Model order reduction and boundary control of incompressible Boussinesq flow

by

Zichuan Wang

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Mechanical Engineering
University of Alberta

Abstract

The time evolution of a two dimensional incompressible, density stratified, Boussinesq flow in a rectangular cavity is numerically simulated for a range of parameters. Boundary control is then implemented along the upper boundary by adjusting the fluid density. More specifically, the top boundary condition of the cavity is a fixed function of space that is modulated by the control input. The resulting numerical simulation for the fluid density and velocity is computed using finite differences in the vertical direction and a spectral method in the horizontal direction. To develop the control strategy, the flow is simulated and a sequence of snapshots of the density and velocity fields are collected. Then, a reduced order modelling method suitable for a linear quadratic regulator (LQR) of the Boussinesq flow is developed using the proper orthogonal decomposition (POD)/Galerkin approach. The reduced order model is obtained by projecting the governing equations of the flow onto the sub space spanned by a finite number of basis functions obtained using the method of snapshots. For the flow in question, the POD method based on the snapshots yields 6 POD modes which capture 99% of the flow energy. In turn, the boundary control is transferred to the governing equations using Duhamel's principle so that the resulting equations contain the control input.

The feasibility of this method is assessed using a LQR boundary controller that is designed based on the reduced order model. The cost functional which is minimized in the LQR control design is defined to be the squared norm of the difference between the actual density field and the desired density field in the cavity. The weighting parameter of the cost functional is found to play a critical role in the process of controller design. To judge the effectiveness of the control, two metrics η_a and η_r are introduced. η_a denotes the absolute effectiveness of the LQR controller and is a metric of the controller's ability to drive the system to the final desired state.

Conversely η_r denotes the relative effectiveness of the LQR controller and is a metric of the improvement of the LQR controller compared to an open-loop controller. For the control cases tested, the LQR controller is found to have $\eta_a = 96.7\%$ and $\eta_r = 22.2\%$ in the best case for the steady state starting configurations, and $\eta_a = 99.3\%$ and $\eta_r = 47.9\%$ for representative transient cases. η_r is more than doubled in the comparison between the steady case and the transient case. This indicates that the LQR controller is able to reject complex transient flow much better than the open-loop controller. In conclusion, a relatively simple feedback control scheme applied on the boundary of a turbulent flow improves the performance in regulating the density field to its desired final state compared to open-loop control.

Acknowledgements

I would like to thank Dr. Bob Koch and Dr. Morris Flynn for their thoughtful guidance, insightful advice and continuous wisdom through my master study at the University of Alberta. Without their endless support along the way, none of this would have been possible.

To my parents, thank you so much for all your support and encouragement. It is their endless love and patience that made me to keep following my dreams.

Finally, to my colleagues and friends over the years, thank you for the camaraderie and happiness that we share along the way.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

Control problems that incorporate partial differential equations (PDEs) as state equations are difficult to solve in real time. One such situation that presents many especially daunting challenges is the control of fluid dynamical systems. Traditionally, the solution of fluid dynamical equations obtained using finite element, finite volume, finite difference or spectral methods is not viable for real-time control because of the high computational complexity [Krstic and Smyshlyaev, 2008]. However, reduced order modeling has proven to be a powerful tool in reducing the computational cost of the mathematical models that arise in the numerical simulations [Antoulas, 2005].

The basic idea of model order reduction is to replace a complicated system modelled by high-fidelity simulations with a much simpler system that still preserves the essential dynamics of the original system. In this thesis, the proper orthogonal decomposition (POD) approach is used for the purpose of model order reduction. POD is a statistical procedure that uses orthogonal transformations to form linearly uncorrelated basis functions of an ensemble of variables [Holmes et al., 1998]. The elements inside this basis are called the principal components of the original ensemble. This method was independently proposed by [Karhunen, 1947] and [Loeve, 1978], and is

sometimes called Karhunen-Loève theorem. In [Holmes et al., 1998], the method was used to study turbulent flow and was first called POD. Subsequently, the method of snapshots [Sirovich, 1987] was incorporated into the framework.

In contrast to traditional numerical methods which use basis functions that have very little connection with the problem they are solving, the POD approach uses basis functions that are generated from either experimental measurements or numerical simulation data. This set of basis functions are optimal in the sense that the averaged projection of the dataset onto those basis functions is larger than any other set of basis functions. When the governing PDEs are projected onto those basis functions using the Galerkin projection, a reduced order model is obtained [Holmes et al., 1998]. The beauty of the POD/Galerkin approach is that it is a non-linear model reduction approach and the governing equations of the resulting reduced order model, also know as the Galerkin system, consist solely of ordinary differential equations [Luchtenburg et al., 2009].

In light of the above, the goals of this thesis are first to simulate incompressible, density stratified Boussinesq flow in a two-dimensional cavity with control being applied on the top boundary. Simulations are carried out using a open-source direct numerical simulation (DNS) solver called Diablo[1]; second, to generate the corresponding reduced order model (ROM) by applying the POD/Galerkin approach. The ROM encapsulates the system dynamics of which the linear quadratic regulator (LQR) boundary controller is designed on.

The type of fluid control outlined above serves as a prototype for an indoor HVAC system that combines distributed heating e.g. due to solar gains and localized cooling e.g. due to the action of an air conditioner. Different HVAC scenarios can be simulated by choosing different boundary conditions on the top boundary. Within this thesis we focus on the technical aspects of algorithm development rather than HVAC

[1]Details are given at `http://www.damtp.cam.ac.uk/user/jrt51/files.html`

application. Correspondingly, and for illustration purposes, attention is restricted to Dirichlet boundary conditions. In other words, the control is applied by directly specifying the fluid temperature or density (spatial-varying but symmetric) along the top boundary.

1.2 Thesis organization

The layout of this thesis is as follows. In chapter 2, the detailed algorithm for the numerical simulation is outlined. Also presented is a characterization of the flow in terms of relevant parameters related to fluid properties, physical dimensions, etc. In chapter 3, the derivation of the reduced order model based on the POD/Galerkin approach is outlined along with the design of the LQR boundary controller. In chapter 4, the evaluation of both the fidelity of the reduced order model and the effectiveness of the LQR controller under different circumstances are considered. Finally in chapter 5, major conclusions and contributions of this thesis are described along with possible future work directions.

Chapter 2

Numerical Method

2.1 Preliminary Remarks

For incompressible, density stratified, Boussinesq flow, the governing equations can be written as [Kundu et al., 2011]:

$$\nabla \cdot \boldsymbol{u} = 0 \tag{2.1}$$

$$\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\boldsymbol{u} = -\frac{1}{\rho_0}\nabla p + \nu \nabla^2 \boldsymbol{u} + \Big(\frac{\rho - \rho_0}{\rho_0}\Big)\boldsymbol{g} \tag{2.2}$$

$$\frac{\partial \rho}{\partial t} = \frac{\nu}{Pr}\nabla^2 \rho - \boldsymbol{u} \cdot \nabla \rho \tag{2.3}$$

where $\boldsymbol{u}$ is the velocity, p is the hydrostatically adjusted pressure, ρ is the fluid density, ρ_0 is a reference density, ν is the kinematic viscosity, Pr is the Prandtl number and $\boldsymbol{g}$ is the gravitational acceleration vector. Note that, according to the equations of state for many simple fluids, density is linearly related to temperature. Therefore equation (2.3) is introduced into the framework with the notion that density variation are synonymous with temperature variations. To solve the above equations numerically, the continuous flow field must be transformed into its discrete counterpart. Furthermore, the spatially discretized equtions must be advanced in time using a discrete (though not necessarily fixed) time step Δt. Ideally, one wants to maintain

both the accuracy and stability of the simulation while choosing reasonable (i.e. not punitively small) values for Δt and the spatial grid sizes, in order to minimize the computational expense. Here equations (2.1) through (2.3) are solved numerically using an open-source DNS solver called Diablo, which was originally developed by Professor Thomas R. Bewley (University of California, San Diego) and Dr. John R Taylor (University of Cambridge). Generally speaking, Diablo can be applied to 2D or 3D turbulent flow within simple geometries. In this current context, a 2D channel flow setup is used in which the flow is aperiodic in the vertical direction y and periodic in the horizontal direction x. Second-order finite differences are used to discretize spatial derivatives in y whereas Fourier spectral methods are used to discretize spatial derivatives in x as described in detail below. Because the code is of DNS type, rather than a large-eddy simulation (LES) numerical model or a Reynolds-average Navier-Stokes (RANS) numerical model, the turbulent motions are supposed to be resolved at all length-scales. This has the advantage of avoiding sub-grid parameterizations, which can be difficult to calibrate or justify in case of stratified turbulence. On the other hand, DNS simulations are more computationally intensive than LES or RANS simulations as a result of which attention is restricted to relatively small domains as described blow.

2.2 Spatial Discretization

In the vertical direction y and with reference to figure 2.1, a staggered (non-stretched) grid is used in which vertical velocities are defined at the GY nodes and horizontal velocities, the fluid pressure and active scalars are located at the GYF nodes. More specifically for our case, a single active scalar density is considered. A staggered grid is used so that the pressure values at adjacent nodes can be coupled together. If central finite differencing is used on a non-staggered grid, pressure values at adjacent

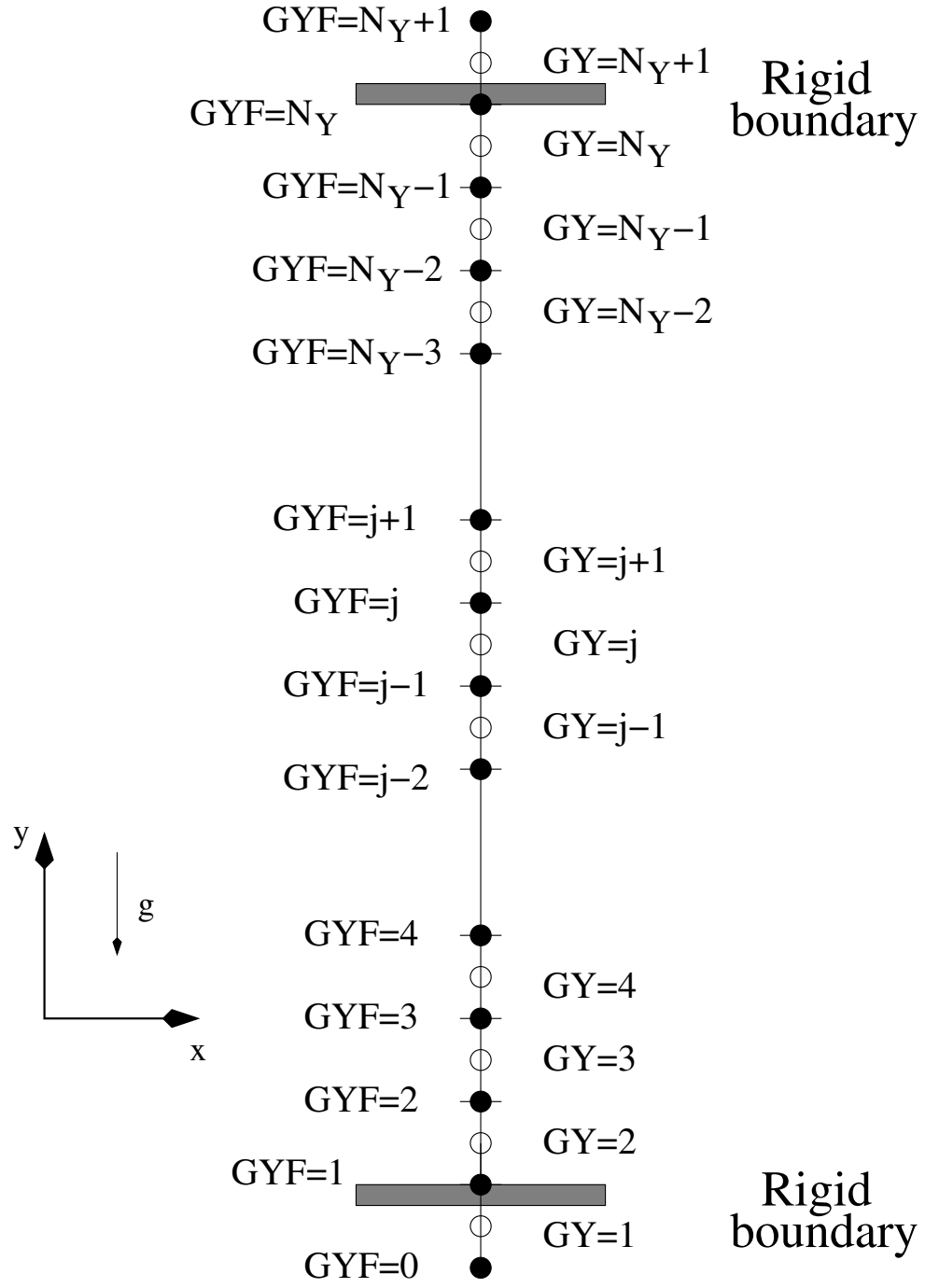

Figure 2.1: Grid structure in the wall-normal (y) direction. After [Flynn, 2006], figure E.1.

nodes would only be coupled through viscous terms. In this case and for flow with large Reynolds number, spurious fine scale oscillations may arise [Fletcher, 2012]. The GYF nodes lies exactly halfway in between neighboring GY nodes which makes it straightforward to interpolate from one grid to the other, i.e. interpolation can be accomplished by simply computing the average of the adjacent values in a second-order accurate arithmetic step.

