ABSTRACT

Title of dissertation:	ITERATIVE SOLUTION METHODS FOR REDUCED-ORDER MODELS OF PARAMETERIZED PARTIAL DIFFERENTIAL EQUATIONS
	Virginia Forstall, Doctor of Philosophy, 2015
Dissertation directed by:	Professor Howard Elman Department of Computer Science

This dissertation considers efficient computational algorithms for solving parameterized discrete partial differential equations (PDEs) using techniques of *reduced-order modeling*. Parameterized equations of this type arise in numerous mathematical models. In some settings, e.g. sensitivity analysis, design optimization, and uncertainty quantification, it is necessary to compute discrete solutions of the PDEs at many parameter values. Accuracy considerations often lead to algebraic systems with many unknowns whose solution via traditional methods can be expensive. Reduced-order models use a reduced space to approximate the parameterized PDE, where the reduced space is of a significantly smaller dimension than that of the discrete PDE. Solving an approximation of the problem on the reduced space leads to reduction in cost, often with little loss of accuracy.

In the reduced basis method, an offline step finds an approximation of the solution space and an online step utilizes this approximation to solve a smaller reduced problem, which provides an accurate estimate of the solution. Traditionally, the

reduced problem is solved using direct methods. However, the size of the reduced system needed to produce solutions of a given accuracy depends on the characteristics of the problem, and it may happen that the size is significantly smaller than that of the original discrete problem but large enough to make direct solution costly. In this scenario, it is more effective to use iterative methods to solve the reduced problem. To demonstrate this we construct preconditioners for the reduced-order models or construct well-conditioned reduced-order models. We demonstrate that by using iterative methods, reduced-order models of larger dimension can be effective.

There are several reasons that iterative methods are well suited to reduced-order modeling. In particular, we take advantage of the similarity of the realizations of parameterized systems, either by reusing preconditioners or by recycling Krylov vectors. These two approaches are shown to be effective when the underlying PDE is linear. For nonlinear problems, we utilize the discrete empirical interpolation method (DEIM) to cheaply evaluate the nonlinear components of the reduced model. The method identifies points in the PDE discretization necessary for representing the nonlinear component of the reduced model accurately. This approach incurs online computational costs that are independent of the spatial dimension of the discretized PDE. When this method is used to assemble the reduced model cheaply, iterative methods are shown to further improve efficiency in the online step.

Finally, when the traditional offline/online approach is ineffective for a given problem, reduced-order models can be used to accelerate the solution of the full model. We follow the solution model of Krylov subspace recycling methods for sequences of linear systems where the coefficient matrices vary. A Krylov subspace recycling method contains a reduced-order model and an iterative method that searches the space orthogonal to the reduced space. We once again use iterative solution techniques for the solution of the reduced models that arise in this context. In this case, the iterative methods converge quickly when the reduced basis is constructed to be naturally well conditioned.

ITERATIVE SOLUTION METHODS FOR REDUCED-ORDER MODELS OF PARAMETERIZED PARTIAL DIFFERENTIAL EQUATIONS

by

Virginia Forstall

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2015

Acknowledgments

Most of all, I would like to thank my advisor, Howard Elman, for his guidance, feedback, and support over the course of my graduate education. His endless patience, encouragement, and editorial skills are much appreciated. Many thanks, as well, to my collaborators at Sandia National Laboratories, Kevin Carlberg, Ray Tuminaro, and Paul Tsuji. I am grateful for the internship experience they provided as well as their input on the collaborative work that is included in this thesis. I am especially thankful for Kevin's mentorship, advice, and enduring enthusiasm. Thanks, as well, to Qifeng Liao for his reduced basis collocation code. In addition, I would like to thank my committee, Radu Balan, Kayo Ide, Ricardo Nochetto, and Konstantina Trivisa, as well as Dianne O'Leary, for their support of this work and throughout my education at the University of Maryland. Finally, I am grateful for my fellow AMSC students and their friendship and moral support during the last few years.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

PDE	Partial differential equation
FOM	Full-order model
ROM	Reduced-order model
LSC	Least-squares commutator
DEIM	Discrete empirical interpolation method
SVD	Singular value decomposition
CG	Conjugate gradient
GMRES	Generalized minimum residual method
FOM	Full orthogonalization method
POD	Proper orthogonal decomposition
AMG	Algebraic multigrid
GMRES-DR	Generalized minimum residual method with deflated restarting
GCRO-DR	Generalized conjugate residuals (orthogonalized) with deflated restarting
GCROT	Generalized conjugate residuals with optimal truncation

Chapter 1

Introduction

Parameterized partial differential equations (PDEs) are useful for modeling physical systems where coefficients, boundary conditions, or initial conditions depend on input parameters. In settings of this type, users may require the computation of discrete solutions of the PDE for many values of the input parameter set, for example, to perform parameter estimation, sensitivity analysis, design optimization, or statistical analysis of random processes. When an accurate spatial discretization is needed, this can be a prohibitively expensive task. One approach for addressing this difficulty is to use reduced-order models. The parameterized problem is approximated on a reduced space of smaller dimension than that of the discrete PDE. This thesis considers reduced-order modeling for efficiently solving such parameterized PDEs with specific focus on incorporating the techniques of iterative linear solvers to improve efficiency.

Instances where reduced-order modeling can be used are the many-query context and real-time applications. In many-query applications, the PDE must be solved at many different parameter values so the cost of generating a reduced-order model is amortized by many cheaper solutions of the reduced model. Examples of the many-query approach are abundant in uncertainty quantification. For example, the expectation and variance of a PDE with random parameters can be computed using Monte-Carlo or stochastic collocation methods, both of which fit the many-query model. Real-time applications, for example estimation and control, require the computation of the solution at a set of parameter values to be done as quickly as possible. Therefore, the cost of generating the reduced-order model can be very high as long as the solutions can be obtained rapidly in the real-time application.

Early work in reduced-order modeling models differential systems in specific domains [1, 50] and more general finite-dimensional systems including ODEs [57]. It can be also used for models in fluid dynamics [56]. Current work focuses on many aspects of reduced-order modeling including improving efficiency and extending these methods to nonlinear and time-dependent problems [11].

In this chapter, we begin with an overview of the techniques of reduced-order modeling including methods for computing the reduced basis. Next, we discuss reduced-order modeling specifically for linear operators with affine dependence on the parameters. For this problem, we describe the offline-online paradigm as well as the computational difficulties that arise when this method is extended to the nonaffine and nonlinear cases. We review methods to address these difficulties—the so-called hyper-reduction methods. We then discuss a few of the drawbacks to the offline-online paradigm and present alternatives to this approach, specifically Krylov subspace recycling methods. We describe some Krylov subspace recycling methods. We conclude with a discussion of the goals and outline of the thesis.

1.1 Reduced-order modeling

Reduced-order modeling can be used when the manifold of solutions for the parameterized PDE can be accurately represented by a low-dimensional vector space [11]. When such a representation exists, reduced-order modeling finds this lower-dimensional subspace, known as the *reduced space*, and projects the original problem onto this space. The projected problem, known as the *reduced model*, is of a smaller dimension and thus, can be solved more efficiently. The solution of the reduced model produces an approximation to the solution with minimal loss of accuracy.

One method for reduced-order modeling is the *reduced basis method* [51]. Let us describe the method using a parameterized elliptic PDE

$$L(\vec{x}, \xi; u) = f(\vec{x}) \tag{1.1}$$

defined on a spatial domain D and subject to boundary conditions on ∂D

$$B(\vec{x}, \xi; u) = g(\vec{x}) \ \ , \tag{1.2}$$

where $\xi = [\xi_1, \xi_2, \ldots, \xi_m]^T$ is a vector of input parameters. Let Γ represent the space of possible parameter values of ξ. Consider a discretization of the PDE of order N such that $A(u;\xi)u(\xi) = f$. This is referred to as the *full model*. Reduced basis methods compute a small number of solutions, $u(\xi_1)$, $\ldots u(\xi_k)$, known as snapshots, and then for other parameters, $\xi \neq \xi_j$, find an approximation to $u(\xi)$ in the space spanned by $\{u(\xi_j)\}_{j=1}^{k}$. In the traditional approach, the computations are divided into *offline* and *online* steps. The (possibly expensive) offline step computes the snapshots and builds a basis of the low-dimensional vector space spanned by them.

The online step, which is intended to be inexpensive (because k is small), computes a projected version of the original problem (using, for example, a Galerkin projection) in the k-dimensional space. The projected problem, known as the *reduced model*, has a solution $\tilde{u}(\xi)$ which is an approximation of the solution $u(\xi)$.

There are several ingredients which define the reduced basis method including the projection method, an a posteriori error estimate (or error indicator), and the method used to construct the reduced basis. We briefly review some of the choices for these ingredients.

1.1.1 Projection methods

The reduced model is defined by the projection of the full model onto the reduced space. Define the *trial basis* Q of the reduced space such that the approximation to the solution is $\tilde{u} = Q\hat{u}$ where $\hat{u}$ is the solution of the reduced model. The reduced model is generated by the projection using a *test basis*. When the test basis is equivalent to the trial basis, the result is the Galerkin projection, so that the reduced model is

$$Q^T A Q \hat{u} = Q^T f \ . \tag{1.3}$$

The Galerkin projection is optimal for minimizing the error in the A norm for the case when A is symmetric positive definite [14]. When the test basis is different from the trial basis, the methodology entails a Petrov-Galerkin projection. For a linear operator, $A(u;\xi) = A(\xi)$, the choice of AQ for the test basis generates the reduced model

$$Q^T A^T A Q \hat{u} = Q^T A^T f \ , \tag{1.4}$$

where the solution $\tilde{u} = Q\hat{u}$, minimizes the state error in the $A^T A$ norm [14]. This problem is equivalent to solving the minimization problem,

$$\tilde{u} = \arg \min_{u \in \text{range}(Q)} ||R(u)||_2 \ , \tag{1.5}$$

where $R(u) = Au - f$. More generally for nonlinear problems, the solution to the least squares problem in equation (1.5) is equivalent to Petrov-Galerkin projection with a test basis $J_R(u)Q$ where $J_R(u) = \frac{\partial R(u)}{\partial u}$ is the Jacobian of the residual $R(u)$ [18].

1.1.2 Error estimate

An important component of successful reduced-order models is a cheap, accurate a posteriori error estimate of the reduced model. This error estimate determines if the solution computed using reduced model is accurate enough. It should be cheap to compute since it is part of the online computation. We will see in Section 1.1.3 that it also plays an important role in several methods for constructing the reduced basis. A common choice for the error estimator is the normalized residual,

$$\eta_\xi = \frac{||A\tilde{u}(\xi) - f||_2}{||f||_2} \ .$$

In the case of affine parameter dependence, this residual can be computed with computation cost independent of N [29]. Further discussion of this point is deferred to Section 1.2.

1.1.3 Offline construction of the reduced basis

In the offline-online paradigm, the primary task of the offline step is the construction of the reduced basis. This portion of the computation may be expensive. The following construction methods vary in offline cost, size of the reduced basis, and accuracy of the resulting reduced-order models.

The proper orthogonal decomposition (POD) derived from solutions obtained for a subset of the parameter space produces an orthogonal basis of an approximation of the space spanned by the snapshots [61]. The POD method takes a set of n_{trial} snapshots of the solution $S = [u(\xi^{(1)}), ..., u(\xi^{(n_{trial})})]$ and takes the singular value decomposition (SVD)

$$S = V \Sigma W^T ,$$

where $V = [v_1, ..., v_{n_{trial}}]$ and W are orthogonal and Σ is a diagonal matrix with the singular values sorted in order of decreasing magnitude. The reduced basis is defined as $Q = [v_1, .., v_k]$ with $k < n_{trial}$. This produces an orthogonal basis Q which contains the important components from the snapshot matrix S. The disadvantage of the POD is that the number of snapshots, n_{trial}, used to construct S is ad hoc. It is possible that the number of solutions of the full model required to find a basis with satisfactory accuracy could be quite large.

Alternatively, a reduced basis can be formed by finding an orthogonal basis for the span of the snapshots constructed using the modified Gram-Schmidt algorithm where the parameters at which the snapshots are taken are chosen carefully. In these methods, the number of full solutions required will be the same as the rank of the reduced basis. The process for choosing the random samples is known as *snapshot selection*, and methods include greedy sampling [11], variations on greedy sampling [27], error minimization methods [14], and sparse grids [29].

The greedy snapshot selection method [11, 72] depends on a subset of the parameter space of n_{trial} samples, denoted Γ_{trial}, and an a posteriori estimate of the reduced model. The basis is initialized with a single snapshot and then the reduced model is solved at all n_{trial} parameters. The sample which maximizes the error estimator is selected and the full model is solved at this parameter. The resulting snapshot is used to augment the basis. This process continues until all n_{trial} parameters have reduced solutions whose error estimate is below some threshold τ. There is a variation of this approach for problems with parameters that are nonuniform random variables, where weights are used to give preference to higher probability solutions during the greedy selection [20].

As described above, greedy methods are performed using a discretization of the parameter space Γ_{trial}. Ideally, the greedy method would choose the parameter which maximizes the error estimator from the continuous parameter space, Γ. If the snapshots were chosen this way, the greedy method would have favorable convergence properties with respect to the Kolmogorov n-width. The Kolmogorov n-width, $d_n(\mathcal{F})$, is defined as the error that would be obtained using the best n-dimensional space that can represent the function space $\mathcal{F}$ [7]. Formally, the Kolmogorov width

$$d_n(\mathcal{F}) = \inf_{\dim(Y)=n} \sup_{f \in \mathcal{F}} \operatorname{dist}(f, Y)$$

where $\operatorname{dist}(f, Y) = ||f - P_Y f||$ and P_Y is the orthogonal projector of a function onto

Y [9]. Let $\mathcal{Q}_n$ be the space spanned by an n-dimensional basis Q found using the greedy algorithm (over the continuous space). It has been shown [13] that

$$\text{dist}(\mathcal{F}, \mathcal{Q}_n) \leq Cn2^n d_n(\mathcal{F}) ,$$

where C is constant. So if the Kolmogorov n-width $d_n(\mathcal{F})$ decays at a rate faster than $(1/n)2^{-n}$, the greedy basis will be optimal. Other relationships between the greedy approximation error and the Kolmogorov width are discussed in [9] including the case where if $d_n(\mathcal{F}) \leq Mn^{-\alpha}$, then

$$\text{dist}(\mathcal{F}, \mathcal{Q}_n) \leq C_\alpha M n^{-\alpha}$$

where C_α depends only on α and improvements on these results are given in [25]. A similar extension for the weighted greedy algorithm is presented in [20].

Note that these results assume that greedy samples are taken over the entire continuous parameter space. With practical greedy algorithms, the performance is limited by how well the discretization or sample represents the parameter space. Bui-Thanh et. al [14] introduce a method to address this issue where the greedy search is performed over a continuous space. This search requires the solution of a PDE-constrained optimization problem. Thus, it is limited by the feasibility of solving the PDE-constrained optimization problem for the given reduced model and the resulting high offline costs.

Other methods have been considered to improve greedy sampling with the discrete approach. For example, Γ_{trial} used for greedy sampling can be constructed adaptively [37]. An extension of this method, the “hp reduced basis” method [27] uses a refinement procedure to construct separate bases for subdomains of the parameter space. First, the parameter space is divided into subdomains based on the errors generated via greedy sampling (the analogue of h-refinement) and then the usual greedy sampling procedure is used on each subdomain (the analogue of p-refinement). Note that this generates a separate basis for each subdomain, so one might expect that the dimension of each of the reduced bases will be smaller than if a single reduced basis was used for the entire parameter domain. This method has been shown to significantly reduce online costs (because each reduced problem is smaller) at an additional offline cost. This method is especially amenable to applications where the solutions vary greatly over the parameter domain. It will take more samples in regions where the solution is varying most, but the resulting larger reduced basis will be used only in that region.

The POD method and the greedy algorithm both rely on a method for sampling a subset of the parameter space effectively. There are several choices for sampling methods. First, uniform sampling is generally too expensive especially as the number of parameters increases. The second, random sampling has the disadvantage that it might fail to recognize regions of the input space. Sampling methods which balance these extremes include Latin hypercube sampling, central Voronoi tessellation (CVT), or sparse grids [14, 29].

The snapshot selection technique used to construct the reduced bases in Chapters 2 and 3 is random sampling. A snapshot is taken only if the reduced solution at the current sample fails some error criterion. The method is defined by a random sample of n_{trial} parameters, Γ_{trial}, and a threshold tolerance τ. The basis is initialized using a single snapshot. Then for each of the parameters, the reduced problem is solved. If the error indicator of the reduced solution is below the tolerance τ, the computation proceeds to the next parameter. Otherwise, the full model is solved and the snapshot is used to augment the reduced basis.

For this strategy (and other snapshot selection techniques), it is easy to enrich the basis at any point during the online computation. If a parameter encountered during the online computation fails to satisfy the tolerance, τ, the full model can be solved and the basis augmented. As long as this occurs infrequently, it will not be too costly. Depending on the application, this approach may be preferable. For example, the goal of many-query applications is the efficient solution at all parameters; therefore, spending less time offline and occasionally augmenting online will lead to overall lower costs. In real-time applications, however, a more careful offline construction may be required to ensure that the reduced model will always produce solutions to the required accuracy. Other methods can be used to gain more accuracy from a reduced model in an online context without a complete solve of the full model. For further discussion of these methods, see Section 1.4.

1.2 Reduced-order models of linear affine operators

In the case of linear operators with affine dependence on the parameters, the cost of the online step is independent of the dimension of the discrete PDE. The assumption of affine dependence allows a discrete operator $A(\xi)$ to be written as a sum of parameter-independent operators, $\{A_i\}$, i.e.

$$A(\xi) = \sum_{i=1}^{s} \varphi_i(\xi) A_i \ ,$$

where $\varphi_i : \mathbb{R}^m \mapsto \mathbb{R}$. Thus, the reduced operator $Q^T A(\xi) Q$ can be constructed as

$$Q^T A(\xi) Q = \sum_{i=1}^{s} \varphi_i(\xi) Q^T A_i Q \ . \tag{1.6}$$

If the members of $\{Q^T A_i Q\}$ are precomputed during the offline portion of the computation, the online cost of forming the matrix of equation (1.6) depends only on the number of parameters m and the dimension of the reduced basis k. The cost of solving the reduced problem depends only on k. As long as $k \ll N$, the cost of solving the reduced problem at each parameter will be significantly cheaper than the cost of solving the full problem. In addition, the norm of the residual $R(\tilde{u}) = AQ\hat{u} - f$ can be computed using matrices precomputed offline as well:

$$||AQ\hat{u} - f||_2^2 = \hat{u}^T \sum_{i=1}^{s} \sum_{j=1}^{s} \varphi_i(\xi)\varphi_j(\xi) Q^T A_i^T A_j Q \hat{u} - 2\hat{u}^T \sum_{i=1}^{s} \varphi_i(\xi) Q^T A_i^T f + f^T f$$

where $Q^T A_i^T A_j Q$ and $Q^T A_i^T f$ are precomputed [11, 29]. Note that for the spatial discretizations considered in this thesis A tends to be sparse and $AQ\hat{u} - f$ is relatively cheap to compute in the usual way. So for simplicity, this is the approach used for the residual computation.

1.3 Reduced order models of nonlinear operators

When this offline-online approach is applied to a nonlinear problem or a problem with nonaffine parameter dependence, the online step using the traditional reduced model is not independent of N. Given a discretized PDE with a nonlinear component $F(u;\xi)$, the full model is

$$G(u) = Au + F(u;\xi) - b = 0 \ . \tag{1.7}$$

The reduced model using the Galerkin projection is

$$G^r(\hat{u}) = Q^T A(\xi) Q\hat{u} + Q^T F(Q\hat{u};\xi) - Q^T b = 0 \ . \tag{1.8}$$

The projection of the nonlinear operator $Q^T F(Q\hat{u};\xi)$ is of dimension k, but since it depends on the solution, it must be assembled at each step of a nonlinear iteration. Using a nonlinear solution method, e.g. the Picard iteration, each nonlinear iteration requires the construction of the Jacobian matrix associated with $F(Q\hat{u}_i;\xi)$ and multiplication by Q^T, where both operations depend on N.

For some cases the operators can be approximated by a sum of solution-independent matrices. For a quadratic operator with affine dependence on the parameters, it is possible to write the reduced operator as a sum of parameter-independent, solution-independent matrices [29]. This is only possible for certain classes of problems and even if the problem can be written as a sum of these reduced matrices it requires storage of these (dense) matrices. A general nonlinear operator or an operator with nonaffine parameter dependence can be treated using so-called *hyper-reduction* techniques which decrease the online costs associated with assembling the nonlinear components.

1.3.1 Hyper-reduction

One example of hyper-reduction is the empirical interpolation method (EIM). This method determines interpolating continuous functions of the governing PDE. This method was originally developed to deal with nonaffine parameter dependence [6] and was extended to nonlinear elliptic and parabolic operators in [36]. The functions are chosen using a greedy procedure where values of the parameter are selected. Then the approximation is formed by interpolating the solution from the selected points. Like the greedy procedures for constructing the reduced basis, it is often more computationally convenient to perform the greedy construction using a discrete set of samples of the parameter space instead of over continuous solution spaces [7].

1.3.1.1 Discrete empirical interpolation method

This leads to the discrete variant of EIM, the discrete empirical interpolation method (DEIM) [19], which generates the approximation from snapshots of the nonlinear component. In addition, the DEIM treats the nonlinear component of the model separately from the linear components. Thus, it requires a basis that represents just the nonlinear component of the solution. Referred to as the *nonlinear basis*, this basis, V, is generated using the POD method, with snapshots $S = [F(u(\xi^{(1)})), ..., F(u(\xi^{(k)}))]$ where $u(\xi^{(i)})$ is the discrete solution. The DEIM selects a subset of spatial grid points from the discretization of the PDE (i.e. indices of F) using a greedy algorithm. Therefore, the DEIM approximation of the nonlinear operator is

$$\bar{F}(u;\xi) = V(P^T V)^{-1} P^T F(u;\xi)$$

where P is a $N \times n_{deim}$ matrix that selects n_{deim} interpolating points from the spatial grid. An error bound for this approximation is given in [19]

$$||F - \bar{F}||_2 \leq ||(P^T V)^{-1}||_2 ||(I - VV^T)F||_2 \; . \tag{1.9}$$

The second factor in this expression depends on how well V represents the solution space of the nonlinear components. This can be decreased by taking more snapshots for S. The growth of the first factor is limited by the greedy selection of indices [19].

This approximation produces the DEIM model, $G^{deim}(\hat{u}) = 0$, where

$$G^{deim}(\hat{u}) = Q^T A Q \hat{u} + Q^T \bar{F}(Q\hat{u};\xi) - Q^T b \; . \tag{1.10}$$

$$= Q^T A Q \hat{u} + Q^T V (P^T V)^{-1} P^T F(u;\xi) - Q^T b \; . \tag{1.11}$$

The matrix $Q^T V (P^T V)^{-1}$ can be computed offline since it is parameter and solution independent. The online computation requires the construction of $P^T F(u;\xi)$ which means that $F(u;\xi)$ is only needed at the interpolation points. This can be done cheaply if the components of $F(u;\xi)$ depend only on a few entries of u. This condition is typically satisfied for discretized PDEs. For further discussion of assembly in the finite element case, see [4]. The subset of elements that must be tracked during a DEIM computation is referred to as the sample mesh. The cost of the offline computation using DEIM scales with the number of elements in the sample mesh and not with the number of elements in the full mesh.

1.3.1.2 Gappy POD method

In the case of DEIM, the number of indices in the interpolation is equivalent to the number of columns of V. This ensures that $(P^TV)^{-1}$ is computable. A variation of this approach is gappy POD [3, 32] where the number of indices, n_g, can exceed the number of basis vectors. The approximation of the nonlinear component is $\hat{F} = V(P^TV)^{\dagger}P^TF(u;\xi)$ where $(P^TV)^{\dagger}$ is the Moore-Penrose pseudoinverse. Equivalently $\hat{F} = V\alpha$ where α is the solution of the least squares problem

$$\alpha = \arg\min_{\hat{\alpha}} ||P^TV\hat{\alpha} - P^TF||_2 \ .$$

The number of indices n_g can now be anything, but for improvement over DEIM it should be larger than the number of columns of V. When n_g is equivalent to the number of columns of V, this approach is equivalent to the DEIM method [3]. The algorithm used to select the indices for gappy POD follows the greedy approaches used in EIM and DEIM; it loops through the basis vectors and chooses the index which maximizes the error in the approximation made with the partial set of indices [18]. There is a similar error bound for gappy POD to that produced for DEIM in equation (1.9) [19]. Define R from the economical QR-factorization of P^TV, and the bound is [18]

$$||F - \hat{F}||_2 \leq ||R^{-1}||_2||(I - VV^T)F||_2 \ .$$

1.3.1.3 Computational costs of hyper-reduction methods

To illustrate the costs of solving equation (1.8) with and without hyper-reduction, consider the case of a nonlinear operator with affine dependence on the parameters. Let $J_F(u)$ denote the Jacobian of $F(u)$. Then

$$J_{G^r}(\hat{u}) = Q^T AQ + Q^T J_F(Q\hat{u})Q \ .$$

Therefore for a given nonlinear iteration for the reduced model in equation (1.8) we have

$$\begin{aligned}\hat{u}_{n+1} &= \hat{u}_n - J_{G^r}(\hat{u}_n)^{-1} G^r(\hat{u}_n)\\ \hat{u}_{n+1} &= \hat{u}_n - (Q^T AQ + Q^T J_F(Q\hat{u}_n)Q)^{-1} G^r(\hat{u}_n) \ .\end{aligned}$$

Thus the following linear system must be solved each iteration

$$(Q^T AQ + Q^T J_F(Q\tilde{u}_n)Q)\delta\hat{u} = -G^r(\hat{u}_n) \ .$$

The primary costs associated with solving the reduced model are the following.

1. Initial assembly of the matrix $Q^T AQ$, performed once; with the assumption of affine structure of the operator, it will have cost $O(mk^2)$.

2. Computation of $J_F(Q\hat{u}_n)$ performed every iteration. This scales with the size of the discretization, N.

3. Assembly of $Q^T J_F(Q\hat{u}_n)Q$ performed every iteration at cost $O(Nk^2)$.

4. Solution of a dense linear system with $k \times k$ matrix $Q^T AQ + Q^T J_F(Q\hat{u}_n)Q$. This costs $O(k^3)$ when using direct methods.

Hyper-reduction methods are meant to decrease the online costs associated with computation of $J_F(Q\hat{u}_n)$ in point 2 and the assembly of the reduced matrix $Q^T J_F(Q\hat{u}_n)Q$ in point 3. For the DEIM and gappy POD methods, the cost of assembling $P^T J_{\bar{F}}(Q\hat{u}_n)$ and $P^T J_{\hat{F}}(Q\hat{u}_n)$ scale with n_{deim} and n_g respectively. Similarly the assembly costs are $O(n_{deim}k^2)$ and $O(n_g k^2)$ respectively. The other costs of the online step remain the same.

1.4 Beyond the offline-online approach

There are many applications where the offline-online approach is not practical. For example, in many-query applications it could be that the goal is to obtain the solution at all of the points as quickly as possible. If the offline cost of the method is too high, the cost savings in the online step might not be enough to amortize the offline cost. For example, greedy algorithms ensure that the dimension of the reduced basis is as small as possible and thus the online cost will be very small, but if a cheaper offline method can produce a reduced basis with only a slightly higher dimension, then it may be the preferred approach for a particular application.

In addition, there are situations where the parameter space cannot be sampled. For example, a parabolic PDE can be viewed as an elliptic PDE parameterized in time [11] and time is treated as the parameter. When this is the case, the problem at some parameter depends on the solution at a previous time step. In addition, the PDE may depend on both time and parameters. In these cases, it is not obvious what would be the best approach for generating snapshots. Some methods have been devised for this [15, 49], where snapshots are taken for a variety of parameters and time steps. However, the performance of the reduced model for time steps beyond where the snapshots were taken is unknown. As an alternative approach, one could treat this problem as a sequence of linear systems. Similarly, we could take this viewpoint for a nonlinear problem where the sequence of linear systems is generated by the nonlinear iteration. Finally, there are situations where the cost of the full model is too high for the full model to be solved at enough samples to produce an accurate basis. In these cases, reduced-order models can be used to accelerate the solution of the full model.

Each of these situations have led to efforts to consider a more blended approach to reduced-order modeling. For example, the reduced basis collocation method [29] obtains the reduced solution using a current reduced basis and if the error estimate does not satisfy the tolerance then the full model is solved and the new solution snapshot is added to the basis. At the end of the computation, the solution is known at each point on the collocation grid. Another method reuses the coarse grid and transfer operators obtained from using algebraic multigrid to so solve nearby linear systems where the linear systems come from a stochastic collocation problem [34,35].

Another approach avoids the solution of the full model when an online solution does not satisfy the tolerance. The adaptive h-refinement reduced-order model [15] splits the basis vectors into vectors with disjoint support, so the resulting reduced model is more accurate. A big advantage of this approach is that as the dimension of the refined reduced basis approaches the dimension of the full model, the error between the full and reduced solutions approaches zero. This is reassuring to be able to recover the full model solution if necessary. The example of a parameterized inviscid Burger's equation in [15] illustrates that a reduced basis constructed using snapshots of the solution taken at times before a shock is able to adapt online to accurately represent the solutions with the shock.

Another class of methods that use a blended approach is Krylov subspace recycling. This methodology can be viewed as using a reduced model to accelerate the convergence of the full model, where the full model is posed as a sequence of varying linear systems. Krylov subspace recycling has been used to accelerate the solution of sequences of linear systems applications such as fracture modeling and diffuse optical tomography [39] and to accelerate the solution of the linear systems that arise in stochastic collocation problems [33].

1.5 Krylov subspace recycling

Krylov subspace recycling methods are used to solve sequences of linear systems

$$A_j \bar{x}_j = b_j, \ j = 1, ..., n_s \ . \tag{1.12}$$

The goal is to find x_j such that

$$\frac{||b_j - A_j x_j||_2}{||b_j||_2} < \delta \ ,$$

as quickly as possible for all n_s systems in the sequence. The idea of recycling is to select a subspace of a generated Krylov space that will most aid in convergence of the Krylov subspace method for the next system in the sequence. This selected subspace is referred to as the *recycled* space. There are two components to a Krylov subspace recycling method. The first finds the solution on range of the recycled space. We can view the first component as the solution to a reduced model where the reduced model is defined by the projection of the linear system onto the recycled space. The second component finds the solution of the full problem to a given tolerance using a Krylov subspace method. The iteration begins with the reduced solution and then the ensuing full solve is accelerated by enforcing orthogonality to the recycle space.

Originally, Krylov recycling methods were developed in the case where A_j is fixed. For full orthogonalization methods like GMRES [65] the computational work per iteration grows with the iteration. Thus it becomes computationally and memory inefficient to keep the full basis. Restarted GMRES [65] simply throws away all of the information about the Krylov basis and begins the iteration again with the current solution as the initial guess for the solution in the next cycle. This procedure solves equation (4.1) where $A_j = A \ \forall j$ and b_j is updated every restart, since it depends on the initial solution.

This restarting process is known to decrease the rate of convergence [46]. This decrease occurs because after the restart the new Krylov basis vectors that are generated can be anywhere in the space. If, instead, a subset of the Krylov basis vectors is retained, new basis vectors can be chosen to be orthogonal to the old Krylov vectors so that the search space is smaller. So, given a set of recycled vectors Y, a Krylov subspace recycling method projects the problem onto this space (reduced model) and then uses a Krylov solver to generated new Krylov vectors that are orthogonal to Y. In methods that are based on the Arnoldi iteration, like GMRES, the basis vectors are orthogonalized using the 2-norm. In the conjugate gradient method, the basis vectors are orthogonalized with respect to the A-norm; this method is known as the augmented conjugate gradient method [66]. The reduced problem in this case is

$$Y^T A_j Y \hat{x} = Y^T b_j \ . \tag{1.13}$$

Once the approximation of the solution on the range of Y is obtained, the procedure requires search directions, p_k, to be A_j-orthogonal to Y such that $Y^T A_j p_k = 0$ for $k = 0, 1, .., n_j$.

In theory, the recycle space for the Krylov subspace recycling method can be any set of vectors. The methods are most effective, however, when the recycle space is formed using the Krylov basis vectors obtained in the previous cycle or cycles. Since the solution to the previous problem lies on the range of the Krylov vectors, for a given system, if the sequence of coefficient matrices and right-hand sides do not vary significantly, the solution to the next linear system in the sequence is probably close to the range of the Krylov vectors. In cases where the convergence of the Krylov method depends on the spectrum of A – as is the case for a conjugate gradient method or for certain classes of matrices using the GMRES method – the best vectors to recycle are the eigenvectors of A. Since it is often the small eigenvalues which hamper convergence, we would like to keep the eigenvectors associated with the smallest eigenvalues. The exact eigenvectors are unavailable, but the harmonic Ritz vectors provide approximations of the eigenvectors associated with the smallest eigenvalues. Using the harmonic Ritz vectors to compress a recycle space gives rise the method known as *deflation*. Deflation is a popular technique for Krylov subspace recycling and is used in GMRES-DR (GMRES with deflated restarting) [47] and the deflated conjugate gradient method [66].

Krylov subspace recycling can be generalized to sequences where the matrix varies. For example, the GCRO-DR method [55], a deflated restarting method for the GCRO Krylov solver uses deflated restarting within a single solve and after convergence of the solution for the jth system, it adapts the Krylov basis generated for A_j to a basis for A_{j+1}.

Another method adapted for varying left hand sides is a recycling method described in [62]. The recycling method keeps all the Krylov vectors $P = [p_0, ..., p_n]$ from the previous solve as the recycle basis for the augmented conjugate gradient algorithm. This solution method also introduces the idea of using an iterative method to solve the reduced problem that arises in the augmented CG method. The search directions are weighted to produce a well-conditioned reduced problem. Furthermore, [16] suggests that using both the direct and iterative methods provide the reduced solution quickly. The key to this approach is that direct methods are used with the most important recycled vectors and the fast-converging iterative method produces the solution on a larger space. This mixed approach is the topic of study of Chapter 4.

