Lecture 24
Modal Expansion Techniques for PDEs

Jason J. Bramburger

The data analysis tools we have learned can be used for much more than just signals and images. Remember, signals and images are just good examples of 1D and 2D datasets that happen to have clearly defined problems associated with them (denoising, separation, etc). Here we are going to see how they can help us with partial differential equations (PDEs). In particular, many pattern-forming PDEs might look intimidating on paper, but their solutions are often low-dimensional. We briefly saw in Lecture 14 that we can use the proper orthogonal decomposition to predict the resulting low-dimensional dynamics and in this lecture we will build on that discussion.

We will start in the abstract with a general evolution equation

$$\mathbf{U}_t = \mathbf{N}(\mathbf{U}, \mathbf{U}_x, \mathbf{U}_{xx}, \dots, x, t)$$

where $\mathbf{U}$ is a vector of physically relevant quantities and the subscripts t and x denote partial differentiation. The left side of the equation tells you that the vectors are changing in time, so we have a *dynamical system.* The right side contains a general function $\mathbf{N}$ that quantifies how $\mathbf{U}$ changes in time based on the current state of $\mathbf{U}$, its spatial derivatives, and time and space.

In general, there are no techniques for getting solutions to a general PDE - otherwise I wouldn't have a job! However, if any of you have taken a PDEs course, you know that there are specific forms of $\mathbf{N}$ that lead to explicit solutions. These methods typically follow the same structure:

$$\text{PDE} \mapsto \text{ODE} \mapsto \text{Algebra}$$

That is, we reduce a PDE to an ODE, then reduce the ODE to some algebra. Provided you can do this, you can follow backwards from your solution to the algebra to back to the PDE solution. Let's illustrate.

Self-Similar Solutions

Many important solutions to PDEs are self-similar: they keep the same profile but might get stretched or compressed in time and space. This method simply attempts to extract a relationship between time and space by making a change of variable to a new variable such as

$$\xi = t^\alpha x^\beta,$$

where the choices for α and β are not always obvious. You can see that the variable ξ couples space and time, and so a PDE in x and t reduces to an ODE in ξ.

Let's illustrate with the heat equation:

$$u_t = u_{xx}.$$

We saw this equation back when we were denoising images. The only difference is that now I will assume space is infinite: $x \in \mathbb{R}$. Introduce

$$\xi = t^{-\frac{1}{2}} x$$

and write $u(x,t) = u(\xi)$. Then,

$$\frac{\partial u}{\partial t} = \frac{\partial u}{\partial \xi} \cdot \frac{\partial \xi}{\partial t} = -\frac{x}{2t^{\frac{3}{2}}} u_\xi$$

and

$$\frac{\partial u}{\partial x} = t^{-\frac{1}{2}} u_\xi \implies \frac{\partial^2 u}{\partial x^2} = t^{-1} u_{\xi\xi}$$

Pugging these derivatives into the heat equation and collecting like terms gives the ODE in the ξ variable:

$$-\frac{\xi}{2} u_\xi = u_{\xi\xi}.$$

One solution comes from the error function, written erf, giving

$$u(\xi) = \operatorname{erf}(\xi/2) \implies u(x,t) = \operatorname{erf}\left(\frac{x}{2t^{\frac{1}{2}}}\right).$$

Separation of Variables

Another way we can get solutions is by assuming that the space and time parts of the solutions can be separated. This amounts to looking for solutions of the form

$$\mathbf{U}(x,t) = \mathbf{F}(t)\mathbf{G}(x).$$

When the function $\mathbf{N}$ is linear in $\mathbf{U}$, two ODEs result: one for the time dynamics $\mathbf{F}(t)$ and one for the space dynamics $\mathbf{G}(x)$. If the function $\mathbf{N}$ is nonlinear in $\mathbf{U}$, then separation cannot be achieved.

