Model Reduction of the Coupled Burgers Equation in Conservation Form

Boris Krämer

Dissertation 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
Martin Day

August 23, 2011
Blacksburg, Virginia

Model Reduction of the Coupled Burgers Equation in Conservation Form

Boris Krämer

(ABSTRACT)

This thesis is a numerical study of the coupled Burgers equation. The coupled Burgers equation is motivated by the Boussinesq equations that are often used to model the thermal-fluid dynamics of air in buildings. We apply Finite Element Methods to the coupled Burgers equation and conduct several numerical experiments. Based on these results, the Group Finite Element method (GFE) appears to be more stable than the standard Finite Element Method. The design and implementation of controllers heavily relies on rapid solutions to complex models such as the Boussinesq equations. Thus, we further examine the feasibility and efficiency of the Proper Orthogonal Decomposition (POD) for the coupled Burgers equation. Using POD, we reduce the system to a "minimal" number of ODE's and conduct numerous numerical studies comparing the POD and GFE method. Further numerical experiments consider an application where the dynamics are projected on a POD basis and then the governing parameters of the system are varied.

Acknowledgements

I would like to express my deepest and most grateful thanks to my advisor and committee chairman Dr. John Burns for his continuous guidance, support and patience that made this research possible. His enthusiasm and insight made my pursuit of mathematics both fun and rewarding. Also, I would like to thank him for his general advice both on my future path of life and my career goals. I want to thank Dr. Martin Day and Dr. Jeff Borggaard for serving on my committee. Thanks also to Dr. Eugene Cliff and Dr. Serkan Gugercin for the supportive discussions we had about this work.

I wish to acknowledge the financial support I received from the Air Force Office of Scientific Research under Grant FA9550-10-10201 ("Computational Methods for Identification, Optimization and Control of PDE Systems") and the Department of Energy under Grant DOE DE-EE0004261 ("Advanced Computer Design Tools for Modeling, Design, Control, Optimization and Sensitivity Analysis of Integrated Whole Building Systems").

Thanks to the Institute of Mathematics of the Karlsruhe Institute of Technology (KIT) for allowing me to participate in this rewarding exchange program. I would like to express my special thanks to Dr. Wolfgang Reichel for his support before and during the exchange.

I would like to thank the group of the Interdisciplinary Center of Applied Mathematics (ICAM) for their friendship, kindness, constant encouragement and helpful discussions. The graduate student community in ICAM builds a fruitful ground for intensive research. Finally, I would like to give many thanks to all my friends here at Virginia Tech and my close friends in Germany for always being there and for helping me to maintain a social and intellectual piece of mind.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction and Motivation

In recent years, considerable attention has been devoted to the understanding and the design of energy efficiency in buildings [BBSZ09]. The adjustment and optimization of Heating, Ventilation and Air Conditioning (HVAC) systems has especially attracted the attention of the scientific community. One crucial aspect in understanding the complex dynamics of thermal flow in buildings arises from the problem of coupling a temperature field to a velocity field. This naturally involves the Navier-Stokes equation. Lighthill [Lig56] derived the Burgers equation as a second order approximation of the one-dimensional unsteady Navier-Stokes equation. Since then, Burgers equation has been used to test various theoretical and numerical ideas for potential application to the Navier-Stokes equation.

For the past century, Burgers equation has been subject to a huge number of studies, each with mathematical and physical motivations. In particular, Burgers equation has been applied to the study of traffic flow, acoustic transmission and supersonic flow around airfoils. As highlighted by Fletcher[Fle82], there are a wide variety of applications for Burgers equation. As a partial differential equation (PDE), Burgers equation also captures some of the interesting non-linear phenomena that occur in fluid flow, such as shock behavior. Thus, this relatively simple model can often provide insight into more complex fluid flow models.

The quasi-linear, unsteady Burgers equation is given by the PDE

$$w_t(t,x) + w(t,x)w_x(t,x) = \mu w_{xx}(t,x) \tag{1.1}$$

and is an evolution equation in an infinite dimensional space of the function w in time. Derivatives of w are denoted by subscripts x and t for space and time, respectively. In a physical setup, w is a velocity like dependent variable and μ is a viscosity coefficient that is

set as the inverse of the dimensionless Reynolds number in fluid dynamical applications.

The availability of exact solutions to Burgers equation [BP72] made it attractive for numerical analysis as a model problem to test various numerical schemes such as Finite Differences, Spectral Methods and Finite Element methods [Fle82], [GGS06],[HGG07],[KCGH07]. On an infinite spacial interval, the Cole-Hopf transformation[Col51], [Hop50] reduces Burgers equation to the linear heat equation. However, for bounded intervals with nonhomogeneous boundary conditions, the Cole-Hopf transformation may not produce a linear system. Thus, exact solutions are not always available in this case. Moreover, when Neumann boundary conditions are used, Burgers equation can be "infinitely sensitive" to perturbations in the boundary terms, which leads to numerical difficulties. The sensitivity of Burgers equation to boundary conditions has been investigated in [ABG08], [BGS01], [BBGS98], [BS01], [CT00], [GK97] and [LO93]. In [ABG$^+$02], it was shown that due to finite precision arithmetic numerical algorithms produce non-constant solutions of Burgers equation. Hence, although theoretical results might be available for a dynamical system, one should always be aware of the existence of purely numerical solutions.

Since Burgers equation is a model that captures the competition of convection and diffusion, it has been used to study fluid flow[Bur48]. In this thesis, we couple Burgers equation to another convection diffusion equation (or heat equation) to take into account the influence of the temperature field on the velocity field. This simple model is motivated by the Boussinesq equations that describe the incompressible fluid flow coupled to thermal dynamics. The Boussinesq equations are often used to model the thermal-fluid dynamics of air in a building.

Thus, we focus on Burgers equation coupled to the heat equation so that the system of interest is defined by the coupled PDE's

$$w_t(t,x) + w(t,x)w_x(t,x) = \mu w_{xx}(t,x) - \kappa T(t,x), \tag{1.2}$$

$$T_t(t,x) + w(t,x)T_x(t,x) = cT_{xx}(t,x) + f(t,x). \tag{1.3}$$

System (1.2)-(1.3) is referred to as the coupled Burgers equation. The function T can be viewed as a temperature field where c is the thermal conductivity and κ is the coefficient of the thermal expansion. Henceforth, we will refer to w as the "velocity" and T as the temperature. We also introduce the forcing on the system f that can also be viewed as a disturbance. System (1.2)-(1.3) is a coupled nonlinear system and has the basic structure of the Boussinesq equations. Herein, the temperature drives the velocity field and the velocity field provides the convective term.

Solving coupled nonlinear partial differential equations is a computationally challenging task. With the rise of computational capacity and power, more exact and realistic models can be solved that were not possible to simulate 30 years ago. We shall focus on Finite Element

Methods (FEM) since there is a rigorous theory for convergence and error estimation. In particular, we consider the standard FEM and its variation, the Group Finite Element Method (GFE).

For Burgers equation, Fletcher [Fle83] showed that the GFE method has some computational advantages over the FEM. Other studies have confirmed these advantages with numerical simulations, see [Pug95], [Smi97], [Ngu01]. However, full model FEM or GFE solutions can be computationally expensive. One motivation for this study is to produce computational tools that can be used for design, optimization and control of energy efficient buildings. Thus, we need new computational methods that allow for rapid (even real-time) computation.

For design and implementation of controllers, one needs quickly executable or even real time solutions of complex models such as the Boussinesq system. Thus, some type of model reduction is required. The method of Proper Orthogonal Decomposition (POD) was first introduced by Loéve [Loè55, 1955] and provides a popular approach to model reduction for general dynamical systems. A key idea behind POD is called the "method of snapshots", which is described in detail in [Sir87]. Roughly, the approach is to obtain time "snapshots" (solutions) of the system from either experimental or numerical data. In particular, these snapshots can be obtained from high fidelity simulations of the full model. Subsequently, "empirical" basis functions are constructed that contain information about the dynamics of the system. In many cases, when employing the POD for model reduction, one obtains good results in terms of accuracy and computational speed-up. Thus, POD provides a promising tool for real-time solutions of complex systems once approximate information about the dynamics can be made available. We shall apply the POD method to the coupled Burgers equation (1.2)-(1.3) and investigate its numerical properties. The recent work of [DS10] suggests to group the nonlinearity in Burgers equation due to proven speed-up for computations.

Dynamical systems are often governed by a set of parameters. However, parameters might be subject to change over time and define a different dynamics. In the setup of control design, controllers should be constructed that are performing well even if the parameters of the system change. The POD is a promising tool for applications in control design and optimization. However, the sensitivity of the "physical" POD basis to parameters is important to assess the quality of a basis. Thus, we conduct several numerical experiments that are dedicated to get insight into the sensitivity of the obtained POD basis.

We begin with a review of notation and then formulate the coupled Burgers equation. In Chapter 2 we discuss the applications of the Finite Element and Group Finite Element method to the coupled Burgers equation. We conduct several numerical experiments to test the accuracy of the FEM and GFE methods. The Method of Manufactured Solutions is

applied [Roa04] to construct analytical solutions for comparison. In Chapter 3 we derive the POD models for the coupled Burgers equation and use the GFE to generate data. A comparison of the GFE and POD will be given. Finally, Chapter 4 provides conclusions and motivates interesting questions for further study.

1.2 Notation

In this thesis, we use the following notation. Let the Hilbert space $X = L^2([0,1])$ be the 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 $< \cdot, \cdot >_X$ 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

$$< f, g >_X = \int_0^1 f(x)g(x) \ dx \ .$$

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

$$\int_0^T \|w(t, \cdot)\|_X^2 dt < \infty \ .$$

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

We now define the Sobolev spaces $H_0^1([0,1])$ and $H_L^1([0,1])$ as

$$H_0^1([0,1]) = \left\{ f \mid f \in L^2([0,1]), \ f_x \in L^2([0,1]), \ f(0) = 0, f(1) = 0 \right\} \ ,$$

$$H_L^1([0,1]) = \left\{ f \mid f \in L^2([0,1]), \ f_x \in L^2([0,1]), \ f(0) = 0 \right\} \ .$$

These function spaces are subspaces of $L^2([0,1])$ with Lebesgue square integrable derivatives and different boundary conditions on the functions.

Let M be an $m \times n$ matrix. For $i \leq j \leq m, k \leq l \leq n$ the matrix $M(i:j, k:l)$ denotes the submatrix of M containing all entries of M between and including the i^{th} and j^{th} row and the k^{th} and l^{th} column[1] .

Finally, the Kronecker delta is defined by

$$\delta_{ij} = \begin{Bmatrix} 0, \text{ if } & i \neq j \\ 1, \text{ if } & i = j \end{Bmatrix} .$$

1.3 Formulation of the Coupled Burgers Equation

A physical motivation and interpretation of the relevant variables and constants occurring in the coupled Burgers equation has been given above. The coupled Burgers equation has the form

$$\begin{aligned} w_t(t,x) + w(t,x)w_x(t,x) &= \mu w_{xx}(t,x) - \kappa T(t,x) , \quad x \in (0,1),\ t > 0 , & (1.4) \\ T_t(t,x) + w(t,x)T_x(t,x) &= cT_{xx}(t,x) + f(t,x) , \quad x \in (0,1),\ t > 0 . & (1.5) \end{aligned}$$

The forcing f is assumed to be at least L^2 in space and time. We shall focus on a particular boundary value problem of the form

$$\begin{aligned} w(t,0) = 0 , \qquad & w_x(t,1) = \delta \in \mathbb{R} , & (1.6) \\ T(t,0) = 0 , \qquad & T(t,1) = 0 , & (1.7) \end{aligned}$$

where $0 \leq \delta \ll 1$. The Neumann boundary condition for the velocity is motivated by [ABG+02]. Herein, Burgers equation was studied and a high sensitivity with respect to δ was observed.

The initial condition for the velocity is given by

$$w(0,x) = w_0(x) \qquad \in L^2([0,1]) \tag{1.8}$$

and the initial condition for the temperature is

$$T(0,x) = T_0(x) \qquad \in L^2([0,1]) . \tag{1.9}$$

The problem is a partial differential equation with an infinite dimensional solution space $L^2((0,\infty), X)$, where $X = L^2([0,1])$. Thus, a reduction to a finite dimensional function space is the first step in obtaining approximate solutions.

[1] If the reader is familiar with MATLAB ® this notation will be intuitive

Chapter 2

Finite Element Methods

The standard Finite Element Method (FEM) is a powerful and frequently used numerical method to find approximate solutions of PDE's. The method involves several steps that approximate the PDE by a set of ordinary differential equations (ODE's). First, a weak formulation to the coupled Burgers equation is derived. To find an approximate solution a finite dimensional function space is constructed. This is done by choosing piecewise linear basis functions where both the temperature T and the velocity w can be approximated. The weak formulation is projected onto these Finite Element spaces and the discretized system is obtained. The resulting approximate system is simplified and rewritten to obtain a vector valued ordinary differential equation.

A second approach is based on approximations of Burgers equation in conservation form. In particular, the grouped variable is expressed in the same basis as the standard variable. This method is called the Group Finite Element method (GFE) or Conservation method. Numerical simulations have shown some advantages of the GFE with respect to the stability of solutions and in some cases also in computational efficiency. We refer the reader to the references given in the introduction.

In the last section, numerical examples are provided with the Method of Manufactured Solutions. We compare results for various MATLAB ® ODE solvers and mesh sizes. The conclusions are made based on considering computational cost and global errors as well as stability issues.

For convenience, we state the equations once more to formulate the FE model.

2.1 The Coupled Burgers Equation

Consider the coupled Burgers equation

$$w_t(t,x) + w(t,x)w_x(t,x) = \mu w_{xx}(t,x) - \kappa T(t,x) \ , \tag{2.1}$$
$$T_t(t,x) + w(t,x)T_x(t,x) = cT_{xx}(t,x) + f(t,x) \ , \tag{2.2}$$

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

$$w(t,0) = 0 \ , \qquad w_x(t,1) = \delta \ , \tag{2.3}$$
$$T(t,0) = 0 \ , \qquad T(t,1) = 0 \ , \tag{2.4}$$

and initial conditions

$$w(0,x) = w_0(x) \ , \quad \in \ L^2([0,1]) \ , \tag{2.5}$$
$$T(0,x) = T_0(x) \ , \quad \in \ L^2([0,1]) \ . \tag{2.6}$$

A vector formulation of system (2.1)-(2.2) is written as

$$\frac{d}{dt}\begin{bmatrix} w(t,x) \\ T(t,x) \end{bmatrix} + w(t,x)\frac{d}{dx}\begin{bmatrix} w(t,x) \\ T(t,x) \end{bmatrix} = \frac{d^2}{dx^2}K_1\begin{bmatrix} w(t,x) \\ T(t,x) \end{bmatrix} + K_2\begin{bmatrix} w(t,x) \\ T(t,x) \end{bmatrix} + \begin{bmatrix} 0 \\ f(t,x) \end{bmatrix} \tag{2.7}$$

and is used as a starting point for the weak formulation. The coefficient matrices K_1 and K_2 are defined by

$$K_1 = \begin{bmatrix} \mu & 0 \\ 0 & c \end{bmatrix} \qquad K_2 = \begin{bmatrix} 0 & -\kappa \\ 0 & 0 \end{bmatrix} \ .$$

2.2 The Finite Element Basis and Test Spaces

The Finite Element Method reduces the space dimension from infinite to finite. Arising from the complexity and specific features of the problem, a choice about the basis has to be made. This basis should be suitable to approximate the solution. Herein, we define piecewise linear basis functions for the approximate solutions w^N, T^N of w and T. Thus, the approximating spaces are the span of these basis functions. The so called "hat" functions deliver accurate numerical results for FEM models. The boundary conditions (2.3)-(2.4) determine if one needs basis functions that are non-zero on the boundaries to approximate either w or T.
The piecewise linear "hat" functions have "small" local support. Therefore, the mass and stiffness matrix are tridiagonal. This feature minimizes computational effort[1]. Throughout,

[1]Matrices are stored as 'sparse' matrices in MATLAB ® which makes operations like inversion and multiplication more efficient in terms of memory space and computational time.

the spatial domain is divided in $N+1$ subintervals $[x_i, x_{i+1}]$ of equal length $\frac{1}{N+1}$, where $x_i = \frac{i}{N+1}$ with $i = 0, 1, \ldots, N$.

We approximate the temperature T with the basis functions

$$\psi_i(x) = \begin{cases} (N+1)(x - x_{i-1}), & \text{if } x_{i-1} \leq x \leq x_i \\ -(N+1)(x - x_{i+1}), & \text{if } x_i \leq x \leq x_{i+1} \\ 0, & \text{else.} \end{cases} \qquad \text{for } i = 1, \ldots, N \ ,$$

Note that due to the Dirichlet boundary conditions (2.4) for the temperature one only needs N basis functions. For the velocity w we have a Dirichlet boundary condition at $x = 0$ and a Neumann boundary condition given by the parameter δ at $x = 1$, see (2.3). Hence, we define the following piecewise linear basis functions

$$\phi_j(x) = \begin{cases} (N+1)(x - x_{j-1}), & \text{if } x_{j-1} \leq x \leq x_j \\ -(N+1)(x - x_{j+1}), & \text{if } x_j \leq x \leq x_{j+1} \\ 0, & \text{else.} \end{cases} \qquad \text{for } j = 1, \ldots, N \ ,$$

Due to the Neumann boundary condition we set in addition the last basis function as

$$\phi_{N+1}(x) = \begin{cases} (N+1)(x - x_N), & \text{if } x_N \leq x \leq x_{N+1} \\ 0, & \text{else.} \end{cases}$$

A noteworthy property of these basis functions is that $\phi_i = \psi_i$ for $i = 1, \ldots, N$. That is, the temperature and velocity field is represented in a similar space. In general, this is not required. For instance, ϕ could be a cubic basis function and ψ piecewise linear. However, for our choice this simplifies both computations and the implementation. However, we distinguish theoretically between ϕ and ψ in order to emphasize that they could be defined differently depending on model requirements. Moreover, we have $\phi_i(x_j) = \psi_i(x_j) = \delta_{ij}$ for $i, j = 1, \ldots, N+1$.

Hence, the FE function spaces for the velocity w and temperature T are the spans of the respective basis functions and are given by

$$V_w^N = \left\{ \sum_{i=1}^{N+1} \alpha_i \phi_i(x) : \alpha_i \in \mathbb{R}, i = 1, \ldots, N+1 \right\} \ ,$$

$$V_T^N = \left\{ \sum_{i=1}^{N} \beta_i \psi_i(x) : \beta_i \in \mathbb{R}, i = 1, \ldots, N \right\} \ .$$

The approximate solutions of the coupled Burgers equation are expanded with respect to the FE basis as

$$w(t, x) \approx w^N(t, x) = \sum_{i=1}^{N+1} \alpha_i(t) \phi_i(x) \ , \tag{2.8}$$

$$T(t,x) \approx T^N(t,x) = \sum_{i=1}^{N} \beta_i(t)\psi_i(x) \ , \tag{2.9}$$

Here, $\alpha_i(\cdot)$ and $\beta_i(\cdot)$ are nodal unknowns of the system. In the next sections we derive a system that governs these unknowns. The weak form of the problem using the basis functions as test functions is worked out in the following section.

2.3 Weak Formulation and Galerkin Principle

In this section, the weak formulation for the coupled Burgers problem (2.1)-(2.6) is derived and the Galerkin principle applied. To construct the weak form of (2.7) assume that the test functions $[\phi \ \psi]^\mathsf{T} \in L^2([0,1]) \times L^2([0,1])$, then take the L^2 scalar product of (2.7) to obtain

$$\begin{aligned} &< \begin{bmatrix} w_t(t,x) \\ T_t(t,x) \end{bmatrix}, \begin{bmatrix} \phi(x) \\ \psi(x) \end{bmatrix} > \ + \ < \begin{bmatrix} w(t,x)w_x(t,x) \\ w(t,x)T_x(t,x) \end{bmatrix}, \begin{bmatrix} \phi(x) \\ \psi(x) \end{bmatrix} > \\ &\quad =< \begin{bmatrix} \mu w_{xx}(t,x) \\ cT_{xx}(t,x) \end{bmatrix}, \begin{bmatrix} \phi(x) \\ \psi(x) \end{bmatrix} > \ + \ < \begin{bmatrix} -\kappa T(t,x) \\ 0 \end{bmatrix}, \begin{bmatrix} \phi(x) \\ \psi(x) \end{bmatrix} > + \ < \begin{bmatrix} 0 \\ f(t,x) \end{bmatrix}, \begin{bmatrix} \phi(x) \\ \psi(x) \end{bmatrix} > \\ &\qquad\qquad \forall \, [\phi \ \psi]^\mathsf{T} \in L^2([0,1]) \times L^2([0,1]) \ . \end{aligned}$$

Evaluating the L^2 inner products on both sides produces the equations

$$\begin{aligned} &\int_0^1 \left[w_t(t,x)\phi(x) + T_t(t,x)\psi(x) \right] dx + \int_0^1 \left[w(t,x)w_x(t,x)\phi(x) + w(t,x)T_x(t,x)\psi(x) \right] dx \\ &\quad = \int_0^1 \left[\mu w_{xx}(t,x)\phi(x) + cT_{xx}(t,x)\psi(x) \right] dx - \int_0^1 \kappa T(t,x)\phi(x)dx + \int_0^1 f(t,x)\psi(x)dx \\ &\qquad\qquad \forall \, [\phi \ \psi]^\mathsf{T} \in L^2([0,1]) \times L^2([0,1]) \ . \end{aligned} \tag{2.10}$$

A straight forward integration by parts yields

$$\begin{aligned} &\int_0^1 \left[\mu w_{xx}(t,x)\phi(x) + cT_{xx}(t,x)\psi(x) \right] dx = - \int_0^1 \mu w_x(t,x)\phi'(x)dx + \mu \left[w_x(t,x)\phi(x) \right]_0^1 \\ &\quad - \int_0^1 cT_x(t,x)\psi'(x)dx + c\left[T_x(t,x)\psi(x) \right]_0^1 \qquad \forall \, [\phi \ \psi]^\mathsf{T} \in H^1([0,1]) \times H^1([0,1]) \ . \end{aligned} \tag{2.11}$$

The restriction on the derivatives of the test functions needs to be made, since after integrating by part, the derivatives occur under the integral. Thus, one has to ensure that the

integral is well defined.

We emphasize that the geometric boundary conditions are $w(t,0) = 0$ and $T(t,0) = T(t,1) = 0$. The test functions ϕ, ψ are chosen as the basis functions[2] which are constructed to reproduce the solution. Thus, the basis functions for T have to vanish at $x = 0$ and $x = 1$ and the basis functions for w have to vanish at $x = 0$. To be precise, $\phi(0) = 0$ and $\psi(0) = \psi(1) = 0$. Note that $w_x(1) = \delta$ is specified, see (2.3). Using these properties for the test functions, (2.11) simplifies to

$$\int_0^1 \left[\mu w_{xx}(t,x)\phi(x) + cT_{xx}(t,x)\psi(x)\right] dx = -\int_0^1 \mu w_x(t,x)\phi'(x) + cT_x(t,x)\psi'(x)dx + \mu\delta\phi(1)$$
$$\forall \left[\phi\ \psi\right]^{\mathsf{T}} \in H_L^1([0,1]) \times H_0^1([0,1])\ .$$

Rearranging terms and plugging in the results of the integration by parts into (2.10) yields

$$\begin{aligned}
&\int_0^1 \left[w_t(t,x)\phi(x) + T_t(t,x)\psi(x)\right] dx = \\
&-\int_0^1 \left[w(t,x)w_x(t,x)\phi(x) + w(t,x)T_x(t,x)\psi(x)\right] dx - \int_0^1 \left[\mu w_x(t,x)\phi'(x) + cT_x(t,x)\psi'(x)\right] dx \\
&+ \mu\delta\phi(1) - \int_0^1 \kappa T(t,x)\phi(x)dx + \int_0^1 f(t,x)\psi(x)dx \qquad \forall \left[\phi\ \psi\right]^{\mathsf{T}} \in H_L^1([0,1]) \times H_0^1([0,1])\ .
\end{aligned} \tag{2.12}$$

Definition: The functions $w(t,\cdot) \in H_L^1([0,1])$ and $T(t,\cdot) \in H_0^1([0,1])$ are called **weak solutions** to (2.1)-(2.6) if they satisfy (2.12)[3] for $t > 0$, $\forall \left[\phi\ \psi\right]^{\mathsf{T}} \in H_L^1([0,1]) \times H_0^1([0,1])$ and

$$\begin{aligned}
w(0,x) = w_0(x)\ , &\qquad \in\ L^2([0,1])\ , \\
T(0,x) = T_0(x)\ , &\qquad \in\ L^2([0,1])\ .
\end{aligned}$$

The weak solutions are still analytical solutions and require approximations. Thus, a reduction of the solutions space to finite dimensions is inevitable. This is the main idea of the Galerkin method.

[2]This is explained in the following part about the Galerkin principle

[3]In the derivation of the weak form one should note that the boundary conditions are already included in (2.12)

The Galerkin principle addresses the issue of finding an approximation to the weak solution by combining the weak form and a projection. This ansatz employs the basis functions as test functions. Recall that the geometrical boundary conditions of the basis functions carry over to the test functions. Let $\phi_i \in V_w^N \subseteq H_L^1([0,1])$ for $i = 1, \ldots, N+1$ and $\psi_j \in V_T^N \subseteq H_0^1([0,1])$ for $j = 1, \ldots, N$ be the FE basis functions. Moreover, assume w^N, T^N are the approximations to w, T in these function spaces. Hence, the weak form (2.12) can be stated in terms of approximate solutions in the subspace $V_w^N \times V_T^N$ of $H_L^1([0,1]) \times H_0^1([0,1])$. It follows that

$$\int_0^1 \left[w_t^N(t,x)\phi(x) + T_t^N(t,x)\psi(x) \right] dx =$$

$$- \int_0^1 \left[w^N(t,x) w_x^N(t,x)\phi(x) + w^N(t,x) T_x^N(t,x)\psi(x) \right] dx$$

$$- \int_0^1 \left[\mu w_x^N(t,x)\phi'(x) + c T_x^N(t,x)\psi'(x) \right] dx + \mu\delta\phi(1) - \int_0^1 \kappa T^N(t,x)\phi(x)dx + \int_0^1 f(t,x)\psi(x)dx$$

$$\forall [\phi \ \psi]^\mathsf{T} \in V_w^N \times V_T^N \ . \quad (2.13)$$

This leads to the following

Definition: The functions $w^N(t,\cdot) \in V_w^N$ and $T^N(t,\cdot) \in V_T^N$ are called **approximations of the weak solution** to (2.1)-(2.6) if they satisfy (2.13) for $t > 0$ and

$$w(0,x) \approx \ w^N(0,x) = \ P_{V_w^N} \ w_0(x) \ ,$$
$$T(0,x) \approx \ T^N(0,x) = \ P_{V_T^N} \ T_0(x) \ .$$

The P operator denotes the projection on the respective spaces. In particular, $P_{V_w^N} : L^2([0,1]) \to V_w^N$ and $P_{V_T^N} : \ L^2([0,1]) \to V_T^N$ are the orthogonal projections onto V_w^N and V_T^N, respectively.

