Lecture 27
Dynamics of DMD Versus POD

Jason J. Bramburger

In the last lecture we went through the theory of DMD, so now we are going to put it into practice! We are going to work with the nonlinear Schrödinger equation (NLS) again, given by

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

Recall that with the initial condition $u(x,0) = 2\mathrm{sech}(x)$ the solution is a soliton with height that oscillates at a steady frequency. Here's something you should keep in mind: the NLS is nonlinear (it's even in the name), but the dynamics are just oscillatory. We can capture oscillations with linear ODEs. So, at least in theory, the DMD should work well here since it turns nonlinear data into solutions to linear ODEs.

We are going to start by generating some synthetic data. That is, we will simulate the NLS with the sech initial condition to arrive at our data.

```
% 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
slices = 20;
t = linspace(0,2*pi,slices+1); dt = t(2) - t(1);

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

% Simulating the NLS
[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
```

We again use the function nls_rhs to capture the right-hand-side of the NLS.

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

The usol matrix in the above code arranges the data into $M = 21$ time slices and $N = 512$ data points at each slice. This constitutes the matrix $\mathbf{X}$ used in the previous lecture to define DMD.

Following the algorithm from the end of the previous lecture, our first step is to create the matrices $\mathbf{X}_1^{M-1}$ and $\mathbf{X}_2^M$.

```
X = usol';
X1 = X(:,1:end-1);
X2 = X(:,2:end);
```

The next step is to obtain the SVD decomposition of $\mathbf{X}_1^{M-1}$. This will allow us to find the eigenvalues of $\tilde{\mathbf{S}}$.

```
[U, Sigma, V] = svd(X1,'econ');
S = U'*X2*V*diag(1./diag(Sigma));
[eV, D] = eig(S); % compute eigenvalues + eigenvectors
mu = diag(D); % extract eigenvalues
omega = log(mu)/dt;
Phi = U*eV;
```

The vector **mu** contains the DMD eigenvalues. Remember, we use these eigenvalues to convert to the **omega** vector, which are the just the eigenvalues in another form. The reason for the different form is that we want to make the DMD solution look more like a solution to a continuous time ODE. With this in mind, the **Phi** vector contains the eigenvectors of the matrix $\mathbf{A}$, which we obtain by multiplying the eigenvectors of $\tilde{\mathbf{S}}$ against $\mathbf{U}$.

The last thing to do is create the DMD solution.

```
y0 = Phi\X1(:,1); % pseudoinverse to get initial conditions

u_modes = zeros(length(y0),length(t));
for iter = 1:length(t)
   u_modes(:,iter) = y0.*exp(omega*t(iter));
end
u_dmd = Phi*u_modes;
```

Let's plot the PDE and DMD solutions to see how they compare.