Let's illustrate with the heat equation again. Writing $u(x,t) = f(t)g(x)$ transforms the heat equation to

$$f'(t)g(x) = f(t)g''(x).$$

We can put the f's on the left side and the g's on the right side to get

$$\frac{f'(t)}{f(t)} = \frac{g''(x)}{g(x)}.$$

The above equation is satisfied *for every* x and t. But, the left contains only t's and the right contains only x's. Therefore, each side has to be equal to a constant, and the sides are equal to the same constant. Calling this (unknown) constant λ, we get two equations

$$\frac{f'(t)}{f(t)} = \lambda \implies f'(t) = \lambda f(t)$$

and

$$\frac{g''(x)}{g(x)} = \lambda \implies g''(x) = \lambda g(x).$$

The solution for $f(t)$ is given as

$$f(t) = a\mathrm{e}^{\lambda t}$$

where $a \in \mathbb{R}$ is any constant. With $\lambda < 0$ we have $f(t) \to 0$ as $t \to \infty$, so everything stays bounded as time goes on. Furthermore, with $\lambda < 0$ the solution(s) for $g(x)$ are sines and cosines:

$$g(x) = b\cos(\sqrt{|\lambda|}x) + c\sin(\sqrt{|\lambda|}x),$$

where $b, c \in \mathbb{R}$ are any constants. Putting this all together means that we have a solution for (at least) every $\lambda < 0$:

$$u(x,t) = A\mathrm{e}^{\lambda t}\cos(\sqrt{|\lambda|}x) + B\mathrm{e}^{\lambda t}\sin(\sqrt{|\lambda|}x)$$

where again A and B are any constants.

Eigenfunction Expansions

In the case of separation of variables, we saw that we had infinitely many solutions: one for each $\lambda < 0$. Furthermore, linearity of the heat equation means that we could take two of these solutions, add them together, and still get a solution to the heat equation. Generally, this leads to looking for *eigenfunction expansions* of solutions in the form

$$u(x,t) = \sum_{n=1}^{\infty} a_n(t)\phi_n(x).$$

The eigenfunction name comes from the idea that the a_n and the ϕ_n usually satisfy a differential eigenvalue problem. This is why I used λ above - it ends up being an eigenvalue and the eigenfunctions are the solutions $f(t)$ and $g(x)$ above.

We've already seen with POD that for the above expansion to make sense we need $\{\phi_n(x)\}_{n=1}^{\infty}$ to form an orthonormal basis. We could use POD to find the basis, or we could use the ones we already know. Two important examples of eigenfunctions come to mind immediately: Fourier modes and Chebyshev modes. These two types of modes/eigenfunctions are used a lot because calculating the $a_n(t)$ can be done quickly. You know this already with Fourier because the DFT takes $\mathcal{O}(N\log(N))$ speed to calculate the coefficients instead of the relatively slow $\mathcal{O}(N^2)$ speed of doing things one at a time.

The Nonlinear Schrödinger Equation

To better understand eigenvalue expansions, let's consider the nonlinear Schrödinger (NLS) equation

$$\mathrm{i}u_t + \frac{1}{2}u_{xx} + |u|^2u = 0,$$

where $u(x,t)$ is a complex-valued function of space x and time t. If the nonlinear term $|u|^2u$ wasn't on the end we could solve the equation using separation of variables, just like with the heat equation. However, if you tried to separate perform separation of variables you won't be able to isolate $f(t)$ and $g(x)$. This is because the nonlinearity $|u|^2u$ tangles or mixes the separated components, making a simple analytic solution not possible.

However, what is difficult for us on paper might not be difficult for the computer. We will take the Fourier transform of the NLS to get

$$\mathrm{i}\hat{u}_t - \frac{k^2}{2}\hat{u} + \widehat{|u^2|u} = 0$$

where the mixing of the Fourier modes comes from taking the Fourier transform of the cubic term $|u^2|u$. First, you can see that we turned the PDE into an ODE in time only by moving to the frequency domain. Second, it is hard for us to solve this, but for the computer the Fourier transform of the cubic term is just another Fourier transform to compute. It doesn't give an analytic answer, it just solves what it is asked to. Before we get to MATLAB, let's isolate for the time derivative term by multiplying through by $-\mathrm{i}$ and moving the other terms to the right side:

$$\hat{u}_t = -\frac{\mathrm{i}k^2}{2}\hat{u} + \mathrm{i}\widehat{|u^2|u}.$$

We are going to look at special solutions to the NLS called solitons. They all resemble sech functions and so our initial condition will be

$$u(x,0) = \mathrm{sech}(x).$$

Let's initialize everything in MATLAB.

```matlab
% Space
L = 40; n = 512;
x2 = linspace(-L/2,L/2,n+1); x = x2(1:n);
k = (2*pi/L)*[0:n/2-1 -n/2:-1]';

% Time
t = 0:0.1:10;

% Initial condition
u = sech(x);
ut = fft(u);
```