2.4 The Finite Element Approximation

The FEM approximates the system of PDE's by a set of ODE's. The core part of the method is the weak formulation for the approximate solutions (2.13). The approximate solutions are substituted with the expansions (2.8)-(2.9). The weak formulation (2.13) is formulated to be valid for all basis functions $\phi_i \in V_w^N, \ i = 1, \ldots, N+1$ and all $\psi_l \in V_T^N, \ l = 1, \ldots, N$. According to the definition of these basis functions, we have $\psi_l(0) = \psi_l(1) = 0$ for $l = 1, \ldots, N$ and $\phi_j(0) = 0$ for $j = 1, \ldots, N+1$. Moreover, $\phi_j(1) = 0$ for $j = 1, \ldots, N$ and

$\phi_{N+1}(1) = 1$. The last equality arises from the Neumann boundary condition at $x = 1$. Hence, for $j = 1, \dots, N+1$ and $l = 1, \dots, N$ it follows that

$$\begin{aligned}
&\sum_{i=1}^{N+1} \dot{\alpha_i}(t) \int_0^1 \phi_i(x)\phi_j(x)dx + \sum_{i=1}^{N} \dot{\beta}_i(t) \int_0^1 \psi_i(x)\psi_l(x)dx \\
&= -\sum_{i=1}^{N+1}\sum_{k=1}^{N+1} \int_0^1 \alpha_i(t)\alpha_k(t)\phi_i(x)\phi_k'(x)\phi_j(x)dx - \sum_{i=1}^{N+1}\sum_{k=1}^{N} \int_0^1 \alpha_i(t)\beta_k(t)\phi_i(x)\psi_k'(x)\psi_l(x)dx \\
&\qquad - \mu \sum_{i=1}^{N+1} \alpha_i(t) \int_0^1 \phi_i'(x)\phi_j'(x)dx - c\sum_{i=1}^{N} \beta_i(t) \int_0^1 \psi_i'(x)\psi_l'(x)dx \\
&\qquad + \mu\delta\phi_{N+1}(1) - \kappa \sum_{i=1}^{N} \beta_i(t) \int_0^1 \psi_i(x)\phi_j(x)dx + \int_0^1 f(t,x)\psi_l(x)dx \ . \quad (2.14)
\end{aligned}$$

We denoted the spatial derivatives "$\frac{d}{dx} = \ '$" and the time derivatives "$\frac{d}{dt} = \dot{}$".

The goal is to express (2.14) as a matrix vector equation which can be implemented in MATLAB ®. Moreover, one has to find approximations of the initial conditions in order to obtain approximate weak solutions.

Linear Terms First, we start with a treatment of the linear terms in (2.14). In the case of time independent basis functions, the FE mass and stiffness matrix can be computed offline. This is advantageous for the linear terms only impose a matrix-vector product in each time step of the ODE solution algorithm. Since the piecewise linear basis is known explicitly, the matrices can be computed exactly. In this step, no numerical errors are introduced. The mass matrix for the velocity term is defined by

$$M_w^N = (m_{ij}^w)_{i,j=1,\dots,N+1} = \left(\int_0^1 \phi_i(x)\phi_j(x) \right)_{i,j=1,\dots,N+1}$$

and the mass matrix needed for the temperature term is defined by

$$M_T^N = (m_{ij}^T)_{i,j=1,\dots,N} = \left(\int_0^1 \psi_i(x)\psi_j(x) \right)_{i,j=1,\dots,N} .$$

The definitions of the basis functions are used to compute the integrals. Note that in the special case of $\phi_i = \psi_i$ for $i = 1, \dots, N$ the computational effort reduces and we obtain

$$M_T^N = M_w^N(1:N, 1:N) \ .$$

The lack of the $(N+1)^{th}$ basis function for T implies that the last row and column of M_w^N cancels out. Thus, we restrict our focus to calculating M_w^N. We obtain the explicit form for M_w^N as the tridiagonal mass matrix

$$M_w^N = \frac{1}{6(N+1)} \begin{bmatrix} 4 & 1 & 0 & \dots & \dots & 0 \\ 1 & 4 & 1 & 0 & \dots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & 1 & 4 & 1 \\ 0 & \dots & \dots & 0 & 1 & 2 \end{bmatrix}_{(N+1)\times(N+1)} .$$

Using this notation, the global mass matrix for the coupled Burgers equation can be conveniently assembled. In particular, $\forall j = 1, \dots, N+1$ and $l = 1, \dots, N$ it follows

$$\underbrace{\begin{bmatrix} M_w^N & 0 \\ 0 & M_T^N \end{bmatrix}}_{\hat{M^N}} \begin{bmatrix} \dot{\alpha}_1(t) \\ \vdots \\ \alpha_{N+1}(t) \\ \beta_1(t) \\ \vdots \\ \beta_N(t) \end{bmatrix} = \sum_{i=1}^{N+1} \dot{\alpha}_i(t) \int_0^1 \phi_i(x)\phi_j(x)dx + \sum_{i=1}^{N} \dot{\beta}_i(t) \int_0^1 \psi_i(x)\psi_l(x)dx \ .$$

A similar procedure can be done with the stiffness matrix. The stiffness matrix for the velocity is defined as

$$S_w^N = (s_{ij}^w)_{i,j=1,\dots,N+1} = \left(\int_0^1 \phi_i'(x)\phi_j'(x) \right)_{i,j=1,\dots,N+1}$$

and likewise for the temperature

$$S_T^N = (s_{ij}^T)_{i,j=1,\dots,N} = \left(\int_0^1 \psi_i'(x)\psi_j'(x) \right)_{i,j=1,\dots,N} .$$

An equivalent result as with the mass matrix can be obtained. In the special case that $\phi_i = \psi_i$ for $i = 1, \dots, N$,

$$S_T^N = S_w^N(1:N, 1:N) \ .$$

The stiffness matrices are also tridiagonal for only "neighboring" basis functions can have a non-zero product of their respective derivatives. Consequently, the stiffness matrix for the

velocity variable is

$$S_w^N = (N+1)\begin{bmatrix} 2 & -1 & 0 & \dots & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & -1 & 2 & -1 \\ 0 & \dots & \dots & 0 & -1 & 1 \end{bmatrix}_{(N+1)\times(N+1)} .$$

To assemble the global stiffness matrix for the coupled Burgers Problem, we proceed as above for the mass matrix and obtain

$$\begin{bmatrix} \mu S_w^N & 0 \\ 0 & cS_T^N \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_{N+1} \\ \beta_1 \\ \vdots \\ \beta_N \end{bmatrix} = \mu \sum_{i=1}^{N+1} \alpha_i(t) \int_0^1 \phi_i'(x)\phi_j'(x)dx + c\sum_{i=1}^{N} \beta_i(t) \int_0^1 \psi_i'(x)\psi_l'(x)dx \quad .$$

Consider the expression for the coupling in equation (2.14)

$$\sum_{i=1}^{N} \beta_i(t) \int_0^1 \psi_i(x)\phi_j(x)dx \qquad \text{for } j = 1, \dots, N+1 .$$

Define the coupling matrix $C^N \in \mathbb{R}^{N+1} \times \mathbb{R}^N$ by

$$C_{ji}^N = \int_0^1 \phi_j(x)\psi_i(x)dx \quad \text{for } j = 1, .., N+1,\ i = 1, \dots, N . \tag{2.15}$$

In general, this matrix has to be computed separately since the basis functions for the two dependent variables can be chosen independently.
In the special case of $\phi_i = \psi_i$ for $i = 1, \dots, N$ the calculations are equivalent to the calculations of the mass matrix M_w^N. For we do not have ψ_{N+1} the last column of M_w^N cancels out. Thus, one obtains

$$C^N = M_w^N(1 : N+1, 1 : N) .$$

Nonlinear Terms We now turn to the nonlinear terms in (2.14). The first nonlinearity

results from Burgers equation and the second one is due to the coupling of velocity and temperature. The nonlinearities result in the time dependent vector $J(\alpha(t), \beta(t)) \in \mathbb{R}^{2N+1} \times \mathbb{R}$ which is given by[4]

$$J = \frac{1}{6}\begin{bmatrix} \alpha_1\alpha_2 + \alpha_2^2 \\ -\alpha_1^2 - \alpha_1\alpha_2 + \alpha_2\alpha_3 + \alpha_3^2 \\ \vdots \\ -\alpha_{N-1}^2 - \alpha_{N-1}\alpha_N + \alpha_N\alpha_{N+1} + \alpha_{N+1}^2 \\ -\alpha_N^2 - \alpha_N\alpha_{N+1} + 2\alpha_{N+1}^2 \\ 2\alpha_1\beta_2 - \alpha_2\beta_1 + \alpha_2\beta_2 \\ -\alpha_1\beta_1 + \alpha_1\beta_2 - 2\alpha_2\beta_1 + 2\alpha_2\beta_3 - \alpha_3\beta_2 + \alpha_3\beta_3 \\ \vdots \\ -\alpha_{N-2}\beta_{N-2} + \alpha_{N-2}\beta_{N-1} - 2\alpha_{N-1}\beta_{N-2} + 2\alpha_{N-1}\beta_N - \alpha_N\beta_{N-1} + \alpha_N\beta_N \\ -\alpha_{N-1}\beta_{N-1} + \alpha_{N-1}\beta_N - 2\alpha_N\beta_{N-1} - \alpha_{N+1}\beta_N \end{bmatrix}$$

$$= \sum_{i=1}^{N+1}\sum_{k=1}^{N+1}\int_0^1 \alpha_i(t)\alpha_k(t)\phi_i(x)\phi_k'(x)\phi_j(x)dx + \sum_{i=1}^{N+1}\sum_{k=1}^{N}\int_0^1 \alpha_i(t)\beta_k(t)\phi_i(x)\psi_k'(x)\psi_l(x)dx \ .$$

Forcing Function and Boundary Conditions To conclude the approximation, the forcing function $f(t,x)$ is multiplied by the basis ψ_i for $i = 1, \ldots, N$ and integrated. This integral has to be evaluated in every time step of the solution routine. Hence, complicated forcing functions increase the computational time remarkably. For brevity, we define the vector valued function of time

$$F^N(t) = \begin{bmatrix} \int_0^1 f(t,x)\psi_1(x)dx \\ \int_0^1 f(t,x)\psi_2(x)dx \\ \vdots \\ \int_0^1 f(t,x)\psi_N(x)dx \end{bmatrix}$$

and use $F^N(t)$ to force the FE model.

Finally, we discuss how the boundary conditions imposed by the Neumann boundary at $x = 1$ enter into the FE model. Let

$$\vec{\delta} = [0, \ldots, 0, \delta, 0, \ldots, 0]^\mathsf{T} \ ,$$

[4]The calculations that led to these results are provided in the Appendix.

where δ is at the $(N+1)^{th}$ entry.
Note that the boundary conditions are independent of the nodal unknowns.

We are now at the stage to write (2.14) in the form of a matrix vector equation

$$\hat{M}^N \begin{bmatrix} \dot{\alpha}_1 \\ \vdots \\ \alpha_{N+1} \\ \beta_1 \\ \vdots \\ \beta_N \end{bmatrix} = -J(\alpha(t),\beta(t)) - \underbrace{\begin{bmatrix} \mu S_w^N & \kappa C^N \\ 0 & cS_T^N \end{bmatrix}}_{\hat{S^N}} \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_{N+1} \\ \beta_1 \\ \vdots \\ \beta_N \end{bmatrix} + \begin{bmatrix} 0 \\ F^N(t) \end{bmatrix} + \mu\vec{\delta} \ .$$

In the following, we denote the first matrix on the right hand side by $\hat{S}^N$. Since the mass matrix is known to be invertible, we can apply $\left[\hat{M}^N\right]^{-1}$ to the above equation and obtain

$$\begin{bmatrix} \dot{\alpha(t)} \\ \beta(t) \end{bmatrix} = \left[\hat{M}^N\right]^{-1} \left(-J(\alpha(t),\beta(t)) - \left[\hat{S}^N\right] \begin{bmatrix} \alpha(t) \\ \beta(t) \end{bmatrix} + \begin{bmatrix} 0 \\ F^N(t) \end{bmatrix} + \mu\vec{\delta} \right) \ .$$

This is a first order, nonlinear, inhomogeneous ordinary differential equation for the vector valued function $[\alpha(t),\beta(t)]^\mathsf{T}$. Having obtained the discretized set of ODE's for the vector components we now turn our focus to obtain the initial conditions in a sense of a weak form.

Initial Conditions Since the initial conditions are in $L^2([0,1])$ they need to be projected onto the approximating space consisting of piecewise linear basis functions. The projection is as above performed using the L^2 scalar product. Recall, that we use the basis functions for the velocity $\phi_j \in V_w^N \subseteq H_L^1([0,1])$ and for the temperature $\psi_l \in V_T^N \subseteq H_0^1([0,1])$. Hence, the scalar product is well defined. We first project the initial velocity (2.5) onto V_w^N. Hence, the approximation of the initial condition for the velocity is given by

$$< w^N(0,x), \phi_j(x) > \approx < w_0(x), \phi_j(x) > \ ,$$

where $j = 1,\ldots,N+1$. This further evaluates to

$$\int_0^1 w^N(0,x)\phi_j(x)dx \approx \int_0^1 w_0(x)\phi_j(x)dx \ .$$

Substituting $w^N(0,x)$ by its expansion $w^N(0,x) = \sum_{i=1}^{N+1} \alpha_i(0)\phi_i(x)$ yields

$$\sum_{i=1}^{N+1} \alpha_i(0) \int_0^1 \phi_i(x)\phi_j(x)dx \approx \int_0^1 w_0(x)\phi_j(x)dx \ .$$

Here one should recognize the integral terms which define the mass matrix M_w^N. Observe that the mass matrix is invertible. A useful notation is

$$\alpha(0) = \begin{bmatrix} \alpha_1(0) \\ \vdots \\ \alpha_{N+1}(0) \end{bmatrix} , \qquad \hat{w_0} = \begin{bmatrix} \int_0^1 w_0(x)\phi_1(x) \\ \vdots \\ \int_0^1 w_0(x)\phi_{N+1}(x) \end{bmatrix} .$$

The initial conditions for the velocity nodes $\alpha(\cdot)$ are recovered as

$$\alpha(0) = \left[M_w^N\right]^{-1} \hat{w_0} .$$

We proceed similarly with the initial temperature field. This time we project T_0 on the space V_T^N. It follows that

$$\int_0^1 T^N(0,x)\psi_l(x)dx \approx \int_0^1 T_0(x)\psi_l(x)dx \qquad \forall l = 1, \ldots, N .$$

To simplify notation, define

$$\beta(0) = \begin{bmatrix} \beta_1(0) \\ \vdots \\ \beta_N(0) \end{bmatrix} , \qquad \hat{T_0} = \begin{bmatrix} \int_0^1 T_0(x)\psi_1(x) \\ \vdots \\ \int_0^1 T_0(x)\psi_N(x) \end{bmatrix} .$$

The initial conditions for the temperature of the FE model are

$$\beta(0) = \left[M_T^N\right]^{-1} \hat{T_0} .$$

Thus, in order to get an approximate solution of the coupled Burgers equation one must solve the system of ordinary differential equations given by

$$\begin{bmatrix} \dot{\alpha(t)} \\ \beta(t) \end{bmatrix} = \left[\hat{M^N}\right]^{-1} \left(-J(\alpha(t), \beta(t)) - \left[\hat{S^N}\right] \begin{bmatrix} \alpha(t) \\ \beta(t) \end{bmatrix} + \begin{bmatrix} 0 \\ F^N(t) \end{bmatrix} + \mu\vec{\delta} \right) \tag{2.16}$$

with initial conditions

$$\begin{bmatrix} \alpha(0) \\ \beta(0) \end{bmatrix} = \left[\hat{M^N}\right]^{-1} \begin{bmatrix} \hat{w_0} \\ \hat{T_0} \end{bmatrix} . \tag{2.17}$$

Note that α and β are vector valued functions of time. The system consists of $2N+1$ nodal unknowns and has $2N+1$ initial conditions. Additionally, there are boundary conditions imposed for both dependent variables. Thus, the ODE is well posed. The above system is obtained by approximating the velocity and temperature in a suitable space. A noteworthy property is that the initial conditions are approximated, although they are known exactly. However, the finite dimensional FE spaces do not contain enough information to represent the L^2 functions exactly.

To address issues in a computational implementation, let us highlight that the nonlinearities impose a vector valued function $J(\alpha, \beta)$ of the time dependent unknowns. This has to be evaluated in every time step of the solution routine. If one can simplify the computation of the nonlinearity, one is likely to speed up the solution of the ODE system. A promising approach for this issue is the Group Finite Element method that is discussed in the next section.

2.5 Group Finite Elements

Motivated by [Fle82, chap.3.1] and previous work from Smith [Smi97], Pugh [Pug95] and Nguyen [Ngu01] a conservation form of the Burgers equation (2.1) or the so called Group Finite Elements (GFE) is considered. Smith [Smi97, p.54] concludes that the GFE [5] yields more realistic results for approximate solutions of Burgers equation. Further, improved computational efficiency and only a slight worse error than the regular FEM was observed. Thus, the GFE provides a "better" approximation technique due to [Smi97, p.30]. However, the L^2 error was computed absolutely and not relatively. Pugh [Pug95, p.32] observed similar results. In some cases the FEM produced unrealistic solutions of the Burgers equation that increased exponentially in time. Pugh's data confirms Smith' work as far as the computational efficiency of the GFE goes. These results motivated our work to study and apply the GFE to the coupled Burgers problem and investigate speed and stability of the implemented method. At this point, it is not known whether the coupled Burgers equation can take advantage of the grouping in the first equation. The numerical results are presented in the next section.
Recall that the following identity holds for the nonlinear term of Burgers equation:

$$w(t,x)w_x(t,x) = \frac{1}{2}(w^2(t,x))_x \ .$$

Hence, the coupled Burgers equation (2.1)-(2.2) in conservation form is given by

$$w_t(t,x) + \frac{1}{2}(w^2(t,x))_x = \mu w_{xx}(t,x) - \kappa T(t,x) \ , \tag{2.18}$$

$$T_t(t,x) + w(t,x)T_x(t,x) = cT_{xx}(t,x) + f(t,x) \ . \tag{2.19}$$

[5] Named Galerkin/Conservation Method herein

The vector formulation can be written as

$$\frac{d}{dt}\begin{bmatrix} w \\ T \end{bmatrix} = -\frac{d}{dx}\begin{bmatrix} \frac{1}{2}w^2 \\ 0 \end{bmatrix} - w\frac{d}{dx}\begin{bmatrix} 0 \\ T \end{bmatrix} + \frac{d^2}{dx^2}\begin{bmatrix} \mu w \\ cT \end{bmatrix} - \begin{bmatrix} \kappa T \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ f \end{bmatrix} \tag{2.20}$$

where all functions are functions of time t and space x. We briefly outline the main steps that lead to the discretization, for more details we refer the reader to the previous sections. The approximations to w, T are denoted by $w^N \in V_w^N$ and $T^N \in V_T^N$. Consequently, the Galerkin ansatz for the coupled Burgers equation in grouped form produces the equation

$$\begin{aligned} < \begin{bmatrix} (w^N)_t \\ (T^N)_t \end{bmatrix}, \begin{bmatrix} \phi \\ \psi \end{bmatrix} > = - < \begin{bmatrix} \frac{1}{2}[(w^N)^2]_x \\ w^N (T^N)_x \end{bmatrix}, \begin{bmatrix} \phi \\ \psi \end{bmatrix} > + < \begin{bmatrix} \mu (w^N)_{xx} \\ c(T^N)_{xx} \end{bmatrix}, \begin{bmatrix} \phi \\ \psi \end{bmatrix} > \\ - < \begin{bmatrix} \kappa T^N \\ 0 \end{bmatrix}, \begin{bmatrix} \phi \\ \psi \end{bmatrix} > + < \begin{bmatrix} 0 \\ f \end{bmatrix}, \begin{bmatrix} \phi \\ \psi \end{bmatrix} > \qquad \forall [\phi \ \psi]^{\mathsf{T}} \in V_w^N \times V_T^N \ . \end{aligned}$$

Evaluating the L^2 inner products yields

$$\begin{aligned} & \int_0^1 \left[[w^N(t,x)]_t \phi(x) + [T^N(t,x)]_t \psi(x) \right] dx \\ & = -\int_0^1 \left[\frac{1}{2}[(w^N(t,x))^2]_x \phi(x) + w^N(t,x)(T^N)_x(t,x)\psi(x) \right] dx - \int_0^1 \mu (w^N)_x(t,x)\phi'(x)dx + \mu\delta\phi(1) \\ & - \int_0^1 c(T^N)_x(t,x)\psi'(x)dx - \int_0^1 \kappa T^N(t,x)\phi(x)dx + \int_0^1 f(t,x)\psi(x)dx \qquad \forall [\phi \ \psi]^{\mathsf{T}} \in V_w^N \times V_T^N \ . \end{aligned} \tag{2.21}$$

One should compare this result to (2.13). It is apparent that the sole difference is the quadratic occurrence of w. The grouped variable should be represented in the same space as the standard representation (2.8). Hence, we postulate that the approximate solution $[(w^N)^2]_x$ of $(w^2)_x$ can be expanded in the finite element basis function space V_w^N as

$$w^2(t,x) \approx (w^N(t,x))^2 = \sum_{i=1}^{N+1} \alpha_i^2(t)\phi_i(x) \ .$$

To justify this assumption one notes that the standard approximation (2.8) should interpolate the grouped variable at the space nodes x_n for $n = 1, \ldots, N+1$. This results in

$$(w^N(t,x_n))^2 = \left(\sum_{i=1}^{N+1} \alpha_i(t)\phi_i(x_n) \right)^2 \ .$$

Recall, that the the basis functions are hat functions with the property that $\phi_i(x_n) = \delta_{in}$. Thus, the above equation simplifies to

$$(w^N(t, x_n))^2 = \sum_{i=1}^{N+1} \alpha_i^2(t)\phi_i(x_n) \ .$$

So far, the function value of the standard approximation squared matches the value of the grouped variable, but only at the nodes. Between the nodes, the same linear behavior of the grouped variable is desired. Thus, we extend it linearly (instead of quadratically) to

$$(w^N(t, x))^2 = \sum_{i=1}^{N+1} \alpha_i^2(t)\phi_i(x)$$

by interpolation between the nodes. One advantage of this approach is that the spacial derivative can be calculated without additional work as

$$[(w^N(t, x))^2]_x = \sum_{i=1}^{N+1} \alpha_i^2(t)\phi_i'(x) \ .$$

We substitute the expansions (2.8), (2.9) and the test functions into the approximating weak form. It turns out, that the only term to be substituted in is

$$\int_0^1 \frac{1}{2}[(w^N(t, x))^2]_x\phi_j(x)dx = \frac{1}{2}\sum_{i=1}^{N+1}\left(\int_0^1 \alpha_i^2(t)\phi_i'(x)\phi_j(x)dx\right) \qquad \forall j = 1, \ldots, N+1 \ .$$

One should observe that only the nonlinear function $J(\alpha(t), \beta(t))$ changes when formulating a matrix-vector scheme.

The remainder of this section deals with computing the nonlinearity. In particular, observe that

$$\frac{1}{2}\sum_{i=1}^{N+1}\left(\int_0^1 \alpha_i^2(t)\phi_i'(x)\phi_j(x)dx\right) = \frac{1}{2}\sum_{i=1}^{N+1}\left(\int_0^1 \phi_j(x)\phi_i'(x)dx\right)\alpha_i^2(t) \ .$$

Then, define the matrix A componentwise as

$$(A_{ji})_{j,i=1,\ldots,N+1} = \int_0^1 \phi_j(x)\phi_i'(x)dx \ . \tag{2.22}$$

As with the mass and stiffness matrix an explicit result is available. One obtains

$$A = \begin{bmatrix} 0 & \frac{1}{2} & 0 & & \\ -\frac{1}{2} & 0 & \frac{1}{2} & & \\ & \ddots & \ddots & \ddots & \\ & & -\frac{1}{2} & 0 & \frac{1}{2} \\ & & & -\frac{1}{2} & \frac{1}{2} \end{bmatrix} \ . \tag{2.23}$$

Thus, computing the nonlinearity requires a single multiplication of the matrix A with another vector. This certainly is an advantage over the FEM . Therein, the nonlinear vector J requires comparably more computational effort. One obtains

$$\frac{1}{2}\sum_{i=1}^{N+1}\alpha_i^2(t)\int_0^1\phi_i'(x)\phi_j(x)dx=\frac{1}{2}A\begin{bmatrix}\alpha_1^2(t)\\ \vdots\\ \alpha_{N+1}^2(t)\end{bmatrix}=\frac{1}{2}\begin{bmatrix}\frac{1}{2}\alpha_2^2\\ -\frac{1}{2}\alpha_1^2+\frac{1}{2}\alpha_3^2\\ \vdots\\ -\frac{1}{2}\alpha_{N-1}^2+\frac{1}{2}\alpha_{N+1}^2\\ -\frac{1}{2}\alpha_N^2+\frac{1}{2}\alpha_{N+1}^2\end{bmatrix}_{(N+1)\times 1}.$$

Finally, it follows that the nonlinear term for the GFE is

$$J_{GFE}(\alpha(t),\beta(t))=\begin{bmatrix}\frac{1}{2}A\alpha^2(t)\\ J(\alpha(t),\beta(t))(N+2:2N+1)\end{bmatrix}.$$

Remark Grouping the nonlinearity only affected Burgers equation of the coupled problem. Hence, the entries of J between $N+2$ and $2N+1$ are the same as in the standard FEM. However, the first part is easier to compute. This can be a reason for the improvements in computational time that Smith[Smi97] and Pugh[Pug95] observed. Recall that the nonlinearity has to be evaluated at every time step. In addition, note that neither the initial conditions nor the linear parts have changed due to the reformulation in grouped form.

Discretizing the coupled Burgers equation with a group FEM results in the set of $2N+1$ ordinary differential equations

$$\begin{bmatrix}\dot{\alpha(t)}\\ \beta(t)\end{bmatrix}=\left[\hat{M^N}\right]^{-1}\left(-J_{GFE}(\alpha(t),\beta(t))-\left[\hat{S^N}\right]\begin{bmatrix}\alpha(t)\\ \beta(t)\end{bmatrix}+\begin{bmatrix}0\\ F^N(t)\end{bmatrix}+\mu\vec{\delta}\right) \tag{2.24}$$

given initial conditions

$$\begin{bmatrix}\alpha(0)\\ \beta(0)\end{bmatrix}=\left[\hat{M^N}\right]^{-1}\begin{bmatrix}\hat{w_0}\\ \hat{T_0}\end{bmatrix}. \tag{2.25}$$

Having solved for $\alpha(\cdot)$ and $\beta(\cdot)$ we use the expansions (2.8)-(2.9) to get approximate solutions for the coupled Burgers equation in grouped form. We conduct several numerical experiments employing different MATLAB ® ODE solvers. A mesh refinement is performed varying the parameter of interest N to demonstrate convergence of the scheme.

2.6 Numerical Results for GFE and FEM

In this section, numerical results for FEM and GFE method simulations are provided and compared. We investigate behaviors for choices of Reynolds numbers $Re = 60, 120, 240$ and varying initial conditions. Further, the MATLAB ® ODE solvers 45, 23 and 15s are

compared with regards to computational efficiency and accuracy of the computed solution. For the coupled Burgers equation the ODE15s solver appears to be the fastest amongst the MATLAB ® solvers . However, we observed non-convergence in example 1 when solving the FEM system with ODE15s. Indeed, when solving the GFE model with ODE15s, the solution converges and the solver is still faster than the others. Hence, our conclusion based on the provided data is to use the GFE formulation of the coupled Burgers equation and solve it with the ODE15s solver. With respect to global errors, we are satisfied with the solutions when using $N = 64$ internal nodes. All conclusions are made based of the Method of Manufactured Solutions (MMS) which allows us to compare exact solutions to numerically computed solutions. Details of the MMS are presented in the following subsection.

In Table 2.1 some information about the computational resources used throughout the numerical study are provided.

Software	MATLAB ® Version R2010a
Processor	INTEL ® CORE ™ 2 Duo
Processor Speed	2.16 GHz
L2 Cache	4MB
Bus Speed	667 MHz
Machine Epsilon	2.2204e-16

Table 2.1: Computational Environment

2.6.1 Method of Manufactured Solutions

In this section, we assess the accuracy and cost measured in CPU time of the Finite Element Method (FEM) and the Group Finite Element Method (GFE). We provide results for the mesh refinement procedure as a tradeoff between accuracy and computational cost. As commonly done in literature, an analytic benchmark solution for both the temperature and velocity is created and compared to the numerical solutions. This allows one to verify the computer code that solves the PDE system. The interested reader is referred to Roache [Roa04] for an outline of this standard approach.

