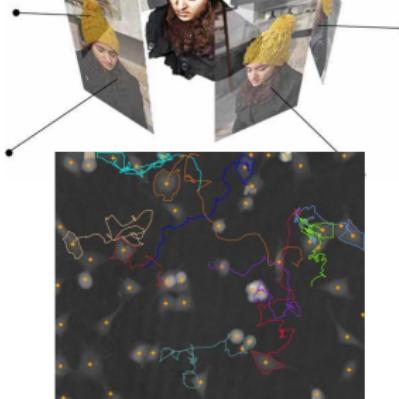
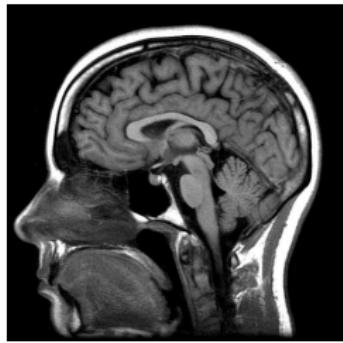
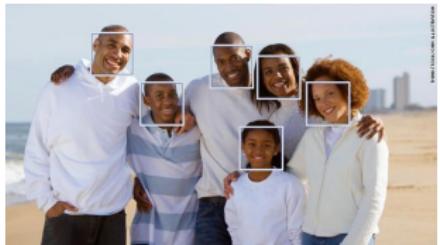


PDEs for Image Processing - CBL Summer School

Martin Robinson

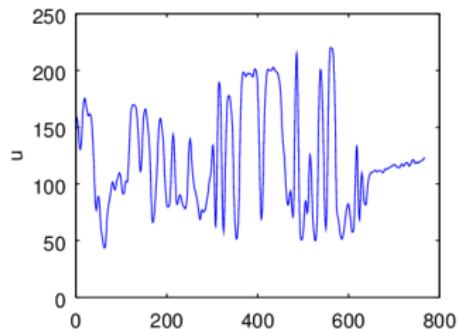
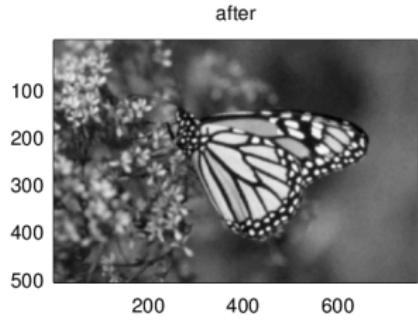
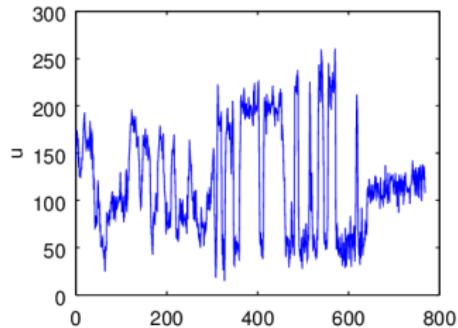
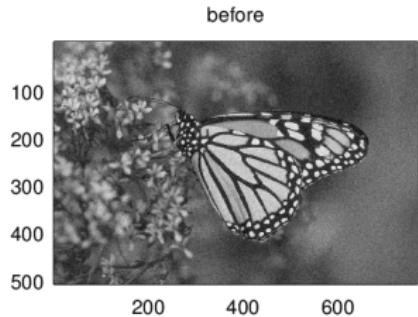
August 10, 2015

Image Processing



- certain image processing operations can be written as solution of partial differential equations (PDEs)
- can apply well-developed mathematical techniques to analyse properties and propose improvements
- use high performance numerical algorithms to generate solutions

Diffusion equation = noise removal



Advection equation = inpainting



Image Domain

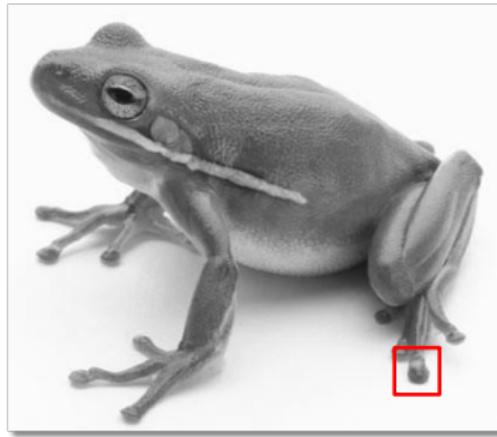


Figure : Digital image $u_{i,j}$ is a discretised and quantised version of “real” scene $u(x, y)$