1.6 Using iterative methods for reduced models

Reduced-order modeling is only effective when online costs are cheap. Given a reduced model (made independent of the spatial dimension using interpolation techniques like DEIM, if necessary), the costs of obtaining solutions of the assembled reduced model depend on the size of the basis k and the solution method. There are two ways to keep costs low in solution of linear systems of the reduced-order model. The first is to construct the smallest possible basis. The second is to choose the most efficient solution method. A great deal of work in reduced-order modeling has been in developing methods for constructing the reduced bases (i.e. keeping

k small). Many of these methods were discussed in Section 1.1.3. Often the best approach will depend on the application at hand. In addition, the application will determine what restrictions (if any) to place on the offline costs. In this thesis, we take the perspective that the user has made a choice based on the problem, accuracy requirements, number of parameters etc. which has defined the dimension k. The question we address is for this fixed k, what is the most efficient solution technique?

The traditional approach to solving the systems reduced-order models is direct methods. For example for the linear affine reduced operator defined in equation (1.6), the solution of the reduced model can be obtained using direct methods with cost $O(k^3)$. In some cases, however, efficient techniques for the full model (like multigrid) exist and could have cost as low as $O(N)$. Therefore, it is possible that $k \ll N$, but $k^3 \not\ll N$. In this range of k, iterative methods with cost $O(pk^2)$ where p is the number of iterations for convergence, can be more efficient – increasing the range of k where reduced-order modeling is effective.

We consider iterative solution methods for reduced-order models of moderate size, and, in particular, we develop preconditioning strategies for the reduced problem. One of the key reasons that using iterative methods with an offline-online paradigm is effective is that the cost of constructing preconditioners can be relegated to the offline step. The relatedness of the linear systems is key for constructing a reduced-order model, but this property also means that good preconditioners for mean or other representative parameter(s) can be effective as preconditioners of the reduced model for nearby problems. Chapter 2 and 3 will discuss preconditioners and iterative methods for problems using the offline-online approach for the linear, affine case (Chapter 2) and the nonlinear case (Chapter 3). Chapter 4 moves aways from the offline-online approach by using iterative methods on the full model which are accelerated by the reduced model. In addition, we can use iterative methods for solving the reduced problems embedded in this approach. In that case the reduced order model and recycled directions are chosen in such away that the reduced problem is naturally well conditioned.

1.7 Outline of Thesis

In Chapter 2, we discuss using iterative methods to solve reduced models and develop preconditioners for the case with linear operators with affine dependence on the parameters. We will demonstrate the effectiveness of these preconditioners and iterative methods for two benchmark problems, the steady-state diffusion and convection-diffusion-reaction equations with random diffusion and reaction coefficients respectively.

In Chapter 3, we extend these ideas to nonlinear operators where the online costs are first made independent of spatial dimension using the discrete empirical interpolation method. Costs of the online computation are then further decreased using iterative methods and preconditioners. This extension is illustrated using the steady-state Navier-Stokes equations.

In Chapter 4, we move past the strictly offline-online approach and discuss a blended method, the POD-augmented Krylov subspace recycling method. This method implements a Krylov subspace recycling framework which compresses vectors using a weighted POD method. This approach is compared to deflation, a traditional approach to Krylov subspace recycling. In addition, the weighted POD method can be used in conjunction with a goal-oriented norm to ensure the fast convergence to an output of interest.

Chapter 5 provides the conclusion to the thesis.

Chapter 2

Preconditioners for reduced-order models of linear operators

2.1 Introduction

The reduced basis method reduces the cost of solving parameterized partial differential equations (PDEs) when the solution is needed at many parameter values. Computational costs are decreased by approximating the parameterized problem using a reduced space of significantly smaller dimension than that of the discrete PDE. Let the PDE

$$L(\vec{x}, \xi; u) = f(\vec{x}) \tag{2.1}$$

be defined on a spatial domain D and subject to boundary conditions on ∂D

$$B(\vec{x}, \xi; u) = g(\vec{x}) \quad , \tag{2.2}$$

where $\xi = [\xi_1, \xi_2, \ldots, \xi_m]^T$ is a vector of input parameters. Reduced basis methods compute a (relatively) small number of solutions, $u(\cdot, \xi^{(1)})$, $\ldots u(\cdot, \xi^{(k)})$, known as snapshots, and then for other parameters, $\xi \neq \xi^{(j)}$, attempt to find $u(\cdot, \xi)$ in the space spanned by $\{u(\cdot, \xi^{(j)})\}_{j=1}^{k}$.

One approach to reduced-order modeling is the offline-online paradigm. The computation is divided into *offline* and *online* steps. The offline step can be expensive and is performed only once. It constructs a basis, Q, of an approximation of the solution space. The online step uses the reduced basis to solve a reduced problem which provides an accurate estimate of the solution at each parameter $\xi = [\xi_1, ..., \xi_m]^T$. We expect the computational cost of the online step to be small since k is small. The underlying philosophy behind this approach is that the expense of the offline computation can be amortized to produce savings for many simulations (using many parameter values) in the online computations. It is also essential in cases where the online step must be performed in real time.

The dimension of the reduced space k is governed by characteristics of the problem, for example the number of parameters and the desired accuracy of reduced solutions. The conventional wisdom is that these systems can be solved cheaply using direct methods, at costs lower than what would be needed to solve the original discrete PDE. This is reasonable when k, the size of the reduced basis, is significantly smaller than N, the size of the discrete space. However, when efficient ($O(N)$) algorithms, such as multigrid, are available for the discrete PDE, it may happen that k is smaller than N by a large amount, but direct methods (of complexity $O(k^3)$) do not lead to reduced costs. In these cases, when k is of moderate size, we have shown that iterative methods can be used to solve the reduced problem more cheaply than direct methods.

The key component to the efficiency of the iterative methods is a preconditioner constructed as part of the offline computation. We use preconditioners that are parameter-dependent, but we have seen in our examples that the preconditioner works nearly as well when it comes from a single mean parameter as when it is constructed using the same parameter as the reduced problem we are solving. Using a single parameter is what enables the cost of constructing the preconditioner to be moved offline. With these preconditioners we will show that iterative methods are more efficient than direct methods when k is above a certain threshold.

An outline of this chapter is as follows. In Section 2.2, we review the reduced basis methodology for linear partial differential operators with affine dependence on parameters. In Section 2.3, we discuss iterative methods for the solution of larger reduced problems and develop the preconditioning strategy we use with such methods. In Section 2.4, we demonstrate the effectiveness of these techniques for solving two benchmark problems, the steady-state diffusion and convection-diffusion-reaction equations, and in Section 2.5, we draw some conclusions.

2.2 Offline-online reduced basis method

In a finite element setting, we seek a discrete solution u_h of the PDE in a finite-dimensional affine space X^h such that

$$\mathcal{L}(u_h(\cdot,\xi),v_h) = l(v_h) \ \ \forall v_h \in X_0^h \ . \tag{2.3}$$

For simplicity, we consider Dirichlet problems, and X_0^h is the subset of X^h corresponding to homogenous Dirichlet boundary conditions. Given a basis Q which spans $\{q_j\}_{j=1}^k$ such that $q_j \in X_0^h$, we solve the reduced model

$$\mathcal{L}(\tilde{u}_0(\cdot,\xi),v_h) = l(v_h) \ \ \forall v_h \in \text{span}(Q) \ , \tag{2.4}$$

for $\tilde{u}_0 \in \text{span}(Q)$, which is used to construct an approximation of the full solution, $\tilde{u} = \tilde{u}_0 + u_{bc}$, where u_{bc} is the solution on the boundary. The accuracy of this approximation depends on how well the reduced basis represents the solution space.

Thus, constructing this basis requires balancing two conflicting requirements: its rank, k, should be small enough so there is a benefit with respect to efficiency from using the reduced model, but k should also be large enough to ensure accuracy of the approximation.

We will also assume that the operators L and B in (2.1) and (2.2) are affinely dependent on ξ, i.e. for L,

$$L(\vec{x}, \xi; u) = \sum_{i=1}^{s_L} \psi_i(\xi)\; l_i(\vec{x}; u) \tag{2.5}$$

where $\{l_i(\vec{x}; u)\}_{i=1}^{s_L}$ are parameter-independent operators and $\psi_i : \mathbb{R}^m \to \mathbb{R}$. This assumption leads to efficiencies for linear operators as well as mildly nonlinear (say, quadratic [29]) ones, because part of the reduced model can be precomputed as part of the offline step and the cost of solving the reduced model does not depend on the size of the full model. For example, for a linear PDE the solution of the full model in equation (2.3) is obtained by solving a matrix equation of the form

$$A(\xi)u_\xi = f\ , \tag{2.6}$$

where the order of the system, N, depends on the number of points in the spatial discretization of D and is assumed to be large. Let Q be an $N \times k$ orthogonal matrix whose columns span the same space as that determined by the coefficient vectors of the set of snapshots.

2.2.1 Online costs for the reduced basis method

The Galerkin projection of the reduced model of order k is

$$Q^T A(\xi) Q u_{r,\xi} = Q^T f\ , \tag{2.7}$$

where $\tilde{u}_\xi = Qu_{r,\xi}$ is the approximation of the solution of equation (2.6) on the interior of D. Because of the assumption of affine dependence, the coefficient matrix has the structure

$$A(\xi) = \sum_{i=1}^{s_L+s_B} \psi_i(\xi) A_i \ , \tag{2.8}$$

and the reduced model can be written

$$\sum_{i=1}^{s_L+s_B} \psi_i(\xi)[Q^T A_i Q] u_{r,\xi} = Q^T f \ . \tag{2.9}$$

In this form, the matrices $\{Q^T A_i Q\}$ are parameter independent and thus can be precomputed as part of the offline step. The online step of the reduced model includes the assembly of the sum in equation (2.9). The cost of this computation is of order $(s_L + s_B)k^2$, and the total online cost is this plus the cost of solving an order k linear system. Hence, the cost of the reduced model is independent of N, the size of the full model. We will consider methods for handling nonlinear and/or nonaffine operators in the next chapter.

The conventional point of view is that the reduced model will be significantly less expensive to solve than the full model. The traditional choice for solving the reduced model in equation (2.7) is direct methods, at a computational cost of $O(k^3)$. On the other hand, it is often possible to use multigrid methods to solve the (full-sized) linear system arising from discretized PDEs, at cost $O(N)$ [12,31]. Therefore, using the reduced model with a direct method is effective only when $k \ll N$. The focus of this study is the case when the rank of the reduced basis k is of magnitude where the cost of direct methods for the reduced problem is not smaller than for solving the full problem, even though k is still of moderate size. In such situations, there may be an advantage to using alternative solution methods.

Consider the use of iterative methods for the reduced model (2.7). In this case, the cost of such methods is $O(pk^2)$ where p is the number of iterations required for convergence; the factor of k^2 comes from the cost of a dense matrix-vector product by $Q^T A(\xi)Q$. Thus, this will be an effective approach when p is small. It is well known that preconditioners are needed for the fast convergence of iterative methods. Thus, we need a preconditioner for the reduced matrix.

2.3 Preconditioners for the reduced model

Consider a reformulated version of equation (2.7) given by

$$\begin{bmatrix} A^{-1} & Q \\ Q^T & 0 \end{bmatrix} \begin{bmatrix} v \\ u_r \end{bmatrix} = \begin{bmatrix} 0 \\ -Q^T f \end{bmatrix} . \tag{2.10}$$

Equation (2.10) has the form of a saddle point system, a well-studied problem, for which a preconditioner may take the form [31]

$$\begin{bmatrix} F & 0 \\ 0 & S \end{bmatrix} .$$

With the formal choice $F = A^{-1}$, it can be shown that the optimal choice for S is the Schur complement [48], which for (2.10) is

$$S = Q^T A Q \ . \tag{2.11}$$

That this is equivalent to the matrix of the reduced model suggests that the reduced model in equation (2.7) can be preconditioned using the Schur complement or an approximation to the Schur complement. The first preconditioner we consider for the reduced-order model in equation (2.7) is the *exact Schur* preconditioner which requires the application of the inverse

$$S^{-1}(\xi) = (Q^T A(\xi) Q)^{-1} \ .$$

This is done by first computing the Cholesky factorization of $Q^T A(\xi) Q$ and then solving two triangular systems.

To produce an approximation of the Schur complement, we will mimic an approach used successfully in a different context (for models of computational fluid dynamics), the so-called least-squares commutator (LSC) method [31]. Here the Schur complement is approximated by the matrix

$$\hat{P}_S \equiv (Q^T Q)(Q^T A^{-1} Q)^{-1}(Q^T Q) \ . \tag{2.12}$$

Since Q is orthogonal, this operator simplifies to

$$\hat{P}_S = \left(Q^T A^{-1} Q\right)^{-1} \ .$$

This is referred to as the *exact LSC* preconditioner.

The exact Schur preconditioner depends on $(Q^T A Q)^{-1}$ the operation we are trying to approximate in the reduced model. The exact LSC preconditioner depends on A^{-1}, which is the operator we are trying to approximate in the full model. Thus both preconditioners are impractical. However, recall that A depends on a parameter ξ. We could choose a single representative vector of parameters, $\xi^{(0)}$, to define the preconditioner, which allows the construction of the preconditioner to be moved offline. Therefore, the exact Schur preconditioner $(Q^T A(\xi^{(0)}) Q)^{-1}$ can be constructed offline by computing its Cholesky factors. In the case of the exact LSC preconditioner we solve k full systems to compute $A^{-1}(\xi^{(0)})Q$ and premultiply by

Q^T. A variation of this idea is to use a collection of representative parameter vectors to define a collection of preconditioners, all computed in the offline step.

In the exact LSC preconditioner $\hat{P}_S$, we can replace A with a spectrally equivalent operator, i.e., one for which there exist σ_0 and σ_1 independent of spatial dimension such that

$$\sigma_0 \leq \frac{x^T A x}{x^T P_A x} \leq \sigma_1 \ . \tag{2.13}$$

Thus we can use a preconditioner designed for A to produce a preconditioner of S, yielding

$$P_S = (Q^T P_A^{-1} Q)^{-1} \ \text{ or } P_S^{-1} = Q^T P_A^{-1} Q \ \ . \tag{2.14}$$

This is referred to as the *approximate LSC* preconditioner. In this case we can construct P_S^{-1} explicitly by

- Constructing what is needed for a representation of P_A^{-1}. We will define P_A^{-1} using an algebraic multigrid (AMG) method. Therefore, this step consists of computing the sequence of coarse grids, grid transfer operators, and smoothing operators obtained for a multigrid solution of systems of discrete PDEs. With these, we have what is needed to apply the action of P_A^{-1} to a vector.

- Explicitly computing the (dense) order-k matrix $Q^T P_A^{-1} Q$ by applying the algebraic multigrid operation to each of the columns of Q and then premultiplying the matrix $P_A^{-1} Q$ by Q^T.

This study will consider the exact Schur preconditioner, the exact LSC preconditioner, and the approximate LSC preconditioner.

2.4 Numerical results

To illustrate the effectiveness of these ideas, we apply the reduced basis method to two examples of PDEs with random coefficients. We compare the performance of the iterative solver for the reduced model with the direct reduced solution and the multigrid solution of the full system.

The first example is a steady-state diffusion equation with parameter-dependent diffusion coefficient,

$$\begin{array}{rcll} -\nabla \cdot a(\vec{x},\xi)\nabla u(\vec{x},\xi) & = & f(\vec{x}) & \text{in} \quad D\times\Gamma \\ u(\vec{x},\xi) & = & g_D(\vec{x}) & \text{on} \quad \partial D_D\times\Gamma \\ a(\vec{x},\xi)\frac{\partial u(\vec{x},\xi)}{\partial n} & = & 0 & \text{on} \quad \partial D_N\times\Gamma \end{array}, \tag{2.15}$$

where $D\subset\mathbb{R}^2$ and the diffusion coefficient, $a(\vec{x},\xi)$, is a random field depending on a vector of m random variables, $\xi=[\xi_1,\xi_2,...,\xi_m]^T$. The second example is a steady-state convection-diffusion-reaction equation with an uncertain reaction coefficient, $r(\vec{x},\xi)$,

$$\begin{array}{rcll} -\nu\nabla^2 u(\vec{x},\xi)+\vec{w}\cdot\nabla u(\vec{x},\xi)+r(\vec{x},\xi)u(\vec{x},\xi) & = & f(\vec{x}) & \text{in} \quad D\times\Gamma \\ u(\vec{x},\xi) & = & g_D(\vec{x}) & \text{on} \quad \partial D_D\times\Gamma \\ \frac{\partial u(\vec{x},\xi)}{\partial n} & = & 0 & \text{on} \quad \partial D_N\times\Gamma, \end{array} \tag{2.16}$$

where the domain $D\subset\mathbb{R}^2$, ν is the diffusion coefficient, $\vec{w}$ is the convective velocity, and $\nabla\cdot\vec{w}=0$.

2.4.1 Adaptive offline construction

We turn now to the methodology used to compute a reduced basis Q. Assume the full discretized model $A(\xi)u_\xi=f$ is defined on a parameter space $\Gamma=\prod_{i=1}^{m}\Gamma_i$

such that $\xi_i \in \Gamma_i := [a_i, b_i]$. The reduced basis is constructed using an adaptive algorithm summarized in Algorithm 1. The procedure begins with Q as a single vector, the normalized discrete solution $u_{\xi^{(0)}}$ where $\xi^{(0)} = E[\xi]$. The parameter space is randomly sampled M times and for each sample, ξ, the reduced model is solved with the current Q. This produces an approximation to the solution $\tilde{u}_\xi = Qu_{r,\xi} + u_{bc}$ whose accuracy is estimated by an error indicator, η_ξ. If η_ξ exceeds a predefined tolerance, τ, the full solution for this ξ is computed and the new snapshot, u_ξ, is used to update the reduced basis. The basis matrix Q is augmented using the modified Gram-Schmidt algorithm, ensuring that the basis will have orthogonal columns. We used as an error indicator the relative residual

$$\eta_\xi = \frac{||A(\xi)\tilde{u}_\xi - f||_2}{||f||_2} \quad . \tag{2.17}$$

This method is applied to the steady-state diffusion equation and the steady-state convection-diffusion-reaction equation beginning with $M = 2000$ random samples of ξ. This produces a candidate basis Q. To assess the quality of this basis, we computed the reduced solution for an additional 100 samples; if each of these reduced solutions satisfied the error tolerance, then Q was accepted as the reduced basis. For case 1 of the diffusion equation (see below) and the convection-diffusion-reaction equation, this strategy produced an acceptable Q with a few exceptions. In general, $M \geq 3000$ was required for some experiments with the diffusion equation (referred to as case 2 below, where the details are stated).

Remark: The convergence properties of this strategy for offline computation of the basis are not known, in contrast to greedy search algorithms, which produce

Algorithm 1 Construction of the reduced basis using random selection

```
Compute $u_{\xi^{(0)}}$ using the full model
$Q = \left[ \frac{u_{\xi^{(0)}}}{||u_{\xi^{(0)}}||_2} \right]$
for j =1:M do
    Select random sample $\xi^{(j)} \in \Gamma$
    Solve the reduced model $Q^T A(\xi^{(j)}) Q u_{r,\xi^{(j)}} = Q^T f$
    Compute $\eta_{\xi^{(j)}}$
    if $\eta_{\xi^{(j)}} > \tau$ then
        Compute $u(\xi^{(j)})$ using the full model
        Use the snapshot to augment $Q$
    end if
end for
```

reduced bases of quasi-optimal dimension [9]. In a comparison of Algorithm 1 with a greedy algorithm, we found that for multiple examples of the benchmark problems studied in this section, the size of the reduced basis was never more than 10% larger than that produced by a greedy algorithm and in many cases the basis sizes were identical. The cost (in CPU time) of Algorithm 1 is significantly lower. Our concern is efficient implementation of the *online* step, and for simplicity we used Algorithm 1 for the offline computation.

2.4.2 Diffusion equation

The steady-state diffusion problem with a random coefficient in equation (2.15) can be used to model the effects of groundwater flow [74]. For more details on this problem, see [20]. The weak formulation for a fixed value of ξ is

$$(a_\xi \nabla u, \nabla v) = (f, v) \;\; \forall v \in H_0^1(D) \; . \tag{2.18}$$

Bilinear Q_1 elements are used to generate a discretized system, $A(\xi)u_\xi = f$ of order N for the full model [31]. We use source term $f(\vec{x}) = 1$. Boundary conditions will be addressed below for each case.

We consider two finite-dimensional representations of the random field for the diffusion coefficient $a(\vec{x},\xi)$: a truncated Karhunen-Loève (KL) expansion (case 1) and a piecewise constant coefficient (case 2). The truncated KL-expansion is defined by

$$a(\vec{x},\xi(\omega)) = \mu(\vec{x}) + \sum_{i=1}^{m} \sqrt{\lambda_i} a_i(\vec{x}) \xi_i(\omega) \; , \tag{2.19}$$

where $\mu(\vec{x})$ is the mean of the random field, λ_i and $a_i(\vec{x})$ are the eigenvalues and eigenfunctions of the covariance function, and $\xi_i(\omega)$ are independent uniform random variables. We take the covariance function to be

$$C(\vec{x},\vec{y}) = \sigma^2 \exp\left(-\frac{|x_1 - x_2|}{c} - \frac{|y_1 - y_2|}{c} \right) \quad , \tag{2.20}$$

where σ is the standard deviation and c is the correlation length, which describes the strength of the relationship between the value of the random field at $\vec{x}_1 = (x_1, y_1)$ and $\vec{x}_2 = (x_2, y_2)$. A large value of c implies that $a(\vec{x}_1,\xi)$ and $a(\vec{x}_2,\xi)$ are likely to be highly correlated. We will also use the truncated KL expansion to represent the reaction coefficient in the convection-diffusion-reaction equation (2.16).

For the piecewise constant diffusion coefficient, the domain, D, is divided into $m = n_d \times n_d$ subdomains as in Figure 2.1, where

$$a(\vec{x},\xi) = \xi_i \; , \tag{2.21}$$

on the ith subdomain. Here $\{\xi_i\}_{i=1}^{m}$ are independent uniform random variables defined on $\Gamma_i = [0.01, 1]$.

Consider the influence of the parameters on the overall value of the coefficient

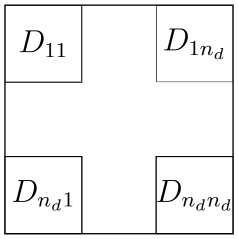


Figure 2.1: Domain for diffusion equation case 2: piecewise random coefficients.

for these two representations. The impact of the parameters in the truncated KL-expansion is unequal because the expansion weights the parameters by the eigenvalues of the covariance operator. Thus, for example, ξ_1 and ξ_2 are more influential to the value of $a(\vec{x}, \xi)$ than ξ_{m-1} and ξ_m, when the eigenvalues are labeled in decreasing order. In contrast, the piecewise random coefficients are equally weighted.

Algorithm 1 is used to generate the reduced basis Q. Once the reduced basis is generated we are able to solve the reduced problem defined in equation (2.7). The preconditioners for this system, discussed in Section 2.3 depend on parameters. For the exact Schur and exact LSC preconditioner, we consider two ways to select this parameter.

1. Offline: The mean parameter $\xi^{(0)} = E[\xi]$. The Cholesky factor of $Q^T A(\xi^{(0)})Q$ is computed offline for the exact Schur preconditioner. For the exact LSC preconditioner, $A(\xi^{(0)})^{-1}$ is applied to the columns of Q using a direct solve and $A(\xi^{(0)})^{-1}Q$ is premultiplied by Q^T.

2. Online: The parameter ξ is the same parameter whose solution we are seeking. This is an expensive online cost. For the exact Schur preconditioner case, it requires solving the reduced problem directly and in the exact LSC precondi-

tioner it requires solving the full model k times. The inclusion of this approach is to provide a comparison for the offline method and not as practical method for constructing preconditioners.

Recall that the approximate LSC preconditioner, defined in equation (2.14), utilizes P_A^{-1}, a preconditioner of A. We will specify P_A^{-1} using multigrid, which is well known to be effective for diffusion problems [12]. For the implementation, we use a smoothed aggregation algebraic multigrid routine from Python Algebraic Multigrid package (PyAMG) with the default settings [8]: the presmoother and postsmoother are one iteration of Gauss-Seidel, the maximum size of the coarse grid is 500, and the pseudoinverse is used to solve the system on the coarse grid. To compute the preconditioner for the reduced problem, the multigrid operator P_A^{-1} is applied to Q, by performing one V-cycle on each of the k columns of Q. We study three ways to select the parameter used to specify P_A.

1. Single-parameter offline: P_0 is derived from multigrid applied to $A(\xi^{(0)})$ where $\xi^{(0)}$ is the mean parameter, $E[\xi]$.

2. Multiple-parameter offline: A set of s parameters is used to define s precomputed offline preconditioners, $P_1, \ldots, P_s$. This is done using multigrid applied to $A(\xi^{(1)}), \ldots, A(\xi^{(s)})$. For the online component given ξ, $\xi^{(j)} \in \{\xi^{(1)}, \ldots, \xi^{(s)}\}$ is selected such that $||\xi^{(j)} - \xi||_2$ is minimized and P_j is used as the preconditioner. There are several possibilities for choosing $\{\xi^{(1)}, \ldots, \xi^{(s)}\}$ including random sampling, quasi-random sampling, and sparse grids. Sparse grids are used to limit costs of quadrature and interpolation of functions depending on

high-dimensional parameter sets. Since we are working with high-dimensional parameter spaces and would like to represent the parameter space with as few parameters as possible, we choose the so-called No Boundary sparse grid [41]. The first level of the grid, of size $s = 2m + 1$, is obtained using the `spinterp` toolbox [42].

3. Online: $P_{A(\xi)}$ comes from multigrid applied to $A(\xi)$ where ξ is the same parameter whose solution we are seeking. The time to construct the preconditioner online is quite large. It requires building the coarse grid and smoothing operators and the significantly more expensive step of applying them to each column of Q in order to compute $Q^T P_{A(\xi)}^{-1} Q$. It is included here to give a lower bound for how well offline preconditioning could perform.

The examples are implemented using Python and run with an Intel 2.9 GHz i7 processor and 8 GB of RAM. (The full model finite element discretizations are imported from the Incompressible Flow and Iterative Solver Software (IFISS) package which is implemented in Matlab [68]). The full solution is obtained using algebraic multigrid with stopping criterion

$$||f - A(\xi)u_j||_2 \leq 10^{-5}||f||_2 \ ,$$

where u_j is the solution after j iterations of multigrid, implemented with the same settings outlined above. For iterative solution of the reduced problem, we use the preconditioned conjugate gradient (PCG) method with stopping criterion

$$\frac{||Q^T f - Q^T A Q u_{r,j}||}{||Q^T f||} < \frac{\tau}{10} \ , \tag{2.22}$$

where $u_{r,j}$ is the reduced iterate at step j. The given times for online preconditioning do not include the significant time required to construct the multigrid preconditioner, and the time for multiple-parameter preconditioning does not include the trivial time to find the minimizer ξ^*.

Case 1: Truncated Karhunen-Loève expansion.

The random field, $a(\vec{x}, \xi)$, is represented by a truncated Karhunen-Loève expansion defined on $D = [0,1] \times [0,1]$ described in equation (2.19). Dirichlet boundary conditions $g_D(\vec{x}) = 0$ are imposed on the boundary where $x = 0$ and $x = 1$ and homogenous Neumann conditions are used on the remainder of the boundary.

We choose ξ_i to be independent and uniformly distributed random variables on $\Gamma_i = [-1, 1]$ and fix $\mu(\vec{x}) = 1$ and $\sigma = 0.5$. The correlation length c is varied; the number of parameters m is chosen to ensure that 95% of the variance in the random field is captured, i.e.

$$\frac{\sum_{i=1}^{m} \lambda_i}{\sum_{i=1}^{N} \lambda_i} \geq 0.95 \ . \tag{2.23}$$

Algorithm 1 with $M = 2000$ was used to construct a basis using both $\tau = 10^{-5}$ and $\tau = 10^{-8}$ for the error tolerance.[1] Decreasing the tolerance has the effect of increasing the size of the reduced basis, and for smaller τ the reduced model solutions from both direct and iterative methods require additional time; this tolerance has no effect on the full system solution.

[1]The example with $m = 325$ parameters (see Table 2.1) required $M = 3000$ for $\tau = 10^{-5}, N = 257^2$, and $\tau = 10^{-8}, N \geq 65^2$.

N	c		3	1.5	0.75	0.375
	m		7	17	65	325
33^2	k		36	91	237	501
	exact Schur	Offline	3.9	4.2	4.5	4.6
	exact Schur	Online	1.0	1.0	1.0	1.0
	exact LSC	Offline	5.1	5.7	5.0	5.0
	exact LSC	Online	5.0	5.0	4.0	4.0
	inexact LSC	Offline	6.0	6.6	6.3	6.4
	inexact LSC	Online	6.0	6.0	6.0	6.0
65^2	k		35	93	250	603
	exact Schur	Offline	3.9	4.1	4.5	4.7
	exact Schur	Online	1.0	1.0	1.0	1.0
	exact LSC	Offline	5.9	6.0	6.0	5.8
	exact LSC	Online	5.0	5.1	5.0	5.0
	inexact LSC	Offline	6.0	6.3	6.2	6.1
	inexact LSC	Online	6.0	6.0	6.0	6.0
129^2	k		35	95	259	642
	exact Schur	Offline	3.9	4.1	4.6	4.7
	exact Schur	Online	1.0	1.0	1.0	1.0
	exact LSC	Offline	5.7	6.2	6.7	6.4
	exact LSC	Online	5.0	5.9	6.0	5.7
	inexact LSC	Offline	6.3	7.3	8.0	8.1
	inexact LSC	Online	6.1	7.0	8.0	8.0
257^2	k		35	96	263	657
	exact Schur	Offline	3.8	4.2	4.6	4.6
	exact Schur	Online	1.0	1.0	1.0	1.0
	exact LSC	Offline	6.0	6.4	6.9	7.2
	exact LSC	Online	5.0	6.0	6.0	6.0
	inexact LSC	Offline	6.9	8.0	8.2	8.7
	inexact LSC	Online	7.0[2]	8.0	8.0	8.3

Table 2.1: Average iteration counts for preconditioned conjugate gradient algorithm applied to the reduced diffusion problem in case 1 (KL expansion), with $\tau = 10^{-5}$.

To assess performance, we solve the reduced problem for 100 randomly chosen parameters using a direct method, the conjugate gradient method without preconditioning, and the conjugate gradient method for the exact Schur, exact LSC, and

[2]This case is anomalous because the offline preconditioners converge in one fewer iteration than the online preconditioner for several samples.

approximate LSC preconditioners. Table 2.1 presents the average iteration counts for the conjugate gradient method for the three preconditioners. The time (in seconds) for the full algebraic multigrid solution, the reduced direct method, and the offline conjugate gradient method are presented in Table 2.2 with the fastest method for each case in bold. Table 2.3 shows the costs of constructing the offline preconditioner for each of the three methods.

N			c / m	3 / 7	1.5 / 17	0.75 / 65	0.375 / 325
33^2			k	36	91	237	501
	Full	AMG		0.0145	0.0142	0.0142	0.0155
	Reduced	Direct		**0.0002**	0.0004	0.0016	0.0092
	Reduced	Iterative	exact Schur	**0.0002**	**0.0003**	**0.0004**	**0.0027**
	Reduced	Iterative	exact LSC	0.0003	**0.0003**	0.0005	0.0031
	Reduced	Iterative	inexact LSC	0.0003	**0.0003**	0.0005	0.0034
65^2			k	35	93	250	603
	Full	AMG		0.1718	0.1643	0.1662	0.1791
	Reduced	Direct		**0.0002**	0.0005	0.0018	0.0165
	Reduced	Iterative	exact Schur	**0.0002**	**0.0003**	**0.0005**	**0.0038**
	Reduced	Iterative	exact LSC	0.0003	**0.0003**	**0.0005**	0.0051
	Reduced	Iterative	inexact LSC	0.0003	**0.0003**	0.0006	0.0051
129^2			k	35	95	259	642
	Full	AMG		0.1041	0.1080	0.1076	0.1227
	Reduced	Direct		**0.0002**	0.0005	0.0020	0.0186
	Reduced	Iterative	exact Schur	**0.0002**	**0.0003**	**0.0007**	**0.0054**
	Reduced	Iterative	exact LSC	0.0003	**0.0003**	0.0009	0.0069
	Reduced	Iterative	inexact LSC	0.0003	0.0004	0.0010	0.0089
257^2			k	35	96	263	657
	Full	AMG		0.3432	0.3289	0.3343	0.3660
	Reduced	Direct		**0.0002**	0.0005	0.0020	0.0194
	Reduced	Iterative	exact Schur	**0.0002**	**0.0003**	**0.0007**	**0.0051**
	Reduced	Iterative	exact LSC	0.0003	**0.0003**	0.0010	0.0078
	Reduced	Iterative	inexact LSC	0.0003	0.0004	0.0011	0.0092

Table 2.2: Average CPU time for solving the reduced diffusion problem in case 1 (KL expansion), with $\tau = 10^{-5}$.

The exact Schur preconditioner, as expected, produces the lowest iteration

N	c / m	3 / 7	1.5 / 17	0.75 / 65	0.375 / 325
33^2	k	36	91	237	501
	exact Schur	0.0007	0.003	0.02	0.08
	exact LSC	0.14	0.37	1.11	2.15
	inexact LSC	0.14	0.15	0.22	0.35
65^2	k	35	93	250	603
	exact Schur	0.002	0.007	0.05	0.33
	exact LSC	0.75	2.07	5.36	14.7
	inexact LSC	0.31	0.37	0.58	1.29
129^2	k	35	95	259	642
	exact Schur	0.009	0.03	0.21	1.05
	exact LSC	4.38	11.9	31.3	93.7
	inexact LSC	0.33	0.50	1.24	4.30
257^2	k	35	96	263	657
	exact Schur	0.03	0.11	1.06	4.83
	exact LSC	28.0	82.4	221	568
	inexact LSC	0.79	1.90	4.89	14.2

Table 2.3: CPU time to construct the (offline) preconditioner for $\tau = 10^{-5}$.

counts of the three preconditioners seen in Table 2.1. The exact LSC, $\hat{P}_S$, performs next best in terms of iteration count, though not significantly. However, Table 2.3 shows that the approximate LSC, P_S, is significantly cheaper to construct than the exact LSC. The advantage of the approximate LSC preconditioner over the exact Schur preconditioner is that is based on a preconditioner of the full model and thus can be adapted for any preconditioner of the full model. Secondly, it has the advantage that it can be updated quickly when Q, the preconditioner, or the parameter is updated. Although, the results presented in this study consider only the case where the preconditioners are $k \times k$ matrices formed offline, we remark that the approximate LSC preconditioner could be applied as a matvec. First, the matrix Q is applied, then the full multigrid preconditioner followed by multiplication by Q^T. Although this cost would scale with the dimension N of the full problem, this approach allows the flexibility of changing the reduced basis or preconditioner online. This approach may be required in cases where the solution method is not strictly divided into offline and online steps. Such a strategy would be impractical with the exact Schur and exact LSC preconditioners.

