Reduced Order Model Study of Burgers' Equation using Proper Orthogonal Decomposition

Christopher Jarvis

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Masters of Science
in
Mathematics

John A.Burns, Chair
Jeff Borggaard
John Rossi

February 21, 2012
Blacksburg, Virginia

Reduced Order Model Study of Burgers' Equation using Proper Orthogonal Decomposition

Christopher Jarvis

(ABSTRACT)

In this thesis we conduct a numerical study of the 1D viscous Burgers' equation and several Reduced Order Models (ROMs) over a range of parameter values. This study is motivated by the need for robust reduced order models that can be used both for design and control. Thus the model should first, allow for selection of optimal parameter values in a trade space and second, identify impacts from changes of parameter values that occur during development, production and sustainment of the designs. To facilitate this study we apply a Finite Element Method (FEM) and where applicable, the Group Finite Element Method (GFE) due its demonstrated stability and reduced complexity over the standard FEM. We also utilize Proper Orthogonal Decomposition (POD) as a model reduction technique and modifications of POD that include Global POD, and the sensitivity based modifications Extrapolated POD and Expanded POD. We then use a single baseline parameter in the parameter range to develop a ROM basis for each method above and investigate the error of each ROM method against a full order "truth" solution for the full parameter range.

Acknowledgements

I would like to express my sincere gratitude to my advisor and committee chairman Dr. John Burns. His profound knowledge, insight and guidance have been invaluable in helping with my thesis and broader education; his enthusiasm drives all who know him to enjoy mathematics. I would like thank Dr. John Rossi and Dr. Jeff Borggaard for serving on my committee and having supportive discussions concerning both theoretical and computational issues within this work. I further wish to thank the entire group of faculty, staff and students of the Interdisciplinary Center of Applied Mathematics (ICAM). ICAM is an incredible environment in which to perform research, have discussions and truly learn.

I also wish to acknowledge my support received through the Department of Defense Science, Mathematics and Research for Transformation (SMART) scholarship and my sponsorship from the United States Air Force Air Armament Center. Specifically, thank you to Ken Kennedy and Kent Watson for their support during my time here.

Thank you to my wife, Vanessa, without her love and support guiding me all the way, none of this would be possible. Finally, I thank our Heavenly Father, who created all things and knows all things. I believe the more we study the more we see His beauty all around us.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction and Motivation

Numerical solutions of complex nonlinear systems can be computationally expensive due to both spatial and temporal discretizations required to meet a desired accuracy. However, in many cases much of the solution information lies within a subspace whose dimension is significantly lower than the full dimension used in the discretization. The goal of Reduced Order Modeling (ROM) is to create a low dimensional model that contains a large percentage of the information from the full discretized system, while saving computational time and storage. There are several widely used model reduction techniques, Singular Value Decomposition (SVD) based methods including Proper Orthogonal Decomposition (POD) and balanced truncation as well as Krylov subspace methods including Lanczos and Arnoldi procedures. For an overview of model reduction techniques and references on Krylov based methods the interested reader is referred to [1–7]. Both POD and balanced truncation have received extensive study in recent years [8–33] to include blending of the two methods resulting in a balanced POD reduction technique for large systems where balanced truncation is intractable by itself [34–40]. In this thesis we focus on the standard POD since we are interested in PDE systems with a range of parameters and the subsequent reduced order models. In the finite dimensional case, for any integer r, where r is less than the dimension of a standard numerical solution, POD generates a set of orthonormal basis functions that "optimally" represents a given set of data. To implement POD, a collection of data snapshots at various states of the system must be found. For this reason POD is also known as the method of snapshots. Typically the ensemble of data snapshots are all from the same problem with fixed coefficients and parameters.

During a design cycle it is useful to understand the impact of changing parameters as variables in a total design trade space. The desire is to use reduced order models to allow for quick analysis of impacts resulting from changes in parameter values both throughout design, production and sustainment of a system. Therefore, we will study the accuracy of using POD based reduced order models across a range of parameter values and potential improvements. To facilitate the study we focus on Burgers' equation in one dimension defined by

$$\frac{\partial}{\partial t}u(t,x) + u(t,x)\frac{\partial}{\partial x}u(t,x) - q\frac{\partial^2}{\partial x^2}u(t,x) = 0. \tag{1.1}$$

This is a nonlinear parabolic PDE, containing both a convective term $u(t,x)\frac{\partial}{\partial x}u(t,x)$ and a dissipative term $\frac{\partial^2}{\partial x^2}u(t,x)$. Burgers' Equation is often used to test computational methods, and because it has properties similar to the Navier-Stokes equations, Burgers' equation is a good test model for control design [41]. Thus we shall think of the parameter q as $q = 1/Re$, where Re plays the role of the Reynolds number. Under certain conditions, exact solutions can be found by using either a Cole-Hopf transformation or decomposition methods [42–45]. We will use the Finite Element Method (FEM) to create our initial numerical approximate solutions to provide a basis for the ROM investigations. Furthermore, it has been demonstrated [46–50] that the Group Finite Element (GFE) applied to the conservation form of Burgers' equation produces the same level of accuracy and has greater stability than the traditional FEM approach.

1.2 Notation

In this thesis, we use the following notation. Let $X = L^2([0,1])$ be the Hilbert space of Lebesgue square integrable functions. Thus, a function f is in X if

$$\int_0^1 |f(x)|^2 dx < \infty \ .$$

We endow X with the inner product $\langle \cdot, \cdot \rangle_X$ and norm $\| \cdot \|_X$. That is, let f, g be functions in X, then the norm of f is defined as

$$\|f\|_X^2 = \int_0^1 |f(x)|^2 \, dx$$

and the inner product of the functions f, g is defined as

$$\langle f, g \rangle_X = \int_0^1 f(x)g(x) \ dx.$$

Moreover, a function w is in $L^2([0,T];X)$ if for each $0 \le t \le T$, $w(t,\cdot) \in X$ and

$$\int_0^T \int_0^1 |f(t,x)|^2 \ dx \, dt < \infty.$$

We will also be interested in the space $\mathbb{R}^n$ with a weighted inner product with weight given by the symmetric positive definite (SPD) $n \times n$ matrix S. Thus for $y, z \in \mathbb{R}^n$ the inner product of the vectors y, z is defined as

$$\langle y, z \rangle_S = y^T S z.$$

Also, we define the induced norm in this case to be

$$\|z\|_S^2 = z^T S z.$$

Throughout this work, the space of interest is L^2. Thus, we neglect the indexing of the inner product and norm for these cases.

1.3 Burgers Equation in Conservation form

We consider Burgers' equation in conservation form given by

$$u_t(t,x) + \frac{1}{2}\left[u(t,x)^2\right]_x - q u_{xx}(t,x) \quad = \quad f(t,x)\ , \quad x \in (0,1),\ t > 0. \tag{1.2}$$

The forcing f is assumed to be at least L^2 in space and time. We shall focus on two particular initial - boundary value problems. The first problem has Dirichlet boundary conditions at both $x = 0$ and $x = 1$ given by

$$u(t,0) = 0 \qquad u(t,1) = 0 \tag{1.3}$$

and initial value given by

$$u(0,x) = u_0(x). \tag{1.4}$$

The second problem has a Neumann boundary condition at $x = 0$ and a Dirichlet boundary condition at $x = 1$ given by

$$u_x(t,0) = \delta \in \mathbb{R} \qquad u(t,1) = 0 \tag{1.5}$$

and initial value given by

$$u(0,x) = u_0(x). \tag{1.6}$$

Here δ represents a constant "small" perturbation of the homogenous boundary condition $u_x(t,0) = 0$. We will use the Finite Element Method (FEM) to produce approximate solutions to these classes of initial boundary value problems.

Chapter 2

The Finite Element Method

The Finite Element Method (FEM) is a rather general numerical method that is often used to to approximate partial differential equations (PDEs). If the PDE is time dependent then the problem can be reduced to a system of ODEs which can be numerically integrated by known techniques. In this thesis we use standard piecewise linear basis functions for our approximations. Therefore, we divide the unit interval $[0,1]$ into N subintervals $[x_i, x_{i+1}]$ of uniform length $h = \frac{1}{N}$ where $x_i = ih$ for $i = 0, \ldots, N$. On each interval the global basis functions are defined by

$$\phi_0(x) = \begin{cases} \frac{x_1 - x}{h} & \text{for } x \in [0, x_1] \\ 0 & \text{otherwise,} \end{cases}$$

for $1 \leq i \leq N-1$

$$\phi_i(x) = \begin{cases} \frac{x - x_{i-1}}{h} & \text{for } x \in [x_{i-1}, x_i] \\ \frac{x_{i+1} - x}{h} & \text{for } x \in [x_i, x_{i+1}] \\ 0 & \text{otherwise,} \end{cases}$$

and

$$\phi_N(x) = \begin{cases} \frac{x - x_{N-1}}{h} & \text{for } x \in [x_{N-1}, x_N] \\ 0 & \text{otherwise.} \end{cases}$$

We form an approximation of $u(t,x)$ in the space spanned by the piecewise linear basis functions by setting

$$u^N(t,x) = \sum_{j=0}^{N} \alpha_j(t)\phi_j(x),$$

where $\alpha_j(t)$ represents the nodal unknown value of $u(t, x_j)$ at the jth node at time t.

The standard Finite Element approach yields a nonlinear time dependent equation. We also take advantage of the Group Finite Element (GFE) method, described by Fletcher [51]. This simplifies

the nonlinear term so that one can take advantage of grouping similar terms. The Burgers' equation in conservation form expresses the nonlinearity $u(t,x)u_x(t,x)$ as $\frac{1}{2}\left[u(t,x)^2\right]_x$ leading us to the approximation

$$u(t,x)^2 \approx u^N(t,x)^2 \approx \sum_{j=0}^{N} \alpha_j(t)^2 \phi_j(x). \tag{2.1}$$

This was shown in [46–50] to provide improved stability and computational efficiency since matrices do not need to be assembled at each time step.

2.1 Burgers' Equation with Dirichlet Boundary Conditions

Consider the first problem

$$u_t(t,x) + \frac{1}{2}\left[u(t,x)^2\right]_x - q u_{xx}(t,x) \quad = \quad f(t,x), \quad x \in (0,1),\ t > 0, \tag{2.2}$$

where $x \in [0,1]$ and $t \in [0,t_f]$. The boundary conditions are given by

$$u(t,0) = 0, \qquad\qquad u(t,1) = 0, \tag{2.3}$$

and the initial condition is

$$u(0,x) = u_0(x). \tag{2.4}$$

Multiplying both sides of (2.2) by a test function $v(x)$ and integrating yields

$$\int_0^1 \left(u_t(t,x) + \frac{1}{2}\left[u(t,x)^2\right]_x - q u_{xx}(t,x) \right) v(x)\, dx \quad = \quad \int_0^1 f(t,x) v(x)\, dx, \tag{2.5}$$

or equivalently,

$$\int_0^1 \left(u_t(t,x) + \frac{1}{2}\left[u(t,x)^2\right]_x \right) v(x)\, dx - q \int_0^1 u_{xx}(t,x) v(x)\, dx \quad = \quad \int_0^1 f(t,x) v(x)\, dx. \tag{2.6}$$

If $v(\cdot)$ is piecewise smooth, then $v'(x) \in L^2(0,1)$ so we can apply integration by parts to the second term on the left and take advantage of the essential Dirichlet boundary conditions to obtain

$$\begin{aligned} \int_0^1 u_{xx}(t,x) v(x)\, dx &= u_x(t,x) v(x) \,|_0^1 - \int_0^1 u_x(t,x) v_x(x)\, dx \\ &= -\int_0^1 u_x(t,x) v_x(x)\, dx. \end{aligned}$$

Thus (2.6) becomes the weak form of Burgers' equation with Dirichlet boundary conditions given by

$$\int_0^1 \left(\left(u_t(t,x) + \frac{1}{2}\left[u(t,x)^2\right]_x \right) v(x) + q u_x(t,x) v_x(x) \right) dx \;=\; \int_0^1 f(t,x) v(x)\, dx. \tag{2.7}$$

Note that (2.7) must hold for any piecewise smooth function $v(\cdot)$. Using the approximation by piecewise linear basis functions we write $u(t,x) \approx u^N(t,x) = \sum_{j=0}^N \alpha_j(t)\phi_j(x)$ and using the group approximation $u^N(t,x)^2 \approx \sum_{j=0}^N \alpha_j(t)^2 \phi_j(x)$, (2.7) becomes

$$\begin{aligned} \int_0^1 \left(\left(\sum_{j=0}^N \dot{\alpha}_j(t)\phi_j(x) \right) + \frac{1}{2}\left[\sum_{j=0}^N \alpha_j(t)^2 \phi_j(x) \right]_x \right) v(x) & \\ + q \left(\sum_{j=0}^N \alpha_j(t) \phi_j'(x) \right) v_x(x)\, dx \;&=\; \int_0^1 f(t,x) v(x)\, dx. \end{aligned} \tag{2.8}$$

Since this identity holds for arbitrary piecewise smooth $v(x)$, for each $i = 0, 1, ..., N$, we can set $v(x) = \phi_i(x)$. Then, from (2.8), for each i we obtain

$$\begin{aligned} \int_0^1 \left(\left(\sum_{j=0}^N \dot{\alpha}_j(t)\phi_j(x) \right) + \frac{1}{2}\left[\sum_{j=0}^N \alpha_j(t)^2 \phi_j(x) \right]_x \right) \phi_i(x) & \\ + q \left(\sum_{j=0}^N \alpha_j(t) \phi_j'(x) \right) \phi_i'(x)\, dx \;&=\; \int_0^1 f(t,x) \phi_i(x)\, dx \end{aligned} \tag{2.9}$$

for $i = 0, 1, ..., N$.

Since for all j, $\alpha_j(t)$ does not depend on x we can move those terms outside the integral to obtain

$$\begin{aligned} \sum_{j=0}^N \Bigg(\dot{\alpha}_j(t) \int_0^1 \phi_j(x)\phi_i(x)\, dx + \frac{1}{2}\alpha_j(t)^2 \int_0^1 \phi_j'(x)\phi_i(x)\, dx & \\ + q\alpha_j(t) \int_0^1 \phi_j'(x)\phi_i'(x)\, dx \Bigg) \;&=\; \int_0^1 f(t,x)\phi_i(x)\, dx. \end{aligned} \tag{2.10}$$

For each $i = 0, 1, ..., N$, the equation (2.10) can be written as a system of $N+1$ ODEs given by

$$M\dot{\boldsymbol{\alpha}}(t) + \frac{1}{2}\mathcal{B}\left(\boldsymbol{\alpha}(t)\right)\boldsymbol{\alpha}(t) + qC\boldsymbol{\alpha}(t) \;=\; F(t), \tag{2.11}$$

where $M_{ij} = \langle \phi_j(x), \phi_i(x) \rangle$, $C_{ij} = \left\langle \phi_j'(x), \phi_i'(x) \right\rangle$ and $F_i(t) = \langle f(t,x), \phi_i(x) \rangle$ for all $i, j = 0, 1, ..., N$ and $\mathcal{B}\left(\boldsymbol{\alpha}(t)\right)$ is a matrix that depends on $\boldsymbol{\alpha}(t)$.

Taking advantage of the Dirichlet boundary conditions we know $\alpha_0(t) = 0$ and $\alpha_N(t) = 0$ for all t. Hence we can eliminate these equations and reduce the size of (2.11) from $N+1$ to $N-1$ equations by solving only for the internal nodes. Thus the matrices M, C and $\mathcal{B}\left(\boldsymbol{\alpha}(t)\right)$ are given by

$$M = \frac{h}{6}\begin{pmatrix} 4 & 1 & & & \\ 1 & 4 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & 4 & 1 \\ & & & 1 & 4 \end{pmatrix}_{[N-1\times N-1]},$$

$$C = \frac{1}{h}\begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}_{[N-1\times N-1]},$$

and

$$\mathcal{B}(\boldsymbol{\alpha}(t)) = \begin{pmatrix} 0 & \frac{1}{2}\alpha_2(t) & & & \\ -\frac{1}{2}\alpha_1(t) & 0 & \frac{1}{2}\alpha_3(t) & & \\ & \ddots & \ddots & \ddots & \\ & & -\frac{1}{2}\alpha_{N-3}(t) & 0 & \frac{1}{2}\alpha_{N-1}(t) \\ & & & -\frac{1}{2}\alpha_{N-2}(t) & 0 \end{pmatrix}_{[N-1\times N-1]},$$

respectively.

During computation the term $\mathcal{B}(\boldsymbol{\alpha}(t))\,\boldsymbol{\alpha}(t)$ can be efficiently implemented as

$$\mathcal{B}(\boldsymbol{\alpha}(t))\,\boldsymbol{\alpha}(t) = \begin{pmatrix} 0 & \frac{1}{2} & & & \\ -\frac{1}{2} & 0 & \frac{1}{2} & & \\ & \ddots & \ddots & \ddots & \\ & & -\frac{1}{2} & 0 & \frac{1}{2} \\ & & & -\frac{1}{2} & 0 \end{pmatrix}_{[N-1\times N-1]} \begin{bmatrix} \alpha_1(t)^2 \\ \alpha_2(t)^2 \\ \vdots \\ \alpha_{N-2}(t)^2 \\ \alpha_{N-1}(t)^2 \end{bmatrix} \tag{2.12}$$

This ensures that we do not need to assemble a matrix in every time step, since the component-wise square is convenient and quick in MATLAB ® using the ".^2" command.

We also need to find the approximate initial condition $u_0(x)$ in terms of the basis functions, $\phi_i(x)$. Thus, we assume that $u_0(x) \approx u_0^N(x) = \sum_{j=0}^{N} \alpha_j(0)\phi_j(x)$ and note that

$$\int_0^1 u_0^N(x)v(x)\,dx = \int_0^1 u_0(x)v(x)\,dx. \tag{2.13}$$

Using the approximation $u_0^N(x) = \sum_{j=0}^{N} \alpha_j(0)\phi_j(x)$ and enforcing the essential boundary condition (2.13) becomes

$$\sum_{j=1}^{N-1} \alpha_j(0) \int_0^1 \phi_j(x)\phi_i(x)\,dx = \int_0^1 u_0(x)\phi_i(x)\,dx. \tag{2.14}$$

For each $i = 1, ..., N-1$, the equation (2.14) generates an equation which can be written as the matrix equation

$$M\boldsymbol{\alpha}(0) = G \tag{2.15}$$

where M is the same as (2.11) and $G_i = \langle u_0(x), \phi_i(x)\rangle$. Consequently, $\boldsymbol{\alpha}(0) = M^{-1}G$.

Combining (2.11) and (2.15) we obtain the initial value ODE system

$$\begin{aligned} M\dot{\boldsymbol{\alpha}}(t) + \frac{1}{2}\mathcal{B}\left(\boldsymbol{\alpha}(t)\right)\boldsymbol{\alpha}(t) + qC\boldsymbol{\alpha}(t) &= F(t) \\ M\boldsymbol{\alpha}(0) &= G. \end{aligned} \tag{2.16}$$

or

$$\begin{aligned} \dot{\boldsymbol{\alpha}}(t) &= M^{-1}\left[F(t) - \frac{1}{2}\mathcal{B}\left(\boldsymbol{\alpha}(t)\right)\boldsymbol{\alpha}(t) - qC\boldsymbol{\alpha}(t)\right] \\ \boldsymbol{\alpha}(0) &= M^{-1}G. \end{aligned} \tag{2.17}$$

Here, $\boldsymbol{\alpha}(t) = [\alpha_1(t), \alpha_2(t), \ldots, \alpha_{N-1}(t)]^T$.

2.2 Burgers' Equation with Neumann-Dirichlet Boundary Conditions

Now consider the second problem

$$u_t(t,x) + \frac{1}{2}\left[u(t,x)^2\right]_x - qu_{xx}(t,x) = f(t,x)\ , \quad x \in (0,1),\ t > 0, \tag{2.18}$$

where $x \in [0,1]$ and $t \in [0, t_f]$. The boundary conditions are given by

$$u_x(t,0) = \delta, \qquad u(t,1) = 0, \tag{2.19}$$

and the initial condition is

$$u(0,x) = u_0(x). \tag{2.20}$$

Again, multiply by a test function $v(x)$ and integrating we obtain

$$\int_0^1 \left(u_t(t,x) + \frac{1}{2}\left[u(t,x)^2\right]_x - qu_{xx}(t,x)\right)v(x)\,dx = \int_0^1 f(t,x)v(x)\,dx \tag{2.21}$$

or equivalently

$$\int_0^1 \left(u_t(t,x) + \frac{1}{2}\left[u(t,x)^2\right]_x\right)v(x)\,dx - q\int_0^1 u_{xx}(t,x)v(x)\,dx = \int_0^1 f(t,x)v(x)\,dx. \tag{2.22}$$

Also, if $v(\cdot)$ is piecewise smooth, then we can apply integration by parts on the second term on the left. We know from the right essential Dirichlet boundary condition that $u_x(t,1)v(1) = 0$ for all t and from the left boundary condition that $u_x(t,0) = \delta$. Hence, it follows that

$$\begin{aligned}\int_0^1 u_{xx}(t,x)v(x)\,dx &= u_x(t,x)v(x)\,|_0^1 - \int_0^1 u_x(t,x)v_x(x)\,dx \\ &= -\delta v(0) - \int_0^1 u_x(t,x)v_x(x)\,dx.\end{aligned}$$

From (2.22) we obtain the weak form of Burgers' equation with Neumann-Dirichlet boundary conditions given by

$$\begin{aligned}\int_0^1 \left(\left(u_t(t,x) + \frac{1}{2}\left[u(t,x)^2\right]_x\right)v(x) + qu_x(t,x)v_x(x)\right)\,dx & \\ +q\delta v(0) &= \int_0^1 f(t,x)v(x)\,dx, \quad (2.23)\end{aligned}$$

for any piecewise smooth function $v(x)$.

Using an approximation by piecewise linear basis functions we write $u(t,x) = \sum_{j=0}^N \alpha_j(t)\phi_j(x)$ and using the group approximation $u(t,x)^2 \approx \sum_{j=0}^N \alpha_j(t)^2\phi_j(x)$, equation (2.23) becomes

$$\begin{aligned}\int_0^1 \left(\left(\sum_{j=0}^N \dot{\alpha}_j(t)\phi_j(x)\right) + \frac{1}{2}\left[\sum_{j=0}^N \alpha_j(t)^2\phi_j(x)\right]_x\right)v(x) & \\ +q\left(\sum_{j=0}^N \alpha_j(t)\phi_j'(x)\right)v_x(x)\,dx + q\delta v(0) &= \int_0^1 f(t,x)v(x)\,dx. \quad (2.24)\end{aligned}$$

Since this holds for arbitrary piecewise smooth $v(x)$, we let $v(x) = \phi_i(x)$ to obtain

$$\begin{aligned}\int_0^1 \left(\left(\sum_{j=0}^N \dot{\alpha}_j(t)\phi_j(x)\right) + \frac{1}{2}\left[\sum_{j=0}^N \alpha_j(t)^2\phi_j(x)\right]_x\right)\phi_i(x) & \\ +q\left(\sum_{j=0}^N \alpha_j(t)\phi_j'(x)\right)\phi_i'(x)\,dx + q\delta\phi_i(0) &= \int_0^1 f(t,x)\phi_i(x)\,dx. \quad (2.25)\end{aligned}$$

Since for all j, $\alpha_j(t)$ does not depend on x we can move those terms outside the integral to produce

$$\begin{aligned}\sum_{j=0}^N \left(\dot{\alpha}_j(t)\int_0^1 \phi_j(x)\phi_i(x)\,dx + \frac{1}{2}\alpha_j(t)^2\int_0^1 \phi_j'(x)\phi_i(x)\,dx\right. & \\ \left.+q\alpha_j(t)\int_0^1 \phi_j'(x)\phi_i'(x)\,dx + q\delta\phi_i(0)\right) &= \int_0^1 f(t,x)\phi_i(x)\,dx. \quad (2.26)\end{aligned}$$

For each $i = 0, 1, ..., N$, equation (2.26) generates $N+1$ equations and hence we have a system of $N+1$ ODEs given by

$$M\dot{\boldsymbol{\alpha}}(t) + \frac{1}{2}\mathcal{B}\left(\boldsymbol{\alpha}(t)\right)\boldsymbol{\alpha}(t) + q\left(D + C\boldsymbol{\alpha}(t)\right) \quad = \quad F(t) \tag{2.27}$$

where $M_{ij} = \langle \phi_j(x), \phi_i(x) \rangle$, $C_{ij} = \left\langle \phi_j'(x), \phi_i'(x) \right\rangle$, $D = \delta\left[1,\, 0,\, \ldots,\, 0\right]^T$ and $F_i(t) = \langle f(t,x), \phi_i(x) \rangle$ and $\mathcal{B}\left(\boldsymbol{\alpha}(t)\right)$ is a matrix that depends on $\boldsymbol{\alpha}(t)$.

Taking advantage of the right Dirichlet boundary condition we know $\alpha_N(t) = 0$ for all t. Hence we can eliminate this equation and reduce the size of (2.27) from $N+1$ to N equations by solving only for the internal nodes. Hence, the $[N \times N]$ matrices M, C and $\mathcal{B}\left(\boldsymbol{\alpha}(t)\right)$ are given as

$$M = \frac{h}{6}\begin{pmatrix} 2 & 1 & & & \\ 1 & 4 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & 4 & 1 \\ & & & 1 & 4 \end{pmatrix}_{[N\times N]},$$

$$C = \frac{1}{h}\begin{pmatrix} 1 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}_{[N\times N]},$$

and

$$\mathcal{B}\left(\boldsymbol{\alpha}(t)\right) = \begin{pmatrix} -\frac{1}{2}\alpha_0(t) & \frac{1}{2}\alpha_1(t) & & & & \\ -\frac{1}{2}\alpha_0(t) & 0 & \frac{1}{2}\alpha_2(t) & & & \\ & \ddots & \ddots & \ddots & & \\ & & -\frac{1}{2}\alpha_{N-3}(t) & 0 & \frac{1}{2}\alpha_{N-1}(t) & \\ & & & -\frac{1}{2}\alpha_{N-2}(t) & 0 & \end{pmatrix}_{[N\times N]},$$

respectively. Again, note that during computation the term $\mathcal{B}\left(\boldsymbol{\alpha}(t)\right)\boldsymbol{\alpha}(t)$ is computed by

$$\mathcal{B}\left(\boldsymbol{\alpha}(t)\right)\boldsymbol{\alpha}(t) = \begin{pmatrix} -\frac{1}{2} & \frac{1}{2} & & & \\ -\frac{1}{2} & 0 & \frac{1}{2} & & \\ & \ddots & \ddots & \ddots & \\ & & -\frac{1}{2} & 0 & \frac{1}{2} \\ & & & -\frac{1}{2} & 0 \end{pmatrix}_{[N\times N]} \begin{bmatrix} \alpha_0(t)^2 \\ \alpha_1(t)^2 \\ \vdots \\ \alpha_{N-2}(t)^2 \\ \alpha_{N-1}(t)^2 \end{bmatrix}. \tag{2.28}$$

We also construct the initial condition $u_0(x)$ in terms of the basis functions, $\phi_i(x)$. Thus, as before we assume that $u_0(x) \approx u_0^N(x) = \sum_{j=0}^{N} \alpha_j(0)\phi_j(x)$ and obtain

$$\int_0^1 u_0^N(x)v(x)\,dx \quad = \quad \int_0^1 u_0(x)v(x)\,dx. \tag{2.29}$$

Using the approximation $u_0^N(x) = \sum_{j=0}^{N} \alpha_j(0)\phi_j(x)$ and enforcing the right essential Dirichlet boundary condition (2.29) becomes

$$\sum_{j=0}^{N-1} \alpha_j(0) \int_0^1 \phi_j(x)\phi_i(x)\,dx \;=\; \int_0^1 u_0(x)\phi_i(x)\,dx. \tag{2.30}$$

For each $i = 0, ..., N-1$, equation (2.30) generates an equation which can be written as the matrix equation

$$M\boldsymbol{\alpha}(0) \;=\; G \tag{2.31}$$

where M is the same as (2.27) and $G_i = \langle u_0(x), \phi_i(x)\rangle$. Hence, $\boldsymbol{\alpha}(0) = M^{-1}G$.

Combining (2.27) and (2.31) we obtain the initial value ODE system

$$\begin{aligned} M\dot{\boldsymbol{\alpha}}(t) + \frac{1}{2}\mathcal{B}\left(\boldsymbol{\alpha}(t)\right)\boldsymbol{\alpha}(t) + q\left(D + C\boldsymbol{\alpha}(t)\right) &= F(t) \\ M\boldsymbol{\alpha}(0) &= G. \end{aligned} \tag{2.32}$$

or

$$\begin{aligned} \dot{\boldsymbol{\alpha}}(t) &= M^{-1}\left[F(t) - \frac{1}{2}\mathcal{B}\left(\boldsymbol{\alpha}(t)\right)\boldsymbol{\alpha}(t) - q\left(D + C\boldsymbol{\alpha}(t)\right)\right] \\ \boldsymbol{\alpha}(0) &= M^{-1}G. \end{aligned} \tag{2.33}$$

2.3 Numerical Experiments

Using finite elements we approximate the true Burgers' equations by finite dimensional ODE systems. We will solve these ODE systems in MATLAB® to obtain the numerical solutions. The Method of Manufactured Solutions (MMS) is used to generate analytical solutions which are used to test convergence of the finite element methods. The method is briefly explained in the next section but extensive details can be found in the paper by Roache [52] or the book by Oberkampf [53]. The computational environment is provided in table 2.1 below.

Processor	INTEL® CORE ™ 2 Duo T9550
Processor Speed	2.66 GHz
Software	MATLAB® R2010b 64-bit Version 7.11.0.584
Machine Epsilon	2.2204×10^{-16}

Table 2.1: Computational Environment

2.3.1 Method of Manufactured Solutions

To ensure that the code produces accurate results and the approximations generated converge to a known solution we employ the Method of Manufactured Solutions (MMS). The MMS allows us to evaluate the error produced by numerical discretizations. The method develops exact solutions that are designed to test interactions in code and thereby verify the code. The easiest implementation requires the incorporation of a source/sink term into the code. Then we develop a nontrivial solution that has all the desired derivative terms and boundary conditions. We allow the source term to be the PDE operator applied to the desired exact solution. The choice of exact solution can be made totally independent of the governing equations and is assumed to meet the problem boundary conditions. To mitigate the evaluation error of the analytic source term, exact solutions should be independent of $u(t,x)$ and composed of simple functions or products of simple functions.

2.3.1.1 Burger's Equation MMS example

Consider Burger's equation operator given by

$$L\left[u(t,x)\right] := u_t(t,x) + \frac{1}{2}\left[u(t,x)^2\right]_x - q u_{xx}(t,x). \tag{2.34}$$

We want $\tilde{u}(t,x)$ to be our exact solution. Thus we apply the operator to $\tilde{u}(t,x)$ to obtain

$$L\left[\tilde{u}(t,x)\right] := \tilde{u}_t(t,x) + \frac{1}{2}\left[\tilde{u}(t,x)^2\right]_x - q\tilde{u}_{xx}(t,x) \tag{2.35}$$

by setting the $f(t,x) = L\left[\tilde{u}(t,x)\right]$ we guarantee that $\tilde{u}(t,x)$ is an exact solution to the modified equation

$$u_t(t,x) + \frac{1}{2}\left[u(t,x)^2\right]_x - q u_{xx}(t,x) = f(t,x). \tag{2.36}$$

Here, we assume the exact solution is of the form

$$\tilde{u}(t,x) \quad = \quad h(t)r(x). \tag{2.37}$$

Then the forcing term has the form

$$f(t,x) \quad = \quad \frac{d}{dt}h(t)r(x) + h(t)^2 r(x)\frac{d}{dx}r(x) - qh(t)\frac{d^2}{dx^2}r(x). \tag{2.38}$$

■

By comparing the approximate solution and the true solution we can determine the error for a particular discretization level. The error is calculated in the L^2 norm. In particular, if $f(t,x) \in L^2\left([0,T]\times[0,1], dt\times dx\right)$, the norm $\|f\|_{L^2}$ is given by

$$\|f\|_{L^2} = \left(\int_0^T \int_0^1 |f(t,x)|^2 \, dx \, dt \right)^{\frac{1}{2}}.$$

Therefore the relative error is given by

$$Err^{rel}(N) = \frac{\left\|u - u^N\right\|_{L^2}}{\|u\|_{L^2}}.$$

To evaluate the L^2 norm we utilize a 3 point gauss quadrature in space and for time we use the approximation

$$\int_0^T f(t) \, dt \approx \sum_{i=0}^{M} f(i)k$$

where $k = T/M$ and M is the number of times steps.

As the mesh spacing $h_N = 1/N$ is refined, standard finite element theory implies convergence to the true solution. If we assume the error for any mesh spacing has the form $e_{N_i} = Ch_{N_i}^p$, where C is a constant that does not depend on h, then we can calculate the observed order of convergence between two discretizations by

$$\begin{aligned} \ln \frac{e_{N_i}}{e_{N_{i+1}}} &= p \ln \frac{h_{N_i}}{h_{N_{i+1}}} \\ p &= \frac{\ln \frac{e_{N_i}}{e_{N_{i+1}}}}{\ln \frac{h_{N_i}}{h_{N_{i+1}}}}. \end{aligned}$$

2.3.2 MMS Simulations

In this section we will verify the code using the MMS found in the previous section. We present several experiments by changing the parameter value for both the Dirichlet boundary conditions and Neumann-Dirichlet boundary conditions problems. We also test several of the built-in MATLAB® ODE solvers for accuracy and performance using the default settings.

2.3.2.1 Dirichlet Boundary Conditions Problem

Consider the problem with Dirichlet boundary conditions and exact solution given by

$$u(t,x) = e^{-t} \sin(\pi x). \tag{2.39}$$

The initial condition is given by

$$u_0(x) = \sin(\pi x). \tag{2.40}$$

This meets the boundary conditions and the associated MMS forcing term is

$$f(t,x) \quad = \quad -e^{-t}\sin(\pi x) + \pi e^{-2t}\sin(\pi x)\cos(\pi x) + q\pi^2 e^{-t}\sin(\pi x). \tag{2.41}$$

Results for various parameter values and solvers are found in Tables 2.2, 2.3, 2.4. In general, ODE45 is accurate but requires a significant amount of time for a fine mesh. We also see that ODE15s is very fast but the errors may actually grow as the mesh is refined. Specifically, in Table 2.2, ODE45 and ODE23 demonstrate monotone convergence while ODE15s actually produced an increase in the global error as the mesh was refined from $N = 64$ to $N = 128$. In Table 2.3, ODE45 and ODE23 again demonstrate monotone convergence with ODE45 producing the more accurate global solution, while the convergence of ODE15s stalls around $N = 32$ and the global error increases as the mesh is refined from $N = 64$ to $N = 128$. In Table 2.4, ODE45 and ODE23 again demonstrate monotone convergence but here we see that the convergence of ODE23 stalls around $N = 64$. Again we note using ODE15s the global error increases as the mesh is refined from $N = 64$ to $N = 128$.

Sample plots of the exact solution, the FEM solution and errors are found in Figures 2.1, 2.2, 2.3. In Figure 2.1 we compare the exact solution to the finite element numerical approximation for 16 elements when $q = 1/100$, we note the visual agreement between them. Figure 2.2 displays the error between the exact solution and the numerical solution. The global L^2 error is less than 4×10^{-3} in time and space. As shown in Figure 2.3, using $N = 16$ elements the discretization error is small and our computed numerical solution matches the exact analytic solution.

