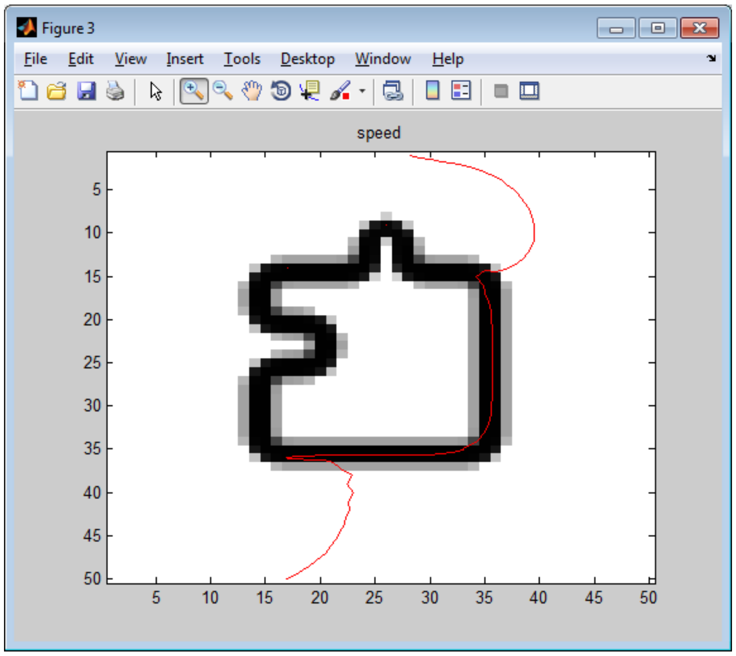
EECE 8395: Medical Image Segmentation
Vanderbilt University
Prof. Jack H. Noble

Level Set Segmentation

This demo is an example of a level set segmentation.

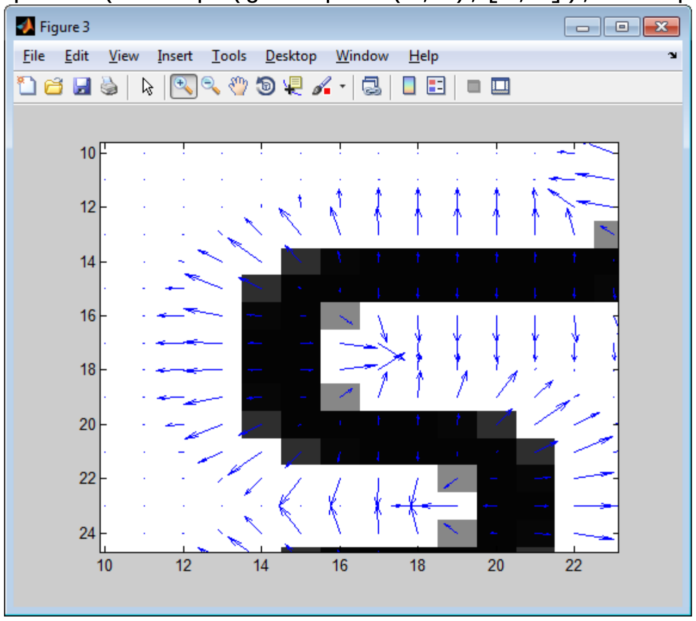
```
sigma = 1;
maxiter = 300;
mindist=2.1;


gamma=.35;


r=50; c=50; d=1;
slc = r*c;
img = zeros(r,c,d);
img(15:35,15:35) = 1;
img(21:25,10:20) = 0;
img(10:14,25:27) = 1;
figure(1); clf; colormap(gray(256));
image(255*img);
title('ground truth');


img = .5 - img;
g = fspecial('gaussian',[5,5],sigma);
imgblur = conv2(img,g,'same');


[Y,X,Z] = meshgrid(1:c,1:r,1:d);
grad = Gradient(imgblur,1:r*c*d);
ngrad = reshape(sum(grad.*grad),[r,c,d]);
speed = exp(-ngrad/(.08));
figure(2); clf; colormap(gray(256));
image(speed*1000);
title('speed');


dmap = ones(size(img));
dmap(20:30,24:32)=-1;
iter = 0;
nb = [];

while iter<maxiter
    iter = iter+1;

    figure(2);
    hold off
    image(speed*1000);
    hold on;
    contour(dmap,[0,0],'r');
      title('speed');

    drawnow;

   [dmap,nbin,nbout] = FastMarch(dmap,mindist,1,nb);
   figure(3);
    hold off;
    image(dmap*10+127);
    hold on;
    contour(dmap,[0,0],'r');
    title(['distance map iter=',num2str(iter)])
    drawnow;

      nb.q = [nbin.q(:,1:nbin.len),nbout.q(:,1:nbout.len)];
    nb.len = nbin.len+nbout.len;

      [kappa,ngrad] = Curvature (dmap,nb);

      figure(4);
    curvature = zeros(size(dmap));
    curvature(nb.q(1,(nb.q(2,1:nb.len)<=1))) = kappa(nb.q(2,1:nb.len)<=1);
    image(curvature*500+127);
      title('curvature');
    drawnow;



      speedc = -speed(nb.q(1,1:nb.len)).*(ngrad).*(kappa+gamma);
      dt = 1/max(abs(speedc(:)));

      dmap(nb.q(1,1:nb.len)) = dmap(nb.q(1,1:nb.len)) + dt*speedc;
end

[dmap,nbin,nbout] = FastMarch(dmap,mindist,1,nb);
figure(2);
hold off
image(speed*1000);
title('speed');
hold on;
contour(dmap,[0,0],'r');
figure(1);
hold on;
contour(dmap,[0,0],'r');
title('distance map');
```



