12 Interpolation for Parametric ROMs

In the last chapter, the mathematical framework of ROMs was outlined. Specifically, Chapter 11 has already highlighted the POD method for projecting PDE dynamics to low-rank subspaces where simulations of the governing PDE model can be more readily evaluated. However, the complexity of projecting into the low-rank approximation subspace remains challenging due to the nonlinearity. Interpolation in combination with POD overcomes this difficulty by providing a computationally efficient method for discretely (sparsely) sampling and evaluating the nonlinearity. This chapter leverages the ideas of the sparse and compressive sampling algorithms of Chapter 3 where a small number of samples are capable of reconstructing the low-rank dynamics of PDEs. Ultimately, these methods ensure that the computational complexity of ROMs scale favorably with the rank of the approximation, even for complex nonlinearities. The primary focus of this chapter is to highlight sparse interpolation methods that enable a rapid and low dimensional construction of the ROMs. In practice, these techniques dominate the ROM community since they are critically enabling for evaluating parametrically dependent PDEs where frequent ROM model updates are required.

12.1 Gappy POD

The success of nonlinear model order reduction is largely dependent upon two key innovations: (i) the well-known POD-Galerkin method [251, 57, 542, 543], which is used to project the high-dimensional nonlinear dynamics onto a low-dimensional subspace in a principled way, and (ii) sparse sampling of the state space for interpolating the nonlinear terms required for the subspace projection. Thus sparsity is already established as a critically enabling mathematical framework for model reduction through methods such as gappy POD and its variants [179, 555, 565, 120, 159]. Indeed, efficiently managing the computation of the nonlinearity was recognized early on in the ROMs community, and a variety of techniques were proposed to accomplish this task. Perhaps the first innovation in sparse sampling with POD modes was the technique proposed by Everson and Sirovich for which the gappy POD moniker was derived [179]. In their sparse sampling scheme, random measurements were used to approximate inner products. Principled selection of the interpolation points, through the gappy POD infrastructure [179, 555, 565, 120, 159] or missing point (best points) estimation (MPE) [400, 21], were quickly incorporated into ROMs to improve performance. More recently, the empirical interpolation method (EIM) [41] and its most successful variant, the POD-tailored discrete empirical interpolation method (DEIM) [127], have provided a greedy algorithm that allows for nearly optimal reconstructions of nonlinear terms of the original high-dimensional system. The

DEIM approach combines projection with interpolation. Specifically, DEIM uses selected interpolation indices to specify an interpolation-based projection for a nearly optimal ℓ_2 subspace approximating the nonlinearity.

The low-rank approximation provided by POD allows for a reconstruction of the solution $\mathbf{u}(x,t)$ in (12.9) with r measurements of the n-dimensional state. This viewpoint has profound consequences on how we might consider measuring our dynamical system [179]. In particular, only $r \ll n$ measurements are required for reconstruction, allowing us to define the sparse representation variable $\tilde{\mathbf{u}} \in \mathbb{C}^r$

$$\tilde{\mathbf{u}} = \mathbf{P}\mathbf{u} \tag{12.1}$$

where the measurement matrix $\mathbf{P} \in \mathbb{R}^{r\times n}$ specifies r measurement locations of the full state $\mathbf{u} \in \mathbb{C}^n$. As an example, the measurement matrix might take the form

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & \cdots & & & & & & \cdots & 0 \\ 0 & \cdots & 0 & 1 & 0 & \cdots & & & \cdots & 0 \\ 0 & \cdots & & & \cdots & 0 & 1 & 0 & \cdots & 0 \\ \vdots & 0 & & & \cdots & 0 & 0 & 1 & \cdots & \vdots \\ 0 & \cdots & & & \cdots & 0 & 0 & 0 & \cdots & 1 \end{bmatrix} \tag{12.2}$$

where measurement locations take on the value of unity and the matrix elements are zero elsewhere. The matrix $\mathbf{P}$ defines a projection onto an r-dimensional space $\tilde{\mathbf{u}}$ that can be used to approximate solutions of a PDE.

The insight and observation of (12.1) forms the basis of the *gappy POD* method introduced by Everson and Sirovich [179]. In particular, one can use a small number of measurements, or gappy data, to reconstruct the full state of the system. In doing so, we can overcome the complexity of evaluating higher-order nonlinear terms in the POD reduction.

Sparse Measurements and Reconstruction

The measurement matrix $\mathbf{P}$ allows for an approximation of the state vector $\mathbf{u}$ from r measurements. The approximation is obtained by using (12.1) with the standard POD projection:

$$\tilde{\mathbf{u}} \approx \mathbf{P}\sum_{k=1}^{r} \tilde{a}_k \boldsymbol{\psi}_k \tag{12.3}$$

where the coefficients $\tilde{a}_k$ minimize the error in approximation: $\|\tilde{\mathbf{u}} - \mathbf{P}\mathbf{u}\|$. The challenge now is how to determine the $\tilde{a}_k$ given that taking inner products of (12.3) can no longer be performed. Specifically, the vector $\tilde{\mathbf{u}}$ has dimension r whereas the POD modes have dimension n, i.e. the inner product requires information from the full range of $\mathbf{x}$, the underlying discretized spatial variable, which is of length n. Thus, the modes $\boldsymbol{\psi}_k(x)$ are in general not orthogonal over the r-dimensional support of $\tilde{\mathbf{u}}$. The support will be denoted as $s[\tilde{\mathbf{u}}]$. More precisely, orthogonality must be considered on the full range versus the support space. Thus the following two relationships hold

$$M_{kj} = \langle \boldsymbol{\psi}_k, \boldsymbol{\psi}_j \rangle = \delta_{kj} \tag{12.4a}$$

$$M_{kj} = \langle \boldsymbol{\psi}_k, \boldsymbol{\psi}_j \rangle_{s[\tilde{\mathbf{u}}]} \neq 0 \quad \text{for all } k, j \tag{12.4b}$$

where M_{kj} are the entries of the Hermitian matrix $\mathbf{M}$ and δ_{kj} is the Kroenecker delta function. The fact that the POD modes are not orthogonal on the support $s[\tilde{\mathbf{u}}]$ leads us to consider alternatives for evaluating the vector $\tilde{\mathbf{a}}$.

To determine the $\tilde{a}_k$, a least-squares algorithm can be used to minimize the error

$$E = \int_{s[\tilde{\mathbf{u}}]} \left[\tilde{\mathbf{u}} - \sum_{k=1}^{r} \tilde{a}_k \boldsymbol{\psi}_k \right]^2 d\mathbf{x} \tag{12.5}$$

where the inner product is evaluated on the support $s[\tilde{\mathbf{u}}]$, thus making the two terms in the integral of the same size r. The minimizing solution to (12.5) requires the residual to be orthogonal to each mode $\boldsymbol{\psi}_k$ so that

$$\left\langle \tilde{\mathbf{u}} - \sum_{k=1}^{r} \tilde{a}_k \boldsymbol{\psi}_k, \boldsymbol{\psi}_j \right\rangle_{s[\tilde{\mathbf{u}}]} = 0 \qquad j \neq k, \;\; j = 1, 2, \cdots, r. \tag{12.6}$$

In practice, we can project the full state vector $\mathbf{u}$ onto the support space and determine the vector $\tilde{\mathbf{a}}$:

$$\mathbf{M}\tilde{\mathbf{a}} = \mathbf{f} \tag{12.7}$$

where the elements of $\mathbf{M}$ are given by (12.4b) and the components of the vector $\mathbf{f}$ are given by

$$f_k = \langle \mathbf{u}, \boldsymbol{\psi}_k \rangle_{s[\tilde{\mathbf{u}}]} \, . \tag{12.8}$$

Note that if the measurement space is sufficiently dense, or if the support space is the entire space, then $\mathbf{M} = \mathbf{I}$, implying the eigenvalues of $\mathbf{M}$ approach unity as the number of measurements become dense. Once the vector $\tilde{\mathbf{a}}$ is determined, a reconstruction of the solution can be performed as

$$\mathbf{u}(x, t) \approx \boldsymbol{\Psi}\tilde{\mathbf{a}} \, . \tag{12.9}$$

As the measurements become dense, not only does the matrix $\mathbf{M}$ converge to the idenity, but $\tilde{\mathbf{a}} \to \mathbf{a}$. Interestingly, these observations lead us to consider the efficacy of the method and/or approximation by considering the condition number of the matrix $\mathbf{M}$ [524]:

$$\kappa(\mathbf{M}) = \|\mathbf{M}\| \|\mathbf{M}^{-1}\| = \frac{\sigma_1}{\sigma_m} \, . \tag{12.10}$$

Here the 2-norm has been used. If $\kappa(\mathbf{M})$ is small then the matrix is said to be well-conditioned. A minimal value of $\kappa(\mathbf{M})$ is achieved with the identify matrix $\mathbf{M} = \mathbf{I}$. Thus, as the sampling space becomes dense, the condition number also approaches unity. This can be used as a metric for determining how well the sparse sampling is performing. Large condition numbers suggest poor reconstruction while values tending toward unity should perform well.

Harmonic Oscillator Modes

To demonstrate the gappy sampling method and its reconstruction efficacy, we apply the technique to the Gauss-Hermite functions defined by (11.25) and (11.26). In the code that follows, we compute the first ten modes as given by (11.26). To compute the second derivative, we use the fact that the Fourier transform $\mathcal{F}$ can produce a spectrally accurate

approximation, i.e. $u_{xx} = \mathcal{F}^{-1}\left[(ik)^2 \mathcal{F}u\right]$. For the sake of producing accurate derivatives, we consider the domain $x \in [-10, 10]$ but then work with the smaller domain of interest $x \in [-4, 4]$. Recall further that the Fourier transform assumes a 2π-periodic domain. This is handled by a scaling factor in the k-wavevectors. The first five modes have been demonstrated in Fig. 11.3. In the code that follows, we view the first 10 modes with a top-view color plot in order highlight the various features of the modes.

Code 12.1 Harmonic oscillator modes.

```
L=10; x3=-L:0.1:L; n=length(x3)-1; % define domain
x2=x3(1:n); k=(2*pi/(2*L))*[0:n/2-1 -n/2:-1]; % k-vector
ye=exp(-(x2.^2)); ye2=exp((x2.^2)/2); % define Gaussians
for j=0:9       % loop through 10 modes
  yd=real(ifft(((i*k).^j).*fft(ye))); % 2nd derivative
  mode=((-1)^(j))*(((2^j)*factorial(j)*sqrt(pi))^(-0.5))*ye2.*yd
      ;
  y(:,j+1)=(mode).';  % store modes as columns
end

x=x2(n/2+1-40:n/2+1+40);  % keep only -4<x<4
yharm=y(n/2+1-40:n/2+1+40,:);
pcolor(flipud((yharm(:,10:-1:1).')))
```

The mode construction is shown in the top panel of Fig. 12.1. Each colored cell represents the discrete value of the mode in the interval $x \in [-4, 4]$ with $\Delta x = 0.1$. Thus there

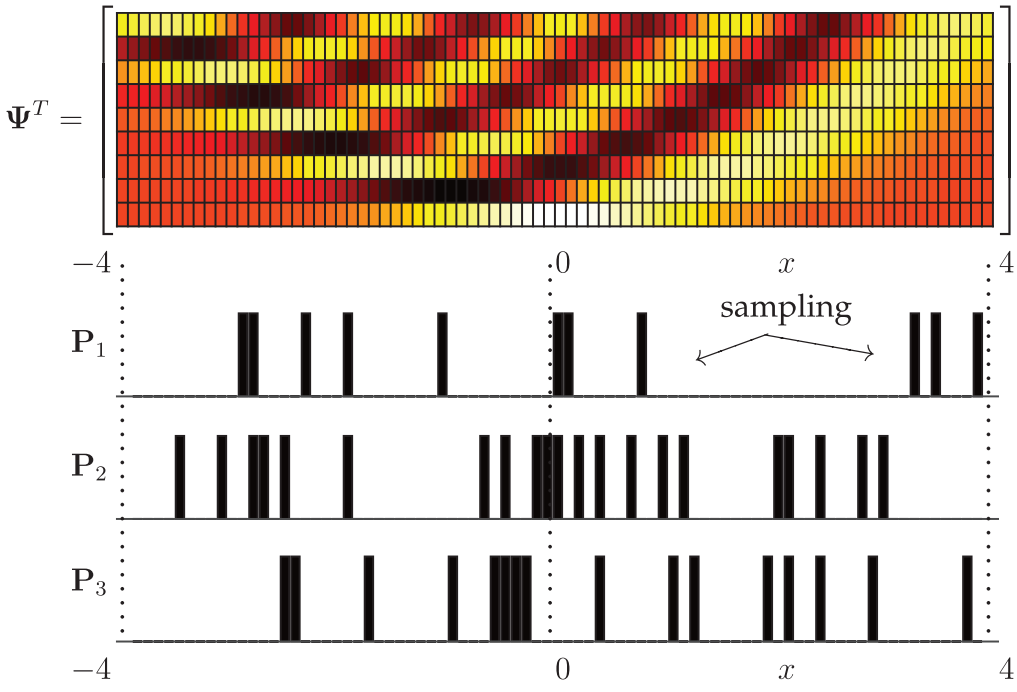

Figure 12.1 The top panel shows the first 10 modes of the quantum harmonic oscillator considered in (11.25) and (11.26). Three randomly generated measurement matrices, $\mathbf{P}_j$ with $j = 1, 2$ and 3, are depicted. There is a 20% chance of performing a measurement at a given spatial location x_j in the interval $x \in [-4, 4]$ with a spacing of $\Delta x = 0.1$.

