Lecture 25
PDE Dynamics in the Right (Best) Basis

Jason J. Bramburger

Last lecture we saw that POD can be used to identify low-dimensional dynamics in high-dimensional systems. In this lecture we are going to see that if we can use the SVD to find the POD modes, then we can use them to find the ODEs for the coefficients too. This would allow us to understand not just the original simulation, but with different initial conditions on the coefficients we could quickly and easily simulate other solutions too.

We will continue with our discussion of the nonlinear Schrödinger equation

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

We will start with the steady soliton solution and then make our way to the oscillatory one.

Steady Soliton Reduction

We saw with the initial condition $u(x,0) = \mathrm{sech}(x)$ the solution to the NLS retains the same shape for all time. It is steady. The SVD told us that this solution is governed by only one POD mode, which we will denote $\phi(x)$. Thus the dynamics of the NLS can be recast as

$$u(x,t) = a(t)\phi(x)$$

where we only have one term in our eigenvalue expansion since there is only one mode. CAREFUL: u is complex, and so a is also complex!

Plugging this form for the solution into the NLS gives

$$\mathrm{i}a_t\phi + \frac{1}{2}a\phi_{xx} + |a|^2 a|\phi|^2\phi = 0$$

We want to project the dynamics onto the mode ϕ using the inner product

$$\langle \phi, \psi \rangle = \int_{-\infty}^{\infty} \phi(x)\psi^*(x)\mathrm{d}x$$

where the $*$ denotes complex conjugation. Taking the inner product of the NLS with ϕ gives an ODE for $a(t)$:

$$\mathrm{i}a_t + \frac{\alpha}{2}a + \beta|a|^2 a = 0,$$

where α and β are the projections

$$\alpha = \frac{\langle \phi_{xx}, \phi \rangle}{\langle \phi, \phi \rangle}, \qquad \beta = \frac{\langle |\phi|^2\phi, \phi \rangle}{\langle \phi, \phi \rangle}.$$

The differential equation for $a(t)$ can be solved explicitly to yield

$$a(t) = a(0)\exp\left[\mathrm{i}\frac{\alpha}{2}t + \mathrm{i}\beta|a(0)|^2 t\right].$$

To get the initial condition, recall that

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

So, taking the inner product of the middle equation and the right equation with ϕ allows us to isolate for $a(0)$:

$$a(0) = \frac{\langle \mathrm{sech}(x), \phi \rangle}{\langle \phi, \phi \rangle}$$

Therefore, the one-mode expansion gives the approximate PDE solution

$$u(x,t) = a(0)\exp\left[\mathrm{i}\frac{\alpha}{2}t + \mathrm{i}\beta|a(0)|^2 t\right]\phi(x).$$

Meaning: We can see that $|u(x,t)|$ with our approximate solution above just gives $|a(0)\phi(x)|$, which is constant in time. The exponential part is effecting the *phase* of the complex-valued solution. It doesn't show up when we plot the modulus. This is very typical of solutions to the NLS and unfortunately it gets missed when first learning the NLS since we always just plot the modulus of the solution.

Let's see how our approximate solution compares with the true solution:

