Lecture 10
Diffusion and Image Processing

Jason J. Bramburger

Filtering is not the only way to denoise an image. In this lecture we are going to use tools and ideas from partial differential equations (PDEs) to see how to reduce the noise in an image. In particular, we are going to make use of the *heat equation* in two spatial dimensions:

$$\frac{\partial u}{\partial t} = D\underbrace{\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right)}_{=:\nabla\cdot(\nabla u)},$$

where $D > 0$ is referred to as a diffusion coefficient. Here $u(x, y, t)$ is the time-dependent profile of the solution evolving in time according to the heat equation (which will represent our image) and we comment that since space (x, y variables) will be finite, we have to impose some boundary conditions as well. We also require an initial condition $u(x, y, 0) = u_0(x, y)$, which we will see is taken to be the noisy image. To build up some intuition on with the heat equation, let's start with only one spatial dimension.

The Heat Equation in 1D

Based on the form of the heat equation in two spatial dimensions above, you can probably guess that in one spatial dimension we just have

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2},$$

where $u = u(x, t)$. To solve PDEs we require two things: an initial condition in time and boundary conditions in space. That is, we need an initial distribution $u(x, 0) = u_0$, and if we assume that $x \in [-L, L]$ then we need to prescribe how u behaves at $x = \pm L$ for all $t \geq 0$. Some common choices for boundary conditions are:

$$\begin{aligned} u(\pm L, t) &= 0, && \text{Fixed/Dirichlet} \\ \frac{\partial u}{\partial x}(\pm L, t) &= 0, && \text{Insulated/Neumann} \\ \begin{cases} u(L, t) = u(-L, t) \\ \frac{\partial u}{\partial x}(L, t) = \frac{\partial u}{\partial x}(-L, t) \end{cases} & && \text{Periodic} \end{aligned}$$

As you are probably aware, PDEs are notoriously hard to solve. Luckily, the heat equation is one of only a handful of PDEs we can solve exactly! How? Using our best friend in this course: the Fourier transform! Recall that the Fourier transform takes derivatives in space and turns them into multiplications of the transformed function with ik, so applying the Fourier transform to the heat equation gives:

$$\frac{\partial \hat{u}}{\partial t} = -Dk^2\hat{u},$$

where $\hat{u} = \hat{u}(k, t)$ is the Fourier transform of $u(x, t)$ at each time t. Now, notice that there are only derivatives in time in the above equation, meaning we now have an ordinary differential equation (ODE). This is a separable ODE, so the solution is given by:

$$\hat{u}(k, t) = \hat{u}_0(k)\mathrm{e}^{-Dk^2 t},$$

where $\hat{u}_0(k)$ is the Fourier transform of the initial condition $u(x, 0) = u_0(x)$ for the heat equation. Hence, the Fourier transform of the solution $u(x, t)$ to the heat equation at some time t is just taking the Fourier transform at time $t = 0$ and multiplying it by a Gaussian filter. As time goes on, the filter gets more and more narrow (coming from t getting bigger in the exponent).

Think about how interesting this is: solving the heat equation is actually the SAME as applying a Gaussian filter. Therefore, to denoise a 1D signal (such as our radar example from Lecture 2), we can just use the noisy signal as an initial condition, $u_0(x)$, and evolve in time with according to the heat equation. This process is the same as Gaussian filtering, but to solving the heat equation doesn't require one to ever take the Fourier transform.

Numerically Solving the Heat Equation

The first thing that needs to be done to solve the heat equation numerically is discretize the spatial domain. Let us consider a spatial step h, so that

$$x_n = -L + nh,$$

where $n = 0, \ldots, N$. Then, we can write $u_n = u(x_n, t)$ to be the time-dependent value of the solution to the heat equation at each point in space x_n, and put them into a vector

$$\vec{u} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix}.$$

Since we have discretized space, we also need to discretize the spatial derivative $\partial^2 u/\partial x^2$. We do this using the finite difference formula:

$$\frac{\partial^2 u}{\partial x^2} = \frac{u(x+h,t) - 2u(x,t) + u(x-h,t)}{h^2},$$

which means that our heat equation becomes a system of ODEs (one for each point):

$$\frac{\mathrm{d}u_n}{\mathrm{d}t} = \frac{D}{h^2}(u_{n+1} - 2u_n + u_{n-1}).$$

This is a linear equation (only linear terms in u_n are present), so we can write this in matrix form as

