Geodesic Active Contours

Name: Edward Rusu
RIN: 660878003

28 April 2016

Abstract

In this report we examine Geodesic Active Contours for Image Segmentation. We derive an energy functional that uses internal and external forces to evolve the curve to the boundary of an object within the image. We show the intrinsic equivalence of this functional and derive its gradient flow. We outline the numerical implementation of the partial differential equation using techniques for parabolic and hyperbolic flows. We implement our algorithm for three different images and discuss the results.

1 Model

The goal of image segmentation is to capture the boundaries of objects within an image domain. For our purposes, we consider an image I to be a function from a two-dimensional domain to luminosity

$$I : \Omega \to \mathbb{L}, \tag{1}$$

where $\mathbb{L}$ is a subset of positive integers from 0 to 255 (for black and white images). With this model, the edges of an image are large jumps (discontinuities) in luminosity. In this way, we can mathematically structure the process of image segmentation.

Consider a domain of interest $D \subset \Omega$ with boundary ∂D such that ∂D is an edge in the image–a discontinuity of our image function I. Now, let C be a close curve in Ω. Our fundamental idea is to evolve of a curve C through internal and external forces so that $C = \partial D$ is at equilibrium. To this end, we consider Geodesic Active Contours by Caselles, Kimmel, and Sapiro [1].

Classical energy based snakes balance smoothness with curve movement. A popular energy functional is

$$E(C) = \int_0^1 \alpha |C'(q)|^2 + \beta |C''(q)|^2 - \lambda |\nabla I(C(q))| dq, \tag{2}$$

but the $|C''(q)|$ term has been found to require too much smoothness, so β is often set to 0:

$$E(C) = \int_0^1 \alpha |C'(q)|^2 - \lambda |\nabla I(C(q))| dq. \tag{3}$$

The second term $|\nabla I|$ acts to detect the edge according to jumps in the image. We will implement an edge detector g. Let $g : [0, \infty) \to \mathbb{R}^+$ be strictly decreasing such that $g(r) \to 0$ as $r \to \infty$. Thus, $|\nabla I|$ can be replaced by $g^2(|\nabla I|)$, resulting in the functional

$$E(C) = \int_0^1 \alpha |C'(q)|^2 - \lambda g^2(|\nabla I(C(q))|) dq. \tag{4}$$

For our implementation, we will consider an edge detector

$$g = \frac{1}{1 + |\nabla I|^2} \tag{5}$$

In the next section, we describe how to use gradient flow to minimize this functional. However, we will ultimately develop and implement an intrinsic functional that does not change under curve parameterization.

2 Methodology

2.1 Variational Derivative

Consider the energy in Equation (). We can remove the parameter α by noting that only the ratio λ/α matters. With some abusive notation, we have the following functional

$$J(C) = \frac{1}{2}\int_a^b \left|\frac{\partial C}{\partial q}(t,q)\right|^2 dq + \frac{\lambda}{2}\int_a^b g^2(|\nabla I(C(t,q))|)dq. \tag{6}$$

We derive the first variational principal below.

$$\begin{aligned}
\frac{\partial J}{\partial t} &= \frac{1}{2}\int_a^b \frac{\partial}{\partial t}\Big[C_{1q}^2 + C_{2q}^2\Big] + \frac{\lambda}{2}\frac{\partial}{\partial t}g^2(|\nabla I(C(t,q))|)dq & (7)\\
&= \int_a^b C_{1q}C_{1qq} + C_{2q}C_{2qq} + \lambda g\frac{\partial g}{\partial t}dq & (8)\\
&= \int_a^b (C_{1q}^2 + C_{2q}^2)^{(1/2)}\langle T, C_{qt}\rangle + \lambda g\langle \nabla g, C_t\rangle dq & (9)\\
&= -\int_a^b \left\langle \frac{\partial}{\partial q}\Big[(C_{1q}^2 + C_{2q}^2)^{(1/2)}T\Big], C_t\right\rangle + \langle \lambda g\nabla g, C_t\rangle dq & (10)\\
\frac{\partial J}{\partial t} &= \int_a^b \left\langle -\frac{\partial}{\partial q}\Big[(C_{1q}^2 + C_{2q}^2)^{(1/2)}T\Big] + \lambda g\nabla g, C_t\right\rangle & (11)
\end{aligned}$$

