

Numerical Analysis & Image Processing in Matlab

Todd Wittman

June 18, 2008

www.math.ucla.edu/~wittman/reu2009/bootcamp.html



Numerical Analysis

- The goal of numerical analysis is to solve partial differential equations (PDEs) on a computer.
- The heart of numerical analysis is *discretizing* a continuous function with a discrete approximation.
- Can you discretize the first derivative df/dx ?

Finite Differences

- Recall the definition of the derivative:

$$f_x = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

- On an image, we can't let h go to zero. We are restricted to the pixel size, so the smallest h can be is 1 pixel.
- The approximation of the derivative is called a *finite difference*.
- Forward Difference: $h=1$
 $f_x = f(x+1,y) - f(x,y)$
- Backward Difference: $h=-1$
 $f_x = f(x,y) - f(x-1,y)$
- Center Difference: $h=2$
 $f_x = (f(x+1,y) - f(x-1,y)) / 2$

Finite Differences

- We have to be careful at the borders of the image.
- We generally assume Neumann boundary conditions ($du/dn=0$). This means the values at the border are repeated.
- Remember Matlab lists row then column, so the derivative in x is on the second subscript.
- Forward Difference
 $m = \text{length}(f);$
 $f_x = f(2:m, m) - f(1:m);$
- Backward Difference
 $f_x = f(1:m) - f(1,1:m-1);$

The 1D Heat Equation

- $du/dt = d/dx (c du/dx)$
- What happens if c is constant?
- Can we discretize this equation?
- Can we code it up in Matlab and visualize the results?

The Heat Equation

- Let's write a m-file that evolves the heat equation.
1D: $u_t = cu_{xx}$
- We need to figure out the finite differences for u_t and u_{xx} .
- We also need to pick a time step dt and a stopping time T .

1D Heat Equation Function

```
function [u] = heat_equation (u0, c)

m = length(u0);
u0 = double(u0);
subplot(121);
plot(u0);
title('Original');

dt = 0.2;
T = 50;
u = u0;

for t = 0:dt:T
    u_xx = u(2:m m) - 2*u + u(1 1:m-1);
    u = u + dt*c*u_xx;
    subplot(122);
    plot(u);
    title(['t=', num2str(t)]);
    drawnow;
end;
```

The 2D Heat Equation

- Can you extend the heat equation to 2D?
- So instead of a signal u , we will evolve a matrix u .
- The 2D heat equation is:

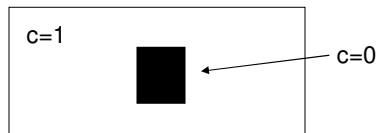
$$u_t = \nabla \cdot (c \nabla u)$$

- If c is constant (isotropic diffusion), we get

$$u_t = c \Delta u$$

Anisotropic Heat Equation

- What if the conductivity c is not-constant (anisotropic diffusion)?
- Write a m file that handles the case when c is a matrix.
- For example, in the picture below we have an insulating block ($c=0$) in the middle.



Now on to images...

- 2D numerical analysis has a neat connection to image processing.
- Let's talk about image processing in Matlab now.
- (We'll see the heat equation come up again at the end.)

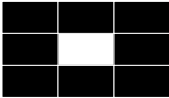
Reading & Writing Images

- Load images with `imread`.
`A = imread('mypic.jpg');`
- Write a matrix to an image with `imwrite`.
- You need to specify the format. To avoid compression / blurring, save as bmp not jpg.
`imwrite(A, 'mypic.bmp', 'bmp');`
- You should get used to converting to `double` after you load an image and converting to `uint8` before you write an image.

Displaying Images

- The `imagesc` command displays a matrix from min (black) to max (white).

0	0	0	0	0	0	0
0	0.1	0	0	5000	0	0
0	0	0	0	0	0	0


- The default `colormap` for a single-channel matrix is `jet`. Change to grayscale with `colormap gray`
- To convert a 3-channel image to 1-channel grayscale, use `rgb2gray`.
- If you want the "true" image colors and size, use `imshow`.
- `imshow` assumes the image is in range [0,255] (`uint8`).
- You can see individual pixel values by clicking on the Data Cursor icon on the figure.