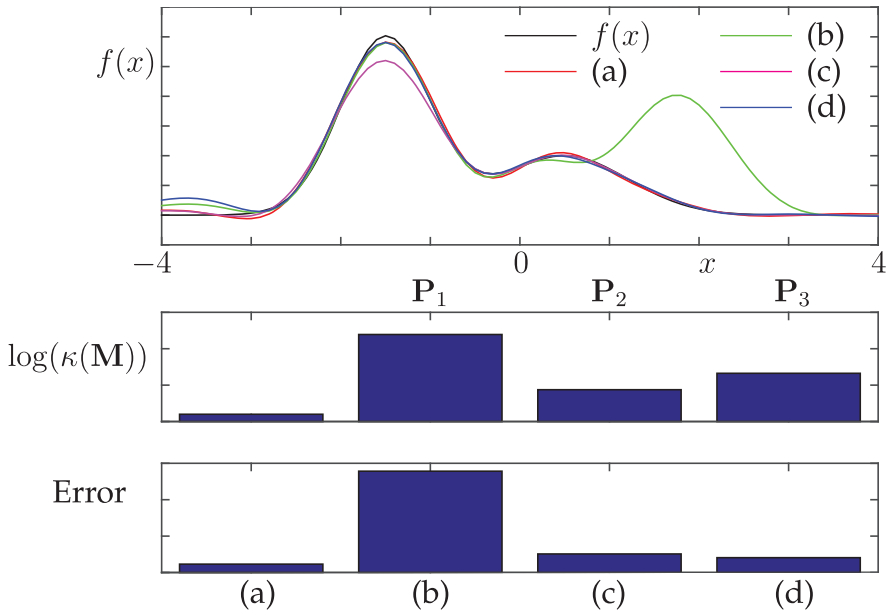

Figure 12.2 The top panel shows the original function (black) along with a 10 mode reconstruction of the test function $f(x) = \exp(-(x-0.5)^2) + 3\exp(-2(x+3/2)^2)$ sampled in the full space (red) and three representative support spaces $s[\tilde{\mathbf{u}}]$ of Fig. 12.1, specifically (b) $\mathbf{P}_1$, (c) $\mathbf{P}_2$, and (d) $\mathbf{P}_3$. Note that the error measurement is specific to the function being considered whereas the condition number metric is independent of the specific function. Although both can serve as proxies for performance, the condition number serves for any function, which is advantageous.

are 81 discrete values for each of the modes $\boldsymbol{\psi}_k$. Our objective is to reconstruct a function outside of the basis modes of the harmonic oscillator. In particular, consider the function

$$f(x) = \exp[-(x-0.5)^2] + 3\exp[-2(x+3/2)^2] \tag{12.11}$$

which will be discretized and defined over the same domain as the modal basis of the harmonic oscillator. The following code builds this function and further numerically constructs the projection of the function onto the basis functions $\boldsymbol{\psi}_n$. The original function is plotted in the top panel of Fig. 12.2. Note that the goal now is to reconstruct this function both with a low-rank projection onto the harmonic oscillator modes, and with a gappy reconstruction whereby only a sampling of the data is used, via the measurements $\mathbf{P}_j$. The following code builds the test function and does a basic reconstruction in the 10-mode harmonic oscillator basis. Further, it builds the matrix $\mathbf{M}$ for the full state measurements and computes its condition number.

Code 12.2 Test function and reconstruction.

```
f=(exp(-(x-0.5).^2)+3*exp(-2*(x+1.5).^2))';
for j=1:10  % full reconstruction
  a(j,1)=trapz(x,f.*yharm(:,j));
end
f2=yharm*a;
subplot(2,1,1), plot(x,f2,'r')
Err(1)=norm(f2-f);  % reconstruction error
```

```
for j=1:10  % matrix M reconstruction
   for jj=1:j
       Area=trapz(x,yharm(:,j).*yharm(:,jj));
       M(j,jj)=Area;
       M(jj,j)=Area;
   end
end
cond(M)    % get condition number
```

Results of the low-rank and gappy reconstruction are shown in Fig. 12.2. The low-rank reconstruction is performed using the full measurements projected to the 10 leading harmonic oscillator modes. In this case, the inner product of the measurement matrix is given by (12.4a) and is approximately the identify. The fact that we are working on a limited domain $x \in [-4, 4]$ with a discretization step of $\Delta x = 0.1$ is what makes $\mathbf{M} \approx \mathbf{I}$ versus being exactly the identify. For the three different sparse measurement scenarios $\mathbf{P}_j$ of Fig. 12.1, the reconstruction is also shown along with the least-square error and the logarithm of the condition number $\log[\kappa(\mathbf{M}_j)]$. We also visualize the three matrices $\mathbf{M}_j$ in Fig. 12.3. The condition number of each of these matrices helps determine its reconstruction accuracy.

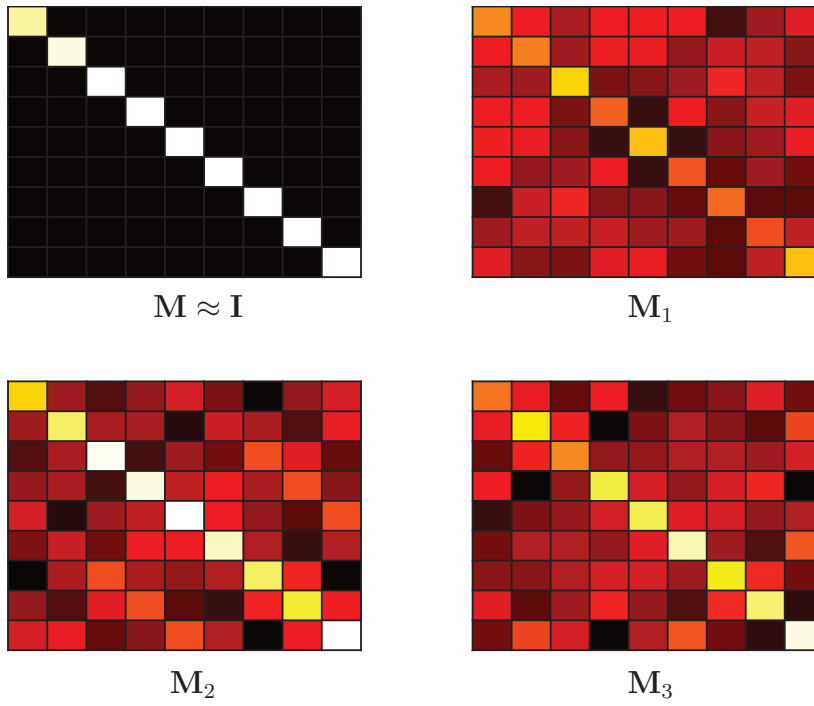

Figure 12.3 Demonstration of the deterioration of the orthogonality of the modal basis in the support space $s[\tilde{\mathbf{u}}]$ as given by the matrix $\mathbf{M}$ defined in (12.4). The top left shows that the identity matrix is produced for full measurements, or nearly so but with errors due to truncation of the domain over $x \in [-4, 4]$. The matrices $\mathbf{M}_j$, which longer look diagonal, correspond to the sparse sampling matrices $\mathbf{P}_j$ in Fig. 12.1. Thus it is clear that the modes are not orthogonal in the support space of the measurements.

Code 12.3 Gappy sampling of harmonic oscillator.

```
c=['g','m','b']; % three different measurement masks
for jloop=1:3
    figure(1), subplot(6,1,3+jloop)
    s=(rand(n,1)>0.8);  % grab 20% random measurements
    bar(x,double(s)), axis([-4.2 4.2 0 1]), axis off

    figure(2)  % construct M_j
    for j=1:10
        for jj=1:j
            Area=trapz(x,s.*(yharm(:,j).*yharm(:,jj)));
            M2(j,jj)=Area; M2(jj,j)=Area;
        end
    end
    subplot(2,2,jloop+1), pcolor(10:-1:1,1:10,(M2')');
    colormap(hot), caxis([-0.1 .3]), axis off
    con(jloop)=cond(M2)

    for j=1:10  % reconstruction using gappy
        ftild(j,1)=trapz(x,s.*(f.*yharm(:,j)));
    end

    atild=M2\ftild;   % compute error
    f2=yharm*atild;
    figure(4),subplot(2,1,1),plot(x,f2,c(jloop))
    Err(jloop+1)=norm(f2-f);
end
```

12.2 Error and Convergence of Gappy POD

As was shown in the previous section, the ability of the gappy sampling strategy to accurately reconstruct a given function depends critically on the placement of the measurement (sensor) locations. Given the importance of this issue, we will discuss a variety of principled methods for placing a limited number of sensors in detail in subsequent sections. Our goal in this section is to investigate the convergence properties and error associated with the gappy method as a function of the percentage of sampling of the full system. Random sampling locations will be used.

Given our random sampling strategy, the results that follow will be statistical in nature, computing averages and variances for batches of randomly selected sampling. The modal basis for our numerical experiments are again the Gauss-Hermite functions defined by (11.25) and (11.26), generated by Code 12.1 and shown in the top panel of Fig. 12.1.

Random Sampling and Convergence

Our study begins with random sampling of the modes at a level of 10%, 20%, 30%, 40%, 50% and 100% respectively. The latter case represents the idealized full sampling of the system. As one would expect, the error and reconstruction are improved as more samples are taken. To show the convergence of the gappy sampling, we consider two error metrics: (i) the ℓ_2 error between our randomly subsampled reconstruction and (ii) the condition number of the matrix $\mathbf{M}$ for a given measurement matrix $\mathbf{P}_j$. Recall that the condition number provides a way to measure the error without knowing the truth, i.e. (12.11).

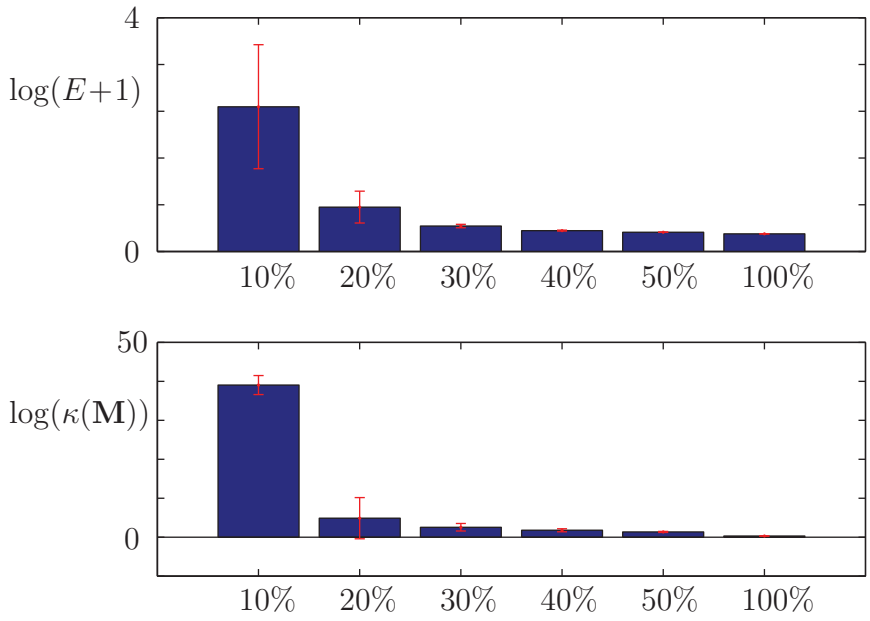

Figure 12.4 Logarithm of the least-square error, $\log(E+1)$ (unity is added to avoid negative numbers), and the log of the condition number, $\log(\kappa(\mathbf{M}))$, as a function of percentage of random measurements. For 10% measurements, the error and condition number are largest as expected. However, the variance of the results, depicted by the red bars is also quite large, suggesting that the performance for a small number of sensors is highly sensitive to their placement.

Fig. 12.4 depicts the average over 1000 trials of the logarithm of the least-square error, $\log(E+1)$ (unity is added to avoid negative numbers), and the log of the condition number, $\log(\kappa(\mathbf{M}))$, as a function of percentage of random measurements. Also depicted is the variance σ with the red bars denoting $\mu \pm \sigma$ where μ is the average value. The error and condition number both perform better as the number of samples increases. Note that the error does not approach zero since only a 10-mode basis expansion is used, thus limiting the accuracy of the POD expansion and reconstruction even with full measurements.

The following code, which is the basis for constructing Fig. 12.4, draws over 1000 random sensor configurations using 10%, 20%, 30%, 40% and 50% sampling. The full reconstruction (100% sampling) is actually performed in Code 12.2 and is used to make the final graphic for Fig. 12.4. Note that as expected, the error and condition number trends are similar, thus supporting the hypothesis that the condition number can be used to evaluate the efficacy of the sparse measurements. Indeed, this clearly shows that the condition number provides an evaluation that does not require knowledge of the function in (12.11).