MATLAB® Solver	Number of Elements	Solver time	Err^{rel}	Order
ODE45	$N = 8$	0.267	1.327×10^{-2}	-
	$N = 16$	1.597	3.473×10^{-3}	1.9335
	$N = 32$	13.24	9.194×10^{-4}	1.9175
	$N = 64$	122.7	2.427×10^{-4}	1.9218
	$N = 128$	1004	1.103×10^{-4}	1.3171
ODE23	$N = 8$	0.132	1.310×10^{-2}	-
	$N = 16$	1.030	3.310×10^{-3}	1.9845
	$N = 32$	8.182	8.780×10^{-4}	1.9113
	$N = 64$	74.99	3.361×10^{-4}	1.3886
	$N = 128$	615.3	1.510×10^{-4}	1.1542
ODE15s	$N = 8$	0.181	1.384×10^{-2}	-
	$N = 16$	0.204	3.156×10^{-3}	2.1329
	$N = 32$	0.486	1.392×10^{-3}	1.1810
	$N = 64$	1.466	1.371×10^{-3}	0.0222
	$N = 128$	4.590	1.443×10^{-3}	-0.074

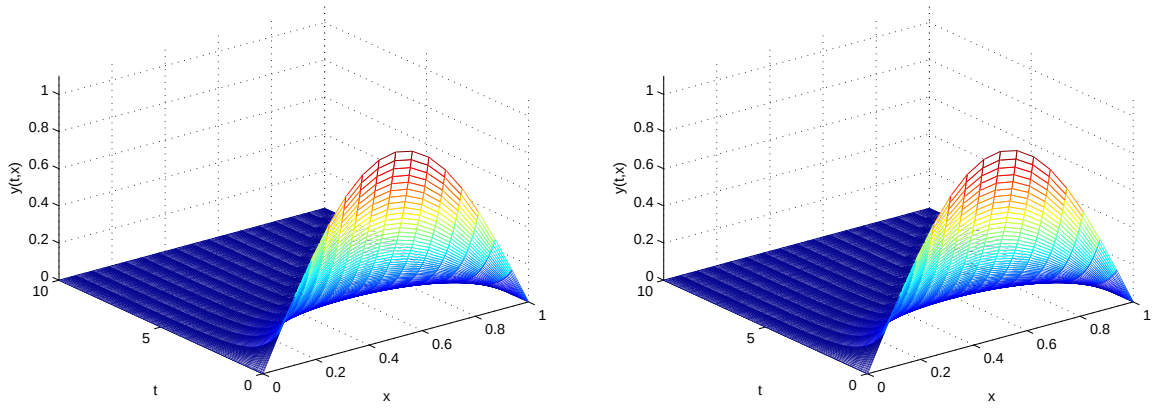
Table 2.2: FEM Results for $T = 10$, $Re = 50$ and Dirichlet boundary conditions

MATLAB® Solver	Number of Elements	Solver time	Err^{rel}	Order
ODE45	$N = 8$	0.119	1.808×10^{-2}	-
	$N = 16$	0.875	4.643×10^{-3}	1.9610
	$N = 32$	6.316	1.244×10^{-3}	1.9007
	$N = 64$	60.45	3.166×10^{-4}	1.9735
	$N = 128$	491.7	8.767×10^{-5}	1.8526
ODE23	$N = 8$	0.075	1.802×10^{-2}	-
	$N = 16$	0.512	4.640×10^{-3}	1.9574
	$N = 32$	4.027	1.507×10^{-3}	1.6228
	$N = 64$	37.07	4.882×10^{-4}	1.6257
	$N = 128$	303.6	1.177×10^{-4}	2.0521
ODE15s	$N = 8$	0.102	1.900×10^{-2}	-
	$N = 16$	0.231	4.310×10^{-3}	2.1403
	$N = 32$	0.560	1.614×10^{-3}	1.4174
	$N = 64$	1.505	1.592×10^{-3}	0.0194
	$N = 128$	4.746	1.695×10^{-3}	-0.090

Table 2.3: FEM Results for $T = 10$, $Re = 100$ and Dirichlet boundary conditions

MATLAB® Solver	Number of Elements	Solver time	Err^{rel}	Order
ODE45	$N = 8$	0.072	2.303×10^{-2}	-
	$N = 16$	0.339	6.000×10^{-3}	1.9406
	$N = 32$	2.585	1.662×10^{-3}	1.8517
	$N = 64$	23.53	4.399×10^{-4}	1.9179
	$N = 128$	197.6	1.131×10^{-4}	1.9595
ODE23	$N = 8$	0.075	2.293×10^{-2}	-
	$N = 16$	0.221	6.245×10^{-3}	1.8763
	$N = 32$	1.615	2.428×10^{-3}	1.3628
	$N = 64$	14.56	1.098×10^{-3}	1.1453
	$N = 128$	121.2	7.128×10^{-4}	0.6232
ODE15s	$N = 8$	0.103	2.404×10^{-2}	-
	$N = 16$	0.239	6.537×10^{-3}	1.8788
	$N = 32$	0.660	2.974×10^{-3}	1.1363
	$N = 64$	1.822	2.197×10^{-3}	0.4369
	$N = 128$	6.312	2.300×10^{-3}	$-.0659$

Table 2.4: FEM Results for $T = 10$, $Re = 250$ and Dirichlet boundary conditions



(a) Exact solution

(b) FEM solution

Figure 2.1: Exact solution and FEM using ODE45 for $N = 16$, $Re = 100$ and Dirichlet boundary conditions

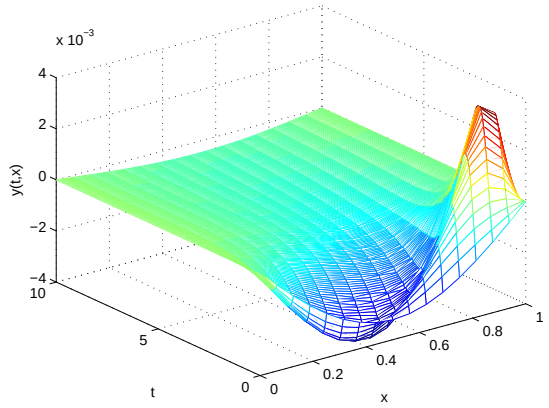


Figure 2.2: FEM error using ODE45 for $N = 16$, $Re = 100$ and Dirichlet boundary conditions

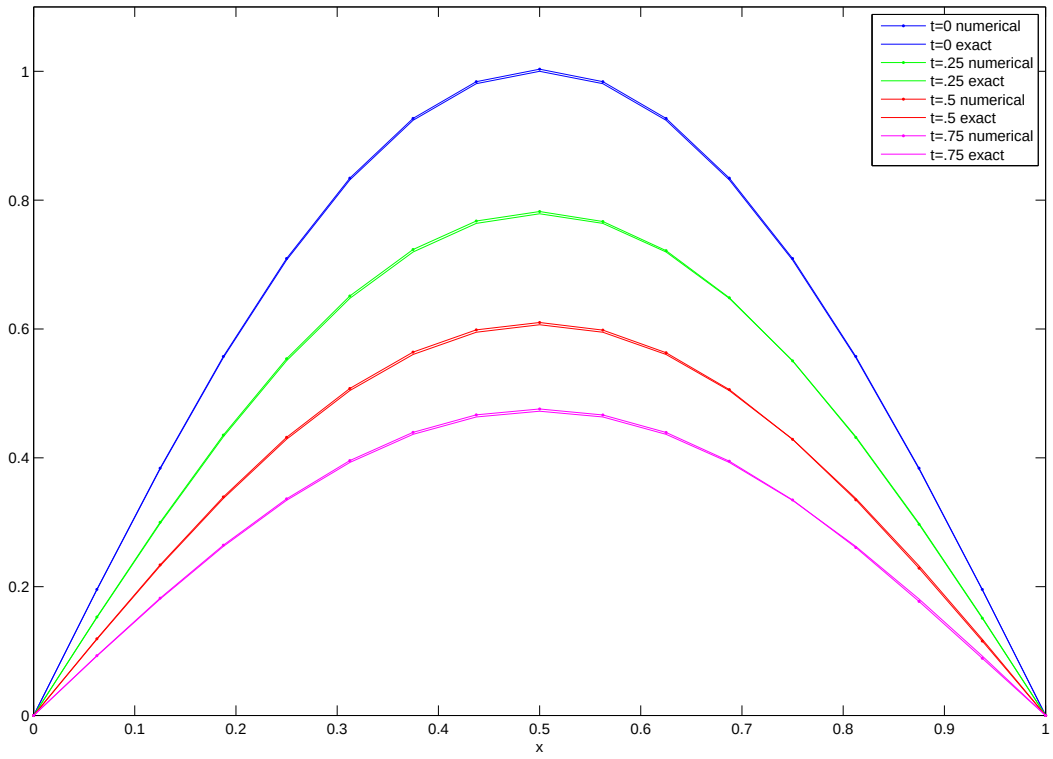


Figure 2.3: Cross section of exact and FEM using ODE45 for $N = 16$, $Re = 100$ and Dirichlet boundary conditions

2.3.2.2 Neumann-Dirichlet Boundary Conditions Problem

Consider the problem with Neumann-Dirichlet boundary conditions, $\delta = 0$ and exact solution given by

$$u(t,x) \quad = \quad e^{-t}(1-x^2). \tag{2.42}$$

The initial condition is given by

$$u_0(x) = (1-x^2). \tag{2.43}$$

This meets the boundary conditions and the associated MMS forcing term is

$$f(t,x) \quad = \quad -e^{-t}\left(1-x^2\right) - e^{-2t}2x\left(1-x^2\right) + 2qe^{-t}. \tag{2.44}$$

Results for various parameter values and solvers are found in Tables 2.5, 2.6, 2.7. We see again that for all values of q, ODE45 produced convergent solutions as the spatial mesh is refined. Specifically, in Table 2.5, ODE45 and ODE23 demonstrate monotone convergence while ODE15s actually produced an increase in the global error as the mesh was refined from $N = 32$ to $N = 64$. In Table 2.6, ODE45 and ODE23 again demonstrate monotone convergence producing comparable global L^2 errors, while for ODE15s the global error increases as the mesh is refined from $N = 64$ to $N = 128$. In Table 2.7, all solvers demonstrate monotone convergence, though in all cases the global L^2 error is larger, than when q is larger. Also we note that though ODE15s has monotone convergence the observed order of accuracy is lower than ODE45 and ODE23.

Sample plots of the exact solution, the FEM solution and errors are found in Figures 2.4, 2.5, 2.6. In Figure 2.4 we compare the exact solution to the finite element numerical approximation for 16 elements when $q = 1/100$, we note the visual agreement between them. Figure 2.5 clearly shows that the global error for the Neumann-Dirichlet boundary condition problem is much larger than for the Dirichlet boundary condition problem. Furthermore, the largest error occurs at the Neumann boundary. In Figure 2.6, we see that as time increases the numerical solution produces the largest error at the Neumann boundary.

MATLAB® Solver	Number of Elements	Solver time	Err^{rel}	Order
ODE45	$N = 8$	0.252	1.779×10^{-1}	-
	$N = 16$	1.647	4.282×10^{-2}	2.0548
	$N = 32$	13.40	1.061×10^{-2}	2.0126
	$N = 64$	120.8	2.649×10^{-3}	2.0025
	$N = 128$	1040	6.640×10^{-4}	1.9660
ODE23	$N = 8$	0.143	1.771×10^{-1}	-
	$N = 16$	1.085	4.280×10^{-2}	2.0492
	$N = 32$	8.443	1.061×10^{-2}	2.0115
	$N = 64$	76.53	2.653×10^{-3}	2.0004
	$N = 128$	655.9	6.837×10^{-4}	1.9560
ODE15s	$N = 8$	0.173	1.832×10^{-1}	-
	$N = 16$	0.200	4.351×10^{-2}	2.0736
	$N = 32$	0.573	9.921×10^{-3}	2.1328
	$N = 64$	1.500	1.364×10^{-2}	-0.460
	$N = 128$	4.860	1.029×10^{-2}	0.4073

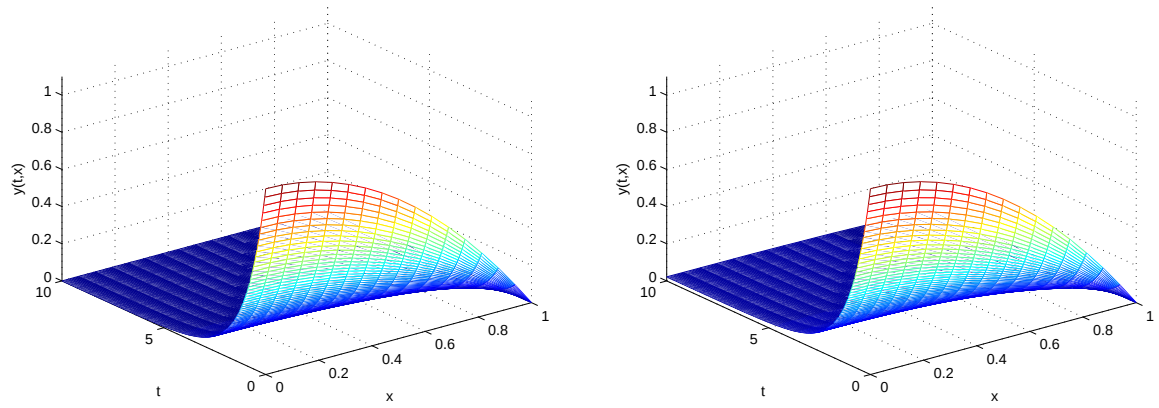
Table 2.5: FEM Results for $T = 10$, $Re = 50$ and Neumann-Dirichlet boundary conditions

MATLAB® Solver	Number of Elements	Solver time	Err^{rel}	Order
ODE45	$N = 8$	0.127	3.998×10^{-1}	-
	$N = 16$	0.886	9.495×10^{-2}	2.0741
	$N = 32$	6.807	2.338×10^{-2}	2.0217
	$N = 64$	61.52	5.823×10^{-3}	2.0055
	$N = 128$	523.7	1.455×10^{-3}	2.0009
ODE23	$N = 8$	0.080	3.984×10^{-1}	-
	$N = 16$	0.545	9.447×10^{-2}	2.0762
	$N = 32$	4.203	2.338×10^{-2}	2.0149
	$N = 64$	37.95	5.823×10^{-3}	2.0050
	$N = 128$	322.0	1.460×10^{-3}	1.9958
ODE15s	$N = 8$	0.100	4.068×10^{-1}	-
	$N = 16$	0.203	1.004×10^{-1}	2.0188
	$N = 32$	0.460	2.255×10^{-2}	2.1543
	$N = 64$	1.891	1.219×10^{-2}	0.8877
	$N = 128$	6.446	1.543×10^{-2}	-0.340

Table 2.6: FEM Results for $T = 10$, $Re = 100$ and Neumann-Dirichlet boundary conditions

MATLAB® Solver	Number of Elements	Solver time	Err^{rel}	Order
ODE45	$N = 8$	0.094	1.030×10^{0}	-
	$N = 16$	0.340	2.555×10^{-1}	2.0111
	$N = 32$	2.662	6.229×10^{-2}	2.0365
	$N = 64$	24.46	1.546×10^{-2}	2.0104
	$N = 128$	207.4	3.858×10^{-3}	2.0026
ODE23	$N = 8$	0.080	1.028×10^{0}	-
	$N = 16$	0.224	2.549×10^{-1}	2.0123
	$N = 32$	1.702	6.220×10^{-2}	2.0348
	$N = 64$	15.22	1.545×10^{-2}	2.0091
	$N = 128$	126.8	3.859×10^{-3}	2.0016
ODE15s	$N = 8$	0.111	1.029×10^{0}	-
	$N = 16$	0.205	2.605×10^{-1}	1.9820
	$N = 32$	0.462	6.447×10^{-2}	2.0148
	$N = 64$	1.819	2.738×10^{-2}	1.2355
	$N = 128$	6.334	1.123×10^{-2}	1.2863

Table 2.7: FEM Results for $T = 10$, $Re = 250$ and Neumann-Dirichlet boundary conditions



(a) Exact solution

(b) FEM solution

Figure 2.4: Exact solution and FEM using ODE45 for $N = 16$, $Re = 100$ and Neumann-Dirichlet boundary conditions

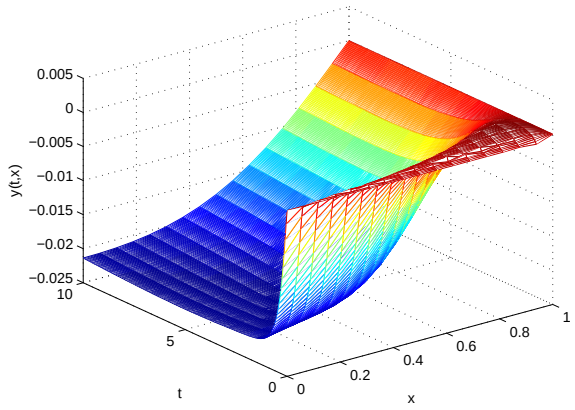


Figure 2.5: FEM error using ODE45 for $N = 16$, $Re = 100$ and Neumann-Dirichlet boundary conditions

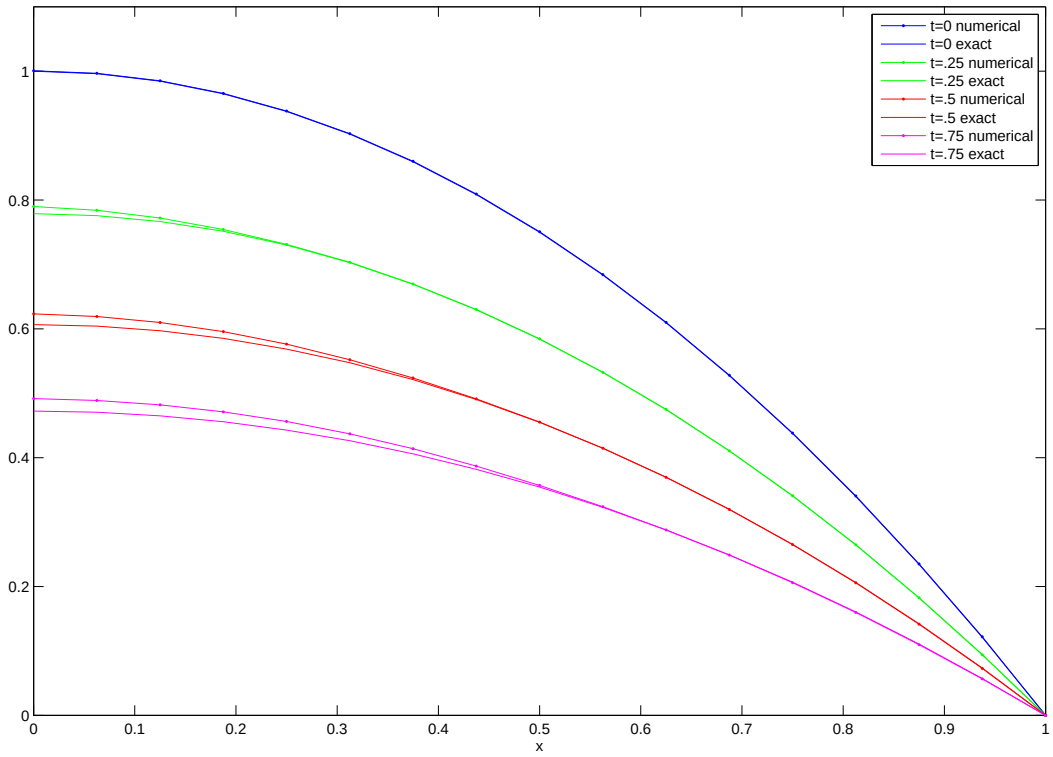


Figure 2.6: Cross section comparison of exact solution and FEM using ODE45 for $N = 16$, $Re = 100$ and Neumann-Dirichlet boundary conditions

Chapter 3

Proper Orthogonal Decomposition

Proper Orthogonal Decomposition (POD) is also known as the Method of Snapshots, Principal Component Analysis, Karhunen-Love decomposition and other names. There are many papers that describe the POD method. However, [19–21] provide a good introduction to the method. The general idea is to generate a set of orthonormal vectors that represent the data optimally in a mean-squared error sense. We note that since the approximation space is spanned by the piecewise linear basis functions, any set of vectors that spans a subspace also can be expressed as linear combinations of the piecewise linear basis functions. As such, in most cases we lose the local support of our original finite element basis functions in exchange for a smaller dense system. We also point out that the L^2 inner product of any pair of FEM basis functions results in a value contained in the mass matrix M. Therefore given any functions $v_1(x), v_2(x)$ that are linear combinations of the piecewise linear basis functions $\phi_i(x)$,

$$\begin{aligned} v_1(x) &= \sum_{i=0}^{N} c_i \phi_i(x) \\ v_2(x) &= \sum_{j=0}^{N} d_j \phi_j(x) \end{aligned}$$

then

$$\begin{aligned} \langle v_1(x), v_2(x) \rangle_{L^2} &= \left\langle \sum_{i=0}^{N} c_i \phi_i(x), \sum_{j=0}^{N} d_j \phi_j(x) \right\rangle_{L^2} \\ &= \sum_{i=0}^{N} \sum_{j=0}^{N} c_i d_j \int_0^1 \phi_i(x) \phi_j(x)\, dx \\ &= \tilde{v}_1^T M \tilde{v}_2 \\ &= \langle \tilde{v}_1, \tilde{v}_2 \rangle_M \end{aligned} \tag{3.1}$$

where $\tilde{v}_j$ is the vector of nodal coefficients associated with the piecewise linear basis function

expansion of $v_j(x)$. Therefore the L^2 inner product of the piecewise continuous functions in our FEM space can be replaced by the weighted $\mathbb{R}^n$ vector inner product, with weight given by the mass matrix M.

3.1 POD Basis

Let $Y_{[n\times m]}$ be a real data matrix composed of m time snapshots of $n \times 1$ vectors containing the spatial information resulting from our finite element approximation. Since the mass matrix is symmetric, real and positive definite there exists a Cholesky decomposition, namely there exists a lower triangular matrix L such that $M = LL^T$. Define $\tilde{Y} = L^T Y$ as the weighted snapshot matrix. We note since M is nonsingular, so is L and hence if Y has rank k then so does $\tilde{Y}$.

We wish to find a unit vector in our FEM space that minimizes the mean squared error, this is equivalent to finding a vector ϕ that satisfies

$$\max \sum_{j=1}^{m} \left| \langle y_j, \phi \rangle_M \right|^2 \qquad \text{s.t. } \|\phi\|_M = 1. \tag{3.2}$$

This has been shown [21, 22, 25] to be equivalent to finding the eigenvectors such that

$$\tilde{Y}\tilde{Y}^T\psi \;=\; \Lambda\psi. \tag{3.3}$$

This solution can be obtained using the Singular Value Decomposition (SVD) of $\tilde{Y}$. In general, SVD guarantees that any $n \times m$ matrix A can be decomposed into

$$A \;=\; U\Sigma V^T \tag{3.4}$$

where U is unitary $n \times n$, Σ is diagonal $n \times m$ and V^T is unitary $m \times m$. Also U and V are solutions to

$$AA^TU \;=\; \Lambda_n U \tag{3.5}$$
$$A^TAV \;=\; \Lambda_m V \tag{3.6}$$

where $\Sigma\Sigma^T = \Lambda_n$ and $\Sigma^T\Sigma = \Lambda_m$. Then using SVD we can decompose $\tilde{Y}$ as

$$\tilde{Y} \;=\; U\Sigma V^T \tag{3.7}$$

and the vectors U_i satisfy

$$\max \sum_{j=1}^{m} \left| \langle \tilde{y}_j, U_i \rangle_{\mathbb{R}^n} \right|^2 \qquad \text{s.t. } \|U_i\|_{\mathbb{R}^n} = 1 \text{ and } \langle U_i, U_k \rangle_{\mathbb{R}^n} = 0 \text{ for } k = 1, ..., i-1. \tag{3.8}$$

Since Y has rank k for $i = 1, ..., k$, only the Σ_{ii} entries are nonzero so we can express $\tilde{Y} = U^k \Sigma^k \left(V^k\right)^T$ where the superscript k indicates the first k vectors of the matrices U, V and the $k \times k$ principal submatrix of Σ.

Given any integer $r \leq k$ we find the L^2 spatial POD basis vectors by taking $\phi_i^r(x) = L^{-T} U_i(x)$ for $i = 1, \ldots, r$. We denote the collection of POD basis vectors as Φ^r. We note that by construction that

$$\begin{aligned} \left\langle \phi_i^r(x), \phi_j^r(x) \right\rangle_{L^2} &= \left\langle \phi_i^r, \phi_j^r \right\rangle_M \\ &= \left(\phi_i^r\right)^T M \phi_j^r \\ &= \left(\phi_i^r\right)^T L L^T \phi_j^r \\ &= \left(L^T \phi_i^r\right)^T \left(L^T \phi_j^r\right) \\ &= U_i^T U_j \\ &= \delta_{ij}. \end{aligned} \tag{3.9}$$

Therefore the vectors $\{\phi_i^r\}_{i=1}^r$ form an orthonormal set and a basis for our subspace. As mentioned before the POD basis vectors are a linear combination of the original basis functions. This can be seen explicitly from Φ^r, for each $i = 1, ..., r$ then the POD basis vectors can be expanded using the hat functions by

$$\phi_i^r(x) = \sum_{k=0}^{N} \Phi_{ki}^r \phi_k(x). \tag{3.10}$$

The particular choice for r determines the ratio of computational time saved to amount of error accepted. One method to aid in the selection for r is based on the ranking of eigenvalues associated with the vectors used in the basis. Since the vectors U_i satisfy (3.8), the singular values in Σ are also ordered by value. Hence, we can compute the amount of "energy" captured by the POD basis vectors by

$$\mathcal{E}(r) = \frac{\sum_{i=1}^r \Sigma_{ii}^2}{\sum_{i=1}^k \Sigma_{ii}^2} \tag{3.11}$$

where $k = \text{rank } \tilde{Y}$.

3.2 Reduced Order Model

To use the POD basis we start again with the weak form of Burgers' equation and expand the solution in terms of the POD basis. Using the approximation for $u^N(t, x) \approx u(t, x)$ in the space spanned by the POD basis functions given by

$$u^N(t, x) = \sum_{j=1}^{r} \alpha_j^r(t) \phi_j^r(x).$$

Then we wish reduce the order of our ODE systems from $N-1$ or N equations depending on boundary conditions to r. This can be accomplished by solving only for the coefficients of the POD basis vectors α_j^r at each time step.

3.2.1 Reduced Order Model for Burgers' Equation with Dirichlet Boundary Conditions

Recall the weak form of Burgers' equation in conservation form with Dirichlet boundary conditions is given by equation (2.7)

$$\int_0^1 \left(\left(u_t(t,x) + \frac{1}{2}\left[u(t,x)^2\right]_x \right) v(x) + q u_x(t,x) v_x(x) \right) dx \quad = \quad \int_0^1 f(t,x) v(x)\, dx. \tag{3.12}$$

Here we now expand the solution in terms of the POD basis functions by using the approximation

$$u(t,x) \approx u^N(t,x) = \sum_{j=1}^{r} \alpha_j^r(t)\phi_j^r(x). \tag{3.13}$$

Therefore, (3.12) becomes

$$\int_0^1 \left(\left(\sum_{j=1}^{r} \dot{\alpha}_j^r(t)\phi_j^r(x) \right) + \frac{1}{2}\left[\left(\sum_{j=1}^{r} \alpha_j^r(t)\phi_j^r(x) \right)^2 \right]_x \right) v(x) + q \left(\sum_{j=1}^{r} \alpha_j^r(t)\phi_{j,x}^r(x) \right) v_x(x)\, dx = \int_0^1 f(t,x)v(x)\, dx \tag{3.14}$$

where the notation $\phi_{j,x}^r(x)$ means $\left[\phi_j^r(x)\right]_x$. Since this holds for any piecewise smooth $v(x)$, for each $i = 1, ..., r$ set $v(x) = \phi_i^r(x)$. Then, (3.14), for each i we obtain

$$\int_0^1 \left(\left(\sum_{j=1}^{r} \dot{\alpha}_j^r(t)\phi_j^r(x) \right) + \frac{1}{2}\left[\left(\sum_{j=1}^{r} \alpha_j^r(t)\phi_j^r(x) \right)^2 \right]_x \right) \phi_i^r(x) + q \left(\sum_{j=1}^{r} \alpha_j^r(t)\phi_j^r, x(x) \right) \phi_{i,x}^r(x)\, dx = \int_0^1 f(t,x)\phi_i^r(x)\, dx. \tag{3.15}$$

For the linear terms, $\alpha(\cdot)$ depends only on t we can move those terms outside the integral so that

(3.15) becomes

$$\sum_{j=1}^{r} \dot{\alpha}_j^r(t) \int_0^1 \phi_j^r(x)\phi_i^r(x)\,dx + \frac{1}{2}\int_0^1 \left[\left(\sum_{j=1}^{r} \alpha_j^r(t)\phi_j^r(x)\right)^2\right]_x \phi_i^r(x)\,dx$$
$$+q\sum_{j=1}^{r} \alpha_j^r(t)\int_0^1 \phi_{j,x}^r(x)\phi_{i,x}^r(x)\,dx = \int_0^1 f(t,x)\phi_i^r(x)\,dx. \tag{3.16}$$

Looking at the linear terms we define $M_{ij}^r = \left\langle \phi_j^r(x), \phi_i^r(x)\right\rangle$, $C_{ij}^r = \left\langle \phi_{j,x}^r(x), \phi_{i,x}^r(x)\right\rangle$ and $F_i^r(t) = \langle f(t,x), \phi_i^r(x)\rangle$. We note that as demonstrated in (3.9), when the basis is orthonormal, $M^r = I_r$ the identity of dimension r. Also we use the expansion of the POD basis vectors in C^r to obtain

$$\begin{aligned} C_{ij}^r &= \left\langle \phi_{j,x}^r(x), \phi_{i,x}^r(x)\right\rangle \qquad (3.17)\\ &= \int_0^1 \phi_{j,x}^r(x)\phi_{i,x}^r(x)\,dx \\ &= \int_0^1 \sum_{k=0}^{N} \Phi_{kj}^r \phi_{k,x}(x) \sum_{l=0}^{N} \Phi_{li}^r \phi_{l,x}(x)\,dx \\ &= \sum_{k=0}^{N}\sum_{l=0}^{N} \Phi_{kj}^r \Phi_{li}^r \int_0^1 \phi_{j,x}(x)\phi_{i,x}(x)\,dx \end{aligned}$$

Therefore we see that we can write the reduced matrix as in terms of the POD basis coefficients and the full dimensional C matrix. The result is

$$C^r = (\Phi^r)^T C \Phi^r. \tag{3.18}$$

Now we need to handle the nonlinear term. A simple approach is given for consistency. If we assume solution $u(t,x)$ can be expressed using the FEM basis $\phi_0(x), ..., \phi_N(x)$ and coefficients $\alpha_0(t), ..., \alpha_N(t)$ and the POD basis $\phi_1^r(x), ..., \phi_r^r(x)$ and coefficients $\alpha_1^r(t), ..., \alpha_r^r(t)$ then we obtain

$$\sum_{j=0}^{N} \alpha_j(t)\phi_j(x) \approx u(t,x) \approx \sum_{k=1}^{r} \alpha_k^r(t)\phi_k^r(x). \tag{3.19}$$

Since the POD basis functions are a linear combination of the FEM basis functions as in (3.10) so we see that

$$\sum_{j=0}^{N} \alpha_j(t)\phi_j(x) \approx u(t,x) \approx \sum_{k=1}^{r} \alpha_k^r(t) \sum_{l=0}^{N} \alpha_l(t)\phi_l(x) \tag{3.20}$$

or in matrix form

$$\boldsymbol{\alpha}(t) \approx \Phi^r \boldsymbol{\alpha}^r(t) \tag{3.21}$$

and the reverse relationship holds

$$\boldsymbol{\alpha}^r(t) \approx (\Phi^r)^T \boldsymbol{\alpha}(t). \tag{3.22}$$

Therefore since the $n \times 1$ FEM nonlinear term is given in (2.11) as $\frac{1}{2}\mathcal{B}(\boldsymbol{\alpha}(t))\boldsymbol{\alpha}(t)$ we can find an $r \times 1$ reduced order approximation using the POD basis functions using the above relationships to give

$$\frac{1}{2}(\Phi^r)^T \mathcal{B}(\Phi^r \boldsymbol{\alpha}^r(t)) \Phi^r \boldsymbol{\alpha}^r(t). \tag{3.23}$$

The initial conditions must also be found in terms of our POD basis functions. Again using that the POD basis functions are a linear combination of the hat functions with coefficients given by Φ^r we can find the POD basis initial coefficients $\boldsymbol{\alpha}^r(0)$ by using the POD coefficient matrix and our FEM initial condition vector as $\boldsymbol{\alpha}^r(0) = (\Phi^r)^T\boldsymbol{\alpha}(0)$.

Therefore the POD reduced order model is given by

$$\begin{aligned} \dot{\boldsymbol{\alpha}}^r(t) &= \left[F^r(t) - \frac{1}{2}(\Phi^r)^T \mathcal{B}(\Phi^r\boldsymbol{\alpha}^r(t))\Phi^r\boldsymbol{\alpha}^r(t) - qC^r\alpha^r(t)\right] \\ \boldsymbol{\alpha}^r(0) &= (\Phi^r)^T\boldsymbol{\alpha}(0). \end{aligned} \tag{3.24}$$

We note that at every time step we are only solving for an $r \times 1$ vector.

3.2.2 Reduced Order Model for Burgers' Equation with Neumann-Dirichlet Boundary Conditions

Just as before we begin with the weak form of the Neumann-Dirichlet boundary conditions Problem given in (2.23). From this we see the only difference we have is given by the Neumann boundary condition on the left. In the Finite Element discretization we obtain the term $q\delta v(0)$. Then due to the local support of the hat functions this term became the vector given by $q\delta\,[1,\,0,\,\ldots,\,0]^T$. Recall the POD basis functions now have global support, therefore for each $i = 0, ..., r$ we let $v(0) = \phi_i^r(0)$ so the term $q\delta v(0)$ generates the vector $D^r = q\delta\,[\phi_1^r(0),\,\phi_2^r(0),\,\ldots,\phi_r^r(0)]^T$. Again, using (3.22), D^r can be written as

$$D^r = q(\Phi^r)^T\,[1,\,0,\,\ldots,\,0]^T \tag{3.25}$$

Therefore the resulting final system is given by

$$\begin{aligned} \dot{\boldsymbol{\alpha}}^r(t) &= \left[F^r(t) - \frac{1}{2}(\Phi^r)^T \mathcal{B}(\Phi^r\boldsymbol{\alpha}^r(t))\Phi^r\boldsymbol{\alpha}^r(t) - q\left(D^r + C^r\boldsymbol{\alpha}^r(t)\right)\right] \\ \boldsymbol{\alpha}^r(0) &= (\Phi^r)^T\boldsymbol{\alpha}(0). \end{aligned} \tag{3.26}$$

We note that at every time step we are only solving for an $r \times 1$ vector.

3.3 Reduced Order Model Numerical Experiments

To test the accuracy and the computational advantages of the ROM we will calculate a solution for the same numerical experiments used in §2.3. We use the FEM to generate the snapshot data and generate the POD basis from the data. The reduced initial condition is found and the reduced order ODE systems are solved as the MMS experiments. Additionally, since the MMS solutions presented are functions that decay over time we would like an initial condition and forcing function pair that test the ability of the ROM to replicate any solution generated using the FEM. Therefore we will test two problems for each boundary condition type.

3.3.1 Dirichlet Boundary Conditions Problem #1 ROM

Recall from §2.3.2.1, the problem with Dirichlet boundary conditions and exact solution given by

$$u(t,x) \quad = \quad e^{-t}\sin(\pi x). \tag{3.27}$$

The initial condition is given by

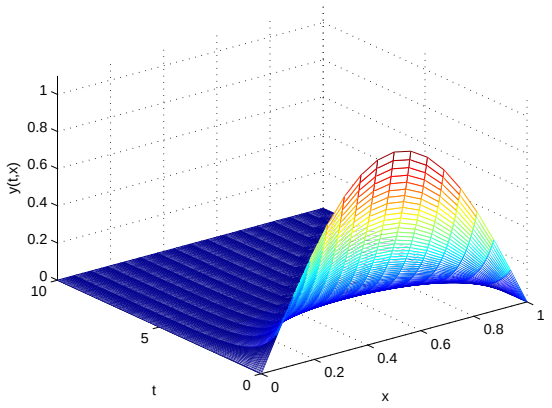
$$u_0(x) = \sin(\pi x). \tag{3.28}$$

This meets the boundary conditions and the associated MMS forcing term is

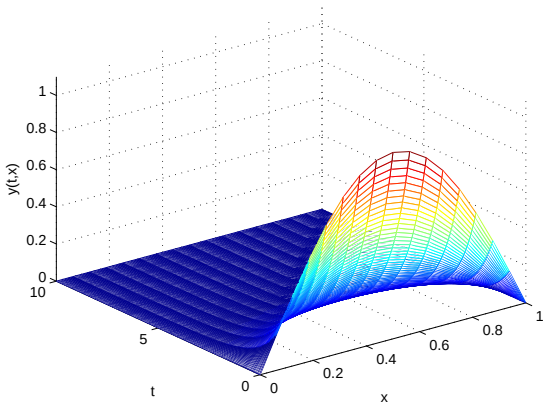
$$f(t,x) \quad = \quad -e^{-t}\sin(\pi x) + \pi e^{-2t}\sin(\pi x)\cos(\pi x) + q\pi^2 e^{-t}\sin(\pi x). \tag{3.29}$$

We implement the system shown in (3.24) using a POD dimension $r = 5$ and the results are shown in Figures 3.1, 3.2, 3.3. In particular, in Figure 3.1, we see good high level agreement of the POD model to the full discretization numerical solution and the exact solution. In Figure 3.2, we compare the error between the FEM numerical solution and the exact solution to the error between the POD model numerical solution error from the exact solution. Here we see almost an exact match, with very small differences relating to the smoothness of the errors. In Figure 3.3, we compare the error between the FEM numerical solution and the exact solution to the error between the POD model numerical solution error from the exact solution for some particular time instances. Here, there is no discernable difference between the errors.

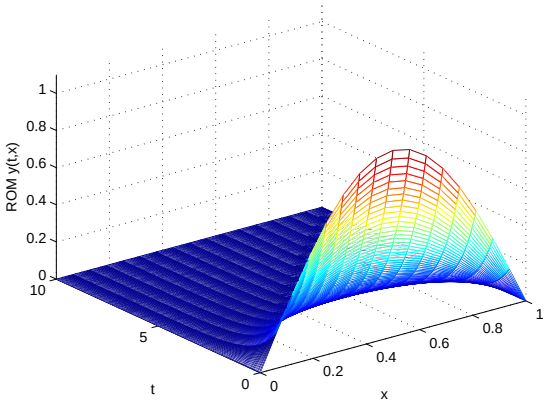
Due to the use of the manufactured solutions, almost all the data can be expressed through the use of only one function that decays over time. This is evident by inspection of the basis coefficients at each time step. On average the first POD basis vector accounts for 96% of the solution at each time step. The POD basis provides a very accurate reduced order model for this problem.



