



Image Processing in Matlab

Todd Wittman
June 18, 2008

www.math.ucla.edu/~wittman/reu2008



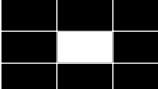
Reading & Writing Images

- 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).

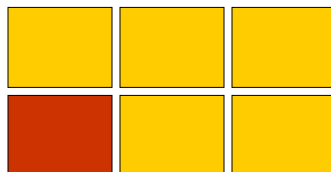
0	0	0	0	0	0	0
0	0.1	0	0	5000	0	0
0	0	0	0	0	0	0



- 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.

$g =$

1	2	3
4	5	6
7	8	9

$f =$

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 $f(x) = 42...$

$$\begin{aligned} (f * g)(x) &= 1 * 99 + 2 * 22 + 3 * 84 + \\ & 4 * 8 + 5 * 42 + 6 * 71 + 7 * 18 + \\ & 8 * 26 + 9 * 57 \\ & = 1910 \end{aligned}$$

Linear Filters

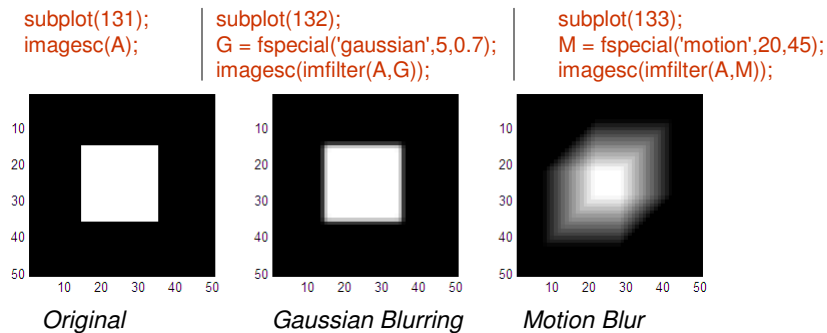
- An operation that can be written as a convolution is called a *linear filter*.
- The matlab function for 2D convolution is `conv2`
- Note the borders of f are a special case. You can specify how to handle the borders in the input parameter.
- The parameter `'same'` makes the result C have the same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

$$C = \text{conv2}(f,g);$$

$$C = \text{imfilter}(f,g);$$

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.



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.

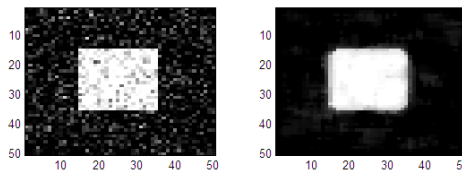
```
fun = @(x) median(x(:));
```

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

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

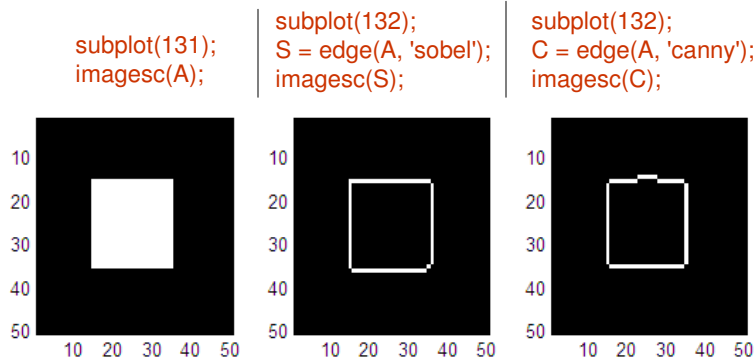
```
subplot(121);
B = imnoise(A,'gaussian',0,0.05);
```

```
subplot(122);
C = nfilter(B,[5 5],fun);
imagesc(C);
```



Edge Detection

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



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.

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
 $[m,n] = \text{size}(A);$
 $A_x = A(:, [2:n, n]) - A(:, 1:n);$
- Backward Difference
 $A_x = A(:, 1:n) - A(:, [1,1:n-1]);$

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.

The Heat Equation

- Let's write a m-file that evolves the heat equation.

$$\underline{1D}: u_t = u_{xx}$$

$$\underline{2D}: u_t = u_{xx} + u_{yy} \quad (u_t = \Delta u)$$

- We need to figure out the finite differences for u_t , u_{xx} , and u_{yy} .
- We also need to pick a time step dt and a stopping time T .

Heat Equation Function

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

[m,n,k] = size(u0);
if k==3
    u0 = rgb2gray(u0);
end;
u0 = double(u0);
subplot(121);
imagesc(u0);
colormap gray;
title('Original');

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

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

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?

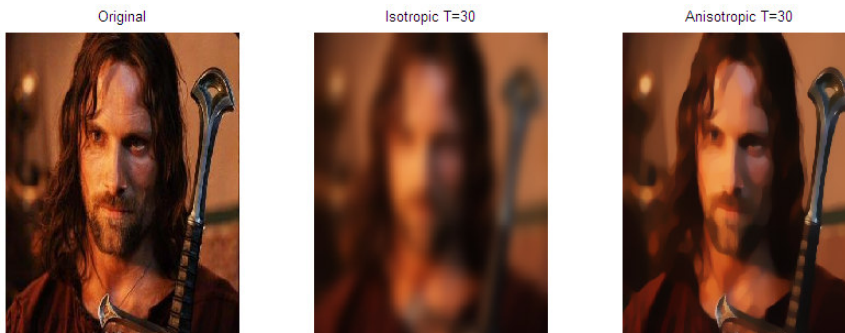
$$u_t = \nabla \cdot \left(\frac{1}{|\nabla u|} \nabla u \right)$$

- 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`.