We begin with a summary of the key ideas behind the MMS. The task of the MMS is to detect errors in coding that effect the computational result obtained from the code. Hence, this method is purely mathematical. It only verifies the computer code whereas a validation of the results, i.e. a physical interpretation, is a completely different task. Therefore, the solution that one chooses to be reproduced by the code can lack any physical meaning. Following the reasoning in [Roa04], it might be preferable for the solution to be purely mathematical. Otherwise one could be inclined to check the code only for physical expectations

but not for purely code based consistency. Note that MMS in this setup is not able to check errors that are made in iterations and cause the code to be slow but correct.

Once one has agreed on a solution, the solution is plugged in to the system and the result is appended as a forcing on the system. If the system has no forcing initially, an artificial forcing needs to be added. The new system is obtained from the old system by adding the analytical forcing.

The concept of MMS asks for a measure of discretization. In our problem, this is chosen to be the mesh size $\frac{1}{N+1}$. The next step is to record the error between the solution of the discretized system (in our case the FEM or GFE) and the manufactured solution. Applying this procedure one should observe convergence of the computed solutions to the exact solution as the measure of discretization is refined.

The system of interest for the Method of Manufactured Solutions is

$$\begin{aligned} w_t(t,x) + w(t,x)w_x(t,x) &= \mu w_{xx} - \kappa T(t,x) + f_1(t,x) \ , & (2.26)\\ T_t(t,x) + w(t,x)T_x(t,x) &= cT_{xx} + f_2(t,x) \ . & (2.27) \end{aligned}$$

The boundary conditions are given

$$\begin{aligned} w(t,0) &= 0 \ , & w_x(t,1) &= 0 \ , \\ T(t,0) &= 0 \ , & T(t,1) &= 0 \end{aligned}$$

and initial conditions are

$$\begin{aligned} w(0,x) &= w_0(x) \ , & &\in L^2([0,1]) \ , \\ T(0,x) &= T_0(x) \ , & &\in L^2([0,1]) \ . \end{aligned}$$

The FE discretized system of ordinary differential equations is given as

$$\begin{bmatrix} \dot{\alpha(t)} \\ \beta(t) \end{bmatrix} = \left[\hat{M^N}\right]^{-1} \left(-J(\alpha(t),\beta(t)) - \left[\hat{S^N}\right] \begin{bmatrix} \alpha(t) \\ \beta(t) \end{bmatrix} + \begin{bmatrix} F_1^N(t) \\ F_2^N(t) \end{bmatrix} + \mu\vec{\delta} \right) \ ,$$

$$\begin{bmatrix} \alpha(0) \\ \beta(0) \end{bmatrix} = \left[\hat{M^N}\right]^{-1} \begin{bmatrix} \hat{w_0} \\ \hat{T_0} \end{bmatrix} \ .$$

When substituting J by J_{GFE} this system corresponds to the GFE method. Rearranging terms in (2.26)-(2.27) yields expressions for f_1 and f_2:

$$\begin{aligned} f_1(t,x) &= w_t(t,x) + w(t,x)w_x(t,x) - \mu w_{xx}(t,x) + \kappa T(t,x) \ , \\ f_2(t,x) &= T_t(t,x) + w(t,x)T_x(t,x) - cT_{xx}(t,x) \ . \end{aligned}$$

Two steps have to be performed to set up the MMF for the coupled Burgers equation. First, the desired manufactured solutions in each example are substituted into (2.26)-(2.27). That way, the first and second equation define the forcing functions f_1 and f_2. Secondly, one needs to add f_1 and f_2 as analytical forcing functions to the system.

The exact solutions were chosen to satisfy the boundary conditions. Consistent initial conditions were derived as a limit of time t to 0. Finally, the FEM and and GFE models are executed with the obtained initial conditions. It is apparent that the solution produced by the code should approximate the manufactured solution when N is increased. Errors and CPU times are compared for several values of internal nodes N and varying Reynolds numbers.

Error calculation Within the procedure of comparing exact and calculated solutions the global L^2 norm is used for error calculations. For a given function $f \in L^2([0,T]; L^2((0,1)))$ it can be computed as

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

Since the considered system is a coupled system, we are interested in the accumulated error of both w and T. This leads us to consider the relative and global error as

$$Err^{rel}(N) = \frac{\|w - w^N\|_{L^2} + \|T - T^N\|_{L^2}}{\|w\|_{L^2} + \|T\|_{L^2}} \tag{2.28}$$

The integrals are calculated with a three point Gauss Quadrature rule. This is known to be exact for polynomial functions up to degree five. For better comparison, the space integral is evaluated at Gauss points independent of N. We set up 33 finite elements within $[0,1]$ where three Gauss points are defined within each element. The time interval $[0,t_f]$ is subdivided into 200 time elements.

2.6.2 Simulations

In the following two examples, the Group Finite Element method (GFE) is compared with the standard Finite Element Method (FEM) for Reynolds numbers $Re = 60, 120, 240$ and internal FE nodes of $N = 8, 16, 32, 64$[6]. We are interested in the results for the mesh refinement procedure which gives insight into the convergence behavior.

The standard ODE45 solver was chosen in MATLAB ® initially. However, simulations show

[6]The resulting coupled system has $2N + 1$ unknowns for w and T are computed at the same time

that ODE23 saves computational cost while maintaining the same accuracy as ODE45. The stiff solver ODE15s is about 20 to 50 times faster for computationally complex cases. However, we observed non-stability of the ODE15s in Example 1.

The errors are measured relatively in the global L^2 norm. The error plots show absolute errors from the exact solution. The CPU times are measured with the MATLAB ® tic-toc command and are thus given in seconds. The tic-toc tool measures the CPU time needed to execute all code between tic and toc. The time values in the table cover the full solution, that is the evaluation of the matrices as well as the actual solution of the ODE system. Thus, the CPU times provide a measure on how long the computation of the full model takes overall. One can directly compare the effect of the different solvers, for the preprocessing effort does not change when changing the solver. Consequently, the time difference is purely caused by the solvers themselves.

Moreover, writing the Burgers equation in grouped form does not affect the linear parts and initial conditions. The only change occurs in the nonlinearity which is crucial for the ODE solution routine. The time difference between FEM and GFE for same Reynolds numbers and external parameters is solely due to the evaluation of the nonlinear term.

Example 1
The parameters for this problem are: $c = 0.01, \kappa = 1.0$. The final time is chosen to be $t_f = 15s$. The velocity and temperature field are taken to be

$$\begin{aligned} w(t,x) &= \exp(-t)(x - \frac{x^2}{2}) \ , \\ T(t,x) &= x(1-x) \ . \end{aligned}$$

Observe that the temperature is time invariant and that both w and T satisfy the boundary conditions. This is all the Method of Manufactured Solutions requires, no matter how physically intuitive the desired solutions are. Accordingly, we have

$$\begin{aligned} w_0(x) &= (x - \frac{x^2}{2}) \ , \\ T_0(x) &= x(1-x) \ . \end{aligned}$$

The analytical forcing functions are calculated and given by

$$\begin{aligned} f_1(t,x) &= \exp(-t)\left(-x + \frac{x^2}{2} + \exp(-t)(x - \frac{x^2}{2})(1-x) - \mu\right) + \kappa x(1-x) \ , \\ f_2(t,x) &= \exp(-t)(x - \frac{x^2}{2})(1-2x) + 2c \ . \end{aligned}$$

Results of the simulations are provided in Tables 2.2, 2.3 and 2.4.

Internal Nodes	Re	CPU Time FEM	CPU Time GFE	Err^{rel} FEM	Err^{rel} GFE
N=8	60	2.92	2.82	0.1668	0.2985
	120	1.75	2.01	0.5079	0.7540
	240	1.88	2.09	0.9464	1.0750
N=16	60	16.30	16.48	0.0438	0.0907
	120	9.54	10.15	0.1576	0.2455
	240	9.87	9.98	0.3626	0.4754
N=32	60	120.16	119.20	0.0114	0.0372
	120	71.20	71.76	0.0416	0.0724
	240	71.23	70.53	0.1028	0.1459
N=64	60	922.87	927.99	0.0030	0.0263
	120	551.53	555.86	0.0107	0.0305
	240	548.72	558.08	0.0265	0.0420

Table 2.2: Example 1: Results with ODE45 Solver for FEM and GFE, $t_f = 15s$

Internal Nodes	Re	CPU Time FEM	CPU Time GFE	Err^{rel} FEM	Err^{rel} GFE
N=8	60	2.11	2.32	0.1648	0.3067
	120	1.64	1.69	0.6164	0.8024
	240	1.95	1.91	1.0519	1.1029
N=16	60	11.04	10.75	0.0439	0.0882
	120	6.73	6.80	0.1585	0.2504
	240	6.54	6.65	0.3609	0.4704
N=32	60	74.17	76.81	0.0114	0.0367
	120	45.51	44.88	0.0421	0.0731
	240	45.03	49.17	0.1032	0.1463
N=64	60	568.54	567.33	0.0032	0.0264
	120	341.71	339.78	0.0100	0.0309
	240	342.81	341.23	0.0267	0.0424

Table 2.3: Example 1: Results with ODE23 Solver for FEM and GFE, $t_f = 15s$

We shall discuss the numerical results successively. The FEM and the GFE converged when using the solvers ODE45 and ODE23. The stiff solver ODE15s converged with the GFE method but did not converge with the FEM method, see Table 2.4. This result is similar to the work of Pugh[Pug95, p.32], where the Burgers equation has been solved with the MATLAB ® ODE45 solver. Hence, the GFE method seems to have advantages in terms of stability of solutions. This also confirms results from [Smi97, p.54] where the GFE provided more realistic results than the FEM.

For all three solvers, there was no significant difference in computational time between the FEM and GFE as observed by Pugh [Pug95, p.42] and Smith[Smi97, p.54]. However, FEM

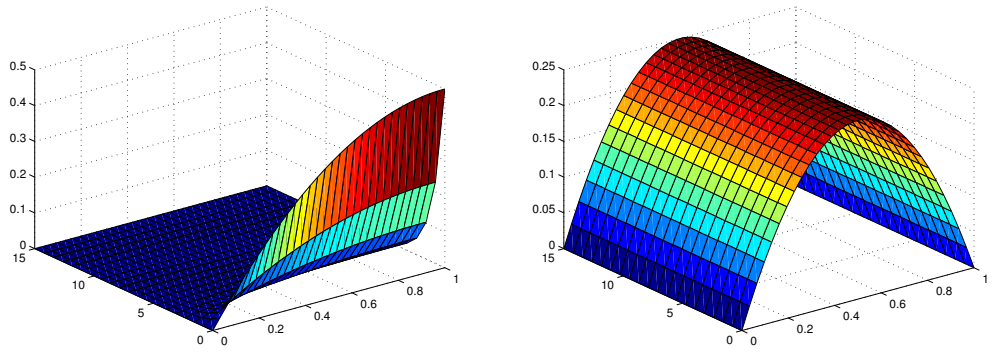
Internal Nodes	Re	CPU Time FEM	CPU Time GFE	Err^{rel}FEM	Err^{rel}GFE
N=8	60	1.50	1.75	0.1581	0.2374
	120	1.61	1.83	0.5097	0.6664
	240	1.94	2.21	0.8338	0.9575
N=16	60	2.33	2.57	0.0433	0.0880
	120	2.40	2.67	0.1518	0.2103
	240	2.70	3.07	0.3534	0.4225
N=32	60	6.57	6.82	0.0128	0.0425
	120	6.09	7.18	0.1114	0.0581
	240	6.18	9.09	0.2349	0.1552
N=64	60	17.84	19.84	0.0322	0.0314
	120	18.02	20.71	0.0954	0.0325
	240	18.94	21.15	0.1902	0.0325

Table 2.4: Example 1: Results with ODE15s Solver for FEM and GFE, $t_f = 15s$

provides better results in terms of accuracy if it converged. For the solvers ODE45 and ODE23 errors differed up to a factor of 10, see Tables 2.2 and 2.3. Throughout, one can observe that the higher the Reynolds number, the higher the error in the solution. This is due to "shocks" (i.e. steep gradients) in the fluid that occur with increasing Reynolds numbers.

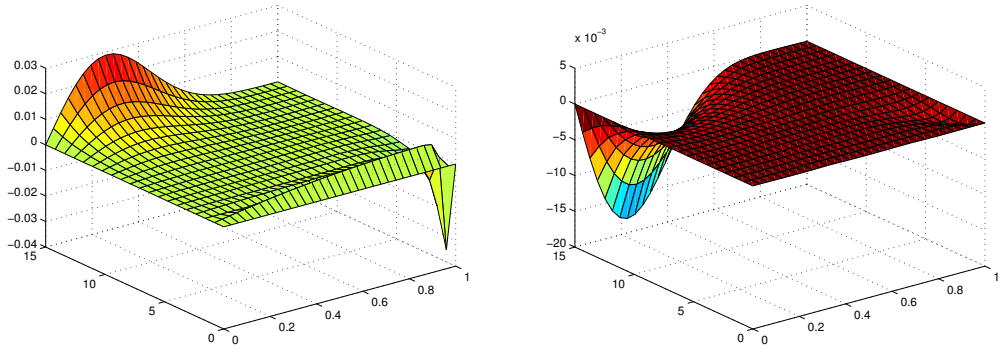
In two interesting cases, plots are provided. First, the manufactured solutions are plotted in Figure 2.1. Then error plots for the temperature as well as the velocity are plotted in Figures 2.2 through 2.5. We obtained the plots from solutions with the ODE15s solver for combinations of Reynolds numbers and numbers of internal nodes $Re = 240, N = 64$ and $Re = 60, N = 32$. We further investigate the locations where the FEM and the GFE propagate errors.

One should pay attention to the scaling of the plots. The FEM propagates the error to the final time and is not converging. The solution should be almost zero at the final time. However, as far as the velocity goes, the error is up to 0.2 for a Reynolds number of $Re = 240$, see Figures 2.2 through 2.3 and Table 2.4. Observe that the FEM is highly accurate within the first 5 seconds. In comparison, the GFE is stable and has only small errors at the final time. However, the GFE is not as exact initially, but as time increases the errors are reduced. This seems to be an advantage over the FEM method.



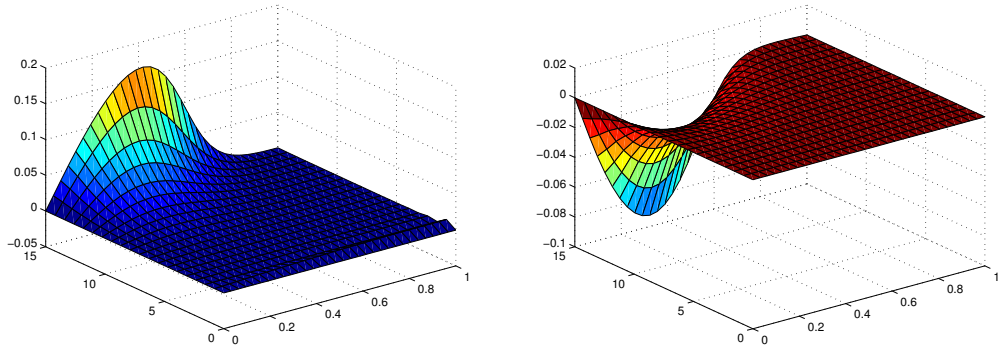
(a) Manufactured Velocity (b) Manufactured Temperature

Figure 2.1: Exact Solutions to Example 1



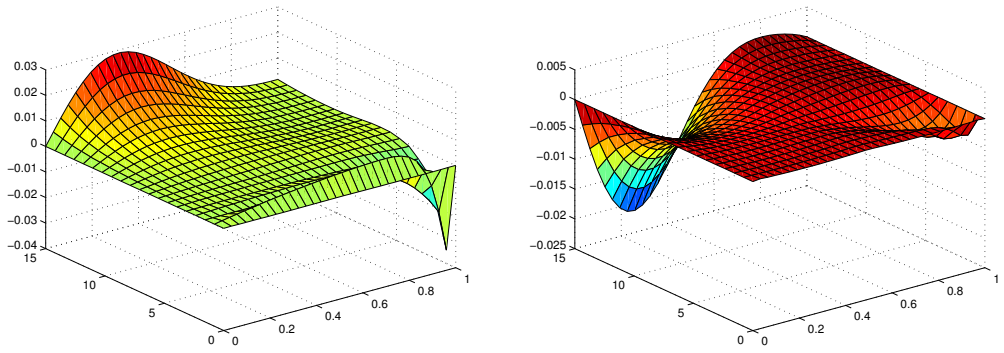
(a) Velocity (b) Temperature

Figure 2.2: GFE Errors with ODE15s for $N = 64, Re = 240$



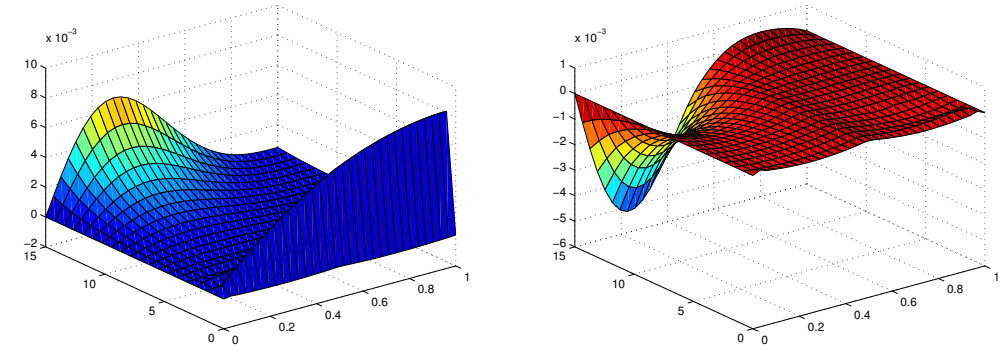
(a) Velocity (b) Temperature

Figure 2.3: FEM Errors with ODE15s for $N = 64, Re = 240$



(a) Velocity (b) Temperature

Figure 2.4: GFE Errors with ODE15s for $N = 32, Re = 60$



(a) Velocity (b) Temperature

Figure 2.5: FEM Errors with ODE15s for $N = 32, Re = 60$

With a lower Reynolds number $Re = 60$ and a mesh of only $N = 32$ internal nodes the FEM is more exact. Again, one should pay attention to the scaling of the plots. This time the FEM simulation starts out with a higher initial error as the GFE in the previous observation, see Figures 2.4 and 2.5. However, the error decays in time when using the FEM. At this point, GFE produces errors on the right part of the interval, too.

All calculations within this example have been executed by employing the ODE15s solver. We are interested in this solver due to computational advantages. Throughout, note that errors in the velocity w are always one dimension bigger than errors in the temperature field. This is due to the complexity of the calculations of the velocity that arises from the Burgers equation, namely the nonlinearity. Moreover, observe that errors at the final time occur in the interval $[0, 0.5]$, at the left side of the interval. Another interesting feature is the

smoothness of the error plots. The manufactured solutions are polynomial in x. Thus, the Gauss integration is exact for the forcing functions f_1, f_2 and the initial conditions w_0, T_0. In this case, the integration of the forcing function against the test function does not introduce new errors. This results in the fact that the higher the mesh size, the more we can see the accuracy in the initial condition. For reference, compare the cases $N = 32$ and $N = 64$.

Example 2
The parameters are chosen to be: $c = 0.01, \kappa = 1.0, \varepsilon = 1/60$. The final time is again $t_f = 15s$. The velocity field and the temperature field shall be

$$\begin{aligned} w(t,x) &= \exp(-\varepsilon t)(1-x)\sin(\pi x) \ , \\ T(t,x) &= \exp(-\varepsilon t)\sin(\pi x) \ . \end{aligned}$$

This time both functions depend on the time and space variable and again satisfy the boundary conditions. From the limit as t goes to 0, the initial conditions are derived as

$$\begin{aligned} w_0(x) &= (1-x)\sin(\pi x) \ , \\ T_0(x) &= \sin(\pi x) \ . \end{aligned}$$

The forcing functions for the MMS are given by

$$\begin{aligned} f_1(t,x) &= \exp(-\varepsilon t)[-\varepsilon(1-x)\sin(\pi x) + \exp(-\varepsilon t)(1-x)[-\sin(\pi x)^2 + \pi(1-x)^2 \\ &\quad -\mu(-2\pi\cos(\pi x) - \pi^2(1-x)\sin(\pi x)\sin(\pi x)\cos(\pi x) + \kappa\sin(\pi x)] \ , \\ f_2(t,x) &= \exp(-\varepsilon t)\sin(\pi x)(-\varepsilon + \pi\exp(-\varepsilon t)(1-x)\cos(\pi x) + c\pi^2) \ . \end{aligned}$$

The numerical results for the global errors and CPU times are given in Table 2.5 through 2.7.

In this example we obtained good results with 3% global error when using only $N = 8$ internal nodes to create the mesh for the temperature and the velocity, respectively. At this point, both methods converged when using the ODE15s solver, see Table 2.7. For all solvers, the results showed 0.08% relative error when using $N = 64$ elements. The numerical results show once more that the ODE45 (see Table 2.5) and ODE23 (see Table 2.6) solvers are slow compared to the ODE15s solver for this problem. The computational cost is in the same range than in the previous example. Further, CPU times do not differ significantly between the FEM and the GFE. Another noteworthy observation is that the accuracy is almost identical for all three solvers. However, the accuracy is by a factor of 2 higher with the FEM than with the GFE.
In this example we do not have issues with stability and good convergence of the ODE15s solver with the FEM.

The errors are plotted as perturbations around the exact solution. Hence they show absolute errors. Thus, one should pay attention to the scaling of the error plots.

Internal Nodes	Re	CPU Time FEM	CPU Time GFE	Err^{rel} FEM	Err^{rel} GFE
N=8	60	2.66	2.91	0.0312	0.0681
	120	1.78	1.81	0.0359	0.0851
	240	1.59	1.63	0.0348	0.0959
N=16	60	16.49	16.65	0.0091	0.0189
	120	9.77	9.36	0.0098	0.0244
	240	9.47	9.65	0.0100	0.0288
N=32	60	121.62	122.08	0.0024	0.0049
	120	71.75	71.42	0.0025	0.0063
	240	72.95	70.70	0.0029	0.0076
N=64	60	928.11	931.49	0.0006	0.0013
	120	561.21	560.27	0.0007	0.0016
	240	563.72	558.39	0.0008	0.0020

Table 2.5: Example 2: Results with ODE45 Solver for FEM and GFE, $t_f = 15s$

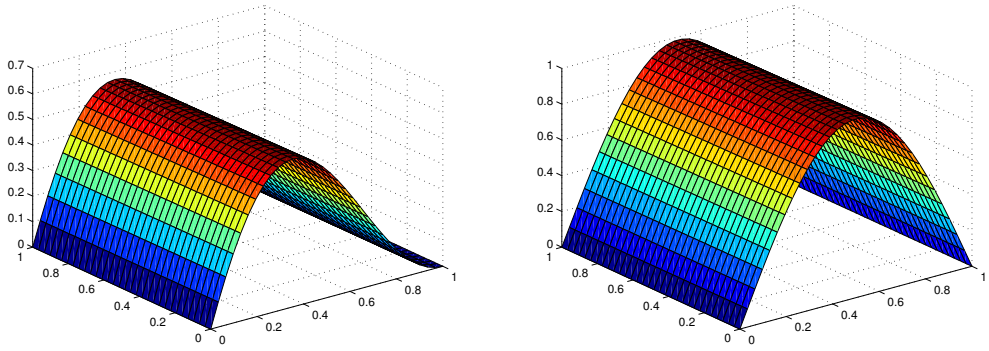
Internal Nodes	Re	CPU Time FEM	CPU Time GFE	Err^{rel} FEM	Err^{rel} GFE
N=8	60	2.15	2.29	0.0316	0.0681
	120	1.58	1.88	0.0379	0.0850
	240	1.54	1.53	0.0359	0.0962
N=16	60	10.75	10.96	0.0093	0.0189
	120	6.42	6.28	0.0104	0.0244
	240	6.34	6.36	0.0100	0.0287
N=32	60	76.15	76.72	0.0024	0.0049
	120	44.89	45.70	0.0025	0.0063
	240	45.22	47.00	0.0030	0.0076
N=64	60	576.28	567.69	0.0006	0.0013
	120	346.02	344.95	0.0007	0.0016
	240	344.30	348.05	0.0008	0.0019

Table 2.6: Example 2: Results with ODE23 Solver for FEM and GFE, $t_f = 15s$

This time we have sinusoidal initial conditions and forcing functions. The sin function is per definition an infinite series of polynomials. However, Gauss integration with three Gauss points in every element is only accurate up to polynomial order five. Therefore, one should observe the non-smoothness of the error functions, see Figures 2.7 through 2.10. This time, numerical errors are made in both the approximating system and the Gauss integration. In the case $N = 64$ both methods provide almost similar results in terms of the error location and error propagation, see Figures 2.9 and 2.10. Note that both methods must match at the boundaries, since the basis functions enforce these conditions.