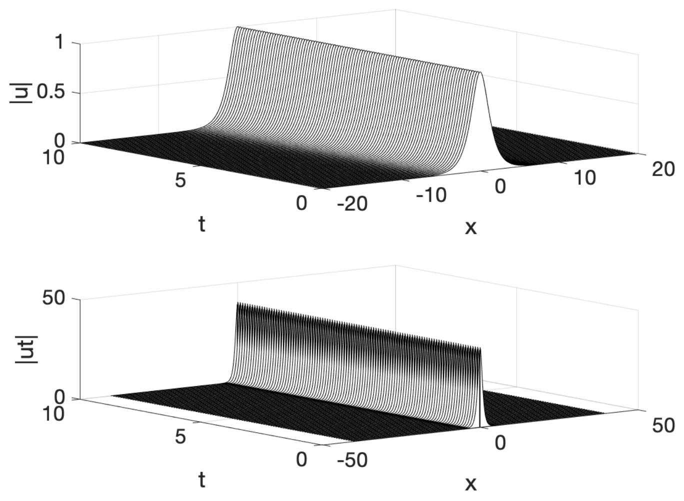
In Fourier space the NLS is an ODE in time, so we can use MATLAB's ode45 to simulate it.

```matlab
[t, utsol] = ode45(@(t,y) nls_rhs(t,y,k),t,ut);
for j = 1:length(t)
   usol(j,:) = ifft(utsol(j,:)); % back to x-space
end

% Solution
subplot(2,1,1), waterfall(x,t,abs(usol)), colormap([0 0 0])
xlabel('x')
```

```
ylabel('t')
zlabel('|u|')
set(gca,'FontSize',16)

% Solution in Fourier space
subplot(2,1,2), waterfall(k,t,abs(utsol))
xlabel('x')
ylabel('t')
zlabel('|ut|')
set(gca,'FontSize',16)
```

The function nls_rhs is the right-hand-side of the NLS ODE in frequency space and must be put at the bottom of your script.

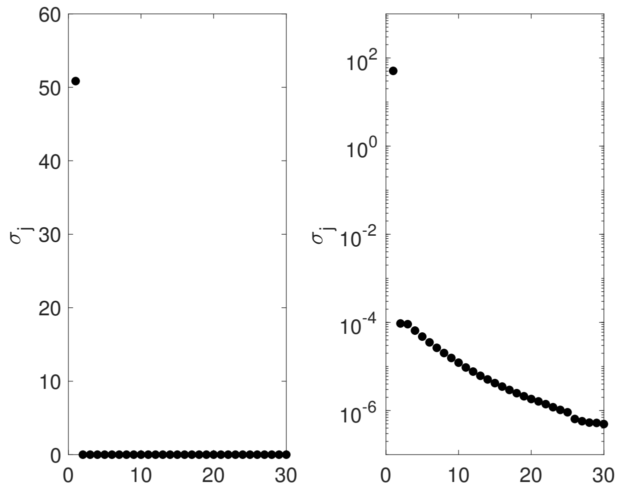
```
function rhs = nls_rhs(t,ut,k)
    u = ifft(ut);
    rhs = -(1i/2)*(k.^2).*ut + 1i*fft( (abs(u).^2).*u );
end
```



You can see from the plot that $|u|$ doesn't change in time, but we need about 50 nonzero Fourier modes to describe this simple little bump. So, the question is: is the soliton (bump) solution really a 50 degree of freedom solution? Certainly not! The problem is that we just don't have the right basis. Let's look at the POD modes obtained by taking the SVD.

```
% SVD
[U, S, V] = svd(u);

% Plot singular values
subplot(1,2,1), plot(diag(S),'k.','Markersize',20)
ylabel('\sigma_j')
set(gca,'FontSize',16,'Xlim',[0 30])

% And their logarithms
subplot(1,2,2), semilogy(diag(S),'k.','Markersize',20)
ylabel('\sigma_j')
set(gca,'FontSize',16,'Xlim',[0 30],'Ylim',[1e-7, 1e3])
```

Only the first singular value is of any significant - the rest are nearly zero. Hence, the solution is really just one-dimensional. The issue is, we chose a basis that required at least 50 degrees of freedom (the Fourier modes) to describe it. That means we are keeping a lot of information for something that is very low-dimensional.