(a) Exact solution

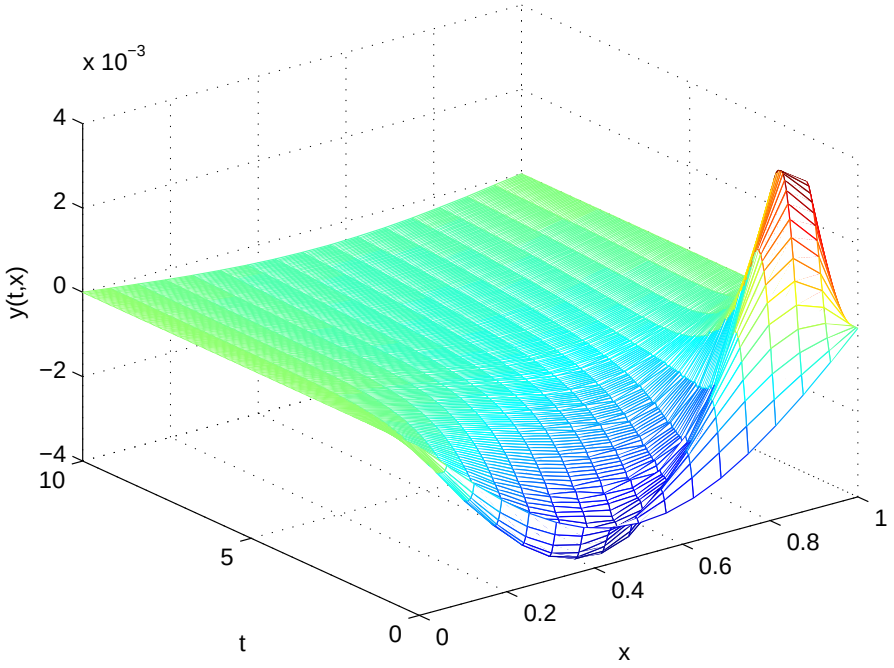


(b) FEM solution

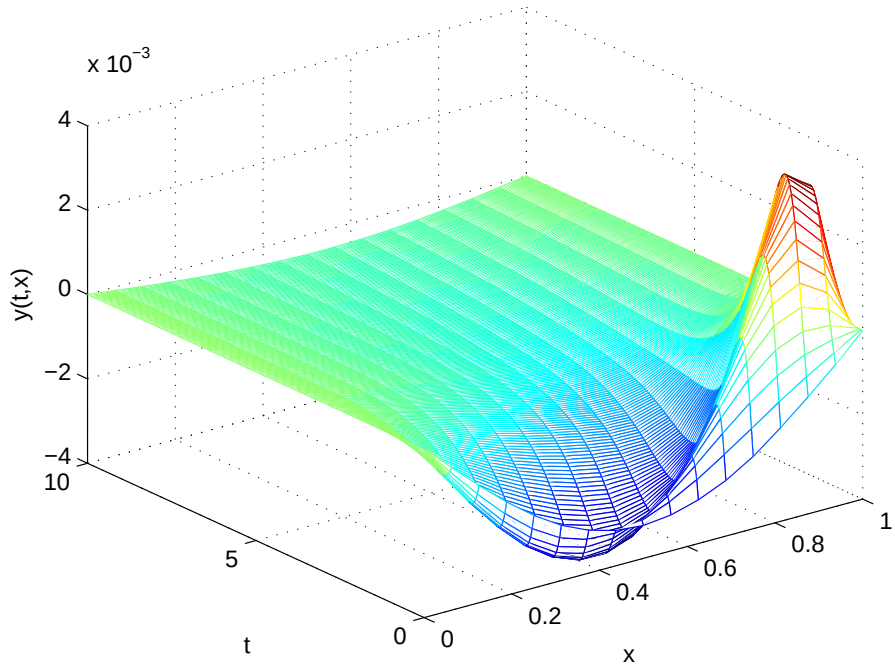


(c) POD solution, $r = 5$

Figure 3.1: Exact solution, FEM, and ROM using ODE45 for $N = 16$, $Re = 100$ and Dirichlet boundary conditions

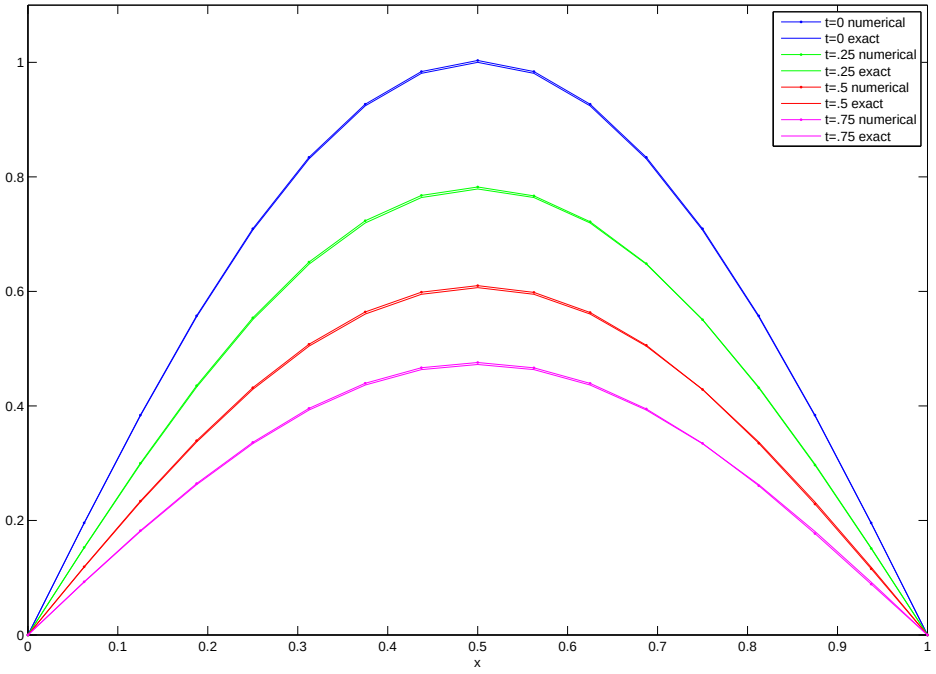


(a) FEM solution error

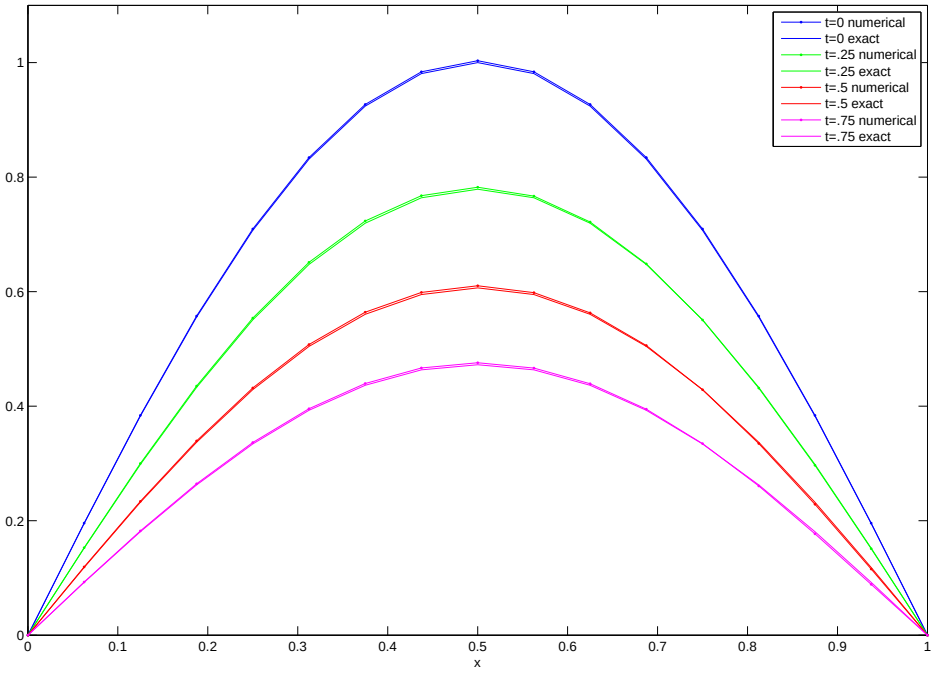


(b) POD solution error, $r = 5$

Figure 3.2: Exact solution, FEM, and ROM using ODE45 for $N = 16$, $Re = 100$ and Dirichlet boundary conditions



(a) FEM solution cross section comparison



(b) POD solution cross section comparison, $r = 5$

Figure 3.3: Comparison of FEM, and ROM cross sections using ODE45 for $N = 16$, $Re = 100$ and Dirichlet boundary conditions

3.3.2 Dirichlet Boundary Conditions Problem #2 ROM

Consider the Dirichlet boundary conditions problem with an initial condition given by

$$u_0(x) = \begin{cases} 1, & \text{if } x \in (0, \frac{1}{4}] \\ 0, & \text{otherwise} \end{cases} \tag{3.30}$$

and forcing term

$$f(t,x) \quad = \quad 0. \tag{3.31}$$

The FEM solution is shown in Figure 3.4. We implement the system shown in (3.24) using a variety of POD basis dimensions and the results are shown in Table 3.1. As shown in the table, when the dimension of the POD model increases we are retaining more information from the FEM solution. Therefore using a larger POD model provides a lower error but increases the cost of computation. We can see the decay of singular values in Figure 3.5. Since the singular values drop off quickly we know that a low dimensional POD basis can provide a reasonably accurate model. Some examples of the ROMs using various dimensions of POD bases are found in Figures 3.6, 3.7, 3.8, 3.9, 3.10, 3.11. In Figure 3.6 we see the POD model using only 3 vectors and note the max error compared to FEM approximation is nearly 1 at the initial condition. In Figure 3.7 we see the POD model using only 5 vectors and note the max error compared to FEM approximation is less than 0.6 at the initial condition. In Figure 3.8 we see the POD model using only 7 vectors and note that even the max error at the initial condition may not have decreased substantially, when we exclude the initial condition the error has decreased. In Figure 3.9 we see the POD model using only 10 vectors and note the max error compared to FEM approximation is less than 0.1 at the initial condition. In Figure 3.10 we see the POD model using 15 vectors and note the max error compared to FEM approximation is less than 0.02 at the initial condition. In Figure 3.11 we see the POD model using 20 vectors and note the max error compared to FEM approximation is less than 1.5×10^{-3} at the initial condition. Here using 20 vectors there is no visual difference between the reduced order model and the full numerical solution.

	Dimension of Basis	Solver Time	Energy	Err^{rel}	Relative Time Savings
FEM	$N = 32$	1.5743	1.0000	-	-
POD	$r = 3$	0.2491	0.9436	2.887×10^{-1}	0.8418
	$r = 5$	0.2743	0.9883	1.302×10^{-1}	0.8258
	$r = 7$	0.3780	0.9974	5.573×10^{-2}	0.7599
	$r = 10$	0.6448	0.9997	1.795×10^{-2}	0.5904
	$r = 15$	1.0335	1.0000	2.740×10^{-3}	0.3435
	$r = 20$	1.2334	1.0000	3.150×10^{-4}	0.2165

Table 3.1: POD Results for $T = 2$, $Re = 100$ and Dirichlet boundary conditions

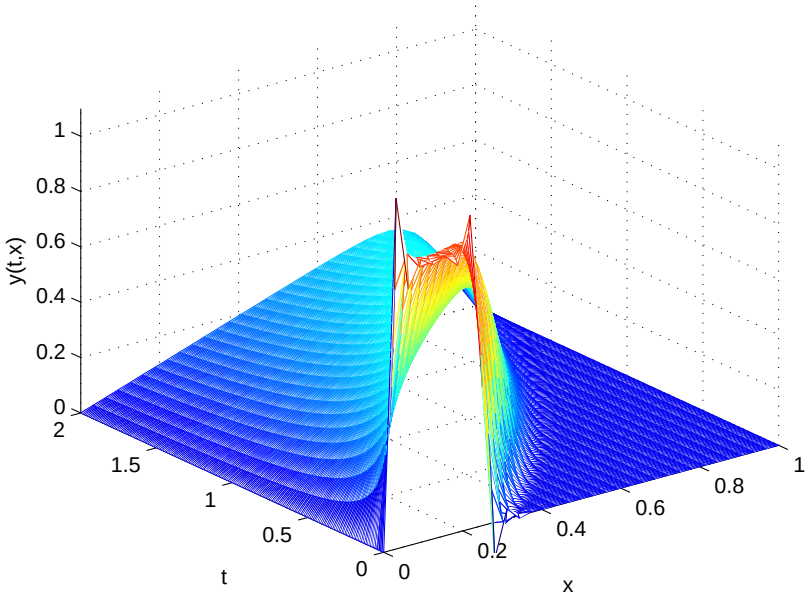


Figure 3.4: FEM solution using ODE45 for $N = 32$, $Re = 100$ and Dirichlet boundary conditions

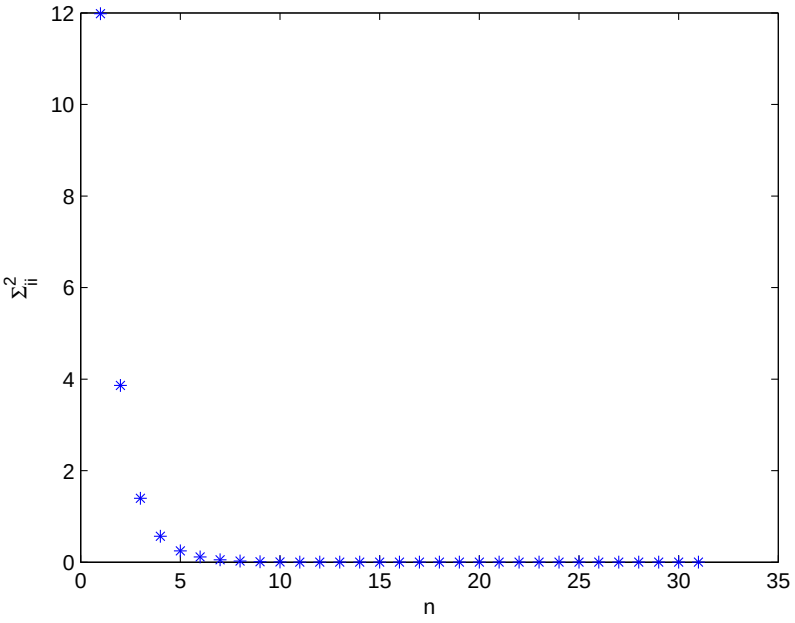
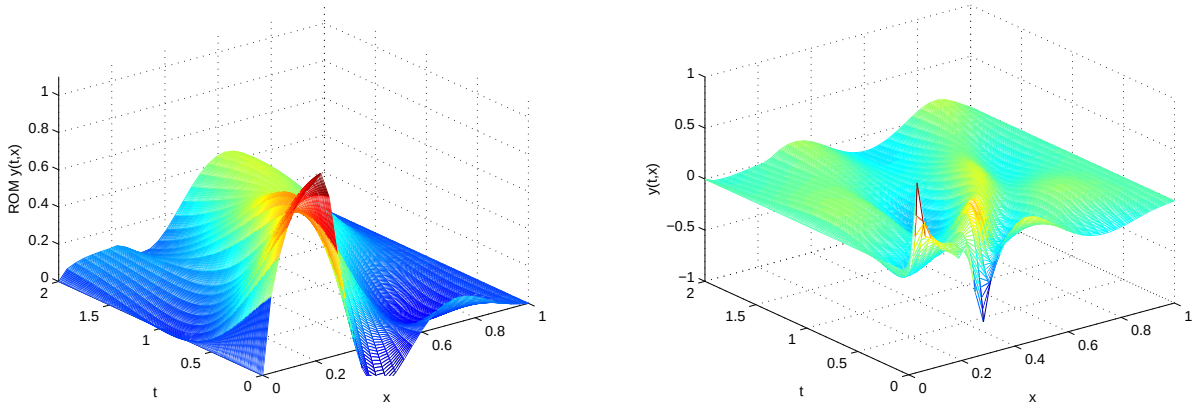
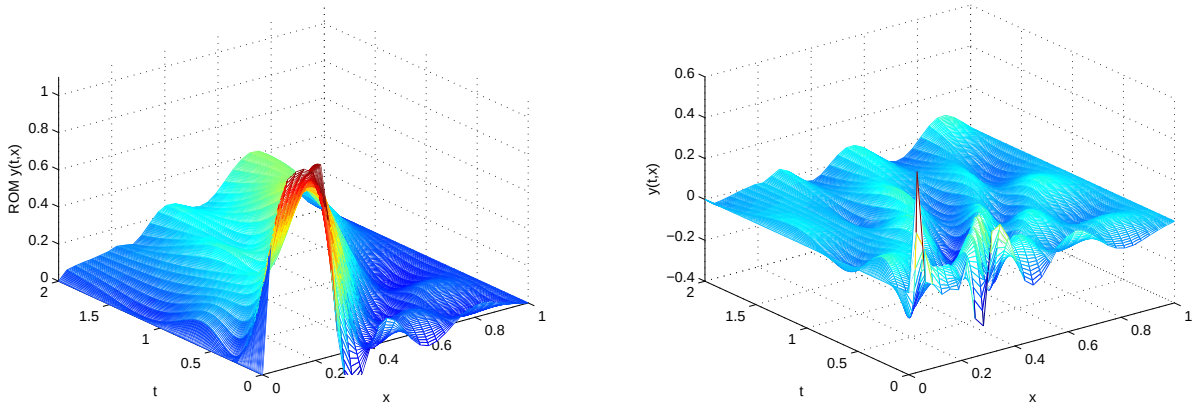


Figure 3.5: Eigenvalues corresponding the POD basis vectors for $N = 32$, $Re = 100$ and Dirichlet boundary conditions



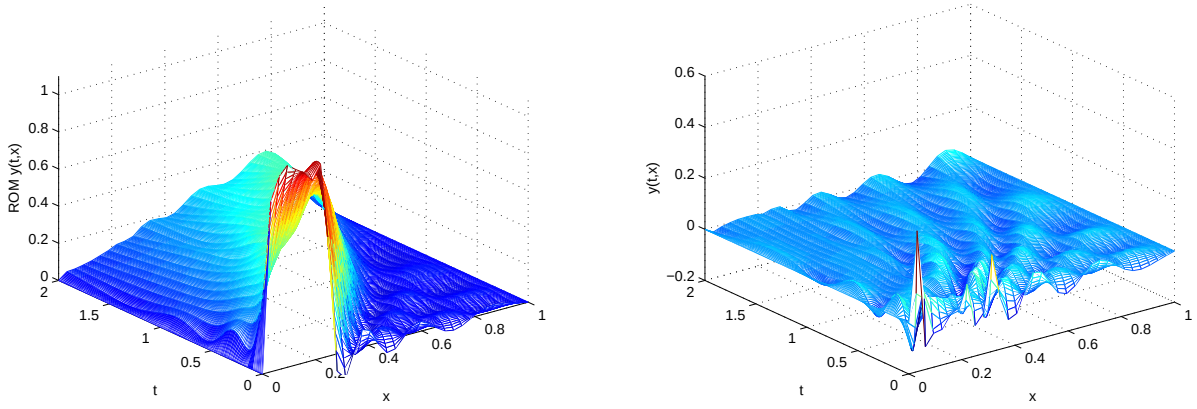
(a) POD solution (b) POD error

Figure 3.6: $r = 3$ POD solution and error using ODE45 for $N = 32$, $Re = 100$ and Dirichlet boundary conditions



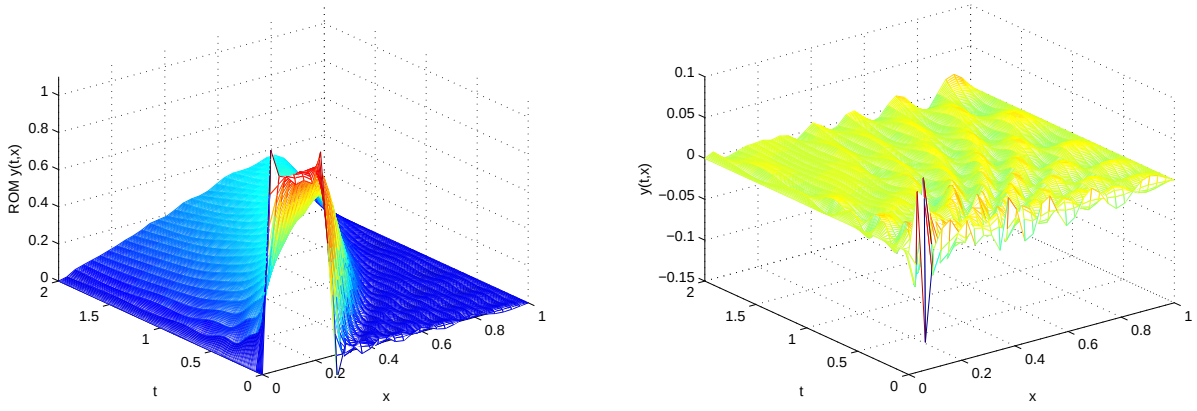
(a) POD solution (b) POD error

Figure 3.7: $r = 5$ POD solution and error using ODE45 for $N = 32$, $Re = 100$ and Dirichlet boundary conditions



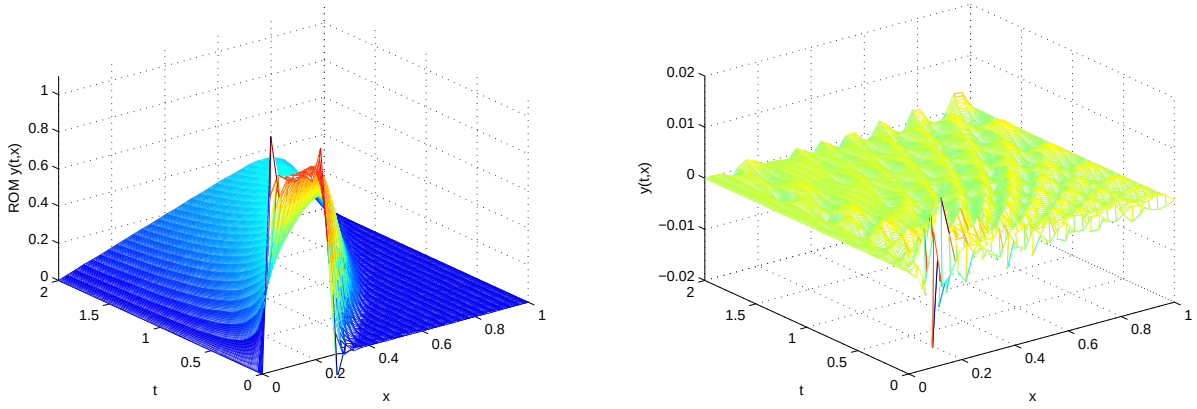
(a) POD solution (b) POD error

Figure 3.8: $r = 7$ POD solution and error using ODE45 for $N = 32$, $Re = 100$ and Dirichlet boundary conditions



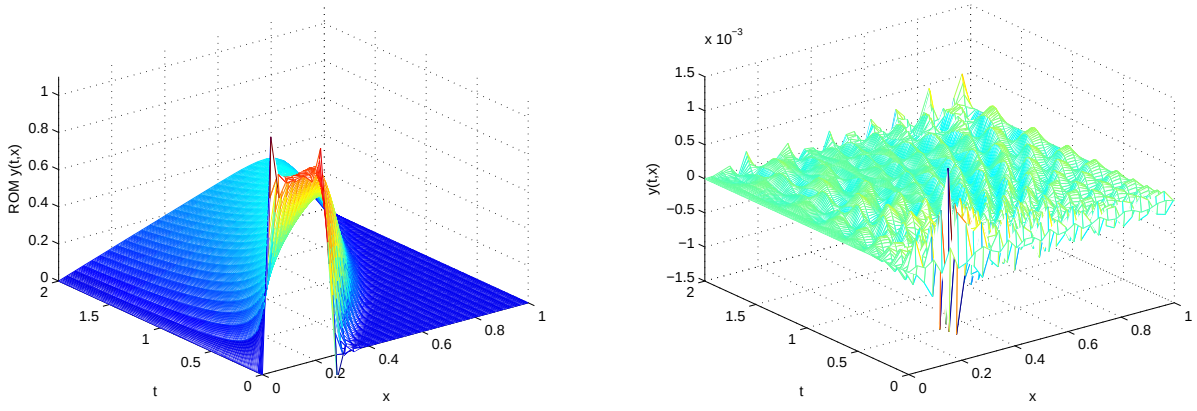
(a) POD solution (b) POD error

Figure 3.9: $r = 10$ POD solution and error using ODE45 for $N = 32$, $Re = 100$ and Dirichlet boundary conditions



(a) POD solution (b) POD error

Figure 3.10: $r = 15$ POD solution and error using ODE45 for $N = 32$, $Re = 100$ and Dirichlet boundary conditions



(a) POD solution (b) POD error

Figure 3.11: $r = 20$ POD solution and error using ODE45 for $N = 32$, $Re = 100$ and Dirichlet boundary conditions

3.3.3 Neumann-Dirichlet Boundary Conditions Problem #1 ROM

Recall from §2.3.2.2, the problem with Neumann-Dirichlet boundary conditions, $\delta = 0$ and exact solution given by

$$u(t,x) \quad = \quad e^{-t}\left(1-x^2\right). \tag{3.32}$$

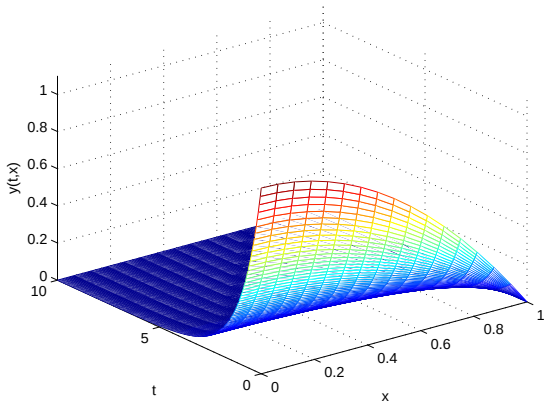
The initial condition is given by

$$u_0(x) = \left(1-x^2\right). \tag{3.33}$$

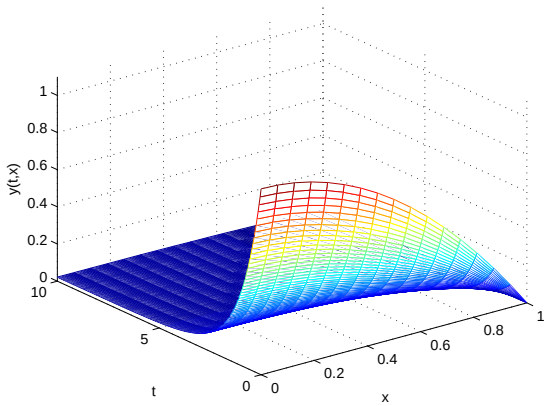
This meets the boundary conditions and the associated MMS forcing term is

$$f(t,x) \quad = \quad -e^{-t}\left(1-x^2\right) - e^{-2t}2x\left(1-x^2\right) + 2qe^{-t}. \tag{3.34}$$

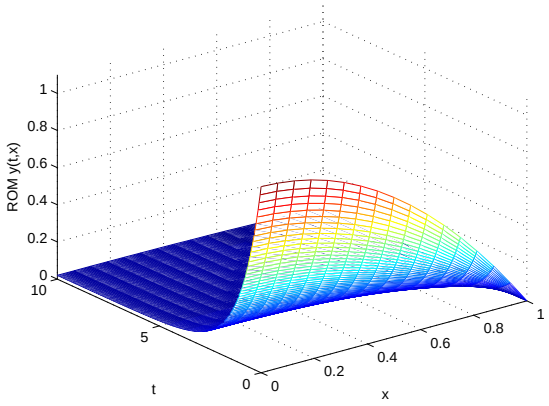
We implement the system shown in (3.26) using a POD dimension $r = 5$ and the results are shown in Figures 3.12, 3.13, 3.14. Again, in Figure 3.12, we see good high level agreement of the POD model to the full discretization numerical solution and the exact solution. In Figure 3.13, we compare the error between the FEM numerical solution and the exact solution to the error between the POD model numerical solution error from the exact solution. Here we see almost an exact match, with very small differences relating to the smoothness of the errors. In Figure 3.14, we compare the error between the FEM numerical solution and the exact solution to the error between the POD model numerical solution error from the exact solution for some particular time instances. Here, there is no discernable difference between the errors.



(a) Exact solution

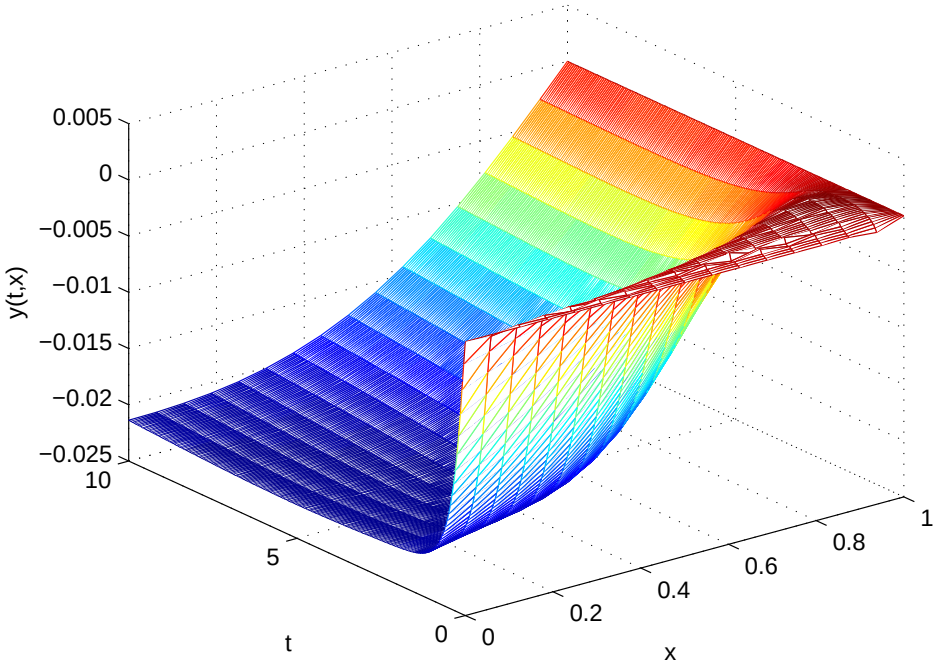


(b) FEM solution

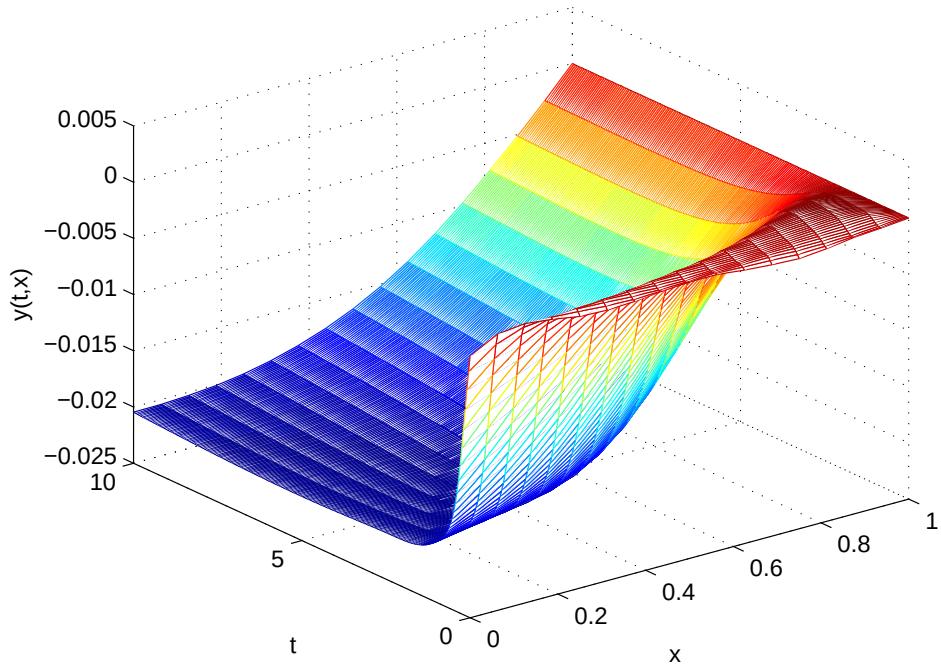


(c) POD solution, $r = 5$

Figure 3.12: Exact solution, FEM, and ROM using ODE45 for $N = 16$, $Re = 100$ and Neumann-Dirichlet boundary conditions

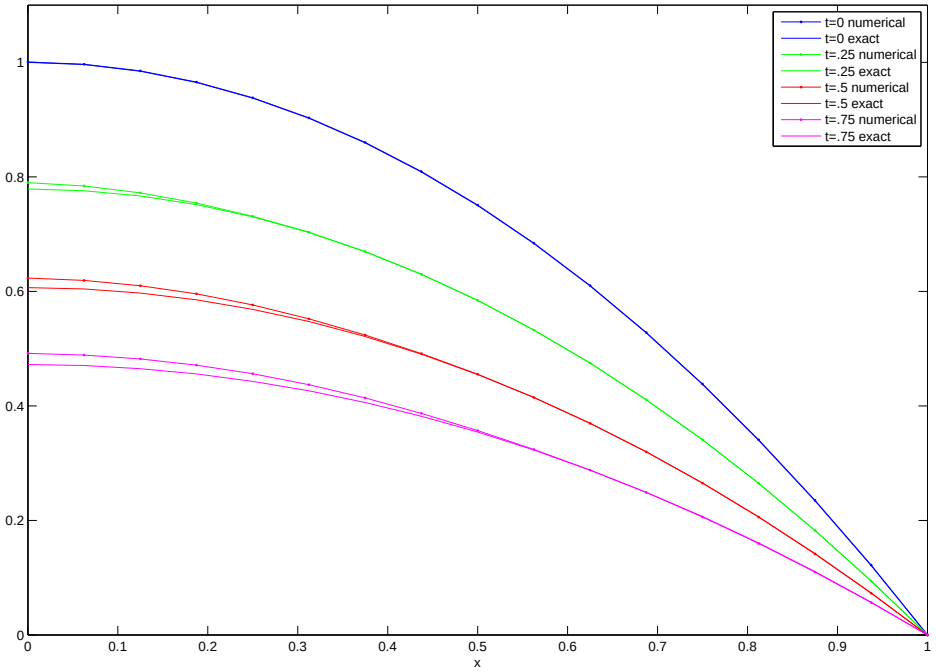


(a) FEM solution error

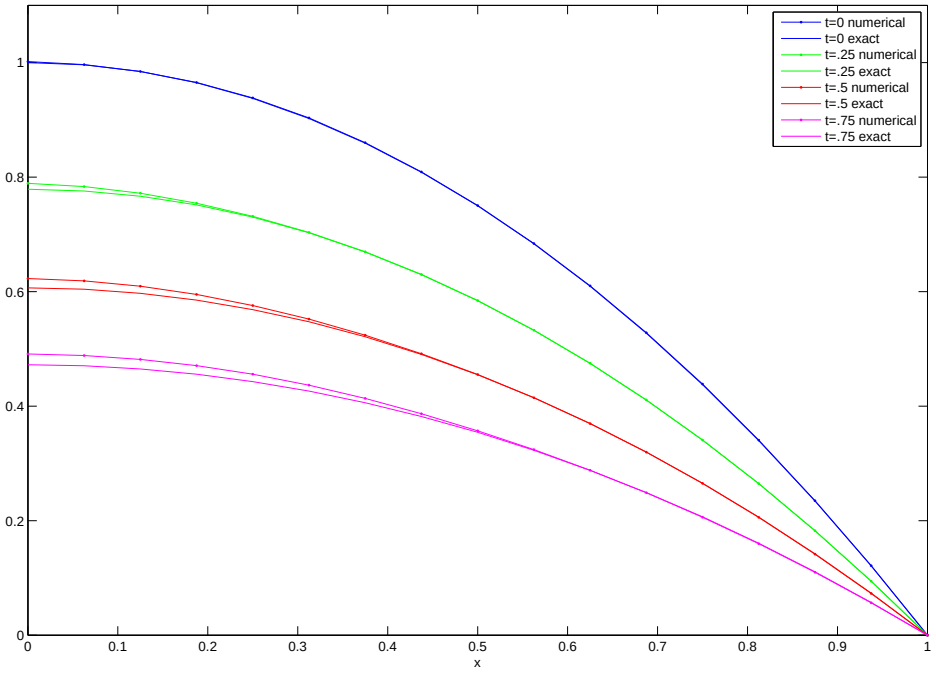


(b) POD solution error, $r = 5$

Figure 3.13: Exact solution, FEM, and ROM using ODE45 for $N = 16$, $Re = 100$ and Neumann-Dirichlet boundary conditions



(a) FEM solution cross section comparison



(b) POD solution cross section comparison, $r = 5$

Figure 3.14: Comparison of FEM, and ROM cross sections using ODE45 for $N = 16$, $Re = 100$ and Neumann-Dirichlet boundary conditions

3.3.4 Neumann-Dirichlet Boundary Conditions Problem #2 ROM

Consider the Neumann-Dirichlet boundary condition problem with an initial condition given by

$$u_0(x) = \begin{cases} \frac{1}{2}\left(1 - \cos 8\pi x\right), & \text{if } x \in (0, \frac{1}{4}] \\ 0, & \text{otherwise} \end{cases} \tag{3.35}$$

and forcing term

$$f(t, x) \quad = \quad 0. \tag{3.36}$$

The FEM solution is shown in Figure 3.15. We implement the system shown in (3.26) using a variety of POD basis dimensions and the results are shown in Table 3.2. As shown in the table, when the dimension of the POD model increases we are retaining more information from the FEM solution. Therefore using a larger POD model provides a lower error but increases the cost of computation. We can see the decay of singular values in Figure 3.16. Since the singular values drop off quickly we know that a low dimensional POD basis can provide a reasonably accurate model. Some examples of the ROMs using various dimensions of POD bases are found in Figures 3.17, 3.18, 3.19, 3.20, 3.21, 3.22. In Figure 3.17 we see the POD model using only 3 vectors and note the max error compared to FEM approximation is nearly 0.4 at the initial condition. In Figure 3.18 we see the POD model using only 5 vectors and note the max error compared to FEM approximation is less than 0.2 at the initial condition. In Figure 3.19 we see the POD model using only 7 vectors and note the max error compared to FEM approximation is less than 0.05 at the initial condition. In Figure 3.20 we see the POD model using only 10 vectors and note the max error compared to FEM approximation is less than 0.01 at the initial condition. In Figure 3.21 we see the POD model using 15 vectors and note the max error compared to FEM approximation is less than 4×10^{-4} at the initial condition. In Figure 3.22 we see the POD model using 20 vectors and note the max error compared to FEM approximation is less than 1×10^{-5}, here the max error is no longer at the initial condition. Even using 10 vectors there is no visual difference between the reduced order model and the full numerical solution.