We leak outside the boundary. Our speed function has to stop an expanding force. If the speed function is not airtight then we will leak out. So we need to modify our speed function

…

```
gradspeed = Gradient(speed,1:r*c*d);
hold on;
quiver(reshape(gradspeed(2,:),[r,c]),reshape(gradspeed(1,:),[r,c]))
```

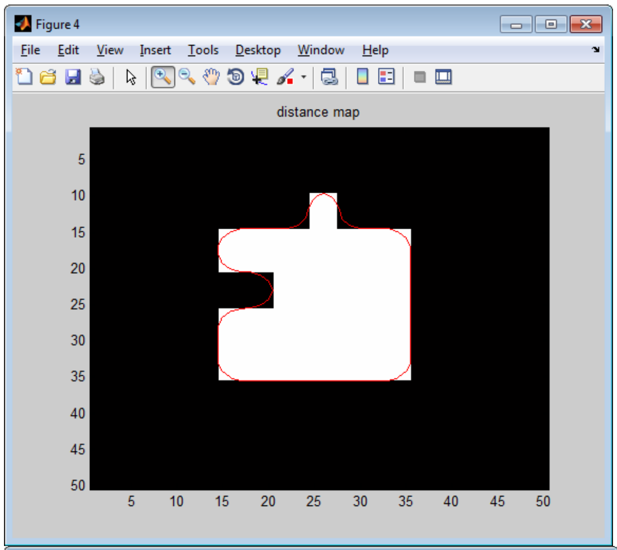


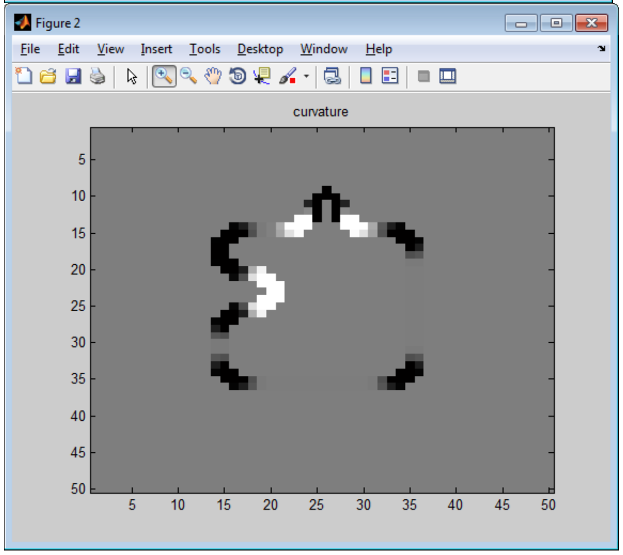
…

```
[kappa,ngrad,grad] = Curvature_div(dmap,nb);
```

…

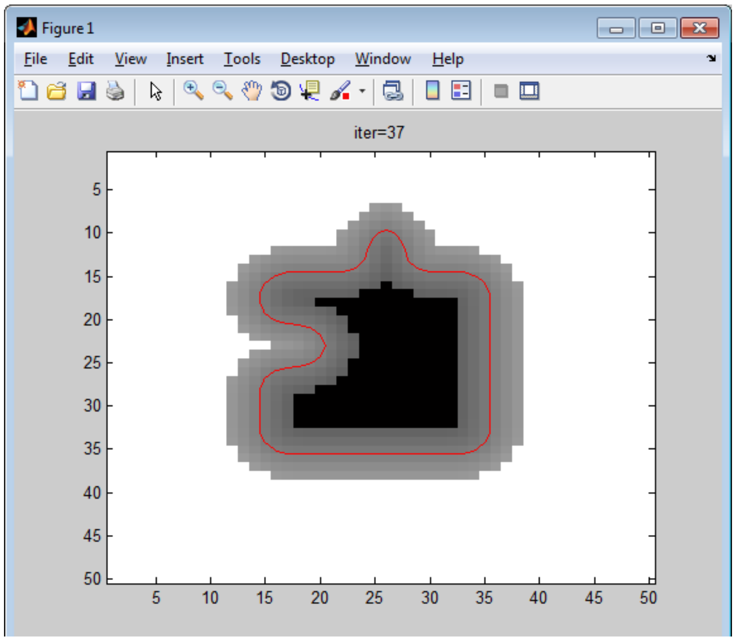
```
speedc = -speed(node).*(ngrad).*(kappa+v) +sum(grad.*gradspeed(:,node));
…
```