Code 12.4 Convergence of error and condition number.

```
for thresh=1:5;
  for jloop=1:1000  % 1000 random trials
    n2=randsample(n,8*thresh);  % random sampling
    P=zeros(n,1); P(n2)=1;
    for j=1:10
        for jj=1:j    % compute M matrix
```

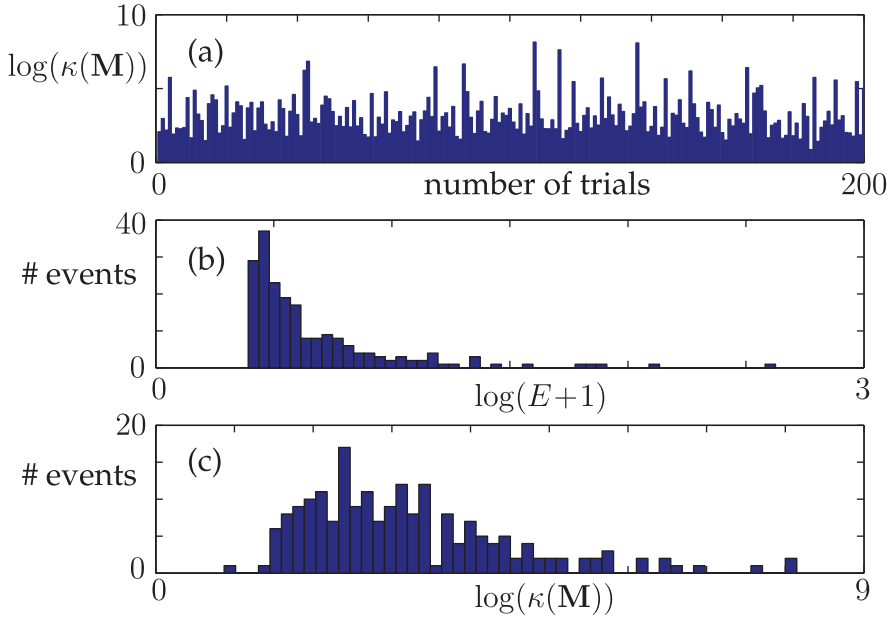

Figure 12.5 Statistics of 20% random measurements considered in Fig. 12.4. The top panel (a) depicts 200 random trials and the condition number $\log(\kappa(\mathbf{M}))$ of each trial. A histogram of (b) the logarithm of the least-square error, $\log(E+1)$, and (c) condition number , $\log(\kappa(\mathbf{M}))$, are also depicted for the 200 trials. The figures illustrate the extremely high variability generated from the random, sparse measurements. In particular, 20% measurements can produce both exceptional results and extremely poor performance depending upon the measurement locations. The measurement vectors $\mathbf{P}$ are generating these statistics are depicted in Fig. 12.6.

```
            Area=trapz(x,P.*(yharm(:,j).*yharm(:,jj)));
            M2(j,jj)=Area; M2(jj,j)=Area;
        end
    end
    for j=1:10  % reconstruction using gappy
        ftild(j,1)=trapz(x,P.*(f.*yharm(:,j)));
    end
    atild=M2\ftild;    % compute error
    f2=yharm*atild;    % compute reconstruction
    Err(jloop)=norm(f2-f); % L2 error
    con(jloop)=cond(M2);   % condition number
  end
  % mean and variance
  E(thresh)=mean(log(Err+1)); V(thresh)=(var(log(Err+1)));
  Ec(thresh)=mean(log(con)); Vc(thresh)=(var(log(con)));
end
E=[E Efull]; V=[V 0];
Ec=[Ec log(Cfull)]; Vc=[Vc 0];
```

Gappy Measurements and Performance

We can continue this statistical analysis of the gappy reconstruction method by looking more carefully at 200 random trials of 20% measurements. Fig. 12.5 shows three key features of the 200 random trials. In particular, as shown in the top panel of this figure,

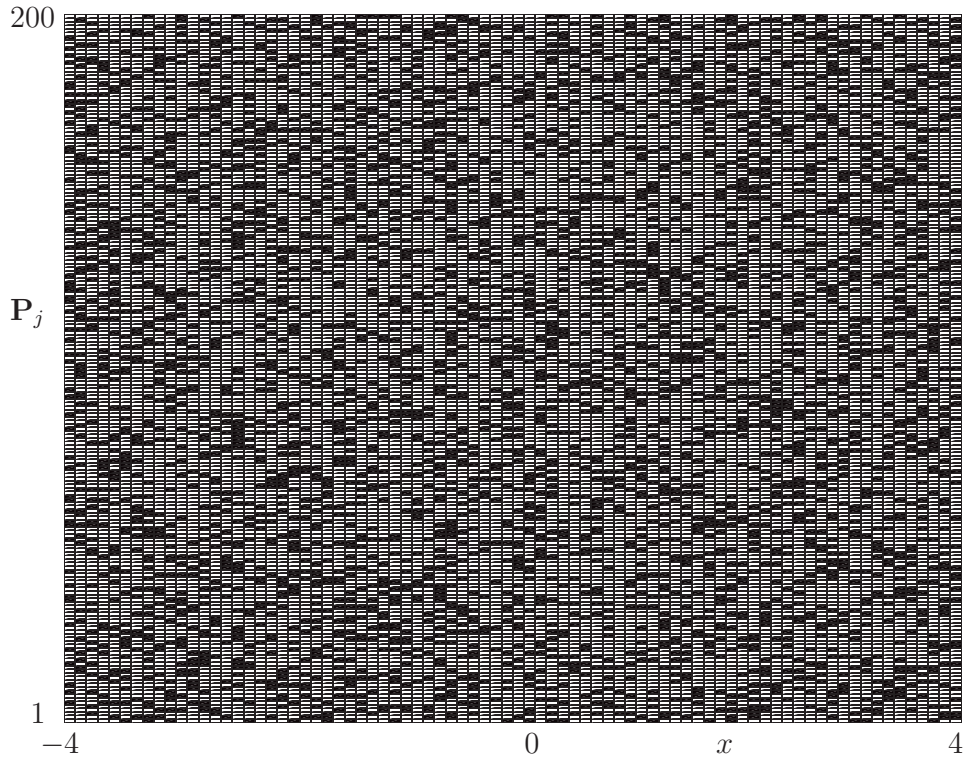

Figure 12.6 Depiction of the 200 random 20% measurement vectors $\mathbf{P}_j$ considered in Fig. 12.5. Each row is a randomly generated measurement trial (from 1 to 200) while the columns represent their spatial location on the domain $x \in [-4, 4]$ with $\Delta x = 0.1$.

there is a large variance in the distribution of the condition number $\kappa(\mathbf{M})$ for 20% sampling. Specifically, the condition number can change by orders of magnitude with the same number of sensors, but simply placed in different locations. A histogram of the distribution of the log error $\log(E+1)$ and the log of the condition number are shown in the bottom two panels. The error appears to be distributed in an exponentially decaying fashion whereas the condition number distribution is closer to a Gaussian. There are distinct outliers whose errors and condition numbers are exceptionally high, suggesting sensor configurations to be avoided.

In order to visualize the random, gappy measurements of the 200 samples used in the statistical analysis of Fig. 12.5, we plot the $\mathbf{P}_j$ measurement masks in each row of the matrix in Fig. 12.6. The white regions represent regions where no measurements occur. The black regions are where the measurements are taken. These are the measurements that generate the orders of magnitude variance in the error and condition number.

As a final analysis, we can sift through the 200 random measurements of Fig. 12.6 and pick out both the ten best and ten worst measurement vectors $\mathbf{P}_j$. Fig. 12.7 shows the results of this sifting process. The top two panels depict the best and worst measurement configurations. Interestingly, the worst measurements have long stretches of missing measurements near the center of the domain where much of the modal variance occurs.

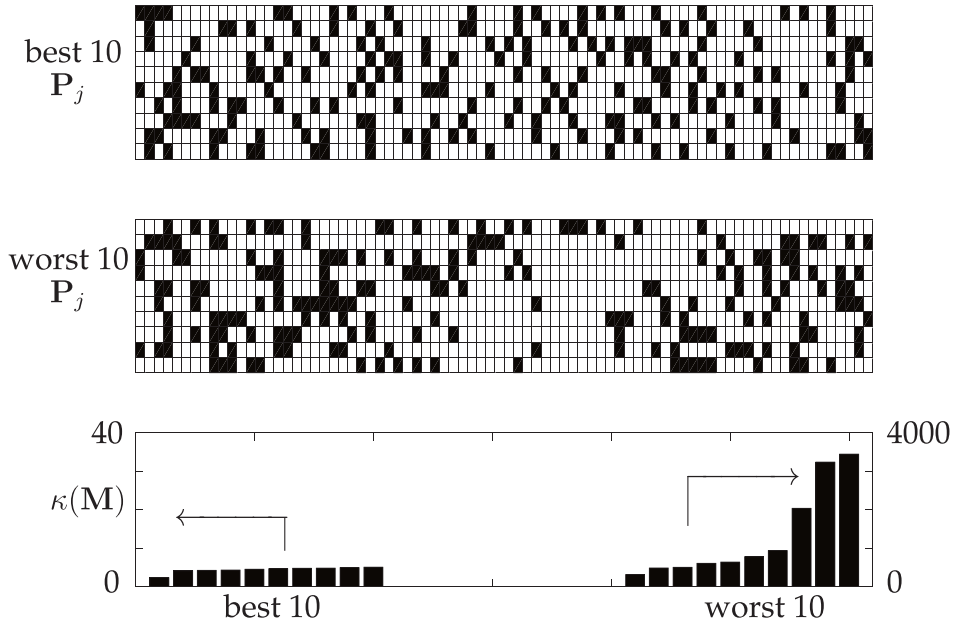

Figure 12.7 Depiction of the 10 best and 10 worst random 20% measurement vectors $\mathbf{P}_j$ considered in Figs. 12.5 and 12.6. The top panel shows that the best measurement vectors sample fairly uniformly across the domain $x \in [-4, 4]$ with $\Delta x = 0.1$. In contrast, the worst randomly generated measurements (middle panel) have large sampling gaps near the center of the domain, leading to a large condition number $\kappa(\mathbf{M})$. The bottom panel shows a bar chart of the best and worst values of the condition number. Note that with 20% sampling, there can be two orders of magnitude difference in the condition number, thus suggesting the importance of prescribing good measurement locations.

In contrast, the best measurements have well sampled domains with few long gaps between measurement locations. The bottom panel shows that the best measurements (on the left) offer an improvement of two orders of magnitude in the condition number over the poor performing counterparts (on the right).

12.3 Gappy Measurements: Minimize Condition Number

The preceding section illustrates that the placement of gappy measurements is critical for accurately reconstructing the POD solution. This suggests that a principled way to determine measurement locations is of great importance. In what follows, we outline a method originally proposed by Willcox [555] for assessing the gappy measurement locations. The method is based on minimizing the condition number $\kappa(\mathbf{M})$ in the placement process. As already shown, the condition number is a good proxy for evaluating the efficacy of the reconstruction. Moreover, it is a measure that is independent of any specific function.

The algorithm proposed [555] is computationally costly, but it can be performed in an offline training stage. Once the sensor locations are determined, they can be used for online reconstruction. The algorithm is as follows:

1. Place sensor k at each spatial location possible and evaluate the condition number $\kappa(\mathbf{M})$. Only points not already containing a sensor are considered.

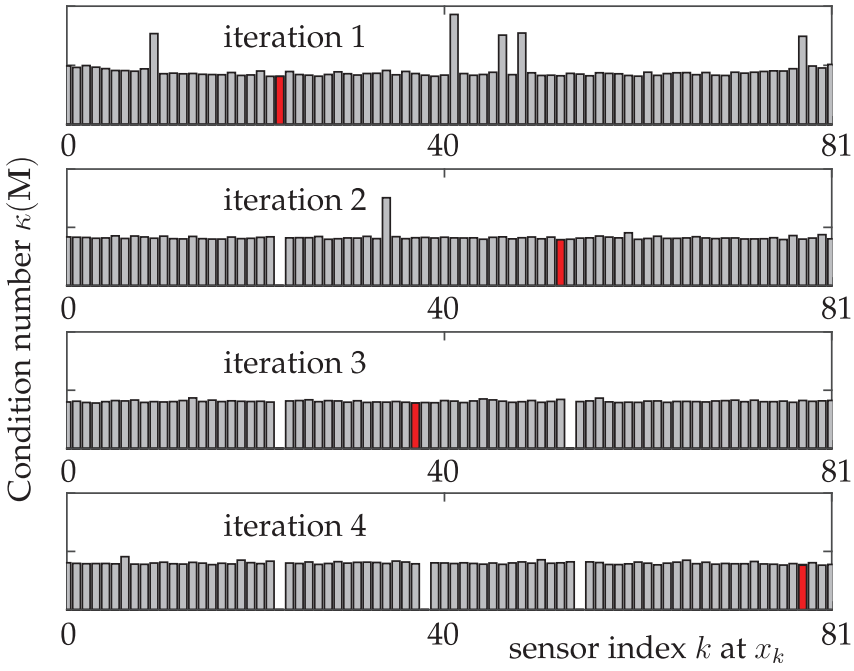

Figure 12.8 Depiction of the first four iterations of the gappy measurement location algorithm of Willcox [555]. The algorithm is applied to a 10-mode expansion given by the Gauss-Hermite functions (11.25) and (11.26) discretized on the interval $x \in [-4, 4]$ with $\Delta x = 0.1$. The top panel shows the condition number $\kappa(\mathbf{M})$ as a single sensor is considered at each of the 81 discrete values x_k. The first sensor minimizes the condition number (shown in red) at x_{23}. A second sensor is now considered at all remaining 80 spatial locations, with the minimal condition number occurring at x_{52} (in red). Repeating this process gives x_{37} and x_{77} for the third and fourth sensor locations for iteration 3 and 4 of the algorithm (highlighted in red). Once a location is selected for a sensor, it is no longer considered in future iterations. This is represented by a gap.

2. Determine the spatial location that minimizes the condition number $\kappa(\mathbf{M})$. This spatial location is now the kth sensor location.
3. Add sensor $k + 1$ and repeat the previous two steps.

The algorithm is not optimal, nor are there guaranteed. However, it works quite well in practice since sensor configurations with low condition number produce good reconstructions with the POD modes.