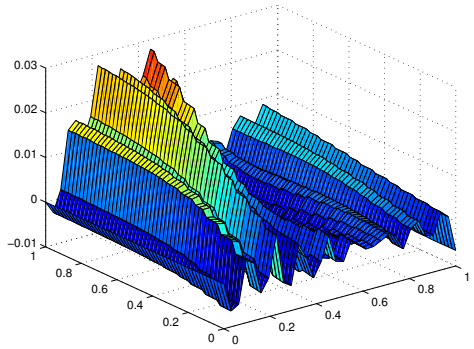
Internal Nodes	Re	CPU Time FEM	CPU Time GFE	Err^{rel} FEM	Err^{rel} GFE
N=8	60	1.55	1.48	0.0303	0.0682
	120	1.47	1.64	0.0300	0.0868
	240	1.45	1.70	0.0304	0.0981
N=16	60	2.28	2.19	0.0088	0.0186
	120	2.21	2.17	0.0091	0.0240
	240	2.16	2.24	0.0098	0.0288
N=32	60	5.28	5.38	0.0024	0.0052
	120	6.43	5.36	0.0025	0.0063
	240	4.77	5.08	0.0029	0.0078
N=64	60	11.67	12.75	0.0008	0.0013
	120	13.16	12.25	0.0008	0.0017
	240	17.78	12.69	0.0009	0.0020

Table 2.7: Example 2: Results with ODE15s Solver for FEM and GFE, $t_f = 15s$

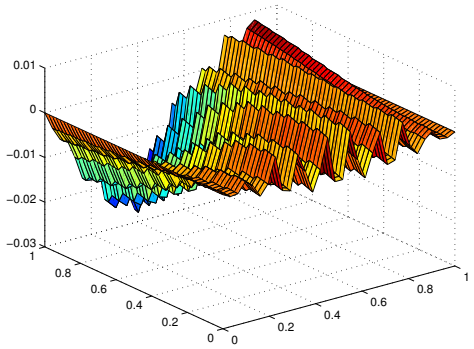


(a) Manufactured Velocity (b) Manufactured Temperature

Figure 2.6: Exact Solutions to Example 1

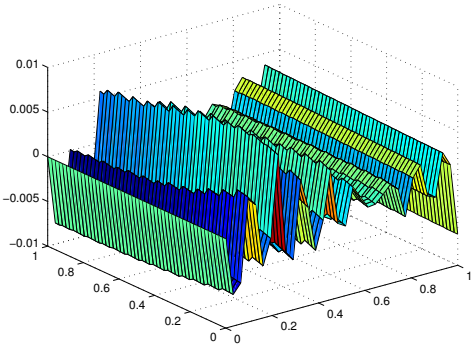


(a) Velocity

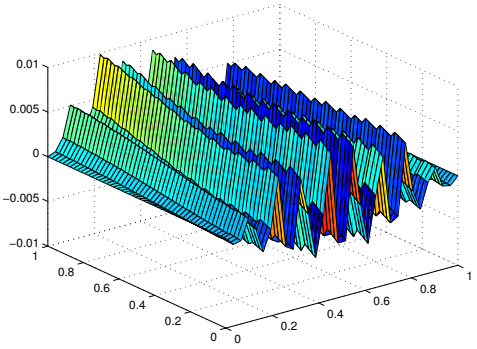


(b) Temperature

Figure 2.7: GFE Errors with ODE15s for $N = 8, Re = 240$

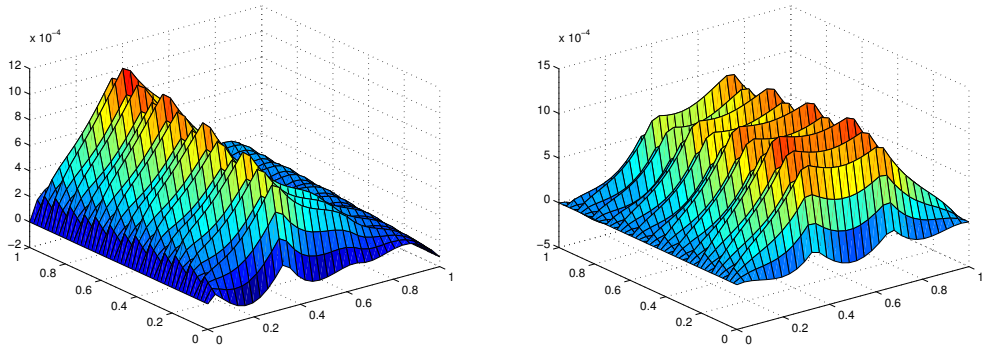


(a) Velocity



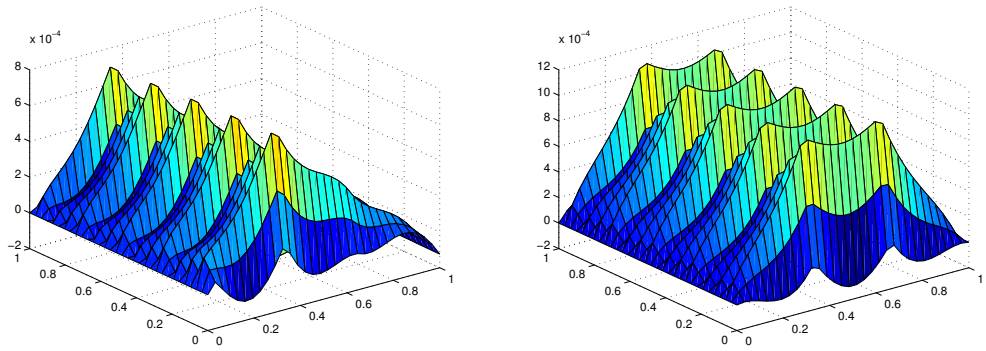
(b) Temperature

Figure 2.8: FEM Errors with ODE15s for $N = 8, Re = 240$



(a) Velocity (b) Temperature

Figure 2.9: GFE Errors with ODE15s for $N = 64, Re = 240$



(a) Velocity (b) Temperature

Figure 2.10: FEM Errors with ODE15s for $N = 64, Re = 240$

Chapter 3

Proper Orthogonal Decomposition

Physical problems often depend on multiple variables, e.g. space, time or densities. The dynamics of those systems are described with partial differential equations (PDE). In order to find solutions various numerical methods can be used that commonly project the governing equations on a finite dimensional subspace. A frequently used technique within this framework is the Finite Element Method. We built the FEM model in Chapter 2 for the coupled Burgers equation. While increasing the dimension of the FE function space, the error can be proven to be within any desired bound. Unfortunately, when doing so, the computational effort increases drastically.

Challenging applications especially in control design and optimization have to deal with large scale systems. One goal is to reduce the simulation time of the system in order to make the model feasible for the design of controllers. Thus, the aim is to find "practical" numerical methods that preserve sufficient accuracy while minimizing the computational cost of the simulation.

Within the second half of the last century, further model reduction techniques have been investigated, making use of the mostly expensive full model solution via FEM. Loéve[Loè55] extracted eigenfunctions out of samplings of data which accelerated the field of model reduction. In this work, we are interested in the Proper Orthogonal Decomposition. The POD method projects the equations onto a low dimensional space of basis functions that carry characteristics of the expected solutions. A crucial difference between the FEM and POD method is that the POD basis is constructed using essential information from the PDE. The FE basis spaces are uncorrelated to the physics of the dynamical system, whereas POD tries to gather a priori knowledge about the evolution of the equation[1]. In many applications

[1]The POD method mostly requires a FEM solution in order to obtain the basis. However, this can be once computed offline. One wishes to amortize the cost of solving the full order model (maybe even for a range of parameters) by running many fast simulations with the new basis later.

one has observed the POD method to be computationally very efficient while still delivering sufficiently precise results.

In their recent work, J.Singler and B.Dickinson [DS10, p.365] showed that grouping the nonlinearity in Burgers equation when doing a POD saves computational effort. The idea stems from the GFE which also proved itself to have advantages over standard FEM. This is also related to the empirical interpolation method which generalizes this to a wider class of nonlinearities[BMNP04]. In this work, we apply the Group POD to the coupled Burgers equation and further demonstrate the advantages with respect to cost for the simulations compared to the GFE.

First, a brief outline of the key ideas behind the POD is given. The main idea of POD is to construct basis functions that capture an "optimal" amount of information from the dynamics of the system. This information is mostly collected using some sort of approximate high-fidelity solution. In a first step, time snapshots of the approximate solution are taken. Subsequently, a correlation matrix of time snapshots is built. The eigenvalues and eigenfunctions of this matrix are used to construct a POD basis. The dimension of the new POD basis d is often chosen by heuristic reasoning. It is desired to be sufficiently smaller than the number of FE basis functions, say $d \ll N$. The obtained POD basis functions are optimal in a sense that they provide the best orthogonal basis to represent the snapshot set[2]. Once the basis is constructed, the system equations for the dependent variables w and T are projected on the POD basis. In a next step, a Galerkin method is applied, where the Burgers equation is written in grouped form. Similar to the FEM, a low dimensional set of discretized ODE's is obtained. The idea is that the reduced order model can be simulated much faster with only a slight loss in accuracy.

Throughout this work, we shall denote nodal functions, basis functions, coefficients, matrices etc. that are used within the POD method with a superscript P to highlight the difference between the POD and FEM models. Moreover, recall that $< \cdot, \cdot >$ denotes the L^2 inner product as defined in Chapter 1.

3.1 POD Basis

The solution of the GFE method is a solution for the nodal unknowns α_i, β_i at several time instances t_i where $i = 1, .., S$. The solution output is often written in matrix form as[3]

[2] "Best" considers the projection on an orthonormal basis and maximizes the coefficients for the basis

[3] This form is the solution output of all MATLAB ® ODE solvers. One needs to provide a time span (conveniently $[t_1, .., t_S]$) and initial conditions $[\alpha_1(0), .., \beta_N(0)]$ and the right hand side, where $[\dot{\alpha}, \dot{\beta}] = rhs(\alpha, \beta)$

$$Y = \begin{bmatrix} \alpha_1(t_1) & \cdots & \alpha_{N+1}(t_1) & \beta_1(t_1) & \cdots & \beta_N(t_1) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \alpha_1(t_S) & \cdots & \alpha_{N+1}(t_S) & \beta_1(t_S) & \cdots & \beta_N(t_S) \end{bmatrix} . \tag{3.1}$$

The columns of Y are time trajectories of the approximate system at fixed points of the space grid. The rows can be viewed as spacial coordinate vectors of the system at an instant of time t_i. In fact, for an understanding of the POD, it is important to view the snapshots as vectors in an $2N+1$ dimensional space. That is, every snapshot can be represented with $2N+1$ basis functions. We frequently refer to the above matrix Y to precisely outline the numerical procedure of computing the POD basis.

The POD method requires snapshots of the dynamical system at multiple times. These snapshots could be obtained by measuring the system at several times. However, the upcoming of numerical simulations led in most cases to the redundancy of physical experiments. Hence, in a mathematical framework where the solution is of course not available a priori, the snapshots are often obtained from the approximate system. Let us emphasize that this really means taking snapshots of w^N and not w. Thus, a POD can only reflect features of the snapshot set. In particular, that is the task of the POD. Consequently, if the snapshots do not cover enough information about the dynamics of the system, the POD basis is likely to not cover the dynamics properly. One should hence consider this fact in the interpretation of solutions.

The GFE model is solved with a time span of length S which automatically produces S snapshots of the approximate system. These can be used to perform a POD. The choice of S can be in a certain range, more shall be said at the end of this section. Snapshot sets are expected to sufficiently represent the dynamics of the PDE. However, to the authors knowledge, there is not yet an effective technique to generate the "best" snapshot set [GPS07, p.1032]. The snapshot sets herein are obtained at equally spaced time instances from $t_1 = 0$ to $t_S = t_f$. Since the system is coupled, some attention has to be spend on how to calculate the POD basis in this case. The snapshots of the temperature and velocity are given as

$$w^N(t_i, x) = \sum_{j=1}^{N+1} \alpha_j(t_i)\phi_j(x) \qquad \text{for } x \in [0,1] \ , \tag{3.2}$$

$$T^N(t_i, x) = \sum_{j=1}^{N} \beta_j(t_i)\psi_j(x) \qquad \text{for } x \in [0,1] \ . \tag{3.3}$$

The sets $\{w^N(t_i,\cdot)\}_{i=1}^S$ and $\{T^N(t_i,\cdot)\}_{i=1}^S$ are called the snapshot sets of the approximate solutions w^N and T^N.

in order to get the solution in this form.

The next step involves the definition of a correlation matrix that captures the information about the dynamics of the system. In particular, two correlation matrices are build for the velocity and temperature, respectively. Although the dependent variables w and T are coupled, the correlation matrices can be calculated independently. This is mainly for two reasons: First, the GFE solution is already a solution of the coupled system. Hence, the solutions of which we take snapshots already contain information about the coupling. Secondly, after expanding the functions w and T in the POD spaces, the projection of the dynamical system onto this basis takes into account the coupling. At this point, the calculation of the POD correlation matrices and basis functions is a purely algebraic procedure.

We define the correlation matrix K_w for the velocity as

$$(K_w)_{ij} = \left(\frac{1}{S} < w^N(t_i, \cdot), w^N(t_j, \cdot) > \right)_{i,j=1,\dots,S}$$

and the correlation matrix K_T for the temperature as

$$(K_T)_{ij} = \left(\frac{1}{S} < T^N(t_i, \cdot), T^N(t_j, \cdot) > \right)_{i,j=1,\dots,S} .$$

Note that K_w can be quickly computed from the solution matrix Y of the FE approximation. To be precise, for $i, j = 1, \dots, S$ one obtains

$$\begin{aligned}(K_w)_{ij} &= \frac{1}{S} < \sum_{k=1}^{N+1} \alpha_k(t_i)\phi_k(x), \sum_{l=1}^{N+1} \alpha_l(t_j)\phi_l(x) > \\ &= \frac{1}{S} \int_0^1 \sum_{k=1}^{N+1} \alpha_k(t_i)\phi_k(x) \sum_{l=1}^{N+1} \alpha_l(t_j)\phi_l(x) = \frac{1}{S} \sum_{k,l=1}^{N+1} \alpha_k(t_i)\alpha_l(t_j) \int_0^1 \phi_k(x)\phi_l(x)dx \\ &= \frac{1}{S}\alpha^T(t_i)\ M_w^N\ \alpha(t_j) = \frac{1}{S} Y(t_i, 1:N+1)\ M_w^N\ Y^{\mathsf{T}}(t_j, 1:N+1) .\end{aligned}$$

Similar calculations for K_T yield

$$(K_T)_{ij} = \frac{1}{S} Y(t_i, N+2:2N+1)\ M_T^N\ Y^{\mathsf{T}}(t_j, N+2:2N+1) .$$

The normalized eigenvectors and eigenvalues of K_w and K_T are denoted λ^w, λ^T and v^w, v^T, respectively.

In the following, the POD basis functions for the velocity w and temperature T are defined. Note that both POD functions are calculated from different eigenvectors and the respective snapshots. The k^{th} POD basis functions for the flow w and temperature T are given by

$$\phi_k^P(x) = \frac{1}{\sqrt{S}\sqrt{\lambda_k^w}} \sum_{i=1}^{S} [v_k^w]_i\, w^N(t_i, x) \tag{3.4}$$

and

$$\psi_k^P(x) = \frac{1}{\sqrt{S}\sqrt{\lambda_k^T}} \sum_{i=1}^{S} \left[v_k^T\right]_i T^N(t_i, x) \ , \tag{3.5}$$

respectively.

As mentioned earlier, some qualitative observations about the construction of the POD basis can serve to find a "good" number of snapshots S. Recall that in the FEM the spacial domain is subdivided into $N+1$ subintervals. The solution of the discretized ODE system is only obtained at these nodal values $x_i,\ i = 1, \ldots, N+2$ for given times $t_i,\ i = 1, \ldots, S$. In between the nodes the solution is expanded linearly. Thus, the approximate solution at one instant in time (or a snapshot) can be viewed as a vector in the $2N+1$ dimensional product space $V_w^N \times V_T^N$. The above formulas for the POD basis are nothing but a linear combination of S snapshots for w^N and T^N adding physical meaning to the basis.

If all snapshots are linearly independent, then employing basic linear algebra implies $S \leq N+1$, for more snapshots would be linearly dependent. However, this assumption cannot be made in general. In general, a convergence analysis of the POD basis functions has to be performed using S as the measure of refinement.

We now turn our focus to simplifying the basis functions. Using the expansions of the approximate solutions in terms of the FE basis and coefficient functions yields

$$\phi_k^P(x) = \frac{1}{\sqrt{S}\sqrt{\lambda_k^w}} \sum_{i=1}^{S} \left[v_k^w\right]_i \sum_{j=1}^{N+1} \alpha_j(t_i)\phi_j(x) = \sum_{j=1}^{N+1} \frac{1}{\sqrt{S}\sqrt{\lambda_k^w}} \sum_{i=1}^{S} \left[v_k^w\right]_i \alpha_j(t_i)\phi_j(x) \ ,$$

$$\psi_k^P(x) = \frac{1}{\sqrt{S}\sqrt{\lambda_k^T}} \sum_{i=1}^{S} \left[v_k^T\right]_i \sum_{j=1}^{N} \beta_j(t_i)\psi_j(x) = \sum_{j=1}^{N} \frac{1}{\sqrt{S}\sqrt{\lambda_k^T}} \sum_{i=1}^{S} \left[v_k^T\right]_i \beta_j(t_i)\psi_j(x) \ .$$

For brevity, define for $j = 1, \ldots, N+1$

$$\gamma_{jk} = \frac{1}{\sqrt{S}\sqrt{\lambda_k^w}} \sum_{i=1}^{S} \left[v_k^w\right]_i \alpha_j(t_i) = \frac{1}{\sqrt{S}\sqrt{\lambda_k^w}} \left[v_k^w\right]^{\mathsf{T}} Y(:, j) \ , \tag{3.6}$$

and equivalently for $j = N+2, \ldots, 2N+1$

$$\gamma_{jk} = \frac{1}{\sqrt{S}\sqrt{\lambda_k^T}} \sum_{i=1}^{S} \left[v_k^T\right]_i \beta_{j-(N+1)}(t_i) = \frac{1}{\sqrt{S}\sqrt{\lambda_k^T}} \left[v_k^T\right]^{\mathsf{T}} Y(:, j) \ . \tag{3.7}$$

Note, that the superscript T indicates that the eigenvalues and eigenvectors are obtained from the temperature correlation matrix. The superscript T denotes the transpose of a matrix or a vector.

Remark The computation involves the solution matrix Y. As a scalar product of a column vector of Y and an eigenvector of K_w, K_T it can be quickly computed.

Practically, the dimension of the basis for the velocity can be different from the dimension of the basis for the temperature. This is commonly used, since the temperature field is mostly slower than the velocity. Hence, more basis functions would be needed to obtain the same accuracy for the velocity as for the temperature. We consequently use subscripts for distinction. Say, one needs d_w basis functions to sufficiently represent w and d_T basis functions to reconstruct the temperature accurately within the POD setup.

We are now at the stage to express the POD basis functions as linear combinations of the FE basis functions, employing the above defined coefficients γ_{jk}. We obtain

$$\phi_k^P(x) = \sum_{j=1}^{N+1} \gamma_{jk}\phi_j(x) \quad \forall k = 1, \dots, d_w \ , \tag{3.8}$$

$$\psi_k^P(x) = \sum_{j=1}^{N} \gamma_{(j+N+1),k}\psi_j(x) \quad \forall k = 1, \dots, d_T \ . \tag{3.9}$$

This means that the POD basis is somehow a clever expansion of the FE basis. It makes use of the solution structure that is stored in the weighting coefficients γ. To be precise

$$\phi_k^P \in \ span\{\phi_1, \phi_2, \dots, \phi_{N+1}\} \quad \forall k = 1, \dots, d_w \ , \tag{3.10}$$

$$\psi_k^P \in \ span\{\psi_1, \psi_2, \dots, \psi_N\} \quad \forall k = 1, \dots, d_T \ . \tag{3.11}$$

The POD function spaces can be defined as

$$V_{w,d_w}^{POD} = \left\{ \sum_{k=1}^{d_w} \alpha_k \phi_k^P(x) : \alpha_k \in \mathbb{R}, k = 1, \dots, d_w \right\} \subseteq V_w^N \ ,$$

$$V_{T,d_T}^{POD} = \left\{ \sum_{k=1}^{d_T} \beta_k \psi_k^P(x) : \beta_k \in \mathbb{R}, k = 1, \dots, d_T \right\} \subseteq V_T^N \ .$$

The set inclusions are due to (3.10)-(3.11). In the special case $d_w = N + 1$ and $d_T = N$ equality holds, respectively.

The coefficients γ_{ik} play an important role in all calculations within the POD setup. Therefore, observe that $\gamma_{jk} = \phi_k^P(x_j)$ for $j = 1, \dots, N + 1$ and similarly $\gamma_{jk} = \psi_k^P(x_{j-(N+1)})$ for

$j = N+2, \ldots, 2N+1$. Moreover, due to the frequent use of these coefficients in the POD method we do henceforth, if possible, express the computations involving the matrix

$$\Gamma = \begin{bmatrix} \gamma_{11} & \cdots & \gamma_{1d} \\ \vdots & \vdots & \vdots \\ \gamma_{(2N+1),1} & \cdots & \gamma_{(2N+1),d} \end{bmatrix} . \tag{3.12}$$

Observe that $\Gamma = \Gamma(c, \kappa, \mu, N, w_0, T_0)$ depends on the problem parameters and initial conditions.

Orthonormality Crucial to the POD approach is the orthonormality of the POD basis functions. This is one advantage of the POD over the GFE or FEM. The multiplication with the inverse of the mass matrix in every time step does not need to be executed and hence saves computational cost.
The POD basis functions are mutually orthonormal so that

$$< \phi_k^P, \phi_j^P > = \int_0^1 \phi_k^P(x)\phi_j^P(x)dx = \delta_{kj} \quad \forall k, j = 1, \ldots, d_w \ , \tag{3.13}$$

$$< \psi_k^P, \psi_j^P > = \int_0^1 \psi_k^P(x)\psi_j^P(x)dx = \delta_{kj} \quad \forall k, j = 1, \ldots, d_T \ . \tag{3.14}$$

Optimality The POD basis functions minimize the error of the projection of the snapshot sets on an orthonormal basis. Therefore, they provide an optimal orthonormal basis to represent the snapshot set. This result is stated in terms of the velocity variable w since the proceeding is equivalent for T. To begin with, choose an arbitrary orthonormal basis $\{b_i\}_{i=1,\ldots,d_w}$ for the snapshot set $\{w^N(t_i, \cdot)\}_{i=1}^S$. Then, the projection of the snapshots onto this basis is

$$P_{b,d_w} w^N(t_j, x) = \sum_{i=1}^{d_w} c_{ji} b_i(x) \qquad \text{where} \quad c_{ji} = b_i^\mathsf{T} w^N(t_j, \cdot) \ .$$

It can be shown that the error

$$Err = \sum_{j=1}^{S} |w^N(t_j, \cdot) - P_{b,d_w} w^N(t_j, \cdot)|^2$$

is minimized if $b_i = \phi_i^P$ for $i = 1, \ldots, d_w$. The interested reader is referred to [Vol08] for these and more results on the POD. A good overview can also be found in [GPS07].

The choice of d_w and d_T is often based on considering the energy that remains in the system when only using a few basis functions. Recall that a POD mode can be viewed as a vector in a function space. Hence, the corresponding eigenvalue indicates the kinetic energy that is

captured by this mode. The energy remaining in the velocity field can be quantified by the eigenvalues λ^w of K_w. Hence, the relative energy is given by

$$E_w(d_w) = \frac{\sum_{i=1}^{d_w} \lambda_i^w}{\sum_{i=1}^{S} \lambda_i^w} \ . \tag{3.15}$$

The equivalent holds for the temperature field T. As mentioned above, the implication is that the basis for T and w can be chosen independently. One should be aware that low energy modes can also capture a lot of the dynamics. However, in most of the cases only the high energy modes are chosen to approximate the system. Consequently, a convergence analysis of the POD modes should always be performed. That way, one can assure that the POD modes capture not only a sufficient amount of energy, but also capture enough of the system dynamics.

3.2 Group POD Model for the coupled Burgers equation

Within this section we derive the approximating POD system for the coupled Burgers equation (2.1)-(2.6). The approach shows similarities to Chapter 2, where the approximating FE and GFE models were derived. The velocity w and temperature T are recovered in terms of the POD basis as

$$w(t,x) \approx w^P(t,x) = \sum_{k=1}^{d_w} \alpha_k^P(t)\phi_k^P(x) \tag{3.16}$$

and

$$T(t,x) \approx T^P(t,x) = \sum_{k=1}^{d_T} \beta_k^P(t)\psi_k^P(x) \ . \tag{3.17}$$

The POD approach to reduce the model order is certainly different from the GFE. However, central to both methods is the Galerkin principle. It states that the error which occurs by projecting the equations on a finite dimensional subspace must be orthogonal to each of the basis functions. We follow the derivation of the GFE in Section 2.5. First, we substitute the POD basis functions for the GFE basis functions in (2.21). Note that the solution is recovered by (3.16) and (3.17). This produces the approximate weak form for the Group

POD method as

$$\int_0^1 \left[[w^P]_t(t,x)\phi^P(x) + [T^P]_t(t,x)\psi^P(x)\right] dx$$

$$= -\int_0^1 \left[\frac{1}{2}[(w^P(t,x))^2]_x\phi^P(x) + w^P(t,x)(T^P)_x(t,x)\psi^P(x)\right] dx - \int_0^1 \mu(w^P)_x(t,x)\phi_x^P(x)dx$$

$$+ \mu\delta\phi^P(1) - \int_0^1 c(T^P)_x(t,x)\psi_x^P(x)dx - \int_0^1 \kappa T^P(t,x)\phi^P(x)dx + \int_0^1 f(t,x)\psi^P(x)dx$$

$$\forall [\phi^P \psi^P]^\mathsf{T} \in V_{w,d_w}^{POD} \times V_{T,d_T}^{POD}$$

where ϕ_x^P is the first spacial derivative with respect to x of the POD basis function for the velocity w. To obtain an approximation of the weak solution, one has to project the initial conditions on the POD function spaces to obtain the initial data for the POD method.

First, we focus on the boundary terms. In the derivation of the weak form in Section 2.3 boundary terms arise from the integration by parts. However, it was argued that the basis functions have to vanish at exactly the same boundaries where the solution vanishes. For the solution, this is enforced by the boundary conditions (2.3)-(2.4). Therefore, every basis used to recover the solution should match the boundary conditions.
A second way to see this is by considering the definitions of the POD basis functions (3.8)-(3.9) and that the ϕ_i and ψ_i are the FE hat functions with local support. Hence, $\psi_j^P(0) = \psi_j^P(1) = 0$ since $\psi_1(0) = \psi_N(1) = 0$. We equivalently obtain $\phi_i^P(0) = 0$ where the equalities hold for $i = 1, \ldots, d_w,\ j = 1, \ldots, d_T$. Moreover, recall the Neumann boundary condition $w_x(t,1) = \delta$. Thus, the following equality holds

$$\mu\left[[w^P(t,x)]_x\phi_i^P(x)\right]_0^1 + c\left[[T^P(t,x)]_x\psi_j^P(x)\right]_0^1 = \mu\delta\phi_i^P(1) \qquad \forall i = 1, \ldots, d_w\ .$$

Having obtained the weak formulation the last step leading to the discretization involves substituting the POD approximations with its expansions in terms of the POD basis as set

in (3.16)-(3.17). The following equality again holds for $i = 1, \ldots, d_w,\ j = 1, \ldots, d_T$:

$$\begin{aligned}
&\int_0^1 \sum_{k=1}^{d_w} \dot{\alpha_k}^P(t)\phi_k^P(x)\phi_i^P(x)dx + \int_0^1 \sum_{k=1}^{d_T} \dot{\beta_k}^P(t)\psi_k^P(x)\psi_j^P(x)dx \\
&= -\frac{1}{2}\int_0^1 \left[\left(\sum_{k=1}^{d_w} \alpha_k^P(t)\phi_k^P(x)\right)^2\right]_x \phi_i^P(x)dx - \int_0^1 \sum_{k=1}^{d_w} \alpha_k^P(t)\phi_k^P(x) \sum_{k=1}^{d_T} \beta_k^P(t)\psi_{k,x}^P(x)\psi_j^P(x)dx \\
&\quad - \int_0^1 \mu \sum_{k=1}^{d_w} \alpha_k^P(t)\phi_{k,x}^P(x)\phi_{i,x}^P(x)dx + \mu\delta\phi_i^P(1) - \int_0^1 c\sum_{k=1}^{d_T} \beta_k^P(t)\psi_{k,x}^P(x)\psi_{j,x}^P(x)dx \\
&\quad - \int_0^1 \kappa \sum_{k=1}^{d_T} \beta_k^P(t)\psi_k^P(x)\phi_i^P(x)dx + \int_0^1 f(t,x)\psi_j^P(x)dx\ .
\end{aligned} \tag{3.18}$$

For clarity and to outline the computational procedure we write the above equation in matrix-vector form and present ways to calculate the respective matrices. One should recognize that most of the matrices can be computed from the matrices in Chapter 2 and the matrix Γ as defined in (3.12). The following discretized system of ODE's needs to be solved

$$\hat{M}^P \begin{bmatrix} \dot{\alpha^P} \\ \beta^P \end{bmatrix} = -\begin{bmatrix} \frac{1}{2}N(\alpha^P) \\ B\Lambda(\alpha^P, \beta^P) \end{bmatrix} - \begin{bmatrix} \mu S_{d_w}^P & \kappa C^P \\ 0 & cS_{d_T}^P \end{bmatrix} \begin{bmatrix} \alpha^P \\ \beta^P \end{bmatrix} + \begin{bmatrix} 0 \\ < f(t,\cdot), \psi_j^P > \end{bmatrix} + \vec{b} \tag{3.19}$$

where $\alpha^P = \left[\alpha_1^P, \ldots, \alpha_{d_w}^P\right]^\mathsf{T}$ and $\beta^P = \left[\beta_1^P, \ldots, \beta_{d_T}^P\right]^\mathsf{T}$ are the time dependent nodal unknowns of the system.
In the following the POD matrices are evaluated and the close connection to the matrices $M_w^N, M_T^N, S_w^N, S_T^N, C^N$ occurring in the FEM will be worked out.

Linear Terms The global mass matrix is block diagonal and consists of the mass matrix for the temperature coefficients and the mass matrix for the velocity coefficients. It follows

$$\hat{M}^P = \begin{bmatrix} M_w^P & 0 \\ 0 & M_T^P \end{bmatrix}\ .$$

Note that the mass matrix $\hat{M}^P$ is the identity matrix, since the POD modes for each dependent variable are orthonormal as stated in (3.13)-(3.14)[4].

