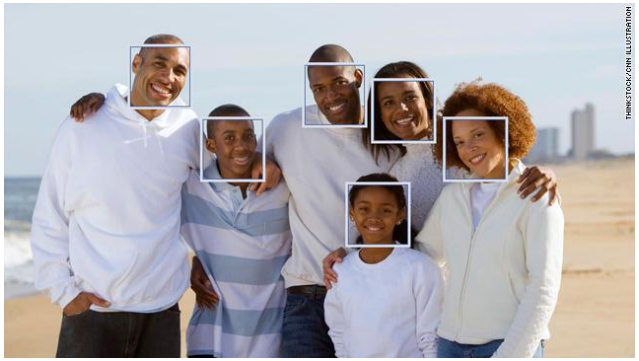
PDEs for Image Processing - CBL Summer School

Martin Robinson

August 10, 2016

Image Processing





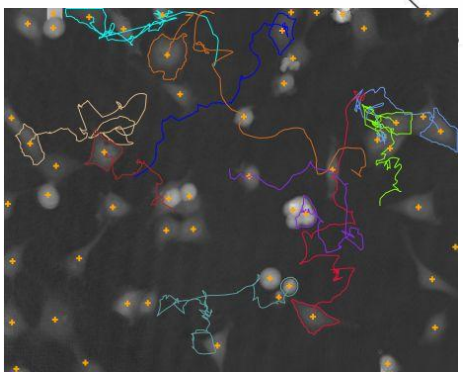




Figure 1:

