The University of Texas at Austin

A Randomized Proper Orthogonal Decomposition Method for Reducing Large Linear Systems

Brad T. Marvin

Supervisor:
Dr. Tan Bui-Thanh

Second Reader:
Dr. Rachel Ward

This thesis is submitted in partial fulfillment
of the requirements for the Engineering
Honors Program

May 18, 2015

Abstract

The proper orthogonal decomposition (POD) method is a powerful tool for reducing large data systems which can quickly overwhelm modern computing tools. In this thesis we provide a link between randomized projections and statistical methods by introducing the randomized POD method. We also apply the POD method to a heat transfer finite element model and image compression. In doing so we demonstrate the practical use and quantify the error introduced by the POD method.

Acknowledgements

I wish to express my sincere thanks to my supervisor, Dr. Tan Bui for his guidance and support over the past year. I am very thankful for his patience while bringing me up to speed on POD methods. This project would have not been possible without his mentorship. I also thank Dr. Rachel Ward for her helpful input. Finally I thank my family and friends for their continued encouragement.

Contents

1 Introduction

1.1 Problem Statement

Suppose we are given a linear system $Ax = b$ and tasked with finding the solution x. This problem is simple when the dimensions of A are small, but when A is very large, issues arise with memory and computational power required to compute solutions. For example, IBM's Watson computer which competed on Jeopardy in 2011 was required to compute solutions to questions by analyzing over 1 Terabyte [1] of data in a span of seconds.

For the case of Watson and our large linear system, we need a systematic way of reducing our data to the most important information and then using that reduced system to compute a solution. Randomized proper orthogonal decomposition methods solve this issue by finding the most important directions within a set of data and using that information to predict a solution rapidly.

1.2 Structure of Thesis

This thesis is divided in to three major sections: Background (Section 2), Theoretical Results (Section 3), and Application (Section 4). In the background section we establish preliminary results needed to understand our results. In the theoretical results section we make a new connection between randomized POD and randomized projection methods. In the application section we show how random projections and randomized POD can be used to solve a heat conduction model and to compress an image. We use the results of the exact heat conduction model to quantify the error incurred by using a POD method.

2 Background

2.1 Proper Orthogonal Decomposition

The goal of the proper orthogonal decomposition method is to seek the most dominant subspace of a set of vectors. This subspace (a.k.a reduced basis) can then be used to solve

the problem under consideration more efficiently. For example, consider a linear system

$$Ax = b, \tag{1}$$

where $A \in \mathbb{R}^{m \times m}$ and A may be a function of time or other parameters. We first find a reduced basis $V \in \mathbb{R}^{m \times k}$ which spans the dominant subspace of A. We then assume that $x \approx Vx_r$, where x_r is the reduced solution. Next we use the Galerkin projection method to establish the reduced system

$$V^T AVx_r = V^T b, \tag{2}$$

and solve for x_r. The full solution can then be recovered by $x \approx Vx_r$. Solving Equation 2 now requires solving a $k \times k$ system instead of an $m \times m$ system required by Equation 1. If $k \ll m$ then this procedure greatly reduces the amount of computing required to solve our original system. In the next three sections we will discuss how the reduced basis V is constructed.

2.2 Snapshot Matrix

Suppose A is a function of a set of parameters μ, i.e. $A = A(\mu)$, each time we want to solve $A(\mu)x = b$ with a new set of parameters we need to solve a new $m \times m$ system of equations. If instead we wish to solve the reduced $k \times k$ system of equations shown in Equation 2, then we must form V so that it captures the most dominant subspace of A for every possible choice of parameters.

Suppose $\mathcal{D}$ represents the space spanned by every possible choice of inputs and we discretize this space into n points. We solve the full system for each point in $\mathcal{D}$ and store the solution in a column of a matrix $X \in \mathbb{R}^{m \times n}$, known as a snapshot matrix. We then assume that any solution of the system $A(\mu)x = b$ can be approximated by the column space of X. Now that we have captured the desired subspace of X, all that is left is to find an orthonormal basis V which approximates the column space of X.

2.3 Model Order Reduction Using Exact SVD

One way to form the reduced basis V is to represent the problem as finding the unit vector v which maximizes $\|X^T v\|_2$, i.e. the vector v which best aligns with the span of X. This can be solved by performing a constrained optimization of the cost function J, i.e.

$$\min_v J = -\frac{1}{2}\|X^T v\|^2 \quad s.t. \quad \|v\|^2 = 1.$$

This constrained optimization problem can then be represented in a Lagrange multiplier form:

$$\min_{v,\Lambda} \mathcal{L} = -\frac{1}{2}\|X^T v\|^2 + \frac{\Lambda}{2}(\|v\|^2 - 1).$$

The benefit of the Lagrangian multiplier approach is that the optimization problem is now unconstrained with a cost of introducing a new unknown Λ. Carrying out the optimization gives the following result.

Lemma 1. *v maximizes $\|X^T v\|$ when (v, Λ) is the eigenpair of XX^T.*

Proof. Recall from before, maximizing $\|X^T v\|$ with respect to v subject to $\|v\|^2 = 1$ is equivalent to minimizing $\mathcal{L}$ with respect to v. To carry out the optimization we require that the gradient of $\mathcal{L}$ go to zero:

$$\frac{\partial \mathcal{L}}{\partial v} = -XX^T v + \Lambda v = 0,$$

which yields:

$$XX^T v = \Lambda v.$$

□

The eigenpair (v, Λ) is the dominant left-singular vector and singular value of X. Taking the singular value decomposition $X = U\Sigma V^T$ gives all of these pairs in the columns of U and along the diagonal of Σ. From the result of Lemma 1 it is apparent that the problem of maximizing $\|X^T v\|$ is solved by computing the singular value decomposition (SVD) of X.

A reduced basis composed of k left-singular vectors can be achieved by truncating the SVD to the k largest singular values. The truncated SVD is denoted as:

$$X_k = U_k \Sigma_k V_k^T,$$

where $U_k \in \mathbb{R}^{m \times k}$, $\Sigma_k \in \mathbb{R}^{k \times k}$, and $V_k \in \mathbb{R}^{n \times k}$.

Using a truncated SVD provides a direct method for computing the reduced basis, but comes at a significant computational cost. The reason is that computing the SVD of an $m \times n$ matrix requires $\mathcal{O}(mn \min\{m, n\})$ operations. Halko et al [3] showed that a more efficient method is to compute the partial QR factorization which can then be used to recover a truncated SVD with $\mathcal{O}(kmn)$ operations. To further reduce the computational cost of computing a reduced basis, it was shown by Sarlós [4] that random projection methods could be used to approximate the SVD of large matrices at a reduced cost. The key principle behind random projection methods was first described by Johnson and Lindenstrauss [5] and is described in the following section.

2.4 Johnson-Lindenstrauss Lemma

Johnson and Lindenstrauss [5] showed that a set of n points in d dimensions can be projected on to a k dimensional space without distorting the distance between any two points more than a factor of $(1 \pm \epsilon)$ provided that k satisfy:

$$k \geq \frac{4 \log n}{\epsilon^2/2 - \epsilon^3/3}.$$

The resulting distortion is bounded by:

$$(1 - \epsilon)\|u - v\|^2 \leq \|f(u) - f(v)\|^2 \leq (1 + \epsilon)\|u - v\|^2, \tag{3}$$

where u and v are any of the original high-dimensional points and f is a map from $\mathbb{R}^d \to \mathbb{R}^k$.