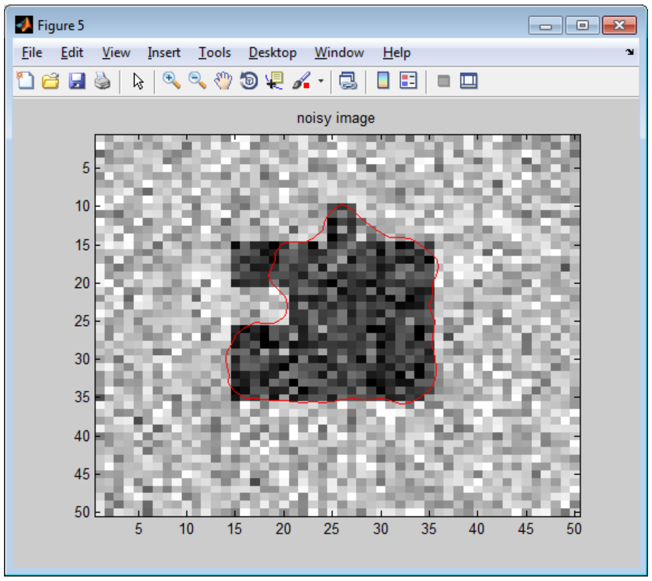
Much better. Now we need convergence criteria.

```
...
errthrsh = 0.01;
...
nbin.len = sum(nbin.q(2,1:nbin.len)<=1);
    nbout.len = sum(nbout.q(2,1:nbout.len)<=1);
    if iter>1
        err = sum(abs(-dmap(nbinold.q(1,1:nbinold.len))-
nbinold.q(2,1:nbinold.len)))+...
            sum(abs(dmap(nboutold.q(1,1:nboutold.len))-
nboutold.q(2,1:nboutold.len)));
        if err<errthrsh
            break;
        end
    end
nboutold = nbout;
nbinold = nbin;
…
```



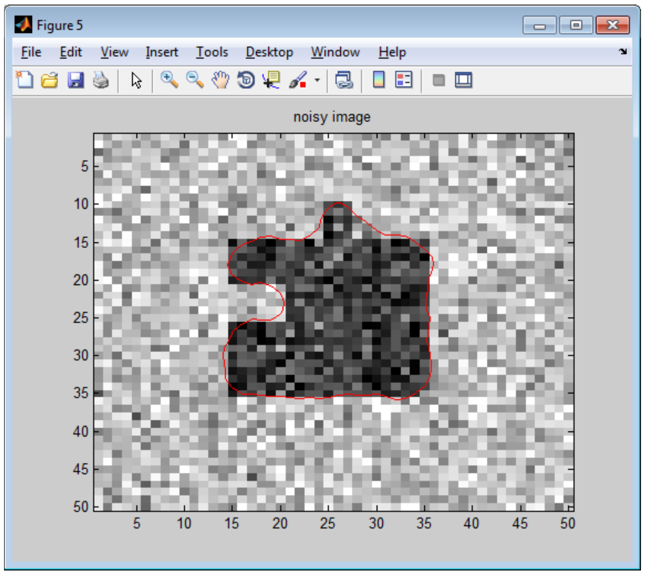
What if test image is not ideal:

```
…
noise  = 0.3;

rng('default');
img = img + noise*randn(size(img));
imgnoise = img;
figure(5); clf; colormap(gray(256));
image(127*imgnoise+127);
title('noisy image');

…
```



So we need to increase our balloon force a little bit.

```
…
gamma=.85;
…
```



Much better. How high can we go?

```
noise = 0.4;
gamma=1;
```

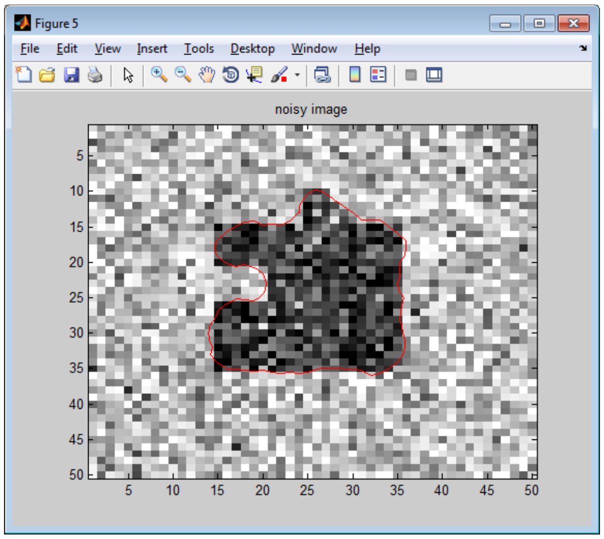


Figure 5
File Edit View Insert Tools Desktop Window Help
noisy image
5
10
15
20
25
30
35
40
45
50
5 10 15 20 25 30 35 40 45 50

```
noise = 0.8
```

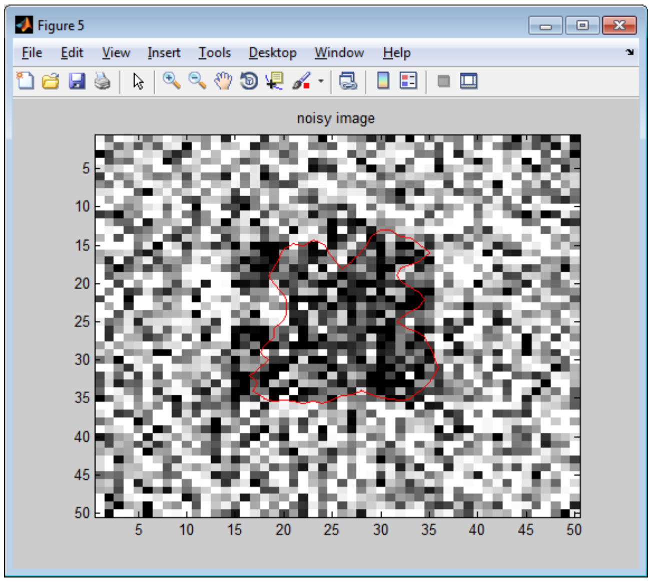
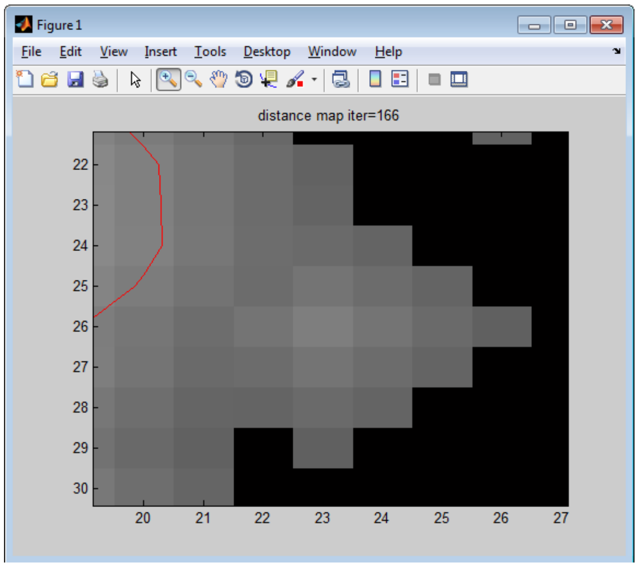