PDEs for 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

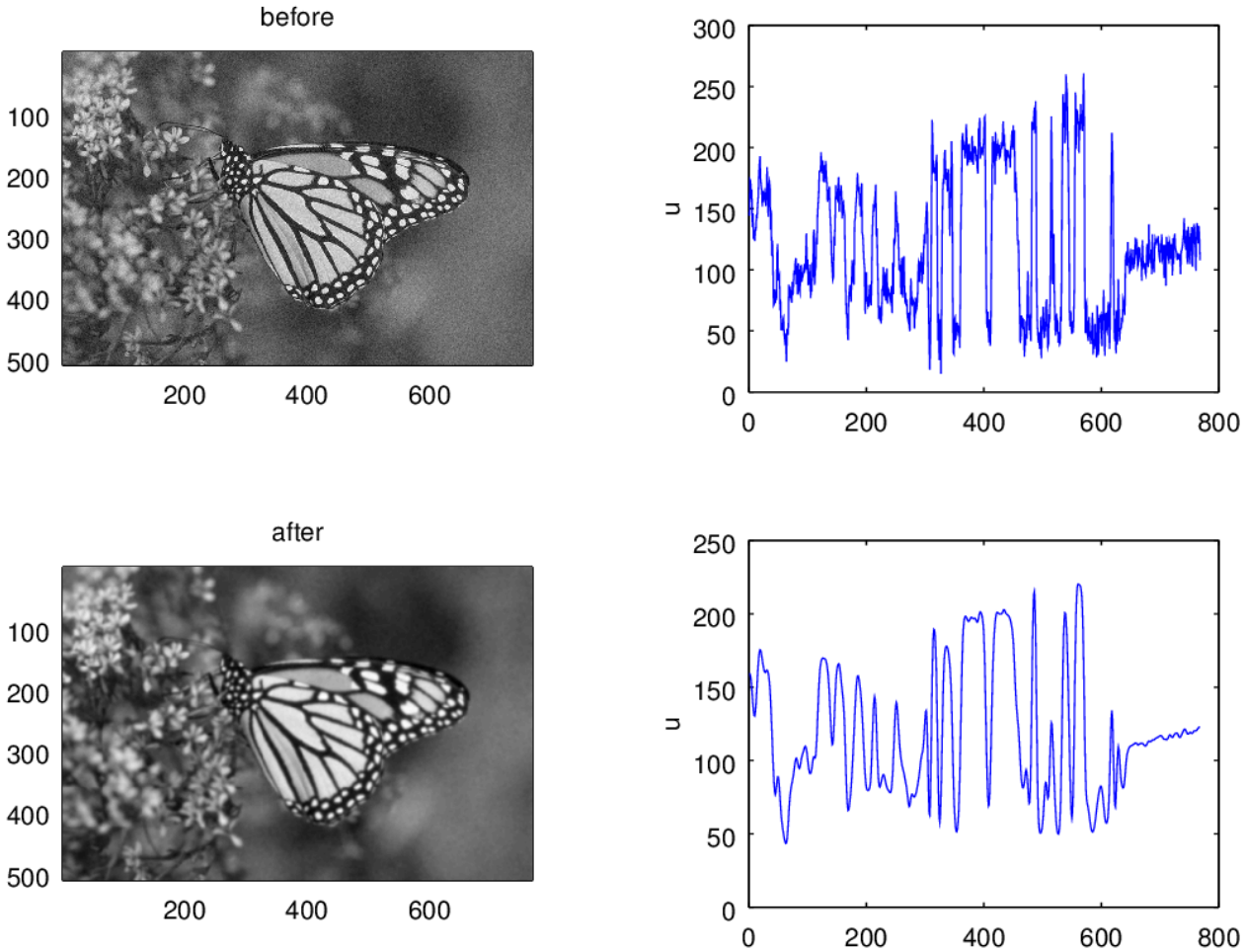


Figure 2:

Advection equation = inpainting

Figure 3:

Image Domain

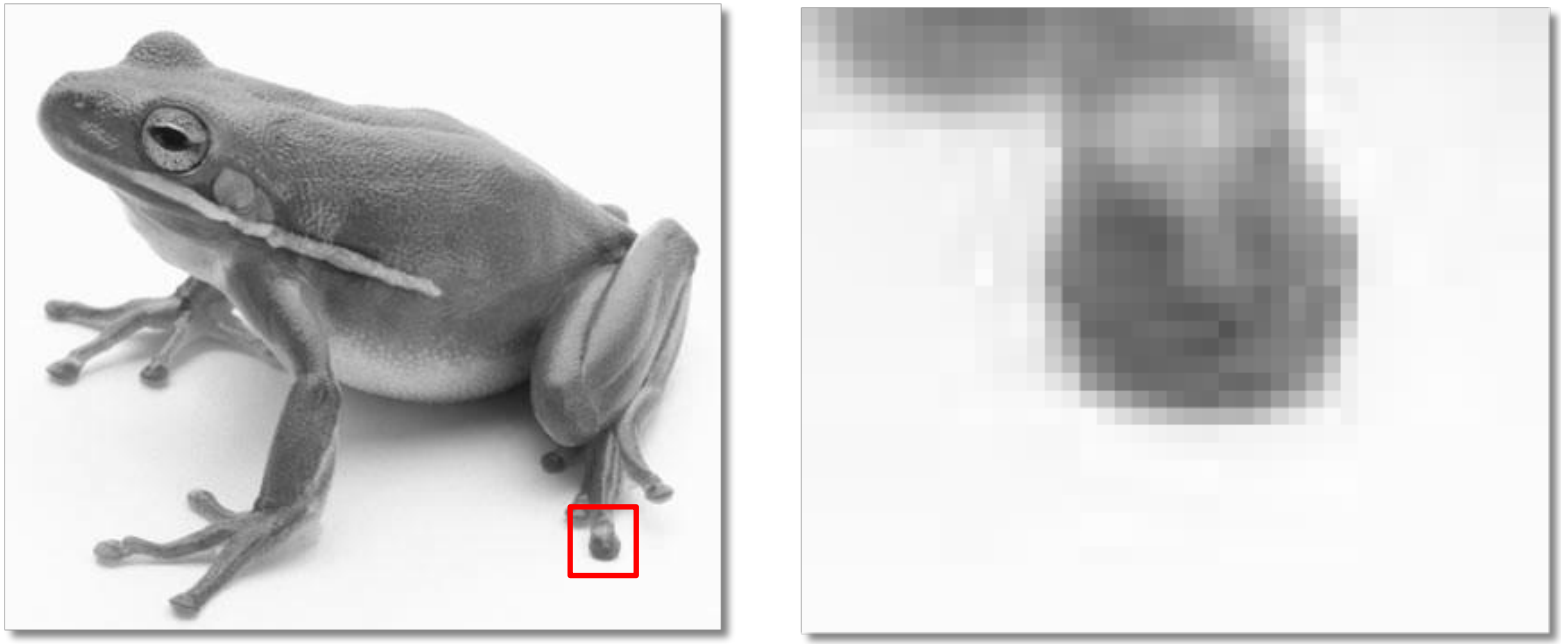


Figure 4: Digital image $u_{i,j}$ is a discretised and quantised version of "real" scene $u(x, y)$