Subplots

- The `subplot` command divides the figure into windows.
`subplot(TotalNumRows, TotalNumCols, index)`
- The index goes from left to right, top to bottom (raster order).
- For example, `subplot(2,3,4)` gets box#4 in a 2x3 figure grid.



- If the numbers are all single digits, we can omit the commas: `subplot(234)`

2D Convolution

- The discrete convolution is given by

$$(f * g)(n) = \sum_{m=-\infty}^{\infty} f[m] \cdot g[n - m]$$
- The idea is that we have an image f and a smaller "mask" g .
- g is generally assumed to be an odd square matrix (3x3, 5x5, etc)
- To compute $(f * g)(x)$, we center g over the pixel $f(x)$, do pointwise multiplication, and add it up.

g =	<table border="1"><tr><td>1</td><td>2</td><td>3</td></tr><tr><td>4</td><td>5</td><td>6</td></tr><tr><td>7</td><td>8</td><td>9</td></tr></table>	1	2	3	4	5	6	7	8	9
1	2	3								
4	5	6								
7	8	9								

f =	<table border="1"><tr><td>10</td><td>45</td><td>19</td><td>6</td><td>20</td><td>33</td></tr><tr><td>17</td><td>99</td><td>22</td><td>84</td><td>12</td><td>66</td></tr><tr><td>39</td><td>8</td><td>42</td><td>71</td><td>3</td><td>52</td></tr><tr><td>0</td><td>18</td><td>26</td><td>57</td><td>64</td><td>76</td></tr><tr><td>21</td><td>37</td><td>98</td><td>7</td><td>43</td><td>88</td></tr></table>	10	45	19	6	20	33	17	99	22	84	12	66	39	8	42	71	3	52	0	18	26	57	64	76	21	37	98	7	43	88
10	45	19	6	20	33																										
17	99	22	84	12	66																										
39	8	42	71	3	52																										
0	18	26	57	64	76																										
21	37	98	7	43	88																										

At $f(x) = 42 \dots$

$$(f * g)(x) = 1 * 99 + 2 * 22 + 3 * 84 + 4 * 8 + 5 * 42 + 6 * 71 + 7 * 18 + 8 * 26 + 9 * 57 = 1910$$

Linear Filters

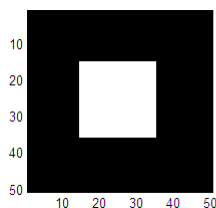
- An operation that can be written as a convolution is called a *linear filter*.
- The matlab function for 2D convolution is `conv2`
`C = conv2(f,g);`
- Note the borders of f are a special case. You can specify how to handle the borders in the input parameter.
- The parameter `'same'` makes the result C have the same size as f.
- We can do multi-channel filtering with `imfilter`.

`C = imfilter(f,g);`

Linear Filters

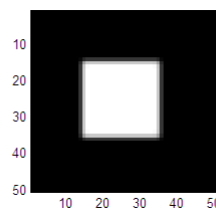
- To preserve the mean gray value of the image, we usually assume the mask values sum to 1.
- The command `fspecial` allows us to build some standard masks.

```
subplot(131);  
imagesc(A);
```



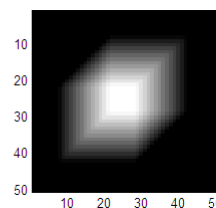
Original

```
subplot(132);  
G = fspecial('gaussian',5,0.7);  
imagesc(imfilter(A,G));
```



Gaussian Blurring

```
subplot(133);  
M = fspecial('motion',20,45);  
imagesc(imfilter(A,M));
```



Motion Blur

Nonlinear Filters

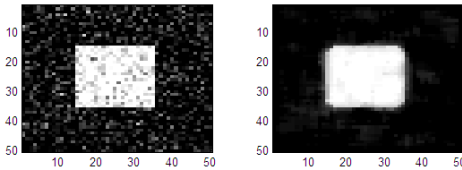
- A nonlinear filter is a neighborhood operation that cannot be written as a convolution.
- To use `nfilter`, we have to provide the name of a function to perform in the neighborhood of each pixel.

```
fun = @(x) median(x(:));
```

```
B = nfilter(A,[3 3],fun);
```

- The median filter is particular good at denoising without blurring. To see this, add some noise with `imnoise`.