The expression on the left inside the inner product is that Gáteaux derivative.

$$\begin{aligned}
\frac{\delta J}{\delta C} &= -\frac{\partial}{\partial q}\Big[(C_{1q}^2 + C_{2q}^2)^{(1/2)}T\Big] + \lambda g\nabla g & (12)\\
&= -\frac{\partial}{\partial q}(C_{1q}^2 + C_{2q}^2)^{(1/2)}T - |C_q|T_q + \lambda g\nabla g & (13)\\
&= -|C_q|^{-1}(C_{1q}C_{1qq} + C_{2q}C_{2qq})T + |C_q|T_q + \lambda g\nabla g & (14)\\
&= -\langle T, C_{qq}\rangle T - |C_q|T_q + \lambda g\nabla g & (15)\\
&= -\langle T, C_{qq}\rangle T - |C_q|\kappa N + \langle g\nabla g, T\rangle T + \langle \lambda g\nabla g, N\rangle N & (16)\\
\frac{\delta J}{\delta C} &= -\Big[\kappa|C_q| - \langle g\nabla g, N\rangle\Big]N + \Big[\langle \lambda g\nabla g, T\rangle - \langle T, C_{qq}\rangle\Big]T & (17)
\end{aligned}$$

Therefore, the gradient descent is

$$\frac{\partial C}{\partial t} = \Big[\kappa|C_q| - \langle g\nabla g, N\rangle\Big]N - \Big[\langle \lambda g\nabla g, T\rangle - \langle T, C_{qq}\rangle\Big]T \tag{18}$$

2.2 Geodesic Active Contours

Energy methods like Equation (6) are intuitive and can be worked out with well-developed procedures. Unfortunately, this functional can take arbitrary form under different parameterizations of the curve C. To get around this, we look to the work of Caselles et. al., who claim that minimizing Equation (4) is the same as minimizing the intrinsic problem

$$L(C) = \int_0^1 g(|\nabla I(C(q))|)|C'(q)|dq. \tag{19}$$

The Gáteaux derivative here is derived very similarly as above, but this time with an arclength parameterization.

$$\frac{\partial L}{\partial t} = \int_0^1 \frac{\partial}{\partial t} g(C(t,q))\Big(C_{1q}^2 + C_{2q}^2\Big)^{(1/2)} dq \tag{20}$$

$$= \int_0^1 \langle \nabla g, C_t \rangle |C_q| + g|C_q|^{-1}(C_{1q}C_{1qt} + C_{2q}C_{2qt}) dq \tag{21}$$

$$= \int_0^1 \langle \nabla g, C_t \rangle |C_q| + g\langle T, C_{qt} \rangle dq \tag{22}$$

$$= \int_0^1 \langle \nabla g, C_t \rangle |C_q| - \Big\langle \frac{\partial}{\partial q}(gT), C_t \Big\rangle dq \tag{23}$$

$$= \int_0^1 \langle \nabla g, C_t \rangle |C_q| - \Big\langle \langle \nabla g, C_q \rangle T + gT_q, C_t \Big\rangle dq \tag{24}$$

$$= \int_0^1 \langle \nabla g, C_t \rangle |C_q| - \langle \nabla g, C_q \rangle \langle T, C_t \rangle - g\langle T_q, C_t \rangle dq \tag{25}$$

$$= \int_0^1 \langle \nabla g, C_t \rangle |C_q| - \langle \nabla g, C_s \rangle \langle T, C_t \rangle |C_q| - g\langle T_s, C_t \rangle |C_q| dq \tag{26}$$

$$= \int_0^{L(C(t,\cdot))} \langle \nabla g, C_t(t,s) \rangle - \langle \nabla g, C_s(t,s) \rangle \langle T, C_t \rangle - g\langle T_s, C_t \rangle ds \tag{27}$$

$$= \int_0^{L(C(t,\cdot))} \Big\langle \nabla g - \langle \nabla g, T \rangle T - gT_s, C_t \Big\rangle ds \tag{28}$$

$$\tag{29}$$

As stated above, the Gáteaux derivative is just the expression in the left of the inner product.