Boundary conditions in y are applied using ghost nodes ($\mathrm{GY} = 1, N_Y + 1; \mathrm{GYF} = 0, N_Y + 1$) as specified in figure 2.1. Boundary conditions can be of Dirichlet, Neumann or Robin type. Although boundary conditions are not always specified exactly at the wall locations due to the usage of a staggered grid, this does not significantly affect the results provided the grid spacing is sufficiently small. When Dirichlet boundary conditions are applied, the horizontal velocity and density are specified exactly at the location of the wall ($\mathrm{GYF} = 1$ or $\mathrm{GYF} = N_Y$), whereas the vertical velocity is prescribed at nodes $GY = 2$ and $GY = N_Y$. When Neumann boundary conditions are applied for horizontal velocity or density, ghost nodes at $\mathrm{GYF} = 0$ and $\mathrm{GYF} = N_Y + 1$, coupled with $\mathrm{GYF} = 1$ and $\mathrm{GYF} = N_Y$ respectively, are used to calculate the relevant vertical gradients corresponding to grid location $\mathrm{GY} = 1$ and $\mathrm{GY} = N_Y + 1$ by incorporating a second-order central finite difference scheme. Since the vertical velocity, located at grid GY, is offset from the wall, its wall-normal derivatives can be specified exactly at the wall.

In order to improve numerical accuracy, a Fourier decomposition is applied to all derivatives in the horizontal direction. Thus for a generic flow variable $f(x, y, t)$, we write

$$f(x, y, t) = \sum_{n=-\frac{N_X}{2}+1}^{\frac{N_X}{2}-1} \hat{f}_n(k_n, y, t) e^{ik_n x} \tag{2.4}$$

where $k_n = 2\pi n / N_X$ is the horizontal wavenumber, N_X is the number of grid points in the x direction, and $\hat{f}_n$ denotes the Fourier transform of f. Differentiating (2.4)

with respect to x gives

$$\frac{\partial f}{\partial x} = \sum_{n=-\frac{N_X}{2}+1}^{\frac{N_X}{2}-1} ik_n \hat{f}_n(k_n, y, t) e^{ik_n x} \tag{2.5}$$

In Diablo, discrete Fourier transforms are calculated using the open source FFTW software library developed by [Frigo and Johnson, 2003]. The FFTW library was updated from version 2 to the latest version 3 by the author, as outlined in appendix A. Moreover, Orszag's 2/3 de-aliasing rule is applied to prevent high-wavenumber modes from feeding spurious energy to their low-wavenumber counterpart [Canuto et al., 1988]. This is accomplished by setting all Fourier modes with $n > N_X/3$ to zero before transforming them back to physical space.

When it comes to the Fourier space evaluation of the nonlinear terms in the Navier-Stokes equations, special care is required. Evaluating a product in Fourier space entails computing a discrete convolution sum requiring $O(N^2)$ operations where N is the total number of grid points in that direction. To avoid this computational overhead, the pseudo-spectral method suggested by [Taylor, 2008] is instead applied where the nonlinear team is first written in conservation form $\partial(f_i f_j)/\partial x$. The product $f_i f_j$ is first computed in physical space and then transferred into Fourier space, requiring only $O(N \log(N))$ operations instead of $O(N^2)$.

2.3 Temporal Advancement

Diablo leverages a time advancement scheme that combines explicit third order low storage[1] Runge-Kutta-Wray (RKW3) and implicit Crank-Nicolson. When numerically solving a diffusion problem using an explicit method, the diffusion number defined as $D = \nu \delta t / \delta x^2$ must be less than some constant in order for the simulation

[1]Low storage is preferred because memory can be one of the major limiting factors associated with DNS-type numerical simulations.

to be numerically stable. This can place onerous restrictions on the size of δt for a given grid spacing. The Crank-Nicolson scheme, on the other hand, is unconditionally stable for pure diffusion problems. This makes Crank-Nicolson a desirable choice for diffusion terms, more specifically $\nu\nabla^2\boldsymbol{u}$ in equation (2.2).

Runge-Kutta methods introduce multiple fractional time steps between time instants t_n and $t_n + \Delta t$ with Δt being the full time step. At the expense of higher computational costs per full time step, higher accuracy is obtained as compared to an explicit single-step scheme like forward Euler [Moin, 2010]. The generic form of the RKW3 algorithm reads as follows [Bewley, 2014].

$$f_1 = f(\theta_n, t_n) \tag{2.6}$$

$$\theta^* = x_n + \beta_1 h_1 f_1 \tag{2.7}$$

$$f_2 = f(\theta^*, t_n + h_1) \tag{2.8}$$

$$\theta^{**} = \theta^* + \beta_2 h_2 f_2 + \zeta_2 h_2 f_1 \tag{2.9}$$

$$f_3 = f(\theta^{**}, t_n + h_1 + h_2) \tag{2.10}$$

$$\theta_{n+1} = \theta^{**} + \beta_3 h_3 f_3 + \zeta_3 h_3 f_2 \tag{2.11}$$

where

$$h_1 = (8/15)\Delta t, \quad h_2 = (2/15)\Delta t, \quad h_3 = (1/3)\Delta t$$

$$\beta_1 = 1, \quad \beta_2 = 25/8, \quad \beta_3 = 9/4$$

$$\zeta_1 = 0, \quad \zeta_2 = -17/8, \quad \zeta_3 = -5/4$$

The variable θ either represents velocity or density. As required by equation (2.1), the velocity field should always remain divergence free. However between time instants t_n

and $t_n + \Delta t$, the intermediate velocity field $\tilde{u}_n^{rk}$ may not satisfy this constraint. Here superscript rk represents the corresponding R-K fractional time step. Therefore, before each individual fractional step of RKW3, a pressure correction technique is applied to ensure the velocity field is always divergence free. More specifically, if u_n^{rk} is the corrected velocity field

$$u_n^{rk} = \tilde{u}_n^{rk} - h_j \frac{\partial \phi}{\partial x_i} \tag{2.12}$$

where $\phi = p^{rk+1} - p^{rk}$ is the pressure correction between successive RKW3 fractional steps. Because u_n^{rk} is divergence free by definition, a Poisson equation for ϕ is obtained by taking the divergence of equation (2.12).

$$\nabla^2 \phi = \frac{1}{h_j} \nabla \cdot \tilde{u}_n^{rk} \tag{2.13}$$

Once ϕ is determined from the solution of (2.13), the pressure can be updated from $p^{rk+1} = p^{rk} + \phi$. In a similar fashion, the velocity field is updated using (2.12). Note that the fluid pressure is not initialized at the very beginning of a numerical simulation, rather it is calculated from (2.2) and (2.3) via:

$$\frac{1}{\rho_0} \nabla^2 p = -\nabla \cdot (\boldsymbol{u} \cdot \nabla \boldsymbol{u}) \tag{2.14}$$

2.4 Algorithm

Having outlined key fundamental components of Diablo above, a detailed description of the code and its execution for the special case of a 2D flow having a single active scalar, referred to below as θ, is now presented. Regarding the notation to follow, R_i denotes the right hand side of the momentum equation (2.2) while F_i saves all the Runge-Kutta terms used in the next Runge-Kutta substep. Consistent with (2.4)

and (2.5), hatted and unhatted variables are defined respectively, in Fourier space and physical space. Also, for the sake of runtime efficiency, Diablo minimizes both the number of stored arrays and the number of calls to FFTW. The "recipe" presented below follows the more expansive discussion of [Taylor, 2008] where additional details may be found.

0. Interpolate the horizontal velocity u_1 and active scalar θ at the GYF nodes to the GY nodes, where the vertical velocity u_2 is located

$$\bar{u}_1(i,j) = \frac{1}{2}(u_1(i,j) + u_1(i,j-1))$$
$$\bar{\theta}(i,j) = \frac{1}{2}(\theta_1(i,j) + \theta_1(i,j-1))$$

1. Build, in Fourier space, the right hand side term $\hat{R}$ using the previously computed velocity, $\hat{u}_i$ and scalar concentration, $\hat{\theta}$

$$\hat{R}_i = \hat{u}_i$$
$$\hat{R}_\theta = \hat{u}_\theta$$

2. Add previous R-K terms $\hat{F}^{PREV}$ to $\hat{R}$ if this is not the first R-K step

$$\hat{R}_i = \hat{R}_i + \zeta_{rk} h_{rk} \hat{F}_i^{PREV}$$
$$\hat{R}_\theta = \hat{R}_\theta + \zeta_{rk} h_{rk} \hat{F}_\theta^{PREV}$$

3. Add the pressure gradient to $\hat{R}$ of the momentum equation

$$\hat{R}_1 = \hat{R}_1 - h_{rk} \hat{i} k_x \hat{P}$$
$$\hat{R}_2(k_x,j) = \hat{R}_2(k_x,j) - h_{rk} \frac{\hat{P}(k_x,j) - \hat{P}(k_x,j-1)}{\Delta y}$$

4. Add the viscous terms containing horizontal derivatives of $\hat{u}_i$ and $\hat{\theta}$ to the current R-K terms $\hat{F}$

$$\hat{F}_i = -\nu k_x^2 \hat{u}_i$$
$$\hat{F}_\theta = -\alpha k_x^2 \hat{\theta}$$

5. Add those non-linear terms that contain horizontal derivatives to $\hat{F}$

$$\hat{F}_1 = \hat{F}_1 - \hat{i} k_x \widehat{u_1 u_1}$$
$$\hat{F}_2 = \hat{F}_2 - \hat{i} k_x \widehat{\bar{u}_1 u_2}$$
$$\hat{F}_\theta = \hat{F}_\theta - \hat{i} k_x \widehat{\theta u_1}$$

6. Compute the non-linear terms that contain vertical derivatives S in the physical domain. $S_1 = \frac{\partial(\bar{u}_1 u_2)}{\partial y}$, $S_2 = \frac{\partial(u_2 u_2)}{\partial y}$, $S_\theta = \frac{\partial(\bar{\theta} u_2)}{\partial y}$

$$\begin{aligned} S_1(i,j) =& \left(\frac{u_1(i,j+1)+u_1(i,j)}{2}\right)\left(\frac{u_2(i,j+1)}{\Delta y}\right) \\ & - \left(\frac{u_1(i,j)+u_1(i,j-1)}{2}\right)\left(\frac{u_2(i,j)}{\Delta y}\right) \\ S_2(i,j) =& \left(\frac{u_2(i,j+1)+u_2(i,j)}{2}\right)^2\left(\frac{1}{\Delta y}\right) \\ & - \left(\frac{u_2(i,j)+u_2(i,j-1)}{2}\right)^2\left(\frac{1}{\Delta y}\right) \\ S_\theta(i,j) =& \left(\frac{\theta(i,j+1)+\theta(i,j)}{2}\right)\left(\frac{u_2(i,j+1)}{\Delta y}\right) \\ & - \left(\frac{\theta(i,j)+\theta(i,j-1)}{2}\right)\left(\frac{u_2(i,j)}{\Delta y}\right) \end{aligned}$$

7. Convert S to Fourier space and then add it to $\hat{R}$

$$\hat{F}_1 = \hat{F}_1 - \hat{S}_1$$

8. Done calculating $\hat{F}$, which is now to be added to $\hat{R}$. From this point on, $\hat{F}$ needs to be kept untouched for the next R-K step

$$\hat{R}_i = \hat{R}_i + \beta_{rk} h_{rk} \hat{F}_i$$
$$\hat{R}_\theta = \hat{R}_\theta + \beta_{rk} h_{rk} \hat{F}_\theta$$

9. Add the viscous terms containing vertical derivatives to $\hat{R}$ as the explicit part of Crank-Nicolson

$$R_i(i,j) = R_i(i,j) + \frac{\nu h_{rk}}{2}\left(\frac{u_i(i,j+1) - 2u_i(i,j) + u_i(i,j-1)}{\Delta y^2}\right)$$
$$R_\theta(i,j) = R_\theta(i,j) + \frac{\alpha h_{rk}}{2}\left(\frac{\theta(i,j+1) - 2\theta(i,j) + \theta(i,j-1)}{\Delta y^2}\right)$$