Table 2.2 demonstrates that the iterative methods are faster than direct methods for $k \geq 91$. For the remainder of this chapter, we will perform comparisons using only the approximate LSC preconditioner for the reduced iterative method. The average iteration counts for the conjugate gradient method for the approximate LSC preconditioner for a single and multiple offline parameters and $\tau = 10^{-8}$ are presented in Table 2.4. The time (in seconds) for the full algebraic multigrid solution, the reduced direct method, and the single-parameter offline conjugate gradient method are presented in Table 2.5 with the fastest method for each case again shown in bold.

Table 2.4 shows that the number of iterations needed for PCG grows only slightly as the size of the reduced basis grows, whereas the iterations for unpreconditioned conjugate gradient grow significantly. Also note that the single-parameter preconditioner performs nearly as well as the online preconditioner, so using the mean parameter to construct the preconditioner is an effective choice for the entire parameter space.

Table 2.5 illustrates that the single-parameter offline preconditioned conjugate gradient method is faster than direct methods when the reduced basis is of size $k \geq 254$. For $\tau = 10^{-8}$ this holds for both $m = 17$ and $m = 65$. The improvement is

N		c	3	1.5	0.75	0.375
		m	7	17	65	325
33^2	k		97	254	607	982
	None		60.1	90.7	101.7	103.9
	Single		10.0	9.3	9.5	8.9
	Multiple		10.0	9.3	9.5	8.9
	Online		10.0	9.0	9.0	8.0
65^2	k		100	265	699	1679
	None		68.8	129.3	175.5	200.3
	Single		10.0	10.0	8.5	8.7
	Multiple		10.0	10.0	8.5	8.7
	Online		10.0	9.8	8.0	8.0
129^2	k		102	269	729	1808
	None		70.1	149.5	252.5	339.1
	Single		11.2	14.6	12.9	11.0
	Multiple		11.2	14.6	12.9	11.0
	Online		11.0	14.8	13.0	11.0
257^2	k		102	275	740	1846
	None		70.4	154.0	293.6	473.7
	Single		11.0	13.7	15.4	13.5
	Multiple		11.0	13.7	15.4	13.5
	Online		11.0	13.0	15.0	13.0

Table 2.4: Average iteration counts for preconditioned conjugate gradient algorithm applied to the reduced diffusion problem in case 1 (KL expansion), with $\tau = 10^{-8}$ using the approximate LSC preconditioner.

N	c		3	1.5	0.75	0.375
	m		7	17	65	325
33^2	k		97	254	607	982
	Full	AMG	0.0202	0.0205	0.0214	0.0228
	Reduced	Direct	**0.0003**	0.0016	0.0181	0.0699
	Reduced	Iterative	0.0004	**0.0008**	**0.0036**	**0.0103**
65^2	k		100	265	699	1679
	Full	AMG	0.1768	0.1961	0.1947	0.1974
	Reduced	Direct	**0.0003**	0.0021	0.0262	0.3207
	Reduced	Iterative	0.0004	**0.0010**	**0.0044**	**0.0252**
129^2	k		102	269	729	1808
	Full	AMG	0.1195	0.1286	0.1347	0.1443
	Reduced	Direct	**0.0003**	0.0020	0.0287	0.4452
	Reduced	Iterative	0.0005	**0.0013**	**0.0070**	**0.0449**
257^2	k		102	275	740	1846
	Full	AMG	0.3163	0.2988	0.3030	0.3778
	Reduced	Direct	**0.0004**	0.0024	0.0302	0.4498
	Reduced	Iterative	0.0005	**0.0012**	**0.0088**	**0.0619**

Table 2.5: Average CPU time for solving the reduced diffusion problem in case 1 (KL expansion), with $\tau = 10^{-8}$ using the offline approximate LSC preconditioner.

more dramatic for the case of $m = 65$, when the reduced basis size is $k \approx 700$. For all values of m and N the reduced iterative method is more efficient than solving the full system.

For this example, the size of the reduced basis is consistent as the spatial size, N, is increased. This is especially clear for the smaller values of m. This is expected; see discussion in [29] suggesting that this size is in correspondence with the rank of the underlying solution space associated with the continuous model. There is some growth in k the basis size, for the larger values of m, but we expect these values to eventually tend toward a constant as the spatial resolution increases. Since the cost of solving the full system grows with N, as expected, the advantage of using the reduced model also increases as the mesh is refined.

Case 2: Piecewise constant coefficient.

The diffusion coefficient, $a(\vec{x}, \xi)$, for this case is defined in equation (2.21) on a domain $D = [-1, 1] \times [-1, 1]$ with $g_D(\vec{x}) = 0$ on the entire boundary. Algorithm 1 with $M = 3000$ and $\tau = 10^{-8}$ was used to construct the bases.[3] The average iteration counts for solving 100 reduced problems are given in Table 2.6 for the conjugate gradient method.

	m	4	16	36	64	100
33^2	k	27	193	321	449	577
	None	31.9	113.9	126.4	127.9	128.0
	Single	17.2	32.3	44.0	52.0	59.5
	Multiple	15.7	30.1	42.4	50.3	57.0
	Online	11.4	13.0	13.6	12.7	12.0
65^2	k	29	309	625	897	1153
	None	42.3	234.1	254.9	258.3	256.4
	Single	20.1	38.2	47.0	54.8	64.2
	Multiple	18.7	35.5	44.9	53.1	65.4
	Online	14.3	17.0	18.2	18.9	18.9
129^2	k	33	359	862	1519	2219
	None	60.3	432.9	493.6	519.2	518.9
	Single	24.2	37.5	47.6	58.1	64.9
	Multiple	22.7	35.2	45.2	56.0	72.4
	Online	19.1	19.0	22.0	24.1	25.2
257^2	k	36	394	979	1789	2801
	None	82.0	808.9	976.8	1035.6	1037.3
	Single	30.4	44.0	50.9	62.2	71.1
	Multiple	28.6	41.7	48.5	60.3	84.1
	Online	25.1	25.8	25.7	27.8	29.7

Table 2.6: Average iteration counts for preconditioned conjugate gradient algorithm applied to the reduced diffusion problem in case 2, with $\tau = 10^{-8}$ with approximate LSC preconditioner.

In contrast to the results for case 1, the iteration counts for the offline approx-

[3] The example with $m = 100$ parameters and $N = 257^2$ required $M = 4000$ to construct a basis that meets the criteria discussed earlier in this section.

N	m		4	16	36	64	100
33^2	k		27	193	321	449	577
	Full	AMG	0.0218	0.0203	0.0215	0.0210	0.0208
	Reduced	Direct	**0.0001**	**0.0010**	**0.0032**	**0.0073**	**0.0152**
	Reduced	Iterative	0.0006	0.0019	0.0045	0.0090	0.0181
65^2	k		29	309	625	897	1153
	Full	AMG	0.1679	0.1601	0.1669	0.1811	0.1760
	Reduced	Direct	**0.0002**	**0.0026**	0.0187	0.0543	0.1088
	Reduced	Iterative	0.0007	0.0034	**0.0176**	**0.0458**	**0.0832**
129^2	k		33	359	862	1519	2219
	Full	AMG	0.1134	0.1202	0.1357	**0.1184**	**0.1194**
	Reduced	Direct	**0.0002**	**0.0038**	0.0461	0.2319	0.6659
	Reduced	Iterative	0.0009	0.0041	**0.0364**	0.1340	0.3060
257^2	k		36	394	979	1789	2801
	Full	AMG	0.3376	0.3519	0.3291	0.3365	**0.3568**
	Reduced	Direct	**0.0002**	**0.0051**	0.0670	0.3555	1.2972
	Reduced	Iterative	0.0010	0.0060	**0.0485**	**0.1928**	0.5365

Table 2.7: Average CPU time solving the reduced diffusion problem in case 2 (piecewise constant), with $\tau = 10^{-8}$ with the approximate LSC preconditioner.

imate LSC preconditioner are somewhat larger than those for the online ones (see Table 2.6). We attribute this to the fact that for this example, all the parameters are weighted equally in their contribution to the model, unlike the situation for the KL expansion. Thus, the single (or small number) of parameter sets used for the offline preconditioners are not as effective at capturing the character of the parameter space. Despite this, the important trends for the preconditioned solvers are the same as for case 1: iteration counts depend only mildly on the number of terms m in equation (2.5) or the size k of the reduced basis. There is little advantage of the "multiple-parameter" over the "single-parameter" approach. Thus we use this single-parameter preconditioned conjugate gradient method as the iterative method to compare to the reduced direct and full multigrid methods in Table 2.7.

We highlight the trends displayed in Table 2.7 as follows.

- For the reduced problem, the iterative solver is more efficient than the direct solver for large reduced bases, in particular whenever the size k of the reduced basis is greater than or equal to 625.

- As the dimension of the spatial discretization increases, the solution of the reduced model is less expensive than solution of the full model. Moreover, as in case 1, the size of the reduced basis tends to a constant as the mesh is refined, so solution costs also tend to a constant.

- For fixed spatial dimension, the costs of solving the full system are constant whereas the size of the reduced model increases with the number of parameters, m, and N. For the largest choices of these values, $m = 100$ and $N = 257^2$, the full AMG costs are lowest. However, for fine enough spatial meshes such that k has stabilized (as in case 1), we expect that the cost of the reduced model will be smaller.

2.4.3 Behavior of eigenvalues

The performance of the preconditioned conjugate gradient method for solving the reduced problem depends on the extremal values of the Rayleigh quotient

$$\frac{\hat{x}^T Q^T A(\xi) Q \hat{x}}{\hat{x}^T P(\xi^{(0)}) \hat{x}} \tag{2.24}$$

where $P(\xi^{(0)})$ is the offline reduced preconditioner. For the exact Schur case when $\xi = \xi^{(0)}$ this quotient is

$$\frac{\hat{x}^T Q^T A(\xi) Q \hat{x}}{\hat{x}^T Q^T A(\xi^{(0)}) Q \hat{x}}\ , \tag{2.25}$$

and it is clearly one. Consider now the case when $\xi \neq \xi^{(0)}$. For case 1, when the diffusion coefficient is defined by the KL expansion, this quantity can be bound using [58, Lemma 3.4]. We discuss the case where $a_k(\vec{x})$, the eigenfunctions of the covariance operators, are uniformly positive. The stiffness matrix is

$$A(\xi) = A_0 + \sum_{k=1}^{m} \xi_k A_k$$

where (i, j) entry of A_0 is

$$A_0(i,j) = \mu \int_D \nabla \phi_i(\vec{x}) \nabla \phi_j(\vec{x}) d\vec{x}$$

and the (i, j) entry of A_k is

$$A_k(i,j) = \sigma \sqrt{\lambda_k} \int_D a_k(\vec{x}) \nabla \phi_i(\vec{x}) \nabla \phi_j(\vec{x}) d\vec{x} \ ,$$

where μ, σ, λ_k, and $a_k(\vec{x})$ are from equation (2.19). Next define

$$\begin{array}{rcl} a_k^{min} & = & \inf_{\vec{x} \in D} a_k(\vec{x}) \\ a_k^{max} & = & \sup_{\vec{x} \in D} a_k(\vec{x}) \ . \end{array}$$

Define $v \in X_0^h$, so $v = \sum_i x_i \phi_i(\vec{x})$ and

$$x^T A(\xi) x = x^T A_0 x + \sum_k \xi_k \sigma \sqrt{\lambda_k} \int_D a_k(\vec{x}) \nabla v \cdot \nabla v \leq x^T A_0 x + \sum_k \xi_k \frac{\sigma}{\mu} \sqrt{\lambda_k} a_k^{max} x^T A_0 x$$

$$x^T A(\xi) x = x^T A_0 x + \sum_k \xi_k \sigma \sqrt{\lambda_k} \int_D a_k(\vec{x}) \nabla v \cdot \nabla v \geq x^T A_0 x + \sum_k \xi_k \frac{\sigma}{\mu} \sqrt{\lambda_k} a_k^{min} x^T A_0 x \ .$$

Thus,

$$1 + \sum_k \xi_k \frac{\sigma}{\mu} \sqrt{\lambda_k} a_k^{min} \leq \frac{x^T A(\xi) x}{x^T A(\xi^{(0)}) x} \leq 1 + \sum_k \xi_k \frac{\sigma}{\mu} \sqrt{\lambda_k} a_k^{max} \ . \tag{2.26}$$

Since this holds for all x, it holds for $x = Q\hat{x}$, yielding bounds for the exact Schur preconditioner

$$1 + \sum_k \xi_k \frac{\sigma}{\mu} \sqrt{\lambda_k} a_k^{min} \leq \frac{\hat{x}^T Q^T A(\xi) Q \hat{x}}{\hat{x}^T Q^T A(\xi^{(0)}) Q \hat{x}} \leq 1 + \sum_k \xi_k \frac{\sigma}{\mu} \sqrt{\lambda_k} a_k^{max} \ . \tag{2.27}$$

For case 2, the definition of ξ is piecewise constant and each component of ξ, ξ_k, is independent and uniform on $\Gamma_k = [a, b]$ where $a = 0.01$ and $b = 1$, with mean $\mu = 0.505$. The stiffness matrix is $A(\xi) = \sum_k \xi_k A_k$ where A_k corresponding to the subdomain D_k is

$$A_k(i,j) = \int_{D_k} \nabla\phi_i(\vec{x})\nabla\phi_j(\vec{x})d\vec{x} \ .$$

Therefore for the mean parameter $A(\xi^{(0)}) = \mu \sum_k A_k$. Then,

$$x^T A(\xi)x = x^T \left(\sum_k \xi_k A_k\right) x \leq \frac{b}{\mu} x^T A_0 x$$

$$x^T A(\xi)x = x^T \left(\sum_k \xi_k A_k\right) x \geq \frac{a}{\mu} x^T A_0 x \ .$$

Therefore, the exact Schur preconditioner has the following bounds

$$0.0198 = \frac{a}{\mu} \leq \frac{\hat{x}^T Q^T A(\xi) Q\hat{x}}{\hat{x}^T Q^T A(\xi^{(0)}) Q\hat{x}} \leq \frac{b}{\mu} = 1.9802 \ . \tag{2.28}$$

For a general preconditioner $P(\xi^{(0)})$,

$$\frac{\hat{x}^T Q^T A(\xi) Q\hat{x}}{\hat{x}^T P(\xi^{(0)})\hat{x}} = \frac{\hat{x}^T Q^T A(\xi) Q\hat{x}}{\hat{x}^T Q^T A(\xi^{(0)}) Q\hat{x}} \frac{\hat{x}^T Q^T A(\xi^{(0)}) Q\hat{x}}{\hat{x}^T P(\xi^{(0)})\hat{x}} \ . \tag{2.29}$$

The first quotient is the Rayleigh for the exact Schur preconditioner, and thus it can be bounded as described above. The second quotient is the Rayleigh quotient associated with the mean parameter. Therefore, for the remainder of the section we restrict our discussion of the bounds for the exact LSC and approximate LSC preconditioners to the Rayleigh quotient for the mean parameter, and for simplicity of notation we denote $A(\xi^{(0)}) = A$.

For the exact LSC preconditioner the Rayleigh quotient is

$$\frac{\hat{x}^T Q^T A Q\hat{x}}{\hat{x}^T (Q^T A^{-1} Q)^{-1}\hat{x}} \ . \tag{2.30}$$

For the approximate LSC preconditioner, the Rayleigh quotient is written as a product of two quotients

$$\frac{\hat{x}^T Q^T A Q \hat{x}}{\hat{x}^T (Q^T P_A^{-1} Q)^{-1} \hat{x}} = \frac{\hat{x}^T Q^T A Q \hat{x}}{\hat{x}^T (Q^T A^{-1} Q)^{-1} \hat{x}} \ \frac{\hat{x}^T (Q^T A^{-1} Q)^{-1} \hat{x}}{\hat{x}^T (Q^T P_A^{-1} Q)^{-1} \hat{x}} \ . \tag{2.31}$$

The first quotient is the Rayleigh quotient associated with the exact LSC preconditioner. Let us consider the second quotient on the right side of (2.31). We have assumed in equation (2.13) that P_A is spectrally equivalent to A. When A and P are symmetric positive definite, an analogous bound also holds for the inverses [73],

$$\sigma_0 \leq \frac{x^T P_A^{-1} x}{x^T A^{-1} x} \leq \sigma_1 \ \ \forall x \in \mathbb{R}^N \ .$$

We have assumed that this bound holds for all y, so specifically it holds for y on the range of Q (i.e. $x = \hat{x}$). Using this fact and applying inverses yields

$$\sigma_0 \leq \frac{\hat{x}^T (Q^T A^{-1} Q)^{-1} \hat{x}}{\hat{x}^T (Q^T P_A^{-1} Q)^{-1} \hat{x}} \leq \sigma_1 \ . \tag{2.32}$$

Therefore the second quotient in (2.31) is bounded by σ_0 and σ_1.

We can obtain insight into the Rayleigh quotient of the exact LSC preconditioner and the first quotient of equation (2.31) by experimentally examining the eigenvalues of

$$Q^T A(\xi^{(0)})^{-1} Q Q^T A(\xi^{(0)}) Q$$

using the benchmark problem from the previous section, case 2 of the diffusion equation. Figure 2.2 illustrates the eigenvalues for four values of m considered for this problem. All eigenvalues are bounded below by 1 and the largest eigenvalues grow only slightly with spatial dimension for the three cases where $m > 4$. This suggests

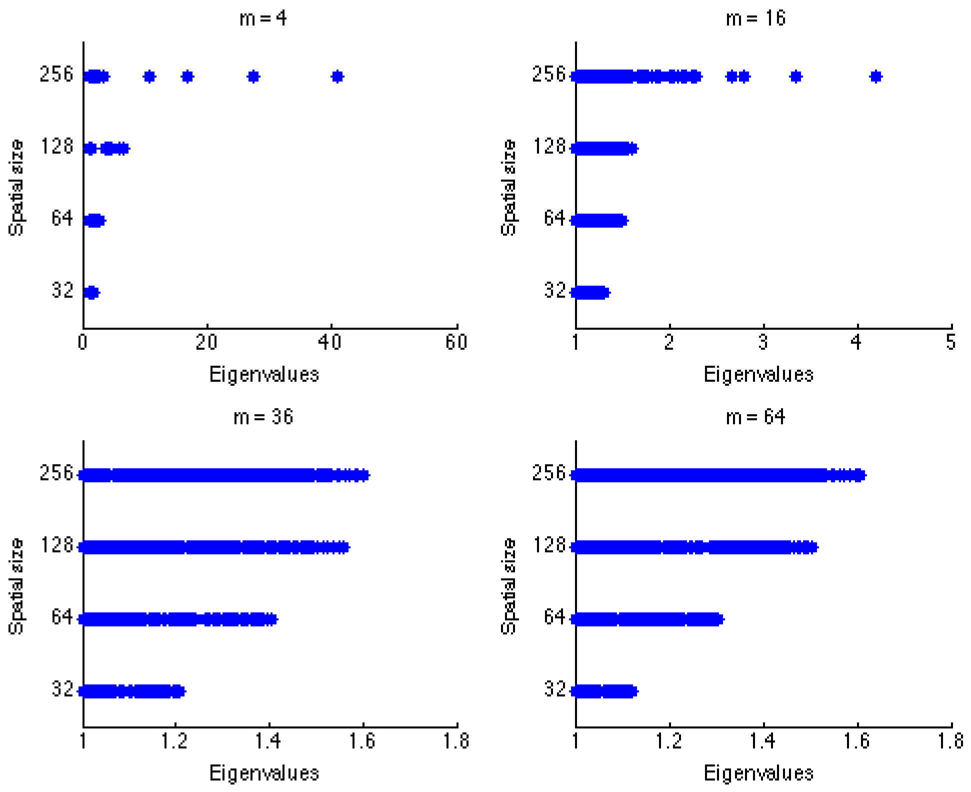


Figure 2.2: Eigenvalues of $Q^T A(\xi^{(0)})^{-1} Q Q^T A(\xi^{(0)}) Q$

that the condition number of the preconditioned reduced matrix is independent of the spatial mesh.

2.4.4 Convection-diffusion-reaction equation

The convection-diffusion-reaction equation (2.16) has applications in modeling fluid flow and chemical reactions. It can be used to model the transportation of contaminants in a flow subject to diffusive effects and/or chemical reactions [43]. Such models depend on parameters for the diffusion coefficient, the velocity, and the reaction coefficient. Any of these parameters could be uncertain [71]; here we consider the case where the reaction rate is taken to be a random field depending linearly on a random vector. The weak formulation is

$$\nu(\nabla u, \nabla v) + (\vec{w} \cdot \nabla u, v) + (r_\xi u, v) = (f, v) \ \ \forall v \in H_0^1(D) \ . \tag{2.33}$$

We present results for the steady-state model posed on domain $D = [-1,1] \times [-1,1]$ with Dirichlet boundary conditions

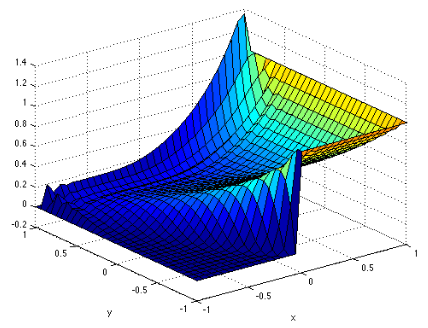
$$g_D(\vec{x}) = \begin{cases} 0 \ \ \text{for} \quad [-1, y] \bigcup [x, 1] \bigcup [-1 \le x \le 0, -1] \\ 1 \ \ \text{for} \quad [1, y] \bigcup [0 \le x \le 1, -1] \end{cases} \tag{2.34}$$

and an inflow boundary condition on the boundaries, $[x, -1]$ and $[1, y]$. We use source term $f(\vec{x}) = 0$ and a constant velocity $\vec{w} = (-\sin\frac{\pi}{6}, \cos\frac{\pi}{6})$. The diffusion coefficient is $\nu = 0.005$. The reaction rate, $r(x, \xi)$, is represented by a truncated Karhunen-Loève expansion as in equation (2.19), with ξ_i independent and uniformly distributed on $\Gamma_i = [-1, 1]$, with mean, $\mu = 1$, and standard deviation, $\sigma = 0.5$. As in case 1 of the diffusion equation, the value of the correlation coefficient c is varied, and the number of parameters m is chosen to capture 95% of the variance of the random field.

We again discretize using bilinear finite elements, which yields operators A, B, and $R(\xi)$ in which A represents the diffusion term, B, the convective term, and $R(\xi)$ the reaction term. We include stabilization by the streamline-diffusion method in the convection-dominated case when the mesh Peclet number,

$$P_h = \frac{h_e ||\vec{w}||}{2\nu} > 1 \ , \tag{2.35}$$

where h_e is a measure of the element length in the direction of the wind. This method produces matrices S_{cd} and S_r, defined in terms of the finite element basis


(a) Without streamline-diffusion method

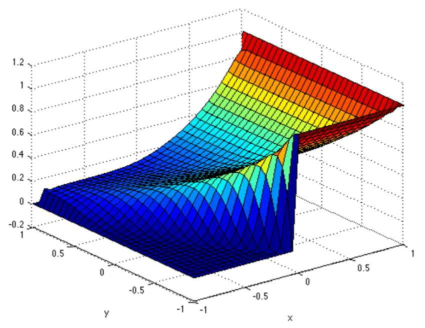

(b) With streamline-diffusion method

Figure 2.3: Solution of the convection-diffusion-reaction problem for $N = 33^2$, $\xi = \xi^{(0)}$, $c = 2$, $m = 36$, $P_h = 7.2$ with and without streamline-diffusion stabilization.

functions $\{\phi_i\}_{i=1}^{n_e}$ as

$$[S_{cd}]_{ij} = \sum_{e=1}^{n_e} \int_{\Omega_e} \delta_e(\vec{w} \cdot \nabla\phi_i)(-\nabla \cdot (\nu\nabla\phi_j) + \vec{w} \cdot \nabla\phi_j)$$

and

$$[S_r(\xi)]_{ij} = \sum_{e=1}^{n_e} \int_{\Omega_e} \delta_e(\vec{w} \cdot \nabla\phi_i) r(\vec{x}, \xi)\phi_j \ ,$$

where [31, p. 247]

$$\delta_e = \frac{h_e}{2||\vec{w}||}\left(1 - \frac{1}{P_h}\right) \ .$$

The resulting linear system has the form

$$F(\xi)u_\xi = f \ , \tag{2.36}$$

where $F(\xi) = A+B+R(\xi)+S_{cd}+S_r(\xi)$. As is well known [4,31,54], this stabilization enhances the quality of solutions with steep gradients obtained using inadequately fine grids, limiting the presence of nonphysical oscillations in discrete solutions; see Figure 2.4.4. As the discretization is refined, the stabilization becomes unnecessary.

We now consider solving the reduced problem

$$Q^T F(\xi) Q u_r = Q^T f \tag{2.37}$$

(This formulation corresponds to a stabilized version of the reduced model referred to as an "offline-online" stabilized method in [54]. Cf. also [21] for alternative ways to handle models containing convection terms.) As above, we use iterative methods where Q is constructed using Algorithm 1 with $M = 2000$ and $\tau = 10^{-8}$. Since the system is not symmetric, we use the stabilized biconjugate gradient method (BICGSTAB) in conjunction with the approximate LSC preconditioner $Q^T P_F^{-1} Q$, where P_F^{-1} is constructed using one of two methods:

1. Offline: P_F^{-1} is a multigrid preconditioner of $F(\xi^{(0)})$ where $\xi^{(0)}$ is the mean of the parameter space, $E[\xi]$.

2. Online: P_F^{-1} is a multigrid preconditioner of $F(\xi)$.

As with the diffusion equation, the multigrid preconditioners are constructed using a smoothed aggregation algebraic multigrid routine from PyAMG [8]. The examples with $N \leq 129^2$ required streamline-diffusion stabilization; for $N = 257^2$, this was not needed. However, in this case AMG required a different smoothing operator, the normed residual Gauss-Seidel smoother where Gauss-Seidel is applied to the normal equations instead of the standard Gauss-Seidel smoother [8, 64]. We attribute this to instability of the coarse grid operators.

Table 2.8 contains the average iterations for BICGSTAB to solve the reduced model for 100 randomly selected parameters. We observe that the offline preconditioner is also effective for this problem. In terms of iterations counts, the offline preconditioner performs nearly as well as the online preconditioner as in case 1 of the diffusion equation. The times for offline preconditioned BICGSTAB, reduced direct, and full multigrid methods are shown in Table 2.9. The reduced iterative method is faster than the direct method for $m = 785$. Since decreasing N has the effect of decreasing k for this problem, the iterative methods perform best for $m = 145$, $N = 33^2, 65^2$, corresponding to $k = 372$ and greater.

N	c	2	1	0.5
	m	36	145	785
33^2	k	210	421	798
	None	49.5	45.9	41.7
	Single	8.2	7.0	6.1
	Online	8.3	7.0	6.0
65^2	k	178	372	952
	None	84.5	87.4	86.5
	Single	12.0	10.0	9.0
	Online	12.0	10.0	9.0
129^2	k	138	265	749
	None	122.8	153.0	176.2
	Single	12.9	13.1	13.0
	Online	12.7	13.5	13.0
257^2	k	99	197	686
	None	126.8	234.0	293.8
	Single	14.2	14.4	15.1
	Online	13.9	14.5	15.0

Table 2.8: Average iteration counts for the reduced problem solved using BICGSTAB for the convection-diffusion-reaction problem, $\tau = 10^{-8}$.

2.5 Conclusion

Reduced basis methods are an efficient way to obtain the solution to parameterized partial differential equations for many parameter values. The effectiveness of reduced basis methods depends on the relatively cheap cost of solving the reduced

N	c		2	1	0.5
	m		36	145	785
33^2	k		210	421	798
	Full	AMG	0.0419	0.0428	0.0440
	Reduced	Direct	0.0011	0.0067	0.0400
	Reduced	Iterative	**0.0009**	**0.0019**	**0.0066**
65^2	k		178	372	952
	Full	AMG	0.2188	0.2258	0.2311
	Reduced	Direct	**0.0009**	0.0046	0.0679
	Reduced	Iterative	0.0013	**0.0022**	**0.0148**
129^2	k		138	265	749
	Full	AMG	0.3228	0.3284	0.3271
	Reduced	Direct	**0.0007**	**0.0020**	0.0323
	Reduced	Iterative	0.0012	**0.0020**	**0.0132**
257^2	k		99	197	686
	Full	AMG	1.5330	1.5468	1.5396
	Reduced	Direct	**0.0003**	**0.0010**	0.0234
	Reduced	Iterative	0.0010	0.0016	**0.0140**

Table 2.9: Comparison of time of the BICGSTAB algorithm with full model solved using multigrid and reduced model solved using direct method for the convection-diffusion-reaction problem, $\tau = 10^{-8}$.

problem. This cost depends on the rank of the reduced basis, which depends on quantities such as the number of parameters, m, and the accuracy desired for the reduced solution, τ. We have shown, using two examples, the steady-state diffusion equation and the convection-diffusion-reaction equation, that this cost can be reduced for larger k when iterative methods are used and we have identified the regime of k where, for these problems, iterative methods for the reduced problem become the most effective choice. This has been shown for several preconditioners that are computed offline, and thus do not increase the online cost of solving the reduced model. The exact Schur preconditioner is illustrated to be the most effective preconditioner, while the approximate LSC preconditioner is shown to be an effective preconditioner as well and, in addition, it may be more effective when offline costs matter or when the preconditioner needs to be updated online.

Chapter 3

The discrete empirical interpolation method for the steady-state Navier-Stokes equations

3.1 Introduction

In this chapter, we consider a method of reduced-order modeling for nonlinear problems. A straightforward implementation of the reduced-basis method is only possible for linear problems that have affine dependence on the parameters. Such problems, like those discussed in the previous chapter, have the form $G(u) = 0$ where

$$G(u) = A(\xi)u - b = \left(\sum_{i=1}^{l} \varphi_i(\xi) A_i \right) u - b \tag{3.1}$$

and $\{A_i\}_{i=1}^{l}$ are parameter-independent matrices. Let Q be a matrix of dimension $N \times k$ representing the reduced basis. With this decomposition, the reduced model obtained from a Galerkin condition is $G^r(\hat{u}) = 0$ where

$$G^r(\hat{u}) = Q^T A(\xi) Q \hat{u} - Q^T b = \left(\sum_{i=1}^{l} \varphi_i(\xi)(Q^T A_i Q) \right) \hat{u} - Q^T b\ . \tag{3.2}$$

Computation of the matrices $\{Q^T A_i Q\}$ can be included as part of the offline step. With this precomputation, the online step requires only the summation of the terms in equation (3.2), an $O(lk^2)$ operation, and then the solution of the system of order k. Clearly, this online computation is independent of N, the dimension of the full model.

However, when this approach is applied to a nonlinear problem, the reduced model is not independent of the dimension of the full model. Consider a problem with a nonlinear component $F(u(\xi))$, so the full model is

$$G(u(\xi)) = Au(\xi) + F(u(\xi)) - b = 0 \ . \tag{3.3}$$

The reduced model obtained from the Galerkin projection is

$$G^r(\hat{u}(\xi)) = Q^T A Q \hat{u}(\xi) + Q^T F(Q\hat{u}(\xi)) - Q^T b = 0 \tag{3.4}$$

Although the reduced operator $Q^T F(Q\hat{u}(\xi))$ is a mapping from $\mathbb{R}^k \to \mathbb{R}^k$, any nonlinear solution algorithm (e.g. Picard iteration), requires the evaluation of the operator $F(Q\hat{u}(\xi))$ as well as the multiplication by Q^T. Both computations have costs that depend on N, the dimension of the full model.

The empirical interpolation method [6,36] and its discrete variant, the discrete empirical interpolation method (DEIM) [19] use interpolation to reduce the cost of the online construction in the case of nonlinear operators and/or nonaffine parameter dependence. The premise of these methods is to interpolate the nonlinear operator using a subset of indices from the full model. The interpolation depends on an empirically derived basis that can also be constructed as part of an offline procedure. This ensures that $F(Q\hat{u}(\xi))$ is evaluated only at a relatively small number (n_{deim}) of indices. These values are used in conjunction with a separate basis constructed to approximate the nonlinear operator. The efficiency of this approach also depends on the fact that for all i, $F_i(u(\xi))$ depends on a relatively small, $O(1)$, number of components of u.

Computing the solution of the reduced model for a nonlinear operator, requires a nonlinear iteration based on a linearization strategy, which requires the solution of a reduced linear system at each step. Thus, each iteration has two primary costs, the computation of the Jacobian corresponding to $Q^T F(u(\xi))$ and the solution of the linear system at each step of the nonlinear iteration. The DEIM addresses the first cost, by using an approximation of $Q^T F(u(\xi))$. To address the second cost, one option is to use direct methods to solve the reduced linear systems. In Chapter 2, however, we have seen that iterative methods are effective for solving reduced models of linear operators of a certain size. In this chapter, we extend this approach, using preconditioners that are precomputed in the offline stage, to nonlinear problems solved using the DEIM. We explore this approach using a Picard iteration for the linearization strategy.