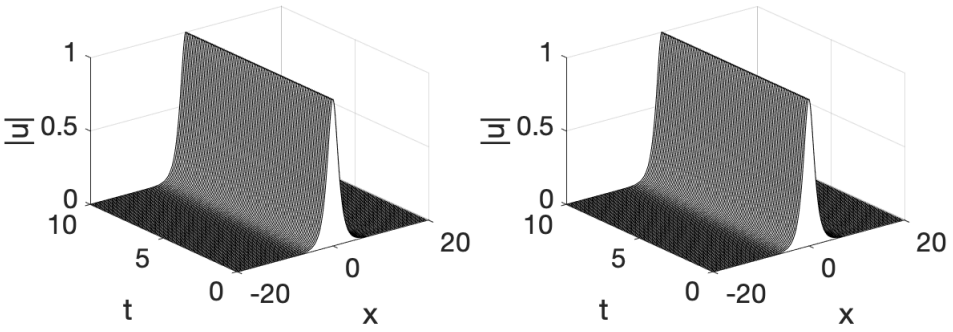
```
% Space
L = 40; n = 512;
x2 = linspace(-L/2,L/2,n+1); x = x2(1:n);
k = (2*pi/L)*[0:n/2-1 -n/2:-1]';

% Time
t = 0:0.1:10;

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

[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

% Obtain the POD mode
[U, S, V] = svd(usol);

% Steady soliton match
phi_xx = ifft(-(k.^2).*fft(V(:,1)));
norm = trapz(x,V(:,1).*conj(V(:,1)));
A0 = trapz(x,sech(x)'.*conj(V(:,1)))/norm;
alpha = trapz(x,phi_xx.*conj(V(:,1)))/norm;
beta = trapz(x,(V(:,1).*conj(V(:,1))).^2)/norm;

a = A0*exp(1i*0.5*alpha*t + 1i*beta*t*abs(A0)^2);

% Plot solitons
subplot(1,2,1), waterfall(x,t,abs(usol)), colormap([0 0 0])
xlabel('x')
ylabel('t')
zlabel('|u|')
set(gca,'FontSize',16)
subplot(1,2,2), waterfall(x,t,abs(a*V(:,1)')), colormap([0 0 0])
xlabel('x')
ylabel('t')
zlabel('|u|')
set(gca,'FontSize',16)
```

We did a pretty good job! Visually it's impossible to tell the difference.

Oscillatory Soliton Reduction

We also saw in the previous lecture that the initial condition $u(x,0) = 2\mathrm{sech}(x)$ leads to soliton solutions whose peaks oscillator up and down in time. We will look at the two mode approximation of these solutions. Set

$$u(x,t) = a_1(t)\phi_1(x) + a_2(t)\phi_2(x),$$



where $\phi_1(x)$ and $\phi_2(x)$ are the dominant POD modes found using the SVD. Plugging this into the NLS gives

$$\mathrm{i}(a_{1t}\phi_1 + a_{2t}\phi_2) + \frac{1}{2}(a_1\phi_{1xx} + a_2\phi_{2xx}) + (a_1\phi_1 + a_2\phi_2)^2(a_1^*\phi_1^* + a_2^*\phi_2^*) = 0.$$

Expanding this out and projecting onto each of the POD modes, ϕ_1 and ϕ_2, results in the system of ODEs that governs a_1 and a_2:

$$\begin{aligned}
&\mathrm{i}a_{1t} + \alpha_{11}a_1 + \alpha_{12}a_2 + (\beta_{111}|a_1|^2 + 2\beta_{211}|a_2|^2)a_1 \\
&\quad + (\beta_{121}|a_1|^2 + 2\beta_{221}|a_2|^2)a_2 + \sigma_{121}a_1^2a_2^* + \sigma_{211}a_2^2a_1^* = 0 \\
&\mathrm{i}a_{2t} + \alpha_{21}a_1 + \alpha_{22}a_2 + (\beta_{112}|a_1|^2 + 2\beta_{212}|a_2|^2)a_1 \\
&\quad + (\beta_{122}|a_1|^2 + 2\beta_{222}|a_2|^2)a_2 + \sigma_{122}a_1^2a_2^* + \sigma_{212}a_2^2a_1^* = 0
\end{aligned}$$

where

$$\begin{aligned}
\alpha_{jk} &= \langle \phi_{jxx}, \phi_k \rangle / 2 \\
\beta_{jkl} &= \langle |\phi_j|^2\phi_k, \phi_l \rangle \\
\sigma_{jkl} &= \langle \phi_j^2\phi_k^*, \phi_l \rangle.
\end{aligned}$$