Opening Image in MATLAB

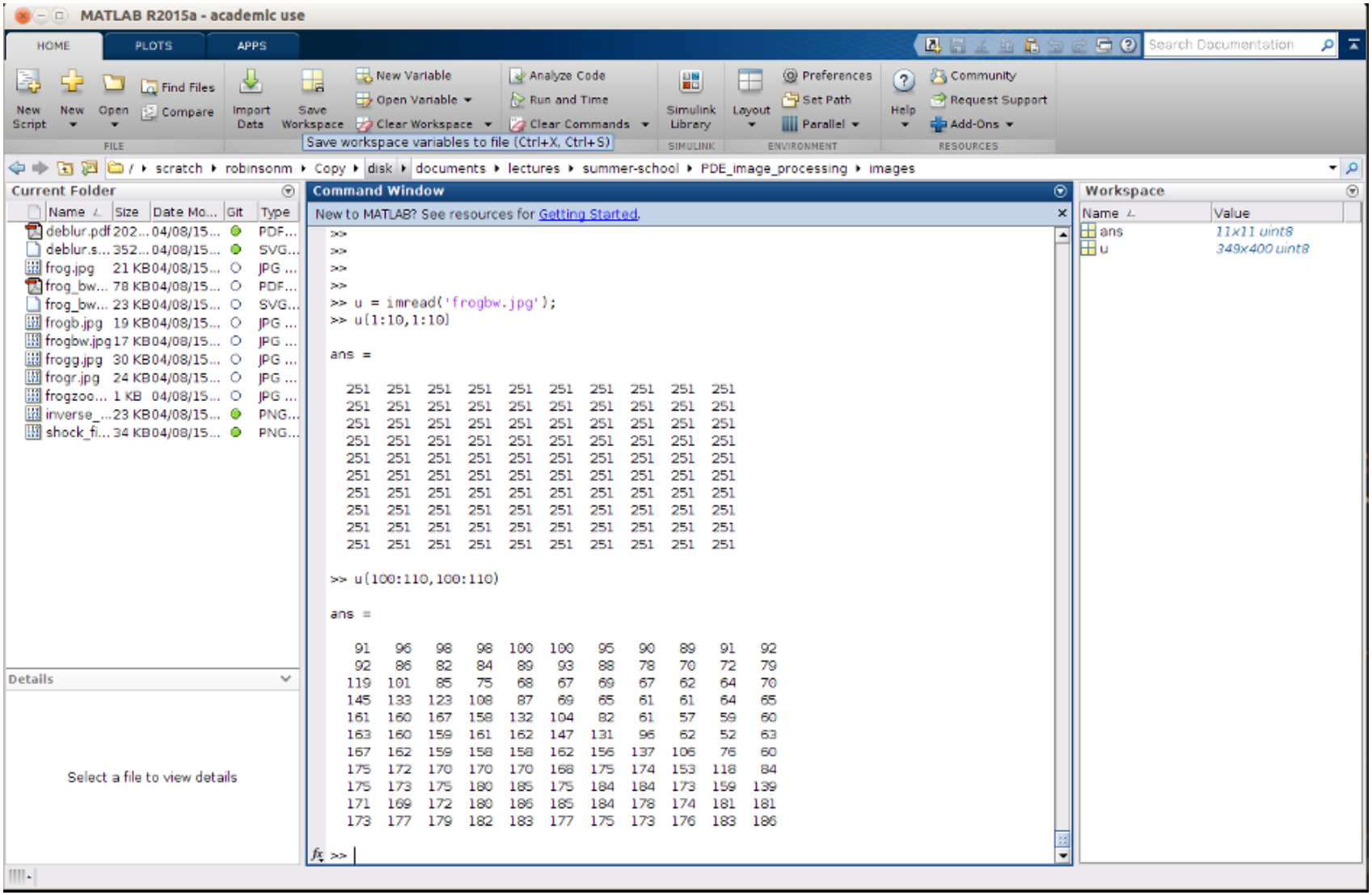


Figure 5: Each pixel is a integer from 0 - 255

Color Images

Red

Green

Blue

Figure 6: The red, green and blue channels from a color image, $u^r_{i,j}$, $u^g_{i,j}$, $u^b_{i,j}$, 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 $u_{i,j}$

$$u_{ij} = u(x_i, y_i) \tag{1}$$
$$x_i = i - 0.5 \tag{2}$$
$$y_j = j - 0.5 \tag{3}$$