We apply this algorithm to construct the gappy measurement matrix $\mathbf{P}$. As before, the modal basis for our numerical experiments are the Gauss-Hermite functions defined by (11.25) and (11.26). The gappy measurement matrix algorithm for constructing $\mathbf{P}$ is shown in Note that the algorithm outlined above sets down one sensor at a time, thus with the 10 POD mode expansion, the system is underdetermined until 10 sensors are placed. This gives condition numbers on the order of 10^{16} for the first 9 sensor placements. It also suggests that the first 10 sensor locations may be generated from inaccurate calculations of the condition number.

The following code builds upon Code 12.1 which is used to generate the 10-mode expansion of the Gauss-Hermite functions. The code minimizes the condition number and

identifies the first 20 sensor locations. Specifically, the code provides a principled way of producing a measurement matrix **P** that allows for good reconstruction of the POD mode expansion with limited measurements.

Code 12.5 Gappy placement: Minimize condition number.

```
n2=20;    % number of sensors
nall=1:n; ns=[];  %
for jsense=1:n2
    for jloop=1:(n-jsense)
        P=zeros(n,1); P(ns)=1;
        P(nall(jloop))=1;
        for j=1:10
            for jj=1:j    % matrix M
                Area=trapz(x,P.*(yharm(:,j).*yharm(:,jj)));
                M2(j,jj)=Area; M2(jj,j)=Area;
            end
        end
        con(jloop)=cond(M2);  % compute condition number
    end  %  end search through all points
    [s1,n1]=min(con); % location to minimize condition #
    kond(jsense)=s1; clear con
    ns=[ns nall(n1)];  % add sensor location
    nall=setdiff(nall,ns);  % new sensor indeces
    P=zeros(n,1);  P(ns)=1;
    Psum(:,jsense)=P;
    for j=1:10
        for jj=1:j
            Area=trapz(x,P.*(yharm(:,j).*yharm(:,jj)));
            M2(j,jj)=Area; M2(jj,j)=Area;
        end
    end
    for j=1:10  % reconstruction using gappy
        ftild(j,1)=trapz(x,P.*(f.*yharm(:,j)));
    end
    atild=M2\ftild;   % compute error
    f1(:,jsense)=yharm*atild;  % iterative reconstruction
    E(jsense)=norm(f1(:,jsense)-f);  % iterative error
end  % end sensor loop
```

In addition to identifying the placement of the first 20 sensors, the code also reconstructs the example function given by (12.11) at each iteration of the routine. Note the use of the **setdiff** command which removes the condition number minimizing sensor location from consideration in the next iteration.

To evaluate the gappy sensor location algorithm, we track the condition number as a function of the number of iterations, up to 20 sensors. Additionally, at each iteration, a reconstruction of the test function (12.11) is computed and a least-square error evaluated. Fig. 12.9 shows the progress of the algorithm as it evaluates the sensor locations for up to 20 sensors. By construction, the algorithm minimizes the condition number $\kappa(\mathbf{M})$ at each step of the iteration, thus as sensors are added, the condition number steadily decreases (top panel of Fig. 12.9). Note that there is a significant decrease in the condition number once 10 sensors are selected since the system is no longer underdetermined with theoretically infinite condition number. The least-square error for the reconstruction of the test function (12.11) follows the same general trend, but the error does not monotonically decrease like the condition number. The least-square error also makes a significant improvement once

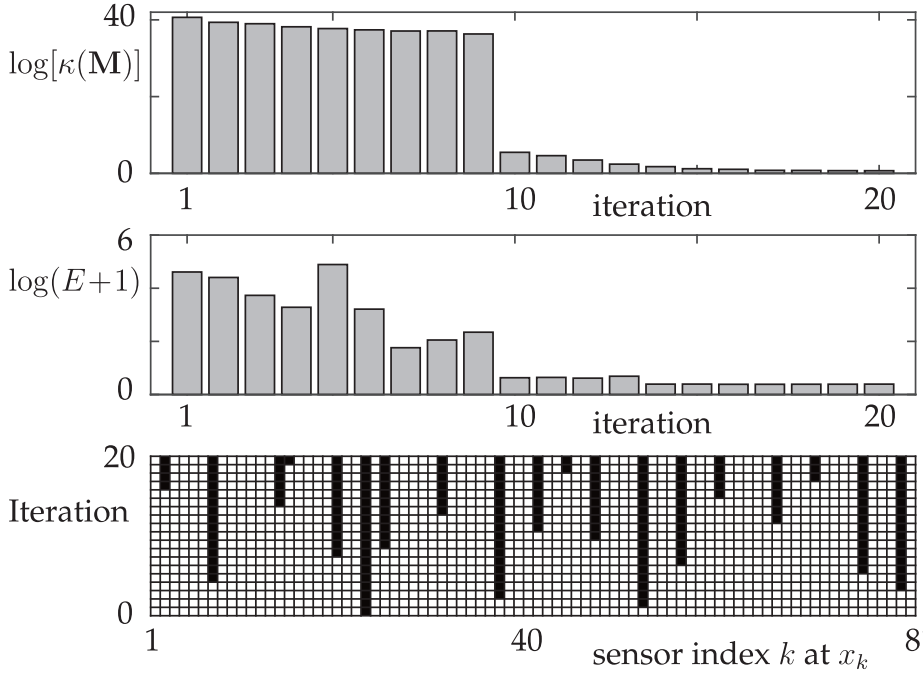

Figure 12.9 Condition number and least-square error (logarithms) as a function of the number of iterations in the gappy sensor placement algorithm. The log of the condition number $\log[\kappa(\mathbf{M})]$ monotonically decreases since this is being minimized at each iteration step. The log of the least-square error in the reconstruction of the test function (12.11) also shows a trend towards improvement as the number of sensors are increased. Once 10 sensors are placed, the system is of full rank and the condition number drops by orders of magnitude. The bottom panel shows the sensors as they turn on (black squares) over the first 20 iterations. The first measurement location is, for instance, at x_{23}.

10 measurements are made. In general, if an r-mode POD expansion is to be considered, then reasonable results using the gappy reconstruction cannot be achieved until r sensors are placed.

We now consider the placement of the sensors as a function of iteration in the bottom panel of Fig. 12.9. Specifically, we depict when sensors are identified in the iteration. The first sensor location is x_{23} followed by x_{52}, x_{37} and x_{77}, respectively. The process is continued until the first 20 sensors are identified. The pattern of sensors depicted is important as it illustrates a fairly uniform sampling of the domain. Alternative schemes will be considered in the following.

As a final illustration of the gappy algorithm, we consider the reconstruction of the test function (12.11) as the number of iterations (sensors) increases. As expected, the more sensors that are used in the gappy framework, the better the reconstruction is, especially if they are placed in a principled way as outlined by Wilcox [555]. Fig. 12.10 shows the reconstructed function with increasing iteration number. In the left panel, iteration one through twenty are shown with the z-axis set to illustrate the extremely poor reconstruction in the early stages of the iteration. The right panel highlights the reconstruction from iteration nine to twenty, and on a more limited z-axis scale, where the reconstruction

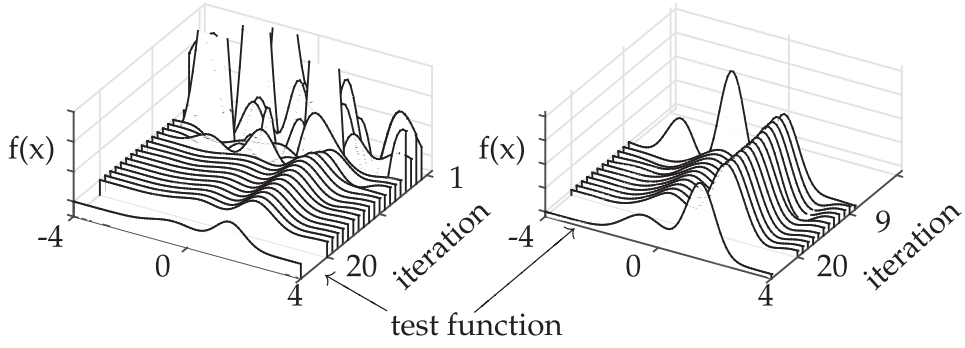

Figure 12.10 Convergence of the reconstruction to the test function (12.11). The left panel shows iterations one through twenty and the significant reconstruction errors of the early iterations and limited number of sensors. Indeed, for the first nine iterations, the condition number and least-square error is quite large since the system is not full rank. The right panel shows a zoom-in of the solution from iteration nine to twenty where the convergence is clearly observed. Comparison in both panels can be made to the test function.

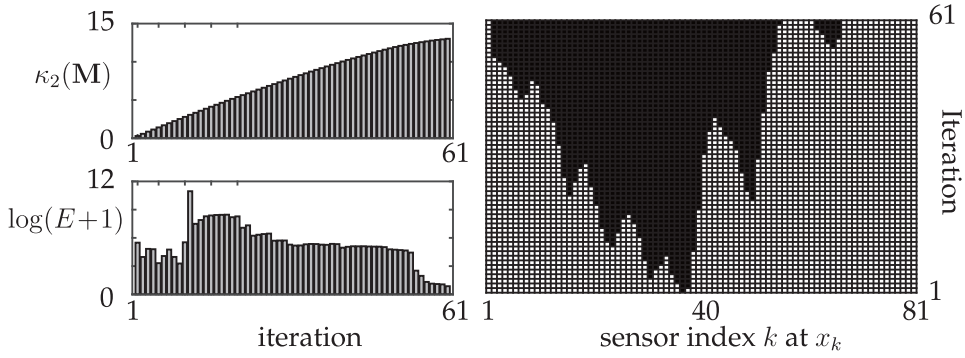

Figure 12.11 Sum of diagonals minus off-diagonals (top left) and least-square error (logarithm) as a function of the number of iterations in the second gappy sensor placement algorithm. The new proxy metric for condition number monotonically increases since this is being maximized at each iteration step. The log of the least-square error in the reconstruction of the test function (12.11) shows a trend towards improvement as the number of sensors are increased, but convergence is extremely slow in comparison to minimizing the condition number. The right panel shows the sensors as they turn on (black squares) over the first 60 iterations. The first measurement location is, for instance, at x_{37}.

converges to the test function. The true test function is also shown in order to visualize the comparison. This illustrates in a tangible way the convergence of the iteration algorithm to the test solution with a principled placement of sensors.

Proxy Measures to the Condition Number

We end this section by considering alternative measures to the condition number $\kappa(\mathbf{M})$. The computation of the condition number itself can be computationally expensive. Moreover, until r sensors are chosen in an r-POD mode expansion, the condition number computation is itself numerically unstable. However, it is clear what the condition number minimization algorithm is trying to achieve: make the measurement matrix $\mathbf{M}$ as near to the identify as

possible. This suggests the following alternative algorithm, which was also developed by Willcox [555].

1. Place sensor k at each spatial location possible and evaluate the difference in the sum of the diagonal entries of the matrix $\mathbf{M}$ minus the sum of the off-diagonal components, call this $\kappa_2(\mathbf{M})$. Only points not already containing a sensor are considered.
2. Determine the spatial location that generates the maximum value of the above quantify. This spatial location is now the kth sensor location.
3. Add sensor $k+1$ and repeat the previous two steps.

This algorithm provides a simple modification of the original algorithm which minimizes the condition number. In particular, the following lines of code provide modifications to Code 12.5. Specifically, where the condition number is computed, the following line is now included:

```
    nall=setdiff(nall,ns);  % new sensor indeces
```

Additonally, the sensor locations are now considered at the maximal points so that the following line of code is applied

```
    P=zeros(n,1);  P(ns)=1;
```

Thus the modification of two lines of code can enact this new metric which circumvents the computation of the condition number.

To evaluate this new gappy sensor location algorithm, we track the new proxy metric we are trying to maximize as a function of the number of iterations along with the least-square error of our test function (12.11). In this case, up to 60 sensors are considered since the convergence is slower than before. Fig. 12.11 shows the progress of the algorithm as it evaluates the sensor locations for up to 60 sensors. By construction, the algorithm maximizes the sum of the diagonals minus the sum of the off-diagonals at each step of the iteration, thus as sensors are added, this measure steadily increases (top left panel of Fig. 12.11). The least-square error for the reconstruction of the test function (12.11) decreases, but not monotonically. Further, the convergence is very slow. At least for this example, the method does not work as well as the condition number metric. However, it can improve performance in certain cases [555], and it is much more computationally efficient to compute.

As before, we also consider the placement of the sensors as a function of iteration in the right panel of Fig. 12.11. Specifically, we depict the turning on process of the sensors. The first sensor location is x_{37} followed by x_{38}, x_{36} and x_{31} respectively. The process is continued until the first 60 sensors are turned on. The pattern of sensors depicted is significantly different than in the condition number minimization algorithm. Indeed, this algorithm, and with these modes, turns on sensors in local locations without sampling uniformly from the domain.

12.4 Gappy Measurements: Maximal Variance

The previous section developed principled ways to determine the location of sensors for gappy POD measurements. This was a significant improvement over simply choosing sensor locations randomly. Indeed, the minimization of the condition number through location

selection performed quite well, quickly improving accuracy and least-square reconstruction error. The drawback to the proposed method was two-fold: the algorithm itself is expensive to implement, requiring a computation of the condition number for every sensor location selected under an exhaustive search. Secondly, the algorithm was ill-conditioned until the rth sensor was chosen in an r-POD mode expansion. Thus the condition number was theoretically infinite, but on the order of 10^{17} for computational purposes.

Karniadakis and co-workers [565] proposed an alternative to the Willcox [555] algorithm to overcome the computational issues outlined. Specifically, instead of placing one sensor at a time, the new algorithm places r sensors, for an r-POD mode expansion, at the first step of the iteration. Thus the matrix generated is no longer ill-conditioned with a theoretically infinite condition number.