10. Use the Thomas algorithm [Thomas, 1949] to solve the tridiagonal matrix for the scalar θ and intermediate velocity v_i

$$v_i(i,j) - \frac{\nu h_{rk}}{2}\left(\frac{v_i(i,j+1) - 2v_i(i,j) + v_i(i,j-1)}{\Delta y^2}\right) = R_2(i,j)$$
$$\theta(i,j) - \frac{\alpha h_{rk}}{2}\left(\frac{\theta(i,j+1) - 2\theta(i,j) + \theta(i,j-1)}{\Delta y^2}\right) = R_\theta(i,j)$$

11. Convert v_i to Fourier space $\hat{v}_i$ then use the Thomas algorithm to solve the tridiagonal matrix for the pressure correction ϕ

$$-k_x^2\hat{\phi}(k_x,j) + \left(\frac{\hat{\phi}(k_x,j+1) - 2\hat{\phi}(k_x,j) + \hat{\phi}(k_x,j-1)}{\Delta y^2}\right) =$$
$$\hat{\imath}k_x\hat{v}_1 + \left(\frac{\hat{v}_2(k_x,j+1) - \hat{v}_2(k_x,j)}{\Delta y}\right)$$

12. Use the pressure correction to calculate the divergence-free velocity field for next R-K step

$$\hat{u}_1^{rk+1} = \hat{v}_1 - \hat{i}k_x\hat{\phi}$$

$$\hat{u}_2^{rk+1}(k_x, j) = \hat{v}_2(k_x, j) - (\hat{\phi}(k_x, j) - \hat{\phi}(k_x, j-1))/\Delta y$$

13. Update the pressure field

$$\hat{P} = \hat{P} + \hat{\phi}/h_{rk}$$

2.5 Parameter study

When simulating 2D incompressible, density stratified, Boussinesq flow with Diablo, it is important to choose parameters that guarantee stability and numerical accuracy and that produce a flow field amenable to control using the scheme described in following chapters. Parameters to be considered in this subsection are the number of grid points in the horizontal (NX) and vertical (NY) directions, the Prandtl number Pr, the kinematic viscosity ν and the time spacing Δt. All simulations are conducted within a 110cm $\times$ 40cm rectangular cavity, i.e. $L_x = 110$cm and $L_y = 40$cm.

Originally, water is used as the working fluid of the simulations, therefore $\nu = 0.01\text{cm}^2/\text{s}$ and $Pr = 1$. Although the choice for Pr significantly overestimates the molecular transport of heat, this is necessary for the numerical stability of the algorithm. According to [Sutherland et al., 2004], the magnitude of Pr does not significantly change the behavior of the flow which is, in any event, dominated by turbulent, not molecular transport. The reduced gravity g' associated with the Boussinesq flow is defined as

$$g' = g\left(\frac{\rho_1 - \rho_0}{\rho_0}\right) \tag{2.15}$$

where $\rho_1 = 1.02\text{g/cm}^3$, $\rho_0 = 1.00\text{g/cm}^3$ and $g = 980\text{cm/s}^2$ is the gravitational acceleration. Therefore $g' = 19.6\text{cm/s}^2$. Here $\rho_1 - \rho_0$ is a characteristic maximum density difference. Although the simulations in Diablo assume cgs units, the density field is everywhere rescaled for improving the numerical accuracy numerical stability by this characteristic density difference. The rescaled dimensionless density r is therefore defined as

$$r = \frac{\rho - \rho_0}{\rho_1 - \rho_0} \tag{2.16}$$

Note that density is referred to as this rescaled dimensionless density r in all the chapters to follow. Since Runge-Kutta is used as the time marching scheme, Courant-Friedrichs-Lewy (CFL) condition has to be satisfied to ensure that the numerical simulations remain stable [Courant et al., 1967]. A Courant number C is defined as follows

$$C = \frac{U_{\text{char}}\Delta t}{\Delta x} = \frac{U_{\text{char}}\Delta t\text{NX}}{L_x} \tag{2.17}$$

where U_{char}, the characteristic velocity, is given by

$$U_{\text{char}} = \sqrt{g' L_y} \tag{2.18}$$

Combining equations (2.19) and (2.18) and using parameter values defined above gives

$$C = 0.2545 s^{-1}\Delta t\text{NX} \leq C_{\max} \tag{2.19}$$

where $C_{\max}$ is the maximum Courant number for the simulation to remain stable. Typically $C_{\max} \simeq 1$.

First, we consider the selection of NX and NY. Because Fourier spectral methods are used in the horizontal direction, NX has to be power of two. In order to maintain high fidelity, flow within the cavity should be resolved in sufficient detail, which requires the horizontal spacing $\Delta x = L_x/\text{NX}$ and vertical spacing $\Delta y = L_y/\text{NY}$ to be

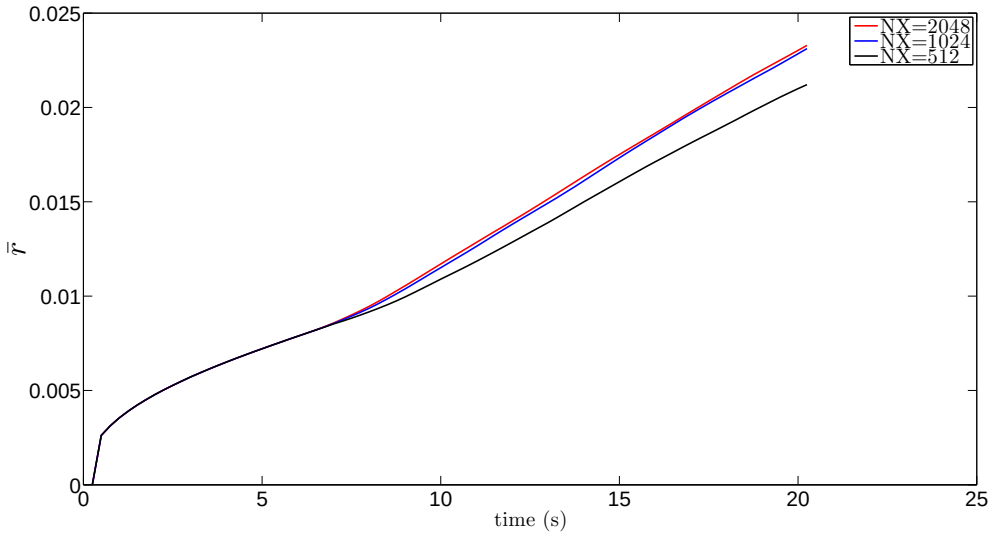

Figure 2.2: Evolution curve of average rescaled dimensionless density $\bar{r}$ of the cavity with different NX.

smaller than the length scale of the smallest vortices. This indicates that Δx and Δy should be comparable in magnitude, i.e. $\Delta x \simeq \Delta y$. To determine the proper value for NX, three otherwise identical simulations with different NX were conducted, i.e. NX = 512, NX = 1024 and NX = 2048. For all three simulations, the top boundary condition for the rescaled dimensionless density r was set to be $0.5(\cos(2\pi x/L_x)+1)$; all other boundary conditions (on r, u or v) are homogeneous Neumann (density) or Dirichlet (velocity). As for the number of grid points in the vertical direction, recall that $\Delta x \simeq \Delta y$. For all three simulations, NY was therefore chosen to be 800 corresponding to the most punitive horizontal grid spacing where NX = 2048. In similar fashion, and motivated by the restriction imposed by (2.19), Δt is chosen to be 0.004s. In order to evaluate the accuracy of the simulations with difference NX, average rescaled dimensionless density $\bar{r}$ of the cavity is plotted against time in figure 2.2. The curves corresponding to NX = 1024 and NX = 2048 almost overlap but the black curve, representing NX = 512, diverges significantly from the other two. Thus the maximum value of NX for fully resolving the fine scale vortices is 1024. As

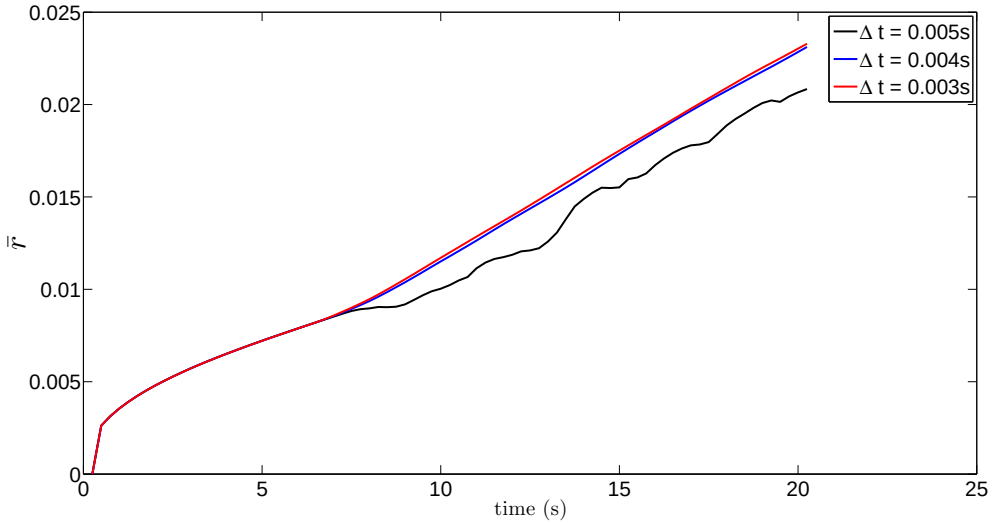

Figure 2.3: Evolution curve of average rescaled dimensionless density $\bar{r}$ of the cavity with different Δt.

such, NX is set to 1024 for all the simulations described hereafter. Recall that finite differencing rather than spectral technique, is used in the vertical direction. Since numerical evaluation of derivatives by finite difference is more costly than by spectral methods, NY is cut in half to 400 from the value described above. In this case, the approximate equality of $\Delta x \simeq \Delta y$ is preserved.

To determine the appropriate value for Δt with the grid size being 1024×400, three otherwise identical simulations with different Δt were conducted in an analogous fashion. As shown in figure 2.3, curves corresponding to $\Delta t = 0.003\,\mathrm{s}$ and $\Delta t = 0.004\,\mathrm{s}$ almost overlap but the black curve, representing $\Delta t = 0.005\,\mathrm{s}$, diverges significantly from the other two. Thus the maximum reasonable value of Δt is $0.004\,\mathrm{s}$. We therefore empirically find that $C_{\max} = 1.0426 \approx 1$. This again confirms the result in equation (2.19).

A velocity quiver plot with all the parameter values defined as above and with a top boundary condition specified by $r = 0.5(\cos(2\pi x/L_x) + 1)$ is presented in figure 2.4. As noted above, our original intention was to conduct numerical simulations using

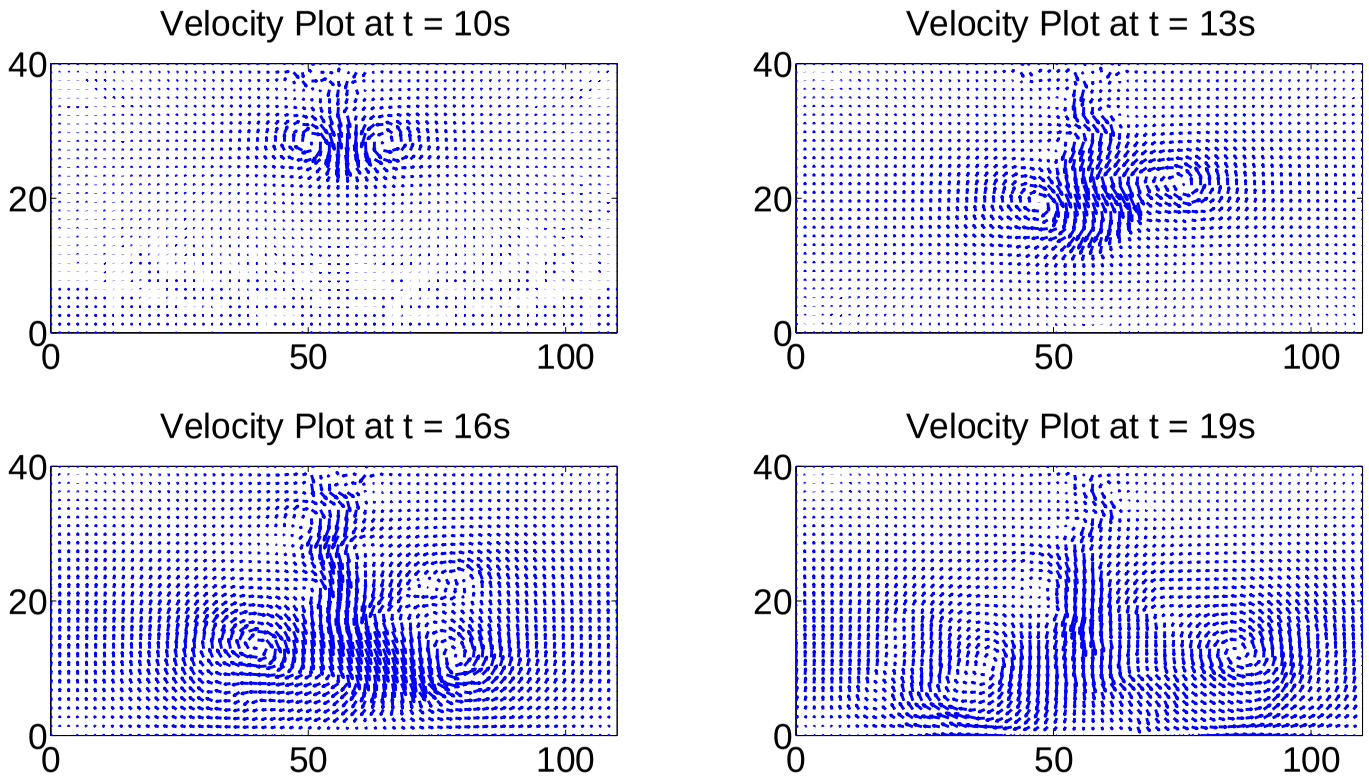

Figure 2.4: Quiver plot showing the direction and relative magnitude of the velocity at four discrete time instants for the case with a cold spot located at upper boundary. The kinematic viscosity of the working fluid is $\nu = 0.01\text{cm}^2/\text{s}$.