The importance of the Johnson-Lindenstrauss Lemma for our purpose is that it indicates that we can project our high dimensional matrix X on to a lower dimensional subspace

$$\tilde{X} = \frac{1}{\sqrt{k}} X\varepsilon$$

without loosing much information due to Equation 3. The SVD can then be computed on this lower dimensional matrix $\tilde{X}$ to reduce the computational cost.

2.5 Random Projections

Most of the projection matrices used to carry out Johnson-Lindenstrauss embeddings draw from random distributions to project the high dimensional data on a random subspace. It was shown by Achlioptas [6] that several random distributions preserve the bounds described by Johnson and Lindenstrauss [5]. Achlioptas suggested two random distributions as alternatives to the obvious choice of drawing from a normal distribution. The first being a Rademacher distribution given by:

$$\varepsilon_{ij} = \begin{cases} +1 & \text{with probability } 1/2 \\ -1 & \text{with probability } 1/2 \end{cases}.$$

The second distribution utilizes sparsity to reduce the cost of computing $X\varepsilon$. We later refer to this distribution simply as an Achlioptas distribution:

$$\varepsilon_{ij} = \sqrt{3} \times \begin{cases} +1 & \text{with probability } 1/6 \\ 0 & \text{with probability } 2/3 \\ -1 & \text{with probability } 1/6 \end{cases}.$$

Achlioptas [6] showed that each of these distribution choices can result in a successful Johnson-Lindenstrauss embedding with a probability of at least $1 - e^{-\beta}$ if

$$k \geq \frac{4 + 2\beta}{\epsilon^2/2 - \epsilon^3/3} \log n.$$

3 Theoretical Results

Our first result is a novel method for showing that we can form an approximate reduced basis $\tilde{V}$ with columns $\tilde{v}$ using a random projection provided that $\mathbb{E}[\varepsilon] = 0$ and $\mathrm{Var}[\varepsilon] = I$.

Theorem 1. *Lemma 1 holds when $(\tilde{v}, \tilde{\Lambda})$ is the eigenpair of $\tilde{X}\tilde{X}^T$ and $\tilde{X} = \frac{1}{\sqrt{k}} X\varepsilon$.*

Proof. Recall from Section 2.3 that maximizing $\|X^T v\|$ is equivalent to the optimization problem:

$$\min_v \mathcal{L} = -\frac{1}{2}\|X^T v\|^2 + \frac{\Lambda}{2}(\|v\|^2 - 1).$$

Using the trace property $\|A\|_2 = \sqrt{\mathrm{Tr}(A^T A)}$, we have

$$\mathcal{L} = -\frac{1}{2}\,\mathrm{Tr}(v^T X X^T v) + \frac{\Lambda}{2}(\|v\|^2 - 1).$$

We now introduce a new term $\mathbb{E}[\varepsilon\varepsilon^T]$ into the expression. This does not change our original expression if $\mathbb{E}[\varepsilon] = 0 \quad \mathbb{E}[\varepsilon\varepsilon^T] = \mathrm{Var}[\varepsilon] = I$.

$$\mathcal{L} = -\frac{1}{2}\,\mathrm{Tr}(v^T X \mathbb{E}[\varepsilon\varepsilon^T] X^T v) + \frac{\Lambda}{2}(\|v\|_2^2 - 1).$$

The trace operator and expectation can be interchanged to get

$$\mathcal{L} = -\frac{1}{2}\mathbb{E}[\mathrm{Tr}(v^T X \varepsilon\varepsilon^T X^T v)] + \frac{\Lambda}{2}(\|v\|_2^2 - 1).$$

Recognizing that $v^T X\varepsilon$ and $\varepsilon^T X^T v$ are both scalar simplifies the expression to

$$\mathcal{L} = -\frac{1}{2}\mathbb{E}[v^T X \varepsilon\varepsilon^T X^T v] + \frac{\Lambda}{2}(\|v\|_2^2 - 1).$$

We now introduce the Monte Carlo approximation

$$\mathbb{E}[f(x)] \approx \frac{1}{k}\sum_{i=1}^{k} f(x_i),$$

and then define $\tilde{\mathcal{L}}$ such that

$$\tilde{\mathcal{L}} = -\frac{1}{2}\frac{v^T}{k}\sum_{i=1}^{k}(X\varepsilon^i)(X\varepsilon^i)^T v + \frac{\Lambda}{2}(\|v\|_2^2 - 1).$$

Carrying out the minimization of $\tilde{\mathcal{L}}$ with respect to v gives

$$\frac{\partial\tilde{\mathcal{L}}}{\partial v} = -\frac{1}{k}\sum_{i=1}^{k}(X\varepsilon^i)(X\varepsilon^i)^T v + \Lambda v = 0. \tag{4}$$

We define a new variable $\tilde{X} := \frac{1}{\sqrt{k}}X\varepsilon$, then Equation 4 becomes

$$\tilde{X}\tilde{X}^T\tilde{v} = \tilde{\Lambda}\tilde{v}.$$

□

It should be noted that Theorem 1 is a new way of arriving at the random projection method described in Section 2.4. If $\tilde{X} \in \mathbb{R}^{m\times k}$ then approximating v with $\tilde{v}$ using the result of Theorem 1 requires computing the eigendecomposition of an $m \times m$ matrix. This computation can be prohibitively costly if m is very large. If $k \ll m$ then a better method would require computing the eigendecomposition of a $k \times k$ matrix. This can be achieved using the following result, known as the method of snapshots, first introduced by Sirovich [2]. Here we present a new way to obtain the method of snapshots.

Theorem 2. *$\tilde{w}$ maximizes $\|\tilde{X}^T\tilde{v}\|$ such that $\tilde{v} = \tilde{X}\tilde{w}$ when $(\tilde{w}, \tilde{\Lambda})$ is the eigenpair of $\tilde{X}^T\tilde{X}$ and $\tilde{X} = \frac{1}{\sqrt{k}}X\varepsilon$.*

Proof. Recall that maximizing $\|X^T v\|$ subject to $\|v\|^2 = 1$ is equivalent to minimizing the Lagrangian multiplier $\mathcal{L} = -\frac{1}{2}\|X^T v\|_2^2 + \frac{\Lambda}{2}(\|v\|^2 - 1)$. Now let us define $v = Xw$ to

obtain

$$\mathcal{L} = -\frac{1}{2}\|X^T X w\|^2 + \frac{\Lambda}{2}(\|Xw\|^2 - 1).$$

Carrying out the minimization with respect to w

$$\frac{\partial \mathcal{L}}{\partial w} = -X^T X X^T X w + \Lambda X^T X w = 0,$$

which reduces to

$$X^T X w = \Lambda w.$$

Solving for w which maximizes $\|X^T v\|$ leads us to finding the right-singular vectors of X. We then recover v by $v = Xw$. Notice that $v = U\Sigma V^T w$, so the columns of V recovered via this method are the left-singular vectors of X scaled by their corresponding singular values. Now we define $\tilde{v} = \tilde{X}\tilde{w}$ with the same approach to obtain:

$$\tilde{X}^T \tilde{X} \tilde{w} = \tilde{\Lambda} \tilde{w} \quad \text{where} \quad \tilde{X} := \frac{1}{\sqrt{k}} X \varepsilon.$$

□

Solving $\tilde{X}^T \tilde{X} \tilde{w} = \tilde{\Lambda} \tilde{w}$ amounts to performing an eigendecomposition on a matrix with dimensions $k \times k$. This method is thereby more efficient than the method outlined in Theorem 1 when $k \ll m$.