The algorithm by Karniadakis further proposes a principled way to select the original r sensor locations. This method selects locations that are extrema points of the POD modes, which are designed to maximally capture variance in the data. Specifically, the following algorithm is suggested:

1. Place r sensors initially.
2. Determine the spatial locations of these first r sensors by considering the maximum of each of the POD modes $\boldsymbol{\psi}_k$.
3. Add additional sensors at the next largest extrema of the POD modes.

The following code determines the maximum of each mode and constructs a gappy measurement matrix $\mathbf{P}$ from such locations.

Code 12.6 Gappy placement: Maximize variance.

```
ns=[];
for j=1:10  % walk through the modes
  [s1,n1]=max(yharm(:,j)); % pick max
  ns=[ns n1];
end
P=zeros(n,1);  P(ns)=1;
```

The performance of this algorithm is not strong for only r measurements, but it at least produces stable condition number calculations. To improve performance, one could also use the minimum of each of the modes $\boldsymbol{\psi}_k$. Thus the maximal value and minimal value of variance are considered. For the harmonic oscillator code, the first mode produces no minimum as the minima are at $x \to \pm\infty$. Thus 19 sensor locations are chosen in the following code:

Code 12.7 Gappy placement: Max and min variance.

```
ns=[];
for j=1:10  % walk through the modes
  [s1,n1]=max(yharm(:,j)); % pick max
  ns=[ns n1];
end
for j=2:10
  [s2,n2]=min(yharm(:,j)); % pick max
  ns=[ns n2];
end
P=zeros(n,1);  P(ns)=1;
```

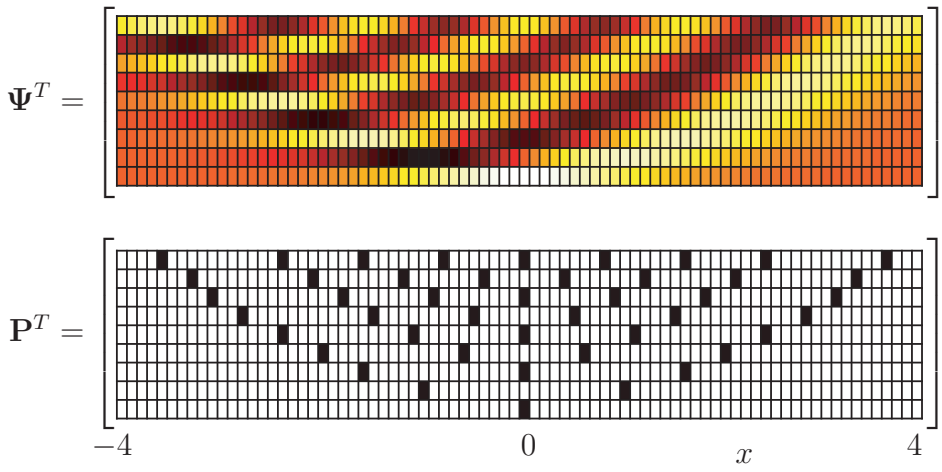

Figure 12.12 The top panel shows the mode structures of the Gauss-Hermite polynomials Ψ in the low-rank approximation of a POD expansion. The discretization interval is $x \in [-4, 4]$ with a spacing of $\Delta x = 0.1$. The color map shows the maximum (white) and minimum (black) that occur in the mode structures. The bottom panel shows the grid cells corresponding to maximum and minimum (extrema) of POD mode variance. The extrema are candidates for sensor locations, or the measurement matrix $\mathbf{P}$, since they represent maximal variance locations. Typically one would take a random subsample of these extrema to begin the evaluation of the gappy placement.

Note that in this case, the number of sensors is almost double that of the previous case. Moreover it only searches for the the locations where variability is highest, which is intuitively appealing for measurements.

More generally, the Karniadakis algorithm [565] advocates randomly selecting p sensors from M potential extrema, and then modifying the search positions with the goal of improving the condition number. In this case, one must identify all the maxima and minima of the POD modes in order to make the selection. The harmonic oscillator modes and their maxima and minima are illustrated in Fig. 12.12. The algorithm used to produce the extrema of each mode, and its potential for use in the gappy algorithm, is as follows:

Code 12.8 Gappy placement: Extrema locations.

```
nmax=[]; nmin=[];
Psum = zeros(n,10);
for j=1:10  % walk through the modes
    nmaxt=[]; nmint=[];
    for jj=2:n-1
       if yharm(jj,j)>yharm(jj-1,j) & yharm(jj,j)>yharm(jj+1,j)
           nmax=[nmax jj];
           nmaxt=[nmaxt jj];
       end
       if yharm(jj,j)<yharm(jj-1,j) & yharm(jj,j)<yharm(jj+1,j)
           nmin=[nmin jj];
           nmint=[nmint jj];
       end
    end
    nst=[nmaxt nmint]
    Psum(nst,j)=1;
end
```

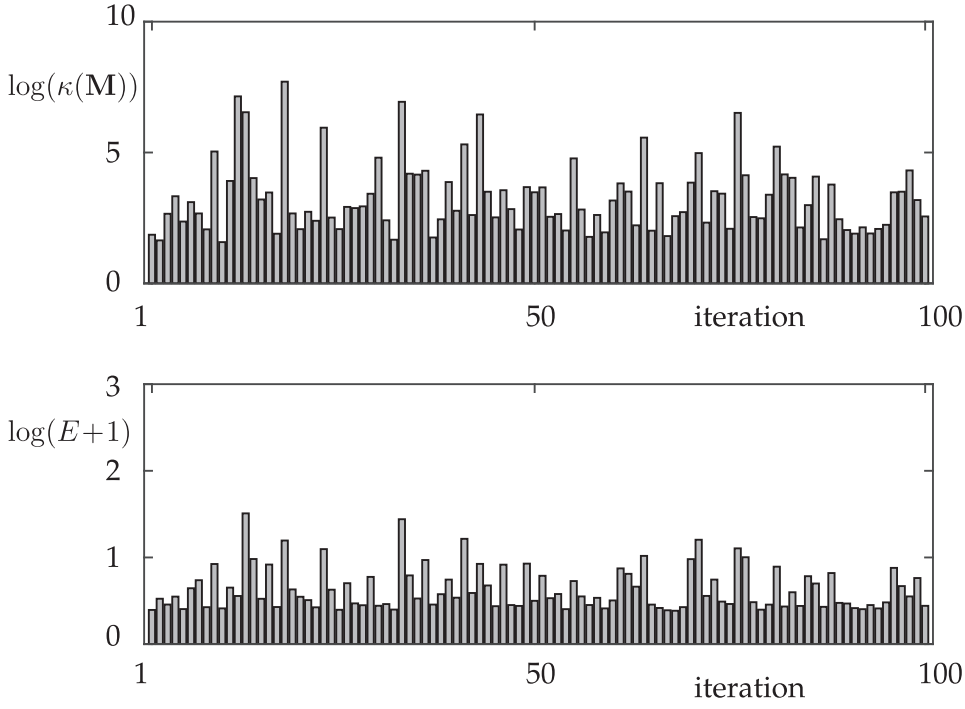

Figure 12.13 Condition number and least-square error to test function (12.11) over 100 random trials that draw 20 sensor locations from the possible 55 extrema depicted in Fig. 12.12. The 100 trials produce a number of sensor configurations that perform close to the level of the condition number minimization algorithm of the last section. However, the computational costs in generating such trials can be significantly lower.

```
ns=[nmax nmin];
ni=randsample(length(ns),20);
nsr=ns(ni);
P=zeros(n,1);   P(nsr)=1;
```

Note that the resulting vector **ns** contains all 55 possible extrema. This computation assumes the data is sufficiently smooth so that extrema are simply found by considering neighboring points, i.e. a maxima exists if its two neighbors have a lower value whereas an minima exists if its neighbors have a higher value.

The maximal variance algorithm suggests trying different configurations of the sensors at the extrema points. In particular, if 20 gappy measurements are desired, then we would need to search through various configurations of the 55 locations using 20 sensors. This combinatorial search is intractable. However, if we simply attempt 100 random trials and select the best performing configuration, it is quite close to the performance of the condition number minimizing algorithm. A full execution of this algorithm, along with a computation of the condition number and least-square fit error with (12.11), is generated by the following code:

Code 12.9 Gappy placement: Random selection.

```
ntot=length(ns);
for jtrials=1:100
```

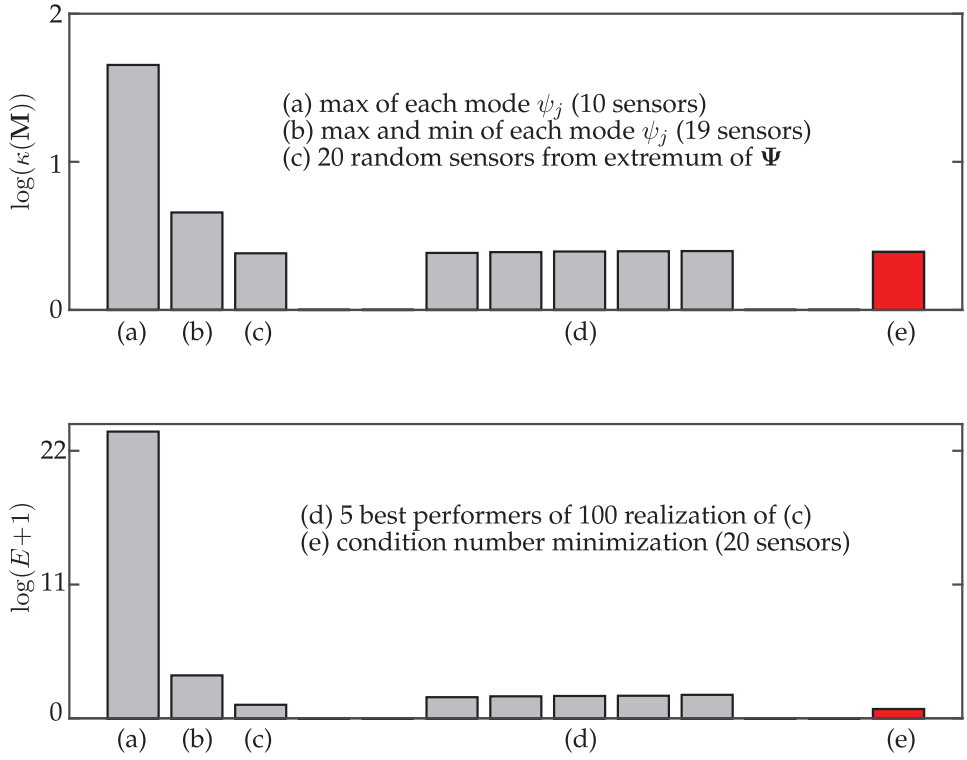

Figure 12.14 Performance metrics for placing sensors based upon the extrema of the variance of the POD modes. Both the least-square error for the reconstruction of the test function (12.11) and the condition number are considered. Illustrated are the results from using (a) the maximum locations of the POD modes, (b) the maximum and minimum locations of each POD mode, and (c) a random selection of 20 of the 55 extremum locations of the POD modes. These are compared against (d) the 5 top selections of 20 sensors from the 100 random trials, and (e) the condition number minimization algorithm (red bar). The random placement of sensors from the extremum locations provides performance close to that of the condition minimization without the same high computational costs.

```
ni=randsample(ntot,20);
nsr=ns(ni);

   P=zeros(n,1);   P(nsr)=1;

    for j=1:10
        for jj=1:j
            Area=trapz(x,P.*(yharm(:,j).*yharm(:,jj)));
            M2(j,jj)=Area; M2(jj,j)=Area;
        end
    end

    for j=1:10  % reconstruction using gappy
        ftild(j,1)=trapz(x,P.*(f.*yharm(:,j)));
    end
    atild=M2\ftild;    % compute error
    f1=yharm*atild;  % iterative reconstruction
    E_tri(jtrials)=norm(f1-f);  % iterative error
    con_tri(jtrials)=cond(M2);
```

```
end
subplot(2,1,1), bar(log(con_tri),'Facecolor',[0.7 0.7 0.7])
subplot(2,1,2), bar(log(E_tri+1),'Facecolor',[0.7 0.7 0.7])
```

The condition number and least-square error for the 100 trials is shown in Fig. 12.13. The configurations perform well compared with random measurements, although some have excellent performance.

A direct comparison of all these methods is shown in Fig. 12.14. Specifically, what is illustrated are the results from using (a) the maximum locations of the POD modes, (b) the maximum and minimum locations of each POD mode, and (c) a random selection of 20 of the 55 extremum locations of the POD modes. These are compared against (d) the best 5 sensor placement locations of 20 sensors selected from the extremum over 100 random trials, and (e) the condition number minimization algorithm in red. The maximal variance algorithm performs approximately as well as the minimum condition number algorithm. However, the algorithm is faster and never computes condition numbers on ill-conditioned matrices. Karniadakis and co-workers [565] also suggest innovations on this basic implementation. Specifically, it is suggested that one consider each sensor, one-by-one, and try placing it in all other available spatial locations. If the condition number is reduced, the sensor is moved to that new location and the next sensor is considered.