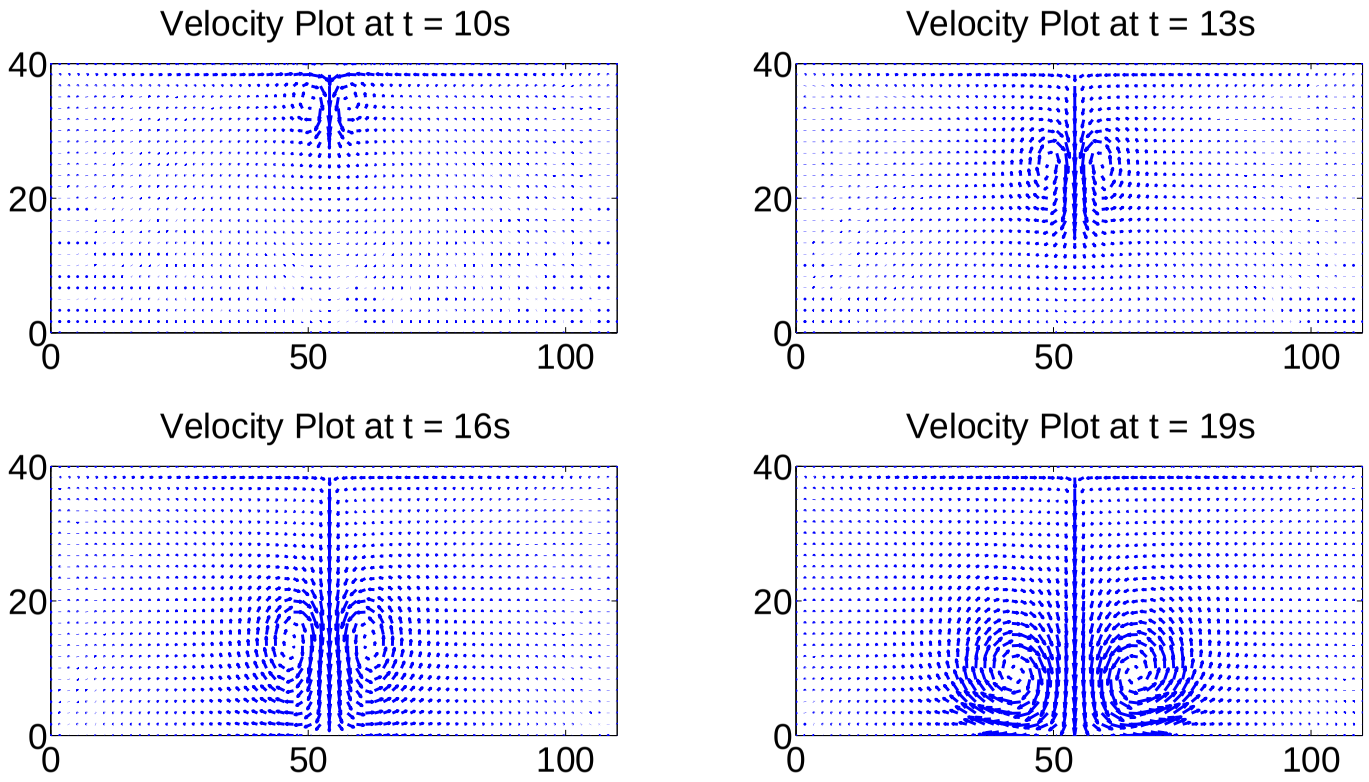

Figure 2.5: Quiver plot showing the direction and relative magnitude of the velocity at four discrete time instants for the case with a cold spot located at upper boundary. The kinematic viscosity of the working fluid is $\nu = 0.1\text{cm}^2/\text{s}$.

water ($\nu = 0.01\text{cm}^2/\text{s}$) as the working fluid. However as shown in figure 2.4, flow in this case is quite chaotic in that it contains vortices of multiple length scales. We expected this to pose challenges insofar as flow control is concerned and therefore elected to conduct further simulations but with artificially elevated ν. Figure 2.5 shows the analogous result to figure 2.4, but with $\nu = 0.1\text{cm}^2/\text{s}$. Here we find that the flow with a higher viscosity $\nu = 0.1\text{cm}^2/\text{s}$ damps the fine scale vortices and remains symmetric (at least for the time interval considered here) which makes it a better candidate in terms of flow control. As such, $\nu = 0.1\text{cm}^2/\text{s}$ is taken to be the kinematic viscosity in the numerical simulations to follow.

In conclusion, the base simulation parameters for this work are $L_x = 110\text{cm}$, $L_y = 40\text{cm}$, NX $= 1024$, NY $= 400$, $Pr = 1$, $\Delta t = 0.004\,\text{s}$ and $\nu = 0.1\text{cm}^2/\text{s}$.

Chapter 3

Model Order Reduction and Boundary Control Method

3.1 Proper Orthogonal Decomposition

Proper orthogonal decomposition, usually abbreviated as POD, is a statistical procedure that uses orthogonal transformations to form a linearly uncorrelated basis of an ensemble of variables, such as data collected from either experiment or numerical simulations [Holmes et al., 1998]. The elements inside this basis are called the principal components of the original ensemble [Sirovich, 1987]. More importantly, this technique has the ability to extract the essential components of a complex and highly non-linear problem using a finite number of modes.

To illustrate the technique, consider an ensemble $\{\theta_i\}$ of a scalar field. The goal is to find a basis $\{\varphi_j\}$, that is the optimal representation of the original ensemble $\{\theta_i\}$ on a specific inner product space. The technical explanation to follow is based on the discussion of [Sirovich, 1987], where further details on the POD method can be found. We consider a linear space $L^2(\Omega)$ of square-integrable real-valued functions so that the inner product is defined as:

$$(f, g)_\Omega = \int_\Omega f(x)g(x)dx \tag{3.1}$$

and the induced $L^2(\Omega)$ norm is defined by:

$$\|f\|_\Omega = \sqrt{(f,f)_\Omega} \tag{3.2}$$

In the present context, optimality requires that the basis functions $\{\varphi_j\}$ are determined such that the normalized averaged projection of $\{\theta_i\}$ onto $\{\varphi_j\}$ is maximized. In symbols

$$\max_{\varphi \in L^2(\Omega)} \frac{\langle |(\theta,\varphi)|^2 \rangle}{\|\varphi\|^2} \tag{3.3}$$

where $\langle \cdot \rangle$ denotes the average, $|\cdot|$ denotes the modulus and $\|\cdot\|$ denotes the L^2-norm. Considering a constrained optimization problem as

$$\max_{\varphi \in L^2(\Omega)} \langle |(\theta,\varphi)|^2 \rangle \quad \text{such that} \quad \|\varphi\|^2 = 1 \tag{3.4}$$

The functional corresponding to this specific optimization problem can be written as:

$$J[\varphi] = \langle |(\theta,\varphi)|^2 \rangle - \lambda(\|\varphi\|^2 - 1) \tag{3.5}$$

and the necessary condition for the existence of extrema is that the derivative of J vanishes for all variations $\varphi + \delta\psi \in L^2(\Omega)$, $\delta \in \mathbb{R}$

$$\frac{d}{d\delta} J[\varphi + \delta\psi]|_{\delta=0} = 0 \tag{3.6}$$

Then we have

$$\begin{aligned}
&\frac{d}{d\delta} J[\varphi + \delta\psi]|_{\delta=0} \\
=&\frac{d}{d\delta}[\langle (\theta, \varphi+\delta\psi)(\varphi+\delta\psi,\theta)\rangle - \lambda(\varphi+\delta\psi,\varphi+\delta\psi)]|_{\delta=0} \\
=&2\Re[\langle (\theta,\psi)(\varphi,\theta)\rangle - \lambda(\varphi,\psi)] = 0
\end{aligned}$$

The expression inside $[\cdot]$ can be rewritten as

$$\left\langle \int_\Omega \theta(x)\psi(x)dx \int_\Omega \varphi(x')\theta(x')dx' \right\rangle - \lambda \int_\Omega \varphi(x)\psi(x)dx$$
$$= \int_\Omega \left[\int_\Omega \langle \theta(x)\theta(x') \rangle \varphi(x')dx' - \lambda\varphi(x) \right] \psi(x)dx = 0$$

Since $\delta\psi(x)$ is an arbitrary variation, in order for the above equation to hold true, the expression inside $[\cdot]$ has to be zero. Therefore

$$\int_\Omega \langle \theta(x)\theta(x') \rangle \varphi(x')dx' = \lambda\varphi(x) \tag{3.7}$$

The term $\theta(x)\theta(x')$ is the averaged autocorrelation function $R(x, x')$ of our original ensemble $\{\theta_i\}$. Defining $R(x, x') = \theta(x)\theta(x')$, the above equation can be rewritten as

$$\int_\Omega R(x, x')\varphi(x')dx = \lambda\varphi(x) \tag{3.8}$$

It is now clear that finding the optimal basis $\{\varphi_j\}$ is the same as finding the eigenvalues of the above eigenvalue problem. For this reason $\{\varphi_j\}$ are often called empirical eigenfunctions [Holmes et al., 1998]. $\{\varphi_j\}$ are also known as the POD modes of the ensemble $\{\theta_i\}$.

3.2 Decomposition in the spatial domain

Since the controller will be designed solely for the density field, only the rescaled dimenionless density r is analyzed under the POD formulation. Similar steps can be used for velocity $\boldsymbol{u}$ if needed. The notation used here follows [Luchtenburg et al., 2009]. Density r is decomposed as follows

$$r(\boldsymbol{x}, t) = r_0(\boldsymbol{x}) + r'(\boldsymbol{x}, t) = r_0(\boldsymbol{x}) + \sum_{i=1}^{N} a_i(t)r_i(\boldsymbol{x}) = \sum_{i=0}^{N} a_i(t)r_i(\boldsymbol{x}) \tag{3.9}$$

For computational convenience, the average density r_0 is added to the summation as the "zeroth" mode, therefore $a_0 = 1$. Also r_0 and r' correspond to the steady and fluctuating parts of the density field, respectively. Meanwhile $\{r_i\}_{i=1}^N$ is POD modes and $\{a_i\}_{i=1}^N$ is the temporal coefficient. Alternatively, $\{a_i\}_{i=1}^N$ can be considered as the POD modes for the temporal domain. The autocorrelation function for the density field is defined as

$$R(\boldsymbol{x}, \boldsymbol{x}') = \langle r'(\boldsymbol{x}, t) \otimes r'(\boldsymbol{x}', t) \rangle \tag{3.10}$$

Here $\langle \cdot \rangle$ denotes the time average of a variable over time period T, i.e.

$$\langle \cdot \rangle = \frac{1}{T} \int_0^T (\cdot) dt \tag{3.11}$$

According to (3.8), the ith POD mode r_i with the corresponding eigenvalue λ_i can be obtained by solving the following eigenvalue problem

$$\int_\Omega R(\boldsymbol{x}, \boldsymbol{x}') r_i(\boldsymbol{x}') d\boldsymbol{x}' = \lambda_i r_i(\boldsymbol{x}) \tag{3.12}$$

$R(\boldsymbol{x}, \boldsymbol{x}')$ is compact self-adjoint and positive semi-definite [Holmes et al., 1998]. This implies that the eigenfunctions (i.e. the POD modes r_i) are mutually orthogonal in $L^2(\Omega)$ and that they are ordered with respect to the real positive eigenvalues in a decending fashion. Note that zero eigenvalues are ignored because they do not contribute to the density field r. According to the orthonormality of r_i

$$(r_i, r_j)_\Omega = \delta_{ij} \tag{3.13}$$

Temporal coefficients a_i can be calculated by projecting the POD modes r_i onto the fluctuating part of the density field r', i.e.

$$a_i(t) = (r', r_i)_\Omega \tag{3.14}$$

Since the temporal coefficients a_i are calculated from the fluctuating part of the density field, it is straightforward to show that the time average of $\{a_i\} = 0$. Because the autocorrelation tensor $R(\boldsymbol{x}, \boldsymbol{x}')$ is positive semi-definite, the temporal coefficients a_i are mutually orthogonal in $L^2([0, T])$ [Holmes et al., 1998], so that

$$\langle a_i \rangle = 0 \tag{3.15}$$

$$\langle a_i a_j \rangle = \lambda_i \delta_{ij} \tag{3.16}$$

A similar approach can be applied for velocity $\boldsymbol{u}$. Then the average kinetic energy in each POD mode for velocity $\boldsymbol{u}_i$ is given by

$$K_E = \frac{1}{2} \int_\Omega \langle \boldsymbol{u}_i \boldsymbol{u}_i \rangle d\boldsymbol{x} = \frac{1}{2} \langle a_i^{\boldsymbol{u}} a_i^{\boldsymbol{u}} \rangle = \frac{1}{2} \lambda_i \tag{3.17}$$

This last result confirms that the eigenvalue λ_i represents twice the average kinetic energy in that particular mode.