The convergence of the cost function $\tilde{J} = -\frac{1}{2}\|\tilde{X}^T \tilde{X} \tilde{w}\|_2^2$ with respect to the number of random vectors k is shown in Figure 1. To numerically evaluate $\tilde{J}$ a snapshot matrix X with dimensions 1333 by 31875 is used to compute $\tilde{w}$ using a Achlioptas distribution [6] for several values of k. The cost function is then evaluated for the first vector in each $\tilde{w}$ and compared against the cost function evaluated with X and w taken from the exact SVD. The results of this procedure show that the relative error between the cost function evaluated with the approximate and exact dominant right-singular vector converges with $\mathcal{O}(\frac{1}{\sqrt{k}})$, which is predicted by Monte Carlo theory.

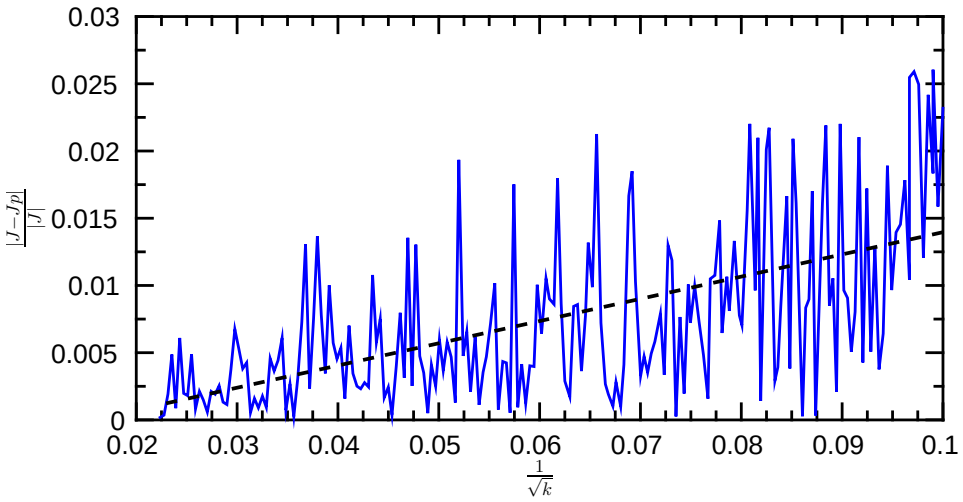
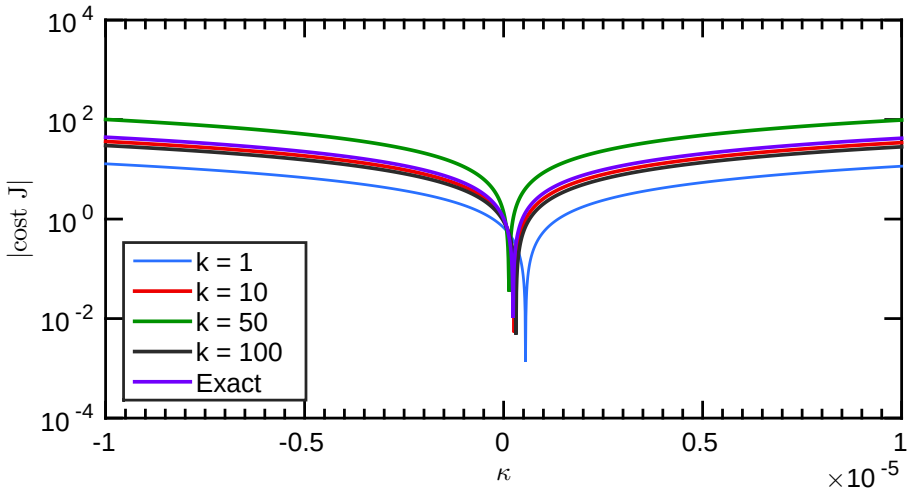


Figure 1: Convergence of the cost function with respect to the number of vectors in the random projection. The cost function converges with $\mathcal{O}(\frac{1}{\sqrt{k}})$. The cost for each k value is computed 75 times and averaged to make the convergence trend more clear.

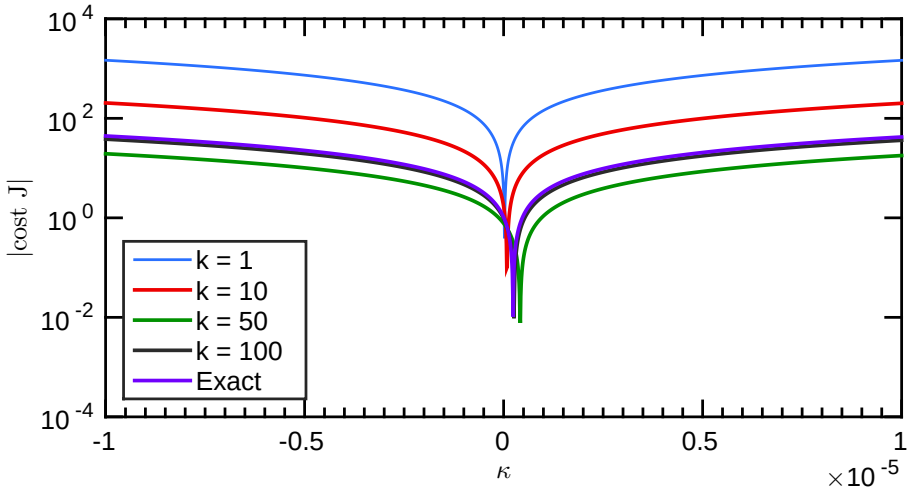
Next we verify that the method of snapshots minimizes the cost function for any choice of k using the three random distributions mentioned in Section 2.5 (Achlioptas, normal, and Rademacher). First we define a new cost function