12.5 POD and the Discrete Empirical Interpolation Method (DEIM)

The POD method illustrated thus far aims to exploit the underlying low-dimensional dynamics observed in many high-dimensional computations. POD is often used for reduced-order models (ROMs), which are of growing importance in scientific applications and computing. ROMS reduce the computational complexity and time needed to solve large-scale, complex systems [53, 442, 244, 17]. Specifically, ROMs provide a principled approach to approximating high-dimensional spatio-temporal systems [139], typically generated from numerical discretization, by low-dimensional subspaces that produce nearly identical input/output characteristics of the underlying nonlinear dynamical system. However, despite the significant reduction in dimensionality with a POD basis, the complexity of evaluating higher-order nonlinear terms may remain as challenging as the original problem [41, 127]. The empirical interpolation method (EIM), and the simplified discrete empirical interpolation method (DEIM) for the proper orthogonal decomposition (POD) [347, 251], overcome this difficulty by providing a computationally efficient method for discretely (sparsely) sampling and evaluating the nonlinearity. These methods ensure that the computational complexity of ROMs scale favorably with the rank of the approximation, even with complex nonlinearities.

EIM has been developed for the purpose of efficiently managing the computation of the nonlinearity in dimensionality reduction schemes, with DEIM specifically tailored to POD with Galerkin projection. Indeed, DEIM approximates the nonlinearity by using a small, discrete sampling of points that are determined in an algorithmic way. This ensures that the computational cost of evaluating the nonlinearity scales with the rank of the reduced POD basis. As an example, consider the case of an r-mode POD-Galerkin truncation. A simple cubic nonlinearity requires that the POD-Galerkin approximation be cubed, resulting in r^3 operations to evaluate the nonlinear term. DEIM approximates the cubic nonlinearity by using $\mathcal{O}(r)$ discrete sample points of the nonlinearity, thus

Table 12.1 DEIM algorithm for finding approximation basis for the nonlinearity and its interpolation indices. The algorithm first constructs the nonlinear basis modes and initializes the first measurement location, and the matrix $\mathbf{P}_1$, as the maximum of $\boldsymbol{\xi}_1$. The algorithm then successively constructs columns of $\mathbf{P}_j$ by considering the location of the maximum of the residual $\mathbf{R}_j$.

DEIM algorithm	
Basis Construction and Initialization	
• collect data, construct snapshot matrix	$\mathbf{X} = [\mathbf{u}(t_1)\ \mathbf{u}(t_2)\ \cdots\ \mathbf{u}(t_m)]$
• construct nonlinear snapshot matrix	$\mathbf{N} = [N(\mathbf{u}(t_1))\ N(\mathbf{u}(t_2))\ \cdots\ N(\mathbf{u}(t_m))]$
• singular value decomposition of $\mathbf{N}$	$\mathbf{N} = \Xi\Sigma_{\mathbf{N}}\mathbf{V}_{\mathbf{N}}^*$
• construct rank-p approximating basis	$\Xi_p = [\boldsymbol{\xi}_1\ \boldsymbol{\xi}_2\ \cdots\ \boldsymbol{\xi}_p]$
• choose the first index (initialization)	$[\rho, \gamma_1] = \max\|\boldsymbol{\xi}_1\|$
• construct first measurement matrix	$\mathbf{P}_1 = [\mathbf{e}_{\gamma_1}]$
Interpolation Indices and Iteration Loop ($j = 2, 3, ..., p$)	
• calculate $\mathbf{c}_j$	$\mathbf{P}_j^T\Xi_j\mathbf{c}_j = \mathbf{P}_j^T\boldsymbol{\xi}_{j+1}$
• compute residual	$\mathbf{R}_{j+1} = \boldsymbol{\xi}_{j+1} - \Xi_j\mathbf{c}_j$
• find index of maximum residual	$[\rho, \gamma_j] = \max\|\mathbf{R}_{j+1}\|$
• add new column to measurement matrix	$\mathbf{P}_{j+1} = [\mathbf{P}_j\ \ \mathbf{e}_{\gamma_j}]$

preserving a low-dimensional ($\mathcal{O}(r)$) computation, as desired. The DEIM approach combines projection with interpolation. Specifically, DEIM uses selected interpolation indices to specify an interpolation-based projection for a nearly ℓ_2 optimal subspace approximating the nonlinearity. EIM/DEIM are not the only methods developed to reduce the complexity of evaluating nonlinear terms; see for instance the missing point estimation (MPE) [400, 21] or gappy POD [555, 565, 120, 462] methods. However, they have been successful in a large number of diverse applications and models [127]. In any case, the MPE, gappy POD, and EIM/DEIM use a small selected set of spatial grid points to avoid evaluation of the expensive inner products required to evaluate nonlinear terms.

POD and DEIM

Consider a high-dimensional system of nonlinear differential equations that can arise, for example, from the finite difference discretization of a partial differential equation. In addition to constructing a snapshot matrix (12.12) of the solution of the PDE so that POD modes can be extracted, the DEIM algorithm also constructs a snapshot matrix of the nonlinear term of the PDE:

$$\mathbf{N} = \begin{bmatrix} | & | & & | \\ \mathbf{N}_1 & \mathbf{N}_2 & \cdots & \mathbf{N}_m \\ | & | & & | \end{bmatrix} \tag{12.12}$$

where the columns $\mathbf{N}_k \in \mathbb{C}^n$ are evaluations of the nonlinearity at time t_k.

To achieve high accuracy solutions, n is typically very large, making the computation of the solution expensive and/or intractable. The POD-Galerkin method is a principled dimensionality-reduction scheme that approximates the function $\mathbf{u}(t)$ with rank-r optimal

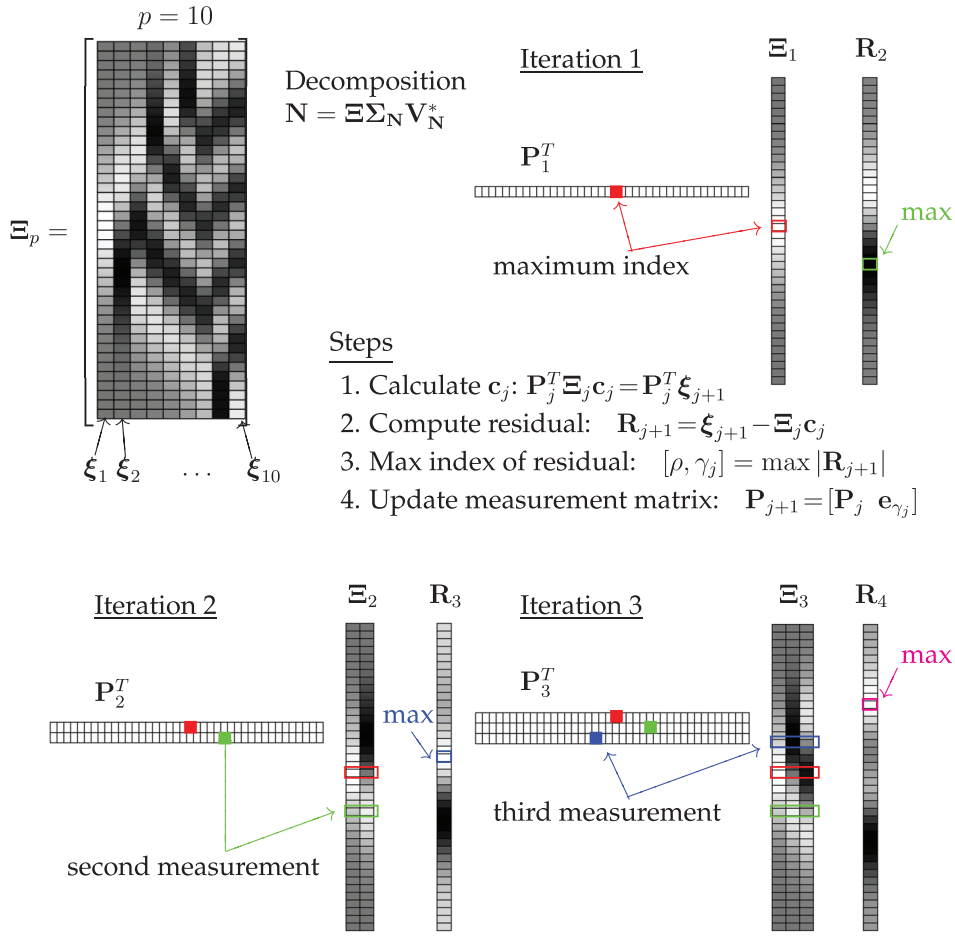

Figure 12.15 Demonstration of the first three iterations of the DEIM algorithm. For illustration only, the nonlinearity matrix $\mathbf{N} = \Xi\Sigma_{\mathbf{N}}\mathbf{V}_{\mathbf{N}}^*$ is assumed to be composed of harmonic oscillator modes with the first ten modes comprising Ξ_p. The initial measurement location is chosen at the maximum of the first mode $\boldsymbol{\xi}_1$. Afterwards, there is a three step process for selecting subsequent measurement locations based upon the location of the maximum of the residual vector $\mathbf{R}_j$. The first (red), second (green) and third (blue) measurement locations are shown along with the construction of the sampling matrix $\mathbf{P}$

basis functions where $r \ll n$. As shown in the previous chapter, these optimal basis functions are computed from a singular value decomposition of a series of temporal snapshots of the complex system.

The standard POD procedure [251] is a ubiquitous algorithm in the reduced order modeling community. However, it also helps illustrate the need for innovations such as DEIM, Gappy POD and/or MPE. Consider the nonlinear component of the low-dimensional evolution (11.21): $\boldsymbol{\Psi}^T\mathbf{N}(\boldsymbol{\Psi}\mathbf{a}(t))$. For a simple nonlinearity such as $N(u(x,t)) = u(x,t)^3$, consider its impact on a spatially-discretized, two-mode POD expansion: $u(x,t) = a_1(t)\psi_1(x) + a_2(t)\psi_2(x)$. The algorithm for computing the nonlinearity requires the evaluation:

$$u(x,t)^3 = a_1^3\psi_1^3 + 3a_1^2a_2\psi_1^2\psi_2 + 3a_1a_2^2\psi_1\psi_2^2 + a_2^3\psi_2^3\,. \tag{12.13}$$

The dynamics of $a_1(t)$ and $a_2(t)$ would then be computed by projecting onto the low-dimensional basis by taking the inner product of this nonlinear term with respect to both ψ_1 and ψ_2. Thus the number of computations not only doubles, but the inner products must be computed with the n-dimensional vectors. Methods such as DEIM overcome this high-dimensional computation.

DEIM

As outlined in the previous section, the shortcomings of the POD-Galerkin method are generally due to the evaluation of the nonlinear term $\mathbf{N}(\mathbf{\Psi a}(t))$. To avoid this difficulty, DEIM approximates $\mathbf{N}(\mathbf{\Psi a}(t))$ through projection and interpolation instead of evaluating it directly. Specifically, a low-rank representation of the nonlinearity is computed from the singular value decomposition

$$\mathbf{N} = \Xi \Sigma_{\mathbf{N}} \mathbf{V}_{\mathbf{N}}^{*} \tag{12.14}$$

where the matrix Ξ contains the optimal basis for spanning the nonlinearity. Specifically, we consider the rank-p basis

$$\Xi_p = [\boldsymbol{\xi}_1 \ \boldsymbol{\xi}_2 \ \cdots \ \boldsymbol{\xi}_p] \tag{12.15}$$

that approximates the nonlinear function ($p \ll n$ and $p \sim r$). The approximation to the nonlinearity $\mathbf{N}$ is given by:

$$\mathbf{N} \approx \Xi_p \mathbf{c}(t) \tag{12.16}$$

where $\mathbf{c}(t)$ is similar to $\mathbf{a}(t)$ in (11.20). Since this is a highly overdetermined system, a suitable vector $\mathbf{c}(t)$ can be found by selecting p rows of the system. The DEIM algorithm was developed to identify which p rows to evaluate.

The DEIM algorithm begins by considering the vectors $\mathbf{e}_{\gamma_j} \in \mathbf{R}^n$ which are the γ_j-th column of the n dimensional identity matrix. We can then construct the projection matrix $\mathbf{P} = [\mathbf{e}_{\gamma_1} \ \mathbf{e}_{\gamma_2} \ \cdots \ \mathbf{e}_{\gamma_p}]$ which is chosen so that $\mathbf{P}^T \Xi_p$ is nonsingular. Then $\mathbf{c}(t)$ is uniquely defined from $\mathbf{P}^T \mathbf{N} = \mathbf{P}^T \Xi_p \mathbf{c}(t)$, and thus,

$$\mathbf{N} \approx \Xi_p (\mathbf{P}^T \Xi_p)^{-1} \mathbf{P}^T \mathbf{N}. \tag{12.17}$$

The tremendous advantage of this result for nonlinear model reduction is that the term $\mathbf{P}^T \mathbf{N}$ requires evaluation of the nonlinearity only at $p \ll n$ indices. DEIM further proposes a principled method for choosing the basis vectors $\boldsymbol{\xi}_j$ and indices γ_j. The DEIM algorithm, which is based on a greedy search, is detailed in [127] and further demonstrated in Table 12.1.

POD and DEIM provide a number of advantages for nonlinear model reduction of complex systems. POD provides a principled way to construct an r-dimensional subspace $\mathbf{\Psi}$ characterizing the dynamics. DEIM augments POD by providing a method to evaluate the problematic nonlinear terms using an p-dimensional subspace Ξ_p that represents the nonlinearity. Thus a small number of points can be sampled to approximate the nonlinear terms in the ROM.

12.6 DEIM Algorithm Implementation

To demonstrate model reduction with DEIM, we again consider the NLS equation (11.29). Recall that the numerical method for solving this equation is given in Codes 11.3 and 11.4.

The output of this code is a matrix **usol** whose rows represent the time snapshots and whose columns represent the spatial discretization points. As in the first section of this chapter, our first step is to transpose this data so that the time snapshots are columns instead of rows. The following code transposes the data and also performs a singular value decomposition to get the POD modes.