Similar to the steady soliton case, the initial conditions are given by

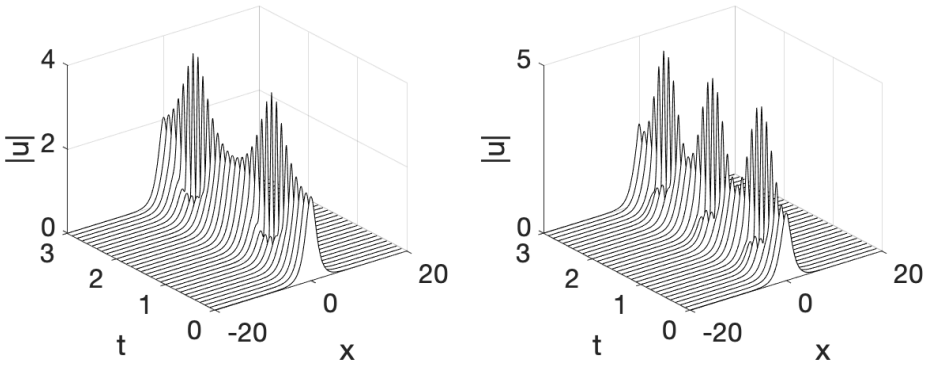
$$a_1(0) = \frac{\langle 2\mathrm{sech}(x), \phi_1 \rangle}{\langle \phi_1, \phi_1 \rangle}, \qquad a_2(0) = \frac{\langle 2\mathrm{sech}(x), \phi_2 \rangle}{\langle \phi_2, \phi_2 \rangle}.$$

We can't solve the system by hand anymore, but we can use MATLAB to see what the $a_1(t)$ and $a_2(t)$ look like.

```
clear all; close all; clc

% 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 = 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

% Obtain the POD modes
[U, S, V] = svd(usol');
```

```
22
23 % compute the second derivatives
24 phi_1_xx = ifft( -(k.^2).*fft(U(:,1)) );
25 phi_2_xx = ifft( -(k.^2).*fft(U(:,2)) );
26
27 % compute the norms of the SVD modes
28
29 norm1=trapz(x,U(:,1).*conj(U(:,1)));
30 norm2=trapz(x,U(:,2).*conj(U(:,2)));
31
32 % compute the initial conditions
33
34 A0_1=trapz(x,(2*sech(x).').*conj(U(:,1)))/norm1;
35 A0_2=trapz(x,(2*sech(x).').*conj(U(:,2)))/norm2;
36
37 % compute inner products alpha, beta, sigma
38 alpha11=0.5*trapz(x,conj(U(:,1)).*phi_1_xx)/norm1;
39 alpha12=0.5*trapz(x,conj(U(:,1)).*phi_2_xx)/norm1;
40 beta11_1=trapz(x,(U(:,1).*conj(U(:,1))).^2)/norm1;
41 beta22_1=trapz(x,(U(:,2).*(abs(U(:,2)).^2).*conj(U(:,1))))/norm1;
42 beta21_1=trapz(x,(U(:,1).*(abs(U(:,2)).^2).*conj(U(:,1))))/norm1;
43 beta12_1=trapz(x,(U(:,2).*(abs(U(:,1)).^2).*conj(U(:,1))))/norm1;
44 sigma12_1=trapz(x,(conj(U(:,2)).*(U(:,1).^2).*conj(U(:,1))))/norm1;
45 sigma21_1=trapz(x,(conj(U(:,1)).*(U(:,2).^2).*conj(U(:,1))))/norm1;
46
47 alpha21=0.5*trapz(x,conj(U(:,2)).*phi_1_xx)/norm2;
48 alpha22=0.5*trapz(x,conj(U(:,2)).*phi_2_xx)/norm2;
49 beta11_2=trapz(x,(U(:,1).*(abs(U(:,1)).^2).*conj(U(:,2))))/norm2;
50 beta22_2=trapz(x,(U(:,2).*(abs(U(:,2)).^2).*conj(U(:,2))))/norm2;
51 beta21_2=trapz(x,(U(:,1).*(abs(U(:,2)).^2).*conj(U(:,2))))/norm2;
52 beta12_2=trapz(x,(U(:,2).*(abs(U(:,1)).^2).*conj(U(:,2))))/norm2;
53 sigma12_2=trapz(x,(conj(U(:,2)).*(U(:,1).^2).*conj(U(:,2))))/norm2;
54 sigma21_2=trapz(x,(conj(U(:,1)).*(U(:,2).^2).*conj(U(:,2))))/norm2;
55
56 y0=[A0_1; A0_2];
57 [t,u_ode]=ode45(@(t,y) nls_ode_rhs(t,y,alpha11,alpha12,alpha21,alpha22, ...
58 beta11_1,beta22_1,beta12_1,beta21_1,beta11_2,beta22_2,beta21_2,beta12_2,...
59 sigma12_1,sigma21_1,sigma12_2,sigma21_2),t,y0);
60
61
62 % Plot solitons
63 subplot(1,2,1), waterfall(x,t,abs(usol)), colormap([0 0 0])
64 xlabel('x')
65 ylabel('t')
66 zlabel('|u|')
67 set(gca,'FontSize',16,'Ylim',[0 3])
68 subplot(1,2,2), waterfall(x,t,abs(u_ode(:,1)*U(:,1)' + u_ode(:,2)*U(:,2)')), colormap([0 0 0])
69 xlabel('x')
70 ylabel('t')
71 zlabel('|u|')
72 set(gca,'FontSize',16,'Ylim',[0 3])
```

```
1  function rhs = nls_ode_rhs(t,y0,alpha11,alpha12,alpha21,alpha22, ...
2      beta11_1,beta22_1,beta12_1,beta21_1,beta11_2,beta22_2,beta21_2,beta12_2,...
3      sigma12_1,sigma21_1,sigma12_2,sigma21_2)
4
5  rhs= [ 1i*( alpha11*y0(1)+alpha12*y0(2)+(beta11_1*abs(y0(1))^2+2*beta21_1*abs(y0(2))^2)*y0(1) ...
6      +(beta12_1*abs(y0(1))^2+2*beta22_1*(abs(y0(2))^2))*y0(2) ...
7      + sigma12_1*y0(1)^2*conj(y0(2)) + sigma21_1*y0(2)^2*conj(y0(1)) );
8      1i*( alpha21*y0(1)+alpha22*y0(2)+(beta11_2*abs(y0(1))^2+2*beta21_2*abs(y0(2))^2)*y0(1) ...
9      +(beta12_2*abs(y0(1))^2+2*beta22_2*(abs(y0(2))^2))*y0(2) ...
10     + sigma12_2*y0(1)^2*conj(y0(2)) + sigma21_2*y0(2)^2*conj(y0(1)) )];
11 end
```