	Dimension of Basis	Solver Time	Energy	Err^{rel}	Relative Time Savings
FEM	$N = 32$	1.5230	1.0000	-	-
POD	$r = 3$	0.2408	0.9784	1.659×10^{-1}	.8419
	$r = 5$	0.3370	0.9975	5.569×10^{-2}	.7787
	$r = 7$	0.4469	0.9997	1.812×10^{-2}	.7066
	$r = 10$	0.6859	1.0000	3.126×10^{-3}	.5497
	$r = 15$	0.9899	1.0000	9.984×10^{-5}	.3500
	$r = 20$	1.3040	1.0000	2.900×10^{-6}	.1438

Table 3.2: POD Results for $T = 2$, $Re = 100$ and Neumann-Dirichlet boundary conditions

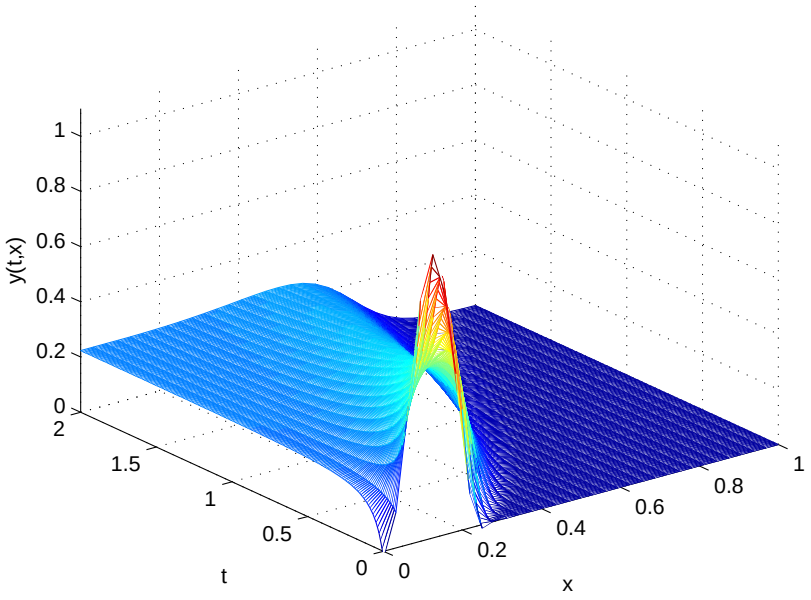


Figure 3.15: FEM solution using ODE45 for $N = 32$, $Re = 100$ and Neumann-Dirichlet boundary conditions

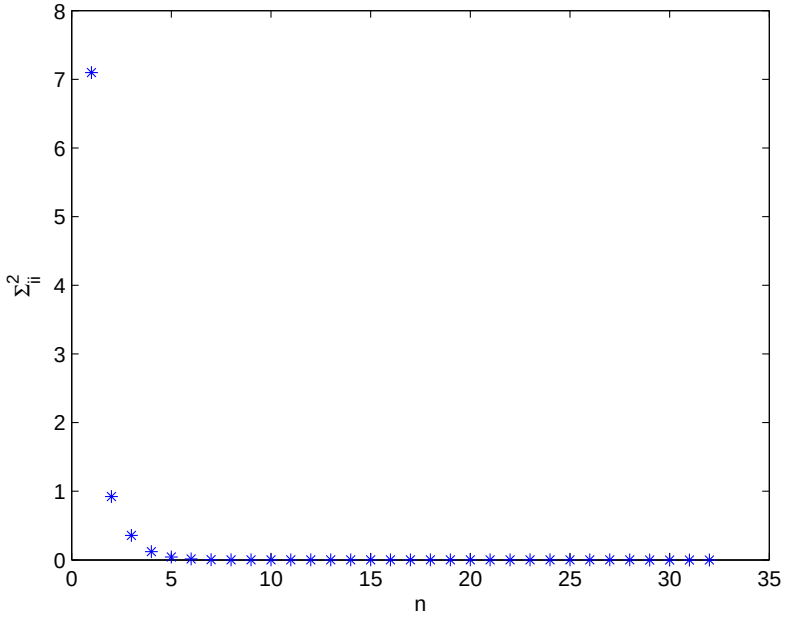
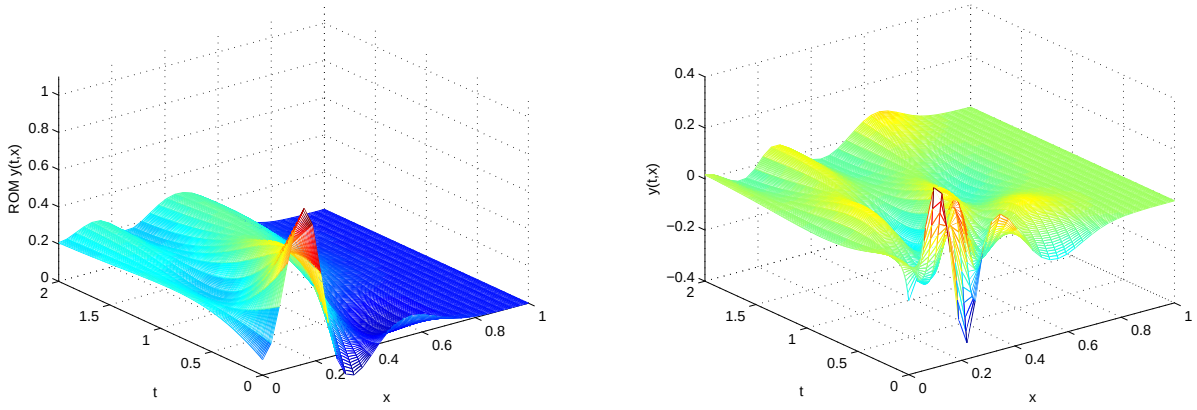
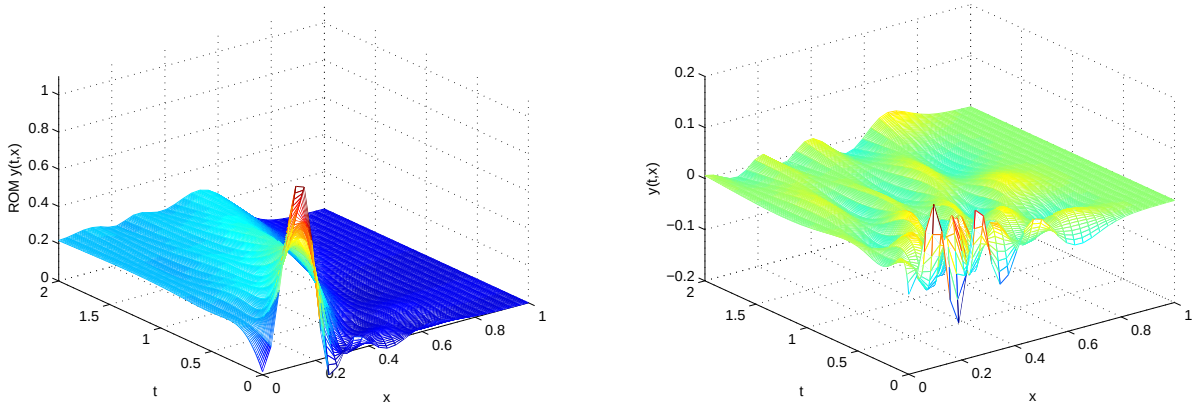


Figure 3.16: Eigenvalues corresponding the POD basis vectors



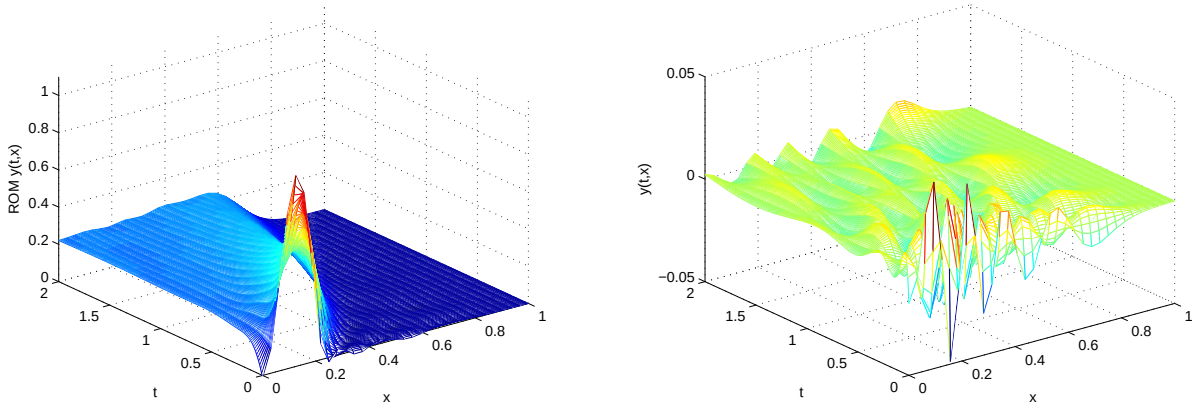
(a) POD solution (b) POD error

Figure 3.17: $r = 3$ POD solution and error using ODE45 for $N = 32$, $Re = 100$ and Neumann-Dirichlet boundary conditions



(a) POD solution (b) POD error

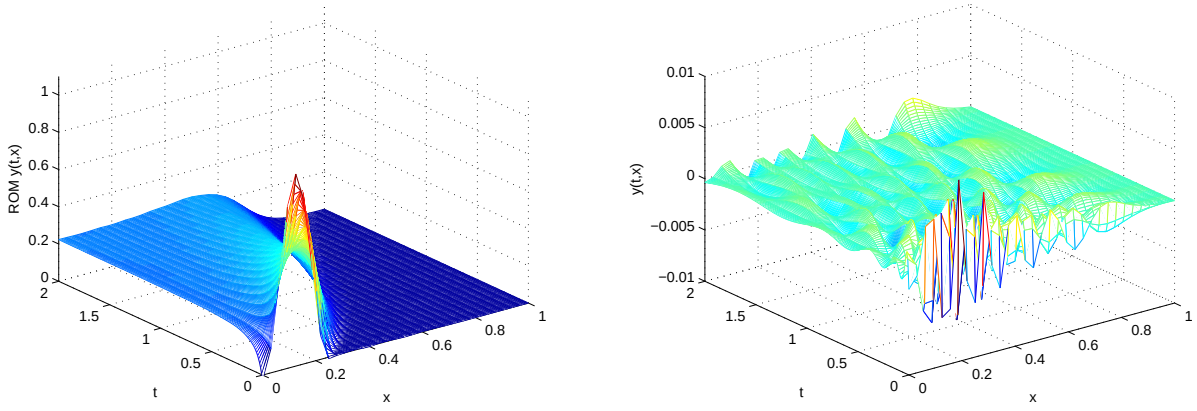
Figure 3.18: $r = 5$ POD solution and error using ODE45 for $N = 32$, $Re = 100$ and Neumann-Dirichlet boundary conditions



(a) POD solution

(b) POD error

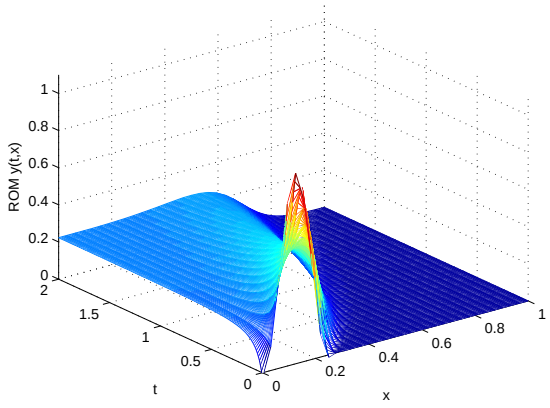
Figure 3.19: $r = 7$ POD solution and error using ODE45 for $N = 32$, $Re = 100$ and Neumann-Dirichlet boundary conditions



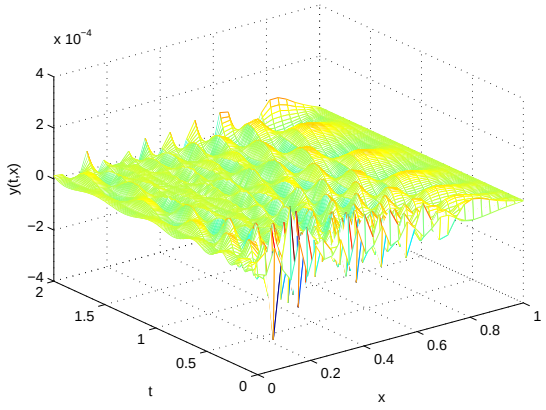
(a) POD solution

(b) POD error

Figure 3.20: $r = 10$ POD solution and error using ODE45 for $N = 32$, $Re = 100$ and Neumann-Dirichlet boundary conditions

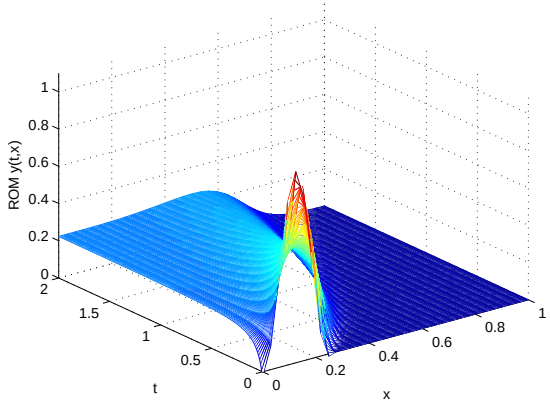


(a) POD solution

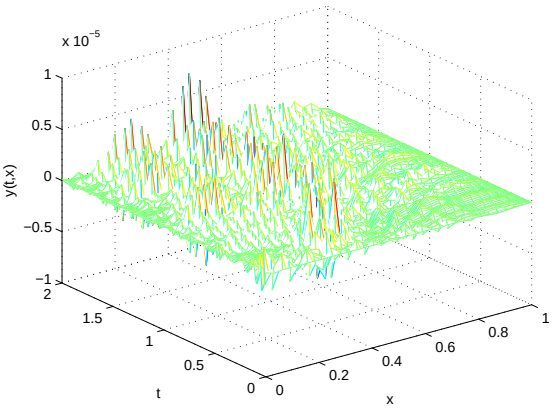


(b) POD error

Figure 3.21: $r = 15$ POD solution and error using ODE45 for $N = 32$, $Re = 100$ and Neumann-Dirichlet boundary conditions



(a) POD solution



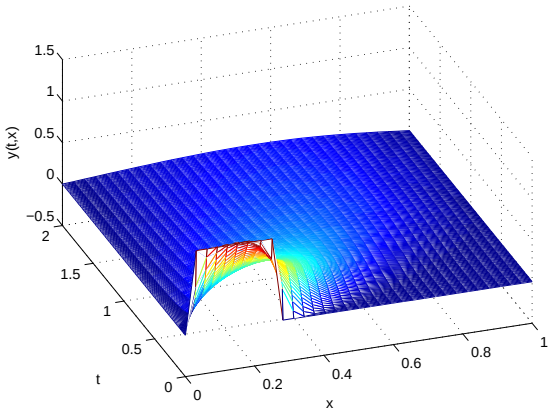
(b) POD error

Figure 3.22: $r = 20$ POD solution and error using ODE45 for $N = 32$, $Re = 100$ and Neumann-Dirichlet boundary conditions

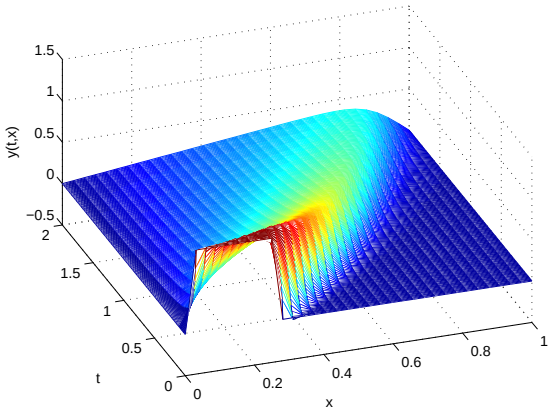
3.4 Parameter dependence

We have demonstrated the ability of the POD method to create a reduced order model that saves a large percentage of time while sacrificing only a little accuracy. In every case the the POD method has found a reduced basis depending on a data matrix which was the output of some direct numerical simulation. These numerical simulations depend upon the parameters used in the computation. In the case of Burgers' equation as a fluid model, the parameter q represents the inverse of the Reynolds number. This number describes the balance of the convection and dissipation in the system. A higher Reynolds number leads to a convection dominated system while a lower Reynolds number leads to a diffusion dominated system. Using the system given in §3.3.2 for $q_1 = 1/20$, $q_2 = 1/100$, $q_3 = 1/200$ we see the effect of the parameter on the system in Figure 3.23. As we see in Figure 3.23, after the initial condition the solution depends entirely on the parameter.

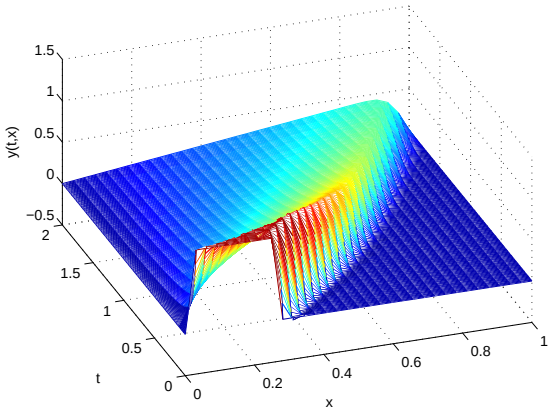
In a typical design sequence there may be many parameters whose values can be varied within specified ranges and computing a solution for each set of parameter values may be prohibitively expensive in terms of time or resources. Therefore we are interested in the range of values for which a single POD basis can still provide an accurate ROM. Thus, with a predetermined baseline parameter we will use POD to generate a basis and test the ability of that basis to accurately represent the dynamics for a range of parameters in order to illustrate an effective range.



(a) $q = 1/20$



(b) $q = 1/100$



(c) $q = 1/200$

Figure 3.23: Comparison different parameters using ODE45 for $N = 32$, and Dirichlet boundary conditions

3.4.1 Dirichlet Boundary Conditions POD range

Again using the system given in §3.3.2, we generate a POD basis of various dimensions using a nominal parameter $q_0 = 1/100$. We then use the POD model as a ROM for a variety of parameter test points q_i for $1/q_i \in [10, 400]$. For each q_i parameter we compare the ROM to a full FEM solution and plot the relative errors in Figure 3.24. We see for lower model dimensions the error is smaller for smaller values of $1/q$. Also, as the dimension increases we recover the specific space that contains our full numerical solution. Using the POD model for any parameter outside of a local neighborhood produces a substantially less accurate solution compared to a full numerical solution.

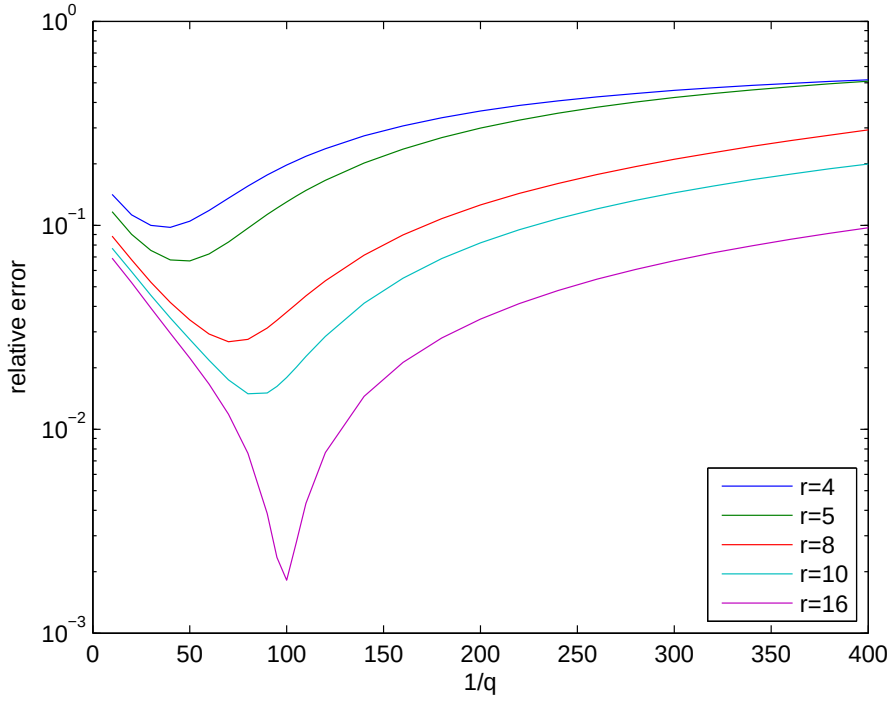


Figure 3.24: POD relative error against FEM for various q, Dirichlet boundary conditions

3.4.2 Neumann-Dirichlet Boundary Conditions POD range

Similarly, using the system given in §3.3.4, we repeat the process for the Neumann-Dirichlet problem. The results are shown in Figure 3.25. Looking at the figure we see the ROM provides a poor approximation outside a local area of the baseline parameter. Thus, the solution of the problem with Neumann-Dirichlet boundary condition is heavily dependent on the parameter chosen. In the next chapter we will investigate several POD modifications that seek to improve the accuracy over a wider range of parameter values.

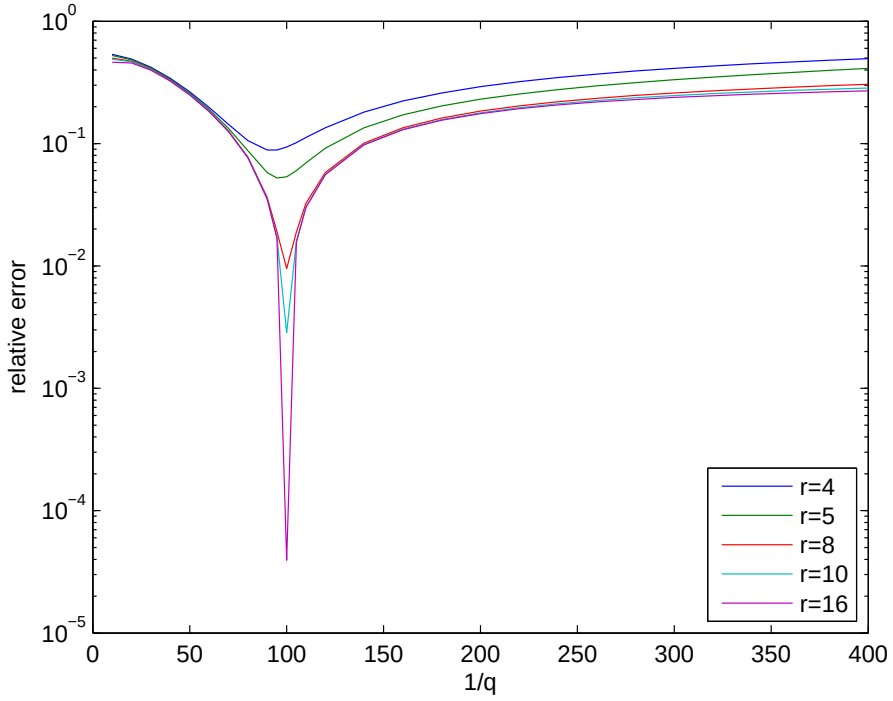


Figure 3.25: POD relative error against FEM for various q, Neumann-Dirichlet boundary conditions

Chapter 4

Modifying POD

In this chapter we seek to improve the accuracy of our POD model over a wider range of values. Recall that the POD process begins with a direct numerical simulation, which may require a significant amount of time to generate, and produces a POD basis by extracting the eigenfunctions that best describe the data in a mean squared error sense. The first modification to our POD model is to develop a Global POD model.

4.1 Global POD

The POD method produces a reduced basis that depends upon the data snapshot inputs, and Global POD seeks to find a single reduced basis that describes a range of parameters. This is accomplished by generating several numerical simulations at various values of the parameter range and assembling all data into a larger snapshot matrix. The POD method is then applied to identify a basis that minimizes the error across all solution samples in the snapshot matrix. Due to the optimality of the POD basis, the resulting Global POD basis should provide a reasonable approximation to the solutions associated with each test parameter used to generated the basis. If the parameters are a reasonable sample of the possible range of parameters then the Global POD ROM should be able to approximate the solution for any parameter in the range. The Global POD algorithm steps are shown below, we assume that all full solutions are $n \times m$ snapshot matrices, but this is not required.

Global POD steps:

[1] Select parameter test points q_j for $j = 1, ..., J$

[2] For each q_j compute a snapshot matrix $[Y_{n\times m}]_j$

[3] Assemble global snapshot matrix $W = \left[[Y_{n\times m}]_1, [Y_{n\times m}]_2, ..., [Y_{n\times m}]_J \right]$

[4] Compute POD basis for W

4.1.1 Global POD Numerical Experiments

For the following two experiments we use the same range of parameters tested in §3.4. We generate solutions using $q_1 = 1/50, q_2 = 1/100, q_3 = 1/200, q_4 = 1/400$ as representative inputs from the parameter range.

4.1.1.1 Global POD Dirichlet Boundary Conditions Problem

We use the same Dirichlet problem from §3.3.2. The relative error of the Global POD ROM for various basis dimensions is shown in Figure 4.1. Again, we see for lower model dimensions the error is smaller for smaller values of $1/q$. Also we compare the Global POD model error against the baseline POD model error in Figure 4.2, this is where we see the major difference between the standard POD and Global POD. As the dimension of the basis increases the standard POD recovers the specific solution subspace for the parameter used to generate the standard basis. As the dimension of the basis increases the Global POD model can provide lower errors for a large portion of the parameter range. The Global POD provides a basis that is optimal over the range of snapshot input data, but may not perform as well as the standard basis for any one parameter.

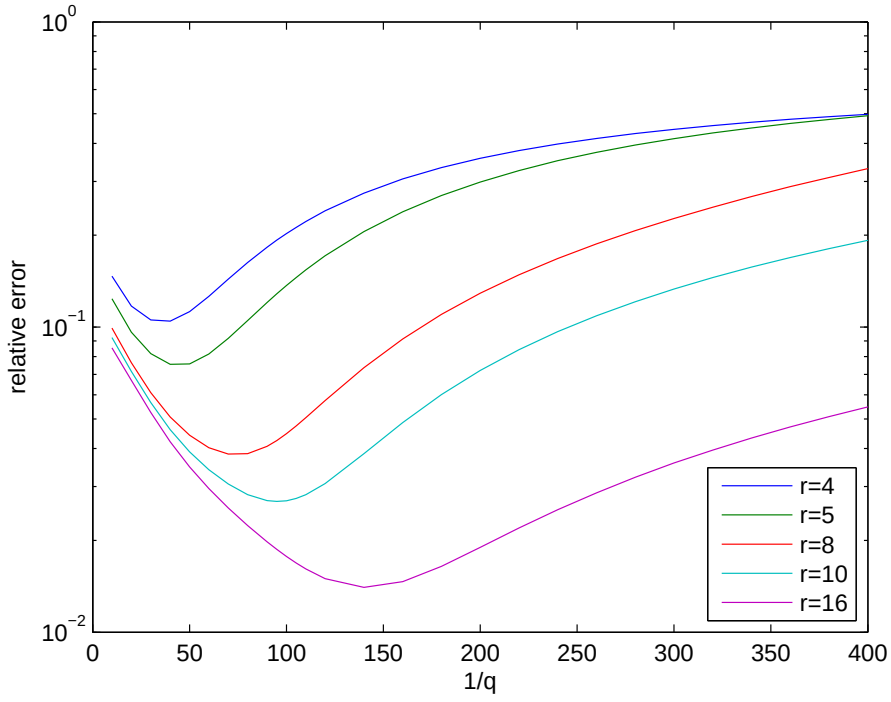


Figure 4.1: Global POD model relative error against FEM for various q, Dirichlet boundary conditions

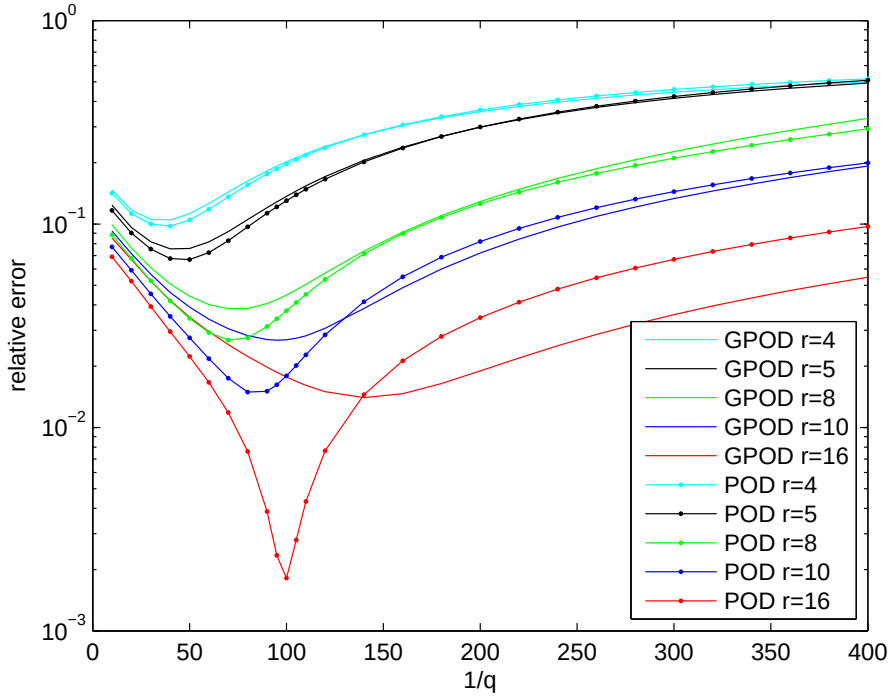


Figure 4.2: Global POD model relative error compared to baseline POD model relative error against FEM for various q, Dirichlet boundary conditions

4.1.1.2 Global POD Neumann-Dirichlet Boundary Conditions Problem

The relative error of the Global POD model for various basis dimensions applied to the Neumann-Dirichlet boundary conditions problem is shown in Figure 4.3. We note for $r = 16$ the error has local minimums at $1/q = 50$ and $1/q = 200$, two of our test points. Also, we compare the Global POD against the baseline POD in Figure 4.4. We see even more dramatically here the Global POD improvement in accuracy against the standard POD, over the full range of parameters except in a local area around the baseline parameter value.

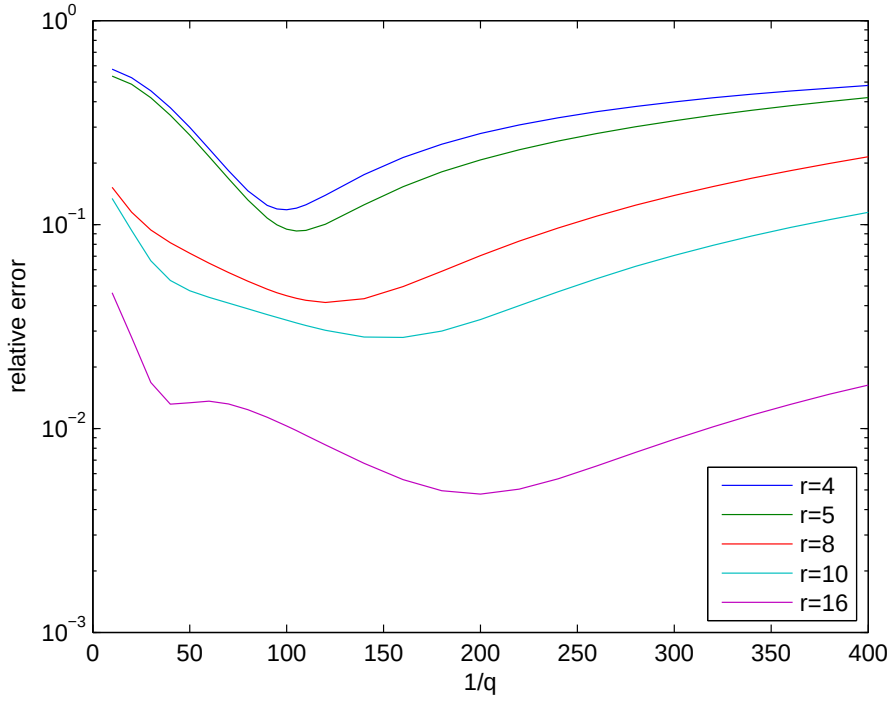


Figure 4.3: Global POD model relative error against FEM for various q, Neumann-Dirichlet boundary conditions

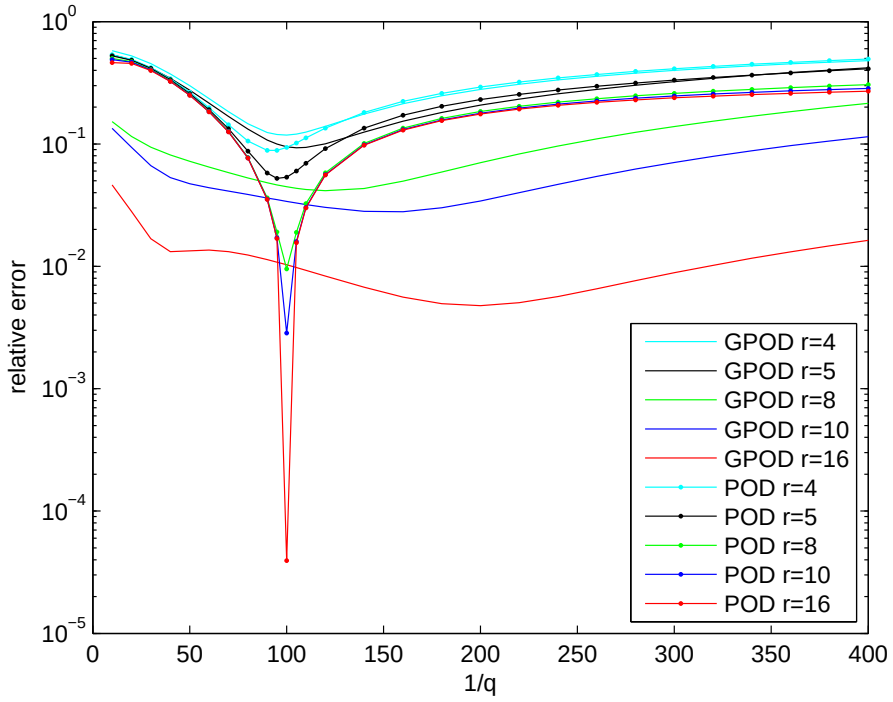


Figure 4.4: Global POD model compared to baseline POD model relative error against FEM for various q, Neumann-Dirichlet boundary conditions

4.2 POD Basis Sensitivity

We wish to understand how each POD basis $\phi_i^r(x)$ vector depends upon the parameter. Here we use the method outlined in [54], tailored to our situation where $m > n$, namely there are more time snapshots than spatial elements.

Since M does not depend on the parameter and $\phi_i^r(x) = L^{-T}U_i(x)$ then the sensitivity of the POD basis vector $\phi_i^r(x)$ can be expressed as

$$\left[\phi_i^r\right]^q(x) = L^{-T}U_i^q(x)$$

where the superscript q represents the derivative with respect to the parameter. Recall from (3.7), $\tilde{Y}$ can be decomposed as $\tilde{Y} = U\Sigma V^T$ where U_i solves

$$\tilde{Y}\tilde{Y}^T U_i = \Lambda_i U_i. \tag{4.1}$$

If we assume that the vectors U_i depend continuously on q, then the sensitivity with respect to q of (4.1) is given by

$$\begin{aligned}
\frac{\partial}{\partial q}\left[\tilde{Y}\tilde{Y}^T U_i\right] &= \frac{\partial}{\partial q}\left[\Lambda_i U_i\right] \\
\frac{\partial}{\partial q}\left[\tilde{Y}\tilde{Y}^T\right] U_i + \tilde{Y}\tilde{Y}^T\frac{\partial}{\partial q}\left[U_i\right] &= \frac{\partial}{\partial q}\left[\Lambda_i\right] U_i + \Lambda_i\frac{\partial}{\partial q}\left[U_i\right] \\
\left(\tilde{Y}^q\tilde{Y}^T + \tilde{Y}\left(\tilde{Y}^q\right)^T\right) U_i + \tilde{Y}\tilde{Y}^T U_i^q &= \Lambda_i^q U_i + \Lambda_i U_i^q \\
\left[\tilde{Y}\tilde{Y}^T - \Lambda_i I\right] U_i^q &= -\left[\left(\tilde{Y}^q\tilde{Y}^T + \tilde{Y}\left(\tilde{Y}^q\right)^T\right) - \Lambda_i^q I\right] U_i.
\end{aligned} \tag{4.2}$$

The matrix Y^q is the sensitivity of the solution matrix, this can be computed by the Continuous Sensitivity Equation (CSE) method and will be described in the next section. Detailed information about the method is given by Borggaard [55].

Multiplying by U_i^T then (4.2) becomes

$$U_i^T\left[\tilde{Y}\tilde{Y}^T - \Lambda_i I\right] U_i^q = -U_i^T\left[\left(\tilde{Y}^q\tilde{Y}^T + \tilde{Y}\left(\tilde{Y}^q\right)^T\right) - \Lambda_i^q I\right] U_i. \tag{4.3}$$

We note the left side can be written as $\left[\left[\tilde{Y}\tilde{Y}^T - \Lambda_i I\right]^T U_i\right]^T U_i^q$. Since $\tilde{Y}\tilde{Y}^T$ is symmetric and Λ_i is the eigenvalue associated with U_i then

$$\left[\left[\tilde{Y}\tilde{Y}^T - \Lambda_i I\right]^T U_i\right]^T U_i^q = 0.$$

Therefore we have

$$\Lambda_i^q = U_i^T\left(\tilde{Y}^q\tilde{Y}^T + \tilde{Y}\left(\tilde{Y}^q\right)^T\right) U_i.$$

Finally we see that all vectors of the form $U_i^q + cU_i$ with $c \in \mathbb{R}$ are a solutions to the system given in (4.2). To find the particular solution U_i^q, we solve for the least squares solution then perform a Gram-Schmidt step so that U_i^q is orthogonal to U_i.

4.2.1 Solution Sensitivity Analysis

In this section we apply the CSE method to find Y^q, the sensitivity of the solution with respect to the parameter. We assume that all the necessary derivatives exist and treat the PDE (2.34) as a function of the parameter q, hence $u(t,x) = u(t,x,q)$. Then we differentiate the problem with respect to the parameter to obtain the sensitivity equation. Thus, for Burgers' equation (1.1) we obtain

$$\begin{aligned}
\frac{\partial}{\partial q}\left[u_t(t,x,q) + \left[u(t,x,q)\, u_x(t,x,q)\right] - q u_{xx}(t,x,q)\right] &= \frac{\partial}{\partial q}\left[f(t,x,q)\right] \\
u_{tq}(t,x,q) + \frac{\partial}{\partial q}\left[u(t,x,q)\, u_x(t,x,q)\right] - \left[q u_{xx}(t,x,q)\right]_q &= f_q(t,x,q) \\
u_{qt}(t,x,q) + \left[u(t,x,q)\, u_q(t,x,q)\right]_x - u_{xx}(t,x,q) - q u_{qxx}(t,x,q) &= f_q(t,x,q) \qquad (4.4)
\end{aligned}$$

For simplicity we define $s(t,x,q) = u_q(t,x,q)$ to represent the sensitivity information for $u(t,x,q)$. Then we can rewrite the sensitivity equation as

$$s_t(t,x,q) + \left[u(t,x,q)\, s(t,x,q)\right]_x - u_{xx}(t,x,q) - q s_{xx}(t,x,q) = f_q(t,x,q). \qquad (4.5)$$

We must also differentiate the boundary and initial conditions so that we have a complete system to solve numerically. Now we use the standard FEM to derive the ODE system approximation for each boundary value problem.