$$J(\kappa) := -\frac{1}{2}\|X^T X(w_0 + \kappa \nabla J(w_0))\|^2.$$

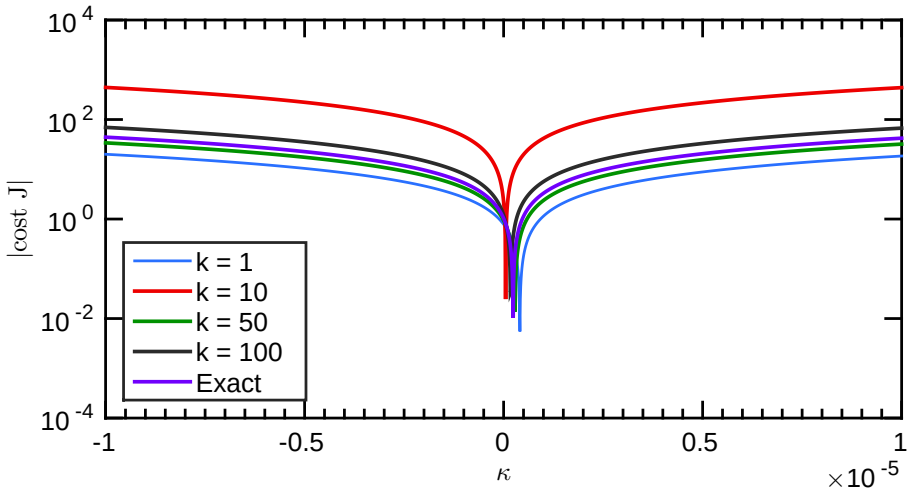
This new function is equivalent to our original cost function in the case that $\kappa = 0$ and $w = w_0$. We chose w_0 to be the dominant right-singular vector of $\tilde{X}$. If the method outlined in Theorem 2 truly minimizes $\|X^T X w\|$ then we expect $J(\kappa)$ to have a minimum at $\kappa = 0$. A snapshot matrix X with dimensions 1333×31875 was used to compute $\tilde{w}$ and evaluate the cost function. Figure 2 shows the results of this procedure. All k and distribution choices show a minimum at approximately $\kappa = 0$ as expected.



(a) Achlioptas Distribution



(b) Normal Distribution



(c) Rademacher Distribution

Figure 2: $J(\kappa) := J(w_0 + \kappa \nabla J(w_0))$.

4 Application

4.1 Heat Transfer FEM Model

We applied the randomized POD method to a heat transfer finite-element model to illustrate the method's potential use for time dependent systems. The particular model represents a heat sink structure with several fins to facilitate convective cooling. The mesh consists of 1333 node points and is shown in Figure 3. The inputs to this system include the thermal conductivity of each set of fins relative to that of the center section and the Biot number for the structure's surface. This model forms the system

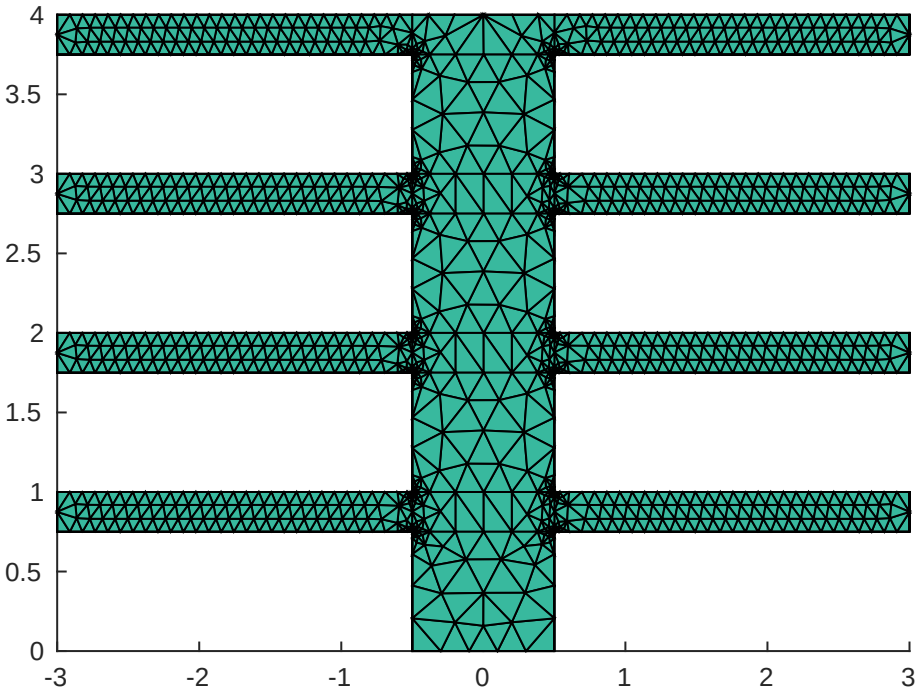


Figure 3: computational domain for heat sink structure

$$M\frac{d\vec{T}}{dt} + A\vec{T} = \vec{F},$$

which can be solved by applying the Euler implicit method:

$$\vec{T}_{i+1} = \vec{T}_i + hM^{-1}[-A\vec{T}_{i+1} + \vec{F}],$$

where the subscript i indicates the time index and $h = t_{i+1} - t_i$. Rearranging terms simplifies the expression to

$$(\frac{M}{h} + A)\vec{T}_{i+1} = \frac{M}{h}\vec{T}_i + F.$$

Solving this system amounts to solving a system of 1333 equations for each time step. We can reduce this system by projecting on to a reduced basis $V \in \mathbb{R}^{1333\times k}$:

$$V^T(\frac{M}{h} + A)V\vec{T}^r_{i+1} = V^T(\frac{M}{h}\vec{T}_i + F)$$

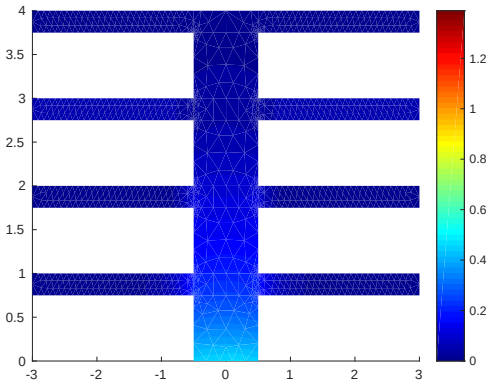
$$\text{where} \quad \vec{T}_{i+1} \approx V\vec{T}^r_{i+1},$$

which amounts to solving a system of k equations for each time step.

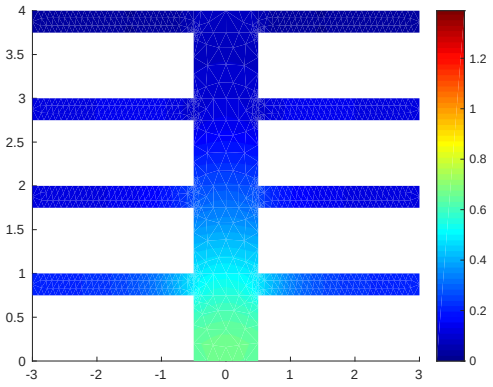
The thermal conductivities μ^i for the four sets of fins were taken from the set $\mathcal{D} = [0.1, 10]$ and the Biot number was held fixed at 0.1. The parameter set $\mathcal{D}$ was discretized into five uniformly spaced points, resulting in 5^4 points in the discrete parameter space.

The model was solved at every point in the parameter space for 101 time steps uniformly distributed from $t = 0$ to $t = 10$. Each solution vector was compiled into a column of a snapshot matrix X. The resulting dimension of X was 1333×63125. A reduced basis was created from this snapshot matrix using the method of snapshots described in Section 3.