Figure 5
File Edit View Insert Tools Desktop Window Help
noisy image
5
10
15
20
25
30
35
40
45
50
5 10 15 20 25 30 35 40 45 50



At this point the result is not great but we also have something weird going on

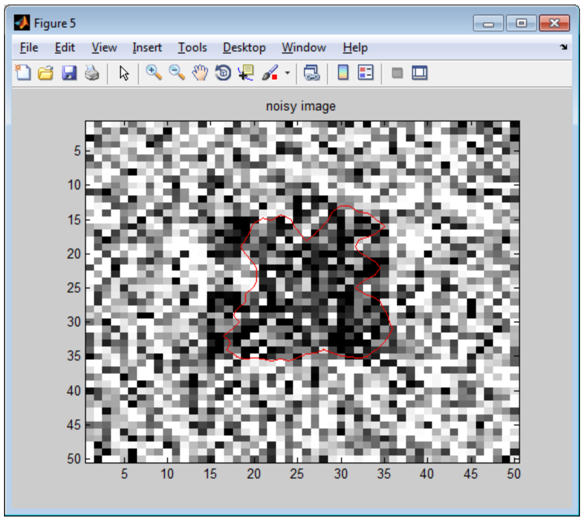
```
dmap(26,23)

ans =

   1.3986e-47
```

Inspecting our speed function:

```
speedc=-speed(node).*(ngrad).*(kappa+gamma) + sum(grad.*gradspeed(:,node));
```

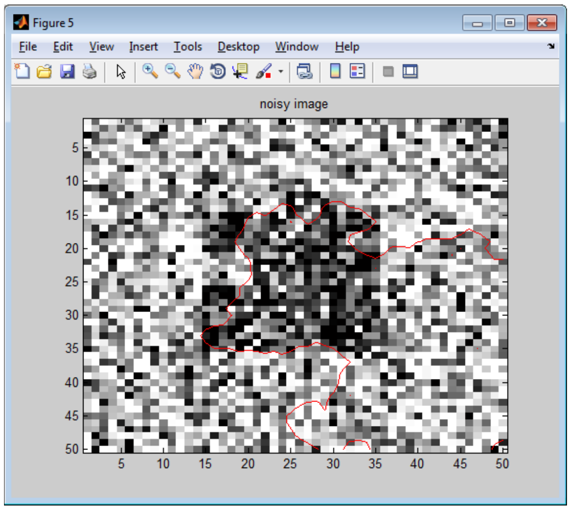
We can see that when ngrad -> 0, the speed function zeros out the curvature and gamma. This kind of makes sense – ngrad is supposed to contstrain evolution in the direction normal to the curve but since the gradient is zero we don’t have a normal direction. However we can fix this with

```
speedc=-speed(node).*(max(ngrad,0.001)).*(kappa+gamma) +
sum(grad.*gradspeed(:,node));
```



We can try to increase gamma to push the contour into the missing parts

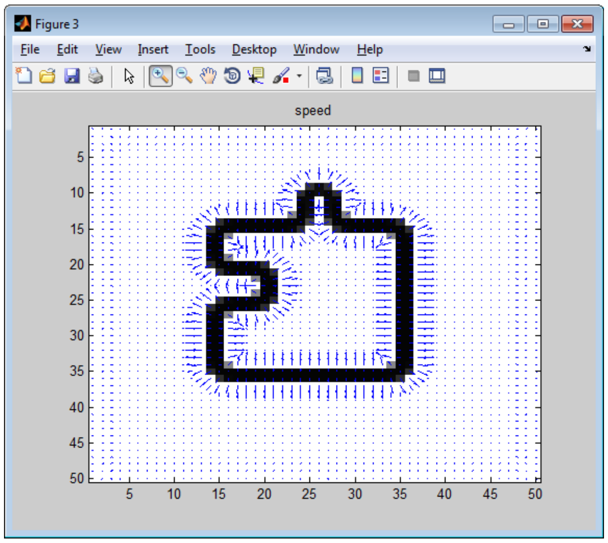
```
gamma=2;
```



But then we leak out and our gradients can't help us anymore. What if we initialize outside of the object?

```
noise = 0;
gamma=.5;
dmap(22:45,5:45)=-1;
```

We are outside the capture range. If you look at the speed function, you can see the gradients are only defined very locally around the border region. So the contour needs to be pretty close at first to get a reasonable result.