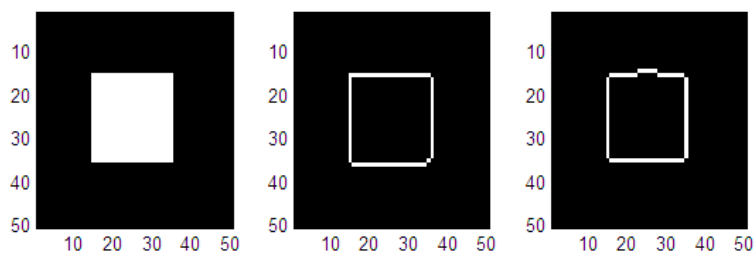
```
subplot(121);  
B = imnoise(A,'gaussian',0,0.05);  
imagesc(B);  
subplot(122);  
C = nfilter(B,[5 5],fun);  
imagesc(C);
```



Edge Detection

- The `edge` function has several methods for detecting edges.

```
subplot(131);  
imagesc(A);  
subplot(132);  
S = edge(A,'sobel');  
imagesc(S);  
subplot(132);  
C = edge(A,'canny');  
imagesc(C);
```



The Gradient

- The norm of the gradient is often used for edge detection.

$$|\nabla u| = \sqrt{u_x^2 + u_y^2}$$

- The gradient should be small in smooth regions.
- The gradient should be large at edges.
- Complicated textures can confound gradient-based edge detection.

Logicals

- We can find values that satisfy certain condition by setting up a boolean statement in brackets.

$$B = [A > 100 \ \& \ A < 200];$$

- B is a 0-1 binary matrix that has a 1 at all places where A was between 100 and 200.

- The complement of this matrix would be:

$$B = [A < 100 \ | \ A > 200];$$

Logicals

- If we want to find the pixel positions where a boolean statement is true, use the `find` function.

```
ind = find(A>100 & A<200);
```

- `ind` will be a column vector of indices where the boolean statement was true. Note `ind` could be empty (size 0).
- Note an index is a single number, not a subscript (row,column). Matlab reads matrices columnwise.

1	4	7	10
2	5	8	11
3	6	9	12

- We can convert between index and subscript with `ind2sub` and `sub2ind`. Note you have to pass the matrix size as input.
- You also get indices from the `min`, `max`, and `sort` functions.

Anisotropic Diffusion

- Evolving the heat equation on an image is called *isotropic diffusion*.
- Isotropic diffusion is mathematically equivalent to Gaussian blurring.
- The problem is that the colors are allowed to flow in all directions equally, including across the edges.
- How can we stop the diffusion at the edges?

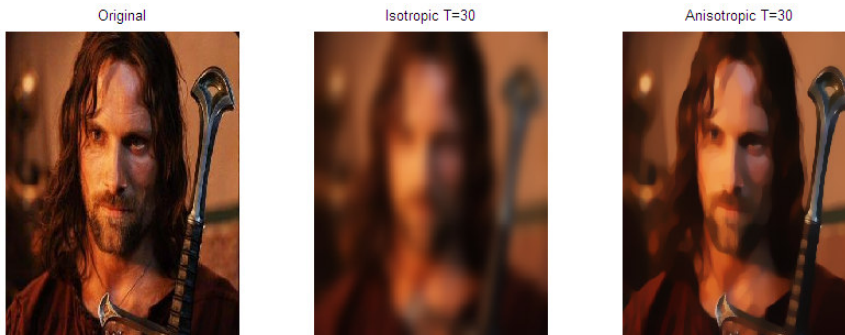
$$u_t = \nabla \cdot \left(\frac{1}{|\nabla u|} \nabla u \right)$$

- Anisotropic diffusion turns real images into "cartoons".

Isotropic vs. Anisotropic Diffusion

- To run the heat equation on a color image, we could just process each of the 3 bands separately.

```
for i=1:3; B(:, :, i) = heat_equation(A(:, :, i)); end;  
imagesc(B);
```



Making Movies

- If you want to make a movie of successive plots, you can use the `getframe` command to turn the active figure into a movie frame.

```
F(i) = getframe;
```

- Be sure to use the `drawnow` command in your loop to display your data.
- You can then play your movie F with

```
movie(F, 1);
```

- If you don't want to plot the results, you can use `im2frame` to convert a matrix directly to a movie frame.
- You can save your movie to an avi file with `movie2avi`.