3.3 Decomposition in the temporal domain

Spatial POD modes and their corresponding temporal coefficients share some similar features, which indicate that time and space might be interchangeable insofar as the POD discretization. Similar to the spatial autocorrelation tensor (3.8), the temporal

autocorrelation tensor can be calculated via

$$C(t,t') = (r'(\boldsymbol{x},t), r'(\boldsymbol{x},t')) \tag{3.18}$$

Then the ith temporal coefficient a_i with its corresponding eigenvalue μ_i can be obtained by solving the following eigenvalue problem

$$\frac{1}{T}\int_0^T C(t,t')a_i(t')dt = \mu_i a_i(t) \tag{3.19}$$

Similar to the spatial domain analysis, the temporal coefficients a_i are ordered with respect to the positive eigenvalues in a decending fashion. Moreover, the eigenvalues obtained here are identical to those obtained from the analysis in the spatial domain, i.e. $\lambda_i = \mu_i$. The orthogonality of a_i shows that

$$\langle a_i a_j \rangle = \frac{1}{T}\int_0^T a_i a_j dt = \lambda_i \delta_{ij} \tag{3.20}$$

Using the scaling in equation (3.20), the spatial POD modes r_i can be calculated by the following projection

$$r_i(\boldsymbol{x}) = \frac{1}{\lambda_i}\langle a_i r'(\boldsymbol{x},t)\rangle \tag{3.21}$$

3.4 Method of snapshots

The method of snapshots [Sirovich, 1987] is a POD procedure applied in the temporal domain. Let an ensemble of M snapshots be given at the discrete times t_m, so that

$$r(\boldsymbol{x},t_m) = r_0(\boldsymbol{x}) + r'(\boldsymbol{x},t_m) \tag{3.22}$$

The temporal autocorrelation tensor matrix $\boldsymbol{C}$ can be written as

$$\boldsymbol{C}_{mn} = \frac{1}{M}(r'(\boldsymbol{x}, t_m), r'(\boldsymbol{x}, t_n))_\Omega \quad \text{for } m, n = 1, \ldots, M \tag{3.23}$$

In order to determine the temporal coefficients a_i, the following eigenvalue problem needs to be solved

$$\boldsymbol{C}\boldsymbol{A} = \boldsymbol{D}\boldsymbol{A} \tag{3.24}$$

where $\boldsymbol{C}, \boldsymbol{A}, \boldsymbol{D}$ are all $M \times M$ matrices. $\boldsymbol{D}$ is a diagonal matrix containing eigenvalues of $\boldsymbol{C}$ in descending order as the diagonal elements, and $\boldsymbol{A}$ consists of the eigenvectors $\{\boldsymbol{a}_i\}$ of the eigenvalue problem (3.24) organized in a column major fashion, i.e. $\boldsymbol{A} = (\boldsymbol{a_1}, \boldsymbol{a_2}, \ldots, \boldsymbol{a_i}, \ldots, \boldsymbol{a_M})$. Each eigenvector $\boldsymbol{a}_i$ corresponds to a specific POD mode, and each element in that vector corresponds to the temporal coefficient of a specific snapshot. For $\boldsymbol{a}_i = (a_i^{[1]}, a_i^{[2]}, \ldots, a_i^{[j]}, \ldots, a_i^{[M]})$, $a_i^{[j]}$ represents the temporal coefficient of ith mode in the jth snapshot. To reiterate, the subscript represents mode number, and the superscript represents snapshot number. Temporal coefficients are scaled such that

$$\langle a_i a_j \rangle = \frac{1}{M} \sum_{m=1}^{M} a_i^{[m]} a_j^{[m]} = \lambda_i \delta_{ij} \tag{3.25}$$

Using this scaling, the POD modes can be expressed as

$$r_i = \frac{1}{M\lambda_i} \sum_{m=1}^{M} a_i^{[m]} r'(t_m) \tag{3.26}$$

Note that the autocorrelation tensor generated by the method of snapshots is an $M \times M$ matrix with M being the number of snapshots. Applying an analogous procedure in the spatial domain would produce a matrix of size $N_{\text{grid}} \times N_{\text{grid}}$, where N_{grid} is the total number of grid points in the domain. Because N_{grid} can far exceed M, the method of snapshots is the more common alternative when considering numerical

simulation data. Obviously, for experiments with a limited number of measurement probes and a long sampling time, decomposition in the spatial domain usually proves to be more suitable.

3.5 Galerkin method and model order reduction

Most of the energy in a given flow is contained within a limited number of POD modes. Recall that the average kinetic energy in any given POD mode is linearly related to the eigenvalue for that particular mode. Thus, assuming a total of M snapshots, the proportion of the kinetic energy represented by the first N POD modes is given by

$$E = \left(\sum_{i=1}^{N} \lambda_i \right) \Big/ \left(\sum_{i=1}^{M} \lambda_i \right) \tag{3.27}$$

Then a POD-based reduced-order model (ROM) can be constructed via the Galerkin method by projecting equations (2.1) (2.2) and (2.3) onto the subspace spanned by the first N POD modes of velocity $\boldsymbol{u}$ and density r. Because equation (2.1) is only ever used along with the pressure correction technique to ensure the velocity field remains divergence free for all times, the POD/Galerkin approach only needs to be applied to equations (2.2) and (2.3). Moreover, the density that is actually being numerically evaluated by Diablo is the rescaled dimensionless density r defined in (2.16). Along with the reduced gravity $\boldsymbol{g'}$ defined in (2.15), equations (2.2) and (2.3) can be respectively rewritten as

$$X(\boldsymbol{u}, r) = \frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\boldsymbol{u} + \frac{1}{\rho_0}\nabla p - \nu \nabla^2 \boldsymbol{u} - r\boldsymbol{g'} = 0 \tag{3.28}$$

$$Y(\boldsymbol{u}, r) = \frac{\partial r}{\partial t} - \frac{\nu}{Pr}\nabla^2 r + \boldsymbol{u} \cdot \nabla r = 0 \tag{3.29}$$

where $\boldsymbol{u} = (u, v)$ is the velocity, $\boldsymbol{g'} = (0, -g')$, p is the hydrostatically adjusted pressure, ρ_0 is a reference density, ν is the kinematic viscosity and Pr is the Prandtl number. Note that $X(\boldsymbol{u}, r)$ denotes the Navier-Stokes operator (N-S operator), and $Y(\boldsymbol{u}, r)$ denotes the convection diffusion operator (C-D operator).

After projecting equations (3.28) and (3.29) onto the velocity POD modes $\boldsymbol{u_i}$ and the density POD modes r_i, the resulting ordinary differential equation set that governs the temporal coefficients $a_i^{\boldsymbol{u}}$ and a_i^r reads

$$\frac{da_i^{\boldsymbol{u}}}{dt} = \sum_{j,k=0}^{N} q_{ijk} a_j^{\boldsymbol{u}} a_k^{\boldsymbol{u}} + \nu \sum_{j=0}^{N} l_{ij} a_j^{\boldsymbol{u}} - g' \sum_{j=0}^{N} f_{ij} a_j^r \quad \text{for } i = 1, \ldots, N \tag{3.30}$$

$$\frac{da_i^r}{dt} = \frac{\nu}{Pr} \sum_{j=0}^{N} (df_{ij}) a_j^r + \sum_{j,k=0}^{N} (cv_{ijk}) a_j^r a_k^{\boldsymbol{u}} \quad \text{for } i = 1, \ldots, N \tag{3.31}$$

where (3.30) and (3.31) correspond to the Galerkin projection of (3.28) and (3.29) respectively. Note that, in order to distinguish between the temporal coefficient of velocity and density, $a_i^{\boldsymbol{u}}$ is used to denote the temporal coefficient of velocity whereas a_i^r is used to denote that of density. The calculations for all terms in (3.30) and (3.31) are as follows. For (3.30)

$$(\frac{\partial \boldsymbol{u}}{\partial t}, \boldsymbol{u}_i)_\Omega = \frac{\partial a_i^{\boldsymbol{u}}}{\partial t} = \frac{da_i^{\boldsymbol{u}}}{dt} \tag{3.32}$$

q_{ijk} is the convection term

$$\begin{aligned} -((\boldsymbol{u} \cdot \nabla)\boldsymbol{u}, \boldsymbol{u}_i)_\Omega &= -\left(\left(\sum_{j=0}^{N} a_j^{\boldsymbol{u}} \boldsymbol{u}_j \cdot \nabla \right) \sum_{k=0}^{N} a_k^{\boldsymbol{u}} \boldsymbol{u}_k, \boldsymbol{u}_i \right)_\Omega \\ &= -\sum_{j,k=0}^{N} ((\boldsymbol{u}_j \cdot \nabla)\boldsymbol{u}_k, \boldsymbol{u}_i)_\Omega a_j^{\boldsymbol{u}} a_k^{\boldsymbol{u}} \end{aligned} \tag{3.33}$$

$$q_{ijk} = -((\boldsymbol{u}_j \cdot \nabla)\boldsymbol{u}_k, \boldsymbol{u}_i)_\Omega \tag{3.34}$$

l_{ij} is the viscous term

$$-(-\nu\nabla^2\boldsymbol{u},\boldsymbol{u}_i)_\Omega = \nu\left(\nabla^2\sum_{j=0}^{N}a_j^{\boldsymbol{u}}\boldsymbol{u}_j,\boldsymbol{u}_i\right)_\Omega = \nu\sum_{j=0}^{N}(\nabla^2\boldsymbol{u}_j,\boldsymbol{u}_i)_\Omega a_j^{\boldsymbol{u}} \tag{3.35}$$

$$l_{ij} = (\nabla^2\boldsymbol{u}_j,\boldsymbol{u}_i)_\Omega \tag{3.36}$$

f_{ij} is the body force term

$$-(-r\boldsymbol{g}',\boldsymbol{u}_i)_\Omega = \left(\sum_{j=0}^{N}a_j^r r_j(0,-g'),\boldsymbol{u}_i\right)_\Omega = -g'\sum_{j=0}^{N}(r_j,v_i)_\Omega a_j^r \tag{3.37}$$

$$f_{ij} = (r_j,v_i)_\Omega \tag{3.38}$$

f_i^p is the pressure term

$$-(\nabla p,\boldsymbol{u}_i)_\Omega = -(\nabla\cdot[p\boldsymbol{u}_i])_\Omega = -\int_\Omega \nabla\cdot(p\boldsymbol{u}_i)dS = -\oint_{\partial\Omega}(p\boldsymbol{u}_i)\cdot\boldsymbol{n}dl \tag{3.39}$$

$$f_i^p = -\oint_{\partial\Omega}(p\boldsymbol{u}_i)\cdot\boldsymbol{n}dl = -\left(\oint_{top}+\oint_{bottom}+\oint_{left}+\oint_{right}\right)(p\boldsymbol{u}_i)\cdot\boldsymbol{n}dl \tag{3.40}$$