Our 2 mode approximation is far from perfect. Indeed, it seems that the temporal periodic is sped up. On top of this, if you look at the u_ode components you'll notice a slow drift, meaning they become less accurate as time goes on. This could be remedied by including the third mode in the system, but things will get significantly messier than they already are.

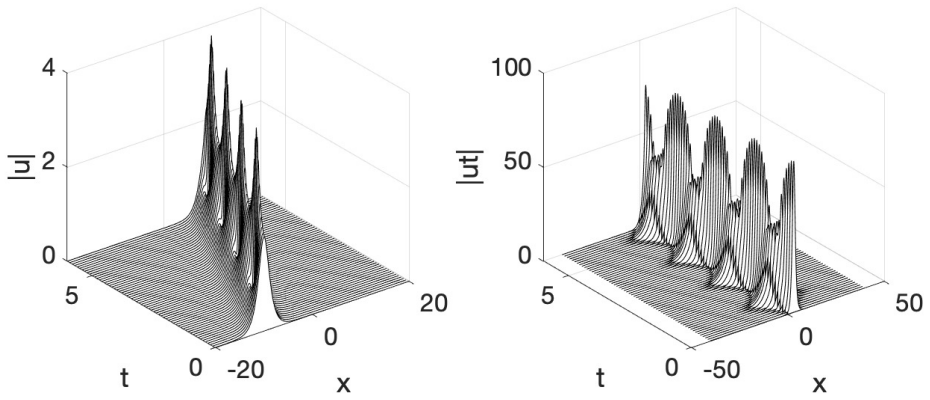
Symmetries and Invariances

So far we have just seen solutions that move in time, but not space. When we have solutions that move in space as well, the SVD starts to break down. As an example, we can consider the NLS with the initial condition