Code 12.10 Dimensionality reduction for NLS.

```
X=usol.';  % data matrix X
[U,S,W]=svd(X,0);  % SVD reduction
```

In addition to the standard POD modes, the singular value decomposition of the nonlinear term is also required for the DEIM algorithm. This computes the low-rank representation of $N(u) = |u|^2 u$ directly as $\mathbf{N} = \Xi\Sigma_{\mathbf{N}}\mathbf{V}_{\mathbf{N}}^*$.

Code 12.11 Dimensionality reduction for nonlinearity of NLS.

```
NL=i*(abs(X).^2).*X;
[XI,S_NL,W]=svd(NL,0);
```

Once the low-rank structures are computed, the rank of the system is chosen with the parameter r. In what follows, we choose $r = p = 3$ so that both the standard POD modes and nonlinear modes, Ψ and Ξ_p have three columns each. The following code selects the POD modes for Ψ and projects the initial condition onto the POD subspace.

Code 12.12 Rank selection and POD modes.

```
r=3;  % select rank truncation
Psi=U(:,1:r); % select POD modes
a=Psi'*u0; % project initial conditions
```

We now build the interpolation matrix $\mathbf{P}$ by executing the DEIM algorithm outlined in the last section. The algorithm starts by selecting the first interpolation point from the maximum of the first most dominant mode of Ξ_p.

Code 12.13 First DEIM point.

```
[Xi_max,nmax]=max(abs(XI(:,1)));
XI_m=XI(:,1);
z=zeros(n,1);
P=z; P(nmax)=1;
```

The algorithm iteratively builds $\mathbf{P}$ one column at a time. The next step of the algorithm is to compute the second to rth interpolation point via the greedy DEIM algorithm. Specifically, the vector $\mathbf{c}_j$ is computed from $\mathbf{P}_j^T \Xi_j \mathbf{c}_j = \mathbf{P}_j^T \boldsymbol{\xi}_{j+1}$ where $\boldsymbol{\xi}_j$ are the columns of the nonlinear POD modes matrix Ξ_p. The actual interpolation point comes from looking for the maximum of the residual $\mathbf{R}_{j+1} = \boldsymbol{\xi}_{j+1} - \Xi_j \mathbf{c}_j$. Each iteration of the algorithm produces another column of the sparse interpolation matrix $\mathbf{P}$. The integers **nmax** give the location of the interpolation points.

Code 12.14 DEIM points 2 through r.

```
for j=2:r
    c=(P'*XI_m)\(P'*XI(:,j));
    res=XI(:,j)-XI_m*c;
    [Xi_max,nmax]=max(abs(res));
    XI_m=[XI_m,XI(:,j)];
    P=[P,z]; P(nmax,j)=1;
end
```

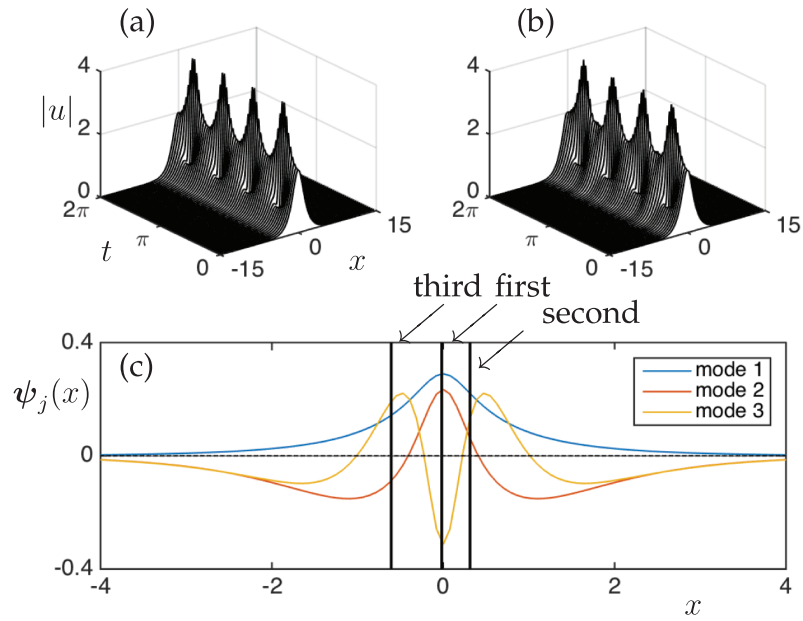

Figure 12.16 Comparison of the (a) full simulation dynamics and (b) rank $r = 3$ ROM using the three DEIM interpolation points. (c) A detail of the three POD modes used for simulation are shown along with the first, second and third DEIM interpolation point location. These three interpolation points are capable of accurately reproducing the evolution dynamics of the full PDE system.

With the interpolation matrix, we are ready to construct the ROM. The first part is to construct the linear term $\mathbf{\Psi}^T\mathbf{L}\mathbf{\Psi}$ of (11.21) where the linear operator for NLS is the Laplacian. The derivatives are computed using the Fourier transform.

Code 12.15 Projection of linear terms.

```
for j=1:r  % linear derivative terms
    Lxx(:,j)=ifft(-k.^2.*fft(Psi(:,j)));
end
L=(i/2)*(Psi')*Lxx;  % projected linear term
```

The projection of the nonlinearity is accomplished using the interpolation matrix $\mathbf{P}$ with the formula (12.17). Recall that the nonlinear term in (11.21) is multiplied by $\mathbf{\Psi}^T$. Also computed is the interpolated version of the low-rank subspace spanned by $\mathbf{\Psi}$.

Code 12.16 Projection of nonlinear terms.

```
P_NL=Psi'*( XI_m*inv(P'*XI_m) ); % nonlinear projection
P_Psi=P'*Psi; % interpolation of Psi
```

It only remains now to advance the solution in time using a numerical time stepper. This is done with a 4th-order Runge-Kutta routine.

Code 12.17 Time stepping of ROM.

```
[tt,a]=ode45('rom_deim_rhs',t,a,[],P_NL,P_Psi,L);
Xtilde=Psi*a';  % DEIM approximation
waterfall(x,t,abs(Xtilde')), shading interp, colormap gray
```

The right hand side of the time stepper is now completely low dimensional.

Code 12.18 Right hand side of ROM.

```
function rhs=rom_deim_rhs(tspan, a,dummy,P_NL,P_Psi,L)
N=P_Psi*a;
rhs=L*a + i*P_NL*((abs(N).^2).*N);
```

A comparison of the full simulation dynamics and rank $r = 3$ ROM using the three DEIM interpolation points is shown in Fig. 12.16. Additionally, the location of the DEIM points relative to the POD modes is shown. Aside from the first DEIM point, the other locations are not on the minima or maxima of the POD modes. Rather, the algorithms places them to maximize the residual.

QDEIM Algorithm

Although DEIM is an efficient greedy algorithm for selecting interpolation points, there are other techniques that are equally efficient. The recently proposed QDEIM algorithm [159] leverages the QR decomposition to provide efficient, greedy interpolation locations. This has been shown to be a robust mathematical architecture for sensor placement in many applications [366]. See Section 3.8 for a more general discussion. The QR decomposition can also provide a greedy strategy to identify interpolation points. In QDEIM, the QR pivot locations are the sensor locations. The following code can replace the DEIM algorithm to produce the interpolation matrix $\mathbf{P}$.

Code 12.19 QR based interpolation points

```
[Q,R,pivot]=qr(NL.');
P=pivot(:,1:r);
```

Using this interpolation matrix gives identical interpolation locations as shown in Fig. 12.16. More generally, there are estimates that show that the QDEIM may improve error performance over standard DEIM [159]. The ease of use of the QR algorithm makes this an attractive method for sparse interpolation.

12.7 Machine Learning ROMs

Inspired by machine learning methods, the various POD bases for a parametrized system are merged into a master library of POD modes Ψ_L which contains all the low-rank subspaces exhibited by the dynamical system. This leverages the fact that POD provides a principled way to construct an r-dimensional subspace Ψ_r characterizing the dynamics while sparse sampling augments the POD method by providing a method to evaluate the problematic nonlinear terms using a p-dimensional subspace projection matrix $\mathbf{P}$. Thus a small number of points can be sampled to approximate the nonlinear terms in the ROM. Fig. 12.17 illustrates the library building procedure whereby a dynamical regime is sampled in order to construct an appropriate POD basis Ψ.

The method introduced here capitalizes on these methods by building low-dimensional libraries associated with the full nonlinear system dynamics as well as the specific nonlinearities. Interpolation points, as will be shown in what follows, can be used with sparse representation and compressive sensing to (i) identify dynamical regimes, (ii) reconstruct the full state of the system, and (iii) provide an efficient nonlinear model reduction and POD-Galerkin prediction for the future state.

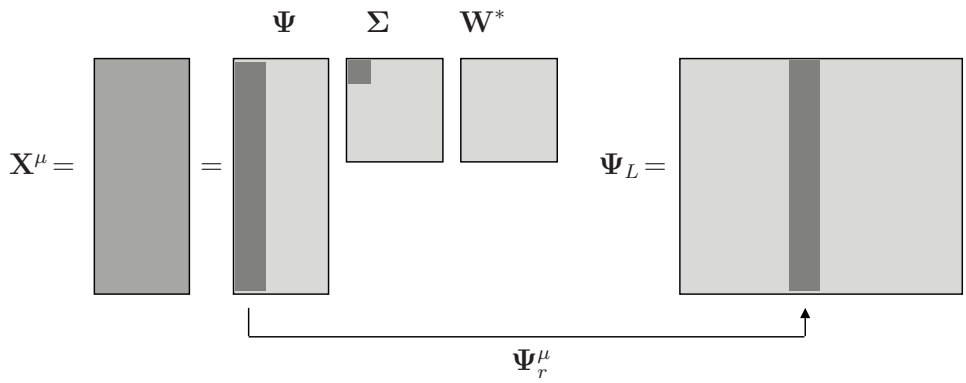

Figure 12.17 Library construction from numerical simulations of the governing equations (11.1). Simulations are performed of the parametrized system for different values of a bifurcation parameter μ. For each regime, low-dimensional POD modes Ψ_r are computed via an SVD decomposition. The various rank-r truncated subspaces are stored in the library of modes matrix Ψ_L. This is the learning stage of the algorithm. (*reproduced from Kutz et al. [319]*)

The concept of library building of low-rank *features* from data is well established in the computer science community. In the reduced-order modeling community, it has recently become an enabling computational strategy for parametric systems. Indeed, a variety of recent works have produced libraries of ROM models [80, 98, 462, 10, 134, 422, 421, 420] that can be selected and/or interpolated through measurement and classification. Alternatively, cluster-based reduced order models use a k-means clustering to build a Markov transition model between dynamical states [278]. These recent innovations are similar to the ideas advocated here. However, our focus is on determining how a suitably chosen **P** can be used across all the libraries for POD mode selection and reconstruction. One can also build two sets of libraries: one for the full dynamics and a second for the nonlinearity so as to make it computationally efficient with the DEIM strategy [462]. Before these more formal techniques based on machine learning were developed, it was already realized that parameter domains could be decomposed into subdomains and a local ROM/POD computed in each subdomain. Patera and co-workers [171] used a partitioning based on a binary tree whereas Amsallem *et al.* [9] used a Voronoi tessellation of the domain. Such methods were closely related to the work of Du and Gunzburger [160] where the data snapshots were partitioned into subsets and multiple reduced bases computed. The multiple bases were then recombined into a single basis, so it doesn't lead to a library, per se. For a review of these domain partitioning strategies, please see Ref. [11].

POD Mode Selection

Although there are a number of techniques for selecting the correct POD library elements to use, including the workhorse k-means clustering algorithm [10, 134, 422, 421, 420], one can also instead make use of sparse sampling and the sparse representation for classification (SRC) innovations outlined in Chapter 3 to characterize the nonlinear dynamical system [80, 98, 462]. Specifically, the goal is to use a limited number of sensors (interpolation points) to classify the dynamical regime of the system from a range of potential POD

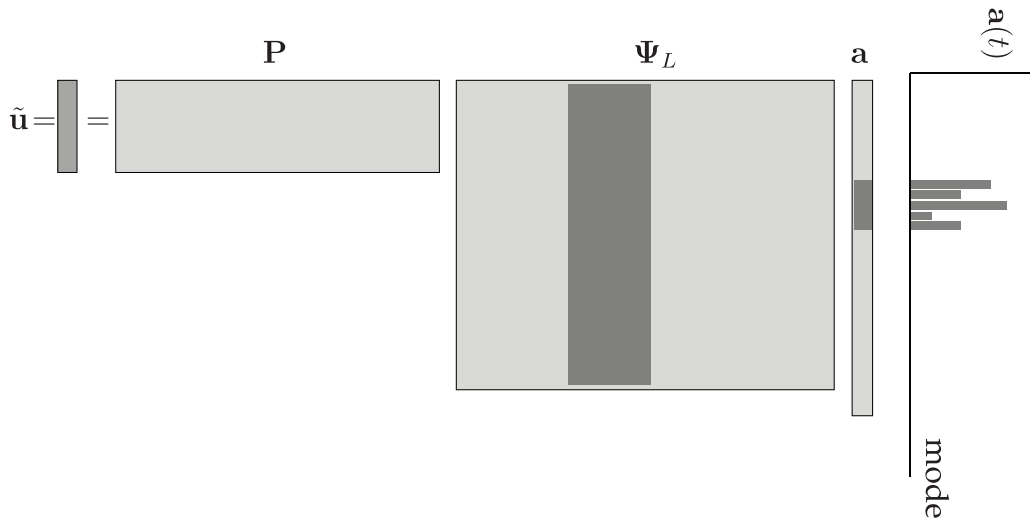

Figure 12.18 The sparse representation for classification (SRC) algorithm for library mode selection; see Section 3.6 for more details. In this mathematical framework, a sparse measurement is taken of the system (11.1) and a highly under-determined system of equations $\mathbf{P}\Psi_L\mathbf{a} = \tilde{\mathbf{u}}$ is solved subject to ℓ_1 penalization so that $\|\mathbf{a}\|_1$ is minimized. Illustrated is the selection of the μth POD modes. The bar plot on the left depicts the nonzero values of the vector $\mathbf{a}$ which correspond to the Ψ_r library elements. Note that the sampling matrix $\mathbf{P}$ that produces the sparse sample $\tilde{\mathbf{u}} = \mathbf{P}\mathbf{u}$ is critical for success in classification of the correct library elements Ψ_r and the corresponding reconstruction. (*reproduced from Kutz et al. [319]*)

library elements characterized by a parameter β. Once a correct classification is a achieved, a standard ℓ_2 reconstruction of the full state space can be accomplished with the selected subset of POD modes, and a POD-Galerkin prediction can be computed for its future.