We can improve the capture range if we do a Laplacian interpolation of the gradients

First let's find the border pixels and compute the gradient. I'll use FastMarch to just find the border pixels.

```
gamma=0
[dmap,nbin,nbout] = FastMarch(imblur,1.1,1,[]);
q = [nbin.q(1,nbin.q(2,1:nbin.len)<=1),nbout.q(1,nbout.q(2,1:nbout.len)<=1)];
gradspeed = Gradient(speed,q);
```

Now we will diffuse those gradients over the image domain by solving the Laplacian equation in Matlab. This is done by defining a matrix A such that the solution z [r*c,1] of A*z=bx are gradients in the x direction and the solution z of A*z=by are the gradients in the y direction. Because the matrix A is so large [r*c,r*c], we need to use the sparse matrix representation in Matlab:

```
help sparse
```

So every row will correspond to the equation for each pixel, and the equation is defined across multiple columns. To create the sparse matrix we need vectors that contain each non-zero entry in the matrix. We have 3 types of equations: (a) Border pixels where we know the gradient. These have an equation in the matrix with one term (1) on the diagonal. (b) Then we have interior pixels that have 1+numneighbors = 5 terms for 2D images with 4-connected neighborhoods. (c) Finally we have border pixels that are treated as ghost nodes and use Neumman boundary conditions which will have 2 terms in the matrix. Thus, to define our sparse matrix we need to define at most r*c*5 terms. We'll start with vectors that define that many entries then pare them down for type (a) and type (c) pixels.

```
slc = r*c;
node = [1:slc]';
rws = [reshape(repmat(node',[5,1]),[5*slc,1])];
cols = rws + [repmat([0;-1;1;-r;r],[slc,1])];
```

```
[rws(1:10),cols(1:10)]
```

`rws` defines the row indices, `cols` defines the column indices.

```
s = repmat([1;-.25;-.25;-.25;-.25],[slc,1]);
```

`s` defines the value of each nonzero entry in the matrix. We initialize the off diagonal elements as the values for type (b) voxels but we need to change these for types (a) and (c).

```
I = zeros(slc*5,1);
```

We will use `I` to mark which rows we want to remove for type (a) and (c) pixels.

```
bx = zeros(slc,1);
by = zeros(slc,1);
```

Finally, bx and by will define the ‘b’ vector in our equation A*z=b.

```
dnode = q;
```

These are the node indices for our type (a) pixels.

```
bx(dnode) = gradspeed(1,:);
by(dnode) = gradspeed(2,:);
```

We want to mark for removal the off diagonal elements for these nodes since we have their exact value.

```
I(reshape(repmat(5*(dnode'-
1)+1,[1,4])+repmat([1:4],[length(dnode),1]),[4*length(dnode),1]))=1;
```

Now we handle the boundary conditions.

```
N = find(X(:)==1 & sum(reshape(I,[5,slc]))'==0);
I((N-1)*5+2)=1;
I((N-1)*5+4)=1;
I((N-1)*5+5)=1;
s((N-1)*5+3)=-1;

N = find(X(:)==r & sum(reshape(I,[5,slc]))'==0);
I((N-1)*5+3)=1;
I((N-1)*5+4)=1;
I((N-1)*5+5)=1;
s((N-1)*5+2)=-1;

N = find(Y(:)==1 & sum(reshape(I,[5,slc]))'==0);
I((N-1)*5+2)=1;
I((N-1)*5+3)=1;
I((N-1)*5+4)=1;
s((N-1)*5+5)=-1;

N = find(Y(:)==c & sum(reshape(I,[5,slc]))'==0);
I((N-1)*5+2)=1;
```

```
I((N-1)*5+3)=1;
I((N-1)*5+5)=1;
s((N-1)*5+4)=-1;
```

And we remove entries that are marked with I(:)=1

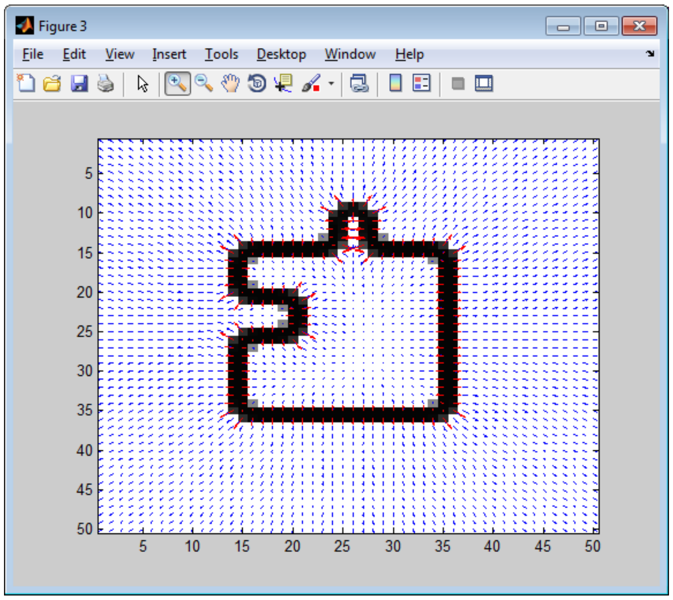
```
rws = rws(~I(:));
cols = cols(~I(:));
s = s(~I(:));
```

Now we construct sparse matrix A and solve A*x=bx and A*y=by

```
A = sparse(rws,cols,s,slc,slc);
x = A\bx;
y = A\by;
```

Let's compare our estimated gradient vector field with the original input.

