Activity 8: Variational Methods

Primary Goal: Learn how to perform Total Variation (TV) inpainting.
Secondary Goal: Learn how to modify the energy of a variational method.

1.) Inpainting

Inpainting is the process of restoring damaged or missing parts of an image. Suppose we have a binary mask D that specifies the location of the damaged pixels in the input image f:

$$D(x,y) = \begin{cases} 0 & \text{if pixel (x,y) is damaged in image } f \\ 1 & \text{if pixel (x,y) is not damaged in image } f \end{cases}$$

Load the "cameraman" image and cast it to double format.

```
A = imread('cameraman.tif');
A = double(A);
```

Recall this image has size 256x256. Let's produce some damaged cameraman A1 through A3.

a.) A1: Vertical Bar

Suppose a vertical piece of the image was scratched out. We first create a matrix of all 1's that is the same size as the cameraman image.

```
D1 = ones(size(A));
```

Now let's remove a piece of the image by setting the mask to zero.

```
D1(20:220,150:160)=0;
```

Now we can apply this binary mask to damage image A by typing:

```
A1 = D1.*A;
```

Note we cast image A to a double earlier so we could perform this multiplication. To view the new image with `imshow`, you must cast to 8-bit format.

```
imshow(uint8(A1));
```

b.) A2: Skip Lines

Let's suppose that every 3rd row of the cameraman image was damaged. This is an effect which actually may occur in video interlacing. First we create a matrix of all 1's that is the same size as the cameraman image.

```
D2 = ones(size(A));
```

Next we set every other row of the image to zero.

```
D2(3:3:256,:) = 0;
```

Apply this mask to the orignal image A to create a damaged image A2 and view the result.

c.) A3: Overlapping Text

Finally let's try putting some text on top of our cameraman. Let's read a built-in Matlab binary text image and flip its colors.

```
D3 = 1 - imread('text.png');
```

Apply this mask to the orignal image A to create a damaged image A3 and view the result.

2.) TV Inpainting

Last week, you wrote code that minimizes the Total Variation (TV) energy:

$$E[u|f] = \int_\Omega ||\nabla u|| \quad + \lambda(u-f)^2 d\vec{x}.$$

For a noisy input image f, this minimization will produce a denoised image u. The TV model is very flexible and we can modify it to perform other image processing tasks.

In the TV energy, we should "turn off" the fidelity term $(u-f)^2$ where $D=0$ since we do not trust the information at these damaged pixels. We can do this by simply multiplying the fidelity term by D:

$$E[u|f,D] = \int_\Omega ||\nabla u|| \quad + \lambda\, D(u-f)^2 d\vec{x}. \tag{1}$$

Then for undamaged pixels where $D=1$, the algorithm will perform TV denoising as normal. For pixels where $D=0$, we will not match to the damaged information in image f.

If we calculate the first variation of energy ∇E of (1) and then evolve $\frac{\partial u}{\partial t} = -\nabla E$ to perform steepest descent minimization, we get the PDE:

$$\frac{\partial u}{\partial t} = \frac{u_{xx}{u_y}^2 - 2u_x u_y u_{xy} + u_{yy}{u_x}^2}{\left({u_x}^2+{u_y}^2\right)^{3/2}} - 2\lambda D(u-f). \tag{2}$$

In other words, the only change from last week's TV denoising model was that we added a term *D* to the second term.

a.) Open your TV code from last week and save it as a new function `TV_inpaint`. Modify the function line so that it also takes the inpainting mask D as input:

```
function [u] = TV_inpaint (f, lambda, D)
```

When you call this function, you will have to provide the input image f, the fidelity weight λ, and the inpainting mask D.

b.) Now modify the update step of your code to take into account the inpainting mask D, as shown in equation (2). Remember to use pointwise multiplication.

c.) Run your code on image A1 and mask D1 from #1. Use a time step $\Delta t = 0.5$, stopping time $T = 100$, and fidelity weight $\lambda = 0.2$.

```
u = tv_inpaint(A1,0.2,D1);
```

d.) You should have seen that the TV inpainting code on part (c) did nothing. This is because the black bar corresponds to a local minimum of the TV energy. To make sure we do not start at a local minimum, we can fill the damaged region of the input image f with random noise. In your TV_inpaint function, add the following initialization code:

```
R = 255*rand(size(f));
f = D.*f + (1-D).*R;
```

This new image will take on the value of the original image A where D=1 and a random value from matrix R where D=0. Run the TV inpainting code on this image A1 and mask D1.

e.) Run your code on the other 2 inpainting examples you set up in #1 (A2&D2 and A3&D3). Note that examples A2 and A3 had more damaged pixels than image A1, but A1 gave the worst inpainting results. Can you explain why?