4.2.1.1 Solution Sensitivity for Dirichlet Boundary Conditions Problem

Consider the problem

$$s_t(t,x,q) + \left[u(t,x,q)\, s(t,x,q)\right]_x - u_{xx}(t,x,q) - q s_{xx}(t,x,q) = f_q(t,x,q) \qquad (4.6)$$

where $s(t,x,q) = u_q(t,x,q)$, $x \in [0,1]$ and $t \in [0,t_f]$. The boundary conditions are given by

$$u_q(t,0,q) = s(t,0,q) = 0, \qquad u_q(t,1,q) = s(t,1,q) = 0, \qquad (4.7)$$

and the initial condition is

$$u_q(0,x,q) = s(0,x,q) = s_0(x,q). \qquad (4.8)$$

Multiplying both sides by a test function $v(x)$ and integrating yields

$$\begin{aligned}
\int_0^1 \left(s_t(t,x,q) + \left[u(t,x,q)\, s(t,x,q)\right]_x - u_{xx}(t,x,q) - q s_{xx}(t,x,q)\right) v(x)\, dx & \\
= \int_0^1 f_q(t,x,q) v(x)\, dx. \qquad (4.9)
\end{aligned}$$

If $v(\cdot)$ is piecewise smooth, then $v'(x) \in L^2(0,1)$ so we can apply integration by parts and take advantage of the essential Dirichlet boundary conditions to obtain

$$\begin{aligned}\int_0^1 u_{xx}(t,x,q)v(x)\,dx &= u_x(t,x,q)v(x)|_0^1 - \int_0^1 u_x(t,x,q)v_x(x)\,dx \\ &= -\int_0^1 u_x(t,x,q)v_x(x)\,dx\end{aligned}$$

$$\begin{aligned}\int_0^1 s_{xx}(t,x,q)v(x)\,dx &= s_x(t,x,q)v(x)|_0^1 - \int_0^1 s_x(t,x,q)v_x(x)\,dx \\ &= -\int_0^1 s_x(t,x,q)v_x(x)\,dx\end{aligned}$$

$$\begin{aligned}\int_0^1 \left[u(t,x,q)\,s(t,x,q)\right]_x v(x)\,dx &= u(t,x,q)\,s(t,x,q)\,v(x)|_0^1 - \int_0^1 \left[u(t,x,q)\,s(t,x,q)\right] v_x(x)\,dx \\ &= -\int_0^1 u(t,x,q)\,s(t,x,q)v_x(x)\,dx.\end{aligned}$$

Thus (4.9) becomes the weak form of the Sensitivity problem with Dirichlet boundary conditions given by

$$\begin{aligned}\int_0^1 \big(s_t(t,x,q)v(x) - u(t,x,q)\,s(t,x,q)\,v_x(x) + u_x(t,x,q)\,v_x(x) + qs_x(t,x,q)\,v_x(x)\big)\,dx \\ = \int_0^1 f_q(t,x,q)v(x)\,dx. \qquad (4.10)\end{aligned}$$

Note that (4.10) must hold for any piecewise smooth function $v(\cdot)$. For consistency we use the same piecewise linear basis functions but now we note that the coefficients of the basis functions depend not just on time, but also on the parameter q. Thus we write

$$u(t,x,q) \approx u^N(t,x,q) = \sum_{j=0}^{N} \alpha_j(t,q)\phi_j(x)$$

$$s(t,x,q) \approx s^N(t,x,q) = \sum_{j=0}^{N} \beta_j(t,q)\phi_j(x).$$

Then (4.10) becomes

$$\int_0^1 \left(\left(\sum_{j=0}^{N} \beta_{j,t}(t,q)\phi_j(x) \right) v(x) - \left(\sum_{j=0}^{N} \alpha_j(t,q)\phi_j(x) \right) \left(\sum_{k=0}^{N} \beta_k(t,q)\phi_k(x) \right) v_x(x) + \left(\sum_{j=0}^{N} \alpha_j(t,q)\phi_{j,x}(x) \right) v_x(x) + q \left(\sum_{j=0}^{N} \beta_j(t,q)\phi_{j,x}(x) \right) v_x(x) \right) dx = \int_0^1 f_q(t,x,q)v(x)\,dx. \tag{4.11}$$

Since this identity holds for arbitrary piecewise smooth $v(x)$, for each $i = 0, 1, ..., N$, we can set $v(x) = \phi_i(x)$. Then from (4.11), for each i we obtain

$$\int_0^1 \left(\left(\sum_{j=0}^{N} \beta_{j,t}(t,q)\phi_j(x) \right) \phi_i(x) - \left(\sum_{j=0}^{N} \alpha_j(t,q)\phi_j(x) \right) \left(\sum_{k=0}^{N} \beta_k(t,q)\phi_k(x) \right) \phi_{i,x}(x) + \left(\sum_{j=0}^{N} \alpha_j(t,q)\phi_{j,x}(x) \right) \phi_{i,x}(x) + q \left(\sum_{j=0}^{N} \beta_j(t,q)\phi_{j,x}(x) \right) \phi_{i,x}(x) \right) dx = \int_0^1 f_q(t,x,q)\phi_i(x)\,dx \tag{4.12}$$

for $i = 0, 1, ..., N$.

Since for all j, k, $\alpha_j(t,q)$, $\beta_k(t,q)$ do not depend on x we can move those terms outside the integral to obtain

$$\sum_{j=0}^{N} \beta_{j,t}(t,q) \left(\int_0^1 \phi_j(x)\phi_i(x)\,dx \right) - \sum_{j=0}^{N}\sum_{k=0}^{N} \alpha_j(t,q)\beta_k(t,q) \left(\int_0^1 \phi_j(x)\phi_k(x)\,\phi_{i,x}(x)\,dx \right) + \sum_{j=0}^{N} \left(\alpha_j(t,q) + q\beta_j(t,q) \right) \left(\int_0^1 \phi_{j,x}(x)\phi_{i,x}(x)\,dx \right) = \int_0^1 f_q(t,x,q)\phi_i(x)\,dx. \tag{4.13}$$

For each $i = 0, 1, ..., N$, the equation (4.13) can be written as a system of $N+1$ ODEs given by

$$M\dot{\boldsymbol{\beta}}(t,q) + \left[qC - \tilde{\mathcal{B}}\left(\boldsymbol{\alpha}(t,q)\right) \right] \boldsymbol{\beta}(t,q) + C\boldsymbol{\alpha}(t,q) \quad = \quad \tilde{F}(t,q) \tag{4.14}$$

where the matrices M and C are the same as before and $\tilde{F}$ is given by $\tilde{F}_i(t,q) = \langle f_q(t,x,q), \phi_i(x) \rangle$ for all $i = 0, 1, ..., N$ and $\tilde{\mathcal{B}}\left(\boldsymbol{\alpha}(t,q)\right)$ is a matrix that depends on $\boldsymbol{\alpha}(t,q)$.

Taking advantage of the Dirichlet boundary conditions we know the value of both $\alpha_0(t,q)$ and $\alpha_N(t,q)$ for all t and q. Thus we eliminate those equations and reduce the size of (4.14) from $N+1$

to $N-1$ equations by solving only for the internal nodes. The matrix $\tilde{\mathcal{B}}(\boldsymbol{\alpha}(t,q))$ has the tridiagonal form shown below suppressing the dependence on (t,q)

$$\tilde{\mathcal{B}}(\boldsymbol{\alpha}(t,q)) = \frac{1}{6}\begin{pmatrix} -\alpha_2 & -(\alpha_1+2\alpha_2) & & & & \\ 2\alpha_1+\alpha_2 & \alpha_1-\alpha_3 & -(\alpha_2+2\alpha_3) & & & \\ & \ddots & \ddots & \ddots & & \\ & & 2\alpha_{N-2}+\alpha_{N-1} & \alpha_{N-2}-\alpha_N & -(\alpha_{N-1}+2\alpha_N) \\ & & & 2\alpha_{N-1}+\alpha_N & \alpha_{N-1} \end{pmatrix}.$$

We also need to find the Galerkin approximation to the initial conditions $s_0(x,q)$. Similar to before the initial condition is given by the matrix equation

$$M\boldsymbol{\beta}(0,q) = \tilde{G} \tag{4.15}$$

where $\tilde{G}_i = \langle s_0(x,q), \phi_i(x)\rangle$.

Combining (4.14) and (4.15) we obtain the initial value ODE system

$$\begin{aligned} M\dot{\boldsymbol{\beta}}(t,q) + \left[qC - \tilde{\mathcal{B}}(\boldsymbol{\alpha}(t,q))\right]\boldsymbol{\beta}(t,q) + C\boldsymbol{\alpha}(t,q) &= \tilde{F}(t,q) \\ M\boldsymbol{\beta}(0,q) &= \tilde{G}. \end{aligned} \tag{4.16}$$

We note that the sensitivity problem is linear in $\boldsymbol{\beta}(t,q)$ but is dependent on the system given in (2.16) and as such we can formulate a single coupled system to solve for the solution and sensitivities at the same time. Thus, from (2.16) and (4.16) we have

$$\begin{aligned} M\dot{\boldsymbol{\alpha}}(t) + \frac{1}{2}\mathcal{B}(\boldsymbol{\alpha}(t))\boldsymbol{\alpha}(t) + qC\boldsymbol{\alpha}(t) &= F(t,q) \\ M\dot{\boldsymbol{\beta}}(t,q) + \left[qC - \tilde{\mathcal{B}}(\boldsymbol{\alpha}(t,q))\right]\boldsymbol{\beta}(t,q) + C\boldsymbol{\alpha}(t,q) &= \tilde{F}(t,q) \\ M\boldsymbol{\alpha}(0,q) &= G \\ M\boldsymbol{\beta}(0,q) &= \tilde{G} \end{aligned}$$

therefore the coupled system is given as

$$\begin{aligned} \begin{bmatrix}\boldsymbol{\alpha}(t,q)\\ \boldsymbol{\beta}(t,q)\end{bmatrix} &= \begin{bmatrix} M & 0 \\ 0 & M\end{bmatrix}^{-1}\left(\begin{bmatrix} -qC - \frac{1}{2}\mathcal{B}(\boldsymbol{\alpha}(t)) & 0 \\ -C & -qC + \tilde{\mathcal{B}}(\boldsymbol{\alpha}(t,q))\end{bmatrix}\begin{bmatrix}\boldsymbol{\alpha}(t,q)\\ \boldsymbol{\beta}(t,q)\end{bmatrix} + \begin{bmatrix}F(t,q)\\ \tilde{F}(t,q)\end{bmatrix}\right) \\ \begin{bmatrix}\boldsymbol{\alpha}(0,q)\\ \boldsymbol{\beta}(0,q)\end{bmatrix} &= \begin{bmatrix} M & 0 \\ 0 & M\end{bmatrix}^{-1}\begin{bmatrix}G\\ \tilde{G}\end{bmatrix}. \end{aligned} \tag{4.17}$$

4.2.1.2 Solution Sensitivity for Neumann-Dirichlet Boundary Conditions problem

Consider the problem

$$s_t(t,x,q) + [u(t,x,q)\, s(t,x,q)]_x - u_{xx}(t,x,q) - qs_{xx}(t,x,q) = f_q(t,x,q). \tag{4.18}$$

where $s(t,x,q) = u_q(t,x,q)$, $x \in [0,1]$ and $t \in [0,t_f]$. The boundary conditions given by

$$u_{q,x}(t,0,q) = s_x(t,0,q) = \delta_q, \qquad u_q(t,1,q) = s(t,1,q) = 0, \tag{4.19}$$

and the initial condition is

$$u_q(0,x,q) = s(0,x,q) = s_0(x,q). \tag{4.20}$$

Multiplying both sides by a test function $v(x)$ and integrating yields

$$\int_0^1 \left(s_t(t,x,q) + \left[u(t,x,q)\, s(t,x,q)\right]_x - u_{xx}(t,x,q) - q s_{xx}(t,x,q)\right) v(x)\, dx = \int_0^1 f_q(t,x,q) v(x)\, dx. \tag{4.21}$$

If $v(\cdot)$ is piecewise smooth, then $v'(x) \in L^2(0,1)$ so we can apply integration by parts and take advantage of the essential Dirichlet boundary conditions to obtain

$$\begin{aligned}\int_0^1 u_{xx}(t,x,q) v(x)\, dx &= u_x(t,x,q)v(x)|_0^1 - \int_0^1 u_x(t,x,q) v_x(x)\, dx \\ &= -\delta v(0) - \int_0^1 u_x(t,x,q) v_x(x)\, dx\end{aligned}$$

$$\begin{aligned}\int_0^1 s_{xx}(t,x,q) v(x)\, dx &= s_x(t,x,q)v(x)|_0^1 - \int_0^1 s_x(t,x,q) v_x(x)\, dx \\ &= -\delta_q v(0) - \int_0^1 s_x(t,x,q) v_x(x)\, dx\end{aligned}$$

$$\begin{aligned}\int_0^1 \left[u(t,x,q)\, s(t,x,q)\right]_x v(x)\, dx &= u(t,x,q)\, s(t,x,q)\, v(x)|_0^1 - \int_0^1 \left[u(t,x,q)\, s(t,x,q)\right] v_x(x)\, dx \\ &= -\left[u(t,0,q)\, s(t,0,q)\, v(0) + \int_0^1 u(t,x,q)\, s(t,x,q) v_x(x)\, dx\right].\end{aligned}$$

Thus (4.21) becomes the weak form of the Sensitivity problem with Neumann-Dirichlet boundary conditions given by

$$\begin{aligned}\int_0^1 s_t(t,x,q)\, v(x)\, dx - \int_0^1 u(t,x,q)\, s(t,x,q)\, v_x(x)\, dx + \int_0^1 u_x(t,x,q) v_x(x)\, dx \\ + q\int_0^1 s_x(t,x,q) v_x(x)\, dx + \left[\delta + q\delta_q - u(t,0,q)\, s(t,0,q)\right] v(0) \\ = \int_0^1 f_q(t,x,q) v(x)\, dx.\end{aligned} \tag{4.22}$$

Note that Equation 4.22 must hold for any piecewise smooth function $v(\cdot)$. Just as the Dirichlet boundary conditions case, we use the same piecewise linear basis functions and note that the coefficients of the basis functions depend not just on time, but also on the parameter q. Thus we write

$$u(t,x,q) \approx u^N(t,x,q) = \sum_{j=0}^{N} \alpha_j(t,q)\phi_j(x)$$
$$s(t,x,q) \approx s^N(t,x,q) = \sum_{j=0}^{N} \beta_j(t,q)\phi_j(x).$$

Then (4.22) becomes

$$\begin{aligned}\int_0^1 \left(\sum_{j=0}^{N} \beta_{j,t}(t,q)\phi_j(x)\right) v(x)\,dx - \int_0^1 \left(\sum_{j=0}^{N} \alpha_j(t,q)\phi_j(x)\right)\left(\sum_{k=0}^{N} \beta_k(t,q)\phi_k(x)\right) v_x(x)\,dx \\ + \int_0^1 \left(\sum_{j=0}^{N} \alpha_j(t,q)\phi_{j,x}(x)\right) v_x(x)\,dx + q\int_0^1 \left(\sum_{j=0}^{N} \beta_j(t,q)\phi_{j,x}(x)\right) v_x(x)\,dx \\ + \left[\delta + q\delta_q - \left(\sum_{j=0}^{N} \alpha_j(t,q)\phi_j(0)\right)\left(\sum_{k=0}^{N} \beta_k(t,q)\phi_k(0)\right)\right] v(0) \\ = \int_0^1 f_q(t,x,q)v(x)\,dx. \qquad (4.23)\end{aligned}$$

Since this identity holds for arbitrary piecewise smooth $v(x)$, for each $i = 0, 1, ..., N$, we can set $v(x) = \phi_i(x)$. Due to the local support we can simplify $\sum_{j=0}^{N} \alpha_j(t,q)\phi_j(0) = \alpha_0(t,q)$ and $\sum_{k=0}^{N} \beta_k(t,q)\phi_k(0) = \beta_0(t,q)$. Then from (4.23), for some i we obtain

$$\begin{aligned}\int_0^1 \left(\sum_{j=0}^{N} \beta_{j,t}(t,q)\phi_j(x)\right) \phi_i(x)\,dx - \int_0^1 \left(\sum_{j=0}^{N} \alpha_j(t,q)\phi_j(x)\right)\left(\sum_{k=0}^{N} \beta_k(t,q)\phi_k(x)\right) \phi_{i,x}(x)\,dx \\ + \int_0^1 \left(\sum_{j=0}^{N} \alpha_j(t,q)\phi_{j,x}(x)\right) \phi_{i,x}(x)\,dx + q\int_0^1 \left(\sum_{j=0}^{N} \beta_j(t,q)\phi_{j,x}(x)\right) \phi_{i,x}(x)\,dx \\ + \left[\delta + q\delta_q - \alpha_0(t,q)\,\beta_0(t,q)\right]\phi_i(0) = \int_0^1 f_q(t,x,q)\phi_i(x)\,dx \qquad (4.24)\end{aligned}$$

for each $i = 0, 1, ..., N$.

Since for all j, k, $\alpha_j(t,q)$, $\beta_k(t,q)$ do not depend on x we can move those terms outside the integral

to get

$$\sum_{j=0}^{N} \beta_{j,t}(t,q) \left(\int_0^1 \phi_j(x)\phi_i(x)\,dx \right) - \sum_{j=0}^{N}\sum_{k=0}^{N} \alpha_j(t,q)\beta_k(t,q) \left(\int_0^1 \phi_j(x)\phi_k(x)\,\phi_{i,x}(x)\,dx \right)$$
$$+ \sum_{j=0}^{N} \left(\alpha_j(t,q) + q\beta_j(t,q)\right) \left(\int_0^1 \phi_{j,x}(x)\phi_{i,x}(x)\,dx \right) + \left[\delta + q\delta_q - \alpha_0(t,q)\,\beta_0(t,q)\right]\phi_i(0)$$
$$= \int_0^1 f_q(t,x,q)\phi_i(x)\,dx. \tag{4.25}$$

For each $i = 0, 1, ..., N$, the equation (4.25) can be written as a system of $N+1$ ODEs given by

$$M\dot{\beta}(t,q) + \left[qC - \tilde{\mathcal{B}}\left(\alpha(t,q)\right)\right]\beta(t,q) + C\alpha(t) + \tilde{D}(t) \quad = \quad \tilde{F}(t,q) \tag{4.26}$$

where the matrices M and C are the same as (2.27) and $\tilde{F}$ is given by $\tilde{F}_i(t,q) = \langle f_q(t,x,q), \phi_i(x)\rangle$ for all $i = 0, 1, ..., N$ and $\tilde{\mathcal{B}}\left(\boldsymbol{\alpha}(t,q)\right)$ is a matrix that depends on $\boldsymbol{\alpha}(t,q)$. The matrix $\tilde{D}(t) = \left(\delta + q\delta_q - \alpha_0(t,q)\,\beta_0(t,q)\right)[1,\, 0,\, \ldots,\, 0]^T$ does depend upon the value at the boundary and therefore needs to be re-evaluated at every time step.

Taking advantage of the Dirichlet boundary conditions we know the value of both $\alpha_N(t,q)$ and $\beta_N(t,q)$ for all t and q. Thus we eliminate the last equation and reduce the size of (4.14) from $N+1$ to N equations by not solving for the $x = 1$ node. The matrix $\tilde{\mathcal{B}}\left(\boldsymbol{\alpha}(t,q)\right)$ has the tridiagonal form shown below suppressing the dependence on (t,q)

$$\tilde{\mathcal{B}}\left(\boldsymbol{\alpha}(t,q)\right) \quad = \quad \frac{1}{6}\begin{pmatrix} -(2\alpha_0+\alpha_1) & -(\alpha_0+2\alpha_1) & & & & \\ 2\alpha_0+\alpha_1 & \alpha_0-\alpha_2 & -(\alpha_1+2\alpha_2) & & & \\ & \ddots & \ddots & \ddots & & \\ & & 2\alpha_{N-2}+\alpha_{N-1} & \alpha_{N-2}-\alpha_N & -(\alpha_{N-1}+2\alpha_N) \\ & & & 2\alpha_{N-1}+\alpha_N & \alpha_{N-1} \end{pmatrix}.$$

We also need to find the Galerkin approximation to the initial conditions $s_0(x,q)$. Similar to before the initial condition is given by the matrix equation

$$M\boldsymbol{\beta}(0,q) \quad = \quad \tilde{G} \tag{4.27}$$

where $\tilde{G}_i = \langle s_0(x,q), \phi_i(x)\rangle$.

Combining (4.26) and (4.27) we obtain the initial value ODE system

$$\begin{aligned} M\dot{\boldsymbol{\beta}}(t,q) + \left[qC - \tilde{\mathcal{B}}\left(\boldsymbol{\alpha}(t,q)\right)\right]\boldsymbol{\beta}(t,q) + C\boldsymbol{\alpha}(t,q) + \tilde{D}(t) &= \tilde{F}(t,q) \\ M\boldsymbol{\beta}(0,q) &= \tilde{G}. \end{aligned} \tag{4.28}$$

We note that the sensitivity problem is dependent on the system given in (2.32) and as such we can formulate a single coupled system to solve for the solution and sensitivities at the same time.

Thus, from (2.32) and (4.28) we have

$$\begin{aligned} M\dot{\boldsymbol{\alpha}}(t,q) + \frac{1}{2}\mathcal{B}\left(\boldsymbol{\alpha}(t,q)\right)\boldsymbol{\alpha}(t,q) + q\left(D + C\boldsymbol{\alpha}(t,q)\right) &= F(t,q) \\ M\dot{\boldsymbol{\beta}}(t,q) + \left[qC - \tilde{\mathcal{B}}\left(\boldsymbol{\alpha}(t,q)\right)\right]\boldsymbol{\beta}(t,q) + C\boldsymbol{\alpha}(t,q) + \tilde{D}(t) &= \tilde{F}(t,q) \\ M\boldsymbol{\alpha}(0,q) &= G \\ M\boldsymbol{\beta}(0,q) &= \tilde{G} \end{aligned}$$

therefore the coupled system is given as

$$\begin{aligned} \begin{bmatrix}\dot{\boldsymbol{\alpha}}(t,q)\\ \dot{\boldsymbol{\beta}}(t,q)\end{bmatrix} &= \begin{bmatrix}M & 0\\ 0 & M\end{bmatrix}^{-1}\left(\begin{bmatrix}-qC - \frac{1}{2}\mathcal{B}\left(\boldsymbol{\alpha}(t,q)\right) & 0\\ -C & -qC + \tilde{\mathcal{B}}\left(\boldsymbol{\alpha}(t,q)\right)\end{bmatrix}\begin{bmatrix}\boldsymbol{\alpha}(t,q)\\ \boldsymbol{\beta}(t,q)\end{bmatrix} + \begin{bmatrix}F(t,q) - qD\\ \tilde{F}(t,q) - \tilde{D}(t)\end{bmatrix}\right) \\ \begin{bmatrix}\boldsymbol{\alpha}(0,q)\\ \boldsymbol{\beta}(0,q)\end{bmatrix} &= \begin{bmatrix}M & 0\\ 0 & M\end{bmatrix}^{-1}\begin{bmatrix}G\\ \tilde{G}\end{bmatrix}. \end{aligned} \tag{4.29}$$

4.2.2 Sensitivity Numerical Experiments

To verify our sensitivity computations we again use the MMS. In order to exercise the sensitivity problem we start with exact solutions that have an explicit dependence on the parameter so that we are not solving a trivial problem. Also, due to the coupled nature of the problem we have made the tolerances in the ODE solver stricter. This allows us to adequately demonstrate the order of convergence in the sensitivity problem solution. Here we use N to be the number of elements solving each part of the coupled problem so that we may compare the relative errors and orders of accuracy to the previous sections.

4.2.2.1 Dirichlet Boundary Conditions Sensitivity Problem

In this example we use the MMS to build a solution that has a nontrivial dependence upon the parameter.

Consider the Dirichlet boundary conditions problem with exact solution given by

$$u(t,x,q) = q^2 e^{-t}\sin(\pi x) \tag{4.30}$$
$$u_q(t,x,q) = 2qe^{-t}\sin(\pi x). \tag{4.31}$$

The initial conditions are given by

$$u_0(x,q) = q^2\sin(\pi x) \tag{4.32}$$
$$s_0(x,q) = 2q\sin(\pi x). \tag{4.33}$$

This meets the boundary conditions and the associated MMS forcing terms are

$$f(t,x,q) = -e^{-t}q^2\sin(\pi x) + \pi q^4 e^{-2t}\sin(\pi x)\cos(\pi x) + q^3\pi^2 e^{-t}\sin(\pi x) \tag{4.34}$$
$$f_q(t,x,q) = -2qe^{-t}\sin(\pi x) + 4\pi q^3 e^{-2t}\sin(\pi x)\cos(\pi x) + 3q^2\pi^2 e^{-t}\sin(\pi x). \tag{4.35}$$

Results for various parameter values are found in Table 4.1.

Parameter value	Elements	CPU time	Burgers' Err^{rel}	Order	Sensitivity Err^{rel}	Order
Re=10	$N = 8$	2.038	1.057×10^{-2}	-	1.167×10^{-2}	-
	$N = 16$	17.37	2.637×10^{-3}	2.0030	2.913×10^{-3}	2.0021
	$N = 32$	157.9	6.589×10^{-4}	2.0007	7.280×10^{-4}	2.0004
	$N = 64$	1268	1.647×10^{-4}	2.0002	1.820×10^{-4}	2.0001
	$N = 128$	10900	4.117×10^{-5}	2.0000	4.550×10^{-5}	2.0000
Re=50	$N = 8$	0.401	7.683×10^{-3}	-	8.556×10^{-3}	-
	$N = 16$	3.378	1.909×10^{-3}	2.0092	2.130×10^{-3}	2.0073
	$N = 32$	31.51	4.763×10^{-4}	2.0024	5.316×10^{-4}	2.0018
	$N = 64$	251.6	1.190×10^{-4}	2.0006	1.328×10^{-4}	2.0005
	$N = 128$	2183	2.976×10^{-5}	2.0002	3.321×10^{-5}	2.0001
Re=100	$N = 8$	0.206	6.750×10^{-3}	-	7.371×10^{-3}	-
	$N = 16$	1.617	1.673×10^{-3}	2.0120	1.830×10^{-3}	2.0103
	$N = 32$	15.30	4.173×10^{-4}	2.0033	4.565×10^{-4}	2.0027
	$N = 64$	129.1	1.043×10^{-4}	2.0009	1.140×10^{-4}	2.0007
	$N = 128$	1092	2.606×10^{-5}	2.0002	2.852×10^{-5}	2.0002
Re=200	$N = 8$	0.134	6.183×10^{-3}	-	6.502×10^{-3}	-
	$N = 16$	0.843	1.531×10^{-3}	2.0135	1.611×10^{-3}	2.0127
	$N = 32$	7.715	3.817×10^{-4}	2.0042	4.018×10^{-4}	2.0039
	$N = 64$	62.23	9.536×10^{-5}	2.0010	1.004×10^{-4}	2.0009
	$N = 128$	536.3	2.384×10^{-5}	2.0003	2.509×10^{-5}	2.0002
Re=400	$N = 8$	0.159	5.940×10^{-3}	-	6.060×10^{-3}	-
	$N = 16$	0.464	1.470×10^{-3}	2.0142	1.500×10^{-3}	2.0140
	$N = 32$	4.073	3.666×10^{-4}	2.0040	3.741×10^{-4}	2.0039
	$N = 64$	30.45	9.157×10^{-5}	2.0012	9.344×10^{-5}	2.0012
	$N = 128$	277.9	2.290×10^{-5}	2.0003	2.336×10^{-5}	2.0003

Table 4.1: FEM Sensitivity MMS Results, $T = 10$, Dirichlet boundary conditions

4.2.2.2 Neumann-Dirichlet Boundary Conditions Sensitivity Problem

Consider the Neumann-Dirichlet boundary conditions problem with $\delta = 0$ and exact solution

$$u(t,x) = q^2 e^{-t}\left(1-x^2\right) \tag{4.36}$$
$$u_q(t,x,q) = 2qe^{-t}\left(1-x^2\right). \tag{4.37}$$

The initial conditions are given by

$$u_0(x,q) = q^2\left(1-x^2\right) \tag{4.38}$$
$$s_0(x,q) = 2q\left(1-x^2\right). \tag{4.39}$$

This meets the boundary conditions and the associated MMS forcing terms are

$$f(t,x,q) = -q^2 e^{-t}\left(1-x^2\right) - 2q^4 e^{-2t}x\left(1-x^2\right) + 2q^3 e^{-t} \tag{4.40}$$
$$f_q(t,x,q) = 6q^2 e^{-t} - 2qe^{-t}\left(1-x^2\right) - 8q^3 e^{-2t}x\left(1-x^2\right). \tag{4.41}$$

Results for various parameter values are found in Table 4.2.

Parameter value	Elements	CPU time	Burgers' Err^{rel}	Order	Sensitivity Err^{rel}	Order
Re=10	$N = 8$	2.606	2.224×10^{-3}	-	2.396×10^{-3}	-
	$N = 16$	18.92	5.891×10^{-4}	1.9162	6.322×10^{-4}	1.9223
	$N = 32$	158.5	1.521×10^{-4}	1.9539	1.627×10^{-4}	1.9583
	$N = 64$	1278	3.865×10^{-5}	1.9759	4.129×10^{-5}	1.9785
	$N = 128$	10950	9.746×10^{-6}	1.9877	1.040×10^{-5}	1.9888
Re=50	$N = 8$	0.760	1.714×10^{-3}	-	1.826×10^{-3}	-
	$N = 16$	3.715	4.590×10^{-4}	1.9006	4.892×10^{-4}	1.9001
	$N = 32$	32.13	1.197×10^{-4}	1.9390	1.274×10^{-4}	1.9414
	$N = 64$	259.6	3.065×10^{-5}	1.9656	3.256×10^{-5}	1.9680
	$N = 128$	2203	7.760×10^{-6}	1.9816	8.236×10^{-6}	1.9830
Re=100	$N = 8$	0.463	1.615×10^{-3}	-	1.681×10^{-3}	-
	$N = 16$	2.029	4.305×10^{-4}	1.9073	4.501×10^{-4}	1.9005
	$N = 32$	15.34	1.122×10^{-4}	1.9394	1.175×10^{-4}	1.9381
	$N = 64$	130.3	2.877×10^{-5}	1.9639	3.009×10^{-5}	1.9647
	$N = 128$	1092	7.293×10^{-6}	1.9800	7.624×10^{-6}	1.9809
Re=200	$N = 8$	0.294	1.560×10^{-3}	-	1.598×10^{-3}	-
	$N = 16$	1.165	4.123×10^{-4}	1.9190	4.253×10^{-4}	1.9094
	$N = 32$	7.874	1.072×10^{-4}	1.9437	1.109×10^{-4}	1.9397
	$N = 64$	63.09	2.747×10^{-5}	1.9643	2.842×10^{-5}	1.9636
	$N = 128$	549.2	6.966×10^{-6}	1.9793	7.207×10^{-6}	1.9796
Re=400	$N = 8$	0.201	1.529×10^{-3}	-	1.550×10^{-3}	-
	$N = 16$	0.688	4.008×10^{-4}	1.9316	4.090×10^{-4}	1.9218
	$N = 32$	4.142	1.037×10^{-4}	1.9506	1.062×10^{-4}	1.9449
	$N = 64$	31.45	2.653×10^{-5}	1.9666	2.272×10^{-5}	1.9645
	$N = 128$	275.7	6.727×10^{-6}	1.9796	6.903×10^{-6}	1.9792

Table 4.2: FEM Sensitivity MMS Results, $T = 10$, Neumann-Dirichlet boundary conditions

4.3 Modifying POD Basis using Sensitivity

The POD basis sensitivities describes how each vector is expected to change as the parameter changes. Therefore we can use this information to modify the baseline POD basis to account for changes in the parameter. We will outline two methods, the first is by extrapolating the POD basis vectors, the second is by expanding the POD basis.

4.3.1 Extrapolated POD basis

Given a baseline POD Basis vector that depends on the parameter q, ϕ_i^r and the corresponding sensitivity vector $[\phi_i^r]^q$, we approximate the appropriate POD basis vector for a new parameter $\tilde{q}$ in our range of consideration using the truncated Taylor series approximation.

$$\phi_i^r(x,\tilde{q}) \quad = \quad \phi_i^r(x,q_0) + (\tilde{q}-q_0)\,[\phi_i^r]^q\,. \tag{4.42}$$

The Extrapolated basis ROM assumes the POD basis vectors depend linearly on the difference in the parameter values. The Extrapolated POD algorithm steps are shown below.

Extrapolated POD steps:

[1] Select baseline parameter q_0

[2] Using q_0, compute a snapshot matrix $[Y_{2n\times m}]$ for the coupled problem

[3] Generate POD basis and sensitivity vectors

[4] For a new parameter $\tilde{q}$, compute extrapolated basis using (4.42)

4.3.2 Expanded POD basis

We note that the POD basis vectors and the corresponding sensitivity vectors have been chosen to be orthonormal. Therefore it is natural to hope that the sensitivity vectors are linearly independent to the POD basis vectors. Therefore we expand the basis by including the sensitivity vectors. We note that this expanded basis has $2r$ vectors and thus the solution is modified from the solution given in (3.2) to

$$u^N(t,x) = \sum_{j=1}^{r} \alpha_j^r(t)\phi_j^r(x) + \sum_{j=r+1}^{2r} \alpha_j^r(t)\left[\phi_{j-r}^r\right]^q(x). \tag{4.43}$$

In this case the basis may not be orthonormal and so the reduced order model mass matrix $M_{ij}^r = \left\langle \phi_j^r(x), \phi_i^r(x) \right\rangle$ may not be the identity of dimension r. Thus the resulting reduced order models are modified to be

$$\begin{aligned} M^r\dot{\boldsymbol{\alpha}}^r(t) &= \left[F^r(t) - \frac{1}{2}\left(\Phi^r\right)^T \mathcal{B}\left(\Phi^r\boldsymbol{\alpha}^r(t)\right)\Phi^r\boldsymbol{\alpha}^r(t) - qC^r\alpha^r(t)\right] \\ M^r\boldsymbol{\alpha}^r(0) &= \left(\Phi^r\right)^T\boldsymbol{\alpha}(0) \end{aligned} \tag{4.44}$$

in the Dirichlet boundary conditions problem and

$$M^r\dot{\boldsymbol{\alpha}}^r(t) = \left[F^r(t) - \frac{1}{2}\left(\Phi^r\right)^T \mathcal{B}\left(\Phi^r\boldsymbol{\alpha}^r(t)\right)\Phi^r\boldsymbol{\alpha}^r(t) - q\left(D^r + C^r\boldsymbol{\alpha}^r(t)\right)\right] \tag{4.45}$$

$$M^r\boldsymbol{\alpha}^r(0) = \left(\Phi^r\right)^T\boldsymbol{\alpha}(0) \tag{4.46}$$

in the Neumann-Dirichlet boundary conditions problem.

The Expanded basis ROM does not assume the POD basis vectors depend linearly on the difference in parameter values. By expanding the basis we allow for any linear combination of the POD and sensitivity vectors to generate a ROM. The Expanded POD algorithm steps are shown below.

Extrapolated POD steps:

[1] Select baseline parameter q_0

[2] Using q_0, compute a snapshot matrix $[Y_{2n\times m}]$ for the coupled problem

[3] Generate POD basis and sensitivity vectors

[4] For a new parameter $\tilde{q}$, compute ROM using the collection of POD basis vectors and sensitivity vectors as Expanded basis

4.4 Numerical Results

In §3.4 we demonstrated the dependence of the POD basis on the parameter. Here we will illustrate the ability of the sensitivities to provide a correction to a baseline POD basis. As before, we obtain a POD basis of various dimensions using a nominal parameter $q_0 = 1/100$. Next we generate a variety of ROM approximations using the fixed POD basis computed from the baseline parameter, Global POD basis, Extrapolated Basis and finally the Expanded Basis for a variety of q parameters ranging from 1/10 to 1/400. For each q parameter we compare the ROM to a full dimension FEM solution and plot the relative errors. Recall that the expanded basis has both POD basis vectors and sensitivity vectors. Since the focus of reduced order modeling is usually real time computation, we want to maintain the same size bases for comparison purposes. This means that if $r = 10$ a proper comparison would include a standard POD basis, Global POD basis, and Extrapolated POD basis of dimension 10 and an Expanded POD basis consisting of 5 standard POD basis vectors together with the 5 corresponding sensitivity vectors.

4.4.1 Dirichlet Boundary Conditions Problem

Here we begin with the same system given in §3.3.2. Thus, consider the Dirichlet boundary conditions problem with an initial condition given by

$$u_0(x,q) = \begin{cases} 1, & \text{if } x \in (0, \frac{1}{4}] \\ 0, & \text{otherwise} \end{cases} \tag{4.47}$$

and forcing term

$$f(t,x,q) \quad = \quad 0. \tag{4.48}$$

The solution sensitivity problem is defined as the Dirichlet boundary conditions problem with an initial condition given by

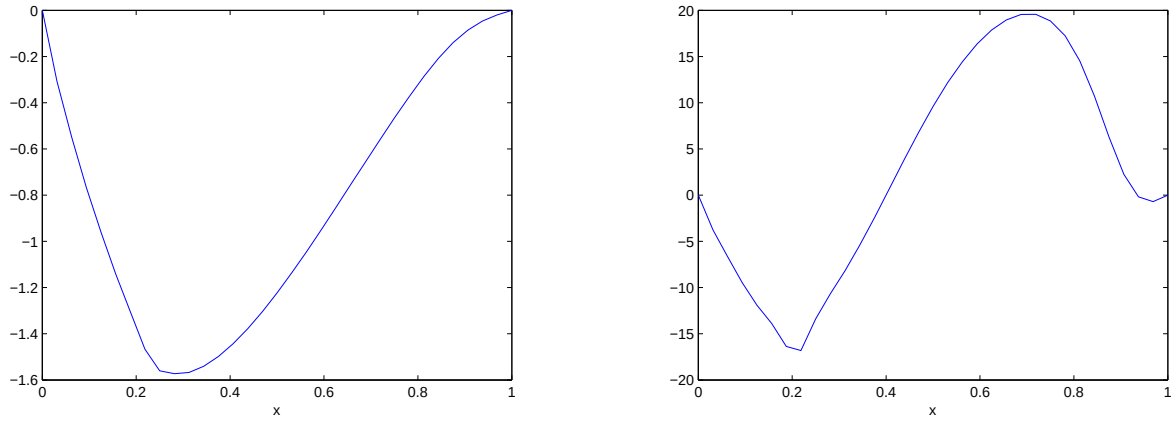
$$s_0(x,q) = 0. \tag{4.49}$$

and forcing term

$$f_q(t,x,q) \quad = \quad 0. \tag{4.50}$$

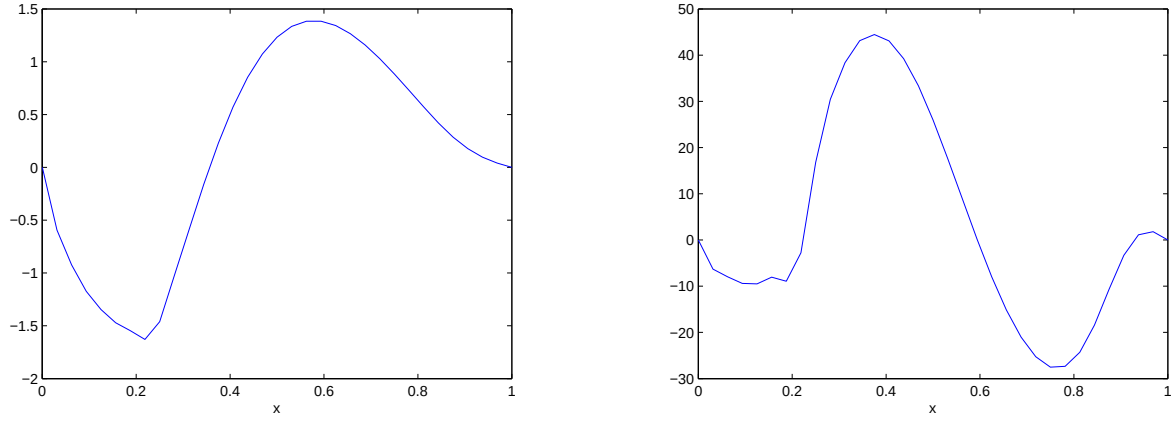
The first 5 POD basis vectors and corresponding sensitivity vectors are shown in Figures 4.5, 4.6, 4.7, 4.8, 4.9. We note that the maximum magnitude of the sensitivity vectors is greater than the previous vector. This implies that the first few POD vectors change the least as we vary the parameter. Therefore for low dimensions the Extrapolated POD basis will perform relatively closely to the baseline POD model. To demonstrate the accuracy of the extrapolated basis vectors, we use the nominal parameter of $q_0 = 1/100$ and the extrapolation method to approximate the true POD basis vectors for $\tilde{q} = 1/300$. The baseline POD vector $\phi_i^r(x, q_0)$, new parameter POD vector $\phi_i^r(x, \tilde{q})$, and the estimated new parameter vector $\hat{\phi}_i^r(x, \tilde{q})$ are compared in Figures 4.10, 4.11, 4.12, 4.13, 4.14. Next in Figures 4.15, 4.16, 4.17, and 4.18 we show the relative errors across our parameter range using the different POD models for various dimensions. In Figure 4.15 there is virtually no difference between the Baseline POD, Global POD and Extrapolated POD models across the entire parameter range. The Expanded POD model performs slightly better at smaller q values, but performs worse at larger q values. In Figure 4.16 the Global POD model performs similar to the Baseline POD model though slightly higher error at larger q values. The Extrapolated POD model has comparable error to the Baseline POD model with better performance in small local neighborhood about the baseline parameter, then higher error at the largest q values. The Expanded POD model does not perform as well as the Baseline POD model except at the largest q values. In Figure 4.17 the Baseline POD, Global POD and Extrapolated POD models perform similarly for $1/q \geq 125$. Again, in a local neighborhood the Extrapolated POD model has better performance and the Global POD higher error for $1/q < 125$. The Expanded POD model does perform better than the Baseline POD model for $1/q < 50$ and higher error for $1/q \geq 50$. In Figure 4.18 the Global POD model has a more consistent error across the entire parameter range, better that the Baseline POD model for most of the parameter range, but almost an order of magnitude larger error at the baseline parameter. The Extrapolated POD model performs comparably to the Baseline POD model for $1/q \geq 70$ but performing significantly higher for $1/q < 50$. Again, the Expanded POD model performs better than the Baseline POD model for $1/q < 70$ but now has an error more comparable across the entire parameter range.