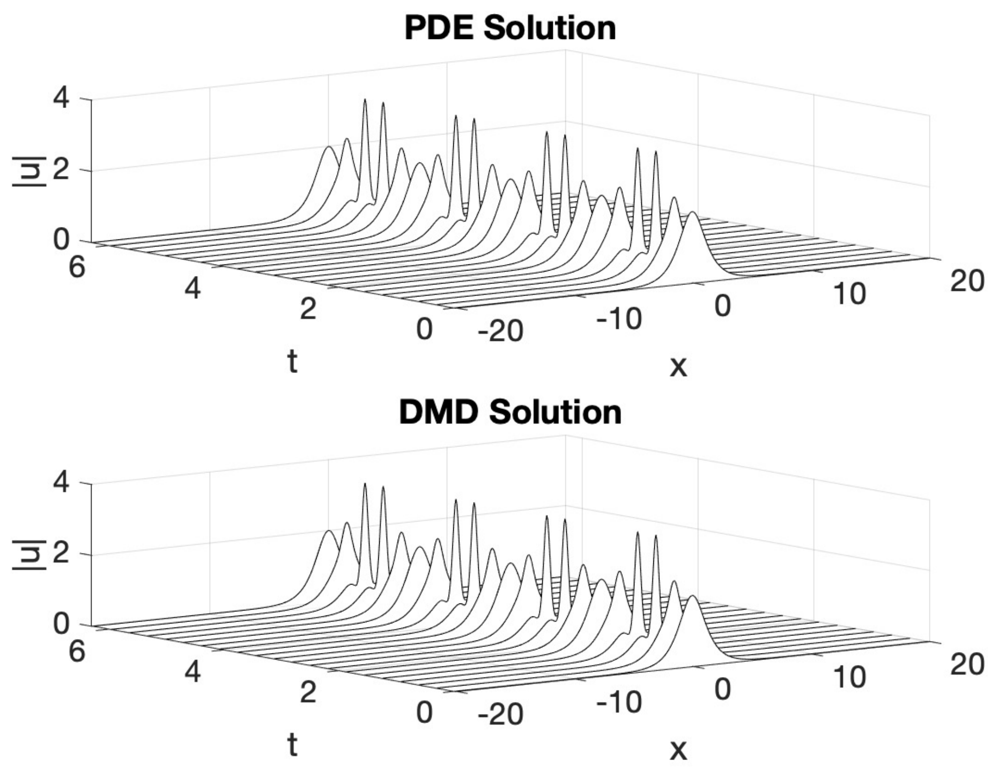
```
% PDE Solution
subplot(2,1,1), waterfall(x,t,abs(usol)), colormap([0 0 0])
xlabel('x')
ylabel('t')
zlabel('|u|')
title('PDE Solution')
set(gca,'FontSize',16)

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

That's pretty good! It's hard to tell the difference! Let's plot the DMD modes too, just to see what they look like.

```
waterfall(x,1:slices,abs(Phi')), colormap([0 0 0])
xlabel('x')
ylabel('modes')
zlabel('|u|')
title('DMD Modes')
set(gca,'FontSize',16)
```

They're fairly complicated, but they seem to get the job done! It's also worth looking at what the eigenvalues ω_k are. I am going to be looking at the ω instead of the μ eigenvalues since I think it will be easier for us to derive meaning from the former. Importantly, if the real part of ω_k is positive, then $e^{\omega_k t}$ is going to blow-up as t gets large. Conversely, if the real part of ω_k is negative, then the associated exponential function will go to zero as $t \to \infty$. Finally, if the real part of ω_k is zero, then the exponential function $e^{\omega_k t}$ is really just sines and cosines, and therefore will just oscillate forever.

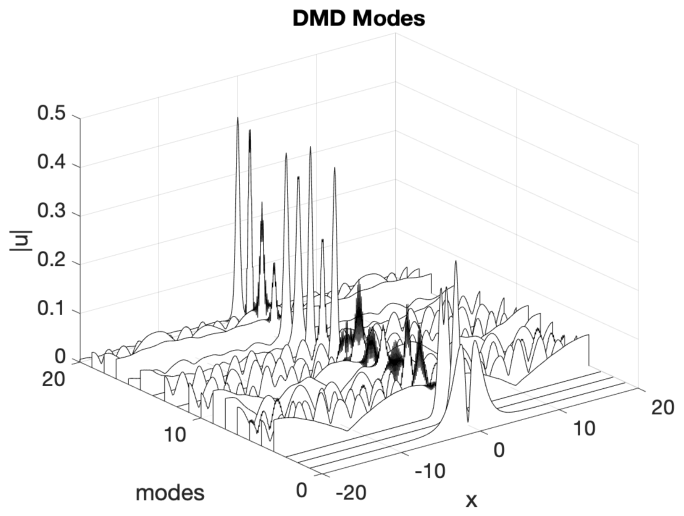
Let's plot our eigenvalues ω_k to see if there are any with positive real part. We will plot $\omega_k \cdot \Delta t$ since the division by Δt is just to scale the time in the DMD solution. The $\omega_k \cdot \Delta t$ are just the logarithm of the eigenvalues of the μ_k, the eigenvalues of $\mathbf{A}$.

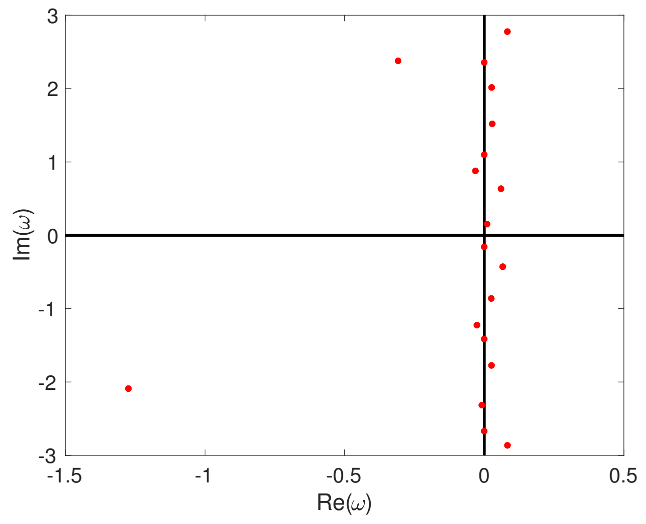