$$\frac{\mathrm{d}\vec{u}}{\mathrm{d}t} = \frac{D}{h^2} A\vec{u},$$

where A is the matrix

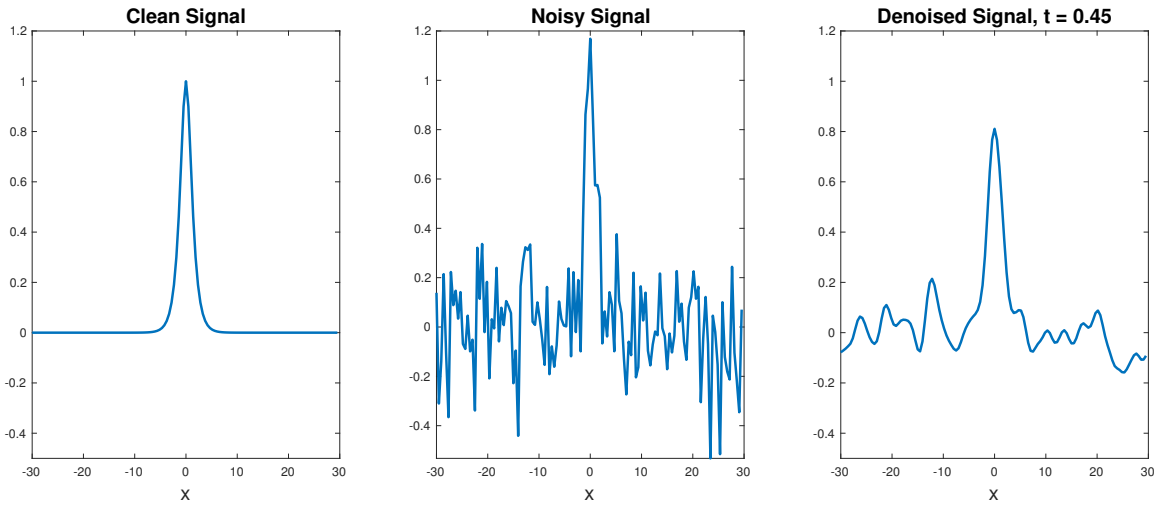
$$A = \begin{bmatrix} -2 & 1 & 0 & \cdots & 0 & 1 \\ 1 & -2 & 1 & 0 & \cdots & 0 \\ 0 & \vdots & \vdots & \ddots & & 0 \\ \vdots & & & & & 0 \\ \vdots & \cdots & 0 & 1 & -2 & 1 \\ 1 & 0 & \cdots & 0 & 1 & -2 \end{bmatrix}.$$

Now that we have a linear ODE, we can use one of any number of methods to advance forward in time. MATLAB has some excellent and accurate built-in solvers, so let's just use one of those. We will use ode45 and the heat equation to denoise a our noisy signal from the first few lectures.

```
L = 30; n =128;
x2 = linspace(-L,L,n+1);
x = x2(1:n);

u0 = sech(x);
ut = fft(u0);
noise = 2;
utn = ut + noise*(randn(1,n) + 1i*randn(1,n));
un = real(ifft(utn));

figure(1)
pos = get(gcf,'Position');
set(gcf,'Position',[0 100 3*pos(3) pos(4)])
```

```
subplot(1,3,1)
plot(x,u0,'Linewidth',2)
axis([-L L -0.5 1.2])
xlabel('x','Fontsize',16)
title('Clean Signal','Fontsize',16)
subplot(1,3,2)
plot(x,un,'Linewidth',2)
axis([-L L -0.5 1.2])
xlabel('x','Fontsize',16)
title('Noisy Signal','Fontsize',16)
```

```
D = 1;
h = x(2) - x(1);

% Create A matrix
e = ones(n,1);
A = spdiags([e -2*e e], [-1 0 1],n,n);
A(1,n) = 1; % periodic BCs
A(n,1) = 1;

% ode45 for time-stepping
rhs = @ (t,u) (D/h^2)*A*u;
[t,u] = ode45(rhs,linspace(0,0.5,101),un);

% Plot a video of the evolving heat equation solution
figure(1)
for j = 1:length(t)
    subplot(1,3,3)
    plot(x,u(j,:),'Linewidth',2)
    axis([-L L -0.5 1.2])
    xlabel('x','Fontsize',16)
    title(['Denoised Signal, t = ',num2str(t(j))],'Fontsize',16)
    pause(0.1)
end
```



Notice that there is a point at which the noise appears to have died down and beyond that point we are just losing the signal as time continues on. This is overfiltering (the filter has become too thin). This is something we have to be careful of with this method - there is an ideal time to stop that has to be evaluated visually.

The Heat Equation in 2D

The previously presented ideas carry over to the heat equation in 2D with little change or augmentation. We will

consider a square $x \in [-L, L]$ and $y \in [-L, L]$ and recall the heat equation from earlier:

$$\frac{\partial u}{\partial t} = D\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right).$$

In 2D the behaviour follows the same general temporal decay that we saw exhibited in the 1D equation. That is, large values of $|u|$ spread out and into regions where $|u|$, eventually moving towards a constant u as $t \to \infty$ (assuming Dirichlet, Neumann, or periodic boundary conditions).