The linear terms of the right hand side of (3.19) are computed next. This demonstrates how the FEM converts into the POD method as far as computation goes. The POD stiffness

[4]To double check within a computational routine, $\Gamma^T \hat{M}^P \Gamma$ has to be the identity.

matrix for the velocity part is defined for $k, l = 1, \ldots, d_w$ by

$$\left(S^P_{d_w}\right)_{kl} = \int_0^1 \phi^P_{k,x}(x)\phi^P_{l,x}(x)dx = \int_0^1 \sum_{i=1}^{N+1} \gamma_{ik}\phi_i'(x) \sum_{j=1}^{N+1} \gamma_{jl}\phi_j'(x)dx$$
$$= \sum_{i,j=1}^{N+1} \gamma_{ik}\gamma_{jl} \int_0^1 \phi_i'(x)\phi_j'(x)dx \ .$$

Recall that the integral term defines the stiffness matrix S^N_w. Hence, this matrix can be computed as

$$S^P_{d_w} = \Gamma(1 : N+1, 1 : d_w)^\mathsf{T} \ S^N_w \ \Gamma(1 : N+1, 1 : d_w) \ . \tag{3.20}$$

Equivalently, for $k, l = 1, \ldots, d_T$ it holds

$$\left(S^P_{d_T}\right)_{kl} = \int_0^1 \psi^P_{k,x}(x)\psi^P_{l,x}(x)dx = \sum_{i,j=1}^{N} \gamma_{(i+N+1),k}\gamma_{(j+N+1),l} \int_0^1 \psi_i'(x)\psi_j'(x)dx$$

which can be computed as

$$S^P_{d_T} = \Gamma(N+2 : 2N+1, 1 : d_T)^\mathsf{T} \ S^N_T \ \Gamma(N+2 : 2N+1, 1 : d_T) \ . \tag{3.21}$$

Note that both matrices are symmetric, as in the FEM.

The matrix C^P is part of the coupling of the Burgers equation to the heat equation. This time, $C^P \in \mathbb{R}^{d_w} \times \mathbb{R}^{d_T}$. For $k = 1, \ldots, d_w$ and $l = 1, \ldots, d_T$ we compute C^P as

$$C^P_{kl} = \int_0^1 \phi^P_k(x)\psi^P_l(x)dx = \sum_{j=1}^{N+1} \sum_{i=1}^{N} \gamma_{jk} \left(\int_0^1 \phi_j(x)\psi_i(x)dx \right) \gamma_{(i+N+1),l}$$

which can be implemented in a computational routine as

$$C^P = \Gamma(1 : N+1, 1 : d_w)^\mathsf{T} \ C^N \ \Gamma(N+2 : 2N+1, 1 : d_T) \ . \tag{3.22}$$

This matrix is non-symmetric in general. We obtain the equality

$$\int_0^1 \sum_{k=1}^{d_T} \beta^P_k(t)\psi^P_k(x)\phi^P_i(x)dx = C^P\beta^P \ .$$

Remark The derivation of the linear terms in the POD model showed a close connection to the linear parts in the Finite Element Methods. The mass matrix turned out to be the

identity. All other matrices were obtained by multiplying the FE matrices with the corresponding parts of Γ^{T} and Γ from the left and the right, respectively. However, observe that this results in the loss of sparsity in the matrices, they are in general not sparse anymore. For the reduced order model, this implies that a dense nonlinear system has to be solved. Thus, for choices of many POD basis functions, the computational speed-up diminishes with increasing basis size.

Boundary Conditions We turn our focus to the vector $\vec{b}$ that arises from the boundary conditions. Define

$$\vec{b} = \mu\delta \left[\phi_1^P(1), \ldots, \phi_{d_w}^P(1), 0, \ldots, 0\right]^{\mathsf{T}} .$$

Due to the definition of the POD basis functions in (3.8), one obtains

$$\phi_i^P(1) = \sum_{j=1}^{N+1} \gamma_{ji}\phi_j(1) = \gamma_{(N+1),i} \quad \forall i = 1, \ldots, d_w$$

which implies that

$$\vec{b} = \mu\delta \left[\gamma_{(N+1),1}, \ldots, \gamma_{(N+1),d_w}, 0, \ldots, 0\right]^{\mathsf{T}} .$$

Nonlinear Terms The nonlinearities have to be evaluated in every time step. Consequently, they have a crucial impact on the run time of the POD. Therefore, one aims to economize the computational cost to evaluate the nonlinear terms. A treatment of the nonlinear coupling term will be given below. Consider for $j = 1, \ldots, d_T$

$$\int_0^1 \sum_{k=1}^{d_w} \alpha_k^P(t)\phi_k^P(x) \sum_{l=1}^{d_T} \beta_l^P(t)\psi_{l,x}^P(x)\psi_j^P(x)dx$$

$$= \sum_{k=1}^{d_w}\sum_{l=1}^{d_T} \alpha_k^P(t)\beta_l^P(t) \int_0^1 \phi_k^P(x)\psi_{l,x}^P(x)\psi_j^P(x)dx = B\Lambda(\alpha^P(t), \beta^P(t)) .$$

The computational implementation of this can be either done by a third order tensor acting on a matrix or by rearranging the coefficients and assembling them in a $d_w \cdot d_T$ dimensional vector. We pursuit the latter approach and provide definitions for the vector Λ and the matrix B[5]. Define the vector valued function Λ of the time dependent nodal unknowns $\alpha(t), \beta(t)$ by

$$\Lambda(\alpha^P(t), \beta^P(t)) = \left[\alpha_1^P\beta_1^P, \ldots, \alpha_1^P\beta_{d_T}^P, \alpha_2^P\beta_1^P, \ldots, \alpha_2^P\beta_{d_T}^P, \ldots, \alpha_{d_w}^P\beta_1^P, \ldots, \alpha_{d_w}^P\beta_{d_T}^P\right]^{\mathsf{T}} .$$

[5]The matrix can be computed offline, so that the integration of a triple product of basis functions does not have to be computed in every time step. However, the vector valued function Λ has to be evaluated by the ODE solver in every time step

Consider the matrix B and let $k = 1, \dots, d_w, l = 1, \dots, d_T$ be row indices and $j = 1, \dots, d_T$ indexes the rows. Define

$$B_{klj} = \int_0^1 \phi_k^P(x)\psi_{l,x}^P(x)\psi_j^P(x)dx$$

which we emphasize is the matrix

$$B = \begin{bmatrix} B_{111} & \cdots & B_{1d_T1} & \cdots & B_{d_w11} & \cdots & B_{d_wd_T1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ B_{11d_T} & \cdots & B_{1d_Td_T} & \cdots & B_{d_w1d_T} & \cdots & B_{d_wd_Td_T} \end{bmatrix}_{d_T \times d_w \cdot d_T} .$$

We turn our focus to the grouped variable. Consider the nonlinearity of the Burgers equation

$$\frac{1}{2}N(\alpha^P) = \frac{1}{2} < \left[\left(\sum_{k=1}^{d_w} \alpha_k^P(t)\phi_k^P(x) \right)^2 \right]_x , \ \phi_i^P(x) > \ . \tag{3.23}$$

The computation of the nonlinearity involves two tasks. First, define a way to approximate the grouped variable in the POD method. Second, one needs to evaluate the inner product. We shall highlight some important issues here. Within the FEM setting, the basis functions have a local support between every second node in the space grid. This resulted in only the products of neighboring basis functions being nonzero. After all, this implies that the mass and stiffness matrix have tridiagonal structure. The POD basis functions as constructed in (3.16)-(3.17) are by definition not locally but globally supported.
Further, the local support of the FEM basis functions results from the property that they interpolate the nodes in a sense that $\phi_i(x_j) = \delta_{ij}$. The POD basis functions lack this feature naturally. This means that a POD basis function will in general have a nonzero value at every space node. We shall soon see, that this makes the computation of the grouped variable more tedious.
We follow the approach of [DS10] that promises computational advantages in grouping Burgers equation. To begin with, let the grouped variable be approximated as

$$w^2(t,x) \approx (w^P(t,x))^2 = \sum_{k=1}^{d_w} F_k(\alpha^P(t))\phi_k^P(x) \ .$$

Observe that this expansion is with respect to the same basis used to expand w. In order to calculate F_k we pursuit a nodal approach. Assume that the grouped variable $(w^P)^2$ is approximated by the standard approximation (3.16) at the spacial nodes, i.e. at the n^{th} node let

$$w^2(t,x_n) \approx (w^P(t,x_n))^2 = \sum_{k=1}^{d_w} F_k(\alpha^P(t))\phi_k^P(x_n) \stackrel{!}{=} \left(\sum_{k=1}^{d_w} \alpha_k^P(t)\phi_k^P(x_n) \right)^2 \ .$$

Note that the last equality is enforced. For convenience and as mentioned earlier, the equality $\gamma_{nk} = \phi_k(x_n)$ for $n = 1, \ldots, N+1, k = 1, \ldots, d_w$ is used. Rewriting the above equation without explicitly mentioning the time dependence yields

$$\sum_{k=1}^{d_w} F_k(\alpha^P)\gamma_{nk} = \left(\sum_{k=1}^{d_w} \alpha_k^P \gamma_{nk}\right)^2 = \sum_{k,l=1}^{d_w} \gamma_{nk}\gamma_{nl}\alpha_k^P\alpha_l^P = \hat{\gamma}_n \hat{\alpha}^P \tag{3.24}$$

which holds at the n^{th} node, for $n = 1, \ldots, N+1$. Proceeding as above this equation shall be expressed as a matrix-vector product. The symmetry $\gamma_{nk}\gamma_{nl} = \gamma_{nl}\gamma_{nk}$ helps to reduce the dimension of the following vectors. Define

$$\hat{\gamma}_n = [\gamma_{n1}\gamma_{n1}, 2\gamma_{n1}\gamma_{n2}, \ldots, 2\gamma_{n1}\gamma_{nd_w}, \gamma_{n2}\gamma_{n2}, \ldots, 2\gamma_{n2}\gamma_{nd_w}, \ldots, \gamma_{n,d_w-1}\gamma_{n,d_w-1}, 2\gamma_{n,d_w-1}\gamma_{nd_w}, \gamma_{nd_w}\gamma_{nd_w}]$$

and

$$\hat{\alpha}^P = [\alpha_1^P\alpha_1^P, \ldots, \alpha_1^P\alpha_{d_w}^P, \alpha_2^P\alpha_2^P, \ldots, \alpha_2^P\alpha_{d_w}^P, \ldots, \alpha_{d_w-1}^P\alpha_{d_w-1}^P, \alpha_{d_w-1}^P\alpha_{d_w}^P, \alpha_{d_w}^P\alpha_{d_w}^P]^\mathsf{T} \quad ,$$

respectively.

Thus, (3.24) can be simplified and rewritten as

$$\Gamma(1:N+1, 1:d_w)F(\alpha^P) = \hat{\Gamma}_{N+1}\hat{\alpha}^P \ , \tag{3.25}$$

where $F(\alpha^P) = [F_1(\alpha^P), \ldots, F_{N+1}(\alpha^P)]^\mathsf{T}$ and $\hat{\Gamma}_{N+1} = [\hat{\gamma}_1, \ldots, \hat{\gamma}_{N+1}]^\mathsf{T}$.

We are ready to return to the starting point (3.23) and evaluate the inner product. This produces the equation

$$\begin{aligned}
\frac{1}{2} < \left(\sum_{k=1}^{d_w} F_k(\alpha^P)\phi_k^P(x)\right)_x , \ \phi_i^P(x) > &= \frac{1}{2}\sum_{k=1}^{d_w} F_k(\alpha^P) \int_0^1 \phi_{k,x}^P(x)\phi_i^P(x)dx \\
&= \frac{1}{2}\sum_{k=1}^{d_w} F_k(\alpha^P) \int_0^1 \sum_{j=1}^{N+1} \gamma_{jk}\phi_{j,x}(x) \sum_{l=1}^{N+1} \gamma_{li}\phi_l(x)dx \\
&= \frac{1}{2}\sum_{k=1}^{d_w}\sum_{j,l=1}^{N+1} F_k(\alpha^P) \left(\gamma_{jk}\int_0^1 \phi_{j,x}(x)\phi_l(x)dx \ \gamma_{li}\right) \\
&= \frac{1}{2}\sum_{k=1}^{d_w}\sum_{j,l=1}^{N+1} \gamma_{li}\left(\int_0^1 \phi_l(x)\phi_{j,x}(x)dx\right)\gamma_{jk}F_k(\alpha^P) \\
&= \frac{1}{2}\Gamma(1:N+1, 1:d_w)^\mathsf{T} \ A^P \ \Gamma(1:N+1, 1:d_w) \ F(\alpha^P) \ .
\end{aligned}$$

Using (3.25) yields

$$N(\alpha^P) = \hat{N}\hat{\alpha}^P \ , \tag{3.26}$$

where $\hat{N}$ is defined as

$$\hat{N} = \Gamma(1 : N+1, 1 : d_w)^\mathsf{T} \ A^P \ \hat{\Gamma}_{N+1} \ . \tag{3.27}$$

Note that this matrix can be computed offline once the matrices Γ and A^P are computed.

The matrix A^P is defined as

$$A^P_{lj} = \int_0^1 \phi_l(x)\phi_{j,x}(x)dx \qquad \text{for} \quad j,l = 1,\dots,N+1 \ .$$

Observe that this is the matrix A from (2.22) as defined in the GFE method. That is

$$A^P = A \ . \tag{3.28}$$

Initial Conditions Similar to the FEM in Chapter 2, the initial conditions have to be approximated in the POD space. To obtain the projection of the initial condition a L^2 scalar product with every POD basis function is computed. The proceeding is similar to the FEM treatment of the initial conditions. Recall that the initial conditions can be stated with respect to the POD basis as

$$w^P(0,x) = \sum_{k=1}^{d_w} \alpha_k^P(0)\phi_k^P(x) \ ,$$

$$T^P(0,x) = \sum_{k=1}^{d_T} \beta_k^P(0)\psi_k^P(x) \ .$$

As an example of the procedure, let us only treat the initial conditions for the velocity w. Evaluating the inner product with each basis function delivers

$$\int_0^1 w^P(0,x)\phi_i^P(x)dx \approx \int_0^1 w_0(x)\phi_i^P(x)dx \quad \text{for } i = 1,\dots,d_w \ .$$

Observe that this is an approximation, so the initial conditions themselves are not exact but contain an error compared to the exact initial conditions. When we plug in the above expansion this equality evaluates to

$$\sum_{k=1}^{d_w} \alpha_k^P(0) \int_0^1 \phi_k^P(x)\phi_i^P(x)dx \approx \int_0^1 w_0(x)\phi_i^P(x)dx \quad \text{for } i = 1,\dots,d_w \ .$$

For convenience denote $\alpha^P(0) = [\alpha_1^P(0), \ldots, \alpha_{d_w}^P(0)]^\mathsf{T}$ and the vector

$$\hat{w}_0^P = \begin{bmatrix} \int_0^1 w_0(x)\phi_1^P(x)dx \\ \vdots \\ \int_0^1 w_0(x)\phi_{d_w}^P(x)dx \end{bmatrix} .$$

Consequently, the initial condition for the approximating velocity variable is

$$\alpha^P(0) = \left[\hat{M}_w^P\right]^{-1} \hat{w}_0^P \ . \tag{3.29}$$

With equivalent notation, the initial condition for the approximation of the temperature field is similarly obtained as

$$\beta^P(0) = \left[\hat{M}_T^P\right]^{-1} \hat{T}_0^P \ . \tag{3.30}$$

Since the POD mass matrices are the identity, the inversions are redundant but are mentioned to be consistent.

Remark To be consistent, we shall mention that the initial conditions for the POD model can be obtained from the initial conditions of the Finite Element Methods. Hence, rewriting (3.2) in terms of the POD basis expansion yields

$$\int_0^1 w_0(x)\phi_k^P(x)dx = \sum_{j=1}^{N+1} \gamma_{jk} \int_0^1 w_0(x)\phi_j(x)dx = \Gamma(1:N+1,k)^\mathsf{T}\hat{w}_0 \quad \forall\ k = 1,..,d_w$$

Thus, the initial condition for the velocity in the POD model is obtained as

$$\hat{w}_0^P = \Gamma(1:N+1, 1:d_w)^\mathsf{T}\ \hat{w}_0$$

A similar result holds for the initial temperature. That is

$$\hat{T}_0^P = \Gamma(N+2:2N+1, 1:d_T)^\mathsf{T}\ \hat{T}_0$$

Final result The coupled Burgers equation is approximated by a projection on the POD basis. In the previous sections all steps that lead to the formulation of the space discretized ODE system were treated thoroughly. Moreover, we presented calculations that can be conveniently implemented in MATLAB ®.

In order to find approximate solutions to the coupled Burgers equation, one needs to solve the system of ODE's given by

$$\hat{M}^P \begin{bmatrix} \dot{\alpha^P} \\ \beta^P \end{bmatrix} = - \begin{bmatrix} \frac{1}{2}\hat{N}\hat{\alpha}^P \\ B\Lambda(\alpha^P, \beta^P) \end{bmatrix} - \begin{bmatrix} \mu S_{d_w}^P & \kappa C^P \\ 0 & cS_{d_T}^P \end{bmatrix} \begin{bmatrix} \alpha^P \\ \beta^P \end{bmatrix} + \begin{bmatrix} 0 \\ < f(t,\cdot), \psi_j^P > \end{bmatrix} + \vec{b} \ , \tag{3.31}$$

with initial conditions

$$\begin{bmatrix} \alpha^P(0) \\ \beta^P(0) \end{bmatrix} = \begin{bmatrix} \hat{w}_0^P \\ \hat{T}_0^P \end{bmatrix} \ . \tag{3.32}$$

All occurring vectors and matrices were defined within this section.

3.3 Numerical Examples

The interesting and desired feature about the POD method is its computational efficiency. This surprisingly often goes along with sufficient accuracy received with only a few basis functions. Thus, it is a powerful method in model reduction and especially employable for real time calculations. Within this section an example focusing on computational effort and accuracy is provided. Especially for the coupled Burgers equation the POD proves itself to be fast and accurate. Thus, it is useful for situations where many simulations need to be performed. Moreover, one can attempt to use POD for the design of controllers and optimization of the coupled Burgers equation.

3.3.1 Comparison of Group POD with GFE

This example compares the two methods that are both based on grouping the nonlinearity of the Burgers equation. However, the implementations are different, as seen previously. The GFE turned out to be a stable method for solving the coupled Burgers equation (see Chapter 2). The motivation to use the Group POD instead of the standard POD is based on the recent work of J.Singler and B.Dickinson [DS10] where computational advantages have been shown.

Again, the dynamics are given by

$$\begin{aligned} w_t(t,x) + w(t,x)w_x(t,x) &= \mu w_{xx}(t,x) - \kappa T(t,x) \ , \\ T_t(t,x) + w(t,x)T_x(t,x) &= cT_{xx}(t,x) + f(t,x) \ , \end{aligned}$$

with boundary conditions

$$\begin{aligned} w(t,0) &= 0 \ , & w_x(t,1) &= 0 \ , \\ T(t,0) &= 0 \ , & T(t,1) &= 0 \ , \end{aligned}$$

and initial conditions

$$w_0(x) = x^2(0.5 - x)^2 \ ,$$
$$T_0(x) = \frac{1}{2}\sin^5(\pi x) \ .$$

The system is forced by the function f given by

$$f(t,x) = \frac{1}{10}|t-5|\cos(2x) \ .$$

and is plotted in Figure 3.1.

The parameters for the problem are $Re = 120$, where $\mu = \frac{1}{Re}$, $c = 0.01$ and $\kappa = 1.0$. The number of internal nodes for the GFE method is $N = 150$. The final time is picked as $t_f = 20s$ and we take $S = 150$ time snapshots of the system[6]. For the integration of the forcing function and initial conditions against the POD basis, 51 FE for the Gauss quadrature points are defined. All error calculations are performed with 33 Gauss elements in space and 51 elements in time. Recall, that the integral is evaluated using three points in each element.

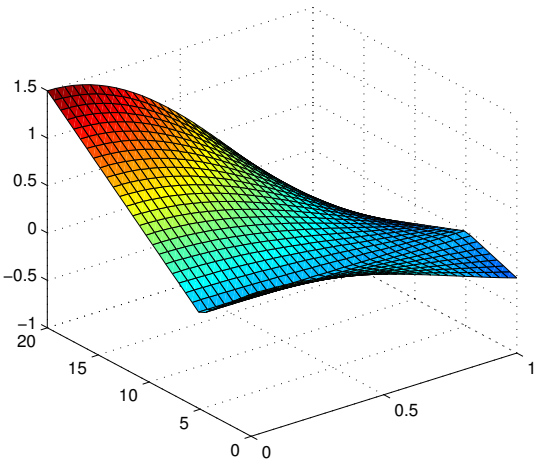


Figure 3.1: Forcing on the System $f(t,x)$

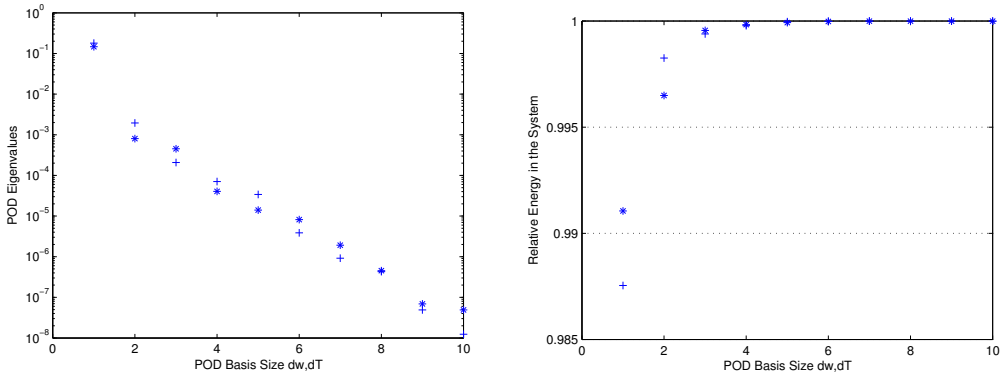
First, the GFE model was solved with the ODE15s solver. As a result we obtained the solution matrix Y. In a next step, the correlation matrices and their eigenvalues and eigenvectors are computed for the temperature and velocity, respectively. We picked a level of accuracy which is determined by the number of POD eigenmodes, see Figure 3.2, and simulations of convergence. Then, the generation of the basis functions, the computation of the matrices

[6]Hence, the solution matrix Y of the GFE has dimension 150×301

for the linear terms and the matrices $B, \hat{N}$ for the nonlinearities are computed. All these computations can be done offline which is certainly an advantage of the POD. Finally, the resulting ODE system is solved with the MATLAB ® ODE15s solver.

The POD method is desired to be a computational tool for rapid simulations. In particular, we are interested in the computational effort for solving the ODE system and do not focus on the offline calculations. Consequently, the CPU time is only measured for solving the ODE's of the GFE method. We do not include the Pre-processing time. [7] The time needed to solve the ODE's in the GFE method was $3.61s$. This is the benchmark for the comparison to the POD simulations.

We first perform the evaluation of the correlation matrices and calculate the respective eigenvalues and eigenvectors. The eigenvalues are ordered by size. A plot of the first 10 eigenvalues of K_w and K_T is given in Figure 3.2. Further, the energy remaining in the system by selecting a certain number of eigenvectors to build the POD basis is plotted. The reader should notice that the eigenvalues decay rapidly.Therefore, the energy in the temperature and flow field quickly reaches a level of 99.5%.



(a) Eigenvalues λ_k^w, λ_k^T of the POD Eigenmodes

(b) Energy remaining in the Systems w^P, T^P by Selection of k Eigenmodes for the POD Basis

Figure 3.2: Eigenvalue Behavior for the POD Basis: $*(w), +(T)$

Now we project the coupled Burgers equation onto the POD basis. Accordingly, the approximating ODE system is solved. The results are compared to the GFE solution. We choose several combinations of POD eigenmodes for the velocity and temperature. The errors are calculated for each of the simulations, see Table 3.1. The global error is computed with the relative error norm given in (2.28), where the approximation w^N has to be replaced by w^P and similarly T^N by T^P.

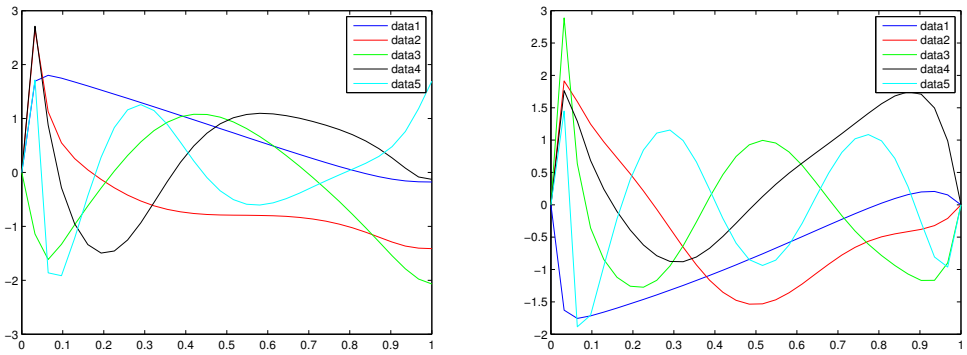
[7] Pre-processing time was included in Chapter 2.

(d_w,d_T)	CPU Time	Err^{rel}
(2,2)	0.181	0.1333
(3,3)	0.215	0.0729
(5,4)	0.287	0.0654
(5,5)	0.368	0.0289
(6,5)	0.335	0.0209

Table 3.1: CPU Times and Global Errors for POD Solutions

Based on the decay of the eigenvalues and the error convergence, the number of POD eigenmodes is chosen to be $d_w = 5, d_T = 5$. This yields a global error of the POD compared to the GFE of 2.9% but with only 8% of the time needed. One might also note that a further increase in accuracy does not increase the computational effort significantly, see Table 3.1. Matching the GFE solution very well, the POD method needs only 1/10 of the time. This is a promising result with regards to real time applications.

Five POD basis functions for the velocity and five POD basis functions for the temperature are plotted in Figure 3.3.

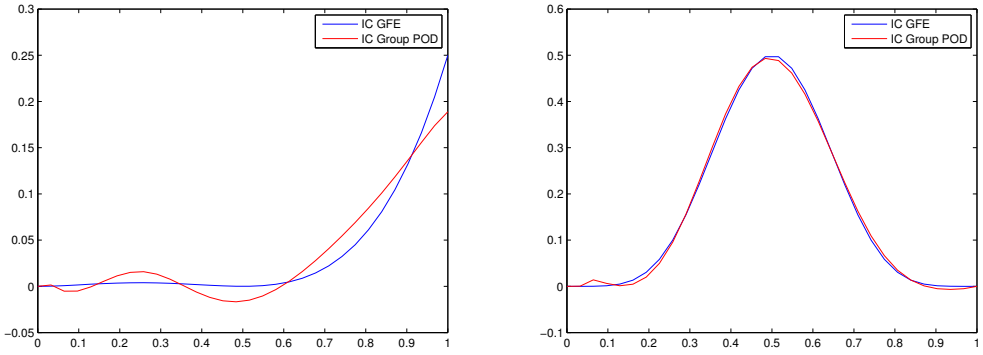


(a) Basis Functions ϕ_k^w (b) Basis Functions ψ_k^w

Figure 3.3: POD Basis Functions

Observe that higher order POD basis functions show similarities to low order ones. This seems intuitive, since the first few modes already capture almost all of the dynamics. Moreover, the basis functions match the boundary conditions which is basically due to the fact that they are expansions of the FEM basis.