Though not generally the case, this last term happens to be exacly zero here [Noack et al., 2005]. Diablo incorporates no-slip, no-penetration boundary conditions for velocity on the top and bottom boundaries, which indicates that $\oint_{top}(p\boldsymbol{u}_i)\cdot\boldsymbol{n}dl = \oint_{bottom}(p\boldsymbol{u}_i)\cdot\boldsymbol{n}dl = 0$. Furthermore since the flow is spatially-periodic in x, the velocity and pressure distributions on both sidewalls are identical with normal vectors pointing in opposite directions. Thus $\oint_{left}(p\boldsymbol{u}_i)\cdot\boldsymbol{n}dl + \oint_{right}(p\boldsymbol{u}_i)\cdot\boldsymbol{n}dl = 0$ and $f_i^p = 0$ too. Then for (3.31)

$$(\frac{\partial r}{\partial t},r_i)_\Omega = \frac{\partial a_i^r}{\partial t} = \frac{da_i^r}{dt} \tag{3.41}$$

df_{ij} is the diffusion term

$$-(-\frac{\nu}{Pr}\nabla^2 r, r_i)_\Omega = \frac{\nu}{Pr}\left(\nabla^2 \sum_{j=0}^{N} a_j^r r_j, r_i\right)_\Omega = \frac{\nu}{Pr}\sum_{j=0}^{N}(\nabla^2 r_j, r_i)_\Omega a_j^r \tag{3.42}$$

$$df_{ij} = (\nabla^2 r_j, r_i)_\Omega \tag{3.43}$$

cv_{ijk} is the convection term

$$\begin{aligned} -(\boldsymbol{u}\cdot\nabla r, r_i)_\Omega &= -\left(\sum_{k=0}^{N} a_k^{\boldsymbol{u}}\boldsymbol{u}_k\cdot\nabla\sum_{j=0}^{N} a_j^r r_j, r_i\right)_\Omega \\ &= -\sum_{j,k=0}^{N}(\boldsymbol{u}_k\cdot\nabla r_j, r_i)_\Omega a_j^r a_k^{\boldsymbol{u}} \end{aligned} \tag{3.44}$$

$$cv_{ijk} = -(\boldsymbol{u}_k\cdot\nabla r_j, r_i)_\Omega \tag{3.45}$$

Therefore, the governing equations (3.30) and (3.31) of the ROM that describe the evolution of the temporal coefficients corresponding to each specific mode are obtained. These equations are also know as the Galerkin system [Luchtenburg et al., 2009].

3.6 Reduced order model with boundary control

For control of partial differential equations (PDEs), depending on the location of the actuator, it is often useful to be categorized as either "in domain" control or "boundary" control. For many fluid problems including the density driven flow discussed here, boundary control is considered to be more feasible as it is difficult to implement intrusive actuation and sensing within the domain [Krstic and Smyshlyaev, 2008].

The POD/Galerkin approach for generating the ROM that was outlined in section 3.5 cannot account for the effect of the boundary controller. In order to obtain the ROM for a system with boundary controller, Duhamel's principle [John, 1982]

is utilized to transfer the boundary condition into a forcing term in the governing equations. Because Duhamel's principle applies only to linear differential equations, the governing equations must first be linearized about some suitable steady state. For density driven flow, a realistic boundary control strategy is to change the temperature or heat flux instead of the fluid velocity on the (stationary) boundary, which is, in any event, zero, by the no-slip/no-penetration boundary conditions. For this reason, control is applied only to the density (linearly related to temperature according to equation of state) field through the convection-diffusion equation (3.29). Note that whatever control is applied to density r will also be manifest on velocity $\boldsymbol{u}$ because density and velocity are coupled together through the Navier-Stokes and convection diffusion equations.

As regards equation (3.29), a Dirichlet boundary condition is used for the upper boundary $y = L_y$ by specifying density values directly, while a no flux boundary condition is applied on the bottom boundary $y = 0$. As usual, periodic boundary condition is incorporated in the horizontal direction. In order words, r and r_x have to be continuous between $x = 0$ and $x = L_x$. Only 2D flows are considered within this context, and the flow domain is defined as $\Omega \in [0, L_x] \times [0, L_y]$. The formal statement of the boundary conditions is

$$\begin{aligned} & r(x, L_y, t) = \mathcal{U}(t) f(x) && \\ & && x \in [0, L_x] \\ & r_y(x, 0, t) = 0 && \\ & r(0, y, t) = r(L_x, y, t) && \\ & && y \in [0, L_y] \\ & r_x(0, y, t) = r_x(L_x, y, t) && \end{aligned} \tag{3.46}$$

The top boundary condition $\mathcal{U}(t)f(x)$ is used as the control input. Here a fixed

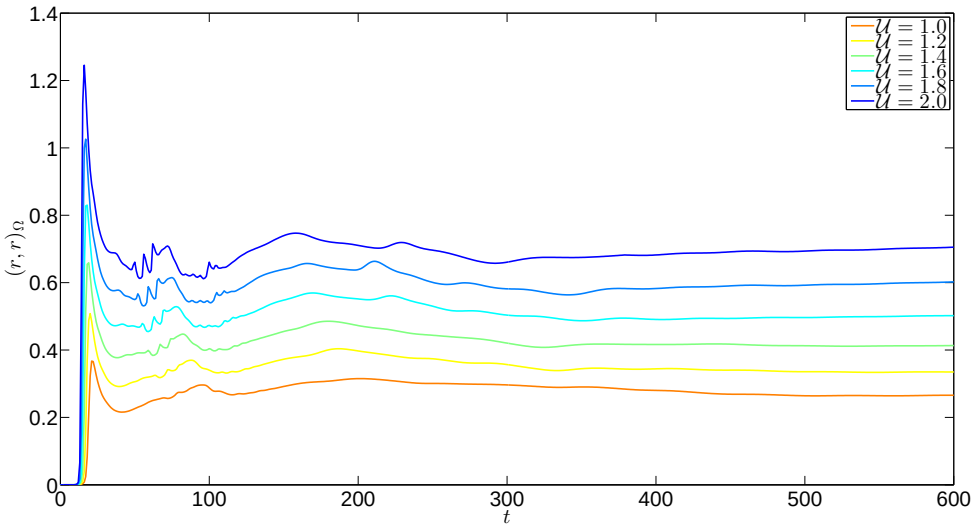

Figure 3.1: $(r, r)_\Omega$ plotted against time for time-invariant top density boundary condition at different $\mathcal{U}$.

function in space, $f(x)$, that is modulated by the control input $\mathcal{U}(t)$.

$$f(x) = -0.5\cos(2\pi x/L_x) - 0.3 \qquad \mathcal{U} \in [0, 2] \tag{3.47}$$

$f(x)$ is carefully chosen in such a way that if $\mathcal{U}$ is set to a constant and $\mathcal{U} \in [0, 2]$, r and $\boldsymbol{u}$ will reach a steady state so that $(r, r)_\Omega$ and $(\boldsymbol{u}, \boldsymbol{u})_\Omega$ (the inner product is defined in (3.1)) become time-invariant for sufficiently large t. As shown in figures 3.1 and 3.2, we find that $t = 500\,\mathrm{s}$ fits this requirement and therefore define the steady state of r and $\boldsymbol{u}$ for a specific $\mathcal{U}$ by their time average fields between 500 s and 600 s. The corresponding steady state density (velocity) field for different constant $\mathcal{U}$ is denoted by $r_\mathcal{U}$ ($\boldsymbol{u}_\mathcal{U}$). Since 500 s is needed for the flow to develop into its steady state, it is advantageous if Diablo can be restart at a given time instant from a existing simulation. The detailed method of achieving this goal is listed in appendix C.

The control strategy is based on a local linearization of the nonlinear convection diffusion equation (3.29). In general, the linearized form of (3.29) is appropriate

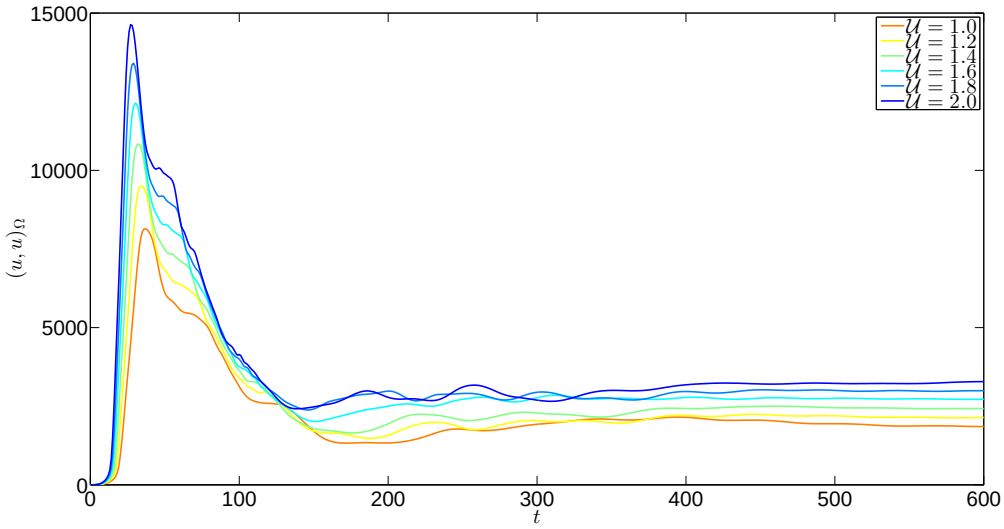

Figure 3.2: $(\boldsymbol{u}, \boldsymbol{u})_\Omega$ plotted against time for time-invariant top density boundary condition at different $\mathcal{U}$.

when the fluid system can be described by a small perturbation around a steady base state. Both density and velocity can be decomposed into a steady base state and a perturbation around that state if $\mathcal{U}(t)$ is close to its linearization point $\mathcal{U}_0$. Without loss of generality, choosing $\mathcal{U}_0 = 1$ and write

$$r(\boldsymbol{x}, t) = r_0(\boldsymbol{x}) + r'(\boldsymbol{x}, t) \tag{3.48}$$

$$\boldsymbol{u}(\boldsymbol{x}, t) = \boldsymbol{u}_0(\boldsymbol{x}) + \boldsymbol{u}'(\boldsymbol{x}, t) \tag{3.49}$$

$$\mathcal{U}(t) = \mathcal{U}_0 + \mathcal{U}'(t) \tag{3.50}$$

$\boldsymbol{u}_0$ and r_0 represent the linearization points for velocity and density, respectively. Then the convection diffusion equation (3.29) can be linearized around $(r_0, \boldsymbol{u}_0)$. Any steady state $(r_\mathcal{U}, \boldsymbol{u}_\mathcal{U})$, including $(r_0, \boldsymbol{u}_0)$, satisfies the governing equation by definition, i.e.

$$\frac{\partial r_0}{\partial t} = 0 = \alpha \nabla^2 r_0 - \boldsymbol{u}_0 \cdot \nabla r_0 \tag{3.51}$$

Meanwhile the linearized equation reads:

$$\frac{\partial r'}{\partial t} = \cancel{\alpha\nabla^2 r_0} + \alpha\nabla^2 r' - \cancel{\boldsymbol{u}_0 \cdot \nabla r_0} - \boldsymbol{u}_0 \cdot \nabla r' - \boldsymbol{u}' \cdot \nabla r_0 - \cancel{\boldsymbol{u}' \cdot \nabla r'} \simeq 0 \tag{3.52}$$

$$\frac{\partial r'}{\partial t} = \alpha\nabla^2 r' - \boldsymbol{u}_0 \cdot \nabla r' - \boldsymbol{u}' \cdot \nabla r_0 \tag{3.53}$$

In order to transfer the perturbation of the top boundary condition, $\mathcal{U}'(t)f(x)$, into the governing equation, Duhamel's principle is applied to the linearized convection diffusion equation (3.53). Following [Christofides, 2001], define

$$r'(\boldsymbol{x}, t) = r(\boldsymbol{x}, t) - r_0(\boldsymbol{x}) = \mathcal{P}(\boldsymbol{x}, t) + b(\boldsymbol{x})\mathcal{U}'(t) \tag{3.54}$$

where $b(\boldsymbol{x})$ is a spatial function and $\mathcal{P}$ is the new modified density whose properties will be discussed below. The snapshots of $\mathcal{P}$ at discrete times t_m are defined by rearranging (3.54)

$$\mathcal{P}(\boldsymbol{x}, t_m) = r(\boldsymbol{x}, t_m) - r_0(\boldsymbol{x}) - b(\boldsymbol{x})\mathcal{U}'(t_m) \qquad 1 \le m \le M \tag{3.55}$$

Substituting equation (3.54) into the first three terms of equation (3.53) and find

$$\begin{aligned} \frac{\partial r'}{\partial t} &= \frac{\partial \mathcal{P}}{\partial t} + bU \\ \alpha\nabla^2 r' &= \alpha\nabla^2 \mathcal{P} + \mathcal{U}'\alpha\nabla^2 b \\ \boldsymbol{u}_0 \cdot \nabla r' &= \boldsymbol{u}_0 \cdot \nabla \mathcal{P} + \mathcal{U}'\boldsymbol{u}_0 \cdot \nabla b \end{aligned} \tag{3.56}$$

where $U \equiv \partial\mathcal{U}'/\partial t$ is the augmented state corresponds to the control $\mathcal{U}$. Notice that, by definition, the top boundary condition of r' is $f(x)\mathcal{U}'(t)$. It is advantageous to choose $b(\boldsymbol{x})$ so as to produce homogenous boundary conditions on the top and bottom boundaries and also to maintain periodicity for the lateral boundary conditions on $\mathcal{P}$