In general, we will have a sparse measurement vector $\tilde{\mathbf{u}}$ given by (12.1). The full state vector $\mathbf{u}$ can be approximated with the POD library modes ($\mathbf{u} = \Psi_L\mathbf{a}$), therefore

$$\tilde{\mathbf{u}} = \mathbf{P}\Psi_L\mathbf{a}, \tag{12.18}$$

where Ψ_L is the low-rank matrix whose columns are POD basis vectors concatenated across all β regimes and $\mathbf{c}$ is the coefficient vector giving the projection of $\mathbf{u}$ onto these POD modes. If $\mathbf{P}\Psi_L$ obeys the restricted isometry property and $\mathbf{u}$ is sufficiently sparse in Ψ_L, then it is possible to solve the highly-underdetermined system (12.18) with the sparsest vector $\mathbf{a}$. Mathematically, this is equivalent to an ℓ_0 optimization problem which is *np*-hard. However, under certain conditions, a sparse solution of equation (12.18) can be found (See Chapter 3) by minimizing the l_1 norm instead so that

$$\mathbf{c} = \arg\min_{\mathbf{a}'} ||\mathbf{a}'||_1, \quad \text{subject to} \quad \tilde{\mathbf{u}} = \mathbf{P}\Psi_L\mathbf{a}. \tag{12.19}$$

The last equation can be solved through standard convex optimization methods. Thus the ℓ_1 norm is a proxy for sparsity. Note that we only use the sparsity for classification, not reconstruction. Fig. 12.18 demonstrates the sparse sampling strategy and prototypical results for the sparse solution $\mathbf{a}$.

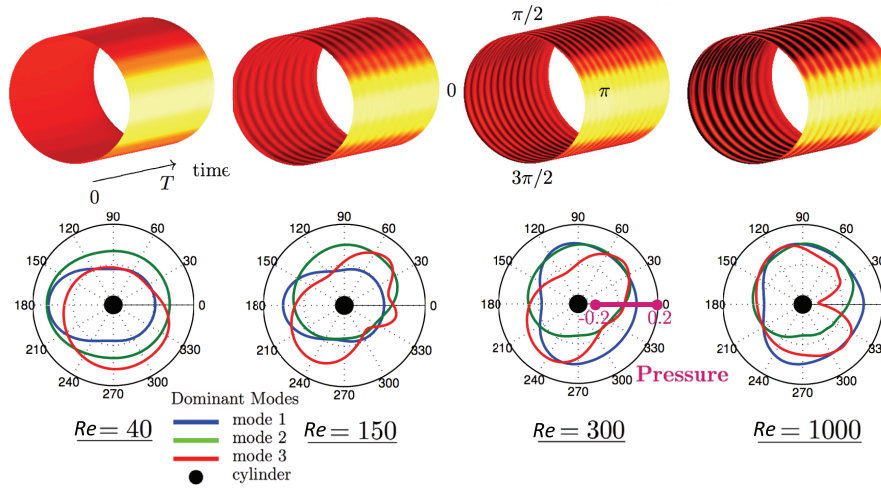

Figure 12.19 Time dynamics of the pressure field (top panels) for flow around a cylinder for Reynolds number $Re = 40, 150, 300$ and 1000. Collecting snapshots of the dynamics reveals low-dimensional structures dominate the dynamics. The dominant three POD pressure modes for each Reynolds number regime are shown in polar coordinates. The pressure scale is in magenta (bottom left). (*reproduced from Kutz et al. [319]*)

Example: Flow around a Cylinder

To demonstrate the sparse classification and reconstruction algorithm developed, we consider the canonical problem of flow around a cylinder. This problem is well understood and has already been the subject of studies concerning sparse spatial measurements [80, 98, 462, 281, 374, 89, 540]. Specifically, it is known that for low to moderate Reynolds numbers, the dynamics are spatially low-dimensional and POD approaches have been successful in quantifying the dynamics. The Reynolds number, Re, plays the role of the bifurcation parameter β in (11.1), i.e. it is a parametrized dynamical system.

The data we consider comes from numerical simulations of the incompressible Navier-Stokes equation:

$$\frac{\partial u}{\partial t} + u \cdot \nabla u + \nabla p - \frac{1}{Re}\nabla^2 u = 0 \tag{12.20a}$$

$$\nabla \cdot u = 0 \tag{12.20b}$$

where $u(x, y, t) \in \mathbb{R}^2$ represents the 2D velocity, and $p(x, y, t) \in \mathbb{R}^2$ the corresponding pressure field. The boundary condition are as follows: (i) Constant flow of $u = (1, 0)^T$ at $x = -15$, i.e., the entry of the channel, (ii) Constant pressure of $p = 0$ at $x = 25$, i.e., the end of the channel, and (iii) Neumann boundary conditions, i.e. $\frac{\partial u}{\partial \mathbf{n}} = 0$ on the boundary of the channel and the cylinder (centered at $(x, y) = (0, 0)$ and of radius unity).

For each relevant value of the parameter Re we perform an SVD on the data matrix in order to extract POD modes. It is well known that for relatively low Reynolds number, a fast decay of the singular values is observed so that only a few POD modes are needed to characterize the dynamics. Fig. 12.19 shows the 3 most dominant POD modes for Reynolds number $Re = 40, 150, 300, 1000$. Note that 99% of the total energy (variance) is selected

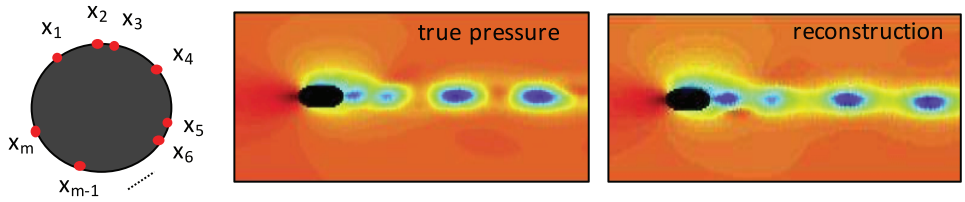

Figure 12.20 Illustration of m sparse sensor locations (left panel) for classification and reconstruction of the flow field. The selection of sensory/interpolation locations can be accomplished by various algorithms [80, 98, 462, 281, 374, 89, 540]. For a selected algorithm, the sensing matrix $\mathbf{P}$ determines the classification and reconstruction performance. (*reproduced from Kutz et al. [319]*)

for the POD mode selection cut-off, giving a total of 1, 3, 3, and 9 POD modes to represent the dynamics in the regimes shown. For a threshold of 99.9%, more modes are required to account for the variability.

Classification of the Reynolds number is accomplished by solving the optimization problem (12.19) and obtaining the sparse coefficient vector $\mathbf{a}$. Note that each entry in $\mathbf{a}$ corresponds to the energy of a single POD mode from our library. For simplicity, we select a number of local minima and maxima of the POD modes as sampling locations for the matrix $\mathbf{P}$. The classification of the Reynolds number is done by summing the absolute value of the coefficient that corresponds to each Reynolds number. To account for the large number of coefficients allocated for the higher Reynolds number (which may be 16 POD modes for 99.9% variance at $Re = 1000$, rather than a single coefficient for Reynolds number 40), we divide by the square root of the number of POD modes allocated in $\mathbf{a}$ for each Reynolds number. The classified regime is the one that has the largest magnitude after this process.

Although the classification accuracy is high, many of the false classifications are due to categorizing a Reynolds number from a neighboring flow, i.e. Reynolds 1000 is often mistaken for Reynolds number 800. This is due to the fact that these two Reynolds numbers are strikingly similar and the algorithm has a difficult time separating their modal structures. Fig. 12.20 shows a schematic of the sparse sensing configuration along with the reconstruction of the pressure field achieved at $Re = 1000$ with 15 sensors. Classification and reconstruction performance can be improved using other methods for constructing the sensing matrix $\mathbf{P}$ [80, 98, 462, 281, 374, 89, 540]. Regardless, this example demonstrate the usage of sparsity promoting techniques for POD mode selection (ℓ_1 optimization) and subsequent reconstruction (ℓ_2 projection).

Finally, to visualize the entire sparse sensing and reconstruction process more carefully, Fig. 12.21 shows both the Reynolds number reconstruction for the time-varying flow field along with the pressure field and flow field reconstructions at select locations in time. Note that the SRC scheme along with the supervised ML library provide an effective method for characterizing the flow strictly through sparse measurements. For higher Reynolds numbers, it becomes much more difficult to accurately classify the flow field with such a small number of sensors. However, this does not necessarily jeopardize the ability to reconstruct the pressure field as many of the library elements at higher Reynolds numbers are fairly similar.

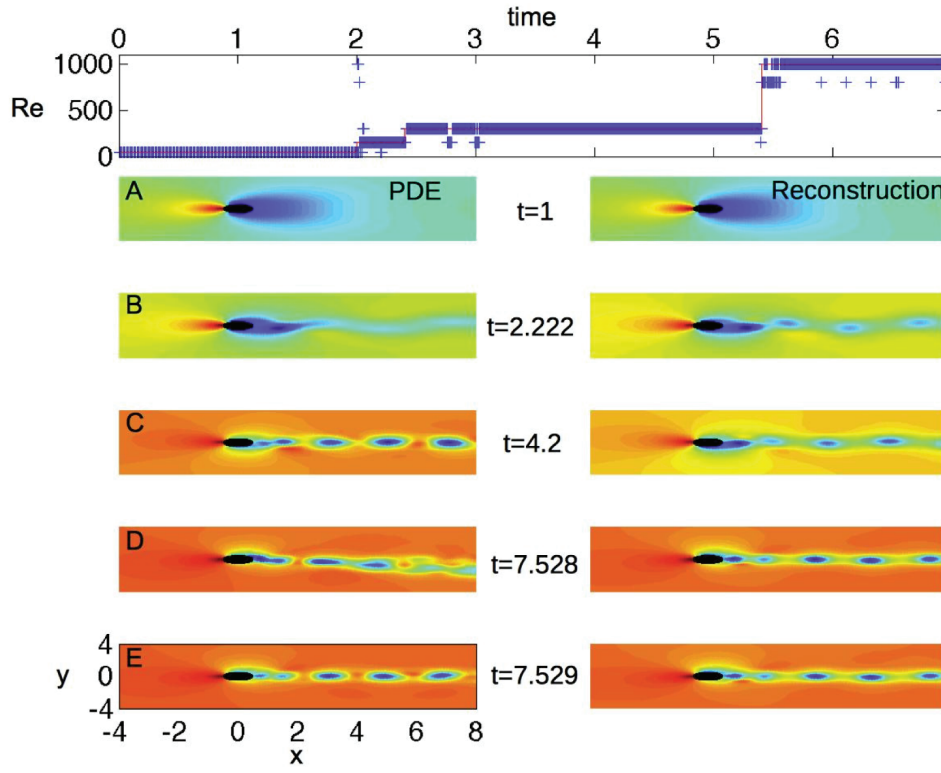

Figure 12.21 Sparse-sensing Reynolds number identification and pressure-field reconstruction for a time-varying flow. The top panel shows the actual Reynolds number used in the full simulation (solid line) along with its compressive sensing identification (crosses). Panels A-D show the reconstruction of the pressure field at four different locations in time (top panel) demonstrating an accurate (qualitatively) reconstruction of the pressure field. (The left side the simulated pressure field is presented, while the right side contains the reconstruction.) Note that for higher Reynolds numbers, the classification becomes more difficult. (*reproduced from Bright et al. [80]*)

Suggested Reading

Texts

(1) **Certified reduced basis methods for parametrized partial differential equations**, by J. Hesthaven, G. Rozza and B. Stamm, 2015 [244].

(2) **Reduced basis methods for partial differential equations: An introduction**, by A. Quarteroni, A. Manzoni and N. Federico, 2015 [442].

(3) **Model reduction and approximation: Theory and algorithms**, by P. Benner, A. Cohen, M. Ohlberger and K. Willcox, 2017 [54].

Papers and Reviews

(1) **A survey of model reduction methods for parametric systems**, by P. Benner, S. Gugercin and K. Willcox, *SIAM Review*, 2015 [53].

(2) **Model reduction using proper orthogonal decomposition**, by S. Volkwein, *Lecture Notes, Institute of Mathematics and Scientific Computing, University of Graz*, 2011 [542].

(3) **Nonlinear model reduction for dynamical systems using sparse sensor locations from learned libraries**, by S. Sargsyan, S. L. Brunton and J. N. Kutz, *Physical Review E*, 2015 [462].

(4) **An online method for interpolating linear parametric reduced-order models**, by D. Amsallem and C. Farhat, *SIAM Journal of Scientific Computing*, 2011 [10].