$$\frac{\delta L}{\delta C} = \nabla g - \langle \nabla g, T \rangle T - gT_s \tag{30}$$

$$= \nabla g - \langle \nabla g, T \rangle T - g\kappa N \tag{31}$$

$$= \langle \nabla g, N \rangle N - g\kappa N \tag{32}$$

$$\frac{\delta L}{\delta C} = [\langle \nabla g, N \rangle - g\kappa] N \tag{33}$$

Therefore, the gradient flow is in the normal direction, accordingly

$$\frac{\partial C}{\partial t} = -[\langle \nabla g, N \rangle - g\kappa] N. \tag{34}$$

This gradient flow is directly derived from the functional (19). However, we will append the flow with a parameter α so that

$$\frac{\partial C}{\partial t} = -[\langle \nabla g, N \rangle - g(\kappa + \alpha)] N, \qquad \kappa + \alpha > 0. \tag{35}$$

This will speed up the flow (which is very slow indeed) and will enforce correct gradient direction in nonconvex objects.

2.3 Level Set Representation

At this point, we have a gradient flow that depends on the curve C. The methods we have described here will not work when the curve becomes disjoint. Instead, we will implement a level set representation. Let $u = u(t, C(t,q))$, and let C be some level set of this function. Let us rewrite Equation (35) as

$$C_t = FN. \tag{36}$$

Now consider the time-derivative of u.

$$u_t = -\nabla u \cdot C_t \tag{37}$$

$$u_t = -\nabla u \cdot FN \tag{38}$$

$$u_t = F||\nabla u|| \tag{39}$$

Thus, our gradient flow (35) for a curve C can be written as the gradient flow of a function u

$$u_t = g||\nabla u|| div\bigg(\frac{\nabla u}{||\nabla u||}\bigg) + \alpha g||\nabla u|| + \nabla g \cdot \nabla u. \tag{40}$$

Using the level set representation makes it easier to deal with multiple curves but at great computational expense. Previously, our representation required the iteration of a single curve, which is one-dimensional. Now, however, we iterate over a two-dimensional function and extract a single level set of this function.

We make one final comment before considering the numerical implementation of Equation (40). We want to enforce the constraint

$$||\nabla u|| = 1. \tag{41}$$

Gradient flow does not guarantee this property, so we reinitialize our function via the iteration

$$\frac{\partial \phi}{\partial t} + sign(\phi)(|\nabla \phi| - 1) = 0. \tag{42}$$

In our algorithm, we will run several iterations of gradient flow, run several iterations of reinitialization, and repeat.

2.3.1 Numerical Implementation of Geodesic Active Contours

Consider again our gradient flow

$$u_t = g||\nabla u|| div\bigg(\frac{\nabla u}{||\nabla u||}\bigg) + \alpha g||\nabla u|| + \nabla g \cdot \nabla u. \tag{43}$$

We systematically discuss the the numerical implementation of each term on the RHS.

First, consider the parabolic effect

$$\frac{\partial u}{\partial t} = g||\nabla u|| div\bigg(\frac{\nabla u}{|\nabla u|}\bigg). \tag{44}$$

We discretize this with second order central differencing

$$u_{ij}^{n+1} = u_{ij}^n + \Delta t\Bigg(g_{ij}\Big((\Delta_x^c u_{ij}^n)^2 + (\Delta_y^c u_{ij}^n)^2\Big)^{(1/2)} \times \tag{45}$$

$$\Bigg(\Delta_x^+\Big(\frac{\Delta_x^- u_{ij}}{(\Delta_x^c u_{ij}^n)^2 + (\Delta_y^c u_{ij}^n)^2)^{(1/2)}}\Big) + \Delta_y^+\Big(\frac{\Delta_y^- u_{ij}}{(\Delta_x^c u_{ij}^n)^2 + (\Delta_y^c u_{ij}^n)^2)^{(1/2)}}\Big)\Bigg)\Bigg). \tag{46}$$

For convenience, we will call Equation (46) K_{ij}^n. Note that K_{ij}^n has a possible divide by zero, so we add a small ϵ parameter to ensure numerical stability.