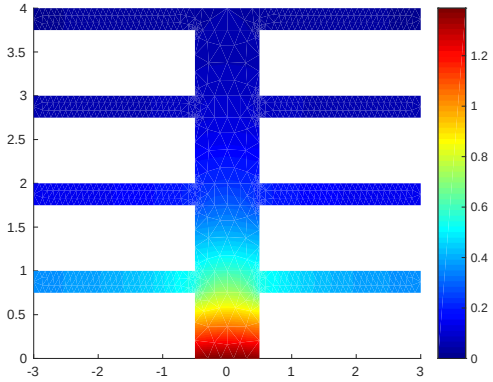
The full model was solved for a particular input from $\mathcal{D}$ which was not one of the 5^4 points used to construct X. The reduced model was solved with the same input set for several values of k and compared against the solution from the full model. Figure 4 shows these solutions for $k = 2, 3, 10$ and the full model solution. The approximate solutions converged to the full model solution as k increased.



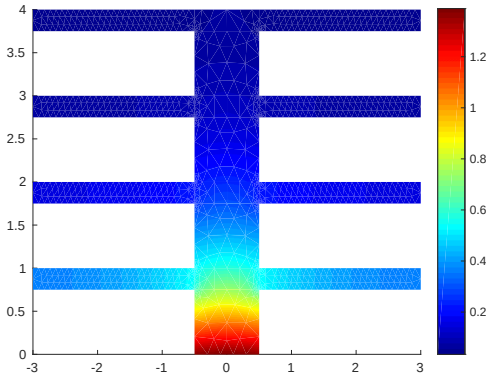
(a) k = 2



(b) k = 3



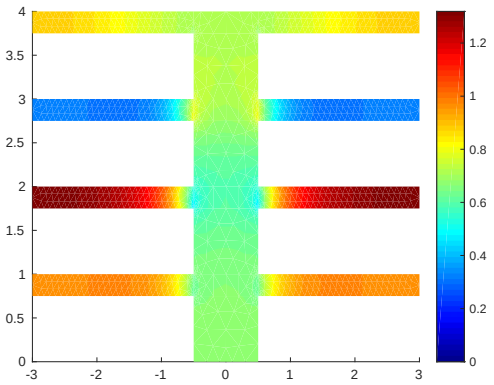
(c) k = 10



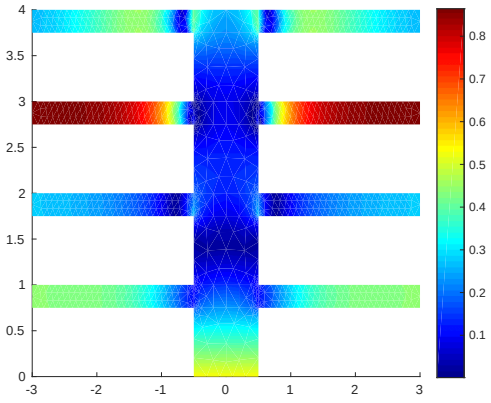
(d) exact

Figure 4: Solutions from the reduced FEM model for various k (a-c) and the solution from the full FEM model (d). These solutions are taken at $t = 10$.

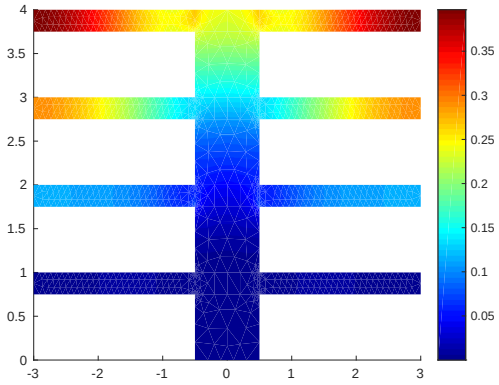
The relative error between the reduced model solutions and full model solution was computed for each node and plotted in Figure 5. Overall the relative error decreased when increasing k. For $k = 25$ the maximum point-wise relative error is 0.8%.



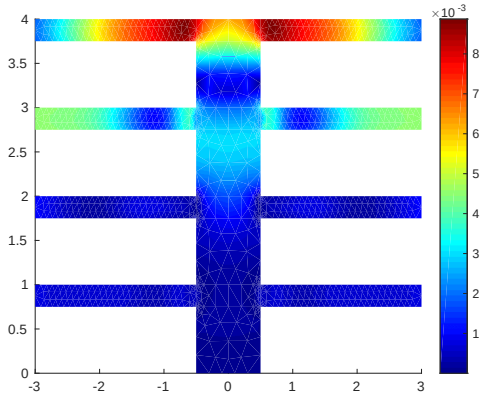
(a) k = 2



(b) k = 3



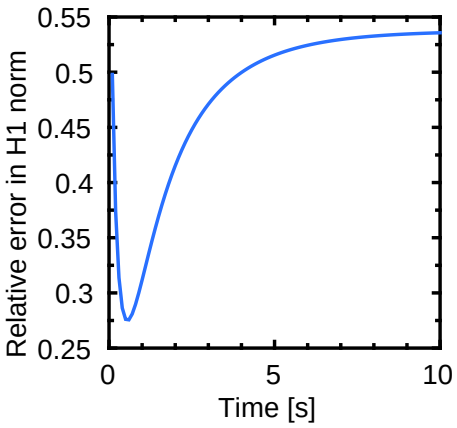
(c) k = 10



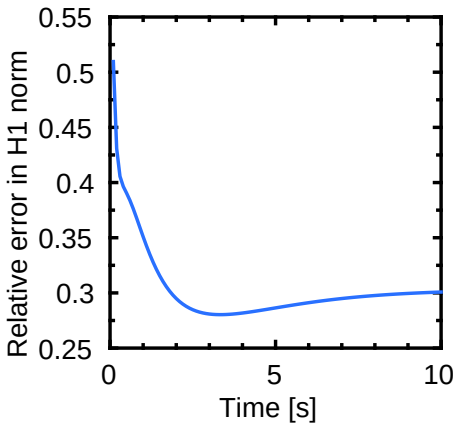
(d) k = 25

Figure 5: Maps of the relative error between the reduced and exact FEM solutions at $t = 10$.

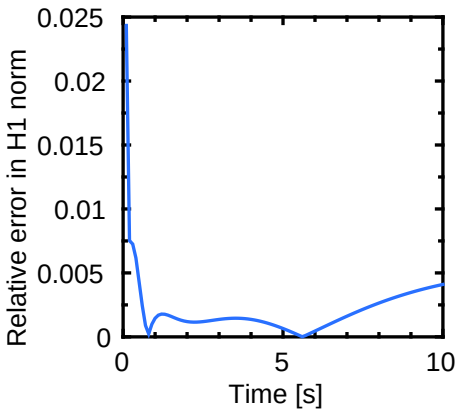
The H^1 norm given by $\|T\|_{H^1} = \sqrt{T^T M T + T^T A T}$ was computed at each time step for the reduced models and the exact model. Figure 6 shows the relative error between H^1 norms for the reduced models and the exact model at each time step. For each choice of k the error decreases rapidly within the first second and then grows slowly with time. The time averaged error decreases as k increases.