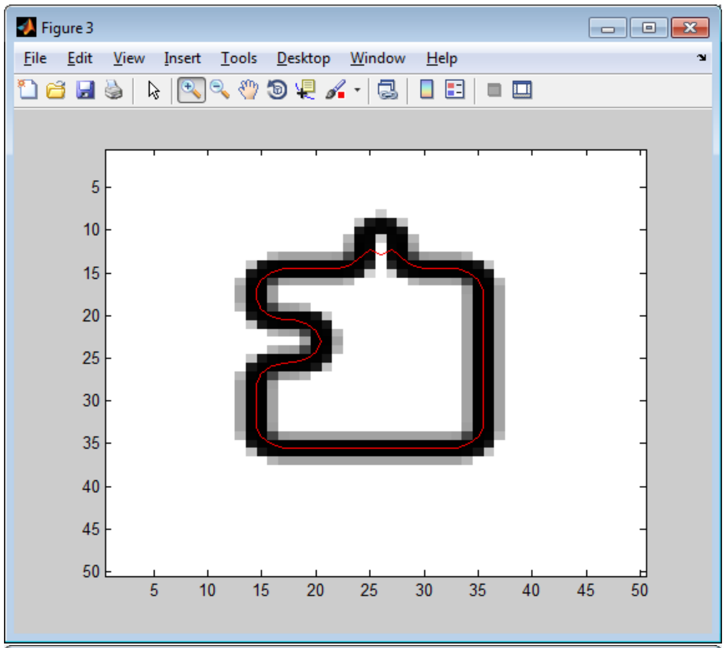
```
ngradspeed = [x';y';zeros(1,length(x))];

hold on;
quiver(reshape(ngradspeed(2,:),[r,c]),reshape(ngradspeed(1,:),[r,c]))

hold on;
quiver(floor((q(1,:)-1)/r)+1,mod(q(1,:)-
1,r)+1,(gradspeed(2,:)),(gradspeed(1,:)),'r')
```

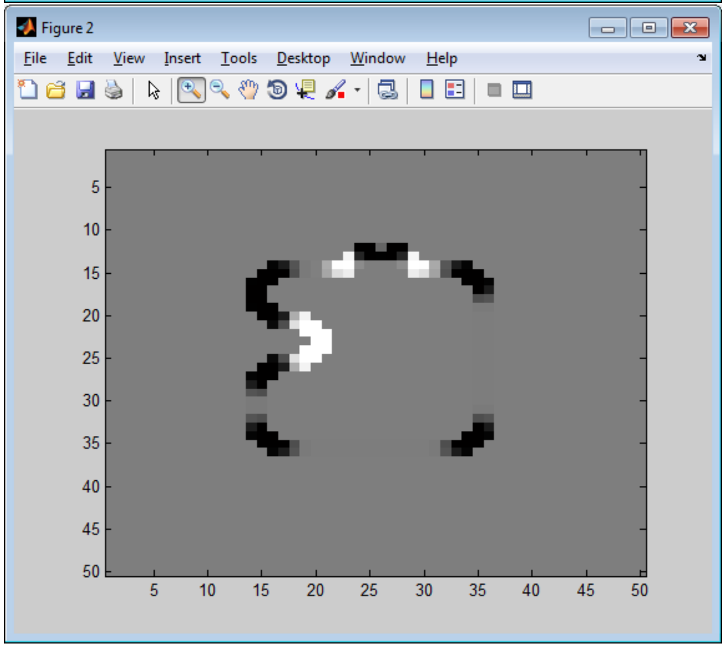


OK so how does it work with our level set.

```
gradspeed = ngradspeed;

dmap(25:45,5:45)=-1;
gamma=.1;
```

That generally works very well, but what’s up with the feature at the top





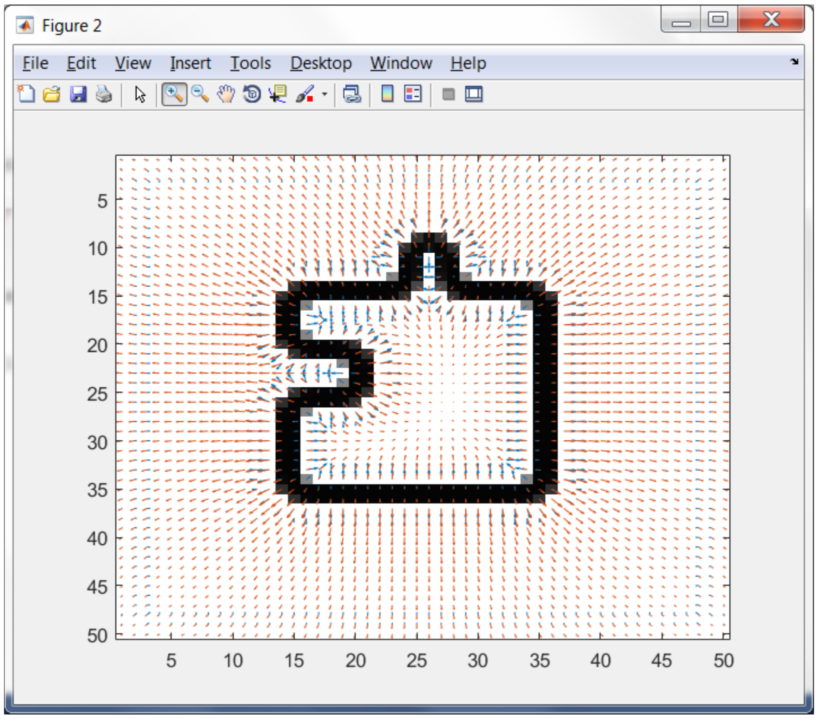
For single voxel-thin structures, finite difference approximation of mean curvature is error prone. A more accurate way to measure curvature is to use the hessian matrix H and gradient F with the equation ((Gradient(F)’*H*Gradient(F) - |Gradient(F)|^2*Trace(H))/(2*|Gradient(F)|^3),

```
nbspeed.q = [nbin.q(:,1:nbin.len),nbout.q(:,1:nbout.len)];
    nbspeed.len = size(nbspeed.q,2);
```

```
[kappa,ngrad,grad] = Curvature2(dmap,nbspeed);
node = nbspeed.q(1,1:nbspeed.len) + (nbspeed.q(2,1:nbspeed.len)-1)*r +
(nbspeed.q(3,1:nbspeed.len)-1)*slc;
speedc=-speed(node).*(max(ngrad,.001)).*(kappa+v) +
sum(grad.*gradspeed(:,node));
dt = 0.5/max(abs(speedc(:)));
dmap(node) = dmap(node) + dt*speedc;
…
curvature(node) = kappa;
```