We will demonstrate the efficiency of combining the DEIM with an iterative linear solver by computing solutions of the steady-state incompressible Navier-Stokes equations with random viscosity coefficient:

$$\begin{array}{rcllr} -\nabla\cdot\nu(\cdot,\xi)\nabla u(\cdot,\xi)+u(\cdot,\xi)\cdot\nabla u(\cdot,\xi)+\nabla p(\cdot,\xi) & = & f(\cdot,\xi) & \text{in} & D\times\Gamma \\ \nabla\cdot u(\cdot,\xi) & = & 0 & \text{in} & D\times\Gamma \\ u(\cdot,\xi) & = & b(\cdot,\xi) & \text{on} & \partial D\times\Gamma\ , \end{array}$$

where $u(\cdot,\xi)$ is the flow velocity, $p(\cdot,\xi)$ is the scalar pressure, and $b(\cdot,\xi)$ is the Dirichlet boundary condition, and the viscosity coefficient satisfies $\nu(\cdot,\xi)>0$. Models of this type have been used to model the viscosity in multiphase flows [40, 53, 69]. The boundary data $b(\cdot,\xi)$ could also contain uncertainty (although we will not consider such examples here).

An outline of this chapter is as follows. In Section 3.2 we review the details of the discrete empirical interpolation method. In Section 3.3 we introduce the steady-state Navier-Stokes equations with an uncertain viscosity coefficient and describe the full, reduced, and DEIM models for this problem. We present numerical results in Section 3.4 including a comparison of snapshot selection methods for DEIM, a discussion of accuracy of the DEIM. In addition, we discuss a generalization of this approach known as a gappy-POD method [18,32]. Finally, in Section 3.5, we discuss preconditioners and iterative methods for the reduced model generated using DEIM.

3.2 The discrete empirical interpolation method

The discrete empirical interpolation method utilizes an approximation $\bar{F}(u)$ of a nonlinear function $F(u)$ [19]. The key to the accuracy of this method is to select the indices of the discrete PDE that are most important to produce an accurate representation of the nonlinear component of the solution projected on the reduced space; as observed above, the key to efficiency in this algorithm is that each component of the nonlinear function depends only on a few indices of the input variable. The latter requirement is clearly satisfied when the nonlinear function is a PDE discretized using the finite element method [4].

Given the full model defined in equation (3.3), let

$$J_G(u) = A + J_F(u)$$

denote the Jacobian matrix. The Jacobian of the reduced model equation (3.4) is then

$$J_{G^r}(\hat{u}) = Q^T J_G(Q\hat{u})Q = Q^T A Q + Q^T J_F(Q\hat{u})Q \, .$$

Let $u(\xi_1), ..., u(\xi_k)$ denote a set of snapshots obtained from the full model. The reduced basis is constructed to span these snapshots. DEIM requires a separate basis to represent the nonlinear component of the solution. The basis is constructed using snapshots $S = [F(u(\xi_1)), F(u(\xi_2)), ..., F(u(\xi_s))]$ where $s \geq k$. Then, using methods similar to finding the reduced basis, a basis is chosen to approximately span the space spanned by these snapshots. One approach for doing this is to use a proper orthogonal decomposition (POD) of the snapshot matrix S

$$S = \bar{V}\Sigma W^T$$

where the singular values in Σ are sorted in order of decreasing magnitude and $\bar{V}$ and W are orthogonal. For computational efficiency, only the important components are retained so the first n_{deim} columns of $\bar{V}$ define V.

Given the nonlinear basis from V, the DEIM selects indices of F so the interpolated nonlinear component on the range of V in some sense represents a good approximation to the complete set of values of $F(u(\xi))$. Thus, the approximation of the nonlinear operator is

$$\bar{F}(u(\xi)) = V(P^T V)^{-1} P^T F(u(\xi))$$

where P^T extracts rows of $F(u)$ corresponding to the interpolation points from the spatial grid. This approximation satisfies $P^T \bar{F} = P^T F$. To construct P, a greedy procedure is used to minimize the error compared with the full representation of $F(u)$ [19, Algorithm 1]. For each column of V, v_i, the algorithm selects the row index with maximum difference between the column v_i and the approximation of v_i obtained using the DEIM model with nonlinear basis and indices from the first

$i-1$ columns left of V, i.e. $r = v_i - \hat{V}(P^T\hat{V})^{-1}P^T v_i$ where $\hat{V}$ denotes the first $i-1$ columns of V. We present this in Algorithm 2.

Algorithm 2 DEIM [19]

Input: $V = [v_1, ..., v_{n_{deim}}]$, an $N \times n_{deim}$ matrix with columns made up of the left singular vectors from the POD of the nonlinear snapshot matrix S.
Output: P, extracts the indices used for the interpolation.[1]

```
1: $\rho = \text{argmax}(|v_1|)$
2: $\widehat{V} = [v_1]$, $P = [e_\rho]$
3: for $i = 2 : n_{deim}$ do
4:     Solve $(P^T\widehat{V})c = P^T v_i$ for $c$
5:     $r = v_i - \widehat{V}c$
6:     $\rho = \text{argmax}(|r|)$
7:     $\widehat{V} = [\widehat{V}, v_i]$, $P = [P, e_\rho]$
8: end for
```

Incorporating this approximation into the reduced model, equation (3.4), yields

$$\bar{F}^r = Q^T\bar{F}(\tilde{u}) = Q^T V(P^T V)^{-1}P^T F(Q\hat{u}) \ . \tag{3.5}$$

The Jacobian of $\bar{F}^r(\hat{u})$ is

$$J_{\bar{F}^r}(\hat{u}) = Q^T J_{\bar{F}}(Q\hat{u})Q = Q^T V(P^T V)^{-1}P^T J_F(u)Q \ .$$

The construction of nonlinear basis matrix V and the interpolation points are part of the offline computation. Since $L^T = Q^T V(P^T V)^{-1}$ is parameter independent, it too can be computed offline. Therefore, the online computations required are to compute $P^T J_F(u)$ and assemble $L^T(P^T J_F(u))Q$. For $P^T J_F(u)$, we need only to compute the components of $J_F(u)$ that are nonzero at the interpolation points. This is where the assumption that each component of $F(u)$ (and thus $J_F(u)$) depends on

[1]Note that in the implementation of DEIM, the matrix P is not constructed. Instead an index list is used to extract the relevant entries of V.

only a few entries of u is utilized. With a finite element discretization, a component $F_i(u)$ depends on the components u_j for which the intersection of the support of the basis functions have measure that is nonzero. See [4] for additional discussion of this point. The elements that must be tracked in the DEIM computations are referred to as the sample mesh. When the sample mesh is small, the computational cost of assembling $L^T(P^TJ_F(u))Q$ scales not with N but with the number of interpolation points. Therefore, DEIM will decrease the online cost associated with assembling the nonlinear component of the solution.

For the Navier-Stokes equations, the nonlinear component is a function of the velocity. We will discretize the velocity space using biquadratic (Q_2) elements. In this case, an entry in $F_i(u)$ depends on at most nine entries of u. Thus this nonlinearity is amenable to using DEIM. We can use an existing finite element routine for the assembly of the Jacobian using the sample mesh, a subset of the original mesh, as the input.

The accuracy of this approximation is determined primarily by the quality of the nonlinear basis V. This can be seen by considering the error bound

$$||F - \bar{F}||_2 \leq ||(P^TV)^{-1}||_2||(I - VV^T)F||_2$$

which is derived and discussed in more detail in [19, Section 3.2]. There it is shown that the greedy selection of indices in Algorithm 2 limits the growth of $||(P^TV)^{-1}||_2$ as the dimension of V grows. The second term $||(I-VV^T)F||_2$ is the quantity that is determined by the quality of V. Note that if V is taken from the truncated POD of S, the matrix of nonlinear snapshots, then $||(I-VV^T)S||_F^2$ is minimized [4] where

$||\cdot||_F$ is the Frobenius norm ($||X||_F^2 = \sum_i \sum_j |x_{ij}|^2$). So the accuracy of the DEIM approximation depends on two factors. First the number, n_{deim}, of singular vectors kept in the POD. The truncated matrix $V\Sigma_{deim}W_{deim}^T$ is the optimal rank-n_{deim} approximation of S, but a higher rank approximation will improve accuracy of the DEIM model. In fact the error $||(I - VV^T)F||_2$ approaches 0 in the limit as n_{deim} approaches N. The second factor is the quality of the nonlinear snapshots in S. The nonlinear component should be sampled well enough to capture the variations of the nonlinear component throughout the solution space. A comparison of methods for selecting the snapshot set is included in Section 3.4.1.

3.3 Steady-state Navier-Stokes equations

A discrete formulation of the steady-state Navier-Stokes equations (3.5) is to find $\vec{u}_h \in X_E^h$ and $p_h \in M^h$ such that

$$\begin{array}{rcll} (\nu(\cdot,\xi)\nabla\vec{u}_h, \nabla\vec{v}_h) + (\vec{u}_h\cdot\nabla\vec{u}_h, \vec{v}_h) - (p_h, \nabla\cdot\vec{v}_h) & = & (f, \vec{v}_h) & \forall\vec{v}_h \in X_0^h \\ (\nabla\cdot\vec{u}_h, q_h) & = & 0 & \forall q_h \in M^h \end{array}$$

where X_E^h and M^h are finite-dimensional subspaces of the Sobolev spaces; see [30] for details. We will use div-stable Q_2-P_{-1} finite element (biquadratic velocities, piecewise constant discontinuous pressure). Let $\{\phi_1, ..., \phi_{n_u}\}$ represent a basis of Q_2 and $\{\psi_1, ..., \psi_{n_p}\}$ represent a basis of P_{-1}.

We define the following vectors and matrices where $\vec{u}$ and p are vectors of the velocity and pressure coefficients, respectively.

$$\mathbf{z} = \begin{bmatrix}\vec{u} \\ p\end{bmatrix} \tag{3.6}$$

$$[A(\xi)]_{ij} = \int \nu(\xi)\nabla\phi_i : \nabla\phi_j \tag{3.7}$$

$$[B]_{ij} = -\int \psi_i(\nabla \cdot \phi_j) \tag{3.8}$$

$$[N(\vec{u})]_{ij} = \int (\vec{u}_h \cdot \nabla\phi_j) \cdot \phi_i \tag{3.9}$$

$$[\mathbf{f}]_i = (f, \phi_i) \tag{3.10}$$

$$[g(\vec{u})]_i = -(\nabla \cdot \vec{u}_h, \psi_i)\ . \tag{3.11}$$

$$\mathbf{b}(\xi) = \begin{bmatrix} \mathbf{f} - A(\xi)\vec{u}_{bc} \\ g(\vec{u}_{bc}) \end{bmatrix} \tag{3.12}$$

where $\vec{u}_{bc}$ is a vector which interpolates the Dirichlet boundary data $b(\cdot, \xi)$ and is zero everywhere on the interior of the mesh. We denote the velocity solution on the interior of the mesh, $\vec{u}_{in}$, so that $\vec{u} = \vec{u}_{bc} + \vec{u}_{in}$ and $\vec{u}_{in}$ satisfies homogenous Dirichlet boundary conditions. The reduced basis is constructed using snapshots of $\vec{u}_{in}$ so the approximation of the velocity solution generated by the reduced model is of the form $\tilde{u} = \vec{u}_{bc} + Q_u\hat{u}$ where Q_u is a basis spanning velocity snapshots with homogeneous Dirichlet boundary conditions.

3.3.1 Full model

Using this notation, the full model for the Navier-Stokes problem is to find $\mathbf{z}(\xi)$ such that $G(\mathbf{z}(\xi)) = 0$ where

$$G(\mathbf{z}(\xi)) = \begin{bmatrix} A(\xi) & B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} \vec{u} \\ p \end{bmatrix} + \begin{bmatrix} N(\vec{u}) & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \vec{u} \\ p \end{bmatrix} - \begin{bmatrix} \mathbf{f} \\ 0 \end{bmatrix}\ . \tag{3.13}$$

We utilize a Picard iteration to solve the full model, monitoring the norm of the nonlinear residual $G(\mathbf{z}_n(\xi))$ for convergence. The nonlinear Picard iteration to solve this model is described in Algorithm 3.

Algorithm 3 Picard iteration for solving the discrete steady-state Navier-Stokes equation

```
1: The nonlinear iteration is initialized with the solution to a Stokes problem
$$\begin{bmatrix} A(1) & B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} \vec{u}_{in,0} \\ p_0 \end{bmatrix} = \mathbf{b}(1) \ . \qquad (3.14)$$
2: Incorporate the boundary conditions
$$\vec{u}_0 = \vec{u}_{bc} + \vec{u}_{in,0} \ .$$
3: Solve
$$\left( \begin{bmatrix} A(\xi) & B^T \\ B & 0 \end{bmatrix} + \begin{bmatrix} N(\vec{u}_n) & 0 \\ 0 & 0 \end{bmatrix} \right) \begin{bmatrix} \delta\vec{u} \\ \delta p \end{bmatrix} = -G(\mathbf{z}_n) \ . \qquad (3.15)$$
4: Update the solutions
$$\vec{u}_{n+1} = \vec{u}_n + \delta\vec{u}$$
$$p_{n+1} = p_n + \delta p \ .$$
5: Exit when
$$||G(\mathbf{z}_{n+1})||_2 < \delta \, ||\mathbf{b}(\xi)||_2 \ .$$
```

3.3.2 Reduced model

Offline we compute a reduced basis

$$Q = \begin{bmatrix} Q_u & 0 \\ 0 & Q_p \end{bmatrix} ,$$

where Q_u represents the reduced basis of the velocity space and Q_p the reduced basis for the pressure space. We defer the details of this offline construction to Section 3.4.1. For a given Q, the Galerkin reduced model is

$$\begin{aligned} G^r(\mathbf{z}) &= Q^T G(\mathbf{z}) \\ &= \left(Q^T \begin{bmatrix} A(\xi) & B^T \\ B & 0 \end{bmatrix} Q \right) \begin{bmatrix} \hat{u} \\ \hat{p} \end{bmatrix} + \left(Q^T \begin{bmatrix} N(\tilde{u}) & 0 \\ 0 & 0 \end{bmatrix} Q \right) \begin{bmatrix} \hat{u} \\ \hat{p} \end{bmatrix} - Q^T \begin{bmatrix} \mathbf{f} \\ 0 \end{bmatrix} . \end{aligned}$$

Using the nonlinear Picard iteration, the reduced model is described in Algorithm 4.

After the convergence of the Picard iteration determined by $G^r(\tilde{\mathbf{z}}_{n+1})$, we compute the "full" residual: $G(\tilde{\mathbf{z}}_n)$. Note that this residual is computed only once: it is not monitored during the course of the iteration. The full residual indicates how well the reduced model approximates the full solution, so it is used to measure the quality of the reduced model via the error indicator:

$$\eta_\xi = ||G(\tilde{\mathbf{z}}_n(\xi))||_2 / ||\mathbf{b}(\xi)||_2 . \tag{3.16}$$

3.3.3 DEIM model

The DEIM model has the structure of the reduced model but with the nonlinear component F replaced by the approximation $\bar{F}$. First in the offline step, we compute V, P, and $L^T = Q_u^T V(P^T V)^{-1}$. The DEIM model is

$$G^{deim}(\mathbf{z}) = \left(Q^T \begin{bmatrix} A(\xi) & B^T \\ B & 0 \end{bmatrix} Q \right) \begin{bmatrix} \hat{u} \\ \hat{p} \end{bmatrix} + \begin{bmatrix} L^T P^T N(\tilde{u}) Q_u & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \hat{u} \\ \hat{p} \end{bmatrix} - Q^T \begin{bmatrix} \mathbf{f} \\ 0 \end{bmatrix} . \tag{3.17}$$

Algorithm 4 Picard iteration for solving the reduced steady-state Navier-Stokes equations

```
1: Initialize the Picard iteration by solving the reduced Stokes problem
   $$Q^T \begin{bmatrix} A(\xi) & B^T \\ B & 0 \end{bmatrix} Q \begin{bmatrix} \hat{u}_0 \\ \hat{p}_0 \end{bmatrix} = Q^T \mathbf{b}(\xi) .$$
2: Solve the reduced problem for the Picard iteration
   $$\left( Q^T \begin{bmatrix} A(\xi) & B^T \\ B & 0 \end{bmatrix} Q + Q^T \begin{bmatrix} N(\tilde{u}_n) & 0 \\ 0 & 0 \end{bmatrix} Q \right) \begin{bmatrix} \delta\hat{u} \\ \delta\hat{p} \end{bmatrix} = -G^r(\tilde{\mathbf{z}}_n) .$$
   Note that the when the dependence on the parameters is affine, the first term
   in the left hand side can be computed primarily offline as in equation (3.2).
3: Update the reduced solutions
   $$\begin{aligned} \hat{u}_{n+1} &= \hat{u}_n + \delta\hat{u} \\ \hat{p}_{n+1} &= \hat{p}_n + \delta\hat{p} . \end{aligned}$$
4: Update the approximation to the full solution
   $$\begin{aligned} \tilde{u}_{n+1} &= \vec{u}_{bc} + Q_u \hat{u}_{n+1} \\ \tilde{p}_{n+1} &= Q_p \hat{p}_{n+1} . \end{aligned}$$
5: Compute $N(\tilde{u}_{n+1})$.
6: Compute $G^r(\tilde{\mathbf{z}}_{n+1})$.
7: Exit when
   $$||G^r(\tilde{\mathbf{z}}_{n+1})||_2 < \delta \left|\left|Q^T \mathbf{b}(\xi)\right|\right|_2 .$$
```

The model is described in Algorithm 5. Recall from the earlier discussion of DEIM that $P^T N(u)$ is not computed by forming the matrix $N(u)$. Instead, $N(u)$ is assembled only for elements of the sample mesh so that $P^T N(u)$ is accurate. Note that error indicator η_ξ in equation (3.16) depends on $G(\tilde{\mathbf{z}})$. This quantity contains $N(\tilde{u}_n)$ and not $P^T N(\tilde{u}_n)$. Therefore, we must assemble $N(\tilde{u}_n)$ on the entire mesh in order to compute the error indicator. Like the reduced model, this computation is performed only once after the convergence of the nonlinear iteration.

Algorithm 5 DEIM model for the steady-state Navier-Stokes equations

```
1: Initialize the Picard iteration by solving the reduced Stokes problem
       $$Q^T \begin{bmatrix} A(\xi) & B^T \\ B & 0 \end{bmatrix} Q \begin{bmatrix} \hat{u}_0 \\ \hat{p}_0 \end{bmatrix} = Q^T \mathbf{b}(\xi) \ . \qquad (3.18)$$
2: Solve the reduced problem for the Picard iteration
       $$\left( Q^T \begin{bmatrix} A(\xi) & B^T \\ B & 0 \end{bmatrix} Q + \begin{bmatrix} L^T (P^T N(\tilde{u}_n)) Q_u & 0 \\ 0 & 0 \end{bmatrix} \right) \begin{bmatrix} \delta\hat{u} \\ \delta\hat{p} \end{bmatrix} = -G^{deim}(\tilde{\mathbf{z}}_n) \ . \qquad (3.19)$$
   Note that the term on the left can be computed cheaply as in equation (3.2)
   so we only need to update the upper left corner of the matrix as the Picard
   iteration proceeds.
3: Update the reduced solutions
       $$\hat{u}_{n+1} = \hat{u}_n + \delta\hat{u}$$
       $$\hat{p}_{n+1} = \hat{p}_n + \delta\hat{p} \ .$$
4: Update the approximation to the full solution
       $$\tilde{u}_{n+1} = u_{bc} + Q_u \hat{u}_{n+1}$$
       $$\tilde{p}_{n+1} = Q_p \hat{p}_{n+1} \ .$$
5: Compute $P^T N(\tilde{u}_{n+1})$.
6: Compute $G^{deim}(\tilde{\mathbf{z}}_{n+1})$.
7: Exit when
       $$||G^{deim}(\tilde{\mathbf{z}}_{n+1})||_2 < \delta \left|\left| Q^T \mathbf{b}(\xi) \right|\right|_2 \ .$$
```

3.3.4 Inf-sup condition

We turn now to the construction of the reduced basis

$$Q = \begin{bmatrix} Q_u & 0 \\ 0 & Q_p \end{bmatrix} .$$

Given k_s snapshots of the full model, a natural choice is to have the following spaces generated by these snapshots,

$$\begin{array}{rcl} \text{span}(Q_u) & = & \text{span}\{\vec{u}_{in}(\xi^{(1)}), ..., \vec{u}_{in}(\xi^{(k_s)})\} \\ \text{span}(Q_p) & = & \text{span}\{p(\xi^{(1)}), ..., p(\xi^{(k_s)})\} . \end{array} \tag{3.20}$$

However, this choice of basis does not satisfy an inf-sup condition

$$\gamma_R := \min_{0 \neq q_R \in \text{span}(Q_p)} \max_{0 \neq \vec{v}_R \in \text{span}(Q_u)} \frac{(q_R, \nabla \cdot \vec{v}_R)}{|\vec{v}_R|_1 ||q_R||_0} \geq \gamma^* > 0 \tag{3.21}$$

where γ^* is independent of Q_u and Q_p [63]. To address this issue, we follow the enrichment procedure of [60]. For $i = 1, ..., k_s$, let $\vec{r}_h(\cdot, \xi^{(i)})$ be the solution to the Poisson problem

$$(\nabla \vec{r}_h(\cdot, \xi^{(i)}), \nabla \vec{v}_h) = (p_h(\cdot, \xi^{(i)}), \nabla \cdot \vec{v}_h) \ \ \forall \vec{v}_h \in X_0^h , \tag{3.22}$$

and let Q_u of equation (3.20) be augmented by the corresponding discrete solutions $\{\vec{r}(\xi^{(i)})\}$, giving the enriched space

$$\text{span}(Q_u) = \text{span}\{\vec{u}_{in}(\xi^{(1)}), ..., \vec{u}_{in}(\xi^{(k_s)}), \vec{r}(\xi^{(1)}), ..., \vec{r}(\xi^{(k_s)})\} .$$

This choice of enriching functions satisfy [60]

$$\vec{r}_h(\cdot, \xi^{(i)}) = \arg \sup_{\vec{v}_h \in X_0^h} \frac{(p_h(\cdot, \xi^{(i)}), \nabla \cdot \vec{v}_h)}{|\vec{v}_h|_1} , \tag{3.23}$$

and thus γ_R defined for the enriched velocity space, span(Q_u), together with span(Q_p), satisfies the inf-sup condition

$$\gamma_R \geq \gamma_h := \min_{0 \neq q_h \in M_h} \max_{0 \neq \vec{v}_h \in X_0^h} \frac{(q_h, \nabla \cdot \vec{v}_h)}{|\vec{v}_h|_1 ||q_h||_0} . \tag{3.24}$$

3.4 Experiments

We consider the steady-state Navier-Stokes equations (3.5) for driven cavity flow posed on a square domain $D = (-1, 1) \times (-1, 1)$. The lid, the top boundary ($y = 1$), has velocity profile

$$u_x = 1 - x^4, \;\; u_y = 0 \, ,$$

whereas the remaining boundaries have no-slip boundary conditions $\vec{u} = (0, 0)^T$. We use source term $f(\xi) = 0$. The n by n discretization of the domain D leads to $n_u = (n + 1)^2$ points in the velocity discretization and $n_p = 3(n/2)^2$ points in the pressure discretization.

To define the uncertain viscosity, divide the domain D into $m = n_d \times n_d$ subdomains as seen in Figure 3.1. The viscosity is taken to be constant and random on each subdomain, $\nu(\xi) = \xi_i$. The random parameter vector, $\xi = [\xi_1, ..., \xi_m]^T \in \Gamma$, is comprised of uniform random variables such that $\xi_i \in \Gamma_i = [0.01, 1]$ for each i. Therefore, the Reynolds number, $\mathcal{R} = 2/\nu$, will vary between 2 and 200 for this problem and within the stable regime for the steady problem [30].

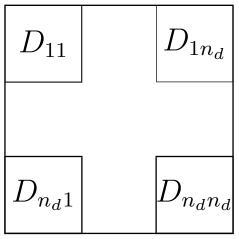


Figure 3.1: Flow domain with piecewise random coefficients for viscosity.

The implementation uses IFISS to generate the finite element matrices for the full model [68]. The matrices are then imported into Python and the full, reduced, and DEIM models are constructed and solved using a Python implementation run on an Intel 2.7 GHz i5 processor and 8 GB of RAM. The full model is solved using the method described in equation (3.25) using sparse direct methods implemented in the UMFPACK suite [22].

For the driven cavity flow, the linear systems in the full model equations (3.14) and (3.15) are singular [10]. This issue is addressed by augmenting the matrix in (3.14), as

$$\begin{bmatrix} A(1) & B^T & 0 \\ B & 0 & 0 \\ 0 & \bar{p} & 0 \end{bmatrix} \begin{bmatrix} \vec{u}_{in} \\ p \\ z \end{bmatrix} = \begin{bmatrix} \mathbf{b}(1) \\ 0 \end{bmatrix}, \qquad (3.25)$$

where $\bar{p}$ is a vector corresponding to the element areas of the pressure elements. This removes the singularity by adding a constraint via a Lagrange multiplier that the average pressure of the solution is zero [67]. The solutions $\vec{u}_{in}$ and p satisfy the solution of the Stokes system. The same constraint is added to the systems in equation (3.15).

3.4.1 Construction of Q and V

We now describe the methodology used to compute the reduced bases, Q_u and Q_p, and the nonlinear basis V. The description of the construction of Q_u and Q_p is presented in Algorithm 6. The reduced bases Q_u and Q_p are constructed using random sampling of n_{trial} samples of Γ, denoted Γ_{trial}. The bases are constructed so that all samples $\xi \in \Gamma_{trial}$ have a residual indicator, η_ξ, less than a tolerance, τ. The procedure begins with single snapshot $\mathbf{z}(\xi^{(0)})$ where $\xi^{(0)} = E(\xi)$. The

Algorithm 6 Construction of reduced basis Q via random sampling, construction of nonlinear basis V

Cost: n_{trial} reduced problems and k full problems.

```
 1: ▷ Compute the reduced basis and the nonlinear snapshots
 2: Solve the full problem $G(\mathbf{z}(\xi^{(0)})) = 0$ to tolerance $\delta$ for $\mathbf{z}_n(\xi^{(0)})$.
 3: Compute the enriched velocity, $\vec{r}(\xi^{(0)})$.
 4: Initialize $Q_u = [\vec{u}_{in,n}(\xi^{(0)}), \vec{r}(\xi^{(0)})]$ and $Q_p = [p_n(\xi^{(0)})]$.
 5: Save the nonlinear component of the solution $S = [N(\vec{u}_n)\vec{u}_n]$.
 6: for $i = 1 : n_{trial}$ do
 7:     Randomly select $\xi^{(i)}$.
 8:     Solve reduced model $G^r(\tilde{\mathbf{z}}(\xi^{(i)})) = 0$ to tolerance $\delta$ and compute the residual
        indicator $\eta_{\xi^{(i)}}$.
 9:     if $\eta_{\xi^{(i)}} > \tau$ then
10:         Solve full model $G(\mathbf{z}(\xi^{(i)})) = 0$.
11:         Compute the enriched velocity, $\vec{r}(\xi^{(i)})$.
12:         Add $\vec{u}_{in,n}(\xi^{(i)})$ and $\vec{r}(\xi^{(i)})$ to $Q_u$ and $p_n(\xi^{(i)})$ to $Q_p$ using modified Gram-
            Schmidt.
13:         Add the nonlinear component of the solution to the matrix of nonlinear
            snapshots, $S = [S, N(\vec{u}_n)\vec{u}_n]$.
14:     end if
15: end for
16: ▷ Compute the nonlinear basis
17: Compute the POD of the nonlinear snapshot matrix:
```

$$S = \bar{V} \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_{k_s} \end{bmatrix} W^T.$$

```
18: Choose $n_{deim}$ so that the singular values to satisfy
```

$$\frac{\sum_{i=1}^{n_{deim}} \sigma_i^2}{\sum_{i=1}^{k_s} \sigma_i^2} > \epsilon \tag{3.26}$$

```
    where $k_s$ is the number of columns in $S$.
19: Define $V = \bar{V}[:, 1 : n_{deim}]$.
20: Compute $P$ using Algorithm 2, with input $V$.
```

bases are initialized using this snapshot, such that $Q_u = [\vec{u}_{in,n}(\xi^{(0)}), \vec{r}(\xi^{(0)})]$ and $Q_p = [p_n(\xi^{(0)})]$. Then for each sample of Γ_{trial}, the reduced-order model is solved with the current bases Q_u and Q_p. The quality of the reduced solution produced by this reduced-order model can be evaluated using the error indicator η_ξ defined in equation (3.16). If η_ξ is smaller than the tolerance τ, the computation proceeds to the next sample. When the error indicator exceeds the tolerance, we solve the full model and the new snapshots, $u_{in,n}(\xi)$ and $p_n(\xi)$, and the enriched velocity $\vec{r}(\xi)$ are used to augment Q_u and Q_p. The experiments in this chapter use $\tau = 10^{-4}$ and in most cases use $n_{trial} = 2000$ parameters to produce bases Q_u and Q_p.

An alternative method to random sampling is greedy sampling which produces a basis of quasi-optimal dimension [9, 13]. This means that for a basis constructed using greedy sampling the maximum error differs from the best error by an exponential factor, where the best possible error is defined by the Kolmogorov n-width. In a comparison described in [28], the random sampling strategy produced a reduced basis that was never more than 10% larger than that produced by a greedy algorithm for several benchmark problems. The two algorithms both use a tolerance to construct the reduced basis, so the resulting reduced models have similar accuracy. The computational cost (in CPU time) of the random sampling strategy is *significantly* lower. Since our concern in this study is online strategies for reducing the cost of the reduced model, we use the random sampling strategy for the offline computation and remark that the online solution strategies of this study can be used for a reduced basis computed using any method.

We turn now to the methodology for determining the nonlinear basis V. In

Section 3.2, it was shown that that the choice of the nonlinear basis, V, is important to the accuracy of the DEIM model. The DEIM uses a POD approach for constructing the nonlinear basis. This POD has a two inputs, S, the set of nonlinear snapshots and n_{deim}, the number of vectors after truncation. Algorithm 6 describes one method for choosing the snapshots S that are input to the POD. We will compare three strategies for sampling S.

1. **Full**(n_{trial}). This method is most similar to the method used to generate the nonlinear basis in [19]. The matrix of nonlinear snapshots, S, is computed from the full solution at every random sample (i.e. $\{N(\vec{u}(\xi^{(i)}))\vec{u}(\xi^{(i)})\}_{i=1}^{n_{trial}}$). Even though this is part of the offline step, the cost of this method, solving n_{trial} full problems, can be quite high.

2. **Full**(k_s). This is the sampling strategy included in Algorithm 6. It saves the nonlinear component only when the full model is solved for augmenting the reduced basis, Q. Therefore the snapshot set S contains k_s snapshots.

3. **Mixed**(n_{trial}). The final approach aims to mimic the Full(n_{trial}) method with less offline work. This method generates a nonlinear snapshot for each of the n_{trial} random samples using full solutions when they are available (from full solution used for augmenting the reduced basis) and reduced solutions when they are not. As the reduced basis is constructed, when the solution to the full problem is not needed (i.e. when $\eta_{\xi^{(i)}} < \tau$) for the reduced basis, use the reduced solution $\tilde{u}(\xi^{(i)})$ to generate the nonlinear snapshot $N(\tilde{u}(\xi^{(i)}))\tilde{u}(\xi^{(i)})$, where $\tilde{u}(\xi^{(i)}) = \vec{u}_{bc} + Q_u\hat{u}(\xi^{(i)})$ and Q_u is the basis at this value of i. Thus S

contains n_{trial} snapshots, but it is constructed using only k_s full model solves.

Figure 3.2 compares the performance of the three methods for generating S when Algorithm 6 is used to generate Q_u and Q_p. For each S, we take the SVD and truncate with varying number of vectors, n_{deim}, and plot the average of the residuals of the DEIM solution for 100 samples. The average residual for the reduced model without DEIM is also shown. We see that as n_{deim} increases, the residual of the DEIM models approach the residual that is obtained without using DEIM. In addition we see that all three methods perform similarly. Thus, the Mixed(n_{trial}) approach provides accurate nonlinear snapshots with fewer full solutions than the Full(n_{trial}) method. For smaller n_{deim}, the Full(k_s) method performs similarly to the other methods. The disadvantage of this method is that the maximum number of DEIM vectors corresponds to the number, k_s, of full solutions needed to generate Q_u and Q_p, while the maximum number of DEIM vectors for the other methods is n_{trial}. Thus, the best residual obtained with this method (when $n_{deim} = k_s$) is higher than the best residual obtained with the other two methods. However, for simplicity we use Full(k_s) for the remainder of this study, and remark that since the Mixed(n_{trial}) has similar offline costs it can be used to improve accuracy, if necessary.

3.4.2 Online component - DEIM model versus reduced model

In Section 3.2, we presented analytic bounds for how accurately the DEIM approximates the nonlinear component of the model. To examine how the approximation affects the accuracy of the reduced model, we will compare the error indicators

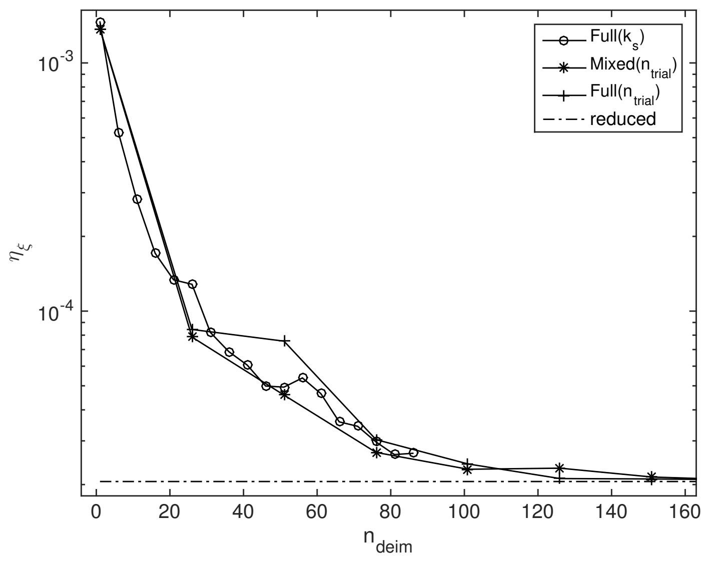


Figure 3.2: A comparison of methods to generate nonlinear snapshots for the DEIM method. DEIM residual versus n_{deim} averaged for $n_s = 100$ samples. $n = 32$, $m = 4$, $\tau = 10^{-4}$, $k = 306$, n_{deim} varies. Q_u, Q_p are generated using the Algorithm 6.

of DEIM and the reduced model without DEIM. We perform the following offline and online computations.