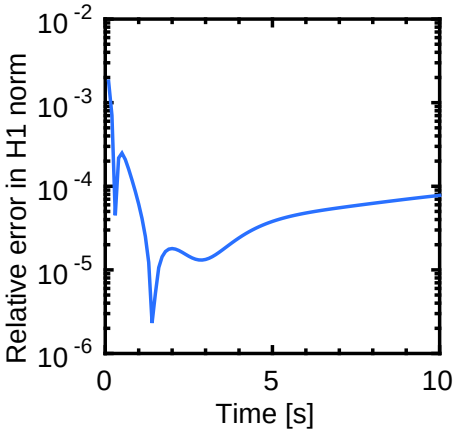
(a) k = 2



(b) k = 3



(c) k = 10



(d) k = 25

Figure 6: Convergence over time of the H^1 norm.

4.2 Image Compression

The POD method and randomized projection method can also be used for image compression. Consider a gray scale image denoted as X and represented by an $m \times n$ element array of doubles. A potential method for compressing this image is to compute the SVD of X and truncate U, S, and V to the first k columns. The cost of storing this decomposition would then be $k(m+n)+k^2$. This procedure was performed on a 413×508 image shown in Figure 8(a) and the resulting compressed image is shown in 8(b) for $k = 50$. We later refer to this compression method as Method 1. The value $k = 50$ was chosen by plotting the singular values of X, shown in Figure 7 and determining where the decay became linear.

We can extend the POD principle to image compression by forming a reduced basis $V \in \mathbb{R}^{m \times k}$ using the first k left-singular vectors of X and then forming a reduced X by

$$X_r = V^T X, \tag{5}$$

where $X_r \in \mathbb{R}^{k \times n}$. Storing X_r and V would then cost $k(n+m)$. This compression method was applied to our sample image and shown in Figure 8(c). We later refer to this compression method as Method 2.

We can reduce the computation required to form a reduced basis of X by first projecting X on to a random subspace spanned by k random vectors as we have done before:

$$\tilde{X} = \frac{1}{\sqrt{k}} X\varepsilon,$$

where $\tilde{X} \in \mathbb{R}^{m \times k}$. From here the basis vectors can be computed using the method of snapshots described in Theorem 2. By applying the projection method in Equation 5, we can form X_r and V that can be stored with a cost of $k(n+m)$. This procedure was applied to our image using an Achlioptas Distribution [6] and the resulting compressed image is shown in Figure 8(d). We later refer to this compression method as Method 3.

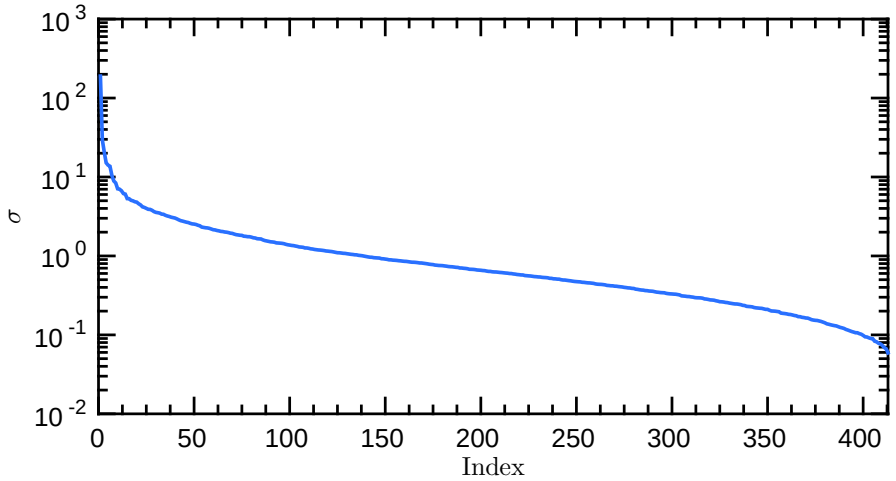
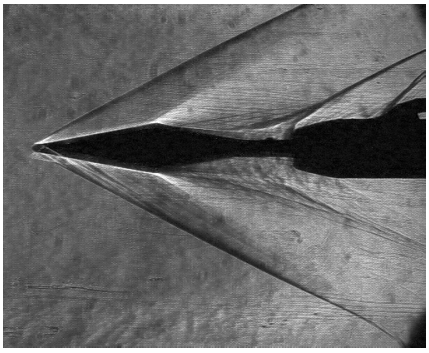
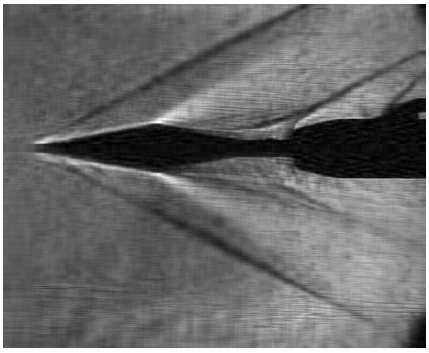


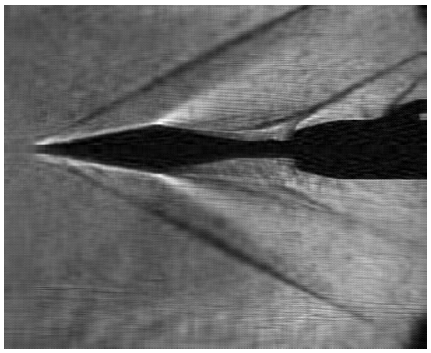
Figure 7: decay of singular values



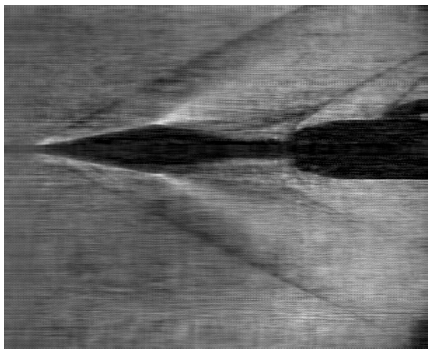
(a) Original Image



(b) Method 1: Truncated SVD with $k = 50$.



(c) Method 2: Truncated SVD with reduced basis approximation and $k = 50$.



(d) Method 3: Randomized SVD with reduced basis approximation and $k = 50$.

Figure 8: Image compression techniques applied to a Schlieren image showing the shockwave structure formed around a wedge in a supersonic windtunnel.

By comparing the compressed images in Figure 8 it is obvious that the randomized method in (d) reduces the quality of the compression. The benefit to this randomized projection method was that it reduced the computational cost required to form a reduced basis. The original image required storing $209,804$ elements. Method 1 required storing $48,550$ elements (a 77% reduction), and Methods 2 and 3 required storing $46,050$ elements (a 78% reduction).

5 Conclusion

The randomized proper orthogonal decomposition method can be used to reduce the cost of solving large linear systems. This method requires computing a reduced basis which

can be accomplished either directly via the singular value decomposition, or approximately via a random projection and reduced singular value decomposition.

In this thesis we have outlined some of the major contributions to POD in the last 30 years and presented some original results. Our contributions included:

- a new way of showing that randomized projections can produce a reduced basis which links Johnson-Lindenstrauss embeddings and statistical methods,
- numerical analysis of the convergence rate associated with randomized projection methods,
- an application of the POD method to a heat transfer FEM model,
- error analysis of the heat transfer solution predicted via the POD method,
- an application of the POD method to image compression.