One more thing we can do to get robustness to noise. Notice I had to choose which gradient vectors to use. In very noisy situations I could make bad choices. To get robustness to noise, lets implement a smoothing Laplacian interpolation of gradients across the whole image. We just want to trust the high magnitude gradients and interpolate over the homogeneous (low magnitude) regions. We use parameter 'mu' to control how much smoothing we want.

```
mu = .9;
…
ngradspeed = GVF(gradspeed,mu,[r,c,d]);

hold on;
quiver(reshape(ngradspeed(2,:),[r,c]),reshape(ngradspeed(1,:),[r,c]))
```

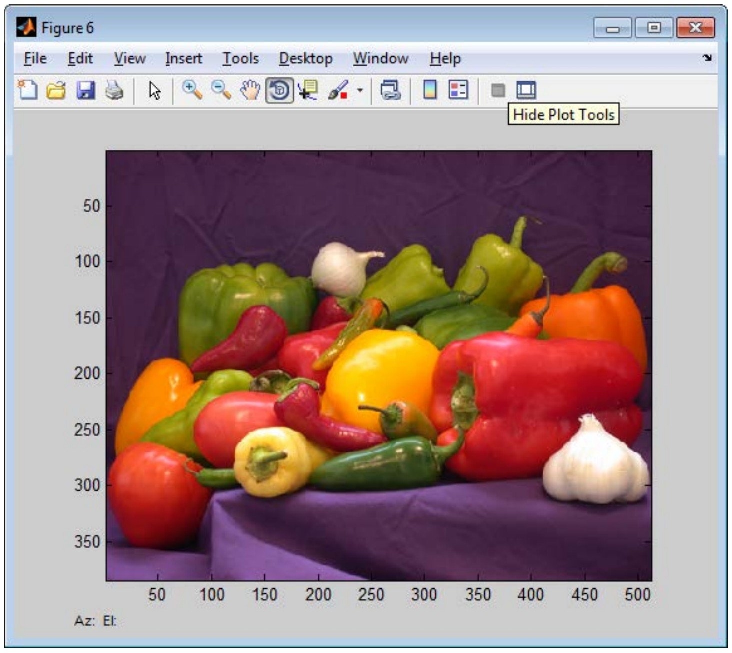


```
gradspeed = ngradspeed;
```

That gives us too much smoothing on our gradient vectors. How about mu=.5? and if we Add noise (noise=.3, mu=.75)? And how about noise=.6? Let's improve our starting position a bit as well dmap(23:40,10:40)=1

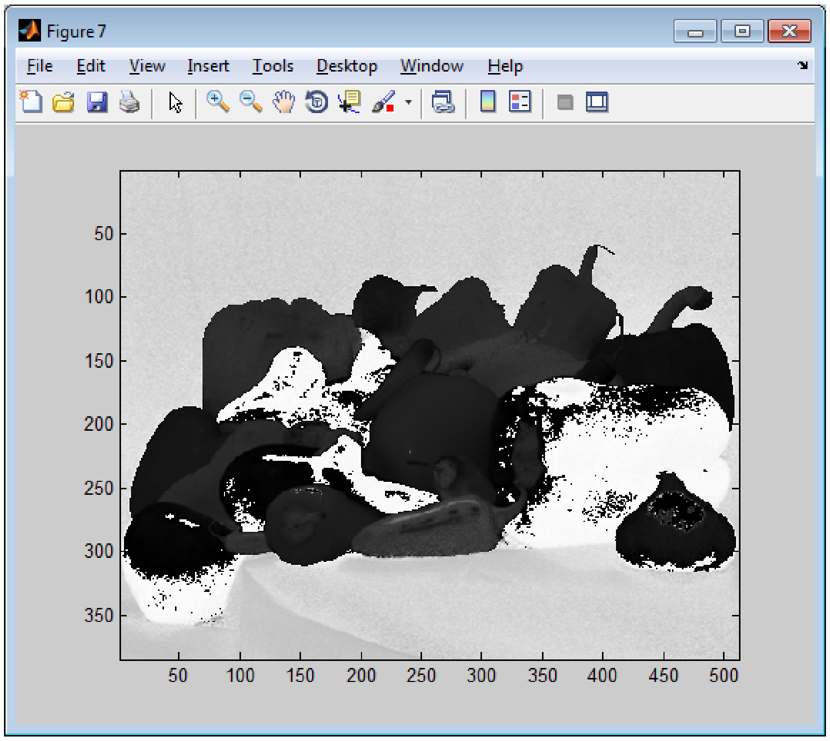
OK, so let's try to segment a real image

```
function dmap = LevelSet(img,dmap, sigma, errthrsh, maxiter, mu, gamma)
…
[r,c,d] = size(img)
..

C = imread('peppers.png');
figure(6);
image(C)
```