1. Offline: Use Algorithm 6 with input $\tau = 10^{-4}$ and $n_{trial} = 2000$ and $\epsilon = 0.99$ to generate the reduced bases Q_u, Q_p, V and the indices P.

2. Online: Solve the problem using the full model, reduced model without DEIM, and the reduced model with DEIM for $n_s = 10$ parameters.

Table 3.1 presents the result of this study where the three models are solved using direct methods. The method with the lowest online computational time is in bold. The times presented for the reduced and DEIM models are the CPU time spent in the online computation for the nonlinear iteration only and do not include assembly time or residual computation time. The condition $\delta = 10^{-8}$ is used to determine convergence of the nonlinear iteration.

The results demonstrate the tradeoff between accuracy and time for the three models. For example, the case of $m = 16$ and $n = 65$, the spatial dimension and parameter dimension are large enough that the DEIM model is fastest. If lower residuals are needed, we can improve accuracy in the DEIM model by either increasing ϵ (which has the effect of increasing n_{deim}) or improving the accuracy of the reduced model. The accuracy of the reduced model is improved by choosing a stricter tolerance τ during the offline computation. It is important to note that the accuracy of the DEIM solution is limited by the accuracy of the reduced solution. Past a certain point, increasing n_{deim} will provide little improvement in the residual. Thus, the best way to improve accuracy in the DEIM model is to reduce τ. By choosing a stricter tolerance for the reduced model, the size of the reduced basis k increases, but the reduced and DEIM solutions are significantly more accurate. These two ways to improve accuracy will both increase online time. Increasing n_{deim} increases the assembly cost and increasing k increases the solution cost of the linear systems. Depending on the problem at hand, the benefit of each approach may change. For this problem, we have found that the solution time is less costly than the assembly time and so decreasing τ in Algorithm 6 is the most efficient way to improve the accuracy of the DEIM model.

In Table 3.1, the results for $n = 32$ and $m \geq 25$ are not shown. For these problems, the number of snapshots required to construct the reduced model, k_s exceeds the size of the pressure space $n_p = 3(n/2)^2 = 768$. This means that the number of snapshots required for the accuracy of the velocity is higher than the number of degrees of freedom in the full discretized pressure space. Therefore, the spatial discretization is not fine enough for reduced-order modeling to be necessary. For $n = 32$, $m = 16$ the full solution is not much slower than the DEIM model. We would expect the full solution to be faster for $m \geq 25$.

For the case where $n = 128$, the cost of the offline construction is significant. First, the full solutions require over 2 minutes of CPU time. For $m = 36$, 1013 full solutions and 2000 reduced solutions were required. The cost of each full solve is 132 seconds and the costs of the reduced solves are as high as 98.1 seconds. The computation time for the assembly of the reduced models are not presented in Table 3.1 and are also high. Since Q is changing during the offline stage, the assembly process cannot be made independent of N. The offline computation for $m = 49$ took approximately five days.

n	m	4		9		16		25		36		49	
32	k	306		942		1485							
	n_{deim}	4		8		9							
		time	res	time	res	time	res	time	res	time	res	time	res
	Full	1.11		1.16		1.06							
	Reduced	0.18	1.00E-05	1.21	3.81E-05	3.00	4.25E-05						
	DEIM	**0.05**	4.18E-04	**0.37**	6.97E-04	**1.02**	6.78E-04						
64	k	273		825		1503		2394		3339		4455	
	n_{deim}	4		7		12		16		21		25	
		time	res	time	res	time	res	time	res	time	res	time	res
	Full	11.0		10.8		10.2		11.3		10.1		**10.3**	
	Reduced	0.48	8.11E-06	2.53	2.93E-05	7.33	4.10E-05	20.5	6.50E-05	39.5	4.96E-05	76.2	8.99E-05
	DEIM	**0.07**	1.41E-04	**0.27**	3.30E-04	**1.08**	2.92E-04	**4.40**	3.26E-04	**9.54**	2.78E-04	20.7	3.47E-04
128	k	237		732		1383		2109		3039		4083	
	n_{deim}	4		9		14		17		23		30	
		time	res	time	res	time	res	time	res	time	res	time	res
	Full	135		141		147		155		132		148	
	Reduced	1.62	1.13E-05	7.25	1.47E-05	23.8	2.85E-05	56.7	5.735E-05	98.1	5.142E-05	191	7.159E-05
	DEIM	**0.09**	8.27E-05	**0.39**	7.71E-05	**1.12**	1.02E-04	**3.59**	1.772E-04	**7.11**	1.553E-04	**15.7**	1.559E-04

Table 3.1: Accuracy and time for Full, Reduced, and DEIM models for $\tau = 10^{-4}$ and $\epsilon = 0.99$.

Figure 3.3 illustrates the tradeoff between accuracy and time for the DEIM. The top plot compares the error indicators for an average $n_s = 10$ parameters and the bottom plot shows the CPU time for the two methods. While the cost of the

DEIM does increase with the number of vectors n_{deim}, we reach similar accuracy as for the reduced model at a much lower cost. We also see that the cost of increasing n_{deim} is small since the maximum considered here ($n_{deim} = 96$) is significantly smaller than $N = 1089$.

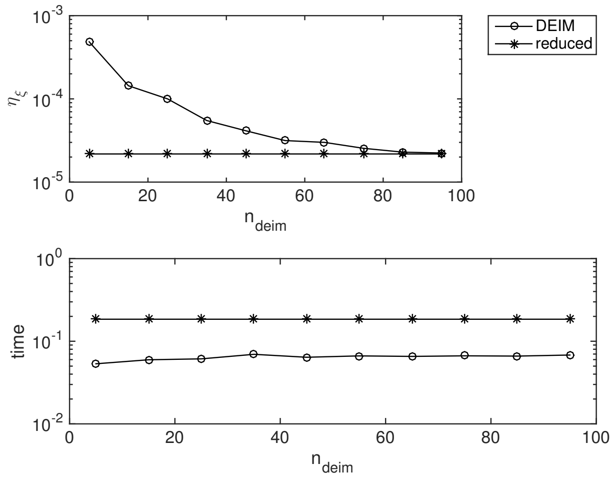


Figure 3.3: Top: Error indicator for DEIM model versus n_{deim}. Bottom: CPU time to solve using DEIM direct versus n_{deim}. For $n = 32$, $m = 4$, $\tau = 10^{-4}$, $k = 306$. Averaged over $n_s = 10$ samples.

3.4.2.1 Gappy POD

Another way to increase the accuracy of the reduced model is to increase the number of interpolation points in the approximation, while keeping the number of basis vectors fixed. This alternative to DEIM for selecting the indices is the so-called gappy POD method [32]. This method allows the number or rows selected by P^T to exceed the number of columns of V.

The approximation of the function using gappy POD looks similar to DEIM, but replaces the inverse of P^TV with the Moore-Penrose pseudoinverse denoted

$(P^TV)^\dagger$ [18]

$$\hat{F}(u) = V(P^TV)^\dagger P^T F(u) \; . \tag{3.27}$$

To apply this inverse we compute $P^T F(u)$ and solve the least squares problem

$$\alpha = \arg\min_{\hat{\alpha}} ||P^T V\hat{\alpha} - P^T F(u)||_2 \; ,$$

which leads to the approximation $\hat{F}(u) = V\alpha$. Like $(P^TV)^{-1}$, the pseudoinverse can be precomputed, in this case using the SVD of $(P^TV) = U\Sigma W^T$,

$$(P^TV)^\dagger = W\Sigma^\dagger U^T$$

where $\Sigma^\dagger$ is the transpose of Σ with the (nonzero) diagonal elements inverted [52].

With this approximation, we turn to the index selection method, which is described in Algorithm 7. Given V where the number of columns is chosen so that the condition (3.26) is satisfied, we require a method to determine the selection of the row indices that will lead to an accurate representation of the nonlinear component. The approach in [18] is an extension of the greedy algorithm used for DEIM (Algorithm 2). The gappy version of algorithm takes as an input the number of grid points and the basis vectors. It simply chooses additional indices per basis vector where the indices correspond to the maximum index of the difference of the basis vector and its projection via the gappy POD model. Recall that in DEIM, the index associated with vector v_i is chosen to maximize $|v_i - V(P^TV)^{-1}P^Tv_i|$. This extension of the algorithm takes additional indices which maximize $|v_i - V(P^TV)^\dagger P^Tv_i|$ (where P and the projection of v_i are updated between selections of the indices).

To compare the accuracy of this method with DEIM, we use Algorithm 6 to compute DEIM and modify line 20 to use Algorithm 7 with $n_g = 2n_{deim}$ for a range

Algorithm 7 Index selection using gappy POD [18]

Input: n_g number of indices to choose, $V = [v_1, ..., v_{n_v}]$, an $N \times n_v$ matrix with columns made up of the left singular vectors from the POD of the nonlinear snapshot matrix S.
Output: P, extracts the indices used for the interpolation.

```
 1: $n_b = 1$, $n_{it} = \min(n_v, n_g)$ $n_{c,\min} = \left\lfloor \frac{n_v}{n_{it}} \right\rfloor$ $n_{a,\min} = \left\lfloor \frac{n_g}{n_v} \right\rfloor$
 2: for $i = 1, ..., n_{it}$ do
 3:     $n_c = n_{c,\min}$, $n_a = n_{a,\min}$
 4:     if $i <= (n_v \mod n_{it})$ then $n_c = n_c + 1$
 5:     end if
 6:     if $i <= (n_g \mod n_v)$ then $n_a = n_a + 1$
 7:     end if
 8:     if $i == 1$ then
 9:         $r = \sum_{q=1}^{n_c} v_q^2$
10:         for $j = 1, ..., n_a$ do $\rho_j = \text{argmax}(r)$, $r[\rho_j] = 0$
11:         end for
12:         $P = [e_{\rho_1}, ..., e_{\rho_{n_a}}]$, $\widehat{V} = [v_1, ..., v_{n_c}]$
13:     else
14:         for $q = 1, ..., n_c$ do
15:             $\alpha = \min_{\hat{\alpha}} ||P^T \widehat{V} \hat{\alpha} - P^T v_{n_b+q}||_2$
16:             $R_q = v_{n_b+q} - \widehat{V}\alpha$
17:         end for
18:         $r = \sum_{q=1}^{n_c} R_q^2$
19:         for $j = 1, .., n_a$ do
20:             $\rho_j = \text{argmax}(r)$, $P = [P, e_{\rho_j}]$
21:             for $q = 1, ..., n_c$ do
22:                 $\alpha = \min_{\hat{\alpha}} ||P^T \widehat{V} \hat{\alpha} - P^T v_{n_b+q}||_2$
23:                 $R_q = v_{n_b+q} - \widehat{V}\alpha$
24:             end for
25:             $r = \sum_{q=1}^{n_c} R_q^2$
26:         end for
27:         $\widehat{V} = [\widehat{V}, v_{n_b+1}, ..., v_{n_b+n_c}]$, $n_b = n_b + n_c$
28:     end if
29: end for
```

of values of n_{deim}. We use both methods to approximate the nonlinear component and solve the resulting models. We present the error indicators for both methods as a function of n_{deim} in Figure 3.4. It is evident that for smaller number of basis vectors the gappy POD provides additional accuracy. Since S is generated using the Full(k_s) described in Section 3.4.1, no additional accuracy is gained for the DEIM method when $n_{deim} > 102$. However, for larger number of basis vectors, the additional accuracy provided by gappy POD is small. Thus, the gappy POD method can be used to improve the accuracy when the number of basis vectors is limited.

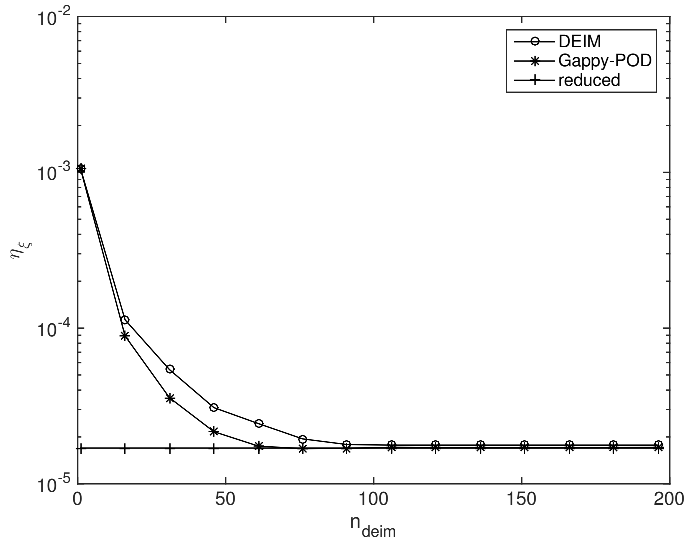


Figure 3.4: Average error indicator as a function of basis vectors for reduced, DEIM, and gappy POD methods. For $n = 32$, $m = 4$, $\tau = 10^{-4}$, $k = 306$, n_{deim} varies, and $n_g = 2n_{deim}$. Averaged over $n_s = 100$ samples.

3.5 Iterative methods

We have seen that the DEIM and gappy POD method generate reduced-order models that produce solutions as accurate as the reduced solution for the steady-state Navier-Stokes system. In addition, Table 3.1 and Figure 3.3 illustrate that as expected, the DEIM model significantly decreases the online time spent constructing the nonlinear component of the reduced model. Since $Q_u^T N(\vec{u}) Q_u$ has been replaced by a cheap approximation, $L^T P^T N(\vec{u}) Q_u$, the remaining cost of the nonlinear iteration in the DEIM is the linear system solve in line 2 of Algorithm 5. The cost of this computation depends on the rank of the reduced basis k. Note the order of the Jacobian matrix is $k = 3k_s$ where k_s is the number of snapshots used to construct the reduced basis. The size of the reduced basis depends on the properties of the problem, for example the number of parameters or the desired level of accuracy. Solving the assembled linear systems in the DEIM and reduced models using direct methods costs $O(k^3)$. The cost of solving the full model could be as small as $O(N)$ for sparse systems where multigrid methods can be utilized. So it can happen that k is much less than N, but k^3 is larger than N. A motivating example is the case of $n = 64$, $m = 49$ in Table 3.1, where the cost (in CPU time) of solving the full model is half the cost of solving the DEIM model using direct methods. Thus, we could consider as an alternative iterative linear methods. Since iterative methods cost $O(k^2 p)$ where p is the number of iterations required for convergence of the iterative method, there are values of k where, if p is small enough, iterative methods will be preferable to direct methods. In this section, we discuss the use of iterative methods based on preconditioned Krylov subspace methods to improve the efficiency of the DEIM model.

For iterative methods to be efficient for such problems, effective preconditioners are needed. The construction of preconditioners can be a nontrivial cost. One reason that iterative solution methods are appealing to use within the offline-online paradigm is that the construction cost of the preconditioner can be moved to the offline component of the computation. Since the preconditioners may depend on the parameter, ξ, the parameter used to construct the preconditioner will impact the performance of the preconditioner. We consider two approaches for choosing this parameter.

- Offline: Use the mean of the parameter space, $\xi^{(0)}$, to generate the preconditioner.
- Online: Construct the preconditioner with the same parameter ξ as the problem we are currently solving. This is not meant to be used in practice since the cost of constructing the preconditioner is high, but it provides insight concerning a lower bound on the iteration count that can be achieved using the offline preconditioner.

There is a third blended option which we do not consider for this problem, where a small number of preconditioners can be computed offline and selected online. This approach may be effective for certain types of problems for example in combination with domain decomposition methods [5]. Similar preconditioners have been considered in the context of stochastic Galerkin methods where preconditioners based on the mean parameter were effective for the steady-state Navier-Stokes equations with uncertainty [59].

We consider two preconditioners of the DEIM model, the *exact Stokes* preconditioner and the *exact* preconditioner.

1. The exact Stokes preconditioner is the inverse of the matrix used for the re-

duced Stokes solve in equation (3.18),

$$M_r = Q^T \begin{bmatrix} A(\xi) & B^T \\ B & 0 \end{bmatrix} Q \,. \tag{3.28}$$

Clearly for the online parameter, the reduced Stokes solve will converge in one iteration.

2. The exact preconditioner uses the converged solution of the full model $\vec{u}_n$ as the input for $P^T N(\vec{u}_n)$ and uses the linear system from the DEIM model from equation 3.19

$$M_r = Q^T \begin{bmatrix} A(\xi) & B^T \\ B & 0 \end{bmatrix} Q + \begin{bmatrix} L^T P^T N(\vec{u}_n) Q_u & 0 \\ 0 & 0 \end{bmatrix} \,. \tag{3.29}$$

3.5.1 Results

For these experiments, we solve the steady-state Navier-Stokes equations for the driven cavity flow problem using the full model, reduced model, and the DEIM model. For the DEIM model the linear systems are solved using both direct and iterative methods.

The offline step construction is described in Algorithm 6; we use $\tau = 10^{-4}$ and $n_{trial} = 2000$. The algorithm chooses k_s snapshots and produces Q_u of rank $2k_s$ and Q_p of rank k_s yielding reduced models of rank $k = 3k_s$. The online experiments are run for $n_s = 10$ random parameters. The average number of iterations required for the convergence of the linear systems is presented in Table 3.2 and the average time for the entire nonlinear solve of each model is presented in Table 3.3. The nonlinear solve time includes the time to compute N or $P^T N$, but not the time for assembly of the linear component of the model nor the computation time for η_ξ in the case of

n	m	4	9	16	25	36	49
32	k	306	942	1485			
	n_{deim}	4	8	9			
	Offline Exact Stokes	11.3	16.2	19.8			
	Online Exact Stokes	2.0	2.3	2.3			
	Offline Exact	11.5	16.1	20.5			
	Online Exact	1.8	1.9	1.9			
64	k	273	825	1503	2394	3339	4455
	n_{deim}	4	7	12	16	21	25
	Offline Exact Stokes	10.3	13.9	16.9	17.6	19.9	23.3
	Online Exact Stokes	2.1	2.1	2.3	2.4	2.2	2.4
	Offline Exact	10.5	13.6	16.5	17.3	19.9	24.0
	Online Exact	1.7	1.8	1.9	2.0	1.9	2.0
128	k	237	732	1383	2109	3039	4083
	n_{deim}	4	9	14	17	23	30
	Offline Exact Stokes	8.8	16.4	21.0	17.8	19.9	25.4
	Online Exact Stokes	1.9	2.2	2.5	2.3	2.2	2.4
	Offline Exact	8.9	16.5	20.5	17.9	20.1	25.1
	Online Exact	1.8	1.8	2.1	2.1	1.9	2.1

Table 3.2: Average iteration count of preconditioned `bicgstab` for solving equation (3.19) for $n_s = 10$ parameters. For these experiments, n_{deim} is chosen such that $\epsilon = 0.99$ in equation (3.26).

the reduced and DEIM problems. The iterative methods presented in this table use offline preconditioners. The method with the lowest online CPU time is boldface. The nonlinear iterations are run to tolerance $\delta = 10^{-8}$ and the `bicgstab` method for a reduced matrix, $Q^T\mathcal{A}Q$, stops when the solution $x^{(i)}$ satisfies

$$\frac{||r - Q^T\mathcal{A}Qx^{(i)}||}{||r||} < 10^{-9} \ .$$

Table 3.2 illustrates that the offline preconditioners using the mean parameter perform well compared to the versions that use the exact parameter. In Table 3.3 we compare these offline parameters with the direct DEIM method and determine that for large enough k the iterative methods are faster than direct methods. We note for the $m = 9$ problems the direct methods are slightly faster while for $m = 16$

n	m		4	9	16	25	36	49
32	k		306	942	1485			
	n_{deim}		4	8	9			
	Full	Direct	1.11	1.16	1.06			
	Reduced	Direct	0.18	1.21	3.00			
	DEIM	Direct	**0.05**	**0.37**	1.02			
	DEIM	Exact stokes	**0.05**	0.42	**0.95**			
	DEIM	Exact	**0.05**	0.44	1.03			
64	k		273	825	1503	2394	3339	4455
	n_{deim}		4	7	12	16	21	25
	Full	Direct	11.0	10.8	10.2	11.3	10.1	10.3
	Reduced	Direct	0.48	2.53	7.33	20.5	39.5	76.2
	DEIM	Direct	**0.07**	**0.27**	1.08	4.40	9.54	20.7
	DEIM	Exact stokes	**0.07**	0.29	**0.86**	2.29	4.62	9.04
	DEIM	Exact	0.08	0.30	0.87	**2.27**	**4.39**	**8.98**
128	k		237	732	1383	2109	3039	4083
	n_{deim}		4	9	14	17	23	30
	Full	Direct	135	141	147	155	132	148
	Reduced	Direct	1.62	7.25	23.8	56.7	98.1	191
	DEIM	Direct	**0.09**	**0.39**	1.12	3.59	7.11	15.7
	DEIM	Exact stokes	**0.09**	0.45	1.10	**2.40**	**3.89**	8.74
	DEIM	Exact	**0.09**	0.45	**1.08**	2.45	3.91	**8.62**

Table 3.3: Average time for the entire nonlinear solve $n_s = 10$ parameters with $\epsilon = 0.99$.

n	m		4	9	16	25	36	49
32	k		306	942	1485			
	n_{deim}		4	8	9			
	DEIM	Exact stokes	0.04	0.36	1.02			
	DEIM	Exact	1.01	1.42	2.06			
64	k		273	825	1503	2394	3339	4455
	n_{deim}		4	7	12	16	21	25
	DEIM	Exact stokes	0.10	0.85	2.38	6.51	13.7	26.7
	DEIM	Exact	8.96	10.0	12.6	15.7	22.8	35.7
128	k		237	732	1383	2109	3039	4083
	n_{deim}		4	9	14	17	23	30
	DEIM	Exact stokes	0.29	1.96	6.84	17.9	33.0	68.2
	DEIM	Exact	116	132	129	143	147	193

Table 3.4: CPU time to construct the (offline) preconditioner

the iterative methods are faster for all values of n. We also note that the fastest DEIM method is faster than the full model for all cases. Returning to the motivating example of $n = 64$ and $m = 49$, the DEIM iterative method is faster than the full model, whereas the DEIM direct method performs twice as slowly as the full model. Thus, utilizing iterative methods has increase the range of k, where reduced-order modeling is practical. Recall that $n = 32$, $m \geq 25$ have more snapshots than the size of the pressure discretization. Preconditioners derived from variants for the saddle point matrices (like those discussed in Chapter 2) were less effective than the exact Stokes and exact preconditioner for this problem.

Table 3.4 presents the (offline) cost of constructing the preconditioners. Since the costs of the exact preconditioner uses the full solution, the cost of constructing that preconditioner scales with the costs of the full solution. However, the cost the exact Stokes preconditioner is significantly smaller and performs similarly in the online computations. Thus, the exact Stokes preconditioner is an efficient option for both offline and online components of this problem.

3.6 Conclusion

We have shown that the discrete interpolation method is effective for solving the steady-state Navier-Stokes equations. This approach produces a reduced-order model that is essentially as accurate as a naive implementation of a reduced basis method without incurring online costs of order N. In cases where the dimension of the reduced basis is larger, performance of the DEIM is improved through the use of preconditioned iterative methods to solve the linear systems arising at each nonlinear Picard iteration. This is achieved using the mean parameter to construct preconditioners. These preconditioners are effective for preconditioning the reduced model in the entire parameter space.

Chapter 4

Krylov subspace recycling via reduced-order modeling

4.1 Introduction

There are applications of parameterized PDEs where the offline-online paradigm may not be suitable, for example when offline costs matter or when linear systems must be solved sequentially. For example, in the case of fatigue and fracture modeling, implicit solvers are required and therefore the solution to a previous system is needed to generate the next linear system and right-hand side [55]. Another example requiring sequential solves is the sequence of linear systems produced with Newton-type methods for nonlinear and optimization problems. In addition, there may be problems where the offline costs are so high that performing more than a few evaluations of the full model is infeasible.

In these situations, either the parameter space cannot be sampled or there are too few snapshots of the solution for the resulting reduced models to meet the required accuracy requirements. In these settings, reduced models can be constructed from information available from previous solves and used to accelerate the solution of the full models. One way to construct reduced models is through Krylov subspace recycling. The premise of Krylov subspace recycling is, in order to solve a (full) model using a Krylov iterative solver, information from all previous linear system solves can be used to accelerate the solution of the current system. Methods of this type were originally developed to restart solution algorithms applied to systems with multiple right-hand sides and a common matrix [46, 47] (e.g. restarted GMRES); they have since been adapted to accelerate the solution of multiple systems in cases where the matrices vary, where the Krylov vectors generated during the course of a given iterative solve are saved to accelerate the solution of the subsequent systems [55, 62, 66].

For memory and cost reasons, it tends to be too expensive to save all Krylov vectors from all previous solves, and thus some form of compression or truncation is required. One common method, known as deflation, retains approximate eigenvectors corresponding to eigenvalues that most inhibit the convergence rate of the iterative method [66]. Our contention is that other strategies for reduced-order modeling can be used in this context, where the reduced-order model is constructed to approximate the recycled space (which contains the solution space of all previous solutions). The POD-augmented Krylov method [16] uses a weighted proper orthogonal decomposition (POD) to construct the reduced basis. The basis is used to solve a small reduced problem (using a mix of direct and iterative methods) and then an augmented conjugate gradient method is used to solve the full problem iteratively to the desired accuracy. The augmented conjugate gradient method constructs search directions orthogonal to the augmenting space, which in this case is the reduced-order model [66]. One advantage of the POD approach to compression is that the reduced space can be tailored for convergence in a goal-oriented norm, when the quantity of interest is a linear function of the output.

The goal of this chapter is to develop efficient methods for solving an ordered sequence of linear systems

$$A_j \bar{x}_j = b_j,\ j = 1, ..., n_s\ , \tag{4.1}$$

where A_j is symmetric positive definite (SPD) and sparse. Krylov-subspace methods are efficient iterative methods for solving such systems. When each system is solved using the preconditioned conjugate gradient method to a tolerance δ i.e.,

$$\frac{||b_j - A_j x_j||_2}{||b_j||_2} < \delta\ ,$$

and the approximate solution x_j can be written as

$$x_j = \sum_{i=1}^{n_j} \alpha_i p_i$$

where the vectors, p_i, referred to as *search directions*, are A_j-orthogonal to each other (meaning $p_i^T A_j p_k = 0$ for $i \neq k$). Denote

$$V_j = [p_0, ..., p_{n_j}] \tag{4.2}$$

whose columns are known to constitute a basis of the Krylov space $\mathcal{K}^{n_j}(A_j, b_j)$ [64].

In the sequence of linear systems, A_j is changing from step to step. However, we anticipate that A_j is not significantly different from A_{j+1}, especially when such matrices are coming from nonlinear solvers or are parameterized by time. The jth computed solution, x_j, lies in the range of V_j. Therefore, it is reasonable to assume that x_{j+1}, a solution for the $(j+1)$st system, can be well approximated in the range of V_j. A recycling algorithm has two main components: an approximation of the solution on a recycled space (which, in the simplest case, is the space spanned by the columns of V_j), and a Krylov method to search for a correction to the approximation in the space that is orthogonal, in some sense, to the recycled space. Various recycling methods are defined by the choice of recycle space.

In this chapter, we study the POD-augmented Krylov method for solving sequences of positive semi-definite systems coming from a nonlinear iteration and we compare its performance with deflation for two examples. First, we review Krylov subspace recycling methods in Section 4.2. In Section 4.3, we outline the three-stage framework used for Krylov-subspace recycling, and in Section 4.4 we present a comparison of the compression methods for two examples.

4.2 Krylov subspace recycling

A Krylov subspace recycling method is defined by a recycle space and a Krylov method which constructs a Krylov subspace orthogonal to the recycle space. The first component of the method is to find an approximate solution on the recycle space, which provides an initial guess of the solution for the Krylov solver. Many methods utilize a recycle space spanning the vectors generating from solving a previous system in the sequence. We could recycle all vectors or some subset of these vectors. We will refer to methods to choose this subset of vectors as compression methods. Initially, Krylov subspace recycling was used to improve a method known as restarted GMRES where the number of vectors that can be stored in a Krylov basis is limited. We will begin this discussion of Krylov subspace recycling with a review of GMRES, a description of the process of restarting, and a review of the recycling methods that came about to improve this method. We describe newer methods which adapt these techniques for solving sequences of linear systems. We conclude with a presentation of the augmented conjugate gradient method, a Krylov subspace recycling method for symmetric positive-definite systems.

4.2.1 Generalized minimum residual method

The generalized minimum residual method (GMRES) [65] constructs a Krylov subspace using an Arnoldi iteration and finds the solution which minimizes the 2-norm of the residual over the Krylov space. The Arnoldi iteration generates Krylov vectors $V_m = [v_1, ..., v_m]$ which satisfy

$$AV_m = V_{m+1}\bar{H}_m \tag{4.3}$$

where $\bar{H}_m$ is an $(m+1) \times m$ upper-Hessenberg matrix. In this iteration, the Krylov vectors are orthogonalized with respect to the 2-norm so this relation satisfies

$$V_m^T A V_m = H_m$$

where H_m is the submatrix of $\bar{H}_{m+1}$ with the last row deleted. GMRES requires the solution of a least squares problem to find the solution of the form $x_m = x_0 + V_m y_m$ where

$$y_m = \arg\min_{y} ||\beta e_1 - \bar{H}_m y||_2 \tag{4.4}$$

and $\beta = ||r_0||_2$. The solution of this problem is obtained using Givens rotation matrices to transform H_m into an upper triangular matrix, R [64]. Denote

$$P\bar{H}_m = R \tag{4.5}$$

where P is a product of orthogonal matrices representing Givens rotations

$$P = Q_1 Q_2 ... Q_m \ .$$

Then the solution $y_m = \arg\min_y ||\beta P e_1 - R y||_2$ can be found by solving the upper triangular system obtained by deleting the last row from the right hand side $\beta P e_1$ and matrix R [64].

The computational cost of a GMRES iteration grows with m since each new vector v_{m+1} must be orthogonalized against all previous vectors, the columns of V_m. Thus it may happen that because of restrictions of memory or computational cost, the iteration count cannot exceed a particular value of m. When this is the case, restarted GMRES [65] is used. This method performs m iterations of GMRES and then takes x_m as an initial approximation for a new, restarted version of the GMRES algorithm. The term for the m iterations between restarts is a *cycle*. Using restarted GMRES ensures that storage requirements and the cost of orthogonalization remain low. However, this approach may lead to poor convergence and can even stagnate [46]. One choice of Krylov subspace recycling method selects a subset of the current Krylov basis vectors to retain at the next restart. These vectors are chosen to improve the rate of convergence over that obtained using restarted GMRES and with smaller computational cost than GMRES without restarting.

For certain classes of matrices, the convergence of GMRES is dependent on the distribution of eigenvalues of the coefficient matrix [46]. If the recycled vector was an exact eigenvector of A, then the corresponding eigenvalue can be eliminated from the spectrum and the convergence of the system now depends on this "deflated" spectrum. This is the idea of deflation. While the eigenvectors of A are unavailable during the Arnoldi iteration, an approximation of the eigenvectors is available. This approximation is discussed in the next section. Particularly, keeping the approximate eigenvectors corresponding to the smallest eigenvalues has been shown to improve convergence [46, 47].

4.2.2 Deflation

One way to approximate the eigenvectors of a matrix A is to use the Ritz vectors. For any subspace $\mathcal{S}$, we define the Ritz value θ and Ritz vector $y \in \mathcal{S}$ of A with respect to a subspace $\mathcal{S}$ such that [55]

$$Ay - \theta y \perp w \ \ \forall w \in \mathcal{S} \ . \tag{4.6}$$

Given a basis S of $\mathcal{S}$, the Ritz vector $y = S\hat{y}$ satisfies

$$S^T(AS\hat{y} - \theta S^T S\hat{y}) = 0$$

Therefore the Ritz values and vectors can be obtained by solving the generalized eigenvalue problem

$$S^T AS\hat{y} = \theta S^T S\hat{y} \ . \tag{4.7}$$

When $\mathcal{S} = \mathcal{K}^{(m)}(A, r)$, a Krylov subspace, the Ritz values are available cheaply. For example, in the case of the Arnoldi iteration $S = V_m$ so that equation (4.7) is

$$V_m^T AV_m\hat{y} = \theta V_m^T V_m\hat{y}$$

$$H_m\hat{y} = \theta\hat{y} \ .$$

So the Ritz values of A correspond to the eigenvalues of H_m and the Ritz vectors are $y = V_m \hat{y}$.

The Ritz values with respect to a Krylov space are known to be good approximations for the eigenvalues of large magnitude [45]. This suggests that the reciprocals of the Ritz values of A^{-1} would be good approximations for the eigenvalues of small magnitude of A. A widely-used technique for obtaining Ritz values of A^{-1} using information obtained from the Krylov space of A is to consider the Ritz values of A^{-1} with respect to the space $A\mathcal{S}$

$$A^{-1}\tilde{y} - \frac{1}{\tilde{\theta}}\tilde{y} \perp w \ \ \forall w \in A\mathcal{S} \ . \tag{4.8}$$

These Ritz values, $\tilde{\theta}$, are known as the harmonic Ritz values of A. The corresponding Ritz vector of A^{-1} is $\tilde{y} \in A\mathcal{S}$. However, the harmonic Ritz vectors are instead defined to be $y \in \mathcal{S}$. This choice is made because y is both available and known to be a better approximation of the corresponding eigenvector than $\tilde{y}$ [45]. It is a better approximation because y is obtained from $\tilde{y}$ with one application of an inverse iteration. Computation of harmonic Ritz vectors corresponding to the smallest harmonic Ritz values of A, from equation (4.8) entails solving the eigenvalue problem

$$S^T A^T A S \hat{y} = \tilde{\theta} S^T A^T S \hat{y} \tag{4.9}$$

where S is a matrix whose columns span $\mathcal{S}$. Keeping the harmonic Ritz vectors $S\hat{y}$ corresponding to the smallest harmonic Ritz values, $\tilde{\theta}$, is known as *deflation*. Deflation is a popular technique for Krylov subspace recycling and is used to choose the recycled space in methods such as GMRES with deflated restarting (GMRES-DR) [47], generalized conjugate residual (GCR) method with orthogonalization and deflated restarting (GCRO-DR) [55], and the deflated conjugate gradient method [66].