```
% make axis lines
line = -50:50;

plot(zeros(length(line),1),line,'k','Linewidth',2) % imaginary axis
hold on
plot(line,zeros(length(line),1),'k','Linewidth',2) % real axis
plot(real(omega),imag(omega),'r.','Markersize',15)
xlabel('Re(\omega)')
ylabel('Im(\omega)')
set(gca,'FontSize',16,'Xlim',[-5 0.5],'Ylim',[-12 12])
```

You can see that there are some eigenvalues ω_k that have positive real part, meaning that if we run the DMD solution long enough it will eventually blow up. Let's forecast the DMD solution all the way up to $t = 100$.

```
t = 0:dt:100;

u_modes = zeros(length(y0),length(t));
for iter = 1:length(t)
   u_modes(:,iter) = y0.*exp(omega*t(iter));
end
u_dmd = Phi*u_modes;

% DMD Solution
subplot(2,1,1), waterfall(x,t(1:100),abs(u_dmd(:,1:100)')), colormap([0 0 0])
xlabel('x')
ylabel('t')
zlabel('|u|')
title('DMD Solution')
set(gca,'FontSize',16)

% Far in the future
subplot(2,1,2), waterfall(x,t(200:end),abs(u_dmd(:,200:end)')), colormap([0 0 0])
xlabel('x')
ylabel('t')
```





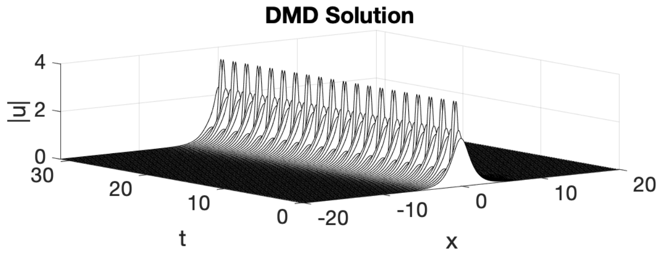
```
zlabel('|u|')
title('DMD Solution')
set(gca,'FontSize',16)
```

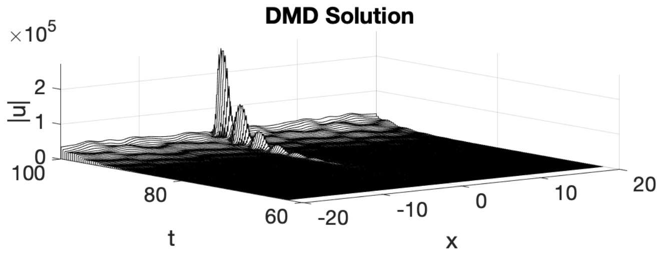
With just the eyeball test we can see that the DMD solution does really well for a long time. We can see that it looks almost perfect up to at least $t = 30$. The second plot shows the solution for $t \in [60, 100]$. Here t is big enough that those modes corresponding to the eigenvalues with positive real part have been able to blow up. Look carefully at the vertical axis - it's on the order of hundreds of thousands!

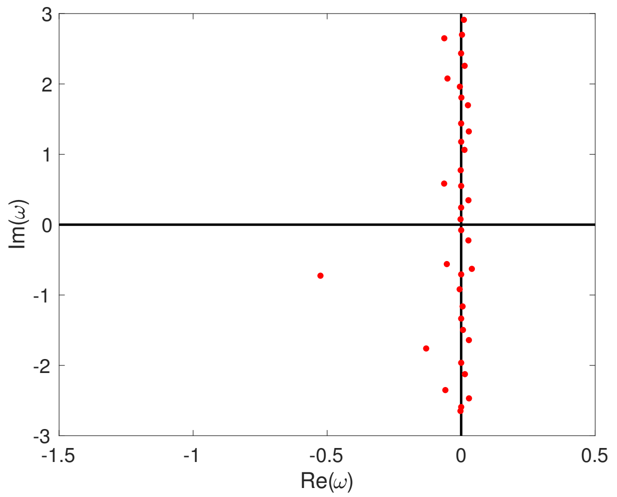
Taking smaller Δt will result in the eigenvalues with positive real part moving toward the imaginary axis, thus becoming stable. This amounts to just taking more slices and is due to the fact that when we have large Δt we are skipping over a lot of information that happens between timesteps. The DMD algorithm tries to interpolate this big jump as best as possible, and we see that it results in unstable modes - modes with eigenvalues with