auto','Spatiotemporal Fourier amplitude');

% You should be able to see aliased copies of the spectrum, parallel to the
% central "stripe".  Why does this happen?  How does it affect the derivative
% filters (look back at dxMag and dtMag)?
```

One way to get around the temporal aliasing problem is to use derivative filters of lower spatial frequency response. These will not "see" the aliased copies, and thus will be able to get a better velocity estimate. Alternatively, we can blur and subsample the images spatially, thus reducing the motion displacement per frame.

```
xtFract2Blur = corrDn(xtFract2,namedFilter('gauss5')','circular',[1 2]);
clf; showIm(xtFract2Blur,'auto','auto','Coarse scale intensity over space-time');
```

Note that the aliasing in the power spectrum will now have a much smaller effect on the filters because most of the energy is now back on the main diagonal:

```
xtFract2BlurMag = fftshift(abs(fft2(xtFract2Blur,32,32)));
clf; showIm(xtFract2BlurMag,[0,100],'auto','Spatiotemporal Fourier amplitude');
```

Now we can compute motion on this blurred and subsampled image. Remember that the resulting velocities must now be multiplied by 2 to get them in units of pixels/frame relative to the sampling of the original images.

```
v2BlurFract2 = 2 * compute1dFlow(xtFract2Blur);
clf; showIm(v2BlurFract2,'auto','auto','Speed estimates');
```

The histogram now shows most values near the correct speed (red line):

```
figure;
hist(v2BlurFract2(:),30);
hold on; line(speed*[1 1], [0 20], 'Color', 'red'); hold off

% We will stop here. But note that the current velocity field is computed
% at reduced resolution, and we may want higher resolution velocity fields.
% In particular, for real images, the motion is often NOT uniform
% translation. Thus we want to use the smallest filters possible to compute
% the most LOCAL velocity possible.

% One solution is to use a "coarse-to-fine" procedure which is a standard
% trick in image processing and computer vision. Get an initial estimate at
% the coarse scale using big filters (or equivalently, using blurred and
% subsampled images). Then refine that estimate at successively finer
% scales. After computing the initial coarse scale estimate of the
% velocity, "undo" motion by aligning the images according to the
% coarse-scale estimate. To undo the motion, translate or "warp" each frame
% (each scan line of the X-T diagram) back toward the center frame
% according to our current motion. Then the finer scale filters are used to
% compute a correction to the coarse-scale flow field.
```

Published with MATLAB® R2014b