For the Arnoldi iteration, the matrix on the left-hand side of equation (4.9) is

$$\begin{aligned} V_m^T A^T A V_m &= \bar{H}_m^T V_{m+1}^T V_{m+1} \bar{H}_m \\ &= \bar{H}_m^T \bar{H}_m \\ &= R^T R \end{aligned}$$

where R is defined in equation (4.5). The matrix on the right-hand side of equation (4.9) corresponds H_m^T [46]. Therefore, the computation of the harmonic Ritz vectors during an Arnoldi iteration requires the solution of an $O(m)$ generalized eigenvalue problem where the matrices R and H_m have been computed during the course of the Arnoldi iteration.

4.2.3 Modification for sequences of linear systems

Now consider a sequence of systems in equation (4.1). The GCRO-DR method uses deflated restarting within the framework of the method GCRO (generalized conjugate residual method with orthogonalization) [23]. In the case where the matrix does not vary from step to step (i.e., only the right-hand side changes), GCRO-DR and GMRES-DR are algebraically equivalent [55]. GCRO-DR was introduced as an alternative to GMRES-DR because it can be used for Krylov subspace recycling in the case where the matrix is changing. GMRES-DR cannot be adapted since the harmonic Ritz vectors in GMRES-DR do not generate a Krylov subspace for another matrix.

In contrast, the harmonic Ritz vectors of GCRO-DR can be adapted when the matrix changes. The GCRO-DR method [55] maintains two bases U_m and C_k which satisfy $AU_k = C_k$ and $C_k^* C_k = I$. It maintains orthogonality by performing the Arnoldi iteration $(I - C_k C_k^*)AV_{m-k} = V_{m-k+1}\bar{H}_{m-k}$. The solution is found on the range of U_k such that $x_m = x_0 + U_k C_k^* r_0$. At the end of each cycle, the solution is found by solving a least squares problem and the recycled space is determined by solving a generalized eigenvalue problem for the harmonic Ritz vectors.

GCRO-DR typically uses several cycles for a given A_j, updating U_k and C_k every m iterations. Once the solution to the jth system is obtained, the current bases U_k and C_k can be modified for a new matrix A_{j+1} such that

$$\begin{array}{rcl} [Q, R] & = & \mathtt{qr}(A_j U_k^{old}) \\ C_k^{new} & = & Q \\ U_k^{new} & = & U_k^{old} R^{-1} \ . \end{array}$$

It is easy to check that $A_{j+1}U_k^{new} = C_k^{new}$ and $C_k^{new*}C_k^{new} = I$ [55]. This approach to updating the recycled space for A_{j+1} can also be used for the GCROT method (GCR with optimal truncation). GCROT chooses a recycle space to minimize the difference of the 2-norm of the residual obtained with the truncated space and the residual obtained by keeping full space [24].

Rey and Risler [62] use a different technique for Krylov subspace recycling for sequences of linear systems with varying coefficient matrices. The methods accelerate the solution for symmetric positive definite systems solved using the conjugate gradient method. Here the recycled vectors are the columns of V_j from the previous system in the sequence as defined in equation (4.2) and are weighted using the components of the solution. For a system with conjugate gradient solution we have

$x_j = \sum_{i=1}^{n_j} \alpha_i p_i$ where p_i are the columns of V_j. The recycled vectors are $V_j \Lambda_j^{-1/2}$ where $\Lambda_j = \mathrm{diag}(\alpha_1, ..., \alpha_{n_j})$. The subsequent solution method solves the following reduced model iteratively:

$$\Lambda_j^{-1/2} V_j^T A_{j+1} V_j \Lambda_j^{-1/2} \hat{x} = \Lambda_j^{-1/2} V_j^T b_{j+1} \; . \tag{4.10}$$

The weights Λ_j ensure that this reduced problem is well conditioned. After the reduced solve, the augmented CG method is used to obtain the solution to the desired accuracy. Note that the methodology uses the previous system (iterative reuse of Krylov subspaces, known as IRKS) or keeps all previous systems (generalized iterative reuse of Krylov subspaces, known as GIRKS). Therefore this method uses recycling between systems with no compression.

4.2.4 Orthogonalization methods

The review above has mostly focused on the choice of recycling space. The other aspect of Krylov-subspace recycling is how to use the space to accelerate convergence for the solution of the next system. This is done by ensuring that the new directions are orthogonal to these recycled vectors, where "orthogonality" depends on the algorithm used. In GMRES-DR, the Arnoldi algorithm maintains orthogonality to the recycled space with respect to the 2-norm. For symmetric positive-definite systems, a good choice is the augmented conjugate gradient method. This method ensures that the new directions are orthogonal in the A-norm to the recycling space. This method is employed in this chapter and is discussed in more detail in the following section.

4.2.4.1 Augmented conjugate gradient method

For SPD systems, we can use the augmented conjugate gradient method for Krylov recycling. For now let us denote a general recycled space by $\mathcal{P}$ which spans the columns of a matrix P. The augmented CG method first finds the solution in the range of P, then uses a conjugate gradient iteration to find the solution where the new search directions are A-orthogonal to the recycled space $\mathcal{P}$. Sometimes this method is referred to in the literature as the deflated conjugate gradient method [66] (though we restrict that usage to the case with a specific choice of P– discussed in Section 4.3.1.1). The augmented CG method is presented in Algorithm 8 with the subscripts j removed for neatness.

This method is a variant of preconditioned CG, modified so that the new directions are constructed to be orthogonal to the old space. It is easy to check that $P^T AV = 0$. Note the initial solve in step 5 contains the same system as the Galerkin projection of a reduced model for a linear operator seen in Chapter 2. In the case where P contains the Krylov vectors from all previous solutions, the recycle space contains the snapshot space used in the reduced basis method. (The solution snapshots for each already-solved system can be obtained as a linear combination of Krylov vectors.) Since the approach in this chapter is to use as much information as possible from a few systems, we will use a recycle space of Krylov vectors from all previous solutions. Obviously this would generate a space of much larger dimension than a snapshot space. When this is the case, it becomes necessary to compress this space and retain only the important components. We will now describe this

Algorithm 8 Augmented conjugate gradient method [66]

```
1: Inputs: Linear system $A$, $b$
2:     $M$: preconditioner of $A$
3:     $P$: the augmenting space
4:     $\delta$: tolerance such that the solution satisfies
           $$\frac{||b - Ax||}{||b||} < \delta$$
5: Solve $P^T AP\hat{x} = P^T b$ for $\hat{x}$.          ▷ Reduced problem
6: Compute $r_0 = b - AP\hat{x}$.
7: Compute $z_0 = M^{-1} r_0$.
8: Solve $P^T AP\mu_0 = P^T Az_0$ for $\mu_0$.          ▷ Same matrix as step 5
9: Set $p_0 = z_0 - P\mu_0$.
10: for $k = 1, 2, ...,$ do
11:     $\alpha_{k-1} = \frac{r_{k-1}^T z_{k-1}}{p_{k-1}^T Ap_{k-1}}$
12:     $x_k = x_{k-1} + \alpha_{k-1} p_{k-1}$
13:     $r_k = r_{k-1} - \alpha_{k-1} Ap_{k-1}$
14:     if $||r_k||/||b|| < \delta$ then
15:         Exit.
16:     end if
17:     $z_k = M^{-1} r_k$
18:     Solve $P^T AP\mu_k = P^T Az_k$ for $\mu_k$.     ▷ Same matrix as step 5
19:     $\beta_{k-1} = \frac{r_k^T z_k}{r_{k-1}^T z_{k-1}}$
20:     $p_k = \beta_{k-1} p_{k-1} + z_k - P\mu_k$
21: end for
22: Outputs: $x$, $V = [p_0, p_1, ..., p_n]$
```

framework in more detail.

4.3 Three-stage framework

The three-stage framework for solving a sequence of symmetric positive-definite linear systems is outlined in Algorithm 9. It is a variation of the augmented conjugate gradient method where the "reduced" problem $P^TAP\hat{x} = P^Tb$ is solved with a hybrid of direct and iterative solution methods. It is an extension of the framework developed for sequences of related linear systems in [16].

The algorithm uses the following ingredients:

1. A stage-1 partial basis of the augmenting subspace W_j with rank k_1. These vectors should be the most important recycled vectors.

2. A stage-2 basis of augmenting subspace $Y_j = [W_j, Z_j]$ with rank $k = k_1 + k_2$. This is a basis for the full recycled subspace and contains the vectors from stage 1.

Stage 1 uses a direct solve to obtain the solution on a subspace of the reduced space range(W_j). The idea is that this space will contain the most important components in the reduced space. This space should be chosen so that a good approximation to the solution is obtained, but the space should also be small enough so that this computation is inexpensive. Note that using a direct solve requires the construction of $W_j^TA_jW_j$, so W_j should have only a few columns. Since we have already computed the solution on a subspace of the reduced space, in stage 2 we solve over the remaining space range(Z_j) using augmented CG to maintaining orthogonality to the stage-1 space W_j. We outline this method in Algorithm 10. We save the search directions from the reduced iterative solve $\hat{Y}_j = [\hat{p}_0, ..., \hat{p}_{n_j}]$. The final stage solves the full problem $A_j x_j = b_j$ using augmented CG with augmenting space $\tilde{Y} = [W_j, Y_j \hat{Y}_j]$, where the solution of the reduced problem $(b_j - W_j w - Y_j \hat{Y}_j y)$ is a starting iterate. Essentially this framework replaces line 5 of Algorithm 8 with stages 1 and 2.

At the end of stage 3 we have a solution for the jth system that meets the required tolerance δ. We save the search directions generated during stage 3, V_j to Y_j. There is an option to compress the search directions. Usually this compression is done when the number of vectors in Y_j exceeds some threshold, n_{max}. There are many choices for compression method. In the next section, we will consider two compression techniques, deflation and POD.

Note that the traditional augmented conjugate gradient algorithm is recovered in the case without stage 2. This framework is related to the algorithms of [62] in the case that stage 1 and compression are ignored.

Remark: The stage-2 solve is an iterative method, suggesting that a preconditioner may be required. However, when W_j and Z_j come from CG directions, we have $V_j^T A_j V_j = D_j$ where D_j is a diagonal matrix $\text{diag}(\alpha_1, ..., \alpha_{n_j})$ with $\{\alpha_j\}$ defined on line 11 of Algorithm 8. As suggested by [62], recycling $V_j D_j^{-1/2}$ ensures that the stage-2 solve is of the form $D_j^{-1/2} V_j^T A_j V_j D_j^{-1/2} = I$ when the matrix, A_j, is invariant. Since the next solve uses A_{j+1}, if A_{j+1} is close to A_j, then $V_j^T A_{j+1} V_j \approx I$ and the system is naturally well conditioned.

Algorithm 9 A three-stage framework for Krylov subspace recycling

```
1: Inputs: Sequence of linear systems $A_j$, $b_j$ for $j = 1, ..., n_s$
2:     $M_j$: preconditioner for the $j$th full model
3:     $\delta$: tolerance such that the final solution $x_j$ satisfies
           $$\frac{||b_j - A_j x_j||}{||b_j||} < \delta$$
4:     $n_{max}$: the maximum size of the augmenting space
5: Solve the $j = 1$ system using preconditioned conjugate gradient method.
6: Save search directions $V_1 = [p_0, ..., p_{n_{it}}]$ to generate subspaces $W_2$, $Y_2$.
7: for $j = 2 : n_s$ do
8:     Stage 1: Solve the reduced model $W_j^T A_j W_j w_j = W_j^T b_j$ using a direct method.
9:     Stage 2: Solve the reduced model $Y_j^T A_j Y_j y_j = Y_j^T (b_j - W_j w_j)$ using augmented conjugate gradient method (maintaining orthogonality to space $W_j$) to tolerance $\epsilon$. Save the search directions generated $\hat{Y}_j = [\hat{p}_0, \hat{p}_1, ..., \hat{p}_{n_{red}}]$. See Algorithm 10.
10:    Stage 3: Solve the full model $A_j x_j = b_j - W_j w_j - Y_j \hat{Y}_j y_j$ iteratively to tolerance $\delta$ using preconditioned augmented conjugate gradient method with augmenting space $\tilde{Y}_j = [W_j, Y_j \hat{Y}_j]$. Save the search directions generated $V_j = [p_0, ..., p_{n_{it}}]$ and coefficients $D = \text{diag}(\alpha_1, ..., \alpha_{n_{it}})$.
11:    Add new search directions $V_j D_j^{-1/2}$ to $W_{j+1}$ and/or $Z_{j+1}$.
12:    if rank($W_{j+1}$) + rank($Z_{j+1}$) $\geq n_{max}$ then
13:        Compress keeping $k_1$ vectors in $W_{j+1} = \Phi_1$ and $k_2$ vectors in $Z_{j+1} = \Phi_2$.
14:    end if
15: end for
```

Algorithm 10 Stage 2: Iterative solve of reduced problem [16, Algorithm 3]

```
1: Goal: Solve $[W,Z]^T A[W,Z]y = [W,Z]^T(b - AWw)$. Note that $Y = [W,Z]$.
   Vectors with hats lie on the reduced space, range$(Y)$, and vectors without hats
   lie on the full space.
2: Inputs:
3:     $x_0 = Ww$: Approximation to the full solution from stage 1
4:     $W$, $Z$, precomputed $AW$ from stage 1
5:     $R$: a Cholesky factor such that $R^T R = W^T AW$ from stage 1
6:     $\epsilon$: tolerance such that the output $y$ satisfies
           $$\frac{||Y^T(b - AYy)||}{||Y^T b||} < \epsilon$$
7: $r_0 = b - Ax_0$
8: $\hat{r}_2^{(0)} = Z^T r_0$
9: $k = 0$
10: while $||\hat{r}_2^{(k)}||_2/||Y^T b||_2 > \epsilon$ do
11:     $\hat{p}_2^{(k)} = \hat{r}_2^{(k)}$
12:     Solve $R^T R\hat{p}_1^{(k)} = -(AW)^T Z\hat{p}_2^{(k)}$.
13:     $\hat{p}^{(k)} = \begin{bmatrix} \hat{p}_1^{(k)} \\ \hat{p}_2^{(k)} \end{bmatrix}$
14:     for $j = 0, ..., k-1$ do
15:         $\hat{p}^{(k)} = \hat{p}^{(k)} - \hat{p}^{(j)}(\sigma^{(j)} z^{(j)T} \hat{p}_2^{(k)})$
16:     end for
17:     $p^{(k)} = [W,Z]\hat{p}^{(k)}$, $v^{(k)} = Ap^{(k)}$
18:     $\sigma^{(k)} = 1/(p^{(k)T} v^{(k)})$, $z^{(k)} = Z^T v^{(k)}$, $\alpha^{(k)} = \sigma(\hat{r}^{(k)T}\hat{p}^{(k)})$
19:     $x_{k+1} = x_k + \alpha^{(k)} p^{(k)}$, $\hat{r}_2^{(k+1)} = \hat{r}_2^{(k)} - \alpha^{(k)} z^{(k)}$
20:     $k = k + 1$
21: end while
22: Outputs: $n_{red} = k$, $x_{n_{red}}$, $\tilde{Y} = [W,Z]\hat{Y} = [p^{(1)}, ..., p^{(n_{red})}]$, $A\tilde{Y} = [v^{(1)}, ..., v^{(n_{red})}]$,
    $\Sigma = \text{diag}(\sigma^{(1)}, ..., \sigma^{(n_{red})})$
```

4.3.1 Compression methods

One important component in this framework is compression. Two different compression methods are considered: (1) deflation using the harmonic Ritz vectors, and (2) weighted proper orthogonal decomposition (POD), a method that finds the key components of the Krylov subspace. These methods will be compared to the method that uses no compression, i.e., where all vectors are kept ($Y_j = [V_1, V_2, ..., V_{j-1}]$).

4.3.1.1 Deflation

As discussed in Section 4.2, the deflation method [47] approximates the eigenvectors corresponding to the eigenvalues of smallest magnitude. When using this method to solve a sequence of linear systems, there is an assumption that the spectra of the matrices do not vary greatly. Several methods that use deflation were discussed in Section 4.2. These methods can be viewed through the lens of the three-stage framework discussed in Section 4.3; the only difference is that the deflation-based methods compress during the course of a single system solve whereas the methodology used here compresses only every few systems.

Compression via deflation is performed with respect to a space, $\mathcal{S} = \text{range}(S)$, where S is a matrix consisting of search directions and previously compressed vectors. Deflation retains the harmonic Ritz vectors $S\psi_i$ where ψ_i are the eigenvectors of a generalized eigenvalue problem originally defined in equation (4.9) and simplified here for a symmetric positive definite matrix

$$S^T A_j^T A_j S \psi_i = \lambda_i S^T A_j S \psi_i \ . \tag{4.11}$$

If the first k eigenvectors in $\Psi = [\psi_1, \psi_2, ..., \psi_k]$ are used, then the compressed space is $\Phi = S\Psi$. Note that this method depends on A_j, the matrix from the most recently solved system.

In the literature [47, 55], deflation is performed in cycles on a single problem, and the final space at the end is deflated and then adapted somehow (depending on the method) for the next system. This mimics the style of restarted GMRES where the matrix is not changing. We adapt deflation to this framework where $S = [V_1, V_2, ..., V_{j-1}]$ with $\{V_i\}$ containing the Krylov directions from the stage-3 solves. When we return to the compression step again we may have $S = [\Phi_k, V_{s+1}, V_{s+2}, ..., V_{j-1}]$ where Φ_k is the $n \times k$ matrix generated from the last compression via deflation.

4.3.1.2 Weighted proper orthogonal decomposition

This approach is similar to techniques of standard reduced-order modeling. Given a subspace, $\mathcal{S} = \text{range}(S)$ where S is a rank-n_w matrix of Krylov vectors, the goal is to compress without losing important information. We use a proper orthogonal decomposition (POD) to identify the essential components of the space. Often POD uses the 2-norm to measure the importance of a component; in this case, the POD is equivalent to taking the singular value decomposition of S. However, in the three-stage framework it is advantageous to instead measure the importance of a vector using the A-norm. The proper orthogonal decomposition with respect to the A-norm is equivalent to the eigendecomposition of S^TAS with corresponding eigenvalue problem

$$S^TAS\psi_i = \lambda_i\psi_i \; .$$

We consider a weighted POD with weights $\Gamma = \text{diag}(\gamma_1, ..., \gamma_{n_w})$, that convey the importance of the vectors of S. This adds flexibility to the POD and emphasizes the important search directions as inputs into the POD. Given S and corresponding weights Γ, compute the POD with the A-norm by solving the eigenvalue problem

$$\Gamma^T S^T A_j S\Gamma\psi_i = \lambda_i\psi_i \; . \tag{4.12}$$

The POD produces the recycled space, $\Phi = S\Gamma\Psi\Lambda^{-1/2}$ where the columns of $\Psi = [\psi_1, .., \psi_k]$ are the eigenvectors corresponding to the largest eigenvalues $\Lambda = \text{diag}(\lambda_1, ..., \lambda_k)$. Define Φ_1 and Φ_2 such that $\Phi = [\Phi_1, \Phi_2]$ where

$$\begin{aligned} \Phi_1 &= S\Gamma[\psi_1, .., \psi_{k_1}] \begin{bmatrix} \lambda_1^{-1/2} & & \\ & \ddots & \\ & & \lambda_{k_1}^{-1/2} \end{bmatrix} \\ \Phi_2 &= S\Gamma[\psi_{k_1+1}, ..., \psi_{k_1+k_2}] \begin{bmatrix} \lambda_{k_1+1}^{-1/2} & & \\ & \ddots & \\ & & \lambda_{k_1+k_2}^{-1/2} \end{bmatrix} \; . \end{aligned}$$

We refer to the range space of Φ as $\mathcal{P}(k, S\Gamma, A)$ where $k = k_1 + k_2$, $S\Gamma$ are the input vectors, and A defines the norm to compute the POD.

The use of the A-norm in this framework is a good choice because the reduced problem with the compressed space leads to a naturally well-conditioned iterative solve in stage 2. Note that

$$\Phi^TA\Phi = \Lambda^{-1/2}\Psi^T(\Gamma^TS^TAS\Gamma)\Psi\Lambda^{-1/2} = \Lambda^{-1/2}\Psi^T\Psi\Lambda\Psi^T\Psi\Lambda^{-1/2} = I \; .$$

Since the compression is performed with A_j, $\Phi^T A_{j+1} \Phi$ will be approximately I.

POD is a good choice for compression in this framework [16]. The subspace generated by a POD,

$$\mathcal{P}(k, [s_1, ..., s_{n_w}], A) \ ,$$

is optimal in the sense that

$$\mathcal{P}(k, [s_1, ..., s_{n_w}], A) = \arg \min_{\mathcal{Y} \in \mathcal{G}(k,N)} \sqrt{\sum_{i=1}^{n_w} ||(I - P_{\mathcal{Y}}^A) s_i||_A^2} \tag{4.13}$$

where the Grassman manifold, $\mathcal{G}(k, N)$ is the set of all k-dimensional linear subspaces of $\mathbb{R}^N$ [2]. Given an augmenting space $\mathcal{Y}$ with basis defined by the columns of Y, the error in the reduced solution is

$$e_{\mathcal{Y}}^A(x) = ||(I - P_{\mathcal{Y}}^A) x||_A \tag{4.14}$$

where $P_{\mathcal{Y}}^A$ is the projector such that $\tilde{x} = P_{\mathcal{Y}}^A x$ is the approximation of the full solution from the Galerkin reduced model (i.e. $\tilde{x} = Y\hat{x}$ where $\hat{x}$ is the solution of $Y^T A Y \hat{x} = Y^T b$).

Clearly the ideal choice is $\mathcal{Y} = \text{span}(x)$ where x is the exact solution. Instead consider an estimate of the solution as a linear combination of the recycled directions $S = [s_1, ..., s_{n_w}]$

$$x_{est} = \sum_{i=1}^{n_w} \gamma_i s_i \ .$$

Thus the error is approximately

$$e_{\mathcal{Y}}^A(x_{est}) = ||(I - P_{\mathcal{Y}}^A) \sum_{i=1}^{n_w} \gamma_i s_i||_A \ .$$

The error can be bounded [16] first using the triangle inequality

$$e_{\mathcal{Y}}^A(x_{est}) \leq \sum_{i=1}^{n_w} ||(I - P_{\mathcal{Y}}^A) \gamma_i s_i||_A$$

and using the norm equivalence relation $||x||_1 \leq n^{1/2}||x||_2$

$$e^A_{\mathcal{Y}}(x_{est}) \leq n_w^{1/2} \sqrt{\sum_{i=1}^{n_w} ||(I - P^A_{\mathcal{Y}})\gamma_i s_i||^2_A} \ . \tag{4.15}$$

From equation (4.13), the choice of $\mathcal{P}(k, S\Gamma, A)$ minimizes the upper bound of the error in equation (4.15) [16].

The weights should be chosen so that x_{est} expressed as a linear combination of the search directions in S is as close to the actual solution as possible. The specific choice of weights used is discussed further in Section 4.4.4. Those weights assume the systems are sequenced so that the most relevant information for the $(j+1)$st system is the information from the jth system.

4.3.1.3 Goal-oriented proper orthogonal decomposition

In some applications the quantity of interest may be a linear function of the output of the form $z = Cx$. In this case the weighted proper orthogonal decomposition can be tailored to ensure fast convergence of this quantity. Given some $C \in \mathbb{R}^{N\times d}$ we define the goal-oriented norm, $||x||_{C^TC} = \sqrt{(Cx, Cx)}$. By replacing the A-norm in the POD with the C^TC-norm, we obtain the following eigenvalue problem

$$\Gamma^T S^T C^T C S \Gamma \psi_i = \lambda_i \psi_i \ . \tag{4.16}$$

The recycled spaces Φ_1 and Φ_2 are defined using $\{\psi_i\}$ and $\{\lambda_i\}$ as for the A-norm. Similar analysis of the POD suggests that the output z would converge quickly using these recycled spaces [17].

4.3.2 Inner iterative variation of the augmented conjugate gradient method

Consider the orthogonalization steps in stage 3 (lines 8 and 18) of Algorithm 8. Note that such a step is efficient for the stage-1 component because the Cholesky decomposition of W^TAW is computed in stage 1 and thus only triangular solves are needed for orthogonalization. Even cheaper is the stage-2 component $Y\hat{Y}$, where the diagonal matrix for $(\hat{Y}^TY^TAY\hat{Y})^{-1}$ is computed during the course of the reduced augmented CG solve. Thus, in the standard version of the framework these orthogonalization steps are cheap.

Unfortunately, orthogonalization is only done with respect to $\tilde{Y} = [W, Y\hat{Y}]$; it might be preferable to orthogonalize against the entire recycle space determined by Y since stage 3 will converge more quickly in that case. We introduce a modification to stage 3 that allows near-orthogonalization against all columns of Y even when starting with $\tilde{Y}$.

This takes the form of an inner iterative method described in Algorithm 11, which replaces the two orthogonalization steps in stage 3. It orthogonalizes each new search direction against the entire space span(Y), and not just the space span$(\tilde{Y})$ on which the reduced solution lies. When stage 2 is not present this method is not needed since in that case $Y = \tilde{Y}$.

The inner iteration is like stage 2, Algorithm 10, but with a few differences. One difference is that the righthand side here is Y^TAr as opposed to Y^Tr as it was in stage 2. The second difference is that stage 2 could obtain the solution on the

Algorithm 11 Inner iteration of stage 3

```
1: Goal: To solve $Y^TAYx = Y^TAb$, where $Y = [\tilde{Y}, \tilde{Z}]$ where $\tilde{Z}$ is unknown. We
   assume that systems with $\tilde{Y}^TA\tilde{Y}$ can be solved efficiently.[1]
2: In general, $r$ lies on the full space and $\hat{r}$ on the reduced space. These vectors
   satisfy $\hat{r} = Y^Tr$ and $p = Y\hat{p}$.
3: Inputs: $Y, \tilde{Y}, A, b$ and $\tilde{Y}^TA\tilde{Y}$, $\bar{\epsilon}$ tolerance
4: Solve $\tilde{Y}^TA\tilde{Y}\hat{y} = \tilde{Y}^TAb$ efficiently.
5: $x_0 = \tilde{Y}\hat{y}$
6: Define $r_0 = Ab - A\tilde{Y}\hat{y}$.
7: Note that $r_0$ satisfies $\tilde{Y}^Tr_0 = 0$.
8: $\hat{r}_0 = Y^Tr_0$
9: Solve $\tilde{Y}^TA\tilde{Y}\hat{\mu}_0 = \tilde{Y}^TAY\hat{r}_0$ efficiently.
10: $p_0 = Y\hat{r}_0 - \tilde{Y}\hat{\mu}_0$ and $\hat{p}_0 = \hat{r}_0 - \begin{bmatrix}\hat{\mu}_0 \\ 0\end{bmatrix}$
11: for $j = 0, 1, ...$ do
12:     $\alpha_j := \frac{(r_j, r_j)}{(Ap_j, p_j)}$
13:     $x_{j+1} := x_j + \alpha_j p_j$ and $\hat{x}_{j+1} = \hat{x}_j + \alpha_j \hat{p}_j$
14:     $r_{j+1} := r_j - \alpha_j Ap_j$ and $\hat{r}_{j+1} = Y^Tr_{j+1}$
15:     Solve $\tilde{Y}^TA\tilde{Y}\hat{\mu}_j = \tilde{Y}^TAY\hat{r}_{j+1}$ efficiently.
16:     $p_{j+1} = Y\hat{r}_{j+1} - \tilde{Y}\hat{\mu}_j$
17:     $\hat{p}_{j+1} = \hat{r}_{j+1} - \begin{bmatrix}\hat{\mu}_j \\ 0\end{bmatrix}$
18:     for $k = 1 : j$ do
19:         $\beta_k = \frac{(r_{j+1}, r_{j+1})}{(r_k, r_k)}$
20:         $p_{j+1} = p_{j+1} + \beta_k p_k$
21:         $\hat{p}_{j+1} = \hat{p}_{j+1} + \beta_k \hat{p}_k$
22:     end for
23:     if $\frac{(\hat{r}_{j+1}, \hat{r}_{j+1})}{(\hat{r}_0, \hat{r}_0)} \leq \bar{\epsilon}^2$ then
24:         Exit.
25:     end if
26: end for
```

range of Y using Z and Z^T instead of the larger Y. This is only possible when Z is known. In the case of the inner iteration, $Y = [\tilde{Y}, \tilde{Z}]$ but $\tilde{Z}$ is not actually computed. Therefore the full orthogonalization is performed on the range of Y. Algorithm 11 describes computations with vectors $\hat{p}$, but in reality these quantities

[1]Efficiently means there are Cholesky factors from stage 1 and/or a diagonal matrix from stage 2 $(\hat{Y}^TY^TAY\hat{Y})^{-1}$.

are also not computed. Instead the algorithm keeps track of the full vectors, p. Note that the loop beginning line 18 uses a full orthogonalization method. However, in our experiments, the number of inner iterations tends to be small so the additional overhead of this choice over CG is not high. Directions generated here can be added to $\tilde{Y}$ throughout stage 3, i.e. at the end of the above algorithm $\tilde{Y}$ can be replaced by $[\tilde{Y}, p_0, p_1, ...]$ for the next stage-3 iteration. In this case we would also save the inverse diagonal and $A[p_0, p_1, ...]$, and $\tilde{Y}$ would more closely approximate Y as stage 3 progresses.

An assumption built into the augmented CG method is that the initial residual satisfies $P^T r_0 = 0$. For the inner iteration, this condition is satisfied within numerical roundoff when $P = \tilde{Y}$. However, it is not the case when $P = Y$. Consider the solution of the form $x = \tilde{Y}\tilde{y} + V\hat{v}$. When we orthogonalize against $\tilde{Y}$, we have searched range($\tilde{Y}$) in the reduced problem (stage 1 and 2) and the space orthogonal to $\tilde{Y}$ in stage 3. With the iterative modification the solution is first considered on the range of $\tilde{Y}$ and then on the range of V where V is orthogonal to $Y = [\tilde{Y}, \tilde{Z}]$. Therefore, there is a space range($\tilde{Z}$) that is not searched. For stage 3 to converge with this modification, this component of the solution $\tilde{Z}$ must be smaller than δ.

In stage 2 the iterative computation produced a solution such that

$$\frac{||Y^T(b - AYy)||_2}{||Y^T b||_2} < \epsilon \ .$$

Since $Y^T(b - AYy) = Y^T r_0$, we have $||Y^T r_0||_2 < \epsilon ||Y^T b||_2$. Since $Y^T r_0 = [\tilde{Y}, \tilde{Z}]^T r_0 = \tilde{Y}^T r_0 + \tilde{Z}^T r_0 = \tilde{Z}^T r_0$ (because $\tilde{Y}^T r_0 = 0$ in exact arithmetic), then $||\tilde{Z}^T r_0||_2 < \epsilon ||Y^T b||_2$ and the component of the exact solution that lies on the range of $\tilde{Z}$ must be small. Therefore, for a given δ the choice of ϵ must be sufficiently small to ensure the convergence of the stage-3 algorithm when using the inner iterative method.

4.4 Numerical results

4.4.1 Background of problems

To test this framework we utilize sample problems from the Adagio code [44], a package used to analyze the deformation of solids. The sequence of linear systems is defined by seeking a quasi-static equilibrium which requires the solution of collection of a nonlinear problems for a series of time steps. For a nonlinear iteration l and time step i, solving for the quasi-static equilibrium requires the solution of a problem of the form

$$\underset{z\in\mathbb{R}^N}{\text{minimize}} \quad g_i^l(z) \ . \tag{4.17}$$

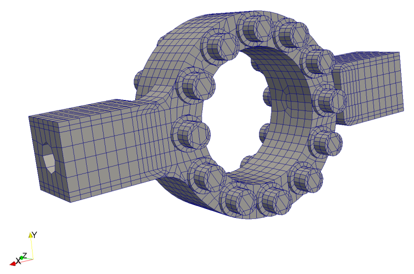
Define $r_i^l := \nabla g_i^l(z)$. The solution of the optimization problem requires the solution of a linear system

$$M^l(u_i)x = r_i^l \tag{4.18}$$

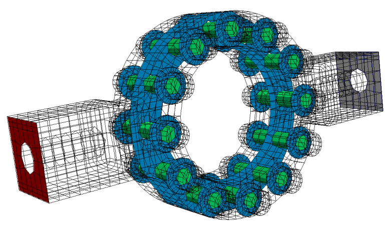
where $M^l(u_i)$ is a symmetric positive definite matrix at each l and i and u_i is the displacement from equilibrium at time step i. This results in a sequence of linear systems of the form in equation (4.1). Thus we loop over the time steps $i = 1, \ldots, n_t$, then at each time step take nonlinear iterations $l = 1, \ldots, n_l$, and save $A_j = M^l(u_i)$ and $b_j = r_i^l$ where $j = (i-1)n_l + l$.

The data is generated by following this solution process to find the quasi-static equilibrium in the deformation of a solid object subjected to a change in temperature and pressure over time. The first problem we consider is the solid in the shape of "pancake" domain pictured in Figure 4.1.

The x-, y-, and z-displacements of the rightmost surface in Figure 4.1(b) are zero. The x- and y- displacements of the leftmost surface are also zero. The leftmost surface is also subjected to the time-dependent pressure load depicted in Figure 4.2. The time-dependent thermal load depicted in Figure 4.3 is applied to the bolts (green components in Figure 4.1(b)). The contact surfaces are shown in blue in Figure 4.1(b).



(a) Finite-element mesh. $N = 27324$



(b) Pressure-loaded surface (red), contact surfaces (blue), prescribed temperature (green), dirichlet boundary condition (gray).

Figure 4.1: Pancake domain.

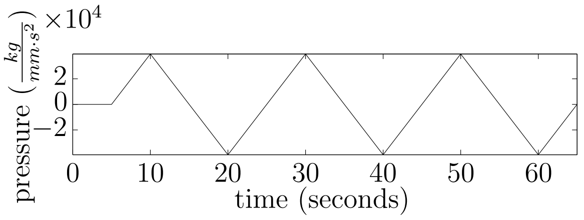


Figure 4.2: Time-dependent pressure load applied to leftmost surface (extrema are $\pm 3.94 \times 10^4 \ \frac{\text{kg}}{\text{mm}\cdot\text{s}^2}$).