The second term is hyperbolic, which we discretize with upwinding. To motivate this idea, consider the equation

$$\frac{\partial u}{\partial t} = ||\nabla u||. \tag{47}$$

For left (right) traveling characteristics in the xt-plane, we want to use a positive (negative) differencing. For the above equation, we accomplish that with the following discretization

$$u_{ij}^{n+1} = u_{ij}^n + \Delta t \nabla^+ u_{ij}^n, \tag{48}$$

where

$$\nabla^+ u_{ij} = \Big(\max(\Delta_x^+ u_{ij}, 0)^2 + \min(\Delta_x^- u_{ij}, 0)^2 + \max(\Delta_y^+ u_{ij}, 0)^2 + \min(\Delta_y^- u_{ij}, 0)^2\Big)^{(1/2)}, \tag{49}$$

and ∇^- is the same but with switched signs. With this operator, we can discretize the second term in our gradient flow

$$u_{ij}^{n+1} = u_{ij}^n + \alpha \Delta t\Big(\max(g_{ij}, 0)\nabla^+ + \min(g_{ij}, 0)\nabla^-\Big)u_{ij}^n. \tag{50}$$

Thus, the sign of g determines the direction of information flow. However, for our use of the edge detector (5), g is always positive, so our numerical implementation only considers the ∇^+ operator.

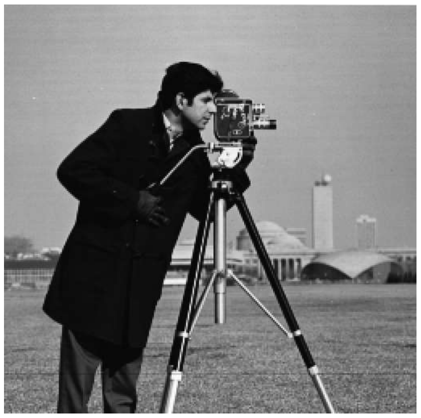
Figure 1: Clean image of cameraman.

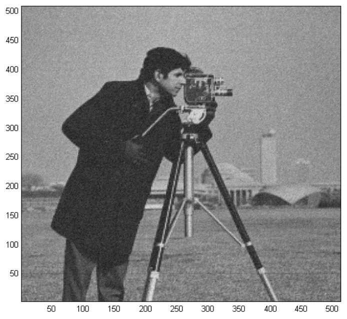

Figure 2: Noisy image of Cameraman.

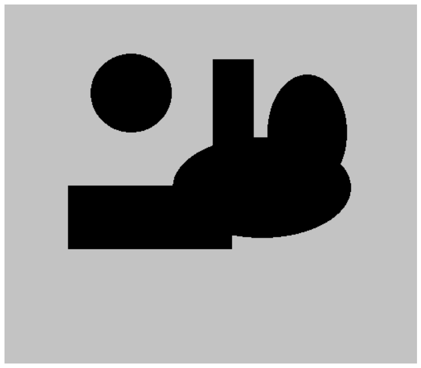
Figure 3: Binary image of shapes.

Finally, consider the final term, which is also hyperbolic,

$$\frac{\partial u}{\partial t} = \nabla u \cdot \nabla g. \tag{51}$$

Although g is always positive, ∇g is not, so we will have to consider its sign in order to choose the correct upwinding. However, unlike u, g does not change over time. We can determine its gradient beforehand and know the direction of the flow for all time. Our discretization here will be like

$$u_{ij}^{n+1} = u_{ij}^n + \Delta t\Big(\max(g_{ij}^x, 0)\Delta_x^+ u_{ij}^n + \min(g_{ij}^x, 0)\Delta_x^- u_{ij}^n + \max(g_{ij}^y, 0)\Delta_y^+ u_{ij}^n + \min(g_{ij}^y, 0)\Delta_y^- u_{ij}^n \Big) \tag{52}$$