Solving the 2D heat equation numerically is handled in a similar manner to the 1D equation, but now we have a whole grid of points resulting from the discretization of both x and y. That is, we can define

$$x_n = -L + hn, \quad y_n = -L + hn,$$

for $n = 0, \ldots, N$. Then, we will write $u_{m,n} = u(x_m, x_n, t)$ and express both $\partial^2 u/\partial x^2$ and $\partial^2 u/\partial y^2$ in terms of finite difference formulas:

$$\frac{\partial^2 u}{\partial x^2} = \frac{u(x+h, y, t) - 2u(x, y, t) + u(x-h, y, t)}{h^2}$$
$$\frac{\partial^2 u}{\partial y^2} = \frac{u(x, y+h, t) - 2u(x, y, t) + u(x, y-h, t)}{h^2}.$$

Hence, the linear ODEs for each $u_{m,n}$ become

$$\frac{\mathrm{d}u_{m,n}}{\mathrm{d}t} = \frac{D}{h^2}(u_{m+1,n} + u_{m,n+1} - 4u_{m,n} + u_{m-1,n} + u_{m,n-1}).$$

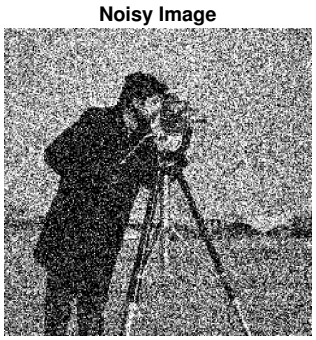
This is sometimes called a 5-point stencil because the differential equation for component $u_{m,n}$ requires the state of five points: itself and its four nearest-neighbours.

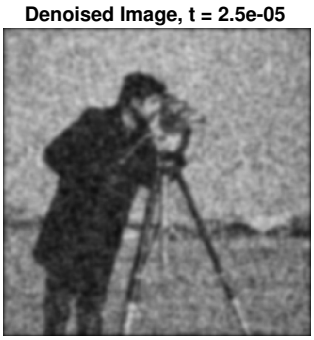
Writing the above ODE in vector form becomes somewhat difficult since the matrix A is tricky to write out properly. Luckily, MATLAB can handle this with relative ease and we don't have to worry too much about this. Now, let's try denoising an image using the 2D heat equation.

```
I = imread('cameraman.tif');
I = im2double(I);
In = imnoise(I,'gaussian',0,0.1);

figure(2)
pos = get(gcf,'Position');
set(gcf,'Position', [0 100 3*pos(3) pos(4)])
subplot(1,3,1)
imshow(I)
title('Clean Image','Fontsize',16)
subplot(1,3,2)
imshow(In)
title('Noisy Image','Fontsize',16)
```

```
D = 1;
[ny,nx] = size(I);
x = linspace(0,1,nx);
y = linspace(0,1,ny);
dx = x(2) - x(1);
dy = y(2) - y(1);
[X,Y] = meshgrid(x,y);

% Build matrix A using sparse matrices
ex = ones(nx,1);
ey = ones(ny,1);
Dx = (spdiags([ex -2*ex ex],[-1 0 1],nx,nx))/dx^2;
Ix = speye(nx);
```

```
Dy = (spdiags([ey -2*ey ey],[-1 0 1],ny,ny))/dy^2;
Iy = speye(ny);
L = kron(Iy,Dx) + kron(Dy,Ix);

% Simulate ODE
rhs = @ (t,u) D*L*u;
[t,u] = ode45(rhs,linspace(0,2.5e-5,101),reshape(In,nx*ny,1));

% Plot denoised image
figure(2)
for j = 1:length(t)
   subplot(1,3,3)
   U = reshape(u(j,:).',ny,nx);
   imshow(U)
   drawnow
   title(['Denoised Image, t = ',num2str(t(j))],'Fontsize',16)
   pause(0.1)
end
```





Final Note: We saw that evolving the heat equation was the same as using Gaussian filtering. In practice, we could allow the diffusion coefficient, D, to vary over space. That is, $D = D(x, y)$, resulting in the spatially inhomogeneous heat equation

$$\frac{\partial u}{\partial t} = \nabla \cdot (D(x, y) \cdot \nabla u).$$

With different choices of $D(x, y)$ we can allow for more diffusion (large D) in certain parts of the photo than others. This is something that is not possible with spectral filtering because the Fourier transform loses all spatial information.