By applying the POD method to a heat transfer FEM model, we show that the system could be reduced from 1333 equations and variables to just 25 equations and variables while only introducing a maximum point-wise relative error of 0.8%. By applying the POD method to image compression, we showed that for a particular image we can store 78% less data while only incurring a modest reduction in image quality.

6 Future Work

Our planned future work in this topic includes:

- applying the POD method to a three-dimensional heat transfer model
- applying the POD method to large scale problems for aerospace applications
- tightening the error bounds on random projection methods using the statistical approach described in Section 3

References

[1] D. Ferrucci, *IBM's Watson/DeepQA*, Proceedings of the 38th annual international symposium on Computer architecture, (2011).

[2] L. Sirovich, *Turbulence and the dynamics of coherent structures*, Quarterly of Applied Mathematics, vol. 45 , no. 3 (1987), pp. 561-590.

[3] N. Halko, P.G. Martinsson, J.A. Tropp, *Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions*, SIAM Rev., 53 (2011), pp. 217-288.

[4] T. Sarlós, *Improved Approximation Algorithms for large Matrices via Random Projections*, Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science, (2006), pp. 143-152.

[5] W. B. Johnson, J. Lindenstrauss, *Extensions of Lipschitz Mappings into a Hilbert Space*, in Contemporary Mathematics, Vol. 26, R. Beals, A. Beck, A. Bellow, A. Hajian, ed., American Mathematical Society, New Haven, CT, 1984, pp. 189-206.

[6] D. Achlioptas,*Database-friendly random projections: Johnson-Lindenstrauss with binary coins*, Journal of Computer and System Sciences, 66 (2003), pp. 671-687.