Piecing together the discretizations for each part of the equation, we arrive at the following numerical algorithm

$$\begin{aligned} u_{ij}^{n+1} = u_{ij}^n + \Delta t\Big(& g_{ij} K_{ij}^n \Big((\Delta_x^c u_{ij}^n)^2 + (\Delta_y^c u_{ij}^n)^2 \Big)^{(1/2)} + \\ & + \alpha \Big(\max(g_{ij}, 0)\nabla^+ + \min(g_{ij}, 0)\nabla^- \Big) u_{ij}^n + \\ & + \max(g_{ij}^x, 0)\Delta_x^+ u_{ij}^n + \min(g_{ij}^x, 0)\Delta_x^- u_{ij}^n \\ & + \max(g_{ij}^y, 0)\Delta_y^+ u_{ij}^n + \min(g_{ij}^y, 0)\Delta_y^- u_{ij}^n \Big) \end{aligned} \tag{53}$$

Finally, we briefly outline the pseudocode for our iteration scheme.

1. Choose initial parameters
2. Iterate u for several iterations according to Equation (53)
3. Reinitialize with Equation (42)
4. Repeat steps 2 and 3 until convergence

3 Results

We will consider Geodesic Active Contours for the images found in Figures (1) - (3). First, let us consider the clean image of the cameraman with a large initial curve shown in Figure (4). With parameters $\alpha = 4$, $\Delta t = 0.01$, ten steps of gradient flow followed by ten steps of reinitialization, we obtain the results found in Figure (5) after 1000 repeat steps.

Now consider the same cameraman image with an initial curve *inside* the object of interest (namely, the cameraman) as shown in Figure (6). With parameters $\alpha = -10$, $\Delta t = 0.01$, ten steps of gradient flow followed by ten steps of reinitialization, we obtain the results found in Figure (7) after 1400 repeat steps.

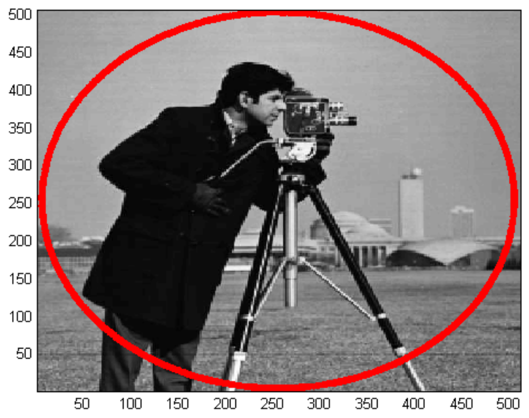

Figure 4: Clean image and outer initial curve.

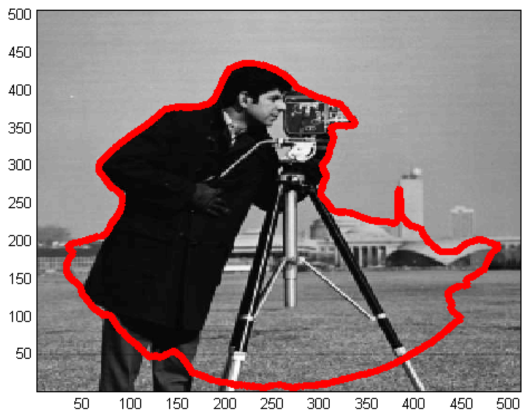

Figure 5: Curve after 1000 iterations.

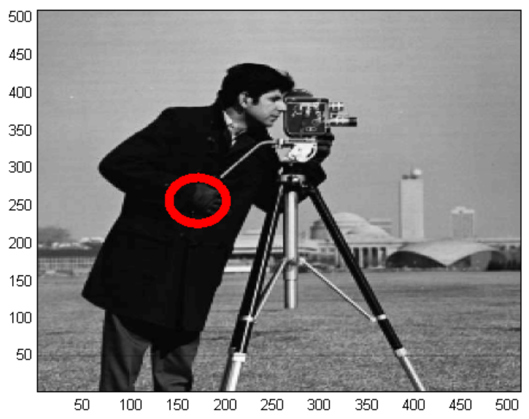

Figure 6: Clean image and inner initial curve.