Finally we mention that the total time to compute the four test parameter solutions and solve for Global POD basis is 7.76 seconds, while the time to solve the coupled system and solve for the POD basis and sensitivity vectors takes only 5.80 seconds, about 25% less time.



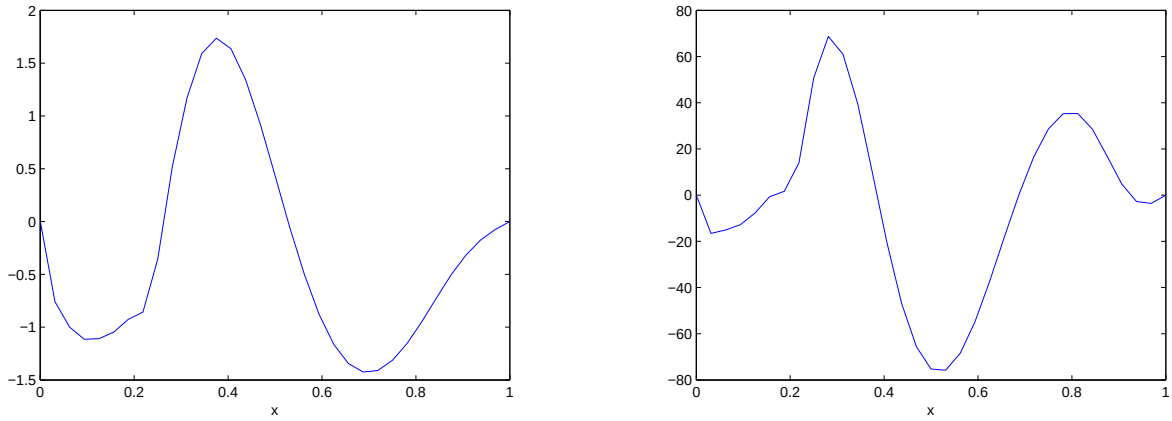
(a) $\phi_1^r(x)$ (b) $[\phi_1^r]^q(x)$

Figure 4.5: POD Basis and Sensitivity vector for $N = 32$, $Re = 100$ and Dirichlet boundary conditions



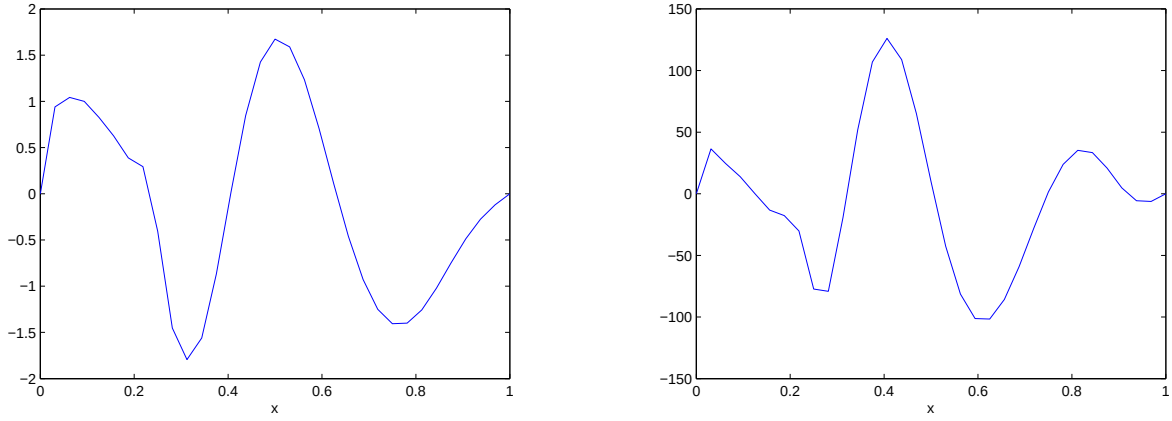
(a) $\phi_2^r(x)$ (b) $[\phi_2^r]^q(x)$

Figure 4.6: POD Basis and Sensitivity vector for $N = 32$, $Re = 100$ and Dirichlet boundary conditions



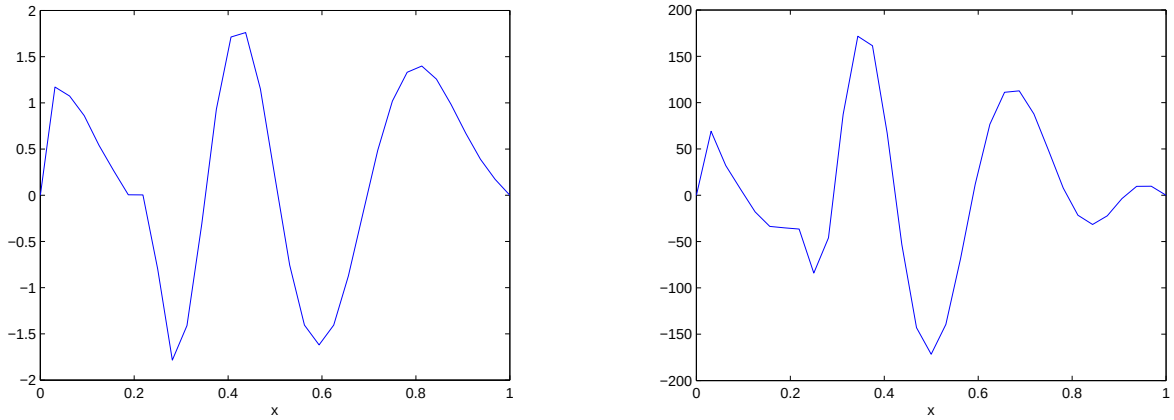
(a) $\phi_3^r(x)$ (b) $[\phi_3^r]^q(x)$

Figure 4.7: POD Basis and Sensitivity vector for $N = 32$, $Re = 100$ and Dirichlet boundary conditions



(a) $\phi_4^r(x)$ (b) $[\phi_4^r]^q(x)$

Figure 4.8: POD Basis and Sensitivity vector for $N = 32$, $Re = 100$ and Dirichlet boundary conditions



(a) $\phi_5^r(x)$ (b) $[\phi_5^r]^q(x)$

Figure 4.9: POD Basis and Sensitivity vector for $N = 32$, $Re = 100$ and Dirichlet boundary conditions

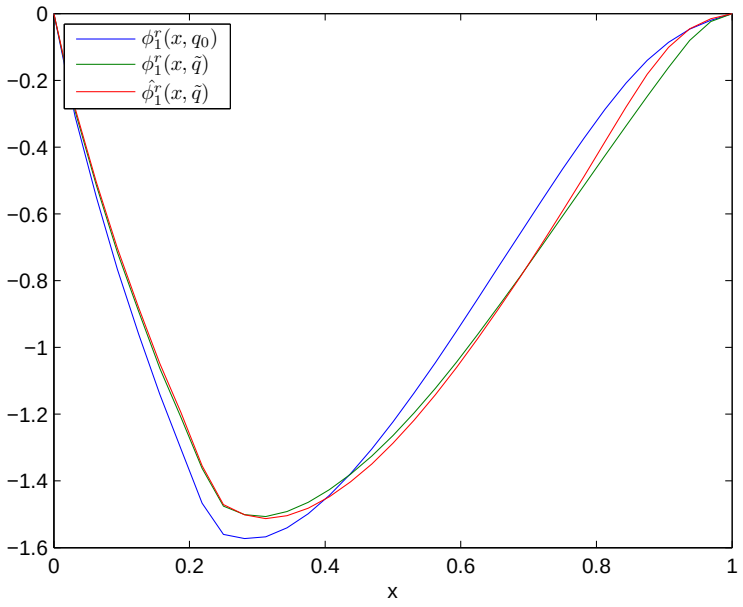


Figure 4.10: Comparison of actual and Estimated basis vector using Extrapolation, Dirichlet boundary conditions

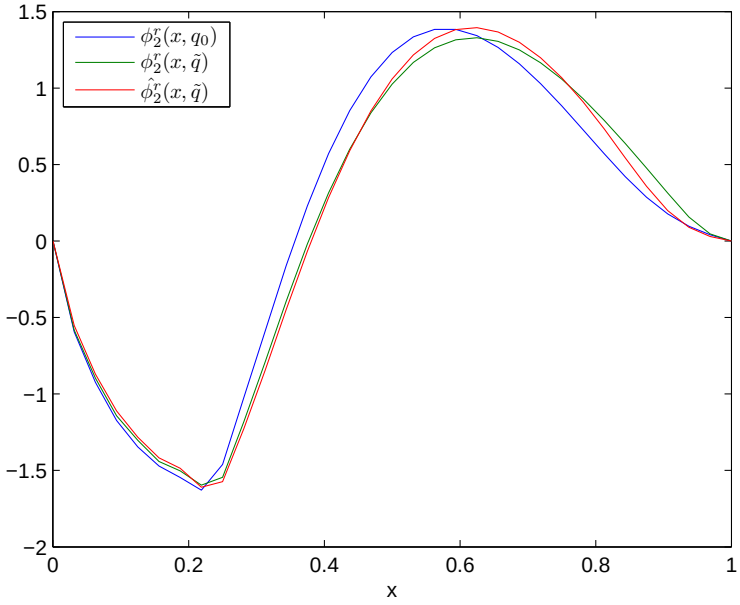


Figure 4.11: Comparison of actual and Estimated basis vector using Extrapolation, Dirichlet boundary conditions

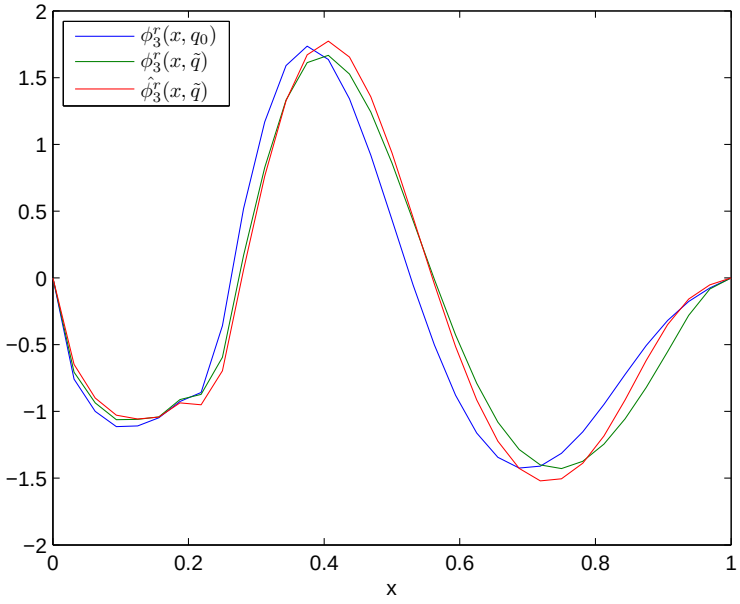


Figure 4.12: Comparison of actual and Estimated basis vector using Extrapolation, Dirichlet boundary conditions

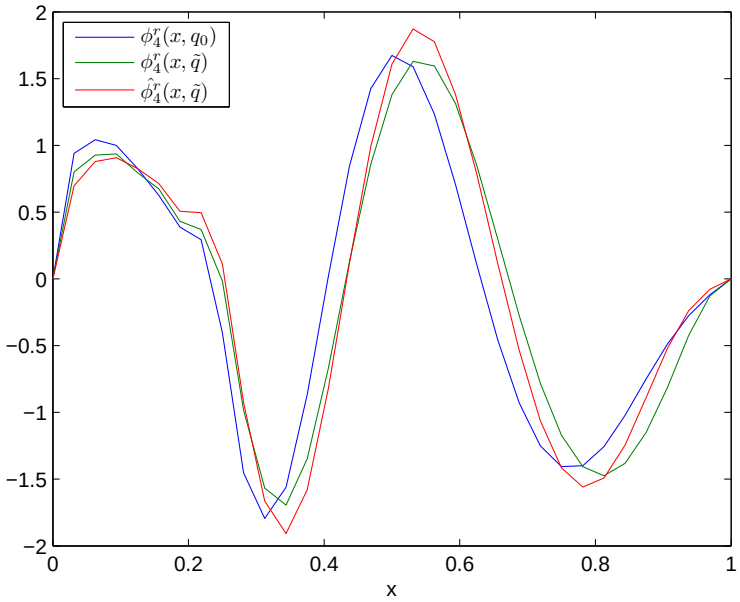


Figure 4.13: Comparison of actual and Estimated basis vector using Extrapolation, Dirichlet boundary conditions

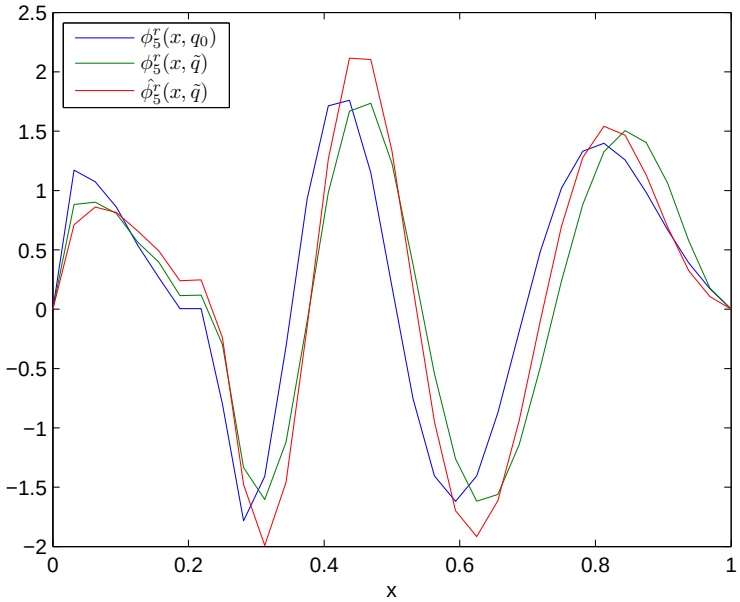


Figure 4.14: Comparison of actual and Estimated basis vector using Extrapolation, Dirichlet boundary conditions

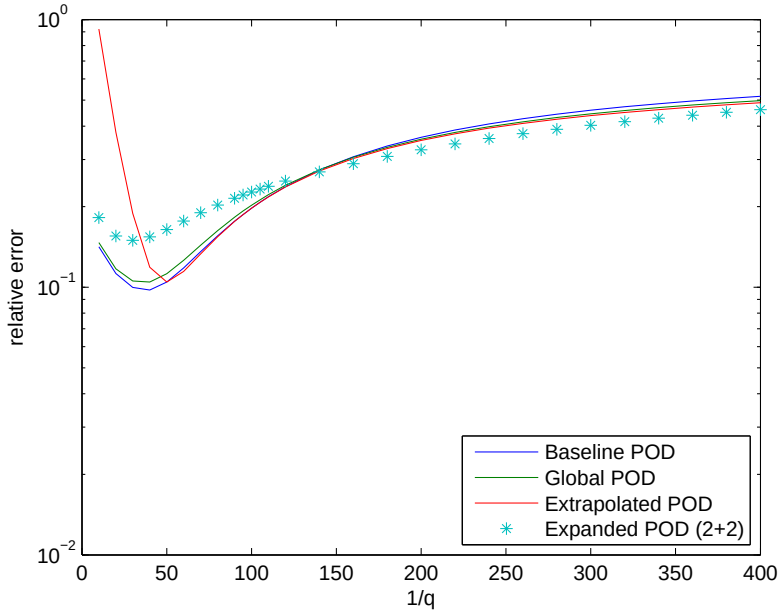


Figure 4.15: Comparison of Baseline, Global, Extrapolated and Expanded POD Models for $r = 4$, Dirichlet boundary conditions

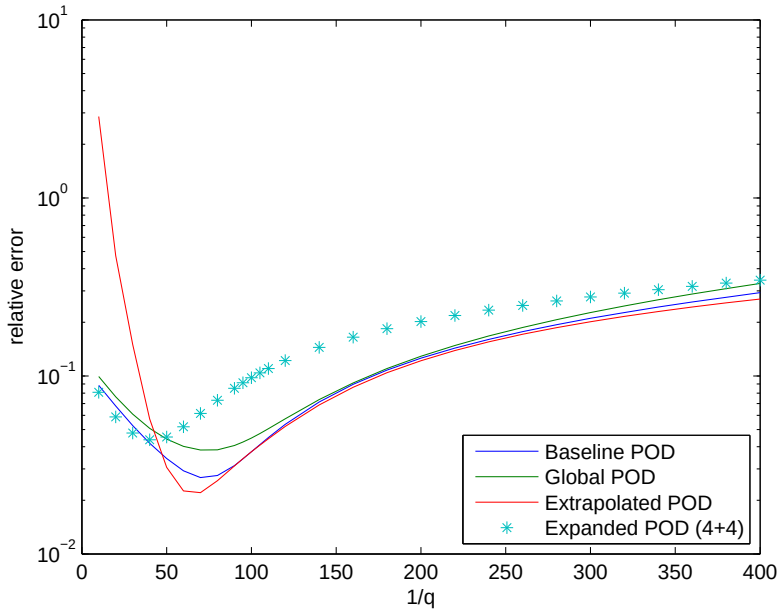


Figure 4.16: Comparison of Baseline, Global, Extrapolated and Expanded POD Models for $r = 8$, Dirichlet boundary conditions

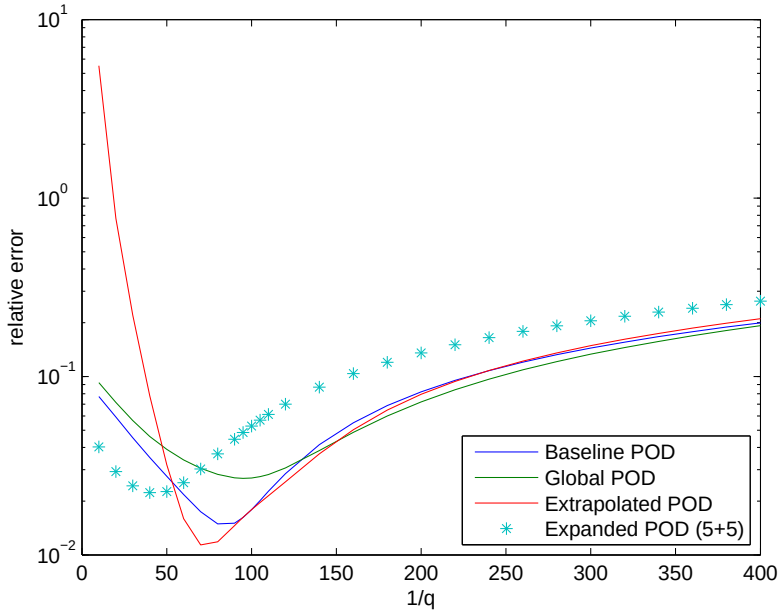


Figure 4.17: Comparison of Baseline, Global, Extrapolated and Expanded POD Models for $r = 10$, Dirichlet boundary conditions

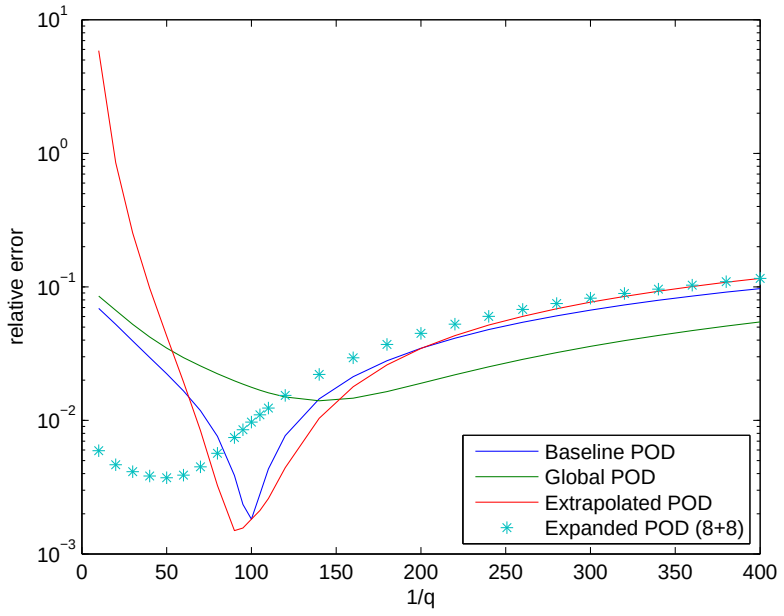


Figure 4.18: Comparison of Baseline, Global, Extrapolated and Expanded POD Models for $r = 16$, Dirichlet boundary conditions

4.4.2 Neumann-Dirichlet Boundary Conditions Problem

Here we begin with the same system given in §3.3.4. Consider the Neumann-Dirichlet boundary conditions problem with an initial condition given by

$$u_0(x,q) = \begin{cases} \frac{1}{2}\left(1-\cos 8\pi x\right), & \text{if } x \in (0, \frac{1}{4}] \\ 0, & \text{otherwise} \end{cases} \tag{4.51}$$

and forcing term

$$f(t,x,q) \;\; = \;\; 0. \tag{4.52}$$

The solution sensitivity problem is defined as the Neumann-Dirichlet boundary conditions problem with an initial condition given by

$$s_0(x,q) = 0 \tag{4.53}$$

and forcing term

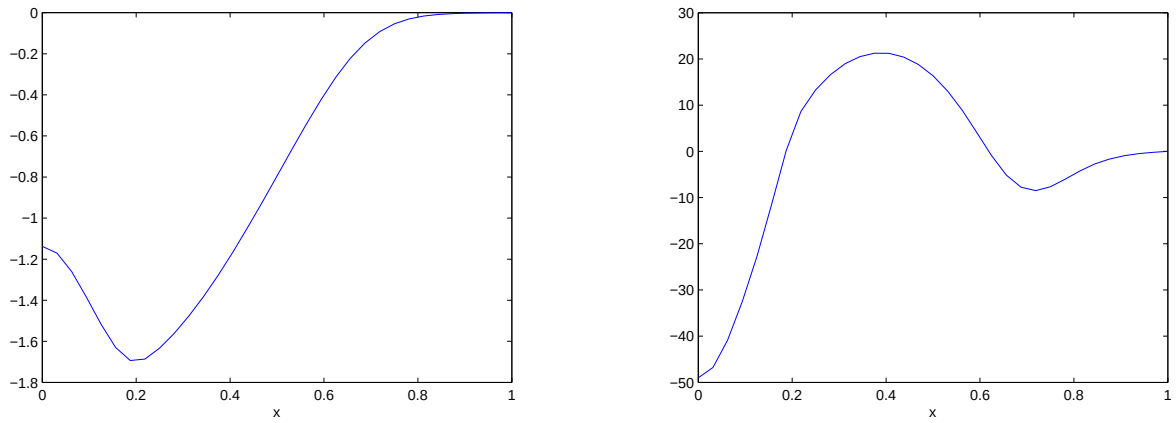
$$f_q(t,x,q) \;\; = \;\; 0. \tag{4.54}$$

The first 5 POD basis vectors and corresponding sensitivity vectors are shown in Figures 4.19, 4.20, 4.21, 4.22, 4.23. Then to demonstrate the accuracy of the extrapolated basis vectors, we use the nominal parameter of $q_0 = 1/100$ and the extrapolation method to approximate the true POD basis vectors for $\tilde{q} = 1/300$, the baseline POD vector $\phi_i^r(x, q_0)$, new parameter POD vector $\phi_i^r(x, \tilde{q})$, and the estimated new parameter vector $\hat{\phi}_i^r(x, \tilde{q})$ are compared in Figures 4.24, 4.25, 4.26, 4.27, 4.28.

It is worth noting that for the Neumann-Dirichlet problem occasionally using the extrapolated basis failed to converge using ODE45, when this is the case the relative error has been set to a value of 10^2 to differentiate it from the convergent results. In Figures 4.29, 4.30, 4.31, and 4.32 we show the relative errors across our parameter range using the different POD models for various dimensions of bases. Across the entire range of model dimensions the Extrapolated POD model performs at least as well as the Baseline POD model for $1/q \geq 50$. In Figure 4.29 there is very little difference between the Baseline POD and Global POD models across the entire parameter range. The Expanded POD model performs better at smaller q values, but performs worse at larger q values. The Extrapolated POD model has lower error across much of the parameter range but did fail to converge for some larger q values. In Figure 4.30 the Global POD model has a more consistent error than the Baseline POD model across the entire parameter range and lower error for most of the parameter range except in a local neighborhood of the baseline parameter. The Extrapolated POD model has better performance for than the Global POD for all $1/q \geq 75$. The Expanded POD model has the best performance for $1/q \leq 75$ and error comparable the Baseline POD model most everywhere else. In Figure 4.31 In this case we see much the same as for Figure 4.30, with the exception that now the Expanded POD model has better performance that the Baseline POD model for the entire parameter range except at the baseline parameter. In Figure 4.32 we see the dramatic dependence upon the parameter where the Baseline POD model error is almost 3 orders of magnitude smaller than the Global POD model at the baseline parameter. Conversely, everywhere

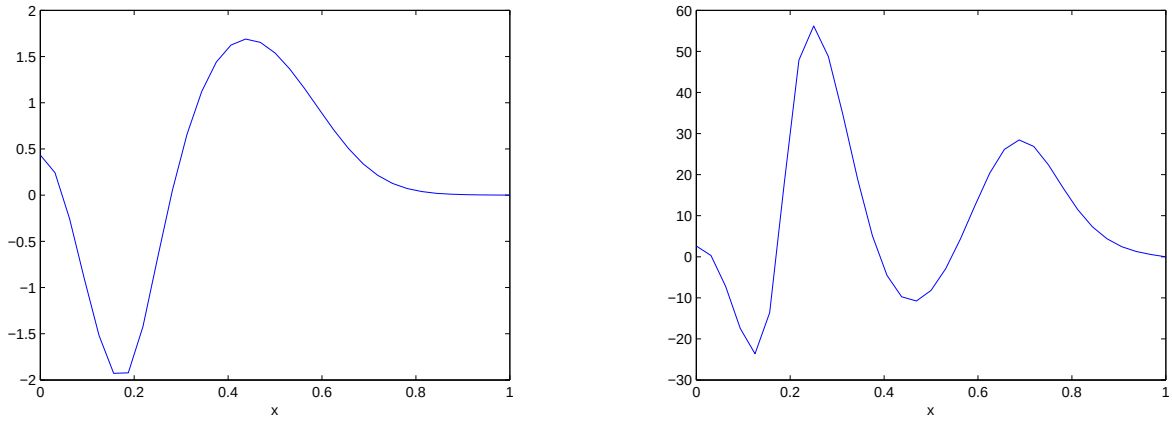
outside of a very small neighborhood of the baseline parameter the Global POD model has an error almost 2 orders of magnitude smaller than the baseline parameter. The Extrapolated POD model error is at least as good a the Baseline POD model for all $1/q \leq 50$ and nearly an order of magnitude better for a majority of the parameter range. The Expanded POD model like the Global POD model also performs better than the Baseline POD model for all but a small neighborhood of the baseline parameter.

Finally we mention that the total time to compute the four test parameter solutions and solve for Global POD basis is 6.42 seconds, while the time to solve the coupled system and solve for the POD basis and sensitivity vectors takes only 4.45 seconds, about 30% less time.



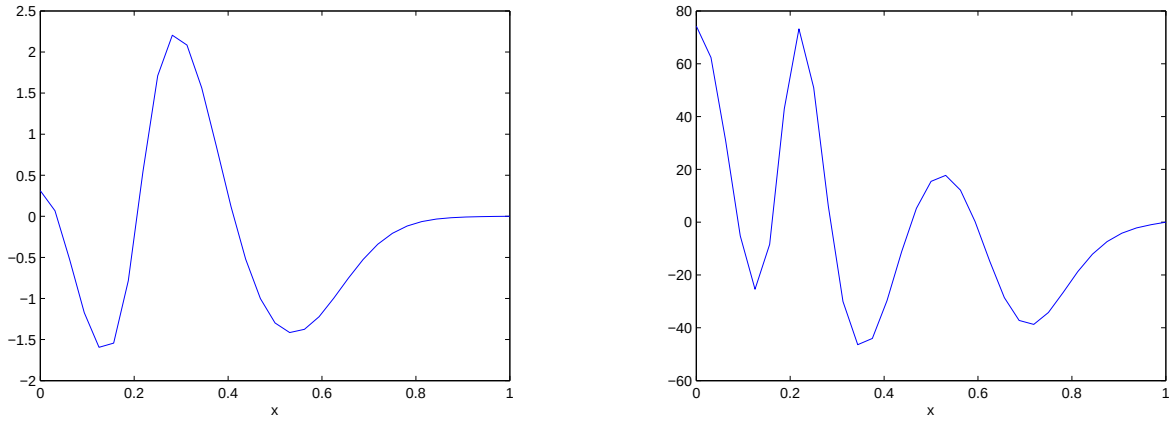
(a) $\phi_1^r(x)$ (b) $[\phi_1^r]^q(x)$

Figure 4.19: POD Basis and Sensitivity vector for $N = 32$, $Re = 100$ and Neumann-Dirichlet boundary conditions



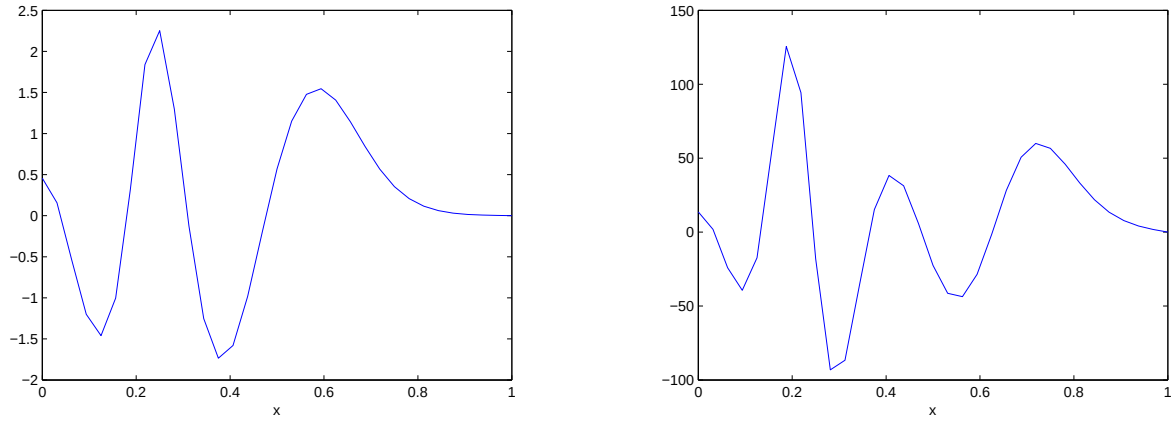
(a) $\phi_2^r(x)$ (b) $[\phi_2^r]^q(x)$

Figure 4.20: POD Basis and Sensitivity vector for $N = 32$, $Re = 100$ and Neumann-Dirichlet boundary conditions



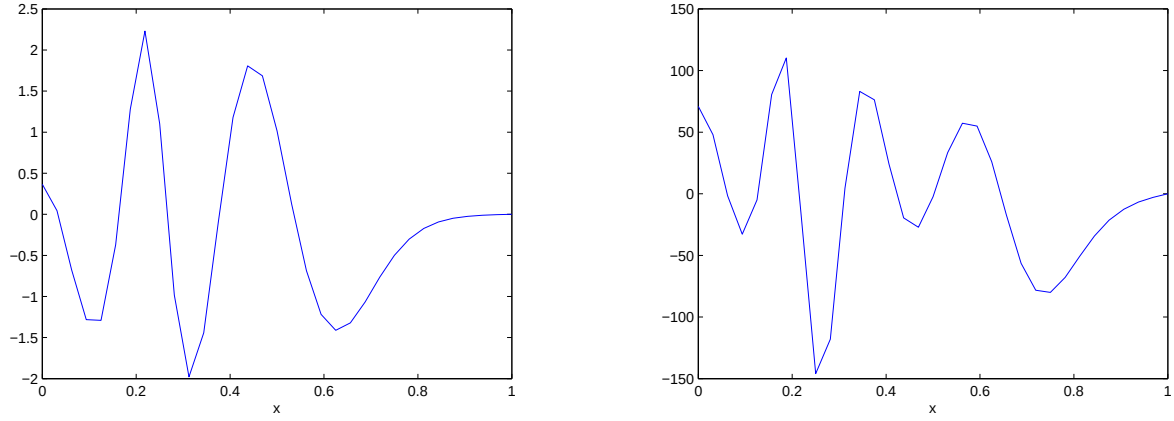
(a) $\phi_3^r(x)$ (b) $[\phi_3^r]^q(x)$

Figure 4.21: POD Basis and Sensitivity vector for $N = 32$, $Re = 100$ and Neumann-Dirichlet boundary conditions



(a) $\phi_4^r(x)$ (b) $[\phi_4^r]^q(x)$

Figure 4.22: POD Basis and Sensitivity vector for $N = 32$, $Re = 100$ and Neumann-Dirichlet boundary conditions



(a) $\phi_5^r(x)$ (b) $[\phi_5^r]^q(x)$

Figure 4.23: POD Basis and Sensitivity vector for $N = 32$, $Re = 100$ and Neumann-Dirichlet boundary conditions

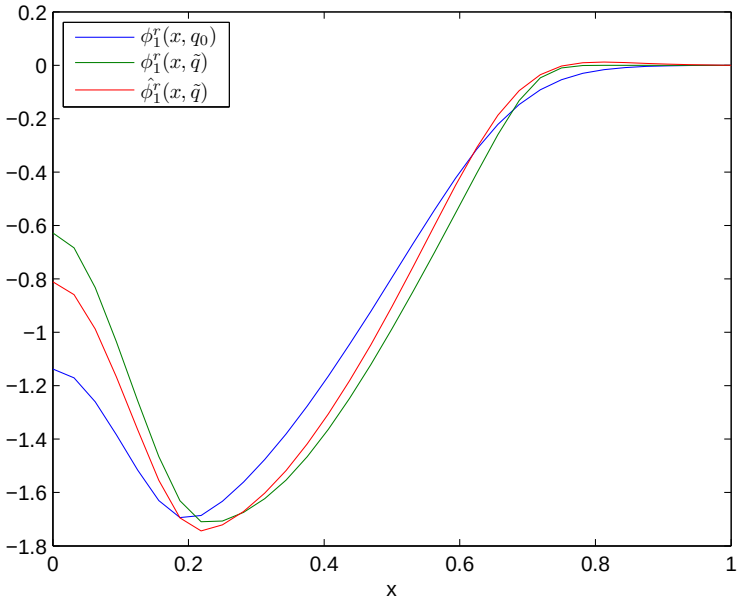


Figure 4.24: Comparison of actual and Estimated basis vector using Extrapolation, Neumann-Dirichlet boundary conditions

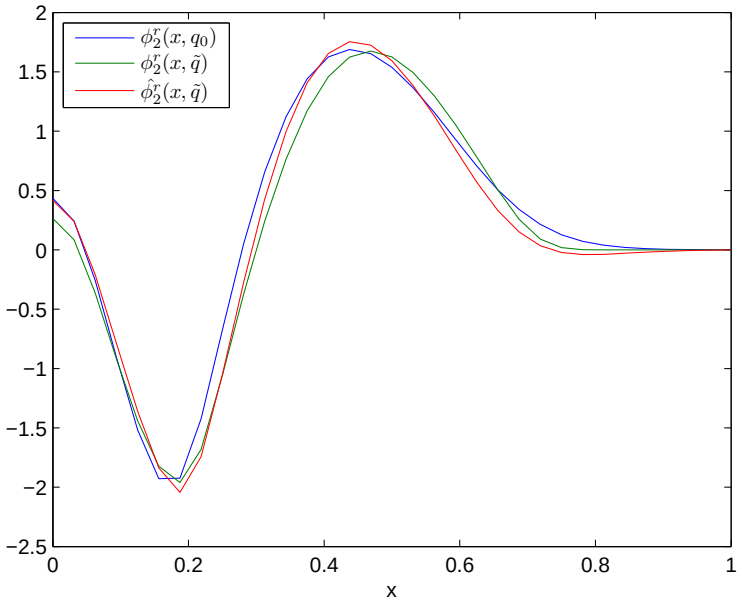


Figure 4.25: Comparison of actual and Estimated basis vector using Extrapolation, Neumann-Dirichlet boundary conditions