Opening Image in MATLAB

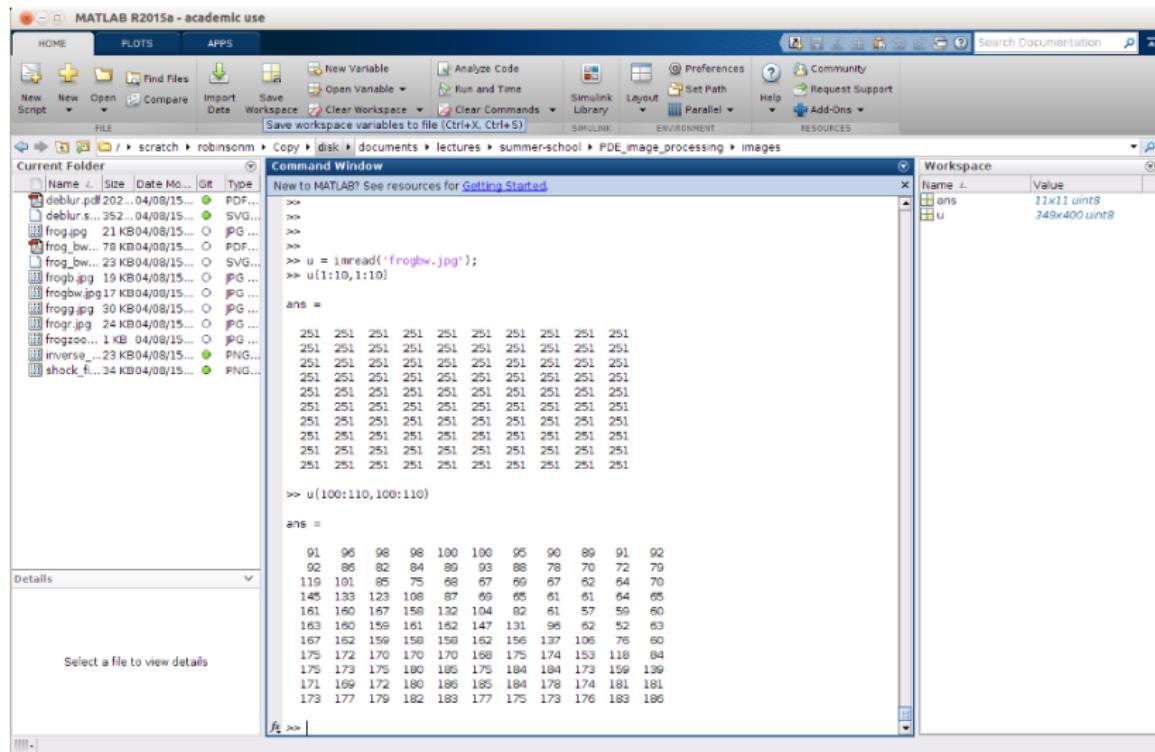
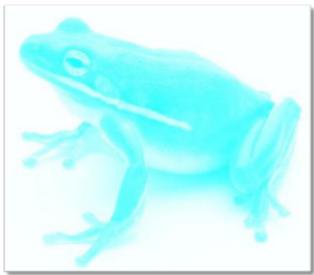


Figure : Each pixel is a integer from 0...255

Color Images



Red



Green



Blue

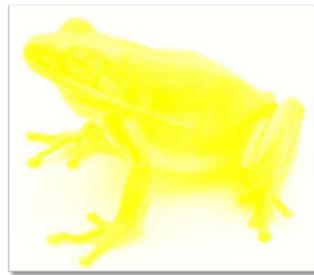


Figure : The red, green and blue channels from a color image, $u_{i,j}^r$, $u_{i,j}^g$, $u_{i,j}^b$, are samples from a set of three related continuous fields, $u^r(x, y)$, $u^g(x, y)$, $u^b(x, y)$

Image Domain

- Image domain $\Omega = \{x, y \in \mathbb{R}^+ : x < L_x \wedge y < L_y\}$
- “Perfect” digital image $u_{i,j}$

$$u_{ij} = u(x_i, y_i) \quad (1)$$

$$x_i = i - 0.5 \quad (2)$$

$$y_j = j - 0.5 \quad (3)$$

- Real images consist of both a source and noise term n

$$u_{i,j} = u(x_i, y_i) + n(x_i, y_i) \quad (4)$$

- Goal is to recover $u(x, y)$ from set of noisy pixel values $u_{i,j}$

- We can examine our image in the frequency domain using a *fourier transform* ($\mathcal{F}(f) = \hat{f}$), defined as

$$\hat{f}(k) = \int_{-\infty}^{\infty} f(x) e^{-ikx} dx$$

where k is the mode frequency in units of radians per unit length

- In two-dimensions

$$\hat{f}(k_x, k_y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) e^{-i(k_x x + k_y y)} dx dy$$

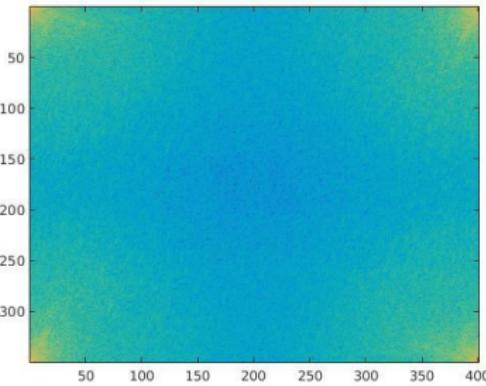


Figure : (left) b&w frog image $u_{i,j}$ (right) its fourier transform \hat{u}_{k_x, k_y}