$$u(x,0) = 2\text{sech}(x+10)\text{e}^{\text{i}\pi x}.$$

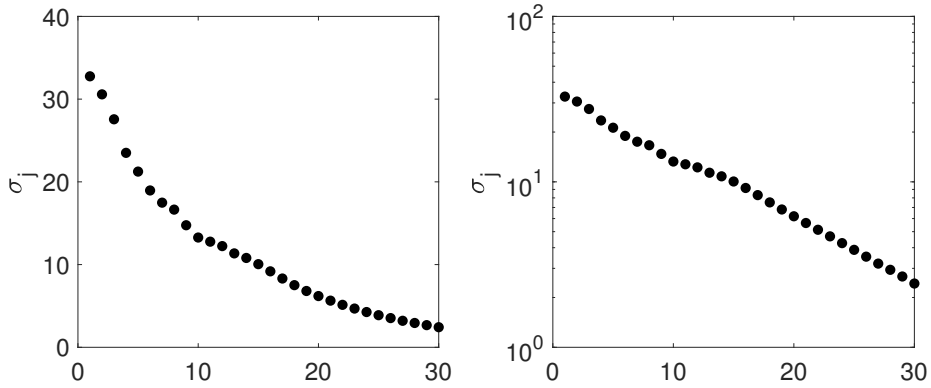
This is similar to our usual sech initial conditions, just shifted 10 units left and shifted to be centred at $k = \pi$ in frequency space. Let's see what the soliton evolution looks like now.

```
clear all; close all; clc

% 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 = 0:0.1:6;

% Initial condition
u = 2*sech(x+10).*exp(1i*pi*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(1,2,1), waterfall(x,t,abs(usol)), colormap([0 0 0])
xlabel('x')
ylabel('t')
zlabel('|u|')
set(gca,'FontSize',16)

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



Now the solution moves from left to right as time goes on. But, it basically looks like the oscillating soliton solution that remained centred around zero. The difference is what the SVD gives you.

```
[U, S, V] = svd(usol');

% 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',[1, 1e3])
```



That's not good... But what went wrong? The solution is basically just the original oscillating soliton, just moving left to right. Shouldn't it have just a few dominant modes again?

The reason the method failed in this case is due to the translational invariance. If the movement in the x-direction is removed, then we get back the usual oscillating soliton and things are good. The idea is to centre your space not at 0, but at the centre of your soliton. This can be achieved by considering the solution $u(x,t)$ rewritten as

$$u(x,t) \to u(x - c(t), t)$$

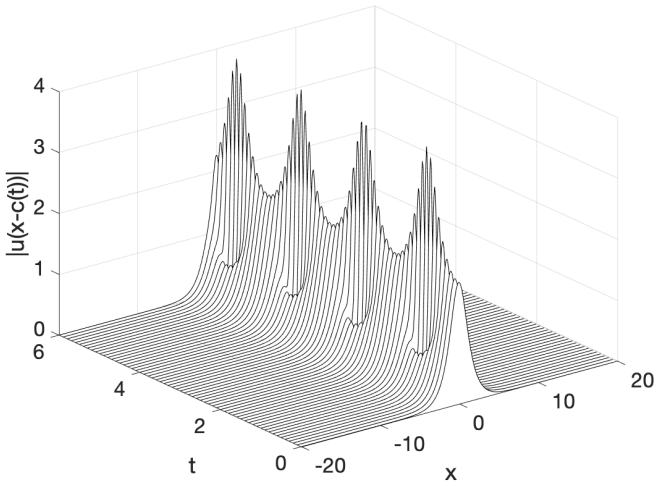
where $c(t)$ corresponds to the centre of the moving soliton. This essentially drags the coordinate frame with the moving soliton so that it looks like it's standing still when we move with it. This is the same as seeing a fast car go down the street from your left to right, but if you're driving at the same speed beside the car it doesn't look like it is moving.

We can find $c(t)$ by doing a simple centre-of-mass calculation with the following code.

```
for j = 1:length(t)
    com = x.*abs(usol(j,:)).^2;
    com2 = abs(usol(j,:)).^2;
    c(j) = trapz(x,com)/trapz(x,com2);
end
```

Now that we have the centre, we can put ourselves in a moving coordinate frame with the following code.

```
for j = 1:length(t)
    [mn,jj] = min(abs(x - c(j)));
    ns = n/2 - jj;
    ushift = [usol(j,:) usol(j,:) usol(j,:)];
    usol_shift(j,:) = ushift(1,n+1-ns:2*n-ns);
end

% Plot the result
waterfall(x,t,abs(usol_shift)), colormap([0 0 0])
xlabel('x')
ylabel('t')
zlabel('|u(x-c(t))|')
set(gca,'FontSize',16)
```



There we go! We have our oscillating soliton that is fixed in space back. Now we can find the POD modes using the SVD again.