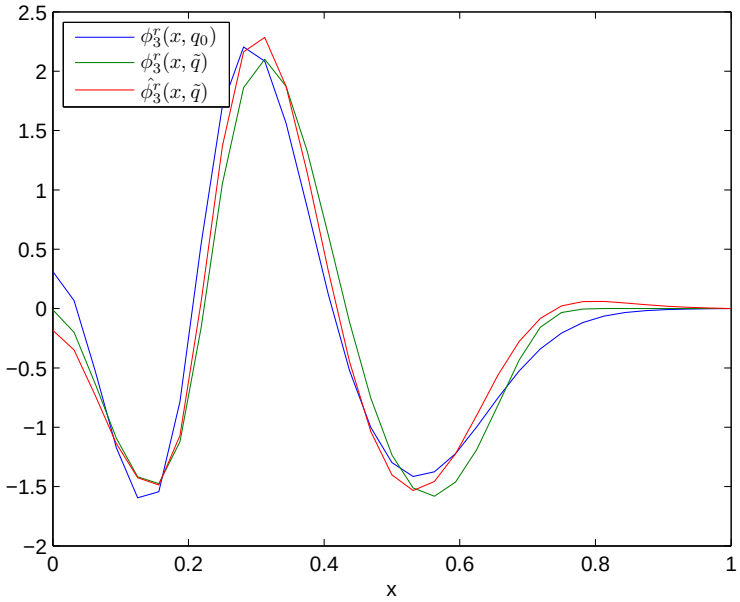


Figure 4.26: Comparison of actual and Estimated basis vector using Extrapolation, Neumann-Dirichlet boundary conditions

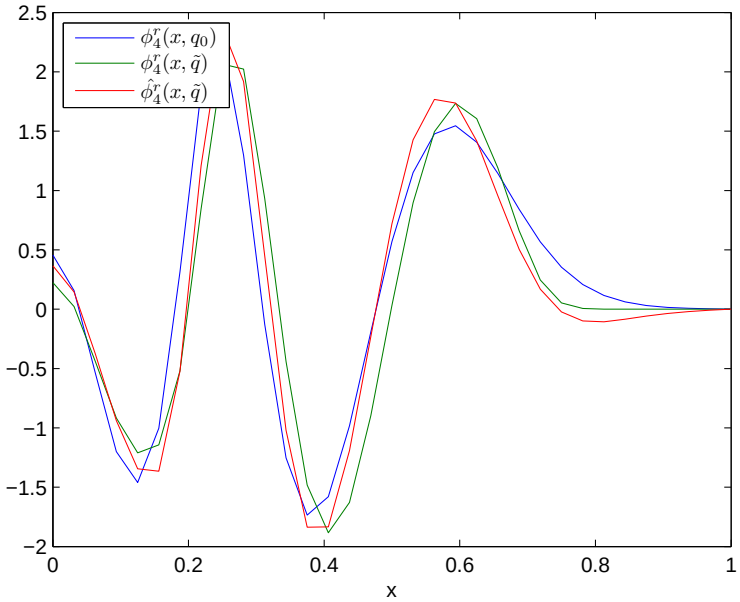


Figure 4.27: Comparison of actual and Estimated basis vector using Extrapolation, Neumann-Dirichlet boundary conditions

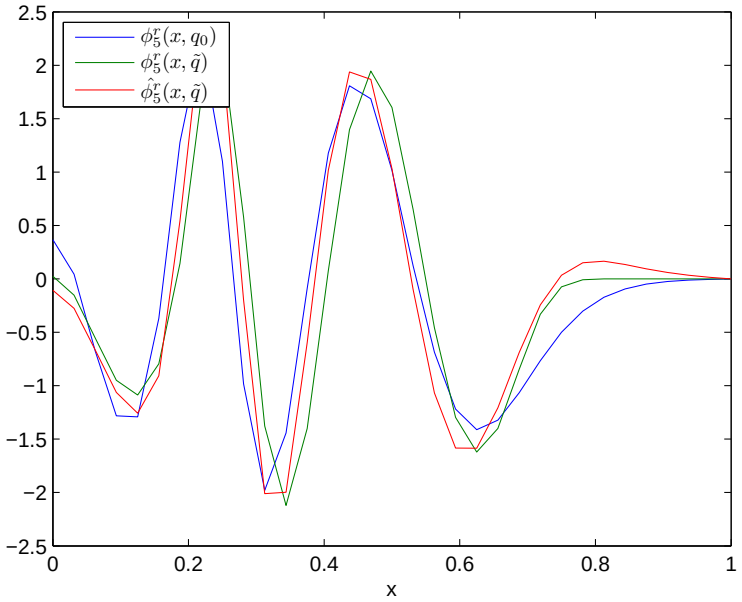


Figure 4.28: Comparison of actual and Estimated basis vector using Extrapolation, Neumann-Dirichlet boundary conditions

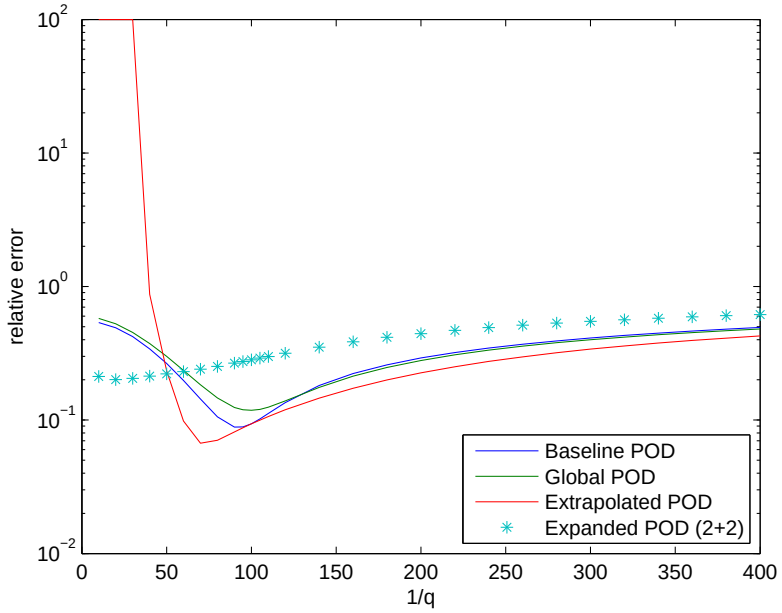


Figure 4.29: Comparison of Baseline, Global, Extrapolated and Expanded POD Models for $r = 4$, Neumann-Dirichlet boundary conditions

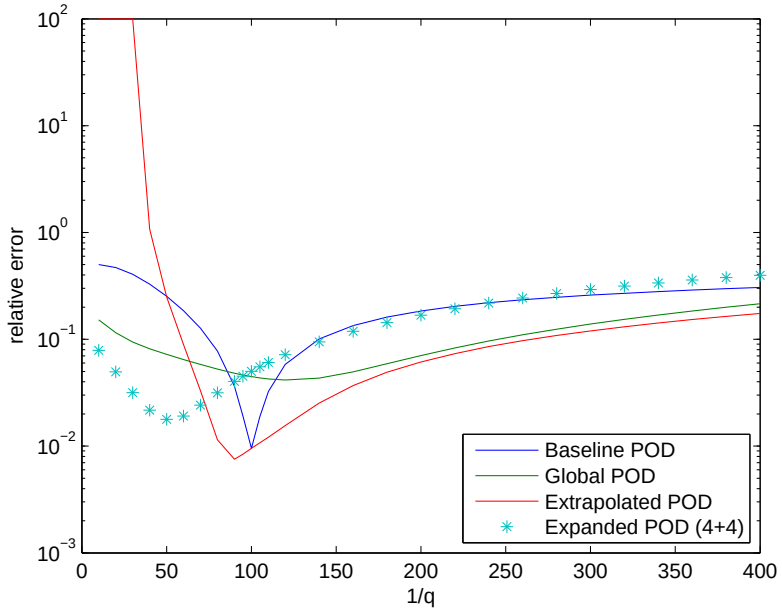


Figure 4.30: Comparison of Baseline, Global, Extrapolated and Expanded POD Models for $r = 8$, Neumann-Dirichlet boundary conditions

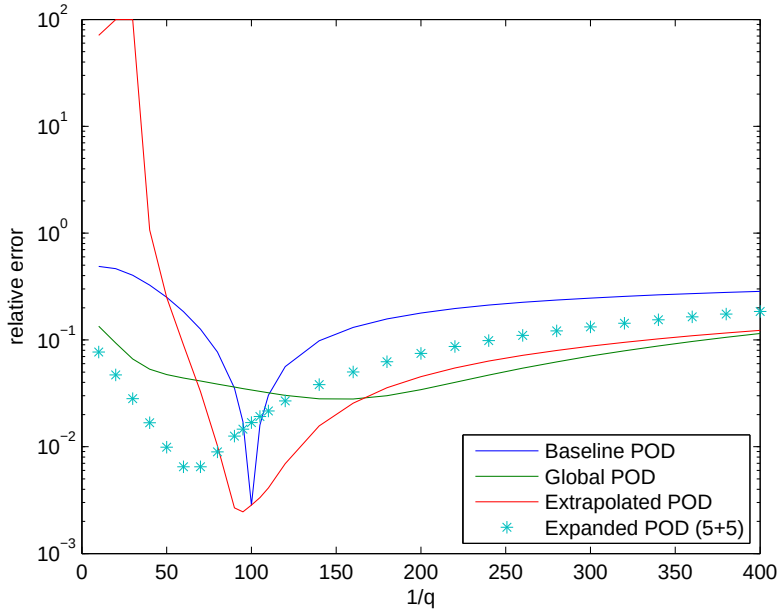


Figure 4.31: Comparison of Baseline, Global, Extrapolated and Expanded POD Models for $r = 10$, Neumann-Dirichlet boundary conditions

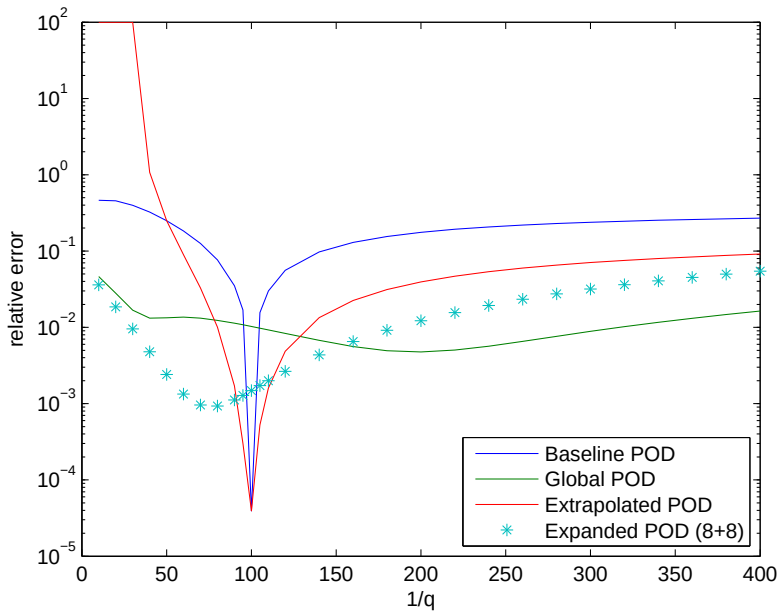


Figure 4.32: Comparison of Baseline, Global, Extrapolated and Expanded POD Models for $r = 16$, Neumann-Dirichlet boundary conditions

Chapter 5

Conclusions

5.1 Overview of Results

In this work, we studied Reduced Order Models (ROMs) of Burgers' equation using Proper Orthogonal Decomposition and modifications that enhance the applicability of the POD basis over a range of parameters. We used the Finite Element Method (FEM) and the Group Finite Element Method (GFE) where applicable to produce a high dimensional model which all ROMs were compared against. Though we initially tested various MATLAB® ODE solvers: ODE45, ODE23 and ODE15s, we choose to utilize ODE45 based on the requirement for a high fidelity full dimensional model for our comparisons.

We verified the FEM code for the Burgers' equation model, the ROM and later the sensitivity problem associated with Burgers' equation using the Method of Manufactured Solutions (MMS) and observed convergence over the relevant levels of discretization. Next, we noted the dependence of POD basis upon the parameter and implemented several methods that may allow reutilization of a basis over a range of parameters without needed to recompute a new basis for every parameter value.

First and easiest is the use of the Global POD basis. This method applies the POD method to a collection of solutions taken from a representative sample of parameters. Next we employed the information obtained from solving for the POD basis sensitivities. The disadvantage is that we must solve the coupled system given in §4.2.1 which has size $2N$ when N is the size of the FEM system. The first sensitivity approach is to provide a correction to the baseline parameter POD basis based on linear extrapolation of the parameter value desired. The second was to simply expand the basis by including the sensitivity vectors. The expanded basis provides the flexibility of providing any linear combination of the baseline POD basis and the sensitivity information not just a linear extrapolation update dependent on the difference in parameter values.

5.2 Conclusions

The motivation for this research is to demonstrate computational tools that can be used for design, optimization and control for PDE systems over a large parameter range. As outlined in the introduction, Burgers' equation is a 2nd order nonlinear PDE model that approximates the dynamics of the Navier-Stokes system. Based on the successful previous results [46–48, 50] we used the Group Finite Element (GFE) method which reduces the computational complexity of the non-linear term while maintaining accuracy. Thus, whenever applicable, we recommend the GFE over the standard FEM as a simplifying computational tool for high-fidelity simulations.

We have demonstrated in §3.2 that using as few 5 basis functions we can generate a reduced order model with less than 6% relative error while saving more than 80% of the computation time. The allows for real time computation of large systems through the use of Reduced Order Models, and has applications in Control of large dynamical systems. We also demonstrated in §3.4.1, and §3.4.2 that a POD basis generated for a single parameter may not be useful as a basis for a range of parameters. However, through the use of the POD basis sensitivities, we showed that the range in which a given POD basis is applicable can be expanded. It is worth noting that the ability of the Extrapolated or Expanded POD methods to consistently improve the accuracy over the standard POD method for the parameter range is dependent on the nature of the problem.

As for computational effort, in one instance it required approximately 45% more time to solve the problem in §4.1.1.2 using 4 parameter test points and generate the Global POD basis as solving the coupled problem from §4.4.2 and generating the POD basis and corresponding sensitivity vectors. This identifies one major flaw for Global POD, there are no criteria that determine what that representative parameter sample should be, either in size or specific choice of parameters. Therefore the specific parameter test sets may be required to be large to capture the underlying dynamics, and the cost to compute a global basis may be prohibitive. Thus, if the desire is to generate a basis that is a applicable over a wide range of parameters either an Extrapolated or an Expanded POD methods may be preferred. In §4.4.1, for every choice of r the extrapolated basis provided a better basis over the expanded basis. In §4.4.2, only for $r = 16$ was the expanded basis able to provide a better basis over the extrapolated basis. This suggests there may exist some $\tilde{r}$ that depends on the problem such that if $r > \tilde{r}$ then the expanded basis is better (over the parameter range) than the extrapolated basis. The particular choice of basis for a ROM must be balanced between the nature of the problem, potential parameter, and total total number or parameter samples required to generate a Global POD basis of sufficient accuracy.

5.3 Open Problems

Although the results of this thesis are purely numerical, we see potential for the use of these results. Moreover, the results give rise to several interesting questions.

- Can we formulate and implement higher order sensitivities for the POD basis vectors in (4.42)?
- Is there a single parameter that can provide an optimal basis standard POD basis for a parameter range? (How to pick the best baseline parameter?)
- What is the best way to pick a parameter test set for a global basis?
- Is there an a priori way to determine the optimal ROM dimension size? (efficiency analysis?)

Bibliography

[1] AC Antoulas, DC Sorensen, and S. Gugercin. A survey of model reduction methods for large-scale systems. 2006.

[2] P. Benner et al. *Numerical linear algebra for model reduction in control and simulation.* Techn. Univ., Fak. f. Math., 2005.

[3] R.W. Freund. Model reduction methods based on krylov subspaces. *Acta Numerica*, 12(1):267–319, 2003.

[4] Y. Chen. *Model order reduction for nonlinear systems.* PhD thesis, Citeseer, 1999.

[5] S. Gugercin and A.C. Antoulas. Model reduction of large-scale systems by least squares. *Linear algebra and its applications*, 415(2-3):290–321, 2006.

[6] Y.J. Yang and K.Y. Shen. Nonlinear heat-transfer macromodeling for mems thermal devices. *Journal of Micromechanics and Microengineering*, 15:408, 2005.

[7] J.K. Chu, L. Guan, D.Z. Qi, and X.C. Hao. Rapid nonlinear analysis for electrothermal microgripper using reduced order model based on krylov subspace. *International Journal of Nonlinear Sciences and Numerical Simulation*, 9(4):333–338, 2008.

[8] B. Moore. Principal component analysis in linear systems: Controllability, observability, and model reduction. *Automatic Control, IEEE Transactions on*, 26(1):17–32, 1981.

[9] D. Mustafa and K. Glover. Controller reduction by h?-balanced truncation. *Automatic Control, IEEE Transactions on*, 36(6):668–682, 1991.

[10] J.M.A. Scherpen. Balancing for nonlinear systems. *Systems & Control Letters*, 21(2):143–153, 1993.

[11] S. Lall, J.E. Marsden, and S. Glavaški. A subspace approach to balanced truncation for model reduction of nonlinear control systems. *International Journal of Robust and Nonlinear Control*, 12(6):519–535, 2002.

[12] M. Farhood and G.E. Dullerud. Model reduction of nonstationary lpv systems. *Automatic Control, IEEE Transactions on*, 52(2):181–196, 2007.

[13] T. Bui-Thanh, K. Willcox, O. Ghattas, and B. van Bloemen Waanders. Goal-oriented, model-constrained optimization for reduction of large-scale systems. *Journal of Computational Physics*, 224(2):880–896, 2007.

[14] E. Arian. Trust-region proper orthogonal decomposition for flow control. Technical report, DTIC Document, 2000.

[15] H.V. Ly and H.T. Tran. Modeling and control of physical processes using proper orthogonal decomposition. *Mathematical and computer modelling*, 33(1):223–236, 2001.

[16] SS 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.

[17] J.A. Atwell, J.T. Borggaard, and B.B. King. Reduced order controllers for burgers' equation with a nonlinear observer. *Applied Mathematics And Computer Science*, 11(6):1311–1330, 2001.

[18] J.A. Atwell. *Proper orthogonal decomposition for reduced order control of partial differential equations*. PhD thesis, Citeseer, 2000.

[19] K. Kunisch and S. Volkwein. Galerkin proper orthogonal decomposition methods for parabolic problems. *Numerische Mathematik*, 90(1):117–148, 2001.

[20] G. Berkooz, P. Holmes, and J.L. Lumley. The proper orthogonal decomposition in the analysis of turbulent flows. *Annual review of fluid mechanics*, 25(1):539–575, 1993.

[21] S. Volkwein. Model reduction using proper orthogonal decomposition. *Lecture Notes, Institute of Mathematics and Scientific Computing, University of Graz. see http://www. uni-graz. at/imawww/volkwein/POD. pdf.*

[22] K. Kunisch and S. Volkwein. Control of the Burgers equation by a reduced-order approach using proper orthogonal decomposition. *Journal of Optimization Theory and Applications*, 102(2):345–371, 1999.

[23] J. Kim. Control of turbulent boundary layers. *Physics of fluids*, 15:1093, 2003.

[24] B.T. DICKINSON and J.R. SINGLER. Nonlinear model reduction using group proper orthogonal decomposition. *Int. J. Numer. Anal. Mod*, 7(2):356–372, 2010.

[25] M. Fahl. *Trust-region methods for flow control based on reduced order modelling*. PhD thesis, Universitätsbibliothek, 2001.

[26] M.Ö. Efe and H.İ. Özbay. Low dimensional modelling and dirichlet boundary controller design for burgers equation. *International Journal of Control*, 77(10):895–906, 2004.

[27] N.C. Nguyen, G. Rozza, and A.T. Patera. Reduced basis approximation and a posteriori error estimation for the time-dependent viscous burgers equation. *Calcolo*, 46(3):157–185, 2009.

[28] S.M. Djouadi, R.C. Camphouse, and J.H. Myatt. Optimal order reduction for the the two-dimensional burgers equation. In *Decision and Control, 2007 46th IEEE Conference on*, pages 3507–3512. IEEE, 2007.

[29] Z. Luo, Y. Zhou, and X. Yang. A reduced finite element formulation based on proper orthogonal decomposition for burgers equation. *Applied numerical mathematics*, 59(8):1933–1946, 2009.

[30] M.D. Gunzburger, J.S. Peterson, and J.N. Shadid. Reduced-order modeling of time-dependent pdes with multiple parameters in the boundary data. *Computer methods in applied mechanics and engineering*, 196(4-6):1030–1047, 2007.

[31] M. Hinze and K. Kunisch. Three control methods for time-dependent fluid flow. *Flow, Turbulence and Combustion*, 65(3):273–298, 2000.

[32] Y. Cao, J. Zhu, Z. Luo, and IM Navon. Reduced-order modeling of the upper tropical pacific ocean model using proper orthogonal decomposition. *Computers and Mathematics with Applications*, 52(8-9):1373–1386, 2006.

[33] EA Gillies. Low-dimensional control of the circular cylinder wake. *Journal of Fluid Mechanics*, 371(1):157–178, 1998.

[34] K. Willcox and J. Peraire. Balanced model reduction via the proper orthogonal decomposition. *AIAA journal*, 40(11):2323–2330, 2002.

[35] C.W. Rowley. Model reduction for fluids, using balanced proper orthogonal decomposition. *International Journal of Bifurcation Chaos in Applied Sciences and Engineering*, 15(3):997–1014, 2005.

[36] M. Ilak and C.W. Rowley. Modeling of transitional channel flow using balanced proper orthogonal decomposition. *Physics of Fluids*, 20:034103, 2008.

[37] T. Bui-Thanh and K. Willcox. Model reduction for large-scale cfd applications using the balanced proper orthogonal decomposition. In *17 th AIAA Computational Flow Dynamics Conference*, pages 1–15, 2005.

[38] J.R. Singler and B.A. Batten. A proper orthogonal decomposition approach to approximate balanced truncation of infinite dimensional linear systems. *International Journal of Computer Mathematics*, 86(2):355–371, 2009.

[39] M. Ilak and C.W. Rowley. Feedback control of transitional channel flow using balanced proper orthogonal decomposition.

[40] J.R. Singler and B.A. Batten. Balanced proper orthogonal decomposition for model reduction of infinite dimensional linear systems. In *Proceedings of the Int. Conf. on Computational and Mathematical Methods in Science and Engineering, CMMSE*, 2007.

[41] C.A.J. Fletcher. Burgers equation: a model for all reasons. *Numerical Solutions of Partial Differential Equations (Noye, J., ed.), North-Holland, Amsterdam*, pages 139–225, 1981.

[42] Alice Gorguis. A comparison between cole-hopf transformation and the decomposition method for solving burgers' equations. *Applied Mathematics and Computation*, 173(1):126 – 136, 2006.

[43] D. Kaya. An explicit solution of coupled viscous burgers' equation by the decomposition method. *International Journal of Mathematics and Mathematical Sciences*, 27(11):675–680, 2001.

[44] S. Abbasbandy and MT Darvishi. A numerical solution of burgers' equation by modified adomian method. *Applied mathematics and computation*, 163(3):1265–1272, 2005.

[45] E. Hopf. The partial differential equation ut+ uux= μxx. *Communications on Pure and Applied Mathematics*, 3(3):201–230, 1950.

[46] S.M. Pugh. Finite element approximations of burgers' equation. 1995.

[47] L.C. Smith III. *Finite Element Approximations of BurgersEquation With RobinS Boundary Conditions*. PhD thesis, Virginia Polytechnic Institute and State University, 1997.

[48] V.Q. Nguyen. *A Numerical Study of Burgers Equation With Robin Boundary Conditions*. PhD thesis, Virginia Polytechnic Institute and State University, 2001.

[49] J.R. Singler. *Sensitivity analysis of partial differential equations with applications to fluid flow*. PhD thesis, Citeseer, 2005.

[50] B. Krämer. *Model Reduction of the Coupled Burgers Equation in Conservation Form*. PhD thesis, Virginia Polytechnic Institute and State University, 2011.

[51] C.A.J. Fletcher et al. *Computational galerkin methods*, volume 260. Springer-Verlag New York, 1984.

[52] P.J. Roache. Code verification by the method of manufactured solutions. *Journal of Fluids Engineering*, 124:4, 2002.

[53] W.L. Oberkampf and C.J. Roy. *Verification and validation in scientific computing*. Cambridge Univ Pr, 2010.

[54] A. Hay, J.T. Borggaard, and D. Pelletier. Local improvements to reduced-order models using sensitivity analysis of the proper orthogonal decomposition. *Journal of Fluid Mechanics*, 629(-1):41–72, 2009.

[55] J. Borggaard and J. Burns. A pde sensitivity equation method for optimal aerodynamic design* 1. *Journal of Computational Physics*, 136(2):366–384, 1997.

Appendix A

Matlab® Source Code

In this section all MATLAB® source code is provided for the Dirichlet boundary conditions problem.

A.1 Main File

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This script runs my research.
%
% Goals:
%
% 1) For a given input parameter solve Burgers' using FEM
% 2) Solve Continuous Sensitivities at given parameter
% 3) Develop ROM using POD basis (for fixed parameter)
% 4) Analyze POD basis ROM performance against other parameters
% 5) Use Sensitivities to Extrapolate POD basis and assess performance
% 6) Use Sensitivities to Expanded POD basis and assess performance
%
% Assumptions:
%
% 1) Dirichlet boundary y(t,0)=0=y (t,1).
%
% Process:
%
% 1) Inputs
% 2) Generate Constant Matrices
% 3) Solve Burgers' and Sensitivity problems
%   3.1) Burgers' ODE step
%   3.2) Solve at time step
% 4) Generate ROM for baseline parameter
%   4.1) Find POD basis
%   4.2) Solve ROM ODE problem
% 5) Test Basline POD ROM at various parameters
%   5.1) Generate Full model at each test parameter
%   5.2) Use Baseline POD basis to build ROM model for each test parameter
```

```
30 % 6) Develop ROM improvments
31 %   6.1) Generate POD sensitivies
32 %   6.2) Extrapolate POD basis
33 %   6.3) Expand POD basis
34 %   6.4) Compare Error
35 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
36
37 % close all
38 clc
39 clear
40
41 % profile on
42
43 % MSGID = 'MATLAB:nearlySingularMatrix'
44 % warning('off', MSGID)
45
46 warning off
47
48 % start clock
49 tic
50
51 % Step 1 - Inputs
52 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
53
54 global n h m q k f fq g M C B y s Psi Br Cr Mr Gr r ROM_Basis GPsi ExtPsi ...
       ExpPsi x
55
56 % Total Spatial elements
57 n=32;
58
59 % Final  time
60 final_t = 2;
61
62 % Total number of Time steps saved (equally spaced)
63 m = 200;
64
65 % Reynolds Number
66 Re = 100;
67
68 qbaseline = 1/Re;
69 q=qbaseline;
70
71 % Function files
72  % RHS functions
73  f  = 'f_func';
74  fq = 'f_q_func';
75
76  % Initial functions
77  g  = 'g_func';
78  gq = 'g_q_func';
79
80 % Solver and options
81 solver = 'ode45';
82 options = odeset('RelTol',1e-6,'AbsTol',1e-10);
83
```

```
84 % POD basis dim
85 E = .99;        % Desired Energy captured
86 r = 5;          % Desired number of basis vectors
87
88 % Test points
89   % Parameter test points
90   REtest = [10:10:90 95 100 105 110 120:20:400];
91   qtest = 1./REtest;
92
93   % Global POD test points
94   Gqtest = [1/50 1/100 1/200 1/400];
95
96   % ROM dimension test points
97   rtest = [2 4 5 ];
98   %rtest = [2 4 5 8 10 16];
99
100 %%%%%%%%%%
101 % SWITCH BLOCKS
102 Only_Burg = 1;
103 Burg_and_Sens = 1;
104
105 FEM_Plot = 1;
106 MMS_Test = 1;
107 MMS_Plot = 1;
108
109 Baseline_ROM = 1;
110 POD_Plot = 1;
111 POD_FEM_Test = 1;
112 POD_FEM_Test_Plot = 1;
113 POD_MMS_Test = 1;
114 POD_MMS_Test_Plot = 1;
115 Baseline_POD_q_Test = 1;
116
117 Global_POD_Basis = 1;
118
119 All_POD_ROM_test = 1;       %Requires Global_POD_Basis = 1
120 Parameter_Dep_Plots = 1;
121
122
123 % End Step 1
124 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
125
126 % Make adjusements from elements to nodes:
127 n=n-1;          % Internal nodes
128
129
130 % Compute step sizes
131 h=1/(n+1);                % space step size
132 x = 0:h:1;                % compute x
133
134 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
135 % Step 2 - Generate Constant Matrices
136
137 % Generate Mass matrix
138     M=zeros(n,n);
```

```
139     for i=1:n
140         M(i,i)=2/3;
141         if i>1
142             M(i-1,i)=1/6;
143         end
144         if i<n
145             M(i+1,i)=1/6;
146         end
147     end
148     M=M*h;
149
150 % Generate B (Convective Term)
151     B=zeros(n,n);
152     for i=1:n
153
154         if i>1
155             B(i-1,i)=1/2;
156         end
157         if i<n
158             B(i+1,i)=-1/2;
159         end
160      end
161
162 % Generate Stiffness Matrix
163     C=zeros(n,n);
164     for i=1:n
165         C(i,i)=2;
166
167         if i>1
168             C(i-1,i)=-1;
169         end
170         if i<n
171             C(i+1,i)=-1;
172         end
173     end
174     C=C*h^(-1);
175
176 % Initial Conditions (Gauss Quadrature)
177   % Generate Initial y
178     G = GQ_three(g);
179     init_y = M\G;
180
181   % Generate Initial s
182     Gq = GQ_three(gq);
183     init_s(:,1) = M\Gq;
184
185 % End Step 2
186 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
187
188
189
190 %%%%%%%%%%%%%%%%%%%%%%%%%%%%
191 % Solve ONLY Burgers' Equation
192 %%%%%%%%%%%%%%%%%%%%%%%%%%%%
193 switch Only_Burg
```

```
194     case 1
195
196         t=0:(final_t/m):final_t;
197
198         timeODEinit = toc;
199         [t,y] = feval(solver,'FEM_Burgers_func',t,init_y);
200         timeODEfinal = toc;
201
202         sprintf ('ODE solver time')
203         Ot=timeODEfinal-timeODEinit
204
205         y=y';
206
207         % Add Dirichlet Boundary conditions
208         tempy=[zeros(1,length(t));y;zeros(1,length(t))];
209
210         %%%%%%%%%%%%%%%%%%%%%%%%
211         % FEM Plot
212         %%%%%%%%%%%%%%%%%%%%%%%%
213         switch FEM_Plot
214             case 1
215                 figure('name',['FEM for q= 1/',num2str(1/q),', ',num2str(n+1),' ...
                        elements']);
216                 mesh(x,t,tempy')
217                 % axis([0 1 0 final_t 0 1.1])
218                 xlabel('x')
219                 ylabel('t')
220                 zlabel('y (t,x)')
221         end
222
223
224         %%%%%%%%%%%%%%%%%%%%%%%%
225         % MMS tests
226         %%%%%%%%%%%%%%%%%%%%%%%%
227         switch MMS_Test
228             case 1
229                 sprintf ('FEM Relative error')
230                 ab=0;
231                 e2n=0;
232                 for i=1:length(t)               % loop across all time steps
233                     [abtemp, e2ntemp]=Error_GQ_three(t(i),tempy(:,i));
234                     ab = ab + abtemp;
235                     e2n = e2n + e2ntemp;
236                 end
237                 ab = sqrt(ab*(t(2)-t(1)));      % total L^2 norm ([0,1] x [0,T])
238                 e2n = sqrt(e2n*(t(2)-t(1)));    % total L^2 norm ([0,1] x [0,T])
239                 relerr = ab/e2n                 % relative error
240
241                 for i=1:length(t)
242                     Exact_Y(:,i)=exact_func(t(i),x');
243                 end
244
245                 FEMerr = Exact_Y-tempy;
246
247                 %%%%%%%%%%%%%%%%%%%%%%%%
```

```
248                 % MMS Plots                     %Requires MMS_Test=1
249                 %%%%%%%%%%%%%%%%%%%%%%%%
250                 switch MMS_Plot
251                     case 1
252                         % Time points for cross section
253                         t1= round(.25*m/final_t)+1;
254                         t2= round(.25*m/final_t)+1;
255                         t3= round(.50*m/final_t)+1;
256                         t4= round(.75*m/final_t)+1;
257
258                         ttest=[t1 t2 t3 t4];
259                         for i=1:4
260                         Ycs(:,i)=exact_func(t(ttest(i)),x');
261                         end
262
263                         fh=figure('name',['Cross section of FEM and exact ...
                                solution for q= 1/',num2str(Re),', ',num2str(n+1),' ...
                                elements']);
264                         RECT1 = [100, 100, 1000, 700];
265                         set(fh,'position', RECT1);
266
267                         RECT2 = [.05, .05, .90, .90];
268                         axes('position', RECT2);
269                         plot(x,tempy(:,ttest(1)),'b.-',x,Ycs(:,1),'b',x, ...
                                tempy(:,ttest(2)),'g.-',x,Ycs(:,2),'g',x, ...
                                tempy(:,ttest(3)),'r.-',x,Ycs(:,3),'r',x, ...
                                tempy(:,ttest(4)),'m.-',x,Ycs(:,4),'m')
270                         xlabel('x')
271                         legend('t=0 numerical','t=0 exact','t=.25 ...
                                numerical','t=.25 exact','t=.5 numerical','t=.5 ...
                                exact','t=.75 numerical','t=.75 ...
                                exact','Location','NorthEast')
272                         axis([0 1 0 1.1])
273
274                         figure('name','Exact solution')
275                         mesh(x,t,Exact_Y')
276                         % axis([0 1 0 final_t 0 1.1])
277                         xlabel('x')
278                         ylabel('t')
279                         zlabel('y (t,x)')
280
281                         figure('name',['FEM error for q= 1/',num2str(Re),', ...
                                ',num2str(n+1),' elements'] )
282                         mesh(x,t,FEMerr')
283                         xlabel('x')
284                         ylabel('t')
285                         zlabel('y (t,x)')
286                 end
287
288         end
289 end
290
291
292
293 %%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
294 % Solve Coupled Burgers' and Sensitivity Problem
295 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
296 switch Burg_and_Sens
297     case 1
298         t=0:(final_t/m):final_t;
299         V = [init_y ; init_s];
300 
301         timeODEinit = toc;
302         [t,V] = feval(solver,'FEM_Burgers_and_Sens_func',t,V,options);
303         timeODEfinal = toc;
304 
305         sprintf ('ODE solver time')
306         Ot=timeODEfinal-timeODEinit
307 
308 
309         V=V';
310         y = V(1:n,:);
311         s = V(n+1:2*n,:);
312 
313         % Add Dirichlet Boundary conditions
314         tempy=[zeros(1,length(t));y;zeros(1,length(t))];
315         temps=[zeros(1,length(t));s;zeros(1,length(t))];
316 
317         %%%%%%%%%%%%%%%%%%%%%%%%%
318         % FEM Plot
319         %%%%%%%%%%%%%%%%%%%%%%%%%
320         switch FEM_Plot
321             case 1
322 
323                 figure('name',['FEM for q= 1/',num2str(1/q),', ',num2str(n+1),' ...
                        elements'])
324                 mesh(x,t,tempy')
325                 xlabel('x')
326                 ylabel('t')
327                 zlabel('y (t,x)')
328 
329                 figure('name',['FEM Sensitivity for q= 1/',num2str(1/q),', n = ...
                        ',num2str(n+1),', m = ',num2str(m) ])
330                 mesh(x,t,temps')
331                 xlabel('x')
332                 ylabel('t')
333                 zlabel('s (t,x)')
334         end
335 
336         %%%%%%%%%%%%%%%%%%%%%%%%%
337         % MMS tests
338         %%%%%%%%%%%%%%%%%%%%%%%%%
339         switch MMS_Test
340             case 1
341 
342                 sprintf ('FEM Relative error for y')
343                 ab=0;
344                 e2n=0;
345                 for i=1:length(t)               % loop across all time steps
346                     [abtemp, e2ntemp]=Error_GQ_three(t(i),tempy(:,i));
```

```
347                     ab = ab + abtemp;
348                     e2n = e2n + e2ntemp;
349                 end
350                 ab = sqrt(ab*(t(2)-t(1)));      % total L^2 norm ([0,1] x [0,T])
351                 e2n = sqrt(e2n*(t(2)-t(1)));    % total L^2 norm ([0,1] x [0,T])
352                 frelerr = ab/e2n                % relative error
353
354                 sprintf ('FEM Relative error for s')
355                 ab=0;
356                 e2n=0;
357                 for i=1:length(t)               % loop across all time steps
358                     [abtemp, e2ntemp]=SError_GQ_three(t(i),temps(:,i));
359                     ab = ab + abtemp;
360                     e2n = e2n + e2ntemp;
361                 end
362                 ab = sqrt(ab*(t(2)-t(1)));      % total L^2 norm ([0,1] x [0,T])
363                 e2n = sqrt(e2n*(t(2)-t(1)));    % total L^2 norm ([0,1] x [0,T])
364                 srelerr = ab/e2n                % relative error
365
366
367                 for i=1:length(t)
368                     Y(:,i)=exact_func(t(i),x');
369                     S(:,i)=exact_sens_func(t(i),x');
370                 end
371
372                 FEMerr = Y-tempy;
373                 SFEMerr = S-temps;
374
375                 %%%%%%%%%%%%%%%%%%%%%%%%
376                 % MMS Plots                     %Requires MMS_Test=1
377                 %%%%%%%%%%%%%%%%%%%%%%%%
378                 switch MMS_Plot
379                     case 1
380
381                         % Time points for cross section
382                         t1= round(.25*length(t)/final_t)+1;
383                         t2= round(.25*length(t)/final_t)+1;
384                         t3= round(.50*length(t)/final_t)+1;
385                         t4= round(.75*length(t)/final_t)+1;
386
387                         ttest=[t1 t2 t3 t4];
388                         for i=1:4
389                             Ycs(:,i)=exact_func(t(ttest(i)),x');
390                             Scs(:,i)=exact_sens_func(t(ttest(i)),x');
391                         end
392
393                         fh=figure('name',['FEM and exact solution for q= ...
                                1/',num2str(Re),', ',num2str(n+1),' elements']);
394                         RECT1 = [100, 100, 1000, 700];
395                         set(fh,'position', RECT1);
396
397                         RECT2 = [.05, .05, .90, .90];
398                         axes('position', RECT2);
399                         plot(x,tempy(:,ttest(1)),x,Ycs(:,1),x,tempy(:,ttest(2)), ...
                                x,Ycs(:,2),x,tempy(:,ttest(3)),x,Ycs(:,3),x, ...
```