Signal Processing: Low-pass filter

- Assume that the noise term is comprised of mainly high frequency terms
- A low-pass filter is a filter that removes or attenuates some of the higher frequency components of the original signal.
- A low-pass filter can be implemented by convolution of the pixel values by a function f .

$$(f * u)(x) = \int_{-\infty}^{\infty} f(x - x')u(x')dx'$$

- A common low-pass filter is the *Gaussian* filter, which uses a gaussian function for f

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$$

- Fourier transform of a gaussian (**exercise**)

$$\hat{f}(k) = e^{-\frac{k^2}{2\sigma^2}}$$

- Convolution property of fourier transform (**exercise**)

$$\mathcal{F}\{f(x) \star g(x)\} = \hat{f}(k)\hat{g}(k)$$

Using the convolution property and the fourier transform of a gaussian, it can be seen that the application of a gaussian filter attenuates the frequency components of the original image by a gaussian. i.e. a low-pass filter

The heat equation

- Another option is to remove the noise term by applying a numerical discretisation of the *diffusion equation*.
- In one-dimension is given by

$$\begin{cases} u_t(x, t) = Du_{xx}(x, t) \\ u(x, 0) = u_0(x) \end{cases}$$

- Speed of evolution u_t is proportional to the concavity u_{xx} .
- i.e. small features (with high u_{xx}) will be smoothed out first.

Fundamental solution

- fourier transform of heat equation (**exercise**)

$$\mathcal{F}\{u_t(x, t)\} = \mathcal{F}\{Du_{xx}(x, t)\} \quad (5)$$

$$\hat{u}_t = -Dk^2 \hat{u}. \quad (6)$$

- therefore $\hat{u}(k) = C(k)e^{-k^2 Dt}$
- use $\hat{u}(k, 0) = \hat{u}_0(k) = C(k)$ to obtain (**exercise**)

$$u(x, t) = \frac{1}{\sqrt{4\pi Dt}} \int_{-\infty}^{\infty} u_0(x') e^{-\frac{(x-x')^2}{4Dt}} dx'$$

- Note that this is the same as the gaussian low-pass filter with $\sigma = \sqrt{2Dt}$.

Finite Differences (forward time, central space)

- time derivative:

$$u_t(x_i, t_n) \approx \frac{u_i^{n+1} - u_i^n}{\Delta t}$$

- space derivative:

$$u_{xx}(x_i, t_n) \approx \frac{1}{\Delta x^2} (u_{i+1}^n - 2u_i^n + u_{i-1}^n) = u_{i+1}^n - 2u_i^n + u_{i-1}^n$$

- put them together: (**exercise**)

$$u_i^{n+1} = \Delta t D u_{i+1}^n + (1 - 2\Delta t D) u_i^n + \Delta t D u_{i-1}^n$$

Boundary Conditions

- We might want to set a dirichlet boundary condition as the initial pixel values at the edges.

$$u(0, t) = u_0(0) \quad (7)$$

$$u(L, t) = u_0(L). \quad (8)$$

- A zero-Neumann b.c. is often used to ensure that the gradient of the image pixels at the boundary is zero.

$$\frac{\partial u}{\partial x}(0) = 0 \quad (9)$$

$$\frac{\partial u}{\partial x}(L) = 0 \quad (10)$$

This could be implemented, for example, by having a set of “mirror” pixels surrounding the image.

$$u_{0,j} = u_{1,j} \tag{11}$$

$$u_{L+1,j} = u_{L,j} \tag{12}$$

2D Heat Equation

$$\begin{cases} u_t(x, y, t) = D(u_{xx}(x, y, t) + u_{yy}(x, y, t)) \\ u(x, y, 0) = u_0(x, y) \end{cases}$$

Finite Differences (forward time, central space)

- time derivative:

$$u_t(x_i, y_j, t_n) \approx \frac{u_{i,j}^{n+1} - u_{i,j}^n}{\Delta t}$$

- spatial derivatives:

$$u_{xx}(x_i, y_j, t_n) \approx \frac{1}{\Delta x^2} (u_{i+1,j}^n - 2u_{i,j}^n + u_{i-1,j}^n) = u_{i+1,j}^n - 2u_{i,j}^n + u_{i-1,j}^n$$

$$u_{yy}(x_i, y_j, t_n) \approx \frac{1}{\Delta y^2} (u_{i,j+1}^n - 2u_{i,j}^n + u_{i,j-1}^n) = u_{i,j+1}^n - 2u_{i,j}^n + u_{i,j-1}^n$$

- putting them together:

$$u_i^{n+1} = \Delta t D (u_{i+1,j}^n + u_{i-1,j}^n + u_{i,j+1}^n + u_{i,j-1}^n) + (1 - 4\Delta t D) u_{i,j}^n$$

The unsharp mask

- image sharpening technique
- widely used (overused?)
- simplified algorithm:
 - ① Diffuse image $u(x, y, 0)$ to obtain $u(x, y, t_1)$
 - ② New image is $u_{new} = u(x, y, 0) + (u(x, y, 0) - u(x, y, t_1))$
- biological motivation: <http://www.cambridgeincolour.com/tutorials/unsharp-mask.htm>