positive real part. In the final image of this document we present two more plots of the eigenvalues $\omega_k \cdot \Delta t$. On the left we use $M = 41$ slices and on the right we use $M = 101$ slices. You can see the eigenvalues converging onto the imaginary axis, representing the periodic motion of the PDE solution.

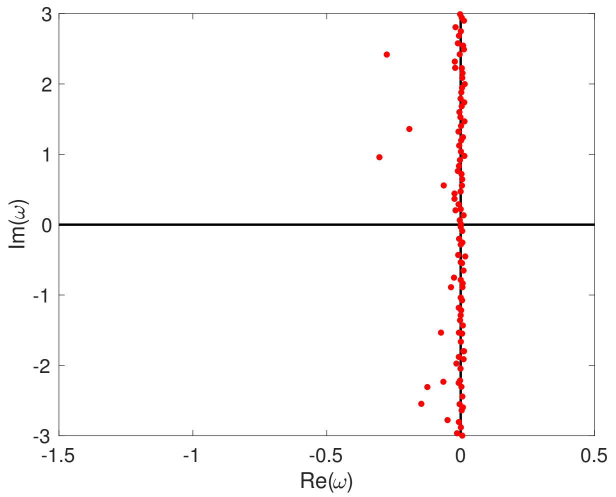
DMD and Control

In the above example the unstable modes were spurious. But, in many physical systems there may indeed be modes that grow exponentially in time, representing an instability in the system. The DMD framework is then a natural way to control the behaviour of your system and prevent such a blow-up. That is, the DMD algorithm tells you which modes are going to grow, so we can place a control algorithm on the complex physical system with









the aim of suppressing the pernicious growth mode(s).

<u>DMD and POD</u>

In the title of the lecture I promised I would compare DMD and POD, so here we are. Let's start with POD. With POD we create a low-dimensional basis by applying the SVD to the snapshot matrix $\mathbf{X}$. The dynamics of the governing equations are then projected onto this reduced basis and the resulting nonlinear dynamical system for the mode amplitudes is evolved forward in time. With DMD we assume that the dynamics are linear so that an exact solution can be constructed from the snapshots. Thus there is no need to simulate a dynamical system forward in time. We already have a solution we just need to plug t into. In this way, the POD method retains the *nonlinear dynamics* of the system through its projection of the governing equations onto the low-dimensional set of modes. DMD attempts to construct the Koopman operator/matrix that best approximates the nonlinear dynamics. You should also keep in mind that if the governing equations are unknown (as they usually are), then DMD is optimal since it doesn't require them. Remember: the POD method was created for fluid models where the equations are extremely difficult, but have been known for over a century.