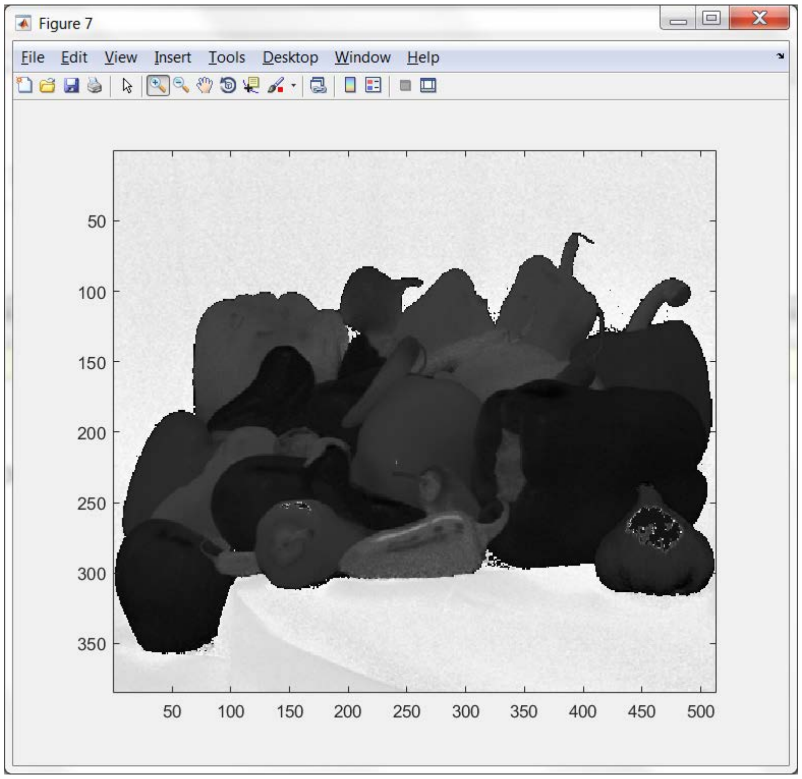
First we convert to an HSV representation and use the hue channel to find a threshold that separates the peppers from the purple background.

```
HSV = rgb2hsv(C);
H = HSV(:,:,1);
figure(7);colormap(gray(256))
image(H*255)
```



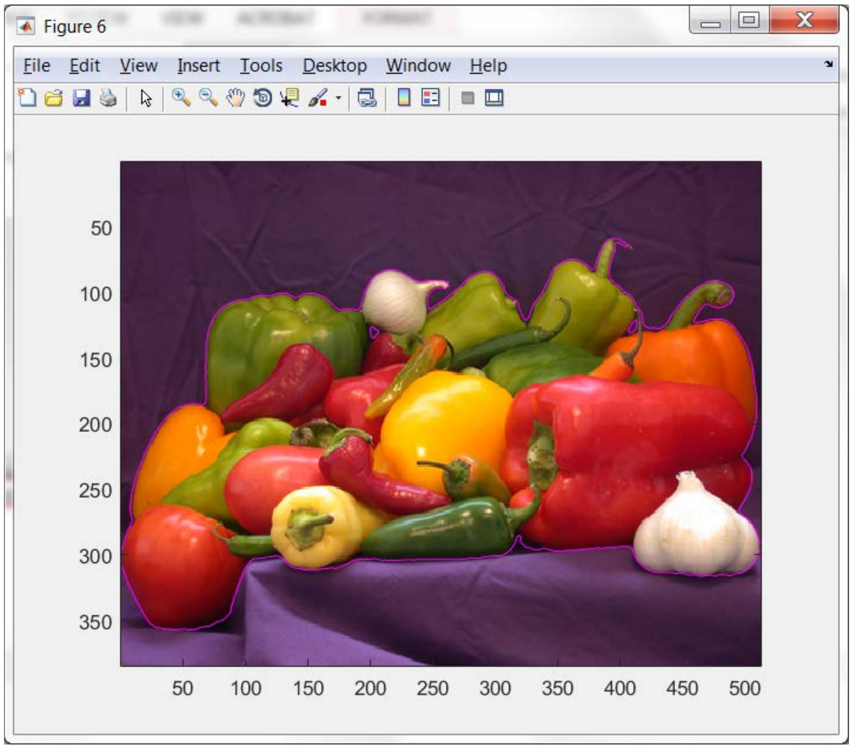
We can find the hue of the blanked by averaging over a region in the top left corner of the image:

```
purpl = mean(mean(HSV(30:60,60:90,1)))
H = H + (1-purpl)/2;
H(H(:)>1) = H(H(:)>1)-1;
image(H*255)
```



We can initialize our distance map by thresholding this image then run it through our level set to clean up all the noise:

```
dmap = 0.5-(H<.5);
dmap = LevelSet(H,dmap,2,.1,30,.9,0);

figure(6);
hold on;
contour(dmap,[0,0], 'm');
```



Level sets give us a nice smooth result!