[Christofides, 2001]. Substituting (3.54) into (3.46) yields

$$\begin{aligned} r'(x, L_y, t) &= \mathcal{P}(x, L_y) + b(x, L_y)\mathcal{U}'(t) = f(x)\mathcal{U}'(t) \\ r'_y(x, 0, t) &= \mathcal{P}_y(x, 0) + b_y(x, 0)\mathcal{U}'(t) = 0 \\ r(0, y, t) &= \mathcal{P}(0, y) + b(0, y)\mathcal{U}'(t) \\ &= \mathcal{P}(L_x, y) + b(L_x, y)\mathcal{U}'(t) = r(L_x, y, t) \\ r_x(0, y, t) &= \mathcal{P}_x(0, y) + b_x(0, y)\mathcal{U}'(t) \\ &= \mathcal{P}_x(L_x, y) + b_x(L_x, y)\mathcal{U}'(t) = r_x(L_x, y, t) \end{aligned} \tag{3.57}$$

Note that L_x and L_y represent the length scale of the cavity whereas the subscripts x and y that appear in other terms indicate partial differentiation. By setting $b(x, L_y) = f(x)$, $b_y(x, 0) = 0$, we achieve the desired boundary conditions on $\mathcal{P}$, namely $\mathcal{P}(x, L_y) = 0, \mathcal{P}_y(x, 0) = 0$. By setting $b(0, y) = b(L_x, y)$, $b_x(0, y) = b_x(L_x, y)$, we have $\mathcal{P}(0, y) = \mathcal{P}(L_x, y), \mathcal{P}_x(0, y) = \mathcal{P}_x(L_x, y)$. Therefore, the boundary conditions on $b(x, y)$ are

$$\begin{aligned} &b(x, L_y) = f(x) && x \in [0, L_x] \\ &b_y(x, 0) = 0 && \\ &b(0, y) = b(L_x, y) && y \in [0, L_y] \\ &b_x(0, y) = b_x(L_x, y) && \end{aligned} \tag{3.58}$$

Provided (3.58) is satisfied, $b(x, y)$ can be chosen arbitrarily. For instance selecting $\nabla b = 0$ removes two of the right hand side terms ($\mathcal{U}'\alpha\nabla^2 b$ and $\mathcal{U}'\boldsymbol{u}_0 \cdot \nabla b$) from (3.56). Alternatively, if b satisfies Laplace's equation, i.e. $\nabla^2 b = 0$, only $\mathcal{U}'\alpha\nabla^2 b$ disappears. In this latter case and when combined with the boundary conditions expressed in (3.58) there is a unique solution to $b(x, y)$ for each choice of $f(x)$. The general

form of this solution, obtained using separation of variables, is

$$
\begin{aligned}
b(x,y) &= \frac{\alpha_0}{2} + \sum_{n=1}^{\infty} \left\{ \alpha_n \cos\frac{2n\pi x}{L_x} + \beta_n \sin\frac{2n\pi x}{L_x} \right\} \cosh\frac{2n\pi y}{L_x} \\
\alpha_n &= \frac{2}{L_x \cosh\frac{2n\pi L_y}{L_x}} \int_0^{L_x} f(x) \cos\frac{2n\pi x}{L_x} dx \\
\beta_n &= \frac{2}{L_x \cosh\frac{2n\pi L_y}{L_x}} \int_0^{L_x} f(x) \sin\frac{2n\pi x}{L_x} dx \\
b_x &= \frac{2n\pi}{L_x} \sum_{n=1}^{\infty} \left\{ -\alpha_n \sin\frac{2n\pi x}{L_x} + \beta_n \cos\frac{2n\pi x}{L_x} \right\} \cosh\frac{2n\pi y}{L_x} \\
b_y &= \frac{2n\pi}{L_x} \sum_{n=1}^{\infty} \left\{ \alpha_n \cos\frac{2n\pi x}{L_x} + \beta_n \sin\frac{2n\pi x}{L_x} \right\} \sinh\frac{2n\pi y}{L_x}
\end{aligned}
\tag{3.59}
$$

The advantage of using (3.59) is that it applies for arbitrary integrable $f(x)$. The disadvantage is that $b(x,y)$ is a artificial field which does not reflect the dynamics of the actual flow. Therefore when (3.59) is substituted into (3.55) to generate the snapshots of $\mathcal{P}$, this $b(x,y)$ serves as an extra mean aside from r_0, which causes the first POD mode of $\mathcal{P}$ and its corresponding temporal coefficient approximately equal $b(\boldsymbol{x})$ and $-\mathcal{U}'(t)$ respectively. In other words, the first POD mode of $\mathcal{P}$, instead of capturing the dominant dynamics of the perturbation of the actual flow, ends up capturing the artificial field generated by $b(x,y)$.

In light of this difficulty, we instead seek to define b in such a way that $\mathcal{P}$ represents the density perturbation around the steady state density field $r_{\mathcal{U}}$. Following [Ravindran, 2000], define

$$
b = (r_{\mathcal{U}_1} - r_0)/(\mathcal{U}_1 - \mathcal{U}_0) \tag{3.60}
$$

where $r_{\mathcal{U}_1}$ is a steady state density field obtained when $\mathcal{U}_1 = 2$ and r_0 is the linearization point for density as shown in figure 3.3. By definition, the ratios $r_{\mathcal{U}_1}/\mathcal{U}_1$ and $r_0/\mathcal{U}_0$ satisfy the same boundary conditions specified for b in (3.58). Substitut-

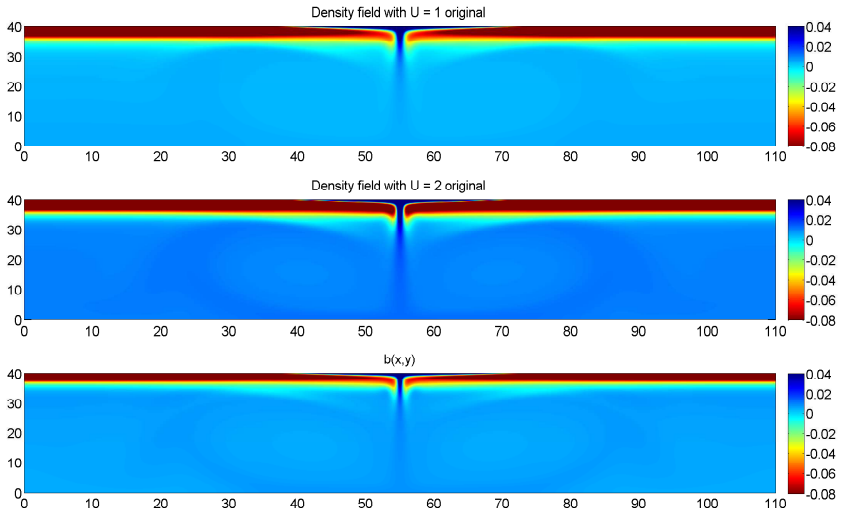

Figure 3.3: (Top) steady state density field r_0 obtained at the linearization point $\mathcal{U} = 1$. (Middle) steady state density field $r_{\mathcal{U}_1}$ obtained when $\mathcal{U}_1 = 2$. (Bottom) density distribution of $b(x, y)$ defined in equation (3.60).

ing (3.60) into (3.58), it can be shown that

$$\begin{aligned} &b(x, L_y) = (f(x)\mathcal{U}_1 - f(x)\mathcal{U}_0)/(\mathcal{U}_1 - \mathcal{U}_0) = f(x) && x \in [0, L_x] \\ &b_y(x, 0) = (0 - 0)/(\mathcal{U}_1 - \mathcal{U}_0) = 0 \\ &b(0, y) = b(L_x, y) && y \in [0, L_y] \\ &b_x(0, y) = b_x(L_x, y) \end{aligned} \tag{3.61}$$

Therefore b as defined in (3.60) satisfies all the boundary conditions in (3.58). When the expression for b given in (3.60) is substituted back into equation (3.54), the following formula for $\mathcal{P}$ is obtained

$$\mathcal{P} = r - \left(r_0 + \mathcal{U}' \frac{r_{\mathcal{U}_1} - r_0}{\mathcal{U}_1 - \mathcal{U}_0}\right) \tag{3.62}$$

As shown in figures 3.4 and 3.5 respectively, $\left(r_0 + \mathcal{U}' \frac{r_{\mathcal{U}_1} - r_0}{\mathcal{U}_1 - \mathcal{U}_0}\right)$ is a good approximation to the stable density field $r_{\mathcal{U}}$ when $\mathcal{U} = 1.3$ and $\mathcal{U} = 1.6$. Other values of

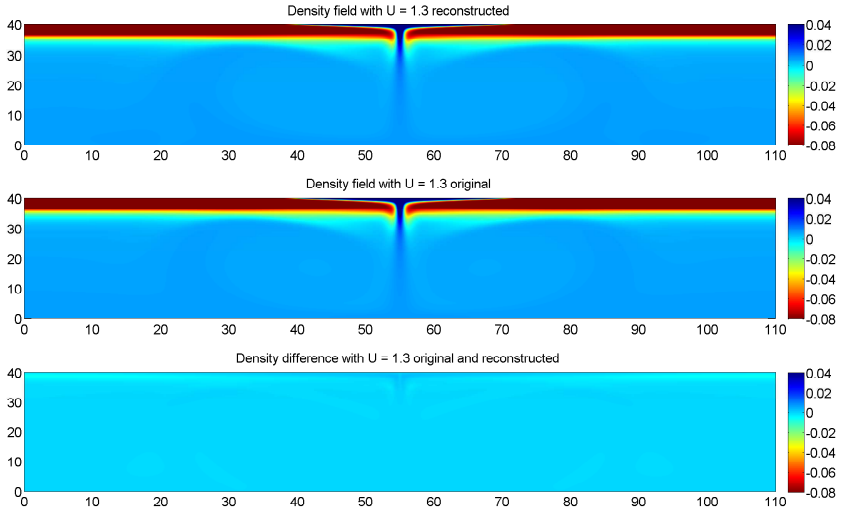

Figure 3.4: (Top) reconstructed steady state density field $r^{\text{recon}}_{\mathcal{U}=1.3}$. (Middle) original steady state density field $r_{\mathcal{U}=1.3}$. (Bottom) the difference between $r^{\text{recon}}_{\mathcal{U}=1.3}$ and $r_{\mathcal{U}=1.3}$.

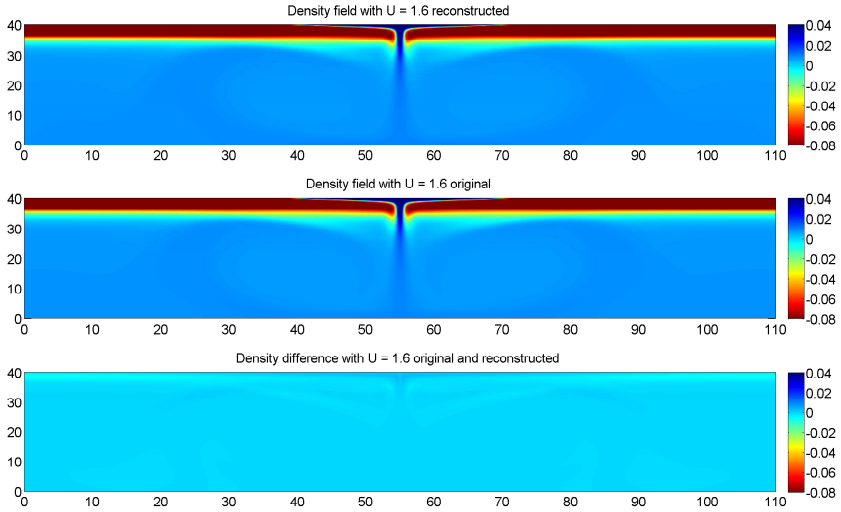

Figure 3.5: (Top) reconstructed steady state density field $r^{\text{recon}}_{\mathcal{U}=1.6}$. (Middle) original steady state density field $r_{\mathcal{U}=1.6}$. (Bottom) the difference between $r^{\text{recon}}_{\mathcal{U}=1.6}$ and $r_{\mathcal{U}=1.6}$.

$\mathcal{U} \in [1, 2]$ (not shown here) exhibit similar results. Therefore, we conclude from equation (3.62) that $\mathcal{P} \simeq r - r_{\mathcal{U}}$. This confirms that our choice of b does indeed force $\mathcal{P}$ to represent the density perturbation around the steady state density field $r_{\mathcal{U}}$.