The problem is discretized by the finite-element method using a mesh gen-

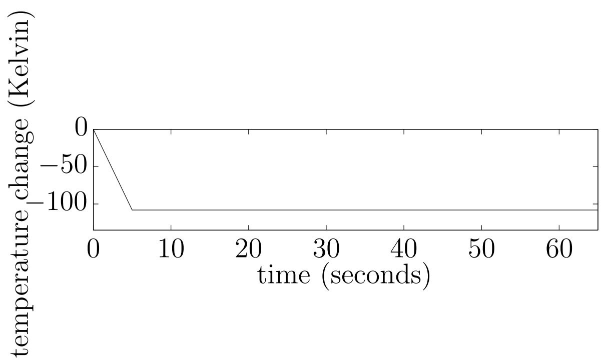


Figure 4.3: Time-dependent temperature load applied to bolts.

erated by the SIERRA toolkit [26]. The mesh consists of 9108 nodes and 4719 hexahedral elements. At each node, there are three degrees of freedom (the x-, y-, and z-displacements), which leads to a total of $27,324$ degrees of freedom. The second problem uses the I-beam domain depicted in Figure 4.4 with a mesh containing $N = 39411$ degrees of freedom.

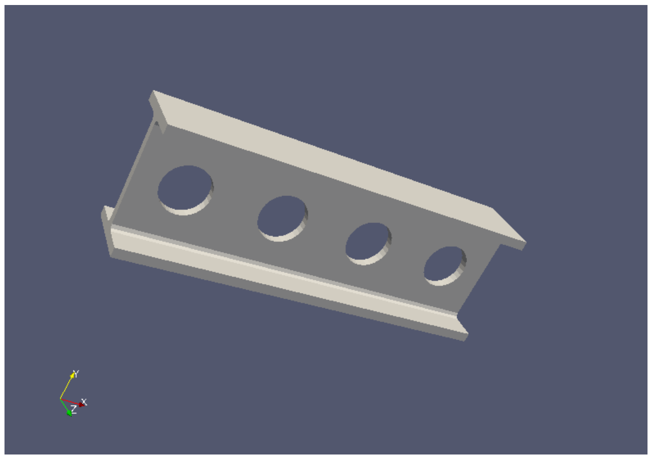


Figure 4.4: I-beam domain with mesh with $N = 39411$ degrees of freedom.

4.4.2 Comparison of recycling methods for problem 1

Given the sequence from the Adagio code described in Section 4.4.1, for the first domain we have a set of 47 linear systems from equation (4.18) each with order

$N = 27324$. The three-stage framework was implemented using Matlab. Each of the methods described in Section 4.3.1 (two compression methods, deflation and POD, and a solver that does not use compression) was used to solve the sequence. We present the average number of matvecs, number of stage-3 iterations (which is equivalent to the number of preconditioner applications), and CPU time per system to solve each of the 47 systems to within a specified tolerance δ. For solvers that use compression, the system is compressed after the augmenting space reaches $n_{max} = 200$ vectors. The resulting compressed space keeps $k = 100$ vectors.

For this data it is necessary to use an augmented full-orthogonalization (FOM) method in stage 3 in place of the augmented CG method. This method replaces line 16 of Algorithm 8 with the following [64]:

```
$p_k = z_k - P\mu_k$
**for** $i = 1 : k - 1$ **do**
    $\beta_i = \frac{r_k^T z_k}{r_i^T z_i}$
    $p_k = p_k + \beta_i p_i$
**end for**
```

In exact arithmetic for symmetric positive-definite systems, these methods are equivalent. However, this modification is necessary for this problem because the augmented FOM ensures that the new directions generated (the columns of V_j in equation 4.2) are numerically full rank. While augmented CG reaches the solution with less work per iteration, it produces numerically rank-deficient search directions that cause problems during the stage-1 solve of the subsequent systems. In addition, in the absence of recycling the number of iterations needed for FOM to converge is smaller than for CG, so with an expensive preconditioner (as is the case for this problem), FOM is more efficient.

One detail left open in the discussion of the framework is whether to add the new directions to the stage-1 subspace W_j or the stage-2 subspace Z_j. Unless otherwise stated we add the directions to the stage-1 subspace. We also considered an alternative, referred to as *mixed*, where we add the new search directions to both stage 1 and stage 2 based on a given direction's relative importance to the other vectors. A search direction v_k is added to the stage-1 subspace if

$$\frac{v_k^T A_j v_k}{\sum_i v_i^T A_j v_i} > 0.001 \ , \tag{4.19}$$

is satisfied. Otherwise, it is added to the stage-2 subspace.

For preconditioning, we use a three-level algebraic multigrid (AMG) preconditioner with incomplete Cholesky smoothing for both pre-smoothing and post-smoothing. This preconditioner tends to be expensive to apply, especially when compared with the cost of a matvec for this system. Therefore, the results presented in this section show faster times for methods that minimize the number of applications of the preconditioning operator. The results also show the number of matvecs required for convergence, since for different data and/or preconditioners the costs of matvecs and applications of the preconditioner may differ. The results presented are averages over the n_s systems.

Figure 4.5 contains the results for the following methods.

1. No recycling. Each of the 47 systems is solved without recycling using preconditioned FOM.

2. No compression. All search directions are used, so the augmenting space is $S = [V_1, V_2, .., V_{j-1}]$. This method requires the fewest stage-3 iterations and

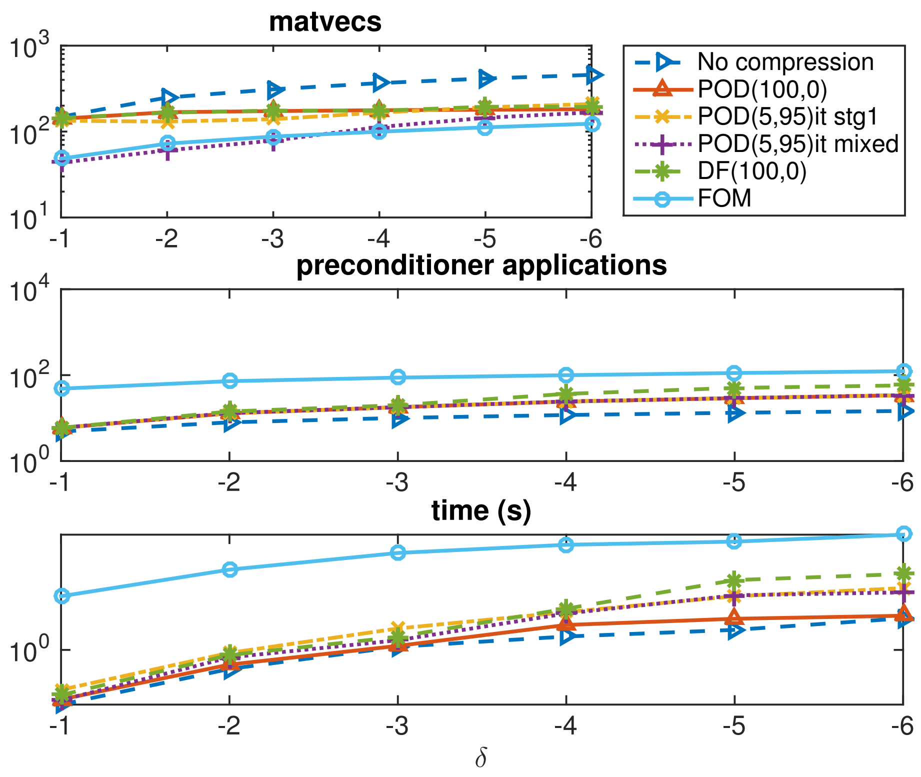


Figure 4.5: Results for problem 1. Average number of matvecs, applications of the preconditioner, and CPU time to compute solutions within tolerances $\delta = 10^{-1}$ through $\delta = 10^{-6}$. Compares FOM without recycling with a variety of recycling methods. See Figure 4.6 for a comparison of only the recycling methods.

thus, the smallest number of applications of the preconditioner.

3. Deflation. When the augmenting space exceeds 200 vectors, the deflation method described in Section 4.3.1.1 is used for compression. No stage-2 computations are done in conjunction with deflation, since the system is not naturally well-conditioned. Therefore, all new directions are added to W_j.

4. POD compression, denoted POD(k_1, k_2).

   - Stage 1 only, with $k_1 = 100$. After compression $W_j = \Phi$, Z_j is empty, and all new search directions are added to W_j.

   - POD(5,95) iterative stage 1. After compression, W_j has 5 vectors and Z_j

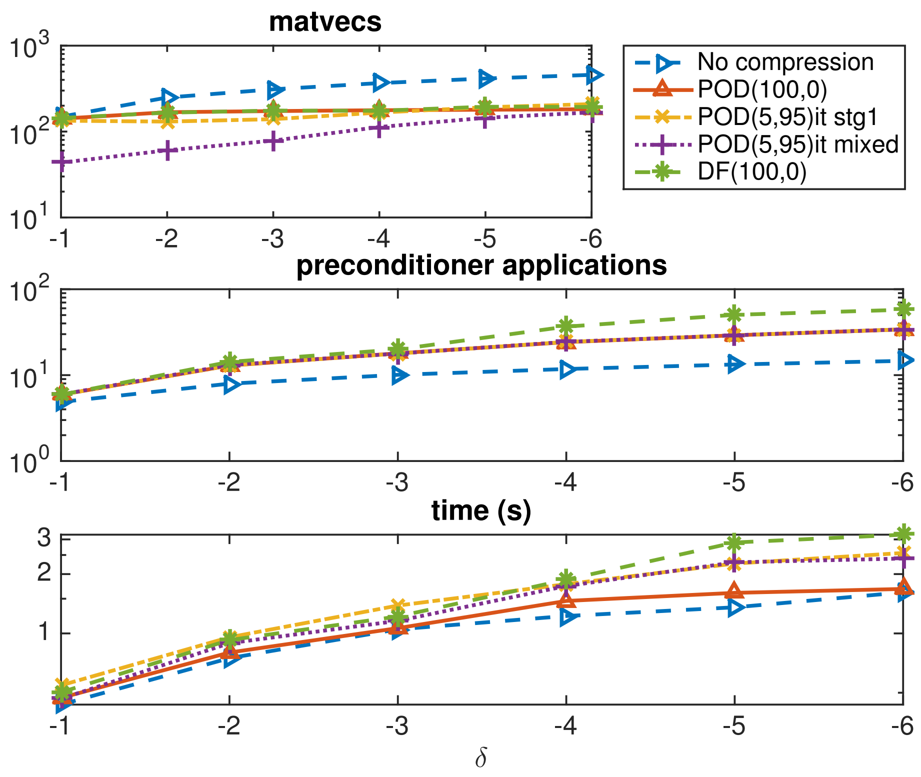


Figure 4.6: Comparison of recycling methods for the problem 1. Average number of matvecs, applications of the preconditioner, and CPU time to compute solutions within tolerances $\delta = 10^{-1}$ through $\delta = 10^{-6}$.

has 95 vectors. For this method all new directions are added to W_j. Note that $\epsilon = 10^{-4}\delta$ for $\delta \geq 10^{-3}$, and $\epsilon = 10^{-5}\delta$ otherwise and $\bar{\epsilon} = 10^{-2}\delta$ for all tolerances.

- POD(5,95) iterative mixed. Same as POD(5,95) iterative stage 1 for the system immediately after compression, but instead of adding all directions to W_j, they are added to W_j and Z_j according to the condition in equation (4.19). Note $\epsilon = 10^{-4}\delta$ for $\delta \geq 10^{-2}$, and $\epsilon = 10^{-6}\delta$ otherwise and $\bar{\epsilon} = 10^{-2}\delta$ for all δ.

First of all, Figure 4.5 demonstrates that recycling provides a huge benefit, since FOM without recycling is the slowest method for all tolerances (seen in the bottom plot). To see the differences between recycling methods more clearly, Figure 4.6 contains the same results without FOM. The no compression case is the fastest method for all tolerances. This means that not only is recycling necessary, there is a benefit to using all the directions and minimizing the number of applications of the preconditioner since that is the most significant cost for this data.

We also note that the POD methods (especially the POD(100,0) method) perform similarly to the no compression method, suggesting that the POD is effectively capturing the important directions. Note that the inner iterative method used in the two POD(5,95) methods produces the same number of preconditioner applications (seen in the middle plot) as POD(100,0). This is exactly what the iterative modification is intended to do. This comes at some additional cost of matvecs and other overhead so POD(100,0) is fastest with respect to time, but the fact that the same behavior is exhibited is important. We might expect this overhead to to be amortized in the case where the recycle space is much larger.

4.4.3 Comparison of recycling methods for problem 2

The I-beam domain produces a total of 49 linear systems with N = 39411 degrees of freedom. For this domain, we use the same AMG preconditioner with incomplete Cholesky smoothing. Since the matrices are larger, four levels of multigrid are used.

The same methods are considered for this data with one additional method, POD(5,95) iterative stage 2 where new directions are added to Z_j. All POD(5,95)

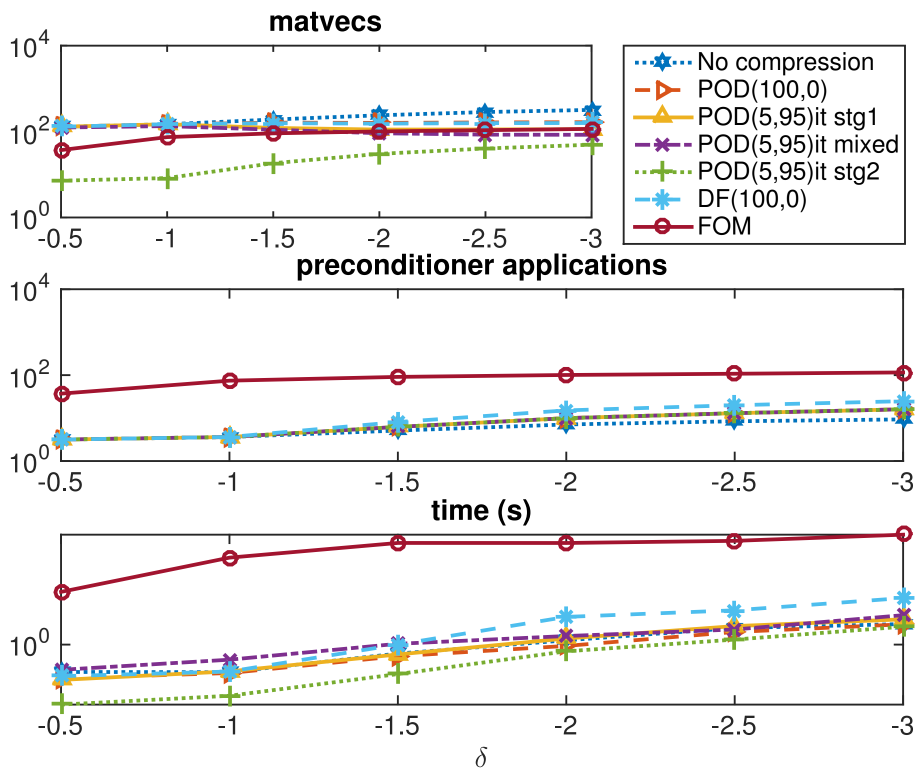


Figure 4.7: Results for the sequence generated for problem 2.

methods for this problem use parameters $\epsilon = 10^{-4}\delta$, $\bar{\epsilon} = 10^{-2}\delta$.

Figure 4.7 again strongly demonstrates the utility of recycling, as the FOM method is the slowest and requires the most applications of the preconditioner. In addition, the results without FOM, shown in Figure 4.8 illustrate the need for compression. While no compression still minimizes the number of preconditioner applications, the lower matvec cost and lower overhead in the orthogonalization steps of stage 3 lead to lower overall costs for some compression methods. Figure 4.8 illustrates the benefit of using a direct and iterative approach for solving the reduced model, since the iterative method POD(5,95) adding directions to stage 2 is best for all tolerances.

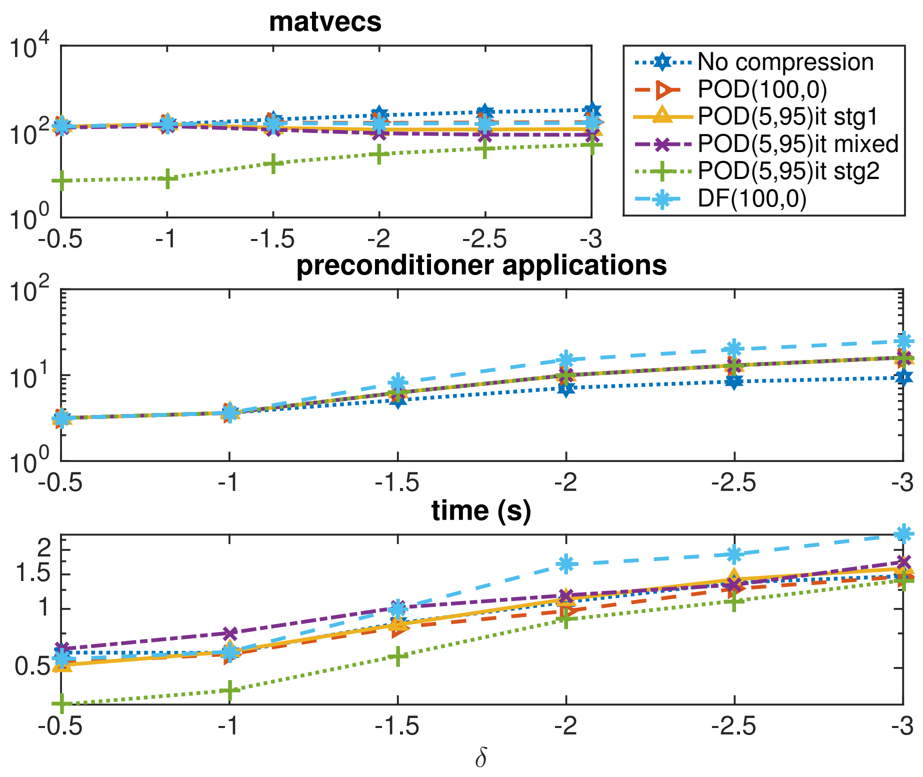


Figure 4.8: Comparison of recycling methods for problem 2.

4.4.4 POD-weight experiments

The discussion of the POD method in Section 4.3.1.2 indicated that the weights, $\{\gamma_i\}$, should be chosen such that

$$x_{est} = \sum_{i=1}^{n_w} \gamma_i s_i$$

where s_i are the vectors being compressed and x_{est} is a good estimate of the solution. Recall that a solution obtained using this framework can be written

$$x = Ww + Y\hat{Y}y + Vv \; . \tag{4.20}$$

Therefore, the exact weights to produce x correspond to w, y, and v. To see this, rewrite equation (4.20) in terms of the set of directions $S = [W, Z, V]$

$$x = \begin{bmatrix} W, Z, V \end{bmatrix} \begin{bmatrix} w + \hat{Y}_1 y \\ \hat{Y}_2 y \\ v \end{bmatrix} = S\gamma$$

where

$$\hat{Y} = \begin{bmatrix} \hat{Y}_1 \\ \hat{Y}_2 \end{bmatrix}$$

and the number of rows in $\hat{Y}_1$ is k_1 and the number of rows in $\hat{Y}_2$ is k_2. Note w is obtained directly in stage 1, while y and v correspond to the parameters $\{\alpha^{(k)}\}$ generated during the course of the stage 2 and stage 3 respectively.

We compare several weighting schemes using the matrices generated for problem 1. After the solution of the first 10 systems, a weighted POD compression is performed. The following three choices are considered.

1. **Ideal weights** are computed by solving the 11th system without compression. The ideal weights are stage-1 and stage-2 vectors (w and y) from the 11th system

$$\gamma_{ideal} = \begin{bmatrix} w_{11} + \hat{Y}_1 y_{11} \\ \hat{Y}_2 y_{11} \end{bmatrix} . \tag{4.21}$$

   This weighting scheme is not practical when solving the entire sequence of systems but is meant to illustrate the best possible choice.

2. **Recent weights** are obtained from the solution of the previous system. The weights are

$$\gamma_{10} = \begin{bmatrix} w_{10} + \hat{Y}_1 y_{10} \\ \hat{Y}_2 y_{10} \\ v_{10} \end{bmatrix} , \tag{4.22}$$

where w_{10}, y_{10}, v_{10} are taken from the solution of the 10th system as defined in equation (4.20). These weights are easily obtained from the information computed during the solution process for the 10th system.

3. **All weights** uses the weights from all previous systems (since the last compression). They are combined as $\gamma = \gamma_{10} + \frac{1}{2}\gamma_9 + \frac{1}{2^2}\gamma_8 + \ldots + \frac{1}{2^9}\gamma_1$ where $\{\gamma_i\}$ are the weights for a particular system, as in equation (4.22), with the appropriate number of zeros appended to the bottom to make the vectors the same length. These weights are also easily computed and can be updated at the end of each solve. These weights are used for all other results with the POD method.

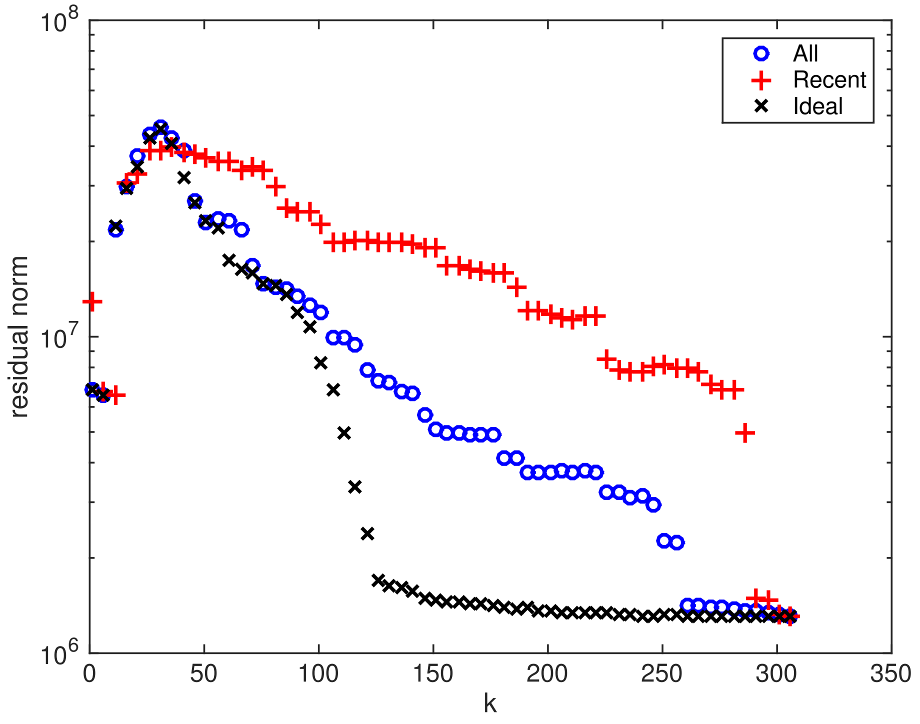


Figure 4.9: The residual norm before stage 3 as a function of number of vectors in the POD for system 11, compressed after system 10 for problem 1.

Figure 4.9 illustrates that the ideal weights minimize the residual after the reduced system is solved. This means the ideal weights lead to a better estimate

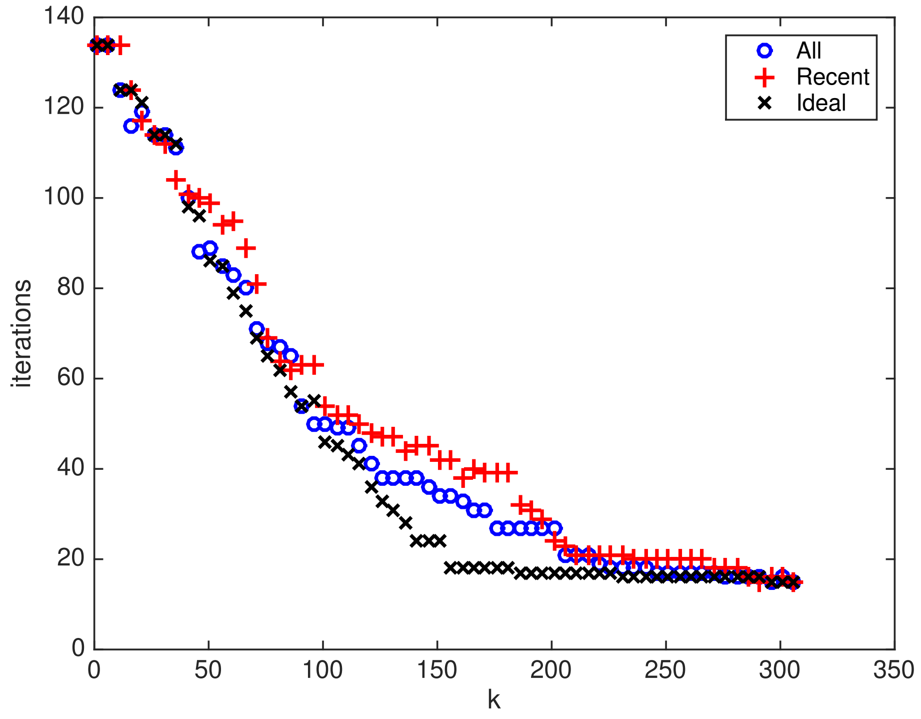


Figure 4.10: Number of stage-3 iterations taken after reduced solve as a function of the number of vectors in the POD with a stage-3 tolerance of $\delta = 10^{-6}$ for system 11, compressed after system 10 for problem 1.

of the solution in the reduced problem. Note that the “all weights” choice (what is used for all POD results) is close to the ideal case. In Figure 4.10, the ideal weights also minimize the number of stage-3 iterations performed in the augmented FOM as expected. In addition, the two other methods produce a similar number of stage-3 iterations as the ideal weights. This suggests that the all weights scheme is a good approximation of the ideal weights for both producing an accurate reduced solution and at providing similar convergence in stage 3.

4.4.5 Goal-oriented proper orthogonal decomposition

As discussed in Section 4.3.1.3, when the quantity of interest, z_j, for the linear sequence is a linear function of the output, i.e.

$$z_j = Cx_j$$

where $C \in \mathbb{R}^{N\times d}$, the weighted POD can be tailored to improve the convergence of the output z_j. The convergence of $z_j^{(i)}$ to the solution z_j is directly connected to the convergence of the solution $x_j^{(i)}$ in the goal-oriented norm

$$||x||_{C^TC} = (Cx, Cx)^{1/2}\ . \tag{4.23}$$

We consider compressing the directions using the weighted POD using the goal-oriented norm.

We design an experiment to demonstrate the advantage of using the goal-oriented POD approach to ensure fast convergence of the solution in the goal-oriented norm. We will use the three-stage framework as before and track the error of the solution in the goal-oriented norm

$$||e_j^{(i)}||_{C^TC} = ||\bar{x}_j - x_j^{(i)}||_{C^TC} \tag{4.24}$$

where $\bar{x}_j$ is the exact solution obtained using a direct method. For this experiment, we randomly generate $C \in \mathbb{R}^{N\times 100}$. Then the matrix, C, is fixed and we assume the quantity of interest is $z_j = Cx_j$ for all $j = 1, ..., n_s$. We proceed with Algorithm 9 using $\delta = 10^{-6}$ for problem 1 and $\delta = 10^{-3}$ for problem 2. For both problems, we use $n_{max} = 200$. During this algorithm, we track the goal-oriented error in equation (4.24). Then we measure the average number of matvecs, applications of the preconditioner, and CPU time for the error to satisfy $||e_j^{(i)}||_{C^TC} < \tau$ for a variety of tolerances τ. With τ on the x-axis, Figures 4.11 - 4.14 compare the performance in these metrics of goal-oriented POD with no compression, POD using the A-norm, and deflation.

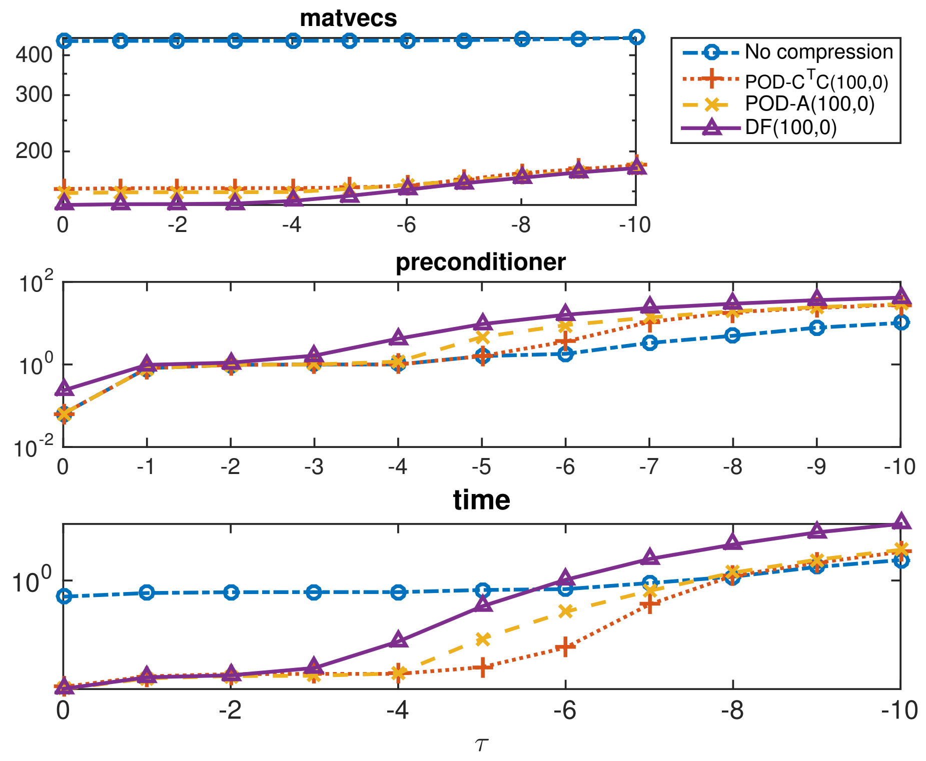


Figure 4.11: Stage 1 POD methods for problem 1 with convergence to the tolerance on the x-axis measured in the goal-oriented norm.

The POD methods using the A-norm and the C^TC-norm are best for inexact tolerances in Figure 4.11 and Figure 4.12. These plots also illustrate the benefit of goal-oriented norm over the A-norm, shown in Figure 4.11 where for tolerances in the middle ranges (between $\tau = 10^{-4}$ and 10^{-7}). In Figure 4.12 we see that the POD(5,95) methods without the iterative modification described in Section 4.3.2 are faster for the inexact tolerances. This occurs because very few stage-3 iterations

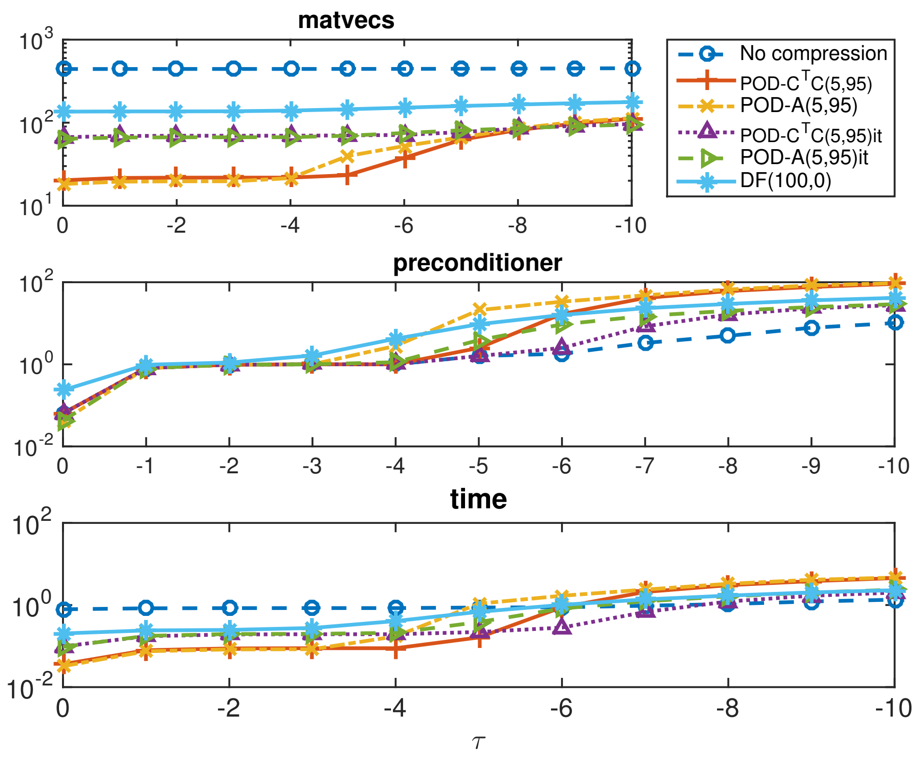


Figure 4.12: Stage 1/2 POD methods for problem 1 with convergence to the tolerance on the x-axis measured in the goal-oriented norm.

are required for convergence in the goal-oriented norm to this level. For $\tau \leq 10^{-6}$, we see that the iterative modification (or no compression) is preferable.

For problem 2, the goal-oriented comparison is presented in Figures 4.13 and 4.14. The results illustrate that there is an advantage to using the goal-oriented norm over the A-norm for most values of τ, since POD-C^TC(100,0) is fastest in Figure 4.13 and POD-C^TC(5,95)it method is fastest for $\tau \leq 10^{-3}$.