```
                            tempy(:,ttest(4)),x,Ycs(:,4))
400                         xlabel('x')
401                         legend('t=0 numerical','t=0 exact','t=.25 ...
                                numerical','t=.25 exact','t=.5 numerical','t=.5 ...
                                exact','t=.75 numerical','t=.75 ...
                                exact','Location','NorthWest')
402
403                         fh=figure('name',['FEM sens and exact solution for q= ...
                                1/',num2str(Re),', ',num2str(n+1),' elements']);
404                         RECT1 = [100, 100, 1000, 700];
405                         set(fh,'position', RECT1);
406
407                         RECT2 = [.05, .05, .90, .90];
408                         axes('position', RECT2);
409                         plot(x,temps(:,ttest(1)),x,Scs(:,1),x,temps(:,ttest(2)) ...
                                ,x,Scs(:,2),x,temps(:,ttest(3)),x,Scs(:,3),x, ...
                                temps(:,ttest(4)),x,Scs(:,4))
410                         xlabel('x')
411                         legend('t=0 numerical','t=0 exact','t=.25 ...
                                numerical','t=.25 exact','t=.5 numerical','t=.5 ...
                                exact','t=.75 numerical','t=.75 ...
                                exact','Location','NorthWest')
412
413                         figure('name','Exact solution')
414                         mesh(x,t,Y')
415                         % axis([0 1 0 final_t 0 1.1])
416                         xlabel('x')
417                         ylabel('t')
418                         zlabel('y(t,x)')
419
420                         figure('name',['FEM error for q= 1/',num2str(Re),', ...
                                ',num2str(n+1),' elements'] )
421                         mesh(x,t,FEMerr')
422                         xlabel('x')
423                         ylabel('t')
424                         zlabel('y(t,x)')
425
426                         figure('name',['Sens FEM error for q= ...
                                1/',num2str(Re),', ',num2str(n+1),' elements'] )
427                         mesh(x,t,SFEMerr')
428                         xlabel('x')
429                         ylabel('t')
430                         zlabel('y(t,x)')
431                 end
432         end
433 end
434 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
435 % End Solve Coupled Burgers' and Sensitivity Problem
436
437
438
439 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
440 % Generate ROM for Baseline Parameter
441 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
442 switch Baseline_ROM
```

```
443     case 1
444
445         % Compute POD basis
446         [Psi, e, re] = POD_Basis(y,r,E);        % Input y is only internal elements
447
448         % Prep POD matrices
449
450             % Cr
451             Cr = q*Psi'*C*Psi;
452
453         % Solve for initial coefficients
454         ar(:,1) = y(:,1)'*M*Psi;                % Weighted Inner Product
455
456         % Local ROM basis
457         ROM_Basis = Psi;
458
459         sprintf ('Solve ROM');
460         timePODODEinit = toc;
461         [t,ar] = feval(solver,'FEM_Burgers_ROM_func',t,ar(:,1),options);
462         timePODODEfinal = toc;
463
464         sprintf ('POD ODE solver time')
465         Rt=timePODODEfinal-timePODODEinit
466
467         ar=ar';
468
469         % Convert to Full Dimension Solution
470         yROM= Psi*ar;
471
472         %%%%%%%%%%%%%%%%%%%%%%%%%
473         % POD Plot
474         %%%%%%%%%%%%%%%%%%%%%%%%%
475         switch POD_Plot
476             case 1
477
478                 tempyROM=[zeros(1,length(t));yROM;zeros(1,length(t))];
479
480                 figure('name',['POD ROM for q= 1/',num2str(Re),', n = ...
                        ',num2str(n+1),', m = ',num2str(m),' using  ',num2str(r),' ...
                        basis vectors'])
481                 mesh(x,t,tempyROM')
482                 % axis([0 1 0 final_t 0 1.1])
483                 xlabel('x')
484                 ylabel('t')
485                 zlabel('ROM y(t,x)')
486         end
487
488         %%%%%%%%%%%%%%%%%%%%%%%%%
489         % POD FEM Test
490         %%%%%%%%%%%%%%%%%%%%%%%%%
491         switch POD_FEM_Test
492             case 1
493
494                 % for comparison against FEM solution
495                 sprintf ('POD Relative error from FEM')
```

```
496                 yROMerr = y-yROM;
497
498                 tempPODerr=0;
499                 tempFEMtrue=0;
500
501                 for ii=1:length(t)
502                 tempPODerr = tempPODerr + (yROMerr (:,ii)'*M*yROMerr(:,ii));
503                 tempFEMtrue = tempFEMtrue + (y (:,ii)'*M*y(:,ii));
504                 end
505
506                 tempPODerr=sqrt(tempPODerr*(t(2)-t(1)));    % total L^2 norm ...
                        ([0,1] x [0,T])
507                 tempFEMtrue=sqrt(tempFEMtrue*(t(2)-t(1)));        % total L^2 ...
                        norm ([0,1] x [0,T])
508                 relerr = tempPODerr/tempFEMtrue
509
510                 sprintf ('Time savings over FEM')
511                 1-(Rt)/Ot
512
513                 %%%%%%%%%%%%%%%%%%%%%%%%
514                 % POD FEM Test Plot
515                 %%%%%%%%%%%%%%%%%%%%%%%%
516                 switch POD_FEM_Test_Plot
517                     case 1
518
519                         tempyROMerr=[zeros(1,length(t));yROMerr;zeros(1,length(t))];
520                         figure('name',['POD error for q= 1/',num2str(Re),', ...
                                ',num2str(r),' basis vectors'])
521                         mesh(x,t,tempyROMerr')
522                         xlabel('x')
523                         ylabel('t')
524                         zlabel('y (t,x)')
525                 end
526         end
527
528         %%%%%%%%%%%%%%%%%%%%%%%%
529         % POD MMS Test
530         %%%%%%%%%%%%%%%%%%%%%%%%
531         switch POD_MMS_Test
532             case 1
533
534                 % for comparison against exact solution
535                 sprintf ('POD Relative error from exact')
536                 ab=0;
537                 e2n=0;
538                 for i=1:length(t)               % loop across all time steps
539                      [abtemp, e2ntemp]=Error_GQ_three(t(i),tempy(:,i));
540                     ab = ab + abtemp;
541                     e2n = e2n + e2ntemp;
542                 end
543                 ab = sqrt(ab*(t(2)-t(1)));      % total L^2 norm ([0,1] x [0,T])
544                 e2n = sqrt(e2n*(t(2)-t(1)));    % total L^2 norm ([0,1] x [0,T])
545                 relerr = ab/e2n                 % relative error
546
547                 for i=1:length(t)
```

```
548                 Y(:,i)=exact_func(t(i),x');
549                 end
550                 PODerr = Y-tempyROM;
551 
552                 %%%%%%%%%%%%%%%%%%%%%%%%
553                 % POD MMS Test Plot
554                 %%%%%%%%%%%%%%%%%%%%%%%%
555                 switch POD_MMS_Test_Plot
556                     case 1
557 
558                         % prepare exact plot
559                         t1=1;
560                         t2= round(.25*length(t)/final_t);
561                         t3= round(.50*length(t)/final_t);
562                         t4= round(.75*length(t)/final_t);
563 
564                         ttest=[t1 t2 t3 t4];
565                         for i=1:4
566                         Ycs(:,i)=exact_func(t(ttest(i)),x');
567                         end
568 
569                         fh=figure('name',['Cross section of POD and exact ...
                                solution for q= 1/',num2str(Re),', ',num2str(n+1),' ...
                                elements']);
570                         RECT1 = [100, 100, 1000, 700];
571                         set(fh,'position', RECT1);
572 
573                         RECT2 = [.05, .05, .90, .90];
574                         axes('position', RECT2);
575                         plot(x,tempyROM(:,ttest(1)),'b.-',x,Ycs(:,1),'b',x, ...
                                tempyROM(:,ttest(2)),'g.-',x,Ycs(:,2),'g',x, ...
                                tempyROM(:,ttest(3)),'r.-',x, ...
                                Ycs(:,3),'r',x,tempyROM(:,ttest(4)),'m.-',x,Ycs(:,4),'m')
576                         xlabel('x')
577                         legend('t=0 numerical','t=0 exact','t=.25 ...
                                numerical','t=.25 exact','t=.5 numerical','t=.5 ...
                                exact','t=.75 numerical','t=.75 ...
                                exact','Location','NorthEast')
578 
579                         figure('name',['POD error for q= 1/',num2str(Re),', ...
                                ',num2str(r),' basis vectors'])
580                         mesh(x,t,PODerr')
581                         xlabel('x')
582                         ylabel('t')
583                         zlabel('y (t,x)')
584                 end
585         end
586 
587         %%%%%%%%%%%%%%%%%%%%%%%%
588         % Test Baseline POD Across Parameter Range
589         %%%%%%%%%%%%%%%%%%%%%%%%
590         switch Baseline_POD_q_Test
591             case 1
592 
593                 % Test Baseline POD
```

```
594                 sprintf ('Test ROM')
595                 ROMTesttoc1= toc;
596
597                 % Loop - Pick Re to test
598                 z = waitbar(0,'Please wait...');            %Needed for ...
                        progress bar
599
600                 % Same loop is used to evaluate Full and Reduced Models at each ...
                        time step
601                 for i=1:size(qtest,2)
602
603                     message2=sprintf ('Total Parameter Test Loop % g of % i', ...
                            i, size(qtest,2));
604                     waitbar(i/size(qtest,2),z,message2);
605
606                     q = qtest(i);
607
608                     % Set up ROM
609                     clear ytest artest
610
611                         % Generate Mr
612                         Mr = Psi'*M*Psi;
613
614                         % Cr POD matrix
615                         Cr = q*Psi'*C*Psi;
616
617                         % Initial Conditions
618
619                             % Full order
620                             ytest(:,1)=y(:,1);
621
622                             % ROM
623                             artest(:,1) = y (:,1)'*M*Psi;
624
625                     % evaluate models
626
627                         % Full order model
628                         [t,ytest] = ode45('FEM_Burgers_func',t,ytest(:,1));
629                         ytest=ytest';
630
631                         % Baseline POD ROM time step
632                         [t,artest] = ode45('FEM_Burgers_ROM_func', t, artest(:,1));
633                         artest=artest';
634
635                     % Convert to Full Dimension Solution
636                     Psiy = Psi*artest;
637
638                     % Compute ROM errors
639                         yROMerr = ytest-Psiy;
640
641                         tempPODerr=0;
642                         tempFEMtrue=0;
643
644                         for ii=1:length(t)
```

```
645                         tempPODerr = tempPODerr + (yROMerr ...
                                (:,ii)'*M*yROMerr(:,ii));
646                         tempFEMtrue = tempFEMtrue + (ytest (:,ii)'*M*ytest(:,ii));
647                         end
648
649                         tempPODerr=sqrt(tempPODerr*(t(2)-t(1)));    % total L^2 ...
                                norm ([0,1] x [0,T])
650                         tempFEMtrue=sqrt(tempFEMtrue*(t(2)-t(1)));        % ...
                                total L^2 norm ([0,1] x [0,T])
651
652                         ROMerr(i)=tempPODerr/tempFEMtrue;
653
654                 end % End Parameter step Loop
655                 delete(z)
656
657                 savefile = ['POD',num2str(r),'relerr.mat'];
658                 save(savefile, 'ROMerr')
659
660                 % Plot Relative Error across Parameter Range
661                 figure('name',['Relative Error with ',num2str(r),' basis ...
                        vectors and n = ',num2str(n+1),' baseline 1/q = ...
                        ',num2str(1/qbaseline)])
662                 semilogy(REtest,ROMerr)
663                 xlabel('1/q')
664                 ylabel('relative error')
665
666                 ROMTesttoc2 = toc;
667                 ROMtesttime = ROMTesttoc2 -ROMTesttoc1
668         end
669 end
670
671
672
673 %%%%%%%%%%%%%%%%%%%%%%%%%%%%
674 % Generate Global POD Basis
675 %%%%%%%%%%%%%%%%%%%%%%%%%%%%
676 switch Global_POD_Basis
677     case 1
678
679         sprintf ('Global POD')
680         GlobY = zeros(n,length(t)*length(Gqtest));
681
682         z = waitbar(0,'Please wait...');
683
684         for i=1:size(Gqtest,2)
685
686             message=sprintf ('Global Parameter Test Loop % g of % i', i, ...
                    size(Gqtest,2));
687             waitbar(i/size(Gqtest,2),z,message);
688
689             q=Gqtest(i);
690             [t,ytest] = feval(solver,'FEM_Burgers_func',t,init_y,options);
691             ytest=ytest';
692
693             if i==1
```

```
694                 GlobY= ytest(:,:);
695             end
696             if i>1
697                 GlobY= [GlobY ytest(:,:)];
698             end
699         end
700 
701         delete(z)
702 end
703 %}
704 
705 
706 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
707 % Test all ROMs for Parameter Range
708 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
709 switch All_POD_ROM_test
710     case 1
711 
712         sprintf ('POD basis loop')
713 
714         % First Loop - POD basis dimension
715         z = waitbar(0,'Please wait...');
716         for iii=1:size(rtest,2)
717 
718         message1=sprintf ('POD basis loop % g of % g', iii, size(rtest,2));
719         waitbar(iii/size(rtest,2),z,message1)
720 
721         r=rtest(iii);
722         sprintf ('Dimension of POD basis % i', r)
723 
724         % Generate POD basis with (iii) basis vectors
725         [Psi, Psia, e, re]=POD_Basis_with_Sens(y,s,r,E);
726 
727         % Generate Global POD basis with (iii) basis vectors
728         [GPsi, e, re] = POD_Basis(GlobY,r,E);
729 
730             % Second Loop - Pick q to test
731             zz = waitbar(0,'Please wait...');
732 
733             ParamTesttoc1= toc;         % parameter test time
734 
735             % Same loop is used to evaluate Full and Reduced Models at each ...
                    time step
736             for i=1:length(qtest)
737 
738                 message2=sprintf ('Total Parameter Test Loop % g of % i', i, ...
                        size(qtest,2));
739                 waitbar(i/size(qtest,2),zz,message2);
740 
741                 q=qtest(i);
742                 deltaq = q-qbaseline;
743 
744                 clear ytest artest gartest extartest expartest
745 
746                 % Generate Extrapolated Basis
```

```
747                     ExtPsi = Psi + deltaq*Psia;
748
749                 % Generate Expanded Basis
750                     ExpPsi = [Psi Psia];
751
752                 % Set initial Conditions
753                   % full order
754                   ytest(:,1)= init_y;
755
756                   % ROM using baseline POD
757                   artest(:,1) = init_y'*M*Psi;
758
759                   %ROM using extrapolated basis
760                   extartest(:,1) = init_y'*M*ExtPsi;
761
762                   % ROM using expanded basis
763                   expartest(:,1) = ExpPsi \ init_y;
764
765                   % Global POD
766                   gartest(:,1) = init_y'*M*GPsi;
767
768                 % Compute Models
769                   % Full order model
770                   [t1,ytest] = ...
                          feval(solver,'FEM_Burgers_func',t,ytest(:,1),options);
771                   ytest=ytest';
772
773
774                   % Baseline POD ROM
775                     % Generate Mr
776                     Mr = Psi'*M*Psi;
777                     % Generate Cr
778                     Cr = q*Psi'*C*Psi;
779
780                     % Local ROM basis
781                     ROM_Basis = Psi;
782
783                   [t1,artest] = ...
                          feval(solver,'FEM_Burgers_ROM_func',t,artest(:,1),options);
784                   artest=artest';
785                   Psiy = Psi*artest;
786
787                   % Extrapolated POD ROM
788                     % Generate Mr
789                     Mr = ExtPsi'*M*ExtPsi;
790                     % Generate Cr
791                     Cr = q*ExtPsi'*C*ExtPsi;
792
793                     % Local ROM basis
794                     ROM_Basis = ExtPsi;
795
796                   [t1,extartest] = ...
                          feval(solver,'FEM_Burgers_ROM_func',t,extartest(:,1),options);
797                   extartest=extartest';
798                   ExtPsiy = ExtPsi*extartest;
```

```
799
800                     % Expanded POD ROM
801                       % Generate Mr
802                       Mr = ExpPsi'*M*ExpPsi;
803                       % Generate Cr
804                       Cr = q*ExpPsi'*C*ExpPsi;
805
806                       % Local ROM basis
807                       ROM_Basis = ExpPsi;
808
809                      [t1,expartest] = ...
                             feval(solver,'FEM_Burgers_ROM_func',t,expartest(:,1),options);
810                     expartest=expartest';
811                     ExpPsiy = ExpPsi*expartest;
812
813                     % Global POD ROM
814                       % Generate Mr
815                       Mr = GPsi'*M*GPsi;
816                       % Generate Cr
817                       Cr = q*GPsi'*C*GPsi;
818
819                       % Local ROM basis
820                       ROM_Basis = GPsi;
821
822
823                     [t1,gartest] = ...
                            feval(solver,'FEM_Burgers_ROM_func',t,gartest(:,1),options);
824                     gartest=gartest';
825                     GPsiy = GPsi*gartest;
826
827                   % Comupute ROM errors compared to against FEM solution
828                   Psierr = ytest-Psiy;
829                   GPsierr = ytest-GPsiy;
830                   ExtPsierr = ytest-ExtPsiy;
831                   ExpPsierr = ytest-ExpPsiy;
832
833                   tempPsierr=0;
834                   tempGPsierr=0;
835                   tempExtPsierr=0;
836                   tempExpPsierr=0;
837                   tempFEMtrue=0;
838
839                   for ii=1:length(t)
840                       tempPsierr = tempPsierr + (Psierr (:,ii)'*M*Psierr(:,ii));
841                       tempGPsierr = tempGPsierr + (GPsierr (:,ii)'*M*GPsierr(:,ii));
842                       tempExtPsierr = tempExtPsierr + (ExtPsierr ...
                                (:,ii)'*M*ExtPsierr(:,ii));
843                       tempExpPsierr = tempExpPsierr + (ExpPsierr ...
                                (:,ii)'*M*ExpPsierr(:,ii));
844
845                       tempFEMtrue = tempFEMtrue + (ytest (:,ii)'*M*ytest(:,ii));
846                   end
847
848                   tempPsierr=sqrt(tempPsierr*(t(2)-t(1)));    % total L^2 norm ...
                       ([0,1] x [0,T])
```

```
849                 tempGPsierr=sqrt(tempGPsierr*(t(2)-t(1)));    % total L^2 norm ...
                        ([0,1] x [0,T])
850                 tempExtPsierr=sqrt(tempExtPsierr*(t(2)-t(1)));    % total L^2 ...
                        norm ([0,1] x [0,T])
851                 tempExpPsierr=sqrt(tempExpPsierr*(t(2)-t(1)));    % total L^2 ...
                        norm ([0,1] x [0,T])
852                 tempFEMtrue=sqrt(tempFEMtrue*(t(2)-t(1)));        % total L^2 ...
                        norm ([0,1] x [0,T])
853
854 %                ROMerr(i)=tempPsierr/tempFEMtrue;
855
856                 Psirelerr(i)=tempPsierr/(tempFEMtrue);
857                 GPsirelerr(i)=tempGPsierr/(tempFEMtrue);
858                 ExtPsirelerr(i)=tempExtPsierr/(tempFEMtrue);
859                 ExpPsirelerr(i)=tempExpPsierr/(tempFEMtrue);
860
861
862
863
864             end % End Parameter step Loop
865             delete(zz)
866
867             ParamTesttoc2 = toc;         % parameter test time
868             ParamTesttime = ParamTesttoc2 - ParamTesttoc1
869
870             savefile = ['POD',num2str(r),'relerr.mat'];
871                 save(savefile, 'Psirelerr')
872
873             savefile = ['GPOD',num2str(r),'relerr.mat'];
874                 save(savefile, 'GPsirelerr')
875
876             savefile = ['ExPOD',num2str(r),'relerr.mat'];
877                 save(savefile, 'ExtPsirelerr')
878
879             savefile = ['EpPOD',num2str(2*r),'relerr.mat'];
880                 save(savefile, 'ExpPsirelerr')
881
882         end % End POD basis dimension Loop
883         delete(z)
884 end
885
886
887
888 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
889 % Parameter dependence plots for POD and GPOD
890 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
891 switch Parameter_Dep_Plots
892     case 1
893
894         % Load data
895
896         load POD2relerr
897         P2 = Psirelerr;
898
899         load POD4relerr
```

```
900         P4 = Psirelerr;
901
902         load POD5relerr
903         P5 = Psirelerr;
904
905         load POD8relerr
906         P8 = Psirelerr;
907
908         load POD10relerr
909         P10 = Psirelerr;
910
911         load POD16relerr
912         P16 = Psirelerr;
913
914         load GPOD2relerr
915         GP2 = GPsirelerr;
916
917         load GPOD4relerr
918         GP4 = GPsirelerr;
919
920         load GPOD5relerr
921         GP5 = GPsirelerr;
922
923         load GPOD8relerr
924         GP8 = GPsirelerr;
925
926         load GPOD10relerr
927         GP10 = GPsirelerr;
928
929         load GPOD16relerr
930         GP16 = GPsirelerr;
931
932         %POD plot
933         figure('name',['Relative Error with ',num2str(r),' basis vectors and n ...
                = ',num2str(n+1),' baseline 1/q = ',num2str(1/qbaseline)])
934         semilogy(REtest,P2,REtest,P4,REtest,P5,REtest,P8,REtest,P10,REtest,P16)
935         xlabel('1/q')
936         ylabel('relative error')
937         legend('r=2','r=4','r=5','r=8','r=10','r=16','Location','SouthEast')
938
939         %GPOD plot
940         figure('name',['GPOD Relative Error with ',num2str(r),' basis vectors ...
                and n = ',num2str(n+1),' baseline 1/q = ',num2str(1/qbaseline)])
941         semilogy(REtest,GP4,REtest,GP5,REtest,GP8,REtest,GP10,REtest,GP16)
942         xlabel('1/q')
943         ylabel('relative error')
944         legend('r=4','r=5','r=8','r=10','r=16','Location','SouthEast')
945
946         %POD and GPOD plot
947         figure('name',['POD and GPOD Relative Error'])
948         semilogy(REtest,GP4,'c-',REtest,GP5,'k-',REtest,GP8,'g-',REtest,GP10, ...
                'b-',REtest,GP16,'r-',REtest,P4,'c.-',REtest,P5,'k.-',REtest,P8,'g.-', ...
                REtest,P10,'b.-',REtest,P16,'r.-')
949         xlabel('1/q')
950         ylabel('relative error')
```

```
951         legend('GPOD r=4','GPOD r=5','GPOD r=8','GPOD r=10','GPOD r=16','POD ...
                r=4','POD r=5','POD r=8','POD r=10','POD r=16','Location','SouthEast')
952 end
953
954 toc
```

A.2 Input Functions

The following files are the input functions for forcing terms, initial conditions and Manufactured exact solutions.

```
1  function [f] = f_func(x,t)
2  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3  % Forcing Function for Burgers Equation
4  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
5
6  global q
7
8  f = 0;
9
10 %Exact solution test - for u(t,x) = h(t) * phi(x) * r(q), need f of form
11 %%% f = h_t(t)*phi(x)*r(q)+(h(t)*r(q))^2*phi(x)*phi'(x)-q*h(t)*r(q)*phi''(x)
12
13 %For u(t,x)=e^-t*sin(pi*x)
14 %f = (-exp(-t)*sin(pi*x))+ exp(-2*t)*sin(pi*x)*pi.*cos(pi*x) - ...
       q*exp(-t)*(-1*pi^2*sin(pi*x));
15
16 %For u(t,x)=e^-t*sin(pi*x)*q^2
17 %f = -q^2*exp(-t).*sin(pi*x) + pi^2*q^3*exp(-t).*sin(pi*x) + ...
       pi*q^4*exp(-2*t).*cos(pi*x).*sin(pi*x);
```

```
1  function [fq] = f_q_func(x,t)
2  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3  % Forcing Function for Sensitivity Equation
4  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
5
6  global q
7
8  fq = 0;
9
10 %Exact solution test - for u(t,x) = h(t) * phi(x) need fq of form
11 %%% f = h_t(t)*phi(x)*r'(q)+2*r(q)*r'(q)*h(t)^2*phi(x)*phi'(x)-
12 %%% ( h(t)*r(q)*phi''(x) + q*h(t)*r'(q)*phi''(x) )
13
14 % u(t,x)=e^-t*sin(pi*x)
15 %fq = -exp(-t)*pi^2*sin(pi*x);
16
17 %For u(t,x)=e^-t*sin(pi*x)*q^2
```

```
%fq = -2*q*exp(-t).*sin(pi*x) + 3*pi^2*q^2*exp(-t).*sin(pi*x) + ...
    4*pi*q^3*exp(-2*t).*cos(pi*x).*sin(pi*x);
```

```
function [g] = g_func(x)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Initial Condition Function for Burgers' Equation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

global q

%For u(t,x)=e^-t*sin(pi*x)
%g = sin(pi*x);

%For u(t,x)=e^-t*sin(pi*x)*q^2
%g = sin(pi*x)*q^2;

%Square Wave Input
%%{
for i=1:length(x)
    if x(i)<= (1/4)
        g(i) = 1;
    else
       g(i) = 0;
    end
end
%}
```

```
function [gq] = g_q_func(x)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Initial Condition Function for Sensitivity Equation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

global q

%For u(t,x)=e^-t*sin(pi*x)
gq = 0;

%For u(t,x)=e^-t*sin(pi*x)*q^2
%gq = sin(pi*x)*q*2;

%For u(t,x)=e^-t*sin(pi*x)*q^(1/2)
%gq=sin(pi*x)*1/(2*q^(1/2));

%Square Wave Input
gq=0;
```

```
function [g] = exact_func(t,x)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% MMS Exact Function for Burgers' Equation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
5
6  global q
7
8  %For u(t,x)=e^-t*sin(pi*x)
9  %g = exp(-t).*sin(pi*x);
10
11 %For u(t,x)=e^-t*sin(pi*x)*q^2
12 g=sin(pi*x).*exp(-t)*q^2;
```

```
1  function [g] = exact_sens_func(t,x)
2  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3  % MMS Exact Function for Sensitivity Equation
4  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
5
6  global q
7
8  %For u(t,x)=e^-t*sin(pi*x)
9  %g = 0;
10
11 %For u(t,x)=e^-t*sin(pi*x)*q^2
12 g=sin(pi*x).*exp(-t)*q*2;
```

A.3 Supporting Functions

The following files are the supporting functions used during the analysis in order of first usage.

```
1  function [F]=GQ_three(func)
2  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3  % This function calculates matrix from inner product with basis by
4  % 3 point Gaussian Quadrature
5  %
6  % **ASSUMES zero boundary conditions!!**
7  %
8  %
9  % Function inputs:
10 %
11 % func    :  Initial function (x) name (type string)
12 %
13 % Function outputs:
14 % F       :   Inner product of func and PLBF
15 %
16 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
17
18 global n h
19
20 F=zeros (n,1);
21
22 xweights = [5/9;8/9;5/9];
23 for i=1:n
```

```
24
25          xi=h*(i-1);
26          xf=xi+h;
27          xnodes = gaussnodes(xi,xf);
28          temp = feval(func,xnodes);
29
30          F(i)=(temp.*(1/h*(xnodes-xi)))*xweights*h/2;
31
32          xi=h*(i);
33          xf=xi+h;
34          xnodes = gaussnodes(xi,xf);
35          temp = feval(func,xnodes);
36
37          F(i)=F(i)+(temp.*(1/h*(xf-xnodes)))*xweights*h/2;
38
39  end
```

```
1  function [x] = gaussnodes(xi,xf)
2  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3  % Computes the  three gauss points on interval [xi,xf]
4  %
5  % xi         : initial x value
6  % xf         : final x value
7  %
8  % Outputs
9  %
10 % x          : vector of three GQ points
11 %
12 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
13
14 %Three nodes%
15 t(1) = -sqrt(3/5);
16 t(2) = 0;
17 t(3) = sqrt(3/5);
18
19
20 for i=1:3
21     x(i) = (xf-xi)/2*t(i)+(xf+xi)/2;
22 end
```

```
1  function [ftxy]=FEM_Burgers_func(in_t, in_y)
2  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3  % This function calculates the derivative of Burgers' equation
4  % and the sensitivity equation.
5  %
6  % Function inputs:
7  %
8  % in_t      :  current time
9  % in_y      :  current y
10 %
11 %
12 % Function outputs:
```

```
% out_y    :   computed derivative
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

global M B q C f;

F = RHS_F(f, in_t);

ftxy = M\ (F-(1/2)*B*in_y.^2-q*C*in_y);
```

```
function [F]=RHS_F(func, t)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This function calculates time dependent F matrix by
% 3 point Gaussian Quadrature
%
% Function inputs:
%
% func    :  RHS function name (string)
% t       :  current time
%
% Function outputs:
% F       :   Inner product of func and PLBF at t
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

global n h

xweights = [5/9;8/9;5/9];

F=zeros (n,1);
for i=1:n

        xi=h*(i-1);
        xf=xi+h;
        xnodes = gaussnodes(xi,xf);
        f = feval(func,xnodes,t);
        F(i)=(f.*(1/h*(xnodes-xi)))*xweights*h/2;


        xi=h*(i);
        xf=xi+h;
        xnodes = gaussnodes(xi,xf);
        f = feval(func,xnodes,t);
        F(i)=F(i)+(f.*(1/h*(xf-xnodes)))*xweights*h/2;


end
```

```
function [out_y]=FEM_Burgers_and_Sens_func(in_t, in_y)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This function calculates the derivative of Burgers' equation
% and the sensitivity equation.
```

```
%
% Function inputs:
%
% in_t      :  current time
% in_y      :  assembled vector of y and s
%
%
% Function outputs:
% out_y     :   computed derivative
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


global q n M B C f fq ;

%%Generate Matrices

F = RHS_F(f, in_t);

Fq = RHS_F(fq, in_t);

Bq = zeros(n,n);
    for i=1:n
        if i==1
            Bq(i,i) = (-in_y(i+1));
            Bq(i,i+1) = -(in_y(i)+2*in_y(i+1));
        end

        if i>1
            if i<n
                Bq(i,i-1) = (2*in_y(i-1)+in_y(i));
                Bq(i,i) = (in_y(i-1)-in_y(i+1));
                Bq(i,i+1) = -(in_y(i)+2*in_y(i+1));
            end
            if i==n
                Bq(i,i-1) = (2*in_y(i-1)+in_y(i));
                Bq(i,i) = (in_y(i-1));
            end
        end
    end
    Bq=Bq*1/6;

bigM = [M zeros(n);zeros(n) M];
bigF = [F-(1/2)*B*(in_y(1:n).^2); Fq];
bigK = [-q*C zeros(n);-C -q*C+Bq];

out_y = (bigM)\( bigF + bigK*in_y );
```

```
function [ab, e2n]=Error_GQ_three(t,y)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This function calculates integral of the FEM error from the
% exact function at time t by 3 point Gaussian Quadrature.
%
% Function inputs:
```

```
%
% t    :  current time step
% y    :  FEM solution vector at time t
%
% Function outputs:
%
% ab   :  absolute error integral
% e2n  :  integral of exact function
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

global x n h

%initialize
ab = 0;          %absolute
e2n = 0;

xweights = [5/9;8/9;5/9];           %3 point GQ weights

for i=1:n-1                         %loop calculates along each element

       xnodes = gaussnodes(x(i),x(i+1));            %Determine the quadrature nodes
       exactf = feval('exact_func',t,xnodes);       %Calulate the exact ...
           function at nodes
       interpy = (y(i)+( (y(i+1)-y(i))/(x(i+1)-x(i))*(xnodes-x(i)) )); %FEM ...
           solution at xnodes

       err = exactf-interpy;                        %error function to integrate

       ab = ab + err.^2*xweights;               %Add error over i-th interval

       e2n = e2n + exactf.^2*xweights;          %Compute L2 norm of exact function

end

ab = ab*h/2;
e2n = e2n*h/2;
```

```
function [ab, e2n]=SError_GQ_three(t,y)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This function calculates integral of the FEM error from the
% exact sensitivity function at time t by 3 point Gaussian Quadrature.
%
% Function inputs:
%
% t    :  current time step
% y    :  FEM solution vector at time t
%
% Function outputs:
%
% ab   :  absolute error integral
% e2n  :  integral of exact function
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
17
18 global x n h
19
20 %initialize
21 ab = 0;          %absolute
22 e2n = 0;
23
24 xweights = [5/9;8/9;5/9];           %3 point GQ weights
25
26 for i=1:n-1                         %loop calculates along each element
27
28        xnodes = gaussnodes(x(i),x(i+1));            %Determine the quadrature nodes
29        exactf = feval('exact_sens_func',t,xnodes);       %Calulate the exact ...
              function at nodes
30        interpy = (y(i)+( (y(i+1)-y(i))/(x(i+1)-x(i))*(xnodes-x(i)) )); %FEM ...
              solution at xnodes
31
32        err = exactf-interpy;                        %error function to integrate
33
34        ab = ab + err.^2*xweights;               %Add error over i-th interval
35
36        e2n = e2n + exactf.^2*xweights;          %Compute L2 norm of exact function
37
38 end
39
40 ab = ab*h/2;
41 e2n = e2n*h/2;
```

```
1  function [Psi,e,re]=POD_Basis(y,r,E)
2  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3  % This function calculates the POD basis vectors
4  %
5  % Function inputs:
6  % y       :  (n x m) snapshot matrix
7  %            n = number of states in full space (plus zero rows)
8  %            m = number of snapshots
9  % r       :  desired # of ROM basis vectors
10 % E       :  percent of energy desired
11 %
12 % Function outputs:
13 % Psi     :  (n x r) matrix containing ROM basis vectors
14 % e       :  percent total energy captured
15 % re      :  number of vectors necessary to capture E energy
16 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
17
18 global M
19
20 %%%Begin Function
21
22     n = size((y),1);
23     m = size((y),2);
24
25     k=min(n,m);
26
```

```
27      L = chol(M,'lower');      %Cholesky Decomp such that M=L*L'
28
29      wY=L'*y;
30
31      [U,S,V] = svd(wY);
32
33      %Determine Energy percentage
34      E2=zeros(n,1);
35      E2(1)=S(1,1)^2;
36
37      E1=zeros(n,1);
38      E1(1)=S(1,1)^2;
39      for i=2:k
40          E2(i)=E2(i-1)+S(i,i)^2;
41          E1(i)=S(i,i)^2;
42      end
43
44      E2=E2/E2(k);
45
46  %{
47      figure('name',['Singular Values'])
48        plot(E1,'o')
49        %title('Singular Values')
50        xlabel('n')
51  %}
52
53
54      %Determine number of POD basis required to meet energy
55      re=0;
56      j=0;
57      while j < E
58          re=re+1;
59          j=E2(re);
60      end
61
62  %{
63      figure('name',['Cum Singular Value Percentage'])
64        plot(E2,'o')
65        %title('Cum Singular Value Percentage')
66        xlabel('n')
67  %}
68
69
70      %Return first r vectors and energy captured
71      %Ur = U(:,1:r);
72      e=E2(r);
73
74      %Psi = L'^(-1)*U;
75      Psi = L'\U;
76      Psi = Psi(:,1:r);
77
78
79  %{
80      %Plot POD basis functions
81        %x values
```

```
82        h = 1/(n+1);
83        x = h:h:(1-h);
84
85        figure(499)
86        plot(x,Psi(:,1),x,Psi(:,2),x,Psi(:,3),x,Psi(:,4),x,Psi(:,5))
87        title('POD basis vectors')
88        xlabel('x')
89        legend('Psi1','Psi2','Psi3','Psi4','Psi5','Location','NorthWest')
90  %}
91  %{
92      %Plot Eigenvalues
93
94        figure('name','Eigenvalues Values')
95        plot(E1,'*')
96        xlabel('n')
97        ylabel('\Sigma_{ii}^2')
98
99
100 %}
101
102 %%%End Function
```

```
1   function [ytemp]=FEM_Burgers_ROM_func(in_t, in_y)
2   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3   % This function calculates the derivative of Burgers' equation
4   % POD ROM and the sensitivity equation.
5   %
6   % Function inputs:
7   %
8   % in_t      :  current time
9   % in_y      :  current y
10  %
11  %
12  % Function outputs:
13  % out_y     :  computed derivative
14  %
15  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
16
17  global B Mr Cr f ROM_Basis ;
18
19  F = RHS_F(f, in_t);
20
21  Fr = ROM_Basis'*F;
22
23  %POD with original nonlinear term
24  ytemp = Mr\(Fr - (1/2)*ROM_Basis'*(B*(ROM_Basis*in_y).^2) - Cr*in_y);
```

```
1   function [Psi, Psia, e, re]=POD_Basis_with_Sens(y,s,r,E)
2   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3   % This function calculates the POD basis vectors
4   %
5   % Function inputs:
```

```
% y       :  (n x m) snapshot matrix of FE coefficients
%            n = number of states in full space
%            m = number of snapshots
% s       :  (n x m) snapshot matrix of sensitivities
%            n = number of states in full space
%            m = number of snapshots
% r       :  desired # of ROM basis vectors
% E       :  percent of energy desired
%
% Function outputs:
% Psi     :  (n x r) matrix containing ROM basis vectors
% Psia    :  (n x r) matrix containing ROM basis sensitivity vectors
% e       :  percent total energy captured
% re      :  number of vectors necessary to capture E energy
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

global M

    n = size((y),1);
    m = size((y),2);
    k=min(n,m);

    L = chol(M,'lower');     %Cholesky Decomp such that M=L*L'

    wY=L'*y;

    [U,S,V] = svd(wY);

    %Determine Energy percentage
    E2=zeros(n,1);
    E2(1)=S(1,1)^2;
    for i=2:k
        E2(i)=E2(i-1)+S(i,i)^2;
    end
    E2=E2/E2(k);

    %Determine number of POD basis required to meet energy
    re=0;
    j=0;
    while j < E
        re=re+1;
        j=E2(re);
    end

    %Return first r vectors and energy captured
    e=E2(r);

    Psi = L'\U;
    Psi = Psi(:,1:r);

%%Generate Sensitivities of POD basis

B = wY*wY';
wS = L'*s;
Ba = wS*wY'+wY*wS';
```

```
61
62
63      for k=1:r
64          lr(k)=S(k,k)^2;
65          lra(k) = U(:,k)'*Ba*U(:,k);
66          sk = (B-lr(k)*eye(n))\( - (Ba-lra(k)*eye(n))*U(:,k));
67
68          %G-S step
69              Ura(:,k) = sk-(sk'*U(:,k))*U(:,k);
70      end
71
72  Psia = L'\Ura;
73
74  %figure()
75  %       plot(x,[0;Psi(:,2);0],x,[0;Psia(:,2);0])
76  %       title('POD basis vectors with sens')
77  %       xlabel('x')
78  %       legend('Psi2','Psi2 sens','Location','NorthWest')
79
80  %%%End Function
```