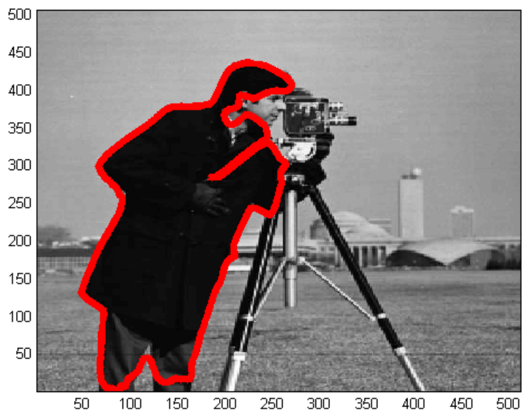

Figure 7: Curve after 1400 iterations.

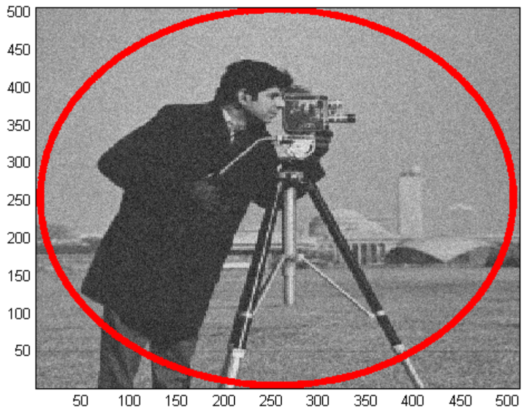

Figure 8: Noisy image and initial curve.

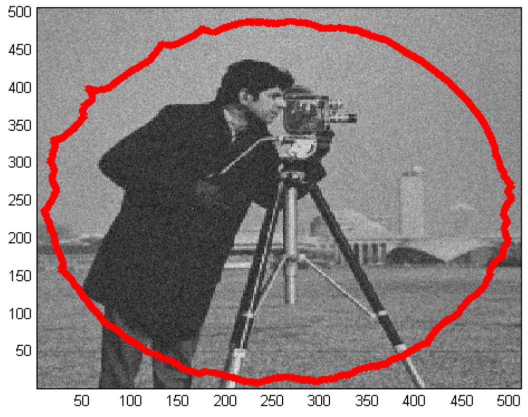

Figure 9: Curve after 1400 iterations.

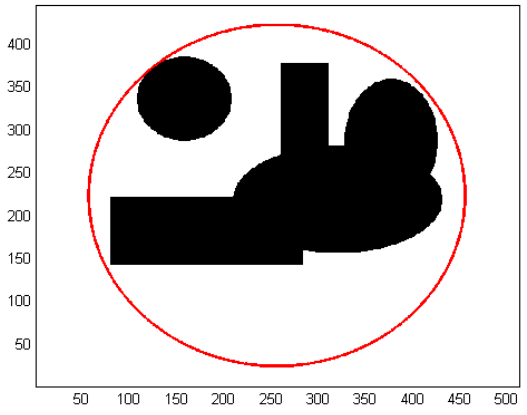

Figure 10: Binary Image and Initial Curve.

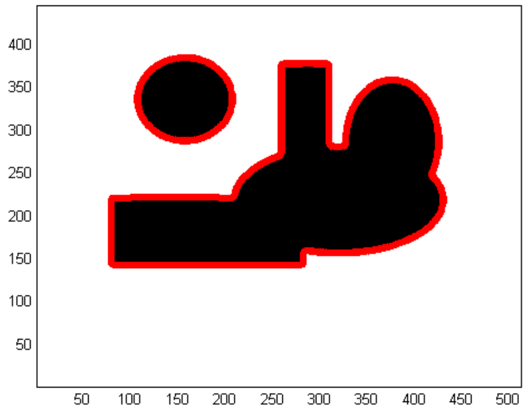

Figure 11: Curve after 600 iterations.

Consider the noisy image with initial curve in Figure (8). With parameters $\alpha = 20$, $\Delta t = 0.01$, ten steps of gradient flow followed by ten steps of reinitialization, we obtain the results found in Figure (9) after 1400 repeat steps.