A Matlab Code

```
%tROM_driver.m
%Author: Brad Marvin
%creates reduced order model of FEM heat model and plots error
clear all
 addpath('./plotpub');
%n: number of training points along single dimension in parameter space
n = 5;

%Time step parameters
h = 0.1;
tmin = 0;
tmax = 10;
t = tmin:h:tmax;

%k: rank of reduced basis
k = 25;

load grids

%form input matrix from sections of parameter space
mu = input_gen(n);
%form snapshot matrix from FEM model
[Aq,Fh,M,Ah0,X] = tsnapshot(coarse,mu,t,h);

%form reduced basis from snapshot matrix using method of snapshots
[V,w,Xnew] = reduced_basis(X,k);

q = length(Fh);

%initial condition
uh0 = zeros(q,1);

%FEM solution using reduced basis
mul=[6,7,8,9,1,0.1];
Ah = Ah0;
for i = 1:6
    Ah = Ah+mul(i)*Aq{i};
end
un = zeros(q,length(t));
un(:,1) = uh0;
xr = zeros(k,1);
kr = V'*(M/h+Ah)*V;
[L,U,p] = lu(kr);
for ii = 1:length(t) - 1
    Br = V'*(M/h*un(:,ii)+Fh);
    y = L\(p*Br);
    xr = U\y;
    un(:,ii+1) = V*xr;
end

%solve exact FEM model for comparison
uh = zeros(q,length(t));
uh(:,1) = uh0;
k = (M/h+Ah);
```

```
[L,U,p] = lu(k);
for ii = 1:length(t) - 1
    B = M/h*uh(:,ii)+Fh;
    y = L\(p*B);
    uh(:,ii+1) = U\y;
end

% plot H1 error over time
err = zeros(length(t),1);
for ii = 1:length(t)
    Hp1 = sqrt(un(:,ii)'*M*un(:,ii)+un(:,ii)'*Ah*un(:,ii));
    Hp2 = sqrt(uh(:,ii)'*M*uh(:,ii)+uh(:,ii)'*Ah*uh(:,ii));
    err(ii,1) = abs(Hp1-Hp2)./Hp2;
end
plot(t,err)
opt.XLabel = 'Time [s]';
opt.YLabel = 'Relative error in H1 norm';
opt.LineWidth = 1.5;
opt.YScale = 'log';
opt.FontSize = 12;
opt.FileName = 'h1error_n25.eps';
opt.BoxDim = [2, 2];
setPlotProp(opt);
% plot relative error at last timestep
err_map = abs(un(:,end)-uh(:,end))./abs(uh(:,end));
plotsolution(coarse,err_map,0);
% plot solution at last timestep
plotsolution(coarse,un(:,end),1);
```

```
% tsnapshot.m
% Author: Brad Marvin
% runs transient FEM heat model for each point in parameter space
function[Aq,Fh,M,Ah0,X]=tsnapshot(grid,mu,t,h)
% build FEM matrices
[Aq,Fh,M] = FEM(grid);
q = length(Fh);
Ah0 = sparse(grid.nodes,grid.nodes);
X=[];
[n,~] = size(mu);
% solve system for each set of inputs and build snapshot
parfor j = 1:n
    muj = mu(j,:);
    Ah = Ah0;
    for i = 1:6
        Ah = Ah+muj(i)*Aq{i};
    end
    uh0 = zeros(q,1);
    uh = zeros(q,length(t));
    uh(:,1) = uh0;
    k = (M/h+Ah);
    [L,U,p] = lu(k);
    for ii = 1:length(t) - 1
        B = M/h*uh(:,ii)+Fh;
        y = L\(p*B);
        uh(:,ii+1) = U\y;
    end
    X=[X uh];
```

```
end
end
```

```
% reduced_basis.m
% Author: Brad Marvin
% forms a reduced basis of rank k for the column space of an input snapshot
    matrix
function [V,w,Xnew]=reduced_basis(X,k)
[~,n] = size(X);
%Xnew=X/sqrt(k)*randraw('rademacher',[],[n,k]);
%Xnew=X/sqrt(k)*randn(n,k);
Xnew=X/sqrt(k)*liRandomMatrix(n,k,3);
[~,~,w]=svd(Xnew);
V=Xnew*w;
end
```

```
%FEM.m
%Author: Unknown

function [Aq,Fh,M] = FEM(grid)
% This function will take the configuration and a triangulation and return
   Aq and Fh
% by using finite element method

% Global node
Node = zeros(3,1);
% Global coordinates
x = zeros(3,1);
y = zeros(3,1);
% Elemental matrices I
AI = zeros(3, 3);
% Right hand side vector F
Fh = zeros(grid.nodes,1);
% Assembly Matrix Aq from AI
M = sparse(grid.nodes,grid.nodes);
Am = [2 1 1;1 2 1; 1 1 2];
nd = length(grid.theta);
for n = 1:(nd-2)
    Aq{n} = sparse(grid.nodes,grid.nodes);
    % Number of elements on each region
    m = size(grid.theta{n},1);
    for k = 1:m
        % Take global node for current element
        Node(1) = grid.theta{n}(k, 1);
        Node(2) = grid.theta{n}(k, 2);
        Node(3) = grid.theta{n}(k, 3);
        % x-y coordinates for each global node
        for i = 1:3
            x(i) = grid.coor(Node(i),1);
            y(i) = grid.coor(Node(i),2);
        end
        % The area of current element
        Area = 0.5*abs(x(2)*y(3)-y(2)*x(3)-x(1)*y(3)+y(1)*x(3)+x(1)*y(2)-y
   (1)*x(2));
        % Calculate cx and cy
```

```
        cx(1) = y(2) - y(3);
        cx(2) = y(3) - y(1);
        cx(3) = y(1) - y(2);
        cy(1) = x(3) - x(2);
        cy(2) = x(1) - x(3);
        cy(3) = x(2) - x(1);
        % establish the elemental matrices
        for i = 1:3
            for j = 1:3
                AI(i,j) = cx(i)*cx(j) + cy(i)*cy(j);
            end
        end
        AI = AI/(4*Area);

        % Element Mass matrix
        AM = (Area/12)*Am;

        % Assembly matrix A
        for alpha = 1:3
            i = Node(alpha);
            for beta = 1:3
                j = Node(beta);
                 Aq{n}(i,j) = Aq{n}(i,j) + AI(alpha, beta);
                 M(i,j) = M(i,j) + AM(alpha,beta);
            end
        end
    end
end

% Elemental matrices II
AII = zeros(2, 2);
Node = zeros(2,1);
x = zeros(2,1);
y = zeros(2,1);
% Assembly Matrix A from elemental matrices AII and RHS
Aq{nd-1} = sparse(grid.nodes,grid.nodes);
for m=(nd-1):nd
    % The number of segments on the boundary excluding the Root
    n = size(grid.theta{m},1);
    for k = 1:n
        % Global node of current segment
        Node(1) = grid.theta{m}(k, 1);
        Node(2) = grid.theta{m}(k, 2);
        % x-y coordinates of global nodes
        for i = 1:2
            x(i) = grid.coor(Node(i),1);
            y(i) = grid.coor(Node(i),2);
        end
        % Length of current segment
        h = sqrt((x(1) - x(2))^2 + (y(1) - y(2))^2);
        if m==(nd-1)
            % Establish the elemental matrices II
            AII =(h/3)*[1 1/2; 1/2 1];
            % Assembly matrix A
            for alpha = 1:2
                i = Node(alpha);
                for beta = 1:2
                    j = Node(beta);
```

```
                    Aq{m} (i,j) = Aq{m}(i,j) + AII(alpha, beta);
                end
            end
        else
            % Establish the elemental matrices for RHS
            Fe = h/2*[1 1];
            % Assembly the RHS
            for alpha = 1:2
                i = Node(alpha);
                Fh(i) = Fh(i) + Fe(alpha);
            end
        end
    end
end
```

```
% input_gen.m
% Author: Brad Marvin
% generates parameter space from input number of training points
function [mu]=input_gen(n)
    x=linspace(0.1,10,n);
    mu=nchoosek([x,x,x,x],4);
    mu=unique(mu,'rows');
    mu=[mu,ones(n^4,1),0.1*ones(n^4,1)];
end
```

```
%liRandomMatrix.m
%Author: Aditya Krishna Menon
function [R] = liRandomMatrix(d,k,s)
R_temp = rand(d,k);
R = zeros(d,k);
R(find(R_temp>1-1/(2*s))) = sqrt(s);
R(find(R_temp<1/(2*s))) = -sqrt(s);
end
```

```
% cost_convergence.m
% Author: Brad Marvin
addpath('./plotpub')
% form input parameter set
mu = input_gen(5);
load grids
% build snapshot matrix
t = 0:0.1:5;
[~,~,~,~,X]=tsnapshot(coarse,mu,t,0.1);
% set k values such that k^(-1/2) is linearly distributed
kmin = 100;
kmax = 2000;
nk = 200;
q = linspace(1/sqrt(kmin),1/sqrt(kmax),nk);
k = 1./q.^2;
k = round(k);
[~,n] = size(X);
% solve cost function for each k
Jp = zeros(1,length(k));
parfor ii = 1:length(k)
```

```
    a = 75;
    for jj = 1:a
  % Random projection using Achlioptas Dist.
        Xp=X/sqrt(k(ii))*liRandomMatrix(n,k(ii),3);
        [~,~,w]=svds(Xp,1,'L');
        V=Xp*w;
        Jp(ii) = Jp(ii) + norm(Xp'*V,'fro');
   end
   Jp(ii) = Jp(ii)/a;
   k(ii)
end
% solve cost function for exact inputs
[~,~,w] = svds(X,1,'L');
V=X*w;
J=norm(X'*V,'fro');
% plot cost function convergence
set(0,'defaulttextinterpreter','latex')
f=abs(J-Jp)/abs(J);
x=1./sqrt(k);
p = polyfit(x,f,1);
y = polyval(p,x);
figure; hold on
plot(x,f)
plot(x,y)
hold off
opt.YLabel = '$\frac{|J-Jp|}{|J|}$';
opt.XLabel = '$\frac{1}{\sqrt{k}}$';
opt.LineWidth = 1.5;
opt.LineStyle = {'-','--'};
opt.FontSize = 14;
opt.Colors = [
      0,0,1;
      0,0,0
      ];
opt.FileName = 'test.eps';
setPlotProp(opt);
```

```
%Jmin.m
%Author: Brad Marvin
%Evaluates cost function along gradient
addpath('./plotpub')
% build snapshot matrix
mu = input_gen(5);
load grids
t = 0:0.1:5;
[~,~,~,~,X]=tsnapshot(coarse,mu,t,0.1);
% set POD and cost function parameters
N = [1,10,50,100];
K = -1E-5:1.0E-8:1E-5;
a = 10^6; % scaling factor
[~,p] = size(X);
% solve cost function for exact inputs
[~,~,w] = svds(X,1,'L');
 XTX = (X'*X);
 gradJ = -XTX*XTX*w;
 J = zeros(1,length(K));
 for jj = 1:length(K)
```

```
    Kj = K(jj);
    J(1,jj) = -1/(2*a^2)*norm(XTX*(a*w+Kj*gradJ),'fro');
 end
 clear XTX
% solve vost function for randomized inputs
Jp = zeros(length(N),length(K));
for ii = 1:length(N)
   Ni = N(ii);
   %Xp = X/sqrt(Ni)*liRandomMatrix(p,Ni,3);
   %Xp = X/sqrt(Ni)*randraw('rademacher',[],[p,Ni]);
   Xp=X/sqrt(Ni)*randn(p,Ni);
   XpTXp = (Xp'*Xp);
   [~,~,wp] = svds(Xp,1,'L');
   gradJp = -XpTXp*XpTXp*wp;
   for jj = 1:length(K)
       Kj = K(jj);
       Jp(ii,jj) = -1/(2*a^2)*norm(XpTXp*(a*wp+Kj*gradJp),'fro');
   end
end
clear XpTXp
% plot cost function vs kappa
figure; hold on
for ii = 1:length(N)
    plot(K,abs(Jp(ii,:)))
end
plot(K,abs(J))
hold off
set(0,'defaulttextinterpreter','latex')
opt.YLabel = '$|$cost J$|$';
opt.XLabel = '$\kappa$';
opt.YScale = 'log';
opt.YLim = [1E-4, 1E4];
opt.XLim = [-1E-5, 1E-5];
opt.YGrid = 'off';
opt.LineWidth = 1.5;
opt.FontSize = 14;
opt.Legend = {'k = 1','k = 10','k = 50', 'k = 100','Exact'};
opt.LegendBox = 'on';
opt.LegendLoc = 'SouthWest';
opt.FileName = 'kmin_normal.eps';
setPlotProp(opt);
```