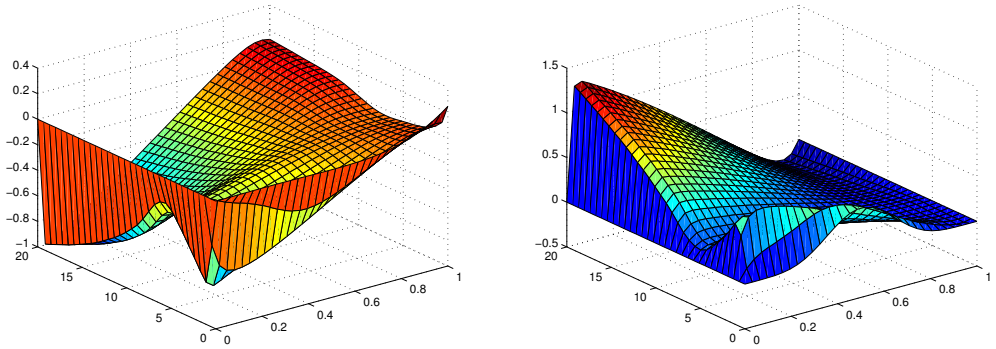
Plots for both GFE solutions and error plots for the POD differing from the GFE are



(a) Initial Velocity $w_0^N,\ w_0^P$ (b) Initial Temperature $T_0^N,\ T_0^P$

Figure 3.4: GFE and POD Initial Conditions

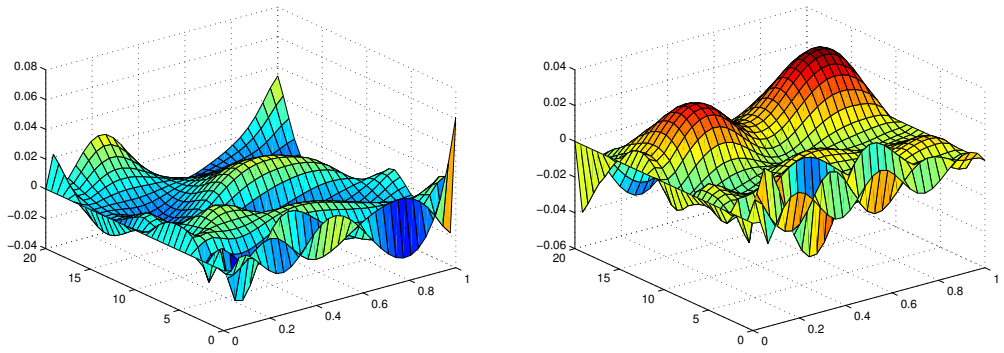
presented in Figures 3.5 and 3.6.



(a) GFE Solution for the Velocity $w^N(t,x)$ (b) GFE Solution for the Temperature $T^N(t,x)$

Figure 3.5: GFE Solution

Note that the error plots are as in Section 2.6 given absolutely.



(a) $w^N(t,x) - w^P(t,x)$ (b) $T^N(t,x) - T^P(t,x)$

Figure 3.6: Error between POD and GFE Solution

The error plots and the global errors reveal that the POD captures the main dynamics of the system fairly well. However, note from Figure 3.4 that the initial conditions are not matched as precise as for instance the GFE method would do. Recall, that the POD is based on snapshots in time. Therefore it captures the evolution of the equations but it is not designed to match the dynamics in a single time step. For the POD the initial condition is equally "important" than any other instance of time. In particular, if the initial conditions are inconsistent with the dynamics, the POD will have more error in the initial condition. This is since POD focuses somehow more on the averaged dynamics of the system.

Overall, we conclude that the POD model provides a good computational tool for model reduction for the coupled Burgers equation. The computational cost was reduced to 10% of the time needed for the GFE simulation. Further, only 2% error in accuracy compared to the GFE full model solution was obtained. The POD model captures the dynamics of the system well for all times.

3.3.2 Parameter Dependence

Many applications in fluid dynamics have to deal with systems that are governed by physical parameters. In our case, the parameters that govern the system are the thermal conductivity c, the coefficient of the thermal expansion κ and the Reynolds number Re. Moreover, the initial conditions w_0, T_0 and the forcing function f influence the system. However, thinking of real applications in the design of energy efficient buildings, these parameters and outer influences are subject to change over time. Loosely speaking, the humidity of air in a room is related to the Reynolds number and varies during the day. The rate at which the air conducts heat is connected to the humidity as well. Moreover, the temperature and velocity

fields change continuously. Therefore one encounters a different initial state each time the simulation is performed.

We are particularly interested in further applications for the design of controllers and optimization. The Neumann boundary condition δ can be viewed as a control on the system. Thinking of it from a control point of view, a closed loop controller would have to be calculated repeatedly. Each time, the system is governed by different initial conditions, forcing functions and parameters. This motivates the idea to find a reduced model that captures ranges of parameters, initial states or disturbances.

An interesting question is if the POD model can cover a range of parameter values for Re, c and κ. In particular, we want to find "averaged" parameter values to build a reduced order POD model. This leads to the question: How accurate is the reduced model if it is applied for other parameter values than the ones it was constructed? How sensitive is the POD model to system parameters?

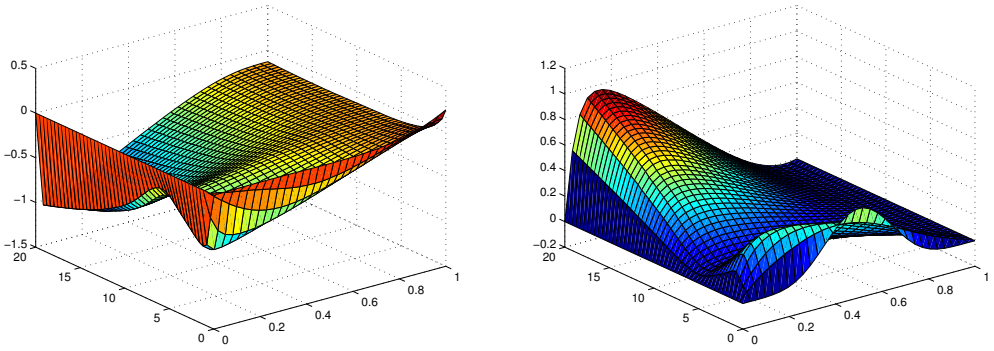
Example 1 The full GFE model for the coupled Burgers equation is solved with parameters $c = 0.01,\ \kappa = 1.0,\ Re = 120$ and $N = 150$. The initial conditions and forcing function are chosen as in the last example. The final time is $t_f = 20s$. The integration of the initial condition and the forcing function against the POD basis was performed with a three point Gauss quadrature rule, where 51 Gauss elements are defined in $[0, 1]$. The number of time snapshots for the construction of the POD basis is $S = 150$. The global L^2 errors are calculated with 51 elements in time and 33 elements in space. Having solved the GFE model, we calculate the POD basis functions based on these solutions. We pick $d_w = 5,\ d_T = 4$ basis functions for the velocity and flow, respectively. Accordingly, the dynamical system is projected on this basis, but the coefficient of the thermal expansion κ and the thermal conductivity c are varied. The solution to this system is compared to the high-fidelity solution for these parameters. This gives us insight into the sensitivity of the reduced POD model to these parameters.

The errors are measured globally in time and space. The results of the simulations are shown in Table 3.2.

c	κ	$Error$
0.02	0.9	0.0950
0.03	1.1	0.0748
0.05	1.1	0.1174
0.05	1.3	0.1167

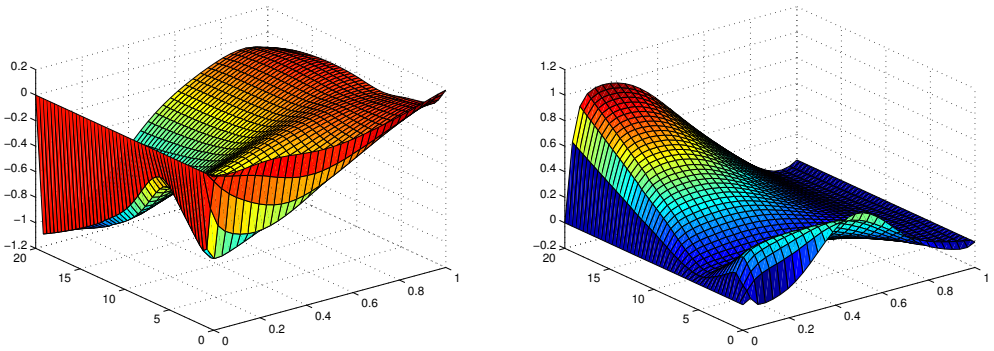
Table 3.2: Global Error between POD and GFE Model for various Parameters

Interestingly, the variation of the parameters still seems to deliver good results. Even with a considerably high deviation of 150% in c and 30% in κ the calculated solutions contain a global error of 11.7%. The results give rise to promising applications of the POD for the coupled Burgers equation. Not only does the POD provide a good low dimensional system for rapid computations, but also captures the dynamics over a range of parameters. Hence, we conclude that a further use of this method is justifiable for control applications. Of course one needs to reevaluate the numerical sensitivity and performance for a control application and this is a sufficiently different task.



(a) Velocity $w^N(t,x)$ (b) Temperature $T^N(t,x)$

Figure 3.7: GFE Solutions for $c = 0.05$ and $\kappa = 1.3$



(a) Velocity $w^P(t,x)$ (b) Temperature $T^P(t,x)$

Figure 3.8: POD Solutions for $c = 0.05$ and $\kappa = 1.3$ with $c = 0.01, \kappa = 1.0$ Basis

In Figure 3.7 the solutions of the full GFE model are plotted for the temperature and velocity, respectively. The solutions were obtained for parameters of $c = 0.05$ and $\kappa = 1.3$.

For comparison, Figure 3.8 shows the POD solutions using 5 basis functions for the velocity and 4 basis functions for the temperature. However, the basis was generated by a solution for the parameters $c = 0.01$ and $\kappa = 1.0$. Note that the POD solutions cover the dynamics fairly well although the basis was not tailored to this solution.

Example 2 In the previous example we considered numerous variations of the parameters c and κ. Within this example we focus on the third parameter of the model, the Reynolds number. The procedure to obtain the basis is similar to the previous example. First, the full GFE model is executed with the parameters $Re = 100,\ N = 150,\ c = 0.01$ and $\kappa = 1.0$. The initial conditions and forcing functions are chosen as in Paragraph 3.3.1. All other specifications were made in the last example. The POD basis is computed from the solutions of the system that is governed by these parameters. We pick $d_w = 5$ basis functions for the velocity field and $d_T = 5$ basis functions to recover the temperature field. The system is projected onto this POD space and the Reynolds number is changed successively. The results for the global L^2 error are given in Table 3.3 and are plotted in Figure 3.9.

Re	80	100	120	130	140	150
Error	0.0448	0.0296	0.0696	0.1028	0.1672	0.2303

Table 3.3: Global Error between POD and GFE model for various Reynolds numbers

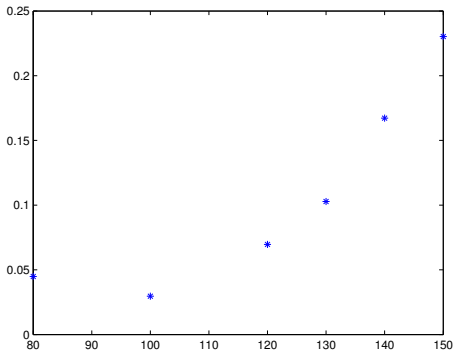
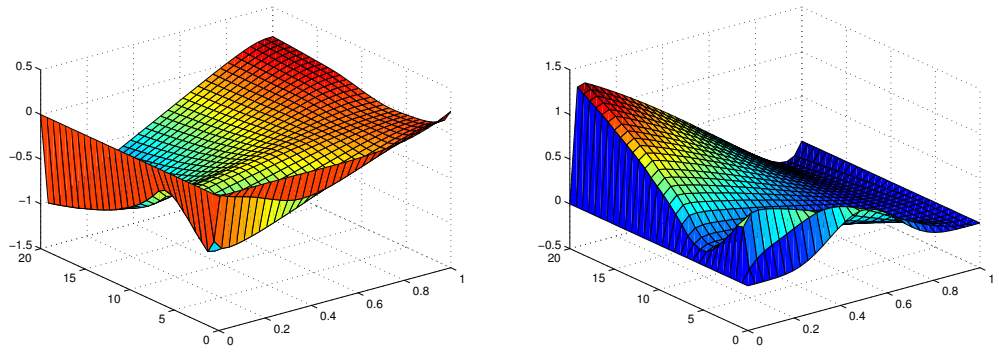


Figure 3.9: Plot of Error Compared to $Re = 100$ Benchmark Solution

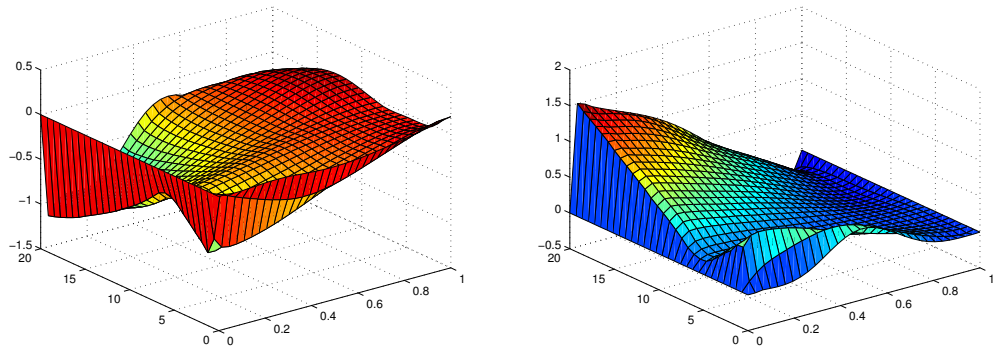
Remarkably, the basis performs well even for computations with varying Reynolds numbers. Note that a low Reynolds number in Burgers equation corresponds to a less steep gradient in the velocity. Therefore, the basis is not subject to change so drastically in ranges of low Reynolds numbers. The reader should notice that it indeed matters if we reproduce a system with a high Reynolds number with a basis obtained from a low Reynolds numbers or vice versa. The benchmark error is 2.9% of the POD versus the GFE when evaluated with the same Reynolds numbers. However, a change of +20 produces 2% more error than a change of -20, see Table 3.3.

The plots of the GFE solution are given in Figure 3.10 for a Reynolds number of $Re = 140$. Figure 3.11 shows the POD solution of the system with a Reynolds number of $Re = 140$



(a) Velocity $w^N(t,x)$ (b) Temperature $T^N(t,x)$

Figure 3.10: GFE Solutions for $Re = 140$



(a) Velocity $w^P(t,x)$ (b) Temperature $T^P(t,x)$

Figure 3.11: POD Solutions for $Re = 140$ with $Re = 100$ Basis

but using the basis obtained from the $Re = 100$ model. The error is given in Table 3.3 as 16.7%. Indeed, the temperature is well recovered with the 5 basis functions whereas the velocity causes the main contribution to the global error. Although the system is coupled, the Reynolds number effects the velocity way more than the temperature, which one would expect.

A key aspect of the POD method is that it somehow averages the dynamics of the system over time. Another interesting question with respect to the "averaging basis" idea is if the POD provides a good basis even for longer time frames. In another numerical example with the same setup as above, we deferred the final time to $t_f = 100s$ and followed the above procedure. The results are displayed in Table 3.4. A plot of these results is given in Figure 3.12. Plots of the GFE solutions based on a Reynolds number $Re = 180$ are shown in Figure

3.13. Moreover, we obtained the POD solutions of the coupled Burgers equation governed by a $Re = 180$ but projected on a basis obtained from a $Re = 100$ simulation as in Figure 3.14.

Re	80	100	120	150	180
Error	0.0748	0.0561	0.0553	0.0916	0.1688

Table 3.4: Global Error between POD and GFE model for various Reynolds numbers and $t_f = 100s$

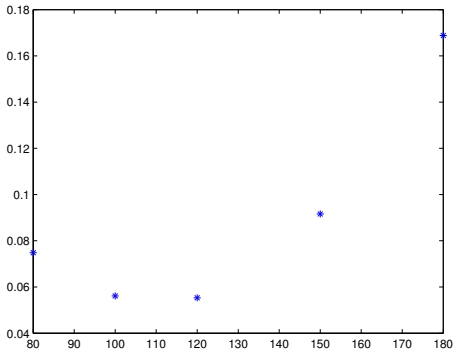
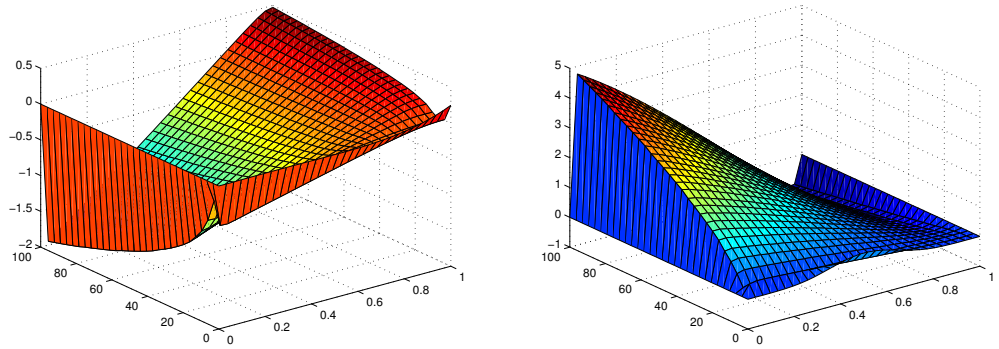


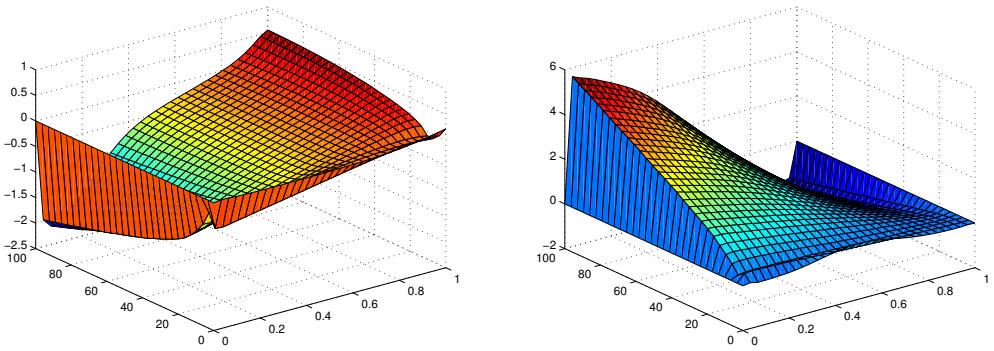
Figure 3.12: Plot of Error Compared to $Re = 100$ Benchmark Solution with $t_f = 100s$



(a) Velocity $w^N(t,x)$ (b) Temperature $T^N(t,x)$

Figure 3.13: GFE Solutions for $Re = 180$ and $t_f = 100s$

The reader should compare Figures 3.13 - 3.14 to Figures 3.10 - 3.11. Observe that the velocity and the temperature field are very smooth and do not show steep gradients when time progresses. Hence, the POD model captured the dynamical system pretty well. Indeed, comparing Table 3.4 and 3.3, we see that the increase of the final time causes a bigger error



(a) Velocity $w^P(t,x)$ (b) Temperature $T^P(t,x)$

Figure 3.14: POD Solutions for $Re = 180$ with $Re = 100$ Basis and $t_f = 100s$

when matching the exact Reynolds number but makes the basis "better" for a range of parameters. Note that a rise of the Reynolds number from 100 to 180 only causes 16.9% of error.

Chapter 4

Conclusions

4.1 Overview of Results

In this work, we studied the coupled Burgers equation as stated in detail in Chapter 1. Two different numerical methods were applied to the coupled Burgers equation: The Finite Element Method (FEM) and the Group Finite Element Method (GFE). We further applied a model reduction method, the Proper Orthogonal Decomposition (POD). The latter builds a very low dimensional model based on the output of either of the former two methods. The use of the GFE is motivated by the results of [Smi97],[Pug95] and [Ngu01]. In these works, an economization of computational cost was observed compared to the FEM. Moreover, some data therein have shown that the GFE produces more realistic, since more stable results. We conducted several numerical simulations with the following parameters: Reynolds numbers $Re = 60, 120, 240$, numbers of internal nodes of $N = 8, 16, 32, 64$, a thermal conductivity $c = 0.01$ and the coefficient of thermal expansion as $\kappa = 1.0$. Further, all these simulations have been executed with the MATLAB ® ODE solvers ODE45, ODE23 and the stiff solver ODE15s, respectively.

Using the Method of Manufactured Solutions (MMF), we verified for the solvers ODE45, ODE23 that both the FEM and the GFE converged. Furthermore, the FEM was often more accurate. A drastic decrease in CPU time was observed for both methods when using ODE15s. However, the time solver ODE15s failed to converge when applied to the FEM approximate model. These results confirm the work of the authors mentioned above with regards to convergence of the methods[1]. Remarkably, no trend of improvement in computational efficiency could be seen.

Since FE/GFE methods can be expensive, in Chapter 3 we presented a POD based model

[1]However, in all of the three works only a ODE45 solver has been used

reduction scheme. This was done using the grouped form of the standard POD. The need for low dimensional models is motivated by the desire to receive rapid solutions for control problems. Secondly, the recent work of [DS10] on Burgers equation has shown that (motivated by the GFE) the Group POD is "better" than the standard POD with regards to computational savings. Hence, we used Group POD to construct a low dimensional approximation of the coupled Burgers equation. We investigated the feasibility of this method to the coupled problem.

The Group POD method was performed based on the provided data from the GFE simulations. At this point, 150 snapshots of the approximate GFE solution have been taken. The simulations used the parameters $Re = 120, N = 150, c = 0.01, \kappa = 1.0, t_f = 20s$ and we specified initial conditions and a forcing function. Accordingly, the POD solution was obtained and showed good results. This is, the CPU times for the actual solution of the ODE system could be reduced by a factor of 10 while being 2.1% close to the GFE solution. Moreover, there have been no observations of non-convergence.

The issue of the sensitivity of the POD basis to parameter values has been addressed in the last section of Chapter 3. A POD basis was obtained for a Reynolds number $Re = 100$ and specified parameters c, κ, boundary and initial conditions. The dynamical system was projected onto this POD basis and afterwards, the Reynolds number was varied. The solutions have been recorded and compared to the full model GFE solutions to the respective Reynolds numbers. For instance, a change of the Reynolds number to $Re = 150$ caused a relative error of 9.2% where the POD solution was computed with the basis obtained from the $Re = 100$ solution.

4.2 Conclusions

The motivation for this research is to develop computational tools that can be used for design, optimization and control of energy efficient buildings. As outlined in the introduction, the coupled Burgers equation is a simple model that mimics the coupling of fluid flow to thermal dynamics. As a computational tool, the Group POD method worked well for the coupled Burgers equation. In this work, we have seen similar numerical results for the coupled Burgers equation that [Smi97], [Pug95] and [Ngu01] showed for the single Burgers equation. Thus, we are inclined to choose the Group Finite Element Method over the standard Finite Element Method as a model reduction technique for high-fidelity or full-model solutions. The numerical results in this work suggest that the Group POD method produces an accurate low dimensional approximation of the coupled Burgers equation. We have seen that only 3 to 5 basis functions were necessary to produce a fairly accurate and quickly solvable reduced order model. This is especially relevant in control problems for controllers have to deal with a variety of different states. Thus, the basis for the controller should cover a range

of parameters (e.g. the Reynolds number, the coefficient of the thermal expansion, initial conditions, forcing function, etc.). Since the Group POD provided satisfactory results, it is cheap to run it for several parameters. The numerical results suggest that the POD model performs fairly well even when the parameters change and the basis is not perfectly tailored to the parameters of the system. Hence, the POD model can be applied to systems where the governing parameters are subject to change.

4.3 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. Perhaps, one could be interested in the following:

- Can one prove the improved stability of the GFE compared to the FEM?
- To develop a rigorous convergence theory for the GFE method
- Designing a control based on the POD model and compare effectiveness to the control problem for the coupled Burgers equation
- Using the POD method for several parameter ranges, initial conditions, forcing functions and time dependent boundary conditions to construct a POD basis that covers a variety of states of the system
- Investigate theoretical properties of the control model such as controllability, stabilizability and observability

Bibliography

[ABG+02] E. Allen, J. Burns, D. Gilliam, J. Hill, and V. Shubov. The impact of finite precision arithmetic and sensitivity on the numerical solution of partial differential equations. *Math. Comput. Modelling*, 35(11-12):1165–1195, 2002.

[ABG08] E. Allen, J.A. Burns, and D. Gilliam. On the use of numerical methods for analysis and control of nonlinear convective systems. In *Proceedings 47th IEEE Conference on Decision and Control*, pages 197–202, December 2008.

[BBGS98] J. Burns, A. Balogh, D. S. Gilliam, and V. I. Shubov. Numerical stationary solutions for a viscous Burgers' equation. *J. Math. Systems Estim. Control*, 8(2):16 pp. (electronic), 1998.

[BBSZ09] J. Borggaard, J.A. Burns, A. Surana, and L. Zietsman. Control, estimation and optimization of energy efficient buildings. In *Proceedings of the 2009 American Control Conference*, 2009.

[BGS01] A. Balogh, D. S. Gilliam, and V. I. Shubov. Stationary solutions for a boundary controlled Burgers' equation. *Math. Comput. Modelling*, 33(1-3):21–37, 2001. Computation and control, VI (Bozeman, MT, 1998).

[BMNP04] Maxime Barrault, Yvon Maday, Ngoc Cuong Nguyen, and Anthony T. Patera. An 'empirical interpolation' method: application to efficient reduced-basis discretization of partial differential equations. *C. R. Math. Acad. Sci. Paris*, 339(9):667–672, 2004.

[BP72] Edward R. Benton and George W. Platzman. A table of solutions of the one-dimensional Burgers equation. *Quart. Appl. Math.*, 30:195–212, 1972.

[BS01] J.A. Burns and J. Singler. *On the Long Time Behavior of Approximating Dynamical Systems*. Distributed Parameter Control. Springer-Verlag, 2001.

[Bur48] J. M. Burgers. A mathematical model illustrating the theory of turbulence. In *Advances in Applied Mechanics*, pages 171–199. Academic Press Inc., New York, N. Y., 1948. edited by Richard von Mises and Theodore von Kármán,.

[Col51] Julian D. Cole. On a quasi-linear parabolic equation occurring in aerodynamics. *Quart. Appl. Math.*, 9:225–236, 1951.

[CT00] C. Cao and E. Titi. Asymptotic behavior of viscous burgers equations with neumann boundary conditions. volume 22 of *SIAM J. Sci. Computing*, pages 368–385. 2000.

[DS10] Benjamin T. Dickinson and John R. Singler. Nonlinear model reduction using group proper orthogonal decomposition. *Int. J. Numer. Anal. Model.*, 7(2):356–372, 2010.

[Fle82] Clive A. J. Fletcher. Burgers' equation: a model for all reasons. In *Numerical solutions of partial differential equations (Parkville, 1981)*, pages 139–225. North-Holland, Amsterdam, 1982.

[Fle83] C. A. J. Fletcher. The group finite element formulation. *Comput. Methods Appl. Mech. Engrg.*, 37(2):225–244, 1983.

[GGS06] Sigal Gottlieb, David Gottlieb, and Chi-Wang Shu. Recovering high-order accuracy in WENO computations of steady-state hyperbolic systems. *J. Sci. Comput.*, 28(2-3):307–318, 2006.

[GK97] M. Garbey and H.G. Kaper. Asymptotic-numerical study of supersensitivity for generalized burgers' equation. *Numer. Funct. Anal. Optim.*, 18(1-2):143–188, 1997.

[GPS07] Max D. Gunzburger, Janet S. Peterson, and John N. Shadid. Reduced-order modeling of time-dependent PDEs with multiple parameters in the boundary data. *Comput. Methods Appl. Mech. Engrg.*, 196(4-6):1030–1047, 2007.

[HGG07] Jan S. Hesthaven, Sigal Gottlieb, and David Gottlieb. *Spectral methods for time-dependent problems*, volume 21 of *Cambridge Monographs on Applied and Computational Mathematics*. Cambridge University Press, Cambridge, 2007.