Lastly, consider the binary image with initial curve in Figure (10). With parameters $\alpha = 4$, $\Delta t = 0.01$, ten steps of gradient flow followed by ten steps of reinitialization, we obtain the results found in Figure (11) after 600 repeat steps.

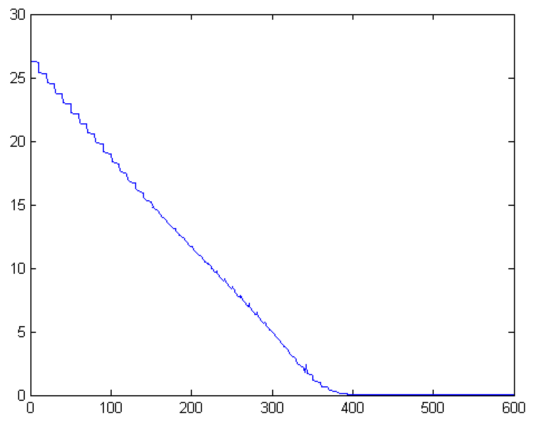

Figure 12: Error for first 60 repeats

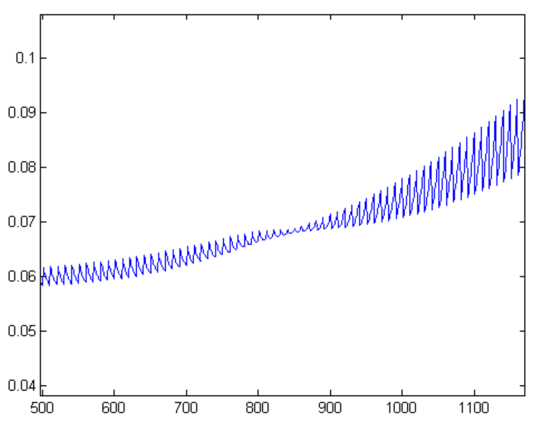

Figure 13: Skewing of "the error-saw."

4 Observation and Conclusions

Now let's chat about our results! Take a look at Figure (5); this is the clean cameraman image with an outer curve. The curve easily passes through the sky to wrap around the upper part of the man's outline, including his head, arm, and the camera. On the left and right, parts of the curve gets stuck on some objects in the background. The bottom right shows the poorest results, where the curve cannot navigate through the grass. We propose that the noisiness of the grass causes the curve to significantly slow down in that region. This is exactly what we see in the noisy image (9), where the curve has barely moved at all in 1400 iterations because the edge detector cannot deal with noise. Caselles et. al. dealt with this issue by applying a Gaussian filter to the image to improve segmentation [1].

Figure (7) demonstrates the results with an expanding curve. Here we start within the cameraman's body and expand the curve outward with a negative α. Since there is very little spatial variation in his pea coat, the curve quickly passed through the jacket to grab the outline of his body and camera handle. It gets stuck on some of the features near his face.

The results for the binary image are very promising. The curve cleanly wraps around the darker objects, even becoming disjoint where needed. To our eyes, this segmentation is perfect. However, it is not able to meet our tolerance. Take a look at the errors for this case, plotted in Figure (12). At first, the gradient flow and reinitialization work together to decrease the convergence error. Near the 40th repeat, we see the error level out close to zero. Figure (13) shows a zoomed in profile of the error. We see that gradient flow pushes the error down, but each reinitialization causes it to jump back up. This pattern repeats until about 850, where the roles actually switch! Now, the reinitialization works to bring the error down and the gradient flow pushes it back up.

Overall, we see that the Geodesic Active Contours does a good job detecting edges over smooth regions of the image but gets stuck in areas of large spatial variation. We don't know how to explain the strange results in the oscillating error. Lastly, we note that this algorithm is hugely expensive, even with some optimized computation strategies. Iterating an entire surface just to move a single curve works for dealing with disjoint curves, but it is not a good computational strategy.

References

[1] Caselles, V., Kimmel, R., & Sapiro, G. (1997). Geodesic active contours. International journal of computer vision, 22(1), 61-79.