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

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

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

Signal Processing - Fourier Domain

- 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

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

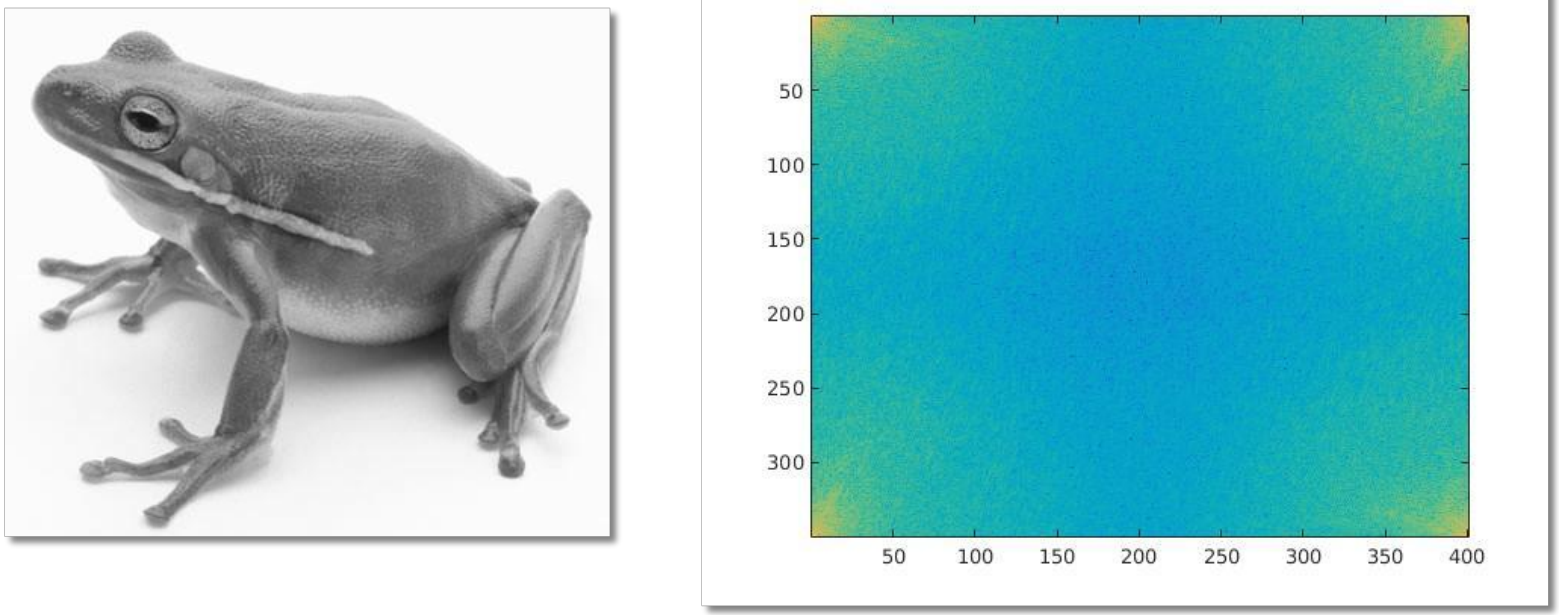


Figure 7: (left) b&w frog image $u_{i,j}$ (right) its fourier transform $\hat{u}_{k_x,k_y}$

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.

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

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

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

Frequency response of gaussian low-pass filter

- Fourier transform of a guassian (**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 guassian, 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 discritisation 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**)

$$\mathcal{F}\{u_t(x,t)\} = \mathcal{F}\{Du_{xx}(x,t)\} \tag{5}$$
$$\hat{u}_t = -Dk^2\hat{u}. \tag{6}$$

- therefore $\hat{u}(k) = C(k)e^{-k^2Dt}$
- 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.

$$u(0,t) = u_0(0) \tag{7}$$
$$u(L,t) = u_0(L). \tag{8}$$

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

$$\frac{\partial u}{\partial x}(0) = 0 \tag{9}$$
$$\frac{\partial u}{\partial x}(L) = 0 \tag{10}$$

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:

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

Practical exercises

- run the examples in the `image_processing` folder in the git repo
- extend the example code `diffusion1.m` to work on color images
- last week we constructed a sparse matrix L to approximate u_{xx} and used this in the solution of the heat eq. Modify the example codes, or create new ones from scratch, and use this technique instead of index vectors
- implement a simplified unsharp mask filter (`http://www.cambridgeincolour.com/tutorials/unsharp-mask.htm`):
  1. Diffuse image $u(x, y, 0)$ to obtain $u(x, y, t_1)$
  2. New image is $u_{new} = u(x, y, 0) + (u(x, y, 0) - u(x, y, t_1))$