[Hop50] Eberhard Hopf. The partial differential equation $u_t + uu_x = \mu u_{xx}$. *Comm. Pure Appl. Math.*, 3:201–230, 1950.

[KCGH07] Alex Kanevsky, Mark H. Carpenter, David Gottlieb, and Jan S. Hesthaven. Application of implicit-explicit high order Runge-Kutta methods to discontinuous-Galerkin schemes. *J. Comput. Phys.*, 225(2):1753–1781, 2007.

[Lig56] M. J. Lighthill. Viscosity effects in sound waves of finite amplitude. In *Surveys in mechanics*, pages 250–351 (2 plates). Cambridge, at the University Press, 1956.

[LO93] Jacques G. Laforgue and Robert E. O'Malley, Jr. Supersensitive boundary value problems. In *Asymptotic and numerical methods for partial differential equations with critical parameters (Beaune, 1992)*, volume 384 of *NATO Adv. Sci. Inst. Ser. C Math. Phys. Sci.*, pages 215–223. Kluwer Acad. Publ., Dordrecht, 1993.

[Loè55] Michel Loève. *Probability theory. Foundations. Random sequences.* D. Van Nostrand Company, Inc., Toronto-New York-London, 1955.

[Ngu01] V.Q. Nguyen. *A numerical study of Burger's equation with Robin boundary conditions*, 2001.

[Pug95] S.M. Pugh. *Finite element approximations of Burgers' equation* , 1995.

[Roa04] Patrick J. Roache. Code verification by the method of manufactured solutions. *Transactions of the ASME*, 124:4–10, March 2004.

[Sir87] Lawrence Sirovich. Turbulence and the dynamics of coherent structures. I. Coherent structures. *Quart. Appl. Math.*, 45(3):561–571, 1987.

[Smi97] L.C. Smith. *Finite element approximations of Burgers' equation with Robin's boundary conditions*, 1997.

[Vol08] S. Volkwein. Model reduction using proper orthogonal decomposition. 2008.

Appendix A

FEM Matrices

For the declined reader we provide additional material that is helpful for computations. The entries of the mass matrix M_w^N of the FEM are calculated for $i, j = 1, .., N+1$ as

$$\begin{aligned}\int_0^1 \phi_i(x)\phi_i(x) &= \int_{x_{i-1}}^{x_i} (N+1)^2(x-x_{i-1})^2 dx + \int_{x_i}^{x_{i+1}} (N+1)^2(x-x_{i+1})^2 dx \\ &= (N+1)^2 \left[\left[\frac{(x-x_{i-1})^3}{3} \right]_{x_{i-1}}^{x_i} + \left[\frac{(x-x_{i+1})^3}{3} \right]_{x_i}^{x_{i+1}} \right] \\ = (N+1)^2 \left[\left(\frac{(i-(i-1))^3}{3(N+1)^3} - \frac{(i-(i+1))^3}{3(N+1)^3} \right] \right] &= \frac{1}{(N+1)} \left(\frac{1}{3} + \frac{1}{3} \right) = \frac{4}{6} \frac{1}{N+1} \ .\end{aligned}$$

Further, the mass matrix has entries in the first upper and first lower diagonal, since ϕ_i and ϕ_{i+1} (and symmetrically) have a nonzero product mutually. This results in

$$\int_0^1 \phi_i(x)\phi_{i+1}(x) = \int_{x_i}^{x_{i+1}} -(N+1)^2(x-x_{i+1})(x-x_i)dx = \frac{1}{6}\frac{1}{N+1} \ .$$

The calculations for the nonlinear coupling vector F are provided below. Note that after fixing the value of the derivative the remaining integral terms can be obtained from the entries of the mass matrix. Setting $j = 1$ the calculations yield

$$\begin{aligned}&\alpha_1 \sum_{k=1}^{N+1} \alpha_k \int_0^1 \phi_1^2 \phi_k' dx + \alpha_2 \sum_{k=1}^{N+1} \alpha_k \int_0^1 \phi_2 \phi_k' \phi_1 dx \\ &\qquad = 0\alpha_1^2 + \frac{2}{6}\alpha_1\alpha_2 - \frac{1}{6}\alpha_2\alpha_1 + \frac{1}{6}\alpha_2^2 = \frac{1}{6}\left(\alpha_1\alpha_2 + \alpha_2^2\right) \ .\end{aligned}$$

Further, the case $j = 2$ delivers

$$\sum_{i=1}^{N+1}\sum_{k=1}^{N+1}\int_0^1 \alpha_i(t)\alpha_k(t)\phi_i(x)\phi_k'(x)\phi_2(x)dx$$
$$= \alpha_1\sum_{k=1}^{N+1}\alpha_k\int_0^1 \phi_1\phi_k'\phi_2 dx + \alpha_2\sum_{k=1}^{N+1}\alpha_k\int_0^1 \phi_2^2\phi_k' dx + \alpha_3\sum_{k=1}^{N+1}\alpha_k\int_0^1 \phi_3\phi_k'\phi_2 dx$$
$$= -\frac{1}{6}\alpha_1^2 + \frac{1}{6}\alpha_1\alpha_2 - \frac{2}{6}\alpha_2\alpha_1 + 0\alpha_2^2 + \frac{2}{6}\alpha_2\alpha_3 - \frac{1}{6}\alpha_3\alpha_2 + \frac{1}{6}\alpha_3^2$$
$$= \frac{1}{6}\left(-\alpha_1^2 - \alpha_1\alpha_2 + \alpha_2\alpha_3 + \alpha_3^2\right) .$$

For $j = N$ we get

$$\alpha_{N-1}\sum_{k=1}^{N+1}\alpha_k\int_0^1 \phi_{N-1}\phi_k'\phi_N dx + \alpha_N\sum_{k=1}^{N+1}\alpha_k\int_0^1 \phi_N^2\phi_k' dx + \alpha_{N+1}\sum_{k=1}^{N+1}\alpha_k\int_0^1 \phi_{N+1}\phi_k'\phi_N dx$$
$$= -\frac{1}{6}\alpha_{N-1}^2 + \frac{1}{6}\alpha_N\alpha_{N-1} - \frac{2}{6}\alpha_N\alpha_{N-1} + 0\alpha_N^2 + \frac{2}{6}\alpha_N\alpha_{N+1} - \frac{1}{6}\alpha_{N+1}\alpha_N + \frac{1}{6}\alpha_{N+1}^2$$
$$= \frac{1}{6}\left(-\alpha_{N-1}^2 - \alpha_N\alpha_{N-1} + \alpha_N\alpha_{N+1} + \alpha_{N+1}^2\right) .$$

Likewise, for $j = N + 1$ we obtain

$$\alpha_N\sum_{k=1}^{N+1}\alpha_k\int_0^1 \phi_N\phi_k'\phi_{N+1}dx + \alpha_{N+1}\sum_{k=1}^{N+1}\alpha_k\int_0^1 \phi_{N+1}^2\phi_k' dx$$
$$= -\frac{1}{6}\alpha_N^2 + \frac{1}{6}\alpha_N\alpha_{N+1} - \frac{2}{6}\alpha_{N+1}\alpha_N + \frac{2}{6}\alpha_{N+1}^2 = \frac{1}{6}\left(-\alpha_N^2 - \alpha_N\alpha_{N+1} + 2\alpha_{N+1}^2\right) .$$

We turn our attention to the nonlinear coupling term in the heat equation (which is $w(t,x)T_x(t,x)$), see (2.14). Frequent use of the fact that $\phi_i = \psi_i$ for $i = 1,..,N$ allows us to use the above results. Consider

$$\sum_{i=1}^{N+1}\sum_{k=1}^{N}\int_0^1 \alpha_i(t)\beta_k(t)\phi_i(x)\psi_k'(x)\psi_l(x)dx \qquad \text{for } l = 1,..,N$$

In the case $l = 1$

$$\alpha_1\sum_{k=1}^{N}\beta_k\int_0^1 \phi_1\psi_k'\psi_1 dx + \alpha_2\sum_{k=1}^{N}\beta_k\int_0^1 \phi_2\psi_k'\psi_1 dx$$
$$= 0\alpha_1\beta_1 + \frac{2}{6}\alpha_1\beta_2 - \frac{1}{6}\alpha_2\beta_1 + \frac{1}{6}\alpha_2\beta_2 = \frac{1}{6}\left(2\alpha_1\beta_2 - \alpha_2\beta_1 + \alpha_2\beta_2\right) .$$

Further, the case $l = 2$ delivers

$$\alpha_1 \sum_{k=1}^{N} \beta_k \int_0^1 \phi_1 \psi_k' \psi_2 dx + \alpha_2 \sum_{k=1}^{N} \beta_k \int_0^1 \phi_2 \psi_k' \psi_2 dx + \alpha_3 \sum_{k=1}^{N} \beta_k \int_0^1 \phi_3 \psi_k' \psi_2 dx$$
$$= -\frac{1}{6}\alpha_1\beta_1 + \frac{1}{6}\alpha_1\beta_2 - \frac{2}{6}\alpha_2\beta_1 + 0\alpha_2\beta_2 + \frac{2}{6}\alpha_2\beta_3 - \frac{1}{6}\alpha_3\beta_2 + \frac{1}{6}\alpha_3\beta_3$$
$$= \frac{1}{6}\left(-\alpha_1\beta_1 + \alpha_1\beta_2 - 2\alpha_2\beta_1 + 2\alpha_2\beta_3 - \alpha_3\beta_2 + \alpha_3\beta_3\right) \ .$$

For $l = N - 1$ we obtain the result

$$\alpha_{N-2} \sum_{k=1}^{N} \beta_k \int_0^1 \phi_{N-2} \psi_k' \psi_{N-1} dx + \alpha_{N-1} \sum_{k=1}^{N} \beta_k \int_0^1 \phi_{N-1} \psi_k' \psi_{N-1} dx + \alpha_N \sum_{k=1}^{N} \beta_k \int_0^1 \phi_N \psi_k' \psi_{N-1} dx$$
$$= -\frac{1}{6}\alpha_{N-2}\beta_{N-2} + \frac{1}{6}\alpha_{N-2}\beta_{N-1} - \frac{2}{6}\alpha_{N-1}\beta_{N-2} + 0\alpha_{N-1}\beta_{N-1} + \frac{2}{6}\alpha_{N-1}\beta_N - \frac{1}{6}\alpha_N\beta_{N-1} + \frac{1}{6}\alpha_N\beta_N$$
$$= \frac{1}{6}\left(-\alpha_{N-2}\beta_{N-2} + \alpha_{N-2}\beta_{N-1} - 2\alpha_{N-1}\beta_{N-2} + 2\alpha_{N-1}\beta_N - \alpha_N\beta_{N-1} + \alpha_N\beta_N\right) \ .$$

Finally, $l = N$ produces the equation

$$\alpha_{N-1} \sum_{k=1}^{N} \beta_k \int_0^1 \phi_{N-1} \psi_k' \psi_N dx + \alpha_N \sum_{k=1}^{N} \beta_k \int_0^1 \phi_N \psi_k' \psi_N dx + \alpha_{N+1} \sum_{k=1}^{N} \beta_k \int_0^1 \phi_{N+1} \psi_k' \psi_N dx$$
$$= -\frac{1}{6}\alpha_{N-1}\beta_{N-1} + \frac{1}{6}\alpha_{N-1}\beta_N - \frac{2}{6}\alpha_N\beta_{N-1} + 0\alpha_N\beta_N - \frac{1}{6}\alpha_{N+1}\beta_N$$
$$= \frac{1}{6}\left(-\alpha_{N-1}\beta_{N-1} + \alpha_{N-1}\beta_N - 2\alpha_N\beta_{N-1} - \alpha_{N+1}\beta_N\right) \ .$$

Some calculations for the stiffness matrix S_w^N are provided below, where $i = 1, .., N$:

$$S_{ii} = \int_0^1 \phi_i'^2(x)dx = \int_{x_{i-1}}^{x_i} (N+1)^2 dx + \int_{x_i}^{x_{i+1}} (N+1)^2 dx = -(N+1)^2 \frac{2}{N+1} = 2(N+1) \ ,$$

$$S_{N+1,N+1} = \int_0^1 \phi_{N+1}'^2(x)dx = \int_{x_N}^{x_{N+1}} (N+1)^2 dx = (N+1)^2 \frac{1}{N+1} = (N+1) \ ,$$

$$S_{i,i+1} = \int_0^1 \phi_i'(x)\phi_{i+1}'(x)dx = \int_{x_i}^{x_{i+1}} (N+1)(-1)(N+1)(+1)dx = -(N+1) = S_{i-1,i} \ .$$

The last result holds due to symmetry.

Appendix B

Matlab ® Source Code

In this section the MATLAB ® source code is provided that was used in chapter two. The Main file for the GFE method and all its functions are provided.

```
% ————————————————————GFE MAIN FILE————————————————————————————————
% This file is the Main file for the GFE Method. It is used for the
% computations in chapter 2 (Method of Manufactured Solutions) where it
% calls initial conditions w0a, T0a and the vector F_N_Hat of the analytic
% forcing functions integrated against the respective basis functions phi
% and psi. In chapter 3 (GFE compared to POD) this file is also used. One
% needs to replace w0a by w0, T0a by T0 and set f1=0 and provide a forcing
% function f2. We did this with a separate fct F_N instead of F_N_Hat.
% Also, tfinal=20 for POD, tfinal=15 for MMF. We wrote a separate file
% gfe_pod to do that, this just includes all changes mentioned above.
% This file can also be used for the FEM, when replacing the call of gferhs
% by femrhs in the input of the ODE solver. Moreover, the CPU times
% have been measured at different places for different purposes.

clear all
%time1=cputime;           % This was within MMF in chapter 2

% Initializing the parameters
global mu c kappa g3p h N node
a=0;
b=1;
delta=0;                  % Value of the Neumann BC
Re=input('Enter the Reynolds number Re   ');
mu=1/Re;                  % Viscosity coefficient
c=0.01;                   % Thermal conductivity
kappa=1.0;                % Coefficient of the  thermal expansion
N=input('Enter the number of internal nodes N   ');

% First define the FE: Set the length of every element in the vector h and
```

```
% define the coordinates of the nodes in the vector node.

consth=(b-a)/(N+1);
h=zeros(N+1,1);
for i=1:N+1
    h(i)=consth;          % Step size was set constant. Mesh refinement
end                       % can be done here if necessary

nnodes=N+2;               % Internal nodes plus nodes at bound.= N+2
node=zeros(nnodes,1);
node(1)=a;
for i=2:nnodes
    node(i)=node(i-1)+h(i-1); % Sets coordinates of FE nodes
end

% Determining the three GP within every element. These are used for the
% integration of the initial cond and the forcing fct against the basis.

% 1 Point Gauss Quadrature Points in the center of the element
g1p=zeros(nnodes-1,1);
for i=1:nnodes-1
    g1p(i,1)=node(i)+h(i)/2;
end

% 3 Point Gauss Quadrature Points in every element
g3p=zeros(nnodes-1,3);
for i=1:nnodes-1
    g3p(i,1)=g1p(i,1)-0.7745966692*h(i)/2;
    g3p(i,2)=g1p(i,1);
    g3p(i,3)=g1p(i,1)+0.7745966692*h(i)/2;
end

% Evaluates the value of the piecewise linear basisfunction at the GP.
% Note that every basis fct has the same function value at the 1st GP,
% the 2nd GP and the 3rd GP in the respective element. Thus, one needs
% to calculate 3 function values and use them throughout the code.

global basisg3p
basisg3p=zeros(3,1);
basisg3p(1,1)=basisfct(1,g3p(1,1),node,h);
basisg3p(2,1)=basisfct(1,g3p(1,2),node,h);
basisg3p(3,1)=basisfct(1,g3p(1,3),node,h);

% The matrices for the linear parts are calculated
global Sglob Mhat deltahat

% Mass Matrix
d=[4*ones(N,1); 2]; d_1=ones(N+1,1);
MM=(1/(6*(N+1)))*spdiags([d_1,d,d_1],[-1,0,1],N+1,N+1);
Mhat=[MM,zeros(N+1,N);zeros(N,N+1),MM(1:N,1:N)];
```

```
% Stiffness Matrix
D=[2*ones(N,1);1]; D_1=-ones(N+1,1);
SS=(N+1)*spdiags([D_1, D ,D_1],[-1,0,1],N+1,N+1);
Sglob=[mu*SS, kappa*MM(1:N+1,1:N);  zeros(N,N+1), c*SS(1:N,1:N)];

% Boundary condition vector
deltahat=[zeros(N,1); delta; zeros(N,1)];

% Matrix AP to calculate the nonlinearity in the GFE method
global AP
dhelp=0.5*ones(N+1,1); dm=[zeros(N,1); 0.5];
AP=spdiags([-dhelp dm dhelp],[-1 0 1],N+1,N+1);

% Gauss integration of initial conditions times the resp. test functions
w_0=zeros(N+1,1);
for nel=1:N
  w_0(nel,1)=w_0(nel,1)+w0a(g3p(nel,1))*basisg3p(1,1)*h(nel)/2*0.55555556;
  w_0(nel,1)=w_0(nel,1)+w0a(g3p(nel,2))*basisg3p(2,1)*h(nel)/2*0.88888889;
  w_0(nel,1)=w_0(nel,1)+w0a(g3p(nel,3))*basisg3p(3,1)*h(nel)/2*0.55555556;

  w_0(nel,1)=w_0(nel,1)+w0a(g3p(nel+1,1))*basisg3p(3,1)*h(nel+1)/2*0.55555556;
  w_0(nel,1)=w_0(nel,1)+w0a(g3p(nel+1,2))*basisg3p(2,1)*h(nel+1)/2*0.88888889;
  w_0(nel,1)=w_0(nel,1)+w0a(g3p(nel+1,3))*basisg3p(1,1)*h(nel+1)/2*0.55555556;
end

  w_0(N+1,1)=w_0(N+1,1)+w0a(g3p(N+1,1))*basisg3p(1,1)*h(N+1)/2*0.55555556;
  w_0(N+1,1)=w_0(N+1,1)+w0a(g3p(N+1,2))*basisg3p(2,1)*h(N+1)/2*0.88888889;
  w_0(N+1,1)=w_0(N+1,1)+w0a(g3p(N+1,3))*basisg3p(3,1)*h(N+1)/2*0.55555556;

T_0=zeros(N,1);
for nel=1:N
  T_0(nel,1)=T_0(nel,1)+T0a(g3p(nel,1))*basisg3p(1,1)*h(nel)/2*0.55555556;
  T_0(nel,1)=T_0(nel,1)+T0a(g3p(nel,2))*basisg3p(2,1)*h(nel)/2*0.88888889;
  T_0(nel,1)=T_0(nel,1)+T0a(g3p(nel,3))*basisg3p(3,1)*h(nel)/2*0.55555556;

  T_0(nel,1)=T_0(nel,1)+T0a(g3p(nel+1,1))*basisg3p(3,1)*h(nel+1)/2*0.55555556;
  T_0(nel,1)=T_0(nel,1)+T0a(g3p(nel+1,2))*basisg3p(2,1)*h(nel+1)/2*0.88888889;
  T_0(nel,1)=T_0(nel,1)+T0a(g3p(nel+1,3))*basisg3p(1,1)*h(nel+1)/2*0.55555556;
end

init=Mhat\[w_0;T_0];

% Set up the input for the solver. The solvers have been changed to ode23,
% ode45 in chapter 2
global tfinal t_points tspan
tfinal=15; t_points=200;
tspan=linspace(0,tfinal,t_points);
fprintf('%d points were used for the time grid \n',t_points);
fprintf=('The solver was ode15s');

% Solving the ODE system
```

```
time1=cputime;        % Time measurement to compare with POD in chapter 3
[T,Y]=ode15s(@gferhs,tspan,init);
time2=cputime;

%-------------------------Postprocessing-----------------------------------------
% A meshgrid is generated to produce the 2D plots. Then absolute Errors are
% ploted globally and at the final time.
x=linspace(0,1,32);
[X,Z]=meshgrid(x,tspan);
subplot(2,2,1)
surf(X,Z,soltempmesh(Y,X,Z)-knowntemp(X,Z));
title('Error in Temperature');
subplot(2,2,2)
surf(X,Z,solflowmesh(Y,X,Z)-knownflow(X,Z));
title('Error in Velocity');
subplot(2,2,3)
plot(x,abs(soltemp(Y,x,tfinal)-knowntemp(x,tfinal)'),'r')
title('Error in Temperature at tfinal');
subplot(2,2,4)
plot(x,abs(solflow(Y,x,tfinal)-knownflow(x,tfinal)'),'b')
title('Error in Velocity  at tfinal');

% The first two scripts set the Gauss points for given vectors x and t of
% nodal positions. If not reset, it will use tspan and x from above.
% Then the global L2 error will be displayed.
g3pspace
g3ptimescript
globL2err
time3=cputime;
cputime=time2-time1
Postprocessingtime=time3-time2


function y = basisfct(j,x,node,h)
% This function generates the basis functions and evaluates them.
% y gives the value of the j th basis function at position x.

help=0;
y=0;
% Note that max(size(node))=N+2

% First the basis function for j=N+1 is defined
if j==max(size(node))-1

    if help==0
       if x<node(j)
        y=0;
        help=1;
       end
    end
```

```
    if help==0
        y=(x-node(j))/h(j);
        help=1;
    end
end

% Define the hat functions for j=1,..,N. Be careful: the ith basis fct.
% is nonzero only in [node(i),node(i+2)] and symmetric.

if help==0
    if x<node(j)
        y=0;
        help=1;
    end
end

if help==0
    if x<node(j+1)
        y=(x-node(j))/h(j);
        help=1;
    end
end

if help==0
    if x<node(j+2)
        y=(node(j+2)-x)/h(j+1);
        help=1;
    end
end

end


function y = basisfctprime(j,x,node,h)
% This function generates the derivatives of basisfct.m and evaluates them.
% y gives the value of the j th basis function at position x.

help=0;
y=0;
% We start with j=N+1.

if j==max(size(node))-1

    if help==0
       if x<node(j)
        y=0;
        help=1;
       end
    end

    if help==0
```

```
        y=1/h(j);
        help=1;
    end
end

% We set it for  j=1,..,N. Be careful: the jth basis funtion has node(j+1)
% in the center.
if help==0
    if x<node(j)
        y=0;
        help=1;
    end
end

if help==0
    if x<node(j+1)
        y=1/h(j);
        help=1;
    end
end

if help==0
    if x<node(j+2)
        y=-1/h(j+1);
        help=1;
    end
end

end


function y = F_N_Hat(t)
% This file is especially for the Method of Manufactured Sol, where we have
% two forcing functions. For other simulations where f1=0 we wrote a
% function called F_N(t).
% Integrates f1(x,t), f2(x,t) against the respective basis fct with 3P Gauss
% and assembles the global forcing funtion F_N_hat as a funtion of the time.

global N g3p h basisg3p

F_N1=zeros(N+1,1);
for nel=1:N
    F_N1(nel,1)=F_N1(nel,1)+f1(g3p(nel,1),t)*basisg3p(1,1)*h(nel)/2*0.55555556;
    F_N1(nel,1)=F_N1(nel,1)+f1(g3p(nel,2),t)*basisg3p(2,1)*h(nel)/2*0.88888889;
    F_N1(nel,1)=F_N1(nel,1)+f1(g3p(nel,3),t)*basisg3p(3,1)*h(nel)/2*0.55555556;

    F_N1(nel,1)=F_N1(nel,1)+f1(g3p(nel+1,1),t)*basisg3p(3,1)*h(nel+1)/2*0.55555556;
    F_N1(nel,1)=F_N1(nel,1)+f1(g3p(nel+1,2),t)*basisg3p(2,1)*h(nel+1)/2*0.88888889;
    F_N1(nel,1)=F_N1(nel,1)+f1(g3p(nel+1,3),t)*basisg3p(1,1)*h(nel+1)/2*0.55555556;
end
```

```
    F_N1(N+1,1)=F_N1(N+1,1)+f1(g3p(N+1,1),t)*basisg3p(1,1)*h(N+1)/2*0.55555556;
    F_N1(N+1,1)=F_N1(N+1,1)+f1(g3p(N+1,2),t)*basisg3p(2,1)*h(N+1)/2*0.88888889;
    F_N1(N+1,1)=F_N1(N+1,1)+f1(g3p(N+1,3),t)*basisg3p(3,1)*h(N+1)/2*0.55555556;


F_N2=zeros(N,1);
for nel=1:N
    F_N2(nel,1)=F_N2(nel,1)+f2(g3p(nel,1),t)*basisg3p(1,1)*h(nel)/2*0.55555556;
    F_N2(nel,1)=F_N2(nel,1)+f2(g3p(nel,2),t)*basisg3p(2,1)*h(nel)/2*0.88888889;
    F_N2(nel,1)=F_N2(nel,1)+f2(g3p(nel,3),t)*basisg3p(3,1)*h(nel)/2*0.55555556;

    F_N2(nel,1)=F_N2(nel,1)+f2(g3p(nel+1,1),t)*basisg3p(3,1)*h(nel+1)/2*0.55555556;
    F_N2(nel,1)=F_N2(nel,1)+f2(g3p(nel+1,2),t)*basisg3p(2,1)*h(nel+1)/2*0.88888889;
    F_N2(nel,1)=F_N2(nel,1)+f2(g3p(nel+1,3),t)*basisg3p(1,1)*h(nel+1)/2*0.55555556;
end

y=[F_N1;F_N2];

end
```

In the following the right hand side of the resulting approximate systems are formulated as functions. The rhs only changes the nonlinearity in the Burgers equation when doing a group formulation.

```
function v = gferhs(t,y)
% This function assembles the Right Hand Side of the GFE approximation.
% It builds the nonlinear part J(alhpa, beta) where we y(i)=alpha(i)
% for i=1,..,N+1 and y(i)=beta(i) for i=N+2,..,2N+1.The nonlinearity in the
% Burgers equation be computed by a matrix vector product.

global  Mhat Sglob deltahat mu

L=max(size(y));         % L=2N+1, so N=(L-1)/2
N=(L-1)/2;              % equivalent to the N from the other parts, but only
J=zeros(L,1);           % known within the function.

J(1:N+1,1)=0.5*AP*(y(1:N+1,1).^2);

% The nonlinearity in Burgers equation for the FEM function femrhs.m
%J(1,1)=y(1)*y(2)+y(2)^2;
%for i=2:N
 %   J(i,1)=-y(i-1)^2-y(i-1)*y(i)+y(i)*y(i+1)+y(i+1)^2;
%end
%J(N+1,1)=-y(N)^2-y(N)*y(N+1)+2*y(N+1)^2;
% J(1:N+1,1)=(1/6)*J(1:N+1,1);

J(N+2,1)=2*y(1)*y(N+3)-y(2)*y(N+2)+y(2)*y(N+3);
for i=(N+3):(2*N)
    J(i,1)=-y(i-(N+2))*y(i-1)+y(i-(N+2))*y(i)-2*y(i-(N+1))*y(i-1)+...
```

```
            2*y(i-(N+1))*y(i+1)-y(i-N)*y(i)+y(i-N)*y(i+1);
end
J(2*N+1,1)=-y(N-1)*y(2*N)+y(N-1)*y(2*N+1)-2*y(N)*y(2*N)-y(N+1)*y(2*N+1);
J(N+2:2*N+1,1)=(1/6)*J(N+2:2*N+1,1);

%v =Mhat\(-J - Sglob*y + F_N(t)+mu*deltahat);  % For the regular GFE/FEM simulation
v =Mhat\(-J - Sglob*y + F_N_Hat(t)+mu*deltahat); % For the MMF

end
```