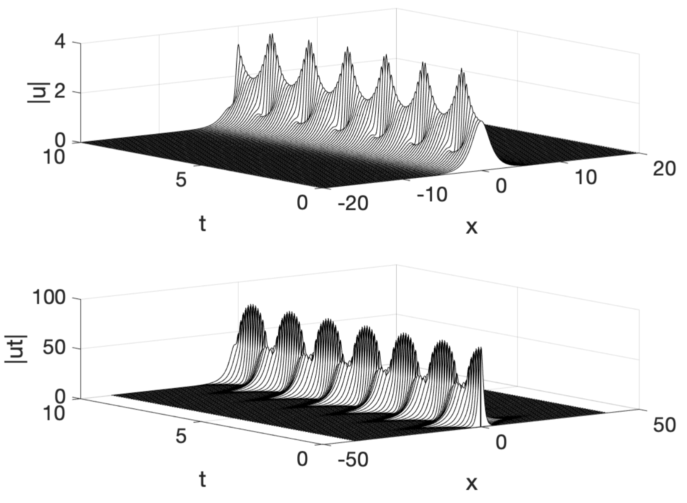
Let's try one more application of this. We will again look at solitons but this time use the initial condition

$$u(x,0) = 2\text{sech}(x).$$

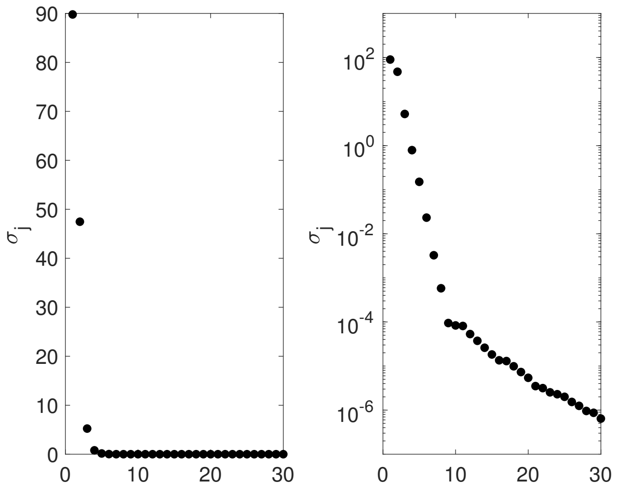
```
% Initial condition
u = 2*sech(x);
ut = fft(u);

[t, utsol] = ode45(@(t,y) nls_rhs(t,y,k),t,ut);
for j = 1:length(t)
   usol(j,:) = ifft(utsol(j,:)); % back to x-space
end

% Solution
subplot(2,1,1), waterfall(x,t,abs(usol)), colormap([0 0 0])
xlabel('x')
ylabel('t')
zlabel('|u|')
set(gca,'FontSize',16)

% Solution in Fourier space
subplot(2,1,2), waterfall(k,t,abs(utsol))
xlabel('x')
ylabel('t')
zlabel('|ut|')
set(gca,'FontSize',16)
```

The solution again retains the same basic shape of the sech function, but now the peaks oscillate. This time we need about 200 Fourier modes to describe the solution, but looking at the SVD reveals the low-dimensional nature of the solution. It appears that the solution is little more than three-dimensional. In the next lecture we will discuss the following question:

Question: If the dynamics are actually low-dimensional, how do we find the appropriate basis to expand them in?



|u|
4
2
0
10
5
0
t
-20
-10
0
10
20
x
|ut|
100
50
0
10
5
0
t
-50
0
50
x



σ_j
90
80
70
60
50
40
30
20
10
0
0
10
20
30
σ_j
10^{2}
10^{0}
10^{-2}
10^{-4}
10^{-6}
0
10
20
30