We repeat the experiments discussed in Section 4.4.4 and compare the three weighting methods for the goal-oriented POD for problem 1. The experiments measure the number of stage-3 iterations required to converge to a goal-oriented error of $\tau = 10^{-6}$ as a function of the number of vectors in the POD was varied. The experiments used a stage-3 tolerance of $\delta = 10^{-8}$ to accrue the search directions

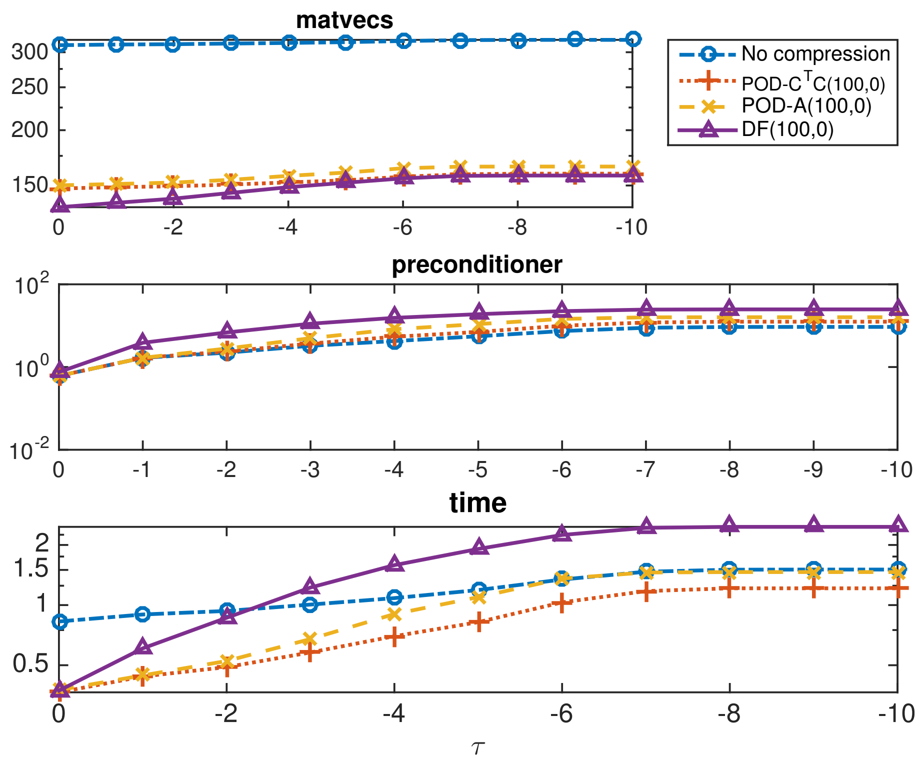


Figure 4.13: POD methods stage 1 only with convergence to tolerance measured in the goal-oriented norm for problem 2.

for 10 systems and compressed using the goal-oriented norm. The resulting stage-3 iterations are for the $j = 11$th system. As was the case for the POD weights for the A-norm, the results of Figure 4.15 show that the ideal weights perform best by minimizing the number of stage-3 iterations required for convergence in the goal-oriented norm. The other two weighting schemes closely follow the ideal weights.

4.4.6 Improvements for deflation methods

For these two problems, the POD method of compression is preferable to deflation. In addition, the results for problem 2 illustrate that using direct and iterative methods with the POD method for the reduced problem can be advantageous. These methods of POD compression give rise to a way to modify the deflation method that

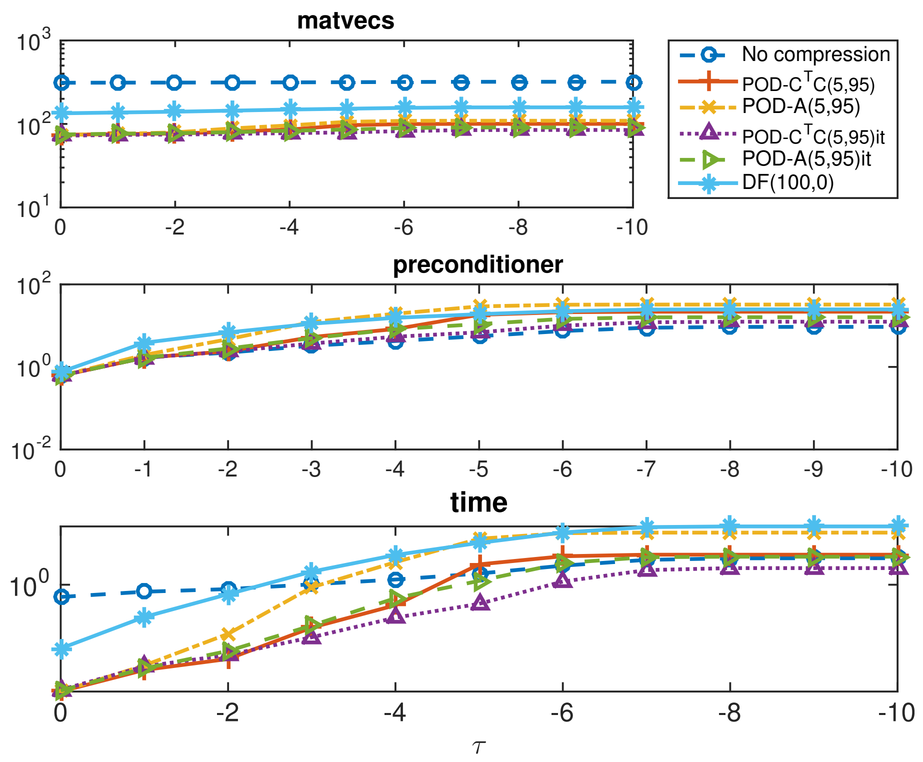


Figure 4.14: POD methods using stage 1 and 2 with convergence measured in goal-oriented norm for problem 2.

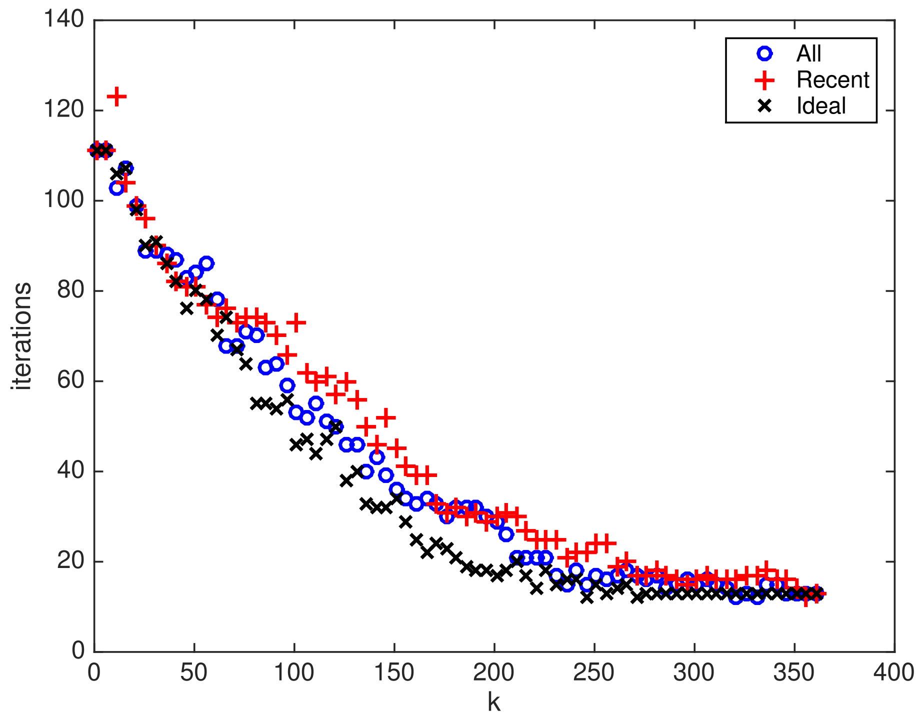


Figure 4.15: Comparison of weighting schemes for the number of stage-3 iterations required for convergence of the goal-oriented error for problem 1.

may permit it to perform successfully in the three-stage framework.

First, the results for deflation presented above entail solution of the generalized eigenvalue problem in equation (4.11) without normalization of the eigenvectors. This causes some numerical difficulties. Consider the case where there is some compressed space Φ_k and the stage-3 search directions come from the next three systems V_{k+1}, V_{k+2}, V_{k+3}. In this situation $S = [\Phi_k, V_{k+1}, V_{k+2}, V_{k+3}]$. Since each set of search directions is produced by augmenting against the previous space (i.e., each vector in V_{k+2} is orthogonal to the columns of $[\Phi_k, V_{k+1}]$), the matrix S is full rank in exact arithmetic. However, Φ_k from the deflation algorithm is not normalized as in the POD compression causing (numerical) rank-deficiencies in S. Such deficiencies eventually cause problems in both the generalized eigenvalue problem in the compression step and stage 1 where a symmetric positive definite matrix is required for the Cholesky factorization.

These rank deficiencies can be corrected:

1. After truncation when $\Phi = S\Psi$, compute the eigenvalues of $\Phi^T A \Phi$. If any of these eigenvalues are less than 10^{-12}, the associated eigenvectors can be discarded. Note in this case the recycle space will have dimension less than k.

2. Stage 1 is modified to use pivoted Cholesky factorization [38]. This shows when $W^T AW$ is rank deficient and provides a Cholesky factor of dimension corresponding to the numerical rank of the reduced problem. In this case the dimension of the reduced space, k_1, is updated accordingly.

Recall that the reason that the reduced problem in the POD method is a good match for a reduced iterative method is that both the added directions, $\{V_i D_i^{-1/2}\}$ and the compressed space, $\Phi = S\Gamma\Psi\Lambda^{-1/2}$, produce a well-conditioned reduced system. If such a well-conditioned recycled space can be generated for compression via deflation, this would ensure that stage 2 is well conditioned. It has the added benefit that it fixes the rank-deficiency issues, since the recycle space and the search directions are both A-orthogonalized.

Since the systems A_j are symmetric positive-definite matrices, the systems $S^T A_j S$ are also symmetric positive definite. Therefore, the eigenvectors of the generalized eigenvalue problem in equation (4.11)

$$S^T A_j^T A_j S\psi_i = \lambda_i S^T A_j S\psi_i \tag{4.25}$$

can be normalized using the $S^T A_j S$-norm such that $\Psi = [\psi_1, \psi_2, ...\psi_k]$ satisfies $\Psi^T S^T A_j S\Psi = I$. Then the reduced problem using the recycled space $\Phi = S\Psi$ will also be perfectly conditioned with the exact A_j and well conditioned for a matrix A_{j+1} that is close to A_j. Therefore, the stage-2 iterative solves after compression via deflation now converge in just a few iterations.

Figures 4.16 and 4.17 illustrate that normalizing the deflation vectors improves the performance of deflation for the tighter tolerances. Furthermore, Figure 4.17 illustrates that a stage-1/2 approach can be used with deflation more effectively than just a stage-1 approach. This is consistent with the results for POD, where the stage-1 methods performed best for the first set of data and the hybrid approach is better suited to the data generated from problem 2. Finally, we illustrate (Figure 4.18) the condition number of the reduced matrix $(Y^T AY)$ for problem 1 for all systems

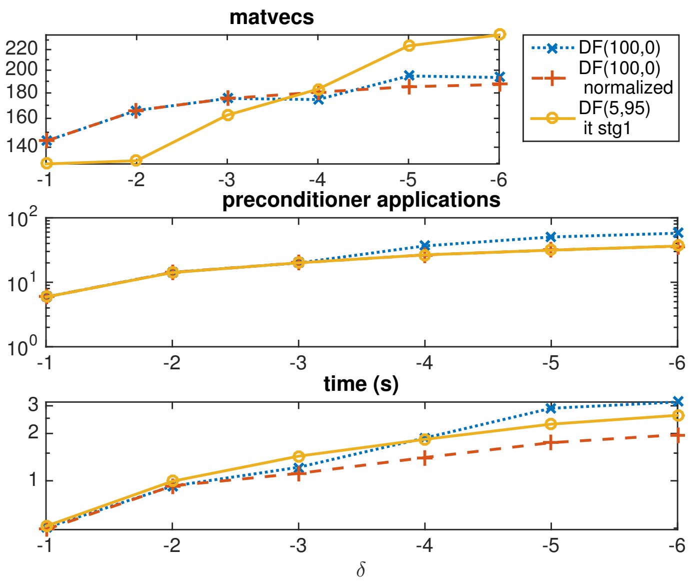


Figure 4.16: Comparison of deflation methods for problem 1.

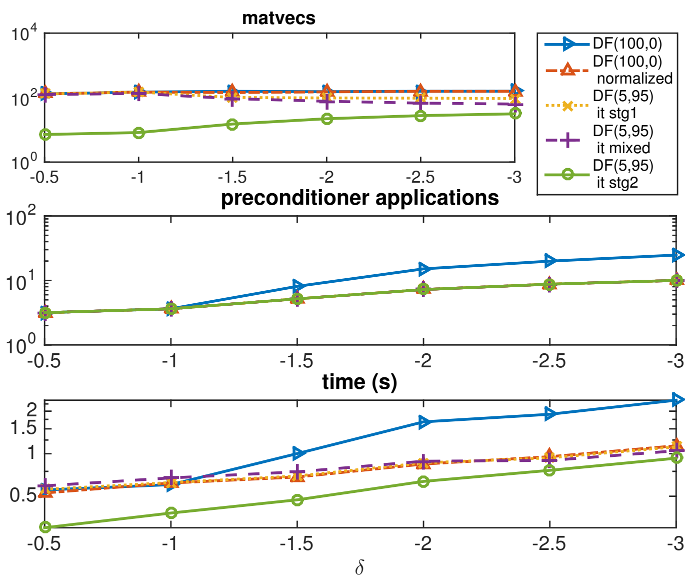


Figure 4.17: Comparison of deflation methods for problem 2.

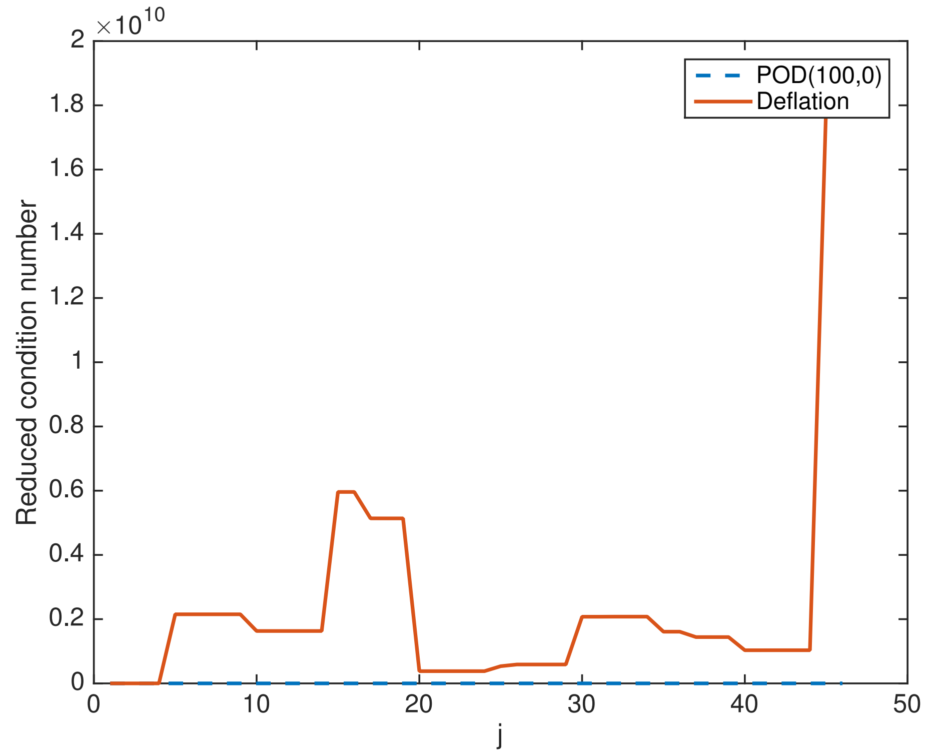
using POD compression and deflation. Note the plot is constructed using the same A_j used for construction of the reduced basis, so we would expect small deviations from these results when the framework is applied. We see in the first plot that the condition number of the deflation method blows up and the normalization ensures that the reduced condition number is close to one.

4.5 Conclusion

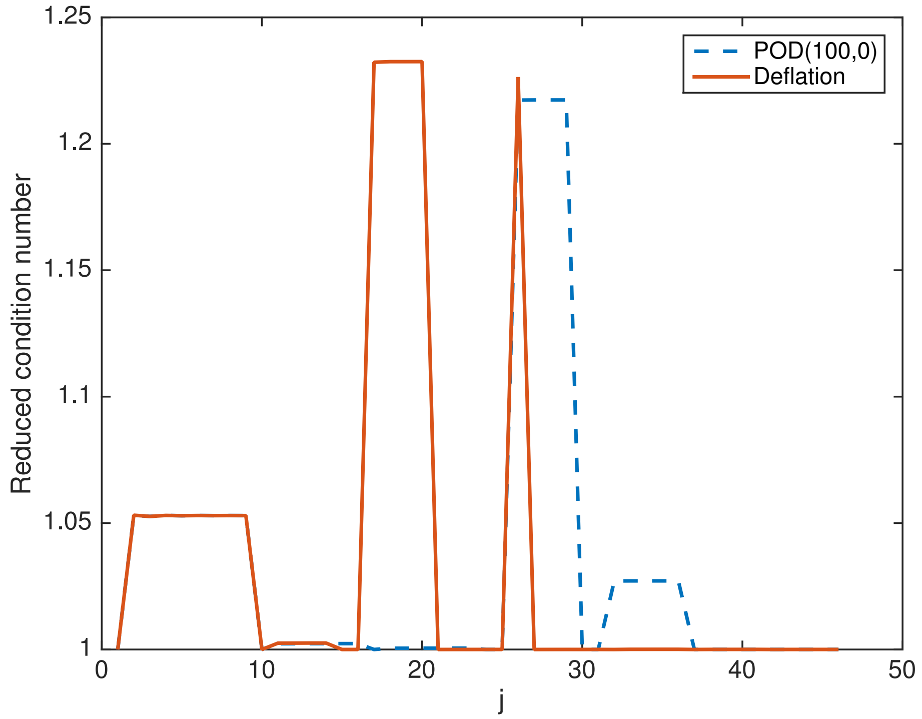
We have shown that Krylov-subspace recycling can be an effective technique for solving multiple related linear systems by leveraging results of some (initial) system solves to produce faster computation times for later system solves, especially when an offline-online paradigm is not available. Augmenting against previous search directions is always helpful when compared to solving the full systems individually. Also, compression is often needed.

We showed that the weighted POD is an alternative to deflation to perform compression and we devised an effective weighting scheme for this method. Weighted POD compression is also useful for efficient computation of a linear function of the output. Finally, we offered several techniques for improving deflation, including incorporating an iterative solve the reduced model.

As reduced-order modeling moves away from a strictly-offline strictly online-approach, the techniques presented in this framework illustrate that reduced-order models can reduce cost as early as the second step of a nonlinear iteration. While this chapter limited its attention to problems coming from iterative solution of



(a) Without normalization



(b) With normalization

Figure 4.18: Reduced condition number for systems using exact A_j .

nonlinear problems, the framework is adaptable to other settings, for example, uncertainty quantification used in conjunction with parameter-ordering methods like those described in [70].

Chapter 5

Conclusion

Reduced-order models constitute an effective method for efficiently solving parameterized PDEs in the many-query context. Given a reduced-order model of a particular dimension, we have shown that Krylov iterative solvers can be used to increase efficiency of the reduced-order model. Specifically, we have shown for the case of linear operators with affine dependence on the parameters that iterative methods with mean-based preconditioners can be used to improve efficiency in reduced-order models of moderate size. We presented examples where the utility of certain reduced-order models was lower when direct methods were used to solve the reduced problem, but utility improved when iterative methods are used. Thus, this work illustrates that the reduction in dimension required for reduced-order models to be effective can be increased when iterative methods are used. For the offline-online approach, this requires moving the cost of constructing preconditioners offline

Iterative methods have a similar benefit in the case of nonlinear operators with affine dependence. The iterative methods have shown speedups for reduced-order models used in conjunction with hyper-reduction techniques, in particular DEIM. The DEIM provides a significant speedup in assembling the system and the preconditioned iterative methods improve this further.

In addition, to address reduced-order modeling without the offline-online approach, we considered Krylov-subspace recycling methods, using a blended direct and iterative approach for solving the reduced problems. We showed that a weighted POD can be used to generate an effective reduced-order model for Krylov-subspace recycling and illustrated the effectiveness of the direct and iterative approach. Finally, we demonstrated several improvements to the deflation method. The iterative methods for the reduced problem in this setting avoid the need for preconditioners, because the reduced models were selected to be naturally well-conditioned. These results were demonstrated for a sequence of linear systems. These ideas represent an alternative to constructing reduced-order models for nonlinear operators based on DEIM since, Krylov subspace recycling methods can be used to treat the linear systems arising from the nonlinear iteration.

Bibliography

[1] B. O. Almroth, P. Stern, and F. A. Brogan. Automatic choice of global shape functions in structural analysis. *AIAA Journal*, 16(5):525–528, 1978.

[2] D. Amsallem and C. Farhat. Interpolation method for adapting reduced-order models and application to aeroelasticity. *AIAA Journal*, 46(7):1803–1813, 2008.

[3] D. Amsallem, M. Zahr, Y. Choi, and C. Farhat. Design optimization using hyper-reduced-order models. *Structural and Multidisciplinary Optimization*, pages 1–22, 2014.

[4] H. Antil, M. Heinkenschloss, and D. C. Sorensen. Application of the discrete empirical interpolation method to reduced order modeling of nonlinear and parametric systems. In G. Rozza, editor, *Springer MS&A series: Reduced Order Methods for Modeling and Computational Reduction*, volume 8. Springer-Verlag, Italia, Milano, 2013.

[5] J. Baiges, R. Codina, and S. Idelsohn. A domain decomposition strategy for reduced order models. Application to the incompressible Navier–Stokes equations. *Computer Methods in Applied Mechanics and Engineering*, 267:23–42, 2013.

[6] M. Barrault, Y. Maday, N. C. Nguyen, and A. T. Patera. An 'empirical interpolation' method: application to efficient reduced-basis discretization of partial differential equations. *Comptes Rendus Mathematique*, 339(9):667–672, 2004.

[7] M. Bebendorf, Y. Maday, and B. Stamm. Comparison of some reduced representation approximations. *Reduced Order Methods for Modeling and Computational Reduction*, pages 67–100, 2014.

[8] W. N. Bell, L. N. Olson, and J. B. Schroder. PyAMG: Algebraic multigrid solvers in Python v2.0, 2011. Release 2.0.

[9] P. Binev, A. Cohen, W. Dahmen, R. DeVore, G. Petrova, and P. Wojtaszczyk. Convergence rates for greedy algorithms in reduced basis methods. *SIAM Journal on Mathematical Analysis*, 43:1457–1472, 2011.

[10] P. Bochev and R. B. Lehoucq. On the finite element solution of the pure Neumann problem. *SIAM Review*, 47(1):50–66, 2005.

[11] S. Boyaval, C. Le Bris, T. Lelièvre, Y. Maday, N. C. Nguyen, and A. T. Patera. Reduced basis techniques for stochastic problems. *Archives of Computational Methods in Engineering*, 17(4):435–454, 2010.

[12] W. L. Briggs, V. E. Henson, and S. F. McCormick. *A Multigrid Tutorial.* 2000.

[13] A. Buffa, Y. Maday, A. T. Patera, C. Prud'homme, and G. Turinici. A priori convergence of the greedy algorithm for the parametrized reduced basis method. *ESAIM: Mathematical Modelling and Numerical Analysis*, 46(03):595–603, 2012.

[14] T. Bui-Thanh, K. Willcox, and O. Ghattas. Model reduction for large-scale systems with high-dimensional parametric input space. *SIAM Journal on Scientific Computing*, 30(6):3270–3288, 2008.

[15] K. Carlberg. Adaptive h-refinement for reduced-order models. *International Journal for Numerical Methods in Engineering*, 102(5):1192–1210, 2015.

[16] K. Carlberg and C. Farhat. An adaptive POD-Krylov reduced-order model for structural optimization. *8th World Congress on Structural and Multidisciplinary Optimization, Lisbon, Portugal*, 2009.

[17] K. Carlberg and C. Farhat. A low-cost, goal-oriented 'compact proper orthogonal decomposition' basis for model reduction of static systems. *International Journal for Numerical Methods in Engineering*, 86(3):381–402, 2011.

[18] K. Carlberg, C. Farhat, J. Cortial, and D. Amsallem. The GNAT method for nonlinear model reduction: Effective implementation and application to computational fluid dynamics and turbulent flows. *Journal of Computational Physics*, 242:623–647, 2013.

[19] S. Chaturantabut and D. C. Sorensen. Nonlinear model reduction via discrete empirical interpolation. *SIAM Journal on Scientific Computing*, 32(5):2737–2764, 2010.

[20] P. Chen, A. Quarteroni, and G. Rozza. A weighted reduced basis method for elliptic partial differential equations with random input data. *SIAM Journal of Numerical Analysis*, 51(6):3163–3185, 2013.

[21] W. Dahmen, C. Plesken, and G. Welper. Double greedy algorithms: reduced basis methods for transport dominated problems. *ESAIM: Mathematical Modelling and Numerical Analysis*, 48:623–663, 2014.

[22] T. A. Davis. Algorithm 832: UMFPACK v4. 3—an unsymmetric-pattern multifrontal method. *ACM Transactions on Mathematical Software (TOMS)*, 30(2):196–199, 2004.

[23] E. de Sturler. Nested Krylov methods based on GCR. *Journal of Computational and Applied Mathematics*, 67(1):15–41, 1996.

[24] E. De Sturler. Truncation strategies for optimal Krylov subspace methods. *SIAM Journal of Numerical Analysis*, 36(3):864–889, 1999.

[25] R. DeVore, G. Petrova, and P. Wojtaszczyk. Greedy algorithms for reduced bases in Banach spaces. *Constructive Approximation*, 37(3):455–466, 2013.

[26] H. C. Edwards, A. B. Williams, G. D. Sjaardema, D. G. Baur, and W. K. Cochran. SIERRA toolkit computational mesh conceptual model. *Sandia National Laboratories SAND Series, SAND*, 1192:2010, 2010.

[27] J. Eftang, A. T. Patera, and E. M. Ronquist. An hp certified reduced basis method for parameterized elliptic partial differential equations. *SIAM Journal on Scientific Computing*, 32(6):3170–3200, 2010.

[28] H. C. Elman and V. Forstall. Preconditioning techniques for reduced basis methods for parameterized partial differential equations. *SIAM Journal on Scientific Computing*, to appear, 2015.

[29] H. C. Elman and Q. Liao. Reduced basis collocation methods for partial differential equations with random coefficients. *SIAM/ASA Journal of Uncertainty Quantification*, 1:192–217, 2013.

[30] H. C. Elman, D. Silvester, and A. Wathen. *Finite Elements and Fast Iterative Solvers: with Applications in Incompressible Fluid Dynamics*. Oxford University Press, 2014.

[31] H. C. Elman, D. J. Silvester, and A. J. Wathen. *Finite Elements and Fast Iterative Solvers: with Applications in Incompressible Fluid Dynamics*. OUP Oxford, second edition, 2014.

[32] R. Everson and L. Sirovich. Karhunen–Loeve procedure for gappy data. *Journal of the Optical Society of America A*, 12(8):1657–1664, 1995.

[33] A. D. Gordon. *Solving PDEs with random data by stochastic collocation*. PhD thesis, University of Manchester, 2012.

[34] A. D. Gordon and C. E. Powell. Solving stochastic collocation systems with algebraic multigrid. In *Numerical Mathematics and Advanced Applications*, 2009.

[35] A. D. Gordon and C. E. Powell. On solving stochastic collocation systems with algebraic multigrid. *IMA Journal of Numerical Analysis*, 32:1051–1070, 2012.

[36] M. A. Grepl, Y. Maday, N. C. Nguyen, and A. T. Patera. Efficient reduced-basis treatment of nonaffine and nonlinear partial differential equations. *ESAIM: Mathematical Modelling and Numerical Analysis*, 41(03):575–605, 2007.

[37] B. Haasdonk, M. Dihlmann, and M. Ohlberger. A training set and multiple bases generation approach for parameterized model reduction based on adaptive grids in parameter space. *Mathematical and Computer Modelling of Dynamical Systems*, 17(4):423–442, 2011.

[38] N. J. Higham. Cholesky factorization. *Wiley Interdisciplinary Reviews: Computational Statistics*, 1(2):251–254, 2009.

[39] M. E. Kilmer and E. de Sturler. Recycling subspace information for diffuse optical tomography. *SIAM Journal on Scientific Computing*, 27(6):2140–2166, 2006.

[40] J. Kim. Phase field computations for ternary fluid flows. *Computer Methods in Applied Mechanics and Engineering*, 196(45):4779–4788, 2007.

[41] A. Klimke. Sparse Grid Interpolation Toolbox – user's guide. Technical Report IANS report 2007/017, University of Stuttgart, 2007.

[42] A. Klimke and B. Wohlmuth. Algorithm 847: spinterp: Piecewise multilinear hierarchical sparse grid interpolation in MATLAB. *ACM Transactions on Mathematical Software*, 31(4), 2005.

[43] J. D. Logan. *Transport modeling in hydrogeochemical systems, Chapter 2*, volume 15. Springer, 2001.

[44] J. A. Mitchell, A. S. Gullerud, W. M. Scherzinger, R. Koteras, and V. L. Porter. Adagio: non-linear quasi-static structural response using the SIERRA framework. 2001.

[45] R. B. Morgan. Computing interior eigenvalues of large matrices. *Linear Algebra and its Applications*, 154:289–309, 1991.

[46] R. B. Morgan. A restarted GMRES method augmented with eigenvectors. *SIAM Journal on Matrix Analysis and Applications*, 16(4):1154–1171, 1995.

[47] R. B. Morgan. GMRES with deflated restarting. *SIAM Journal on Scientific Computing*, 24(1):20–37, 2002.

[48] M. F. Murphy, G. H. Golub, and A. J. Wathen. A note on preconditioning for indefinite linear systems. *SIAM Journal on Scientific Computing*, 21(6):1969–1972, 2000.

[49] C. Nguyen, G. Rozza, D. B. Huynh, and A. T. Patera. Reduced basis approximation and a posteriori error estimation for parametrized parabolic pdes; application to real-time Bayesian parameter estimation. *Large Scale Inverse Problems and Quantification of Uncertainty*, (CMCS-CHAPTER-2008-001):151–178, 2010.

[50] A. K. Noor. Recent advances in reduction methods for nonlinear problems. *Computers & Structures*, 13(1):31–44, 1981.

[51] A. K. Noor and J. M. Peters. Reduced basis technique for nonlinear analysis of structures. *AIAA Journal*, 18(4):455–462, 1980.

[52] D. P. O'Leary. *Scientific Computing with Case Studies*. SIAM, 2009.

[53] M. A. Olshanskii and A. Reusken. Analysis of a Stokes interface problem. *Numerische Mathematik*, 103(1):129–149, 2006.

[54] P. Pacciarini and G. Rozza. Stabilized reduced basis method for parametrized advection-diffusion PDEs. *Computer Methods in Applied Mechanics and Engineering*, 274:1–18, 2014.

[55] M. L. Parks, E. De Sturler, G. Mackey, D. D. Johnson, and S. Maiti. Recycling Krylov subspaces for sequences of linear systems. *SIAM Journal on Scientific Computing*, 28(5):1651–1674, 2006.

[56] J. S. Peterson. The reduced basis method for incompressible viscous flow calculations. *SIAM Journal on Scientific and Statistical Computing*, 10(4):777–786, 1989.

[57] T. A. Porsching and M. L. Lee. The reduced basis method for initial value problems. *SIAM Journal on Numerical Analysis*, 24(6):1277–1287, 1987.

[58] C. E. Powell and H. C. Elman. Block-diagonal preconditioning for spectral stochastic finite-element systems. *IMA Journal of Numerical Analysis*, 29(2):350–375, 2009.

[59] C. E. Powell and D. J. Silvester. Preconditioning steady-state Navier–Stokes equations with random data. *SIAM Journal on Scientific Computing*, 34(5):A2482–A2506, 2012.

[60] A. Quarteroni and G. Rozza. Numerical solution of parametrized Navier-Stokes equations by reduced basis methods. *Numerical Methods for Partial Differential Equations*, 23(4):923–948, 2007.

[61] S. Ravindran. A reduced-order approach for optimal control of fluids using proper orthogonal decomposition. *International Journal for Numerical Methods in Fluids*, 34(5):425–448, 2000.

[62] F. Risler and C. Rey. Iterative accelerating algorithms with Krylov subspaces for the solution to large-scale nonlinear problems. *Numerical Algorithms*, 23(1):1–30, 2000.

[63] G. Rozza and K. Veroy. On the stability of the reduced basis method for Stokes equations in parametrized domains. *Computer Methods in Applied Mechanics and Engineering*, 196(7):1244–1260, 2007.

[64] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Other Titles in Applied Mathematics. SIAM, 2003.

[65] Y. Saad and M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986.

[66] Y. Saad, M. Yeung, J. Erhel, and F. Guyomarc'h. A deflated version of the conjugate gradient algorithm. *SIAM Journal on Scientific Computing*, 21(5):1909–1926, 2000.

[67] D. Silvester. IFISS 3.3 release notes. `http://www.maths.manchester.ac.uk/%7Edjs/ifiss/release.txt`, 2013.

[68] D. Silvester, H. C. Elman, and A. Ramage. Incompressible Flow and Iterative Solver Software (IFISS) version 3.3, October 2013. `http://www.manchester.ac.uk/ifiss/`.

[69] Z. Tan, D. V. Le, Z. Li, K. M. Lim, and B. C. Khoo. An immersed interface method for solving incompressible viscous flows with piecewise constant viscosity across a moving elastic membrane. *Journal of Computational Physics*, 227(23):9955–9983, 2008.

[70] E. Ullman. *Solution Strategies for Stochastic Finite Element Discretizations.* PhD thesis, 2008.

[71] D. Venturi, D. M. Tartakovsky, A. M. Tartakovsky, and G. E. Karniadakis. Exact PDF equations and closure approximations for advective-reactive transport. *Journal of Computational Physics*, 2013.

[72] K. Veroy, C. Prud'homme, D. V. Rovas, and A. T. Patera. A posteriori error bounds for reduced-basis approximation of parametrized noncoercive and nonlinear elliptic partial differential equations. In *Proceedings of the 16th AIAA computational fluid dynamics conference*, volume 3847, pages 23–26, 2003.

[73] J. Xu. Iterative methods by space decomposition and subspace correction. *SIAM Review*, 34(4):581–613, 1992.

[74] D. Zhang. *Stochastic Methods for Flow in Porous Media: Coping with Uncertainties.* Academic press, 2001.