The two following scripts create Gauss points for the time and space integration following a three point Gauss quadrature rule.

```
%-----------------------------g3pspace.m-------------------------------------
% Creates the 3 GP in [x(i-1),x(i)] for a given vector of x and stores them
% in g3px. The vector xstep contains the length of the resp. intervals.

global xstep Lx g3px

Lx=max(size(x))-1;
xstep=zeros(Lx,1);
for i=1:Lx
    xstep(i)=x(i+1)-x(i);
end

% 1 Point Gauss Quadrature Points are in the center of each interval.
g1px=zeros(Lx,1);
for i=1:Lx
    g1px(i,1)=x(i)+(xstep(i)/2);
end

% 3 Point Gauss Quadrature Points
g3px=zeros(Lx,3);
for i=1:Lx
    g3px(i,1)=g1px(i,1)-0.7745966692*(xstep(i)/2);
    g3px(i,2)=g1px(i,1);
    g3px(i,3)=g1px(i,1)+0.7745966692*(xstep(i)/2);
end
```

```
%-----------------------------g3ptimescript.m--------------------------------
% Creates 3 GP in [tspan(i-1),tspan(i)] for a given vector tspan and
% stores them in g3ptime. The vector tstep contains the length
% of the resp. intervals.

Lt=t_points-1;
tstep=zeros(Lt,1);
for i=1:Lt
    tstep(i)=tspan(i+1)-tspan(i);
```

```
end

% 1 Point Gauss Quadrature Points are in the center of the interval.
g1ptime=zeros(Lt,1);
for i=1:Lt
    g1ptime(i,1)=tspan(i)+(tstep(i)/2);
end

% 3 Point Gauss Quadrature Points
g3ptime=zeros(Lt,3);
for i=1:Lt
    g3ptime(i,1)=g1ptime(i,1)-0.7745966692*(tstep(i)/2);
    g3ptime(i,2)=g1ptime(i,1);
    g3ptime(i,3)=g1ptime(i,1)+0.7745966692*(tstep(i)/2);
end
```

The computations of the L2 errors is done with the following script and two functions.

```
%------------------------------globL2err.m-------------------------------------
% Calculates the relative L2 error globally (time and space).
% Calls L2err(Y,t) to obtain the L2 error at every GP in time and
% then does the time integration with these values.
% GP for space are independent of N and created within 33 elements.
% L2norm just calculates the norm instead of the error.

err=0;
for i=1:Lt
    err=err+L2err(Y,g3ptime(i,1))^2*(tstep(i))/2*0.55555556;
    err=err+L2err(Y,g3ptime(i,2))^2*(tstep(i))/2*0.88888889;
    err=err+L2err(Y,g3ptime(i,3))^2*(tstep(i))/2*0.55555556;
end

err=sqrt(err);

norm=0;
for i=1:Lt
    norm=norm+L2norm(g3ptime(i,1))^2*(tstep(i))/2*0.55555556;
    norm=norm+L2norm(g3ptime(i,2))^2*(tstep(i))/2*0.88888889;
    norm=norm+L2norm(g3ptime(i,3))^2*(tstep(i))/2*0.55555556;
end
norm=sqrt(norm);

globerr=err/norm


function x = L2err(Y,t)
% For fixed t we integrate the difference function between the known
% solution and the calculated solution in space with a three point gauss
% quadrature rule.
```

```
global xstep Lx g3px
% The error is absolute and sums up the temperature and velocity error.
y=0;
for nel=1:Lx
    y=y+abs(solflow(Y,g3px(nel,1),t)-knownflow(g3px(nel,1),t))^2*xstep(nel)/2*0.55555556;
    y=y+abs(solflow(Y,g3px(nel,2),t)-knownflow(g3px(nel,2),t))^2*xstep(nel)/2*0.88888889;
    y=y+abs(solflow(Y,g3px(nel,3),t)-knownflow(g3px(nel,3),t))^2*xstep(nel)/2*0.55555556;

end
y=sqrt(y);

z=0;
for nel=1:Lx
    z=z+abs(soltemp(Y,g3px(nel,1),t)-knowntemp(g3px(nel,1),t))^2*xstep(nel)/2*0.55555556;
    z=z+abs(soltemp(Y,g3px(nel,2),t)-knowntemp(g3px(nel,2),t))^2*xstep(nel)/2*0.88888889;
    z=z+abs(soltemp(Y,g3px(nel,3),t)-knowntemp(g3px(nel,3),t))^2*xstep(nel)/2*0.55555556;

end
z=sqrt(z);

x=y+z;

end


function x = L2norm(t)
% We integrate both the squared known temperature and the squared known flow
% in space with a three point gauss quadrature rule.
% Hence, L2norm(t) gives the L2 norm at every time t.

global xstep Lx g3px

y=0;
for nel=1:Lx
    y=y+abs(knownflow(g3px(nel,1),t))^2*xstep(nel)/2*0.55555556;
    y=y+abs(knownflow(g3px(nel,2),t))^2*xstep(nel)/2*0.88888889;
    y=y+abs(knownflow(g3px(nel,3),t))^2*xstep(nel)/2*0.55555556;

end
y=sqrt(y);

z=0;
for nel=1:Lx
    z=z+abs(knowntemp(g3px(nel,1),t))^2*xstep(nel)/2*0.55555556;
    z=z+abs(knowntemp(g3px(nel,2),t))^2*xstep(nel)/2*0.88888889;
    z=z+abs(knowntemp(g3px(nel,3),t))^2*xstep(nel)/2*0.55555556;

end
z=sqrt(z);

x=y+z;
```

```
end


function y=solflow(Y,x,t)

global tfinal node h
% t needs to be a scalar, x can be a vector.
% After having solved the time dependent system, we assemble the
% approximate solution by multiplying the respective nodal time dependent
% solutions with the corresponding basis functions and summing it up. t is
% the time at which we want to evaluate the solution

% This assigns every time instance the corresponding index in the time vector
% tspan. e.g if t=tfinal, tint=t_points. In the special case where t=0,
% we set the index to 1, since index of arrays starts at 1.
if t==0
    tint=1;
else
tint=ceil((t/tfinal)*max(size(Y(:,1)))); %ceil rounds to the next integer
end

K=max(size(Y(1,:)));  % K=2N+1 number of time dependent nodal funtions
N=(K-1)/2;
L=length(x(1,:));  % L= # of space points to evaluate the solution
y=zeros(L,1);

for j=1:L
    for i=1:N+1

        y(j,1)=y(j,1)+Y(tint,i).*basisfct(i,x(j),node,h);
    end
    % This for loop has to be replaced by the beyond one for soltemp.m
    %for i=(N+2):K

       %   y(j,1)=y(j,1)+Y(tint,i).*basisfct(i-(N+1),x(j),node,h);
    %end

end

end


function y=solflowmesh(Y,x,t)

global tfinal node h
% This file is especially written for x and t as provided by
% meshgrid(xvector,tvector), so x and t are matrices. Note that when doing
% so, x(j,m)=x(1,m) forall j and t(j,m)=t(j,1)
```

```
lx=length(x(1,:));      % lx= # of space points to evaluate the sol.
lt=length(t(:,1));      % lt= # of time instances to eval. the solution

tint=ceil((t/tfinal)*max(size(Y(:,1))));
tint(1,:)=1;

% tint now contains the node numbers in tspan instead of the values of the
% input matrix t at which we want to construct the solution. ceil rounds
% up to the next integer value towards infinity.

K=max(size(Y(1,:)));  % K=2N+1 number of time dependent nodal functions
N=(K-1)/2;
y=zeros(lt,lx);

for j=1:lt
   for m=1:lx
      for i=1:N+1

          y(j,m)=y(j,m)+Y(tint(j,1),i).*basisfct(i,x(1,m),node,h);
      end
      %The beyond loop has to be replace to get the function soltempmesh.m
      %for i=(N+2):K

       %  y(j,m)=y(j,m)+Y(tint(j,1),i).*basisfct(i-(N+1),x(1,m),node,h);
     %  end
   end
end

end
```

This is the main file for the POD calculations. All functions and scripts are called herein. The scripts and functions are attached subsequently.

```
% ------------------------------POD MAIN FILE-------------------------------------

%------------------------------PREPROCESSING-------------------------------------
tic;
% We start the clock to count the time until the solver starts

% Correlation matrices Kw,KT are computed, herein also their eigenvectors
% and eigenvalues that are needed to compute the respective POD basis.

CorrelationMatrix

% The coefficients \gamma_ik are calculated and stored in the matrix Gamma

PODCoefficients

% Having the coefficients, one can compute the POD basis functions
```

```
% Plot of the first dw, dT basis functions of the POD for temp. and vel.
x=linspace(0,1,32);
for i=1:dw
    hold on
    plot (x,POD_basis_w(x,i))
    title('POD Basisfunction for the Velocity');
    hold off
end

for i=1:dT
    hold on
    plot (x,POD_basis_T(x,i))
    title('POD Basisfunction for the Temperature');
    hold off
end


% For Gauss integration 51 subintervals of [0,1] are defined and the Gauss
% points set herein in the script g3pspace. These are used for the spacial
% integration of the initial conditions and the forcing function f against
% the basis POD_basis_T.
x=linspace(0,1,50);
g3pspace

% As in the GFE Main file, we precompute the values of the POD basis fcts.
% at all Gauss Points g3px

global PODw PODT
PODw=zeros(max(size(g3px)),3*dw);
for j=1:dw
PODw(:,(j-1)*3+1)=POD_basis_w(g3px(:,1),j);
PODw(:,(j-1)*3+2)=POD_basis_w(g3px(:,2),j);
PODw(:,(j-1)*3+3)=POD_basis_w(g3px(:,3),j);
end

% Thus, PODw(nel,(j-1)*dw+1)=POD_basis_w(g3px(nel,1),j) forall nel

PODT=zeros(max(size(g3px)),3*dT);
for j=1:dT
PODT(:,(j-1)*3+1)=POD_basis_T(g3px(:,1),j);
PODT(:,(j-1)*3+2)=POD_basis_T(g3px(:,2),j);
PODT(:,(j-1)*3+3)=POD_basis_T(g3px(:,3),j);
end

% Check if the POD basis funtions are orthonormal. The entries of the
% Matrix Testw,T are L2 scalar products of the respective basis fcts. We
% substract Id to receive the error of the basis from being orthonormal

%TestOrthw=testPOD(dw,dT)
Testw=Gamma(1:N+1,1:dw)'*MM*Gamma(1:N+1,1:dw)-eye(dw)
TestT=Gamma(N+2:2*N+1,1:dT)'*MM(1:N,1:N)*Gamma(N+2:2*N+1,1:dT)-eye(dT)
```

```
% Stiffness Matrix S and matrix C for the Linear parts are computed.

global S
Sdw=Gamma(1:N+1,1:dw)'*mu*SS*Gamma(1:N+1,1:dw);
SdT=Gamma(N+2:2*N+1,1:dT)'*c*SS(1:N,1:N)*Gamma(N+2:2*N+1,1:dT);
C=Gamma(1:N+1,1:dw)' *MM(1:N+1,1:N)*Gamma(N+2:2*N+1,1:dT);
S=[Sdw, kappa*C; zeros(dT,dw), SdT];


% The matrices B, Nhat can be precomputed for the nonlinearities
% We follow the notation from the thesis, Chapter 3

Gammahat=zeros(N+1,dw^2/2+dw/2);
for n=1:N+1
    Gammahelp=(Gamma(n,1:dw)'*Gamma(n,1:dw)).*(ones(dw,dw)+triu(ones(dw,dw),1));
    m=0;
    for i=1:dw
        for j=i:dw
            m=m+1;
               Gammahat(n,m)=Gammahelp(i,j);
        end
    end
end

% The matrix AP is computed in the Main file for the GFE
global Nhat
Nhat=Gamma(1:N+1,1:dw)'*AP*Gammahat;

global B
B=zeros(dT,dw*dT);
for j=1:dT
    m=0;
    for k=1:dw
        for l=1:dT
            m=m+1;
            for nel=1:max(size(g3px))

       B(j,m)=B(j,m)+PODw(nel,(k-1)*3+1)*PODbasisTpri(g3px(nel,1),l)...
             *PODT(nel,(j-1)*3+1)*xstep(nel)/2*0.55555556;
       B(j,m)=B(j,m)+PODw(nel,(k-1)*3+2)*PODbasisTpri(g3px(nel,2),l)...
              *PODT(nel,(j-1)*3+2)*xstep(nel)/2*0.88888889;
       B(j,m)=B(j,m)+PODw(nel,(k-1)*3+3)*PODbasisTpri(g3px(nel,3),l)...
              *PODT(nel,(j-1)*3+3)*xstep(nel)/2*0.55555556;

            end
        end
    end
end

% The initial conditions are integrated against the respective POD basis
```

```
% w0.m,T0.m have to be provided here

initpod=zeros(dw+dT,1);
for k=1:dw
      for nel=1:max(size(g3px))
     initpod(k,1)=initpod(k,1)+w0(g3px(nel,1))*PODw(nel,(k-1)*3+1)...
                              *xstep(nel)/2*0.55555556;
     initpod(k,1)=initpod(k,1)+w0(g3px(nel,2))*PODw(nel,(k-1)*3+2)...
                              *xstep(nel)/2*0.88888889;
     initpod(k,1)=initpod(k,1)+w0(g3px(nel,3))*PODw(nel,(k-1)*3+3)...
                              *xstep(nel)/2*0.55555556;
    end
end

for k=1:dT
    for nel=1:max(size(g3px))
    initpod(k+dw,1)=initpod(k+dw,1)+T0(g3px(nel,1))*PODT(nel,(k-1)*3+1)...
                                *xstep(nel)/2*0.55555556;
    initpod(k+dw,1)=initpod(k+dw,1)+T0(g3px(nel,2))*PODT(nel,(k-1)*3+2)...
                                *xstep(nel)/2*0.88888889;
    initpod(k+dw,1)=initpod(k+dw,1)+T0(g3px(nel,3))*PODT(nel,(k-1)*3+3...
                              )*xstep(nel)/2*0.55555556;

    end
end

%---------------------------------ONLINE COMPUTATION---------------------------------
% The ODE system is solved with the same final time as in gfe_pod.
% Another time measurement is made for the pure ODE solver execution time

toc;
tfinal=20; t_points=N;
tspan=linspace(0,tfinal,t_points);
tic;
[TPOD,YPOD]=ode15s(@podrhs,tspan,initpod);
toc;

%---------------------------------POSTPROCESSING---------------------------------
% Plots of the GFE and POD solutions at t=0 and t=tfinal. We also show
% plots of the deviation between the POD and GFE globally over the time
% space mesh. For the plots the mesh of x,t values was taken coarser to not
% spend to much time for evaluations.

tic;
x=linspace(0,1,32);
t=linspace(0,1,33);
[X,Z]=meshgrid(x,t);
subplot(3,3,1)
plot(x,solflow(Y,x,tfinal),x,flowPOD(x,tfinal,dw,YPOD),'r');
title('Velocity at final time GFE(blue), POD(red)');
subplot(3,2,2)
```

```
plot(x,soltemp(Y,x,tfinal),x,tempPOD(x,tfinal,dT,YPOD),'r');
title('Temp at final time GFE(blue), POD(red)');
subplot(3,2,3)
plot(x,solflow(Y,x,0),x,flowPOD(x,0,dw,YPOD),'r');
title('Velocity at t=0 GFE(blue), POD(red)');
subplot(3,2,4)
plot(x,soltemp(Y,x,0),x,tempPOD(x,0,dT,YPOD),'r');
title('Temp at t=0 GFE(blue), POD(red)');
subplot(3,2,5)
surf(X,Z,solflowmesh(Y,X,Z)-flowpodmesh(YPOD,X,Z));
title('Deviation in Velocity between POD and GFE');
subplot(3,2,6)
surf(X,Z,soltempmesh(Y,X,Z)-temppodmesh(YPOD,X,Z));
title('Deviation in Temperature between POD and GFE');

% Error calculation of the POD compared to GFE with the L2 norm in time and
% space. First the time and space 3GP are set. We set the time points again
% to 50 but for the space we use 32 as done in the error calculation for
% the GFE/FEM in chapter two have a comparable accuracy.

t_points=50;
tspan=linspace(0,tfinal,t_points);
g3ptimescript
g3pspace;
globL2errPOD;
fprintf('Global relative error of the POD vs GFE was %e  \n',PODerr);
toc;
```

In the following the Matlab scripts and functions that are called within the POD main file are listed.

```
%-----------------------------CorrelationMatrix.m-----------------------------
% Calculates the correlation matrices Kw and KT obtained from
% S (=t_points for simplicity) snapshots of the approximate GFE
% solution. The entries in Kw,KT are L2 inner products of two snapshots of
% the system at different times.

% Y is the solution matrix of the ODE solver from the GFE full model sol.
% The eigenvalues are automatically orthonormal when using the eig command.

Kw=(1/t_points)*Y(:,1:N+1)*MM*Y(:,1:N+1)';
[v1,val1]=eig(Kw);

KT=(1/t_points)*Y(:,N+2:2*N+1)*MM(1:N,1:N)*Y(:,N+2:2*N+1)';
[v2,val2]=eig(KT);

% Evaluation of the energy that remains in the system when picking i
% POD basis functions. The energy is represented by the normalized sum of
% eigenvalues and plotted for the temperature and velocity field.
% Only the first 10 EV are plotted, since they decay rapidly.
```

```
Energy=zeros(10,2);
for i=1:10
   Energy(i,1)=sum(diag(val1(1:i,1:i)))/sum(diag(val1));
   Energy(i,2)=sum(diag(val2(1:i,1:i)))/sum(diag(val2));
end

for i=1:10
    hold on
    plot(i,Energy(i,1),'*');
    plot(i,Energy(i,2),'+');
    hold off
end

% Semilogplot of the eigenvalues of Kw and KT
for i=1:10
    hold on
    semilogy(i,diag(val1(i,i)),'*');
    semilogy(i,diag(val2(i,i)),'+');
    hold off
end

% Both systems can be chosen to have different accuracy.
global dw dT
dw=input('How many POD Basis functions for w?   ');
dT=input('How many POD Basis functions for T?   ');


%------------------------------PODCoefficients.m-----------------------------------
% The matrix Gamma contains the coefficients for the POD Basis functions.
% The ith column provides POD coeff. for the ith POD Basis fcts for w and
% T.

global Gamma

Gamma=zeros(2*N+1,max(dw,dT));

% The first N+1 rows contain the coefficients for the velocity basis
for i=1:dw
    for j=1:N+1

        Gamma(j,i)=Y(:,j)'*v1(:,i);
    end

       Gamma(1:N+1,i)=(1/sqrt(t_points*val1(i,i)))*Gamma(1:N+1,i);
end

% The following N rows contain the coefficients for the temperature basis
for i=1:dT
    for j=N+2:2*N+1
```

```
        Gamma(j,i)=Y(:,j)'*v2(:,i);
    end

       Gamma(N+2:2*N+1,i)=(1/sqrt(t_points*val2(i,i)))*Gamma(N+2:2*N+1,i);
end
```

The POD basis functions and their derivatives are very similar in structure, hence, for brevity we inserted the lines in the code that need to be replaced to obtain the other function. Thus, the following are the codes for $PODbasisw.m$, $PODbasisT.m$ and $PODbasisTpri.m$.

```
function phi = POD_basis_w(x,k)
% Builds the kth POD basis function for w from the correlation matrix Kw.

global  node h N Gamma

phi=zeros(length(x),1);
% phi generates a vector of function values of the kth basis function.

for i=1:length(x)
    for j=1:N+1

        phi(i,1)= phi(i,1)+Gamma(j,k)*basisfct(j,x(i),node,h);
    end
    %for j=1:N  % This is for the POD_basis_T(x,k)

    %   psi(i,1)= psi(i,1)+Gamma(j+N+1,k)*basisfct(j,x(i),node,h);
    %psi(i,1)= psi(i,1)+Gamma(j+N+1,k)*basisfctprime(j,x(i),node,h);
    % The above is for PODbasisTpri(x,k)
    %end

end
end


function v = podrhs(t,y)
% Assembles the RHS of the POD approximation.
% It builds the nonlinear part J(alhpaPOD, betaPOD) where we set
% y(i)=alphaPOD(i) for i=1,..,dw and y(i)=betaPOD(i) for i=dT+1,..,dw+dT.
% The function F_POD(t) has to be provided. All matrices were computed in
% the POD Main file.

global dw dT S Nhat ahat B

J=zeros(dw+dT,1);

% First the nonlinearity in Burgers equation
Ahelp=y(1:dw,1)*y(1:dw,1)'; % Helps to assemble the vector ahat
ahat=zeros(dw^2/2+dw/2,1);  % This is the number of elements in the upper
                            % diag. matrix of Ahelp.
```

```
m=0;
for i=1:dw
    for j=i:dw
    m=m+1;
        ahat(m,1)=Ahelp(i,j);
    end
end

J(1:dw,1)=0.5*Nhat*ahat;    % Nonl. coming from d/dx*w(x,t)^2

% The nonlinearity in the second equation is evaluated.
Lahelp=y(1:dw,1)*y(dw+1:dw+dT,1)'; % To assemble the vector Lambda
Lambda=zeros(dw*dT,1);       %
m=0;
for i=1:dw
    for j=1:dT
        m=m+1;
        Lambda(m,1)=Lahelp(i,j);
    end
end

J(dw+1:dw+dT,1)=B*Lambda;   % Nonlinearity arising from w(x,t)*T(x,t)


v =(-J - S*y + F_POD(t));

end
```

First, the function $FPOD.m$ has to be provided that does the actual integration of the forcing f in every time step.

```
function y = F_POD(t)
% We integrate the product of the forcing function f(x,t) and the POD
% basis for T with a 3P Gauss quadrature. The function f.m has to be
% provided.

global g3px xstep dw dT PODT

F_N=zeros(dT,1);
for k=1:dT
    for nel=1:max(size(g3px))
    F_N(k,1)=F_N(k,1)+f(g3px(nel,1),t)*PODT(nel,(k-1)*3+1)*xstep(nel)/2*0.55555556;
    F_N(k,1)=F_N(k,1)+f(g3px(nel,2),t)*PODT(nel,(k-1)*3+2)*xstep(nel)/2*0.88888889;
    F_N(k,1)=F_N(k,1)+f(g3px(nel,3),t)*PODT(nel,(k-1)*3+3)*xstep(nel)/2*0.55555556;
    end
end

y=[zeros(dw,1);F_N];

end
```

For the error calculation in the POD setup, the script globL2err.m and the functions $L2errPOD.m$ and $L2normPOD.m$ are written. Note the similarity to the L2 error calculation in the FEM/GFE setup.

```
%————————————————————globL2errPOD.m————————————————————————————
% Script calculates the relative L2 error globally. It uses L2err(Y,YPOD,t)
% to call the L2 error at a specified time t. GP are constructed before
% for space and time independently of the mesh size N.

err=0;
for i=1:Lt
    err=err+L2errPOD(Y,YPOD,g3ptime(i,1))^2*(tstep(i))/2*0.55555556;
    err=err+L2errPOD(Y,YPOD,g3ptime(i,2))^2*(tstep(i))/2*0.88888889;
    err=err+L2errPOD(Y,YPOD,g3ptime(i,3))^2*(tstep(i))/2*0.55555556;
end

err=sqrt(err);

norm=0;
for i=1:Lt
    norm=norm+L2normPOD(Y,g3ptime(i,1))^2*(tstep(i))/2*0.55555556;
    norm=norm+L2normPOD(Y,g3ptime(i,2))^2*(tstep(i))/2*0.88888889;
    norm=norm+L2normPOD(Y,g3ptime(i,3))^2*(tstep(i))/2*0.55555556;
end
norm=sqrt(norm);

PODerr=err/norm;


function x = L2errPOD(Y,YPOD,t)
% We integrate the difference function between the gfe solution and the POD
% sol. for fixed t in space with a 3P Gauss quadrature. This gives the
% L2error at a instant in time.

global xstep Lx g3px dw dT

% The gauss weights for 3P Gauss are 0.55555556 and 0.88888889
y=0;
for nel=1:Lx
    y=y+abs(flowPOD(g3px(nel,1),t,dw,YPOD)-solflow(Y,g3px(nel,1),t))^2...
                *xstep(nel)/2*0.55555556;
    y=y+abs(flowPOD(g3px(nel,2),t,dw,YPOD)-solflow(Y,g3px(nel,2),t))^2...
                *xstep(nel)/2*0.88888889;
    y=y+abs(flowPOD(g3px(nel,3),t,dw,YPOD)-solflow(Y,g3px(nel,3),t))^2...
                *xstep(nel)/2*0.55555556;

end
y=sqrt(y);
```

```
z=0;
for nel=1:Lx
    z=z+abs(tempPOD(g3px(nel,1),t,dT,YPOD)-soltemp(Y,g3px(nel,1),t))^2...
                *xstep(nel)/2*0.55555556;
    z=z+abs(tempPOD(g3px(nel,2),t,dT,YPOD)-soltemp(Y,g3px(nel,2),t))^2...
                *xstep(nel)/2*0.88888889;
    z=z+abs(tempPOD(g3px(nel,3),t,dT,YPOD)-soltemp(Y,g3px(nel,3),t))^2...
                *xstep(nel)/2*0.55555556;

end
z=sqrt(z);

x=y+z;

end


function x = L2normPOD(Y,t)
% Similar to L2errPOD, except that we only consider the benchmark solution
% from the gfe method. Thus, this is the L2norm of the gfe solution.


global xstep Lx g3px

y=0;
for nel=1:Lx
    y=y+abs(solflow(Y,g3px(nel,1),t))^2*xstep(nel)/2*0.55555556;
    y=y+abs(solflow(Y,g3px(nel,2),t))^2*xstep(nel)/2*0.88888889;
    y=y+abs(solflow(Y,g3px(nel,3),t))^2*xstep(nel)/2*0.55555556;
end
y=sqrt(y);

z=0;
for nel=1:Lx
    z=z+abs(soltemp(Y,g3px(nel,1),t))^2*xstep(nel)/2*0.55555556;
    z=z+abs(soltemp(Y,g3px(nel,2),t))^2*xstep(nel)/2*0.88888889;
    z=z+abs(soltemp(Y,g3px(nel,3),t))^2*xstep(nel)/2*0.55555556;
end
z=sqrt(z);

x=y+z;

end
```

The next four functions generate the solution functions for temperature and velocity, respectively. Again, due to similarities in the code, the lines that change in the code have been inserted.

```
function y = flowPOD(x,t,dw,YPOD)
```

```
% We construct the POD solution for the velocity w using dw basis fcts.
% YPOD contains all information needed and has size (t_points x dw+dT).
global tfinal

if t==0
    tint=1;
else
tint=ceil((t/tfinal)*max(size(YPOD(:,1))));
% Converts a numerical time value into an index in the corresponding time
% vector T. ceil rounds up to the next integer value
end

y=zeros(max(size(x)),1);
for j=1:max(size(x))
    for i=1:dw

        y(j,1)=y(j,1)+YPOD(tint,i)*POD_basis_w(x(j),i);
    end
    % The following has to replace the above to obtain tempPOD.m
    %for i=1:dT

      %  y(j,1)=y(j,1)+YPOD(tint,i+dw)*POD_basis_T(x(j),i);
    %end
end

end


function y=flowpodmesh(YPOD,x,t)
% This file is especially written for x and t as provided by
% meshgrid(xvector,tvector), so x and t are matrices. Note that when doing
% so, x(j,m)=x(1,m) forall j and y(j,m)=y(j,1)
global tfinal dw  % dT for temppodmesh.m

lx=length(x(1,:));      % lx= # of space points to evaluate the sol.
lt=length(t(:,1));      % lt= # of time instances to eval. the solution


% This converts num. time values to the corresponding index in the time
% vector tspan. e.g if t=tfinal, tint=t_points. In the special case where
% t=0, we set the index to 1, since index of arrays starts at 1.

tint=ceil((t/tfinal)*max(size(YPOD(:,1))));
tint(1,:)=1;


y=zeros(lt,lx);
for j=1:lt
   for m=1:lx
      for i=1:dw
```

```
          y(j,m)=y(j,m)+YPOD(tint(j,1),i).*POD_basis_w(x(1,m),i);

        end
        % For temppodmesh.m the following has to be replaced
        % for i=1:dT

        %   y(j,m)=y(j,m)+YPOD(tint(j,1),i+dw).*POD_basis_T(x(1,m),i);

       % end
    end
end

end
```