5 Appendix

5.1 Geodesic Active Contours

```
% Geodesic Active Contours

clearvars
close all
clc

[I0, ~] = LoadData(0);
[y,x] = size(I0); % Get the dimensions important for differencing
I = I0(:); % Reshape the image into a vector: L-R columns stack T-B

% Pick some parameters here
maxIter = 1000; % Number of times to repeat gf + reinit steps
tol = 1e-6;
a = -10;
exitflag = 0; % Iteration flag, 0 for incomplete, 1 for complete
dt = 0.01;
gfIter = 10; % Number of iterations to take before reinitialization
reinitIter = 10; % Number of iterations to take in the reinitialization phase
epsin = 1e-6;
% edge detector g calculated below


%% Construct Operators
% y = 3; x = 4;
n = y*x;
e = ones(n,1);
e1 = ones(n,1);
for i = y:y:n
    e1(i) = 0;
end
e2 = ones(n,1);
for i = y+1:y:n
    e2(i) = 0;
end
e3 = ones(n,1);
for i = y:y:n
    e3(i) = 0;
end

% x-Gradient
Dpx = spdiags([-e e],[0 y],n,n);
Dmx = spdiags([-e e],[-y 0],n,n);
Dcx = (1/2)*spdiags([-e e],[-y y],n,n);

% y-Gradient
Dpy = spdiags([e1 -e],[-1 0],n,n);
Dmy = spdiags([e -e2],[0 1],n,n);
Dcy = (1/2)*spdiags([e3 -e2],[-1 1],n,n);


%% Edge Detector
% Using g = 1./(1+||grad(I)||^2), here I is the initial image
% g should be a vector same length as u

g = 1./(1 + ((Dcx*I).^2 + (Dcy*I).^2));
```

```
g1 = max(Dcx*g,0);
g2 = min(Dcx*g,0);
g3 = max(Dcy*g,0);
g4 = min(Dcy*g,0);


%% Construct function u
% Recall that C is the zero level set of some function u
[X,Y] = meshgrid(1:x,1:y);
u = sqrt((X-1/3*x).^2 + (Y-1/2*y).^2) - 30;
% u = sin(Y./1.5) + cos(X./1.5) - 0.5;
u0 = u(:);


%% Main iterations
u_nxt = u0;
figure(2), clf
C = zeros(1,maxIter);
for k = 1:maxIter
    disp(['k is ' num2str(k)])
    for n = 1:gfIter
        u = u_nxt;

        ngu = ((Dcx*u).^2 + (Dcy*u).^2).^(1/2) + epsin; % Norm of the gradient of u
        gpxu = ((max(Dpx*u,0)).^2 + (min(Dmx*u,0)).^2 + (max(Dpy*u,0)).^2 + (min(Dmy*u,0)).^2).^(1/2); % Weir
        K = Dpx*(Dmx*u./ngu) + Dpy*(Dmy*u./ngu);
        hypr = g1.*(Dpx*u) + g2.*(Dmx*u) + g3.*(Dpy*u) + g4.*(Dmy*u);
        RHS = g.*K.*(ngu-epsin) + a*g.*gpxu + hypr;
        u_nxt = u + dt*RHS;

        C((k-1)*gfIter + n) = norm(u-u_nxt);
        if (C((k-1)*gfIter + n) < tol)
            exitflag = 1;
            break;
        end
    end

    if (mod(k,12) == 1)
        clf
        uplot = flipud(reshape(u_nxt,y,x));
        pcolor(flipud(I0)), shading interp, colormap gray, hold on
        contour(uplot,[0 0],'r','linewidth',5);
        drawnow
    end

    u_nxt = Reinitial2D(reshape(u_nxt,y,x),reinitIter);
    u_nxt = u_nxt(:);
end


%% Final Plot
figure(2), clf
uplot = flipud(reshape(u_nxt,y,x));
pcolor(flipud(I0)), shading interp, colormap gray, hold on
contour(uplot,[0.02 0.02],'r','linewidth',5);
drawnow

figure(1), clf
```

```
C = C(1:k);
plot(1:k,C);
```