To capture the dynamics of $\mathcal{P}$ based on the snapshots of $\mathcal{P}$ defined in (3.55), the system must be excited properly by applying a specific control input $\mathcal{U}$. A step function is used here, because it contains a broad spectral content [Oppenheim et al., 1989]. Fluid mechanical systems contain infinite degrees of freedom, however, to keep the problem analytically tractable, it is necessary to limit the number of modes which is done based on equation (3.27). This prescribes the proportion of the flow energy captured by the first N modes. $\mathcal{P}$ can be decomposed as follows

$$\mathcal{P}(\boldsymbol{x}, t) = \sum_{i=1}^{N} a_i(t) r_i(\boldsymbol{x}) \tag{3.63}$$

In order to obtain the expression for r, we substitute (3.63) into (3.54) and rearrange

$$r(\boldsymbol{x}, t) = r_0(\boldsymbol{x}) + \sum_{i=1}^{N} a_i(t) r_i(\boldsymbol{x}) + b(\boldsymbol{x}) \mathcal{U}'(t) \tag{3.64}$$

Although the velocity boundary conditions are, by the no-slip and no-penetration boundary conditions, homogeneous, boundary control applied on the density field will influence the velocity: ρ and $\boldsymbol{u}$ are coupled together in the Navier-Stokes and convection-diffusion equations. Similar to (3.54), define

$$\boldsymbol{u}'(\boldsymbol{x}, t) = \boldsymbol{u}(\boldsymbol{x}, t) - \boldsymbol{u}_0(\boldsymbol{x}) = \boldsymbol{\mathcal{P}}^{\boldsymbol{u}}(\boldsymbol{x}, t) + \boldsymbol{b}^{\boldsymbol{u}}(\boldsymbol{x}) \mathcal{U}'(t) \tag{3.65}$$

so that

$$\begin{aligned} \boldsymbol{u}'(x, L_y, t) &= \boldsymbol{\mathcal{P}}^{\boldsymbol{u}}(x, L_y) + \boldsymbol{b}^{\boldsymbol{u}}(x, L_y) \mathcal{U}'(t) = 0 \\ \boldsymbol{u}'(x, 0, t) &= \boldsymbol{\mathcal{P}}^{\boldsymbol{u}}(x, 0) + \boldsymbol{b}^{\boldsymbol{u}}(x, 0) \mathcal{U}'(t) = 0 \end{aligned} \tag{3.66}$$

By setting $\boldsymbol{b^u}(x, L_y) = 0$, $\boldsymbol{b^u}(x, 0) = 0$, we have $\boldsymbol{\mathcal{P}^u}(x, L_y) = 0$, $\boldsymbol{\mathcal{P}^u}(x, 0) = 0$. Combined with the periodic boundary conditions in the horizontal direction, the boundary condition for $\boldsymbol{b^u}(x, y)$ must satisfy the following

$$\begin{aligned} &\boldsymbol{b^u}(x, L_y) = 0 \\ &\qquad\qquad x \in [0, L_x] \\ &\boldsymbol{b^u}(x, 0) = 0 \\ &\boldsymbol{b^u}(0, y) = \boldsymbol{b^u}(L_x, y) \\ &\qquad\qquad y \in [0, L_y] \\ &\boldsymbol{b^u_x}(0, y) = \boldsymbol{b^u_x}(L_x, y) \end{aligned} \tag{3.67}$$

Provided (3.67) is satisfied, $\boldsymbol{b^u}$ can be chosen arbitrarily. Because control is only applied on r, the velocity field is much more stable than is the density field. As a result, the velocity perturbation $\boldsymbol{u'}$ around the steady state $\boldsymbol{u}_{\mathcal{U}}$ is negligible so that

$$\boldsymbol{u} \simeq \boldsymbol{u}_{\mathcal{U}} \tag{3.68}$$

Substitute (3.68) into (3.69) yields

$$\mathcal{P}^{\boldsymbol{u}} \simeq \boldsymbol{u}_{\mathcal{U}} - \boldsymbol{u}_0 - \boldsymbol{b^u}\mathcal{U}' \tag{3.69}$$

Similar to the definition of b in (3.60), $\boldsymbol{b^u}$ is defined in an analogous fashion

$$\boldsymbol{b^u} = (\boldsymbol{u}_{\mathcal{U}_1} - \boldsymbol{u}_0)/(\mathcal{U}_1 - \mathcal{U}_0) \tag{3.70}$$

where $\boldsymbol{u}_{\mathcal{U}_1}$ is a steady state velocity field with $\mathcal{U}_1 = 2$ and $\boldsymbol{u}_0$ is the linearization point for velocity as shown in figure 3.6 in which the colormap indicates the magnitude of velocity. By definition, the ratios $\boldsymbol{u}_{\mathcal{U}_1}/\mathcal{U}_1$ and $\boldsymbol{u}_0/\mathcal{U}_0$ satisfy the same boundary conditions specified for $b^{\boldsymbol{u}}$ in (3.67). Substituting equation (3.70) into equation (3.69)

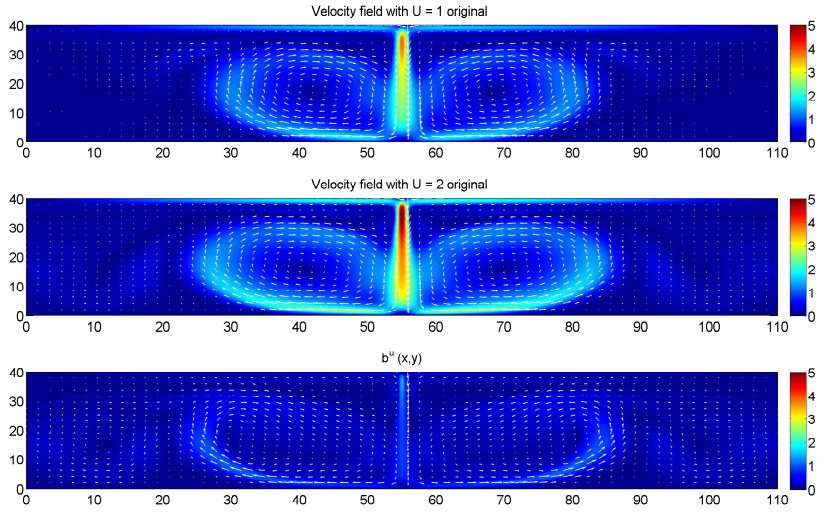

Figure 3.6: (Top) steady state velocity field $\boldsymbol{u}_0$ obtained at the linearization point $\mathcal{U} = 1$. (Middle) steady state velocity field $\boldsymbol{u}_{\mathcal{U}_1}$ obtained when $\mathcal{U}_1 = 2$. (Bottom) velocity profile of $b^{\boldsymbol{u}}(x, y)$ defined in equation (3.70).

results in

$$\mathcal{P}^{\boldsymbol{u}} \simeq \boldsymbol{u}_{\mathcal{U}} - \left(\boldsymbol{u}_0 + \mathcal{U}' \frac{\boldsymbol{u}_{\mathcal{U}_1} - \boldsymbol{u}_0}{\mathcal{U}_1 - \mathcal{U}_0}\right) \tag{3.71}$$

As shown in figures 3.7 and 3.8, $\left(\boldsymbol{u}_0 + \mathcal{U}' \frac{\boldsymbol{u}_{\mathcal{U}_1} - \boldsymbol{u}_0}{\mathcal{U}_1 - \mathcal{U}_0}\right)$ is a good approximation to the stable velocity field $\boldsymbol{u}_{\mathcal{U}}$ when $\mathcal{U} = 1.3$ and $\mathcal{U} = 1.6$. Other values of $\mathcal{U} \in [1, 2]$ (not shown here) exhibit similar results. Therefore

$$\mathcal{P}^{\boldsymbol{u}} \simeq \boldsymbol{u}_{\mathcal{U}} - \boldsymbol{u}_{\mathcal{U}} = 0 \tag{3.72}$$

Similar to $\mathcal{P}$, which is a linear combination of the POD modes of density $\{r_i(\boldsymbol{x})\}_{i=1}^N$ as shown in (3.63), the numerical value of $\mathcal{P}^{\boldsymbol{u}}$ can only be obtained by projecting it onto the subspace spanned by the POD modes of velocity $\{\boldsymbol{u}_i(\boldsymbol{x})\}_{i=1}^N$. However, equation (3.72) indicates that $\mathcal{P}^{\boldsymbol{u}}$ is punitively small, i.e. $\mathcal{P}^{\boldsymbol{u}} \simeq 0$. Therefore according

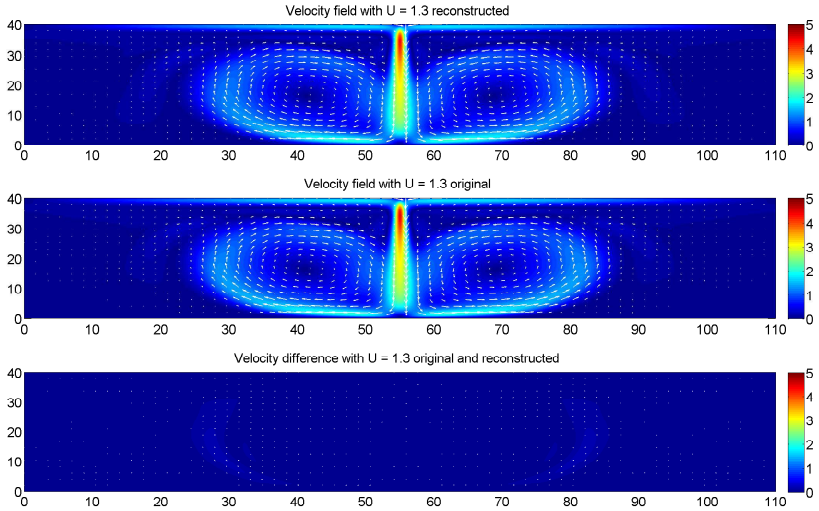

Figure 3.7: (Top) reconstructed steady state velocity field $\boldsymbol{u}^{\text{recon}}_{\mathcal{U}=1.3}$. (Middle) original steady state velocity field $\boldsymbol{u}_{\mathcal{U}=1.3}$. (Bottom) the difference between $\boldsymbol{u}^{\text{recon}}_{\mathcal{U}=1.3}$ and $\boldsymbol{u}_{\mathcal{U}=1.3}$.

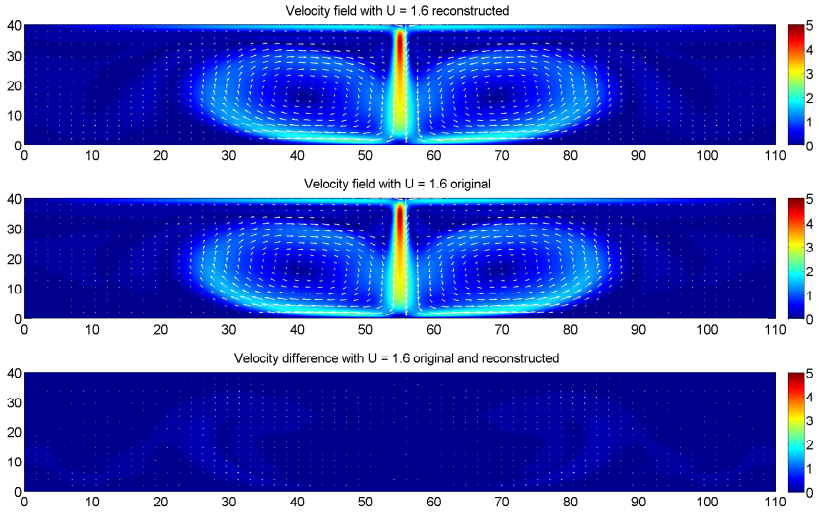

Figure 3.8: (Top) reconstructed steady state velocity field $\boldsymbol{u}^{\text{recon}}_{\mathcal{U}=1.6}$. (Middle) original steady state velocity field $\boldsymbol{u}_{\mathcal{U}=1.6}$. (Bottom) the difference between $\boldsymbol{u}^{\text{recon}}_{\mathcal{U}=1.6}$ and $\boldsymbol{u}_{\mathcal{U}=1.6}$.

to equation (3.69), the velocity perturbation $\boldsymbol{u}'$ only depends on $\mathcal{U}'(t)$, i.e.

$$\boldsymbol{u}' = \boldsymbol{b}^{\boldsymbol{u}}\mathcal{U}' \tag{3.73}$$

In order to rewrite equation (3.53) in terms of b, $b^{\boldsymbol{u}}$ and $\mathcal{P}$, substituting both (3.56) and (3.73) back into (3.53) and rearrange to yield

$$\frac{\partial \mathcal{P}}{\partial t} = \alpha\nabla^2\mathcal{P} - \boldsymbol{u}_0 \cdot \nabla\mathcal{P} + \mathcal{U}'(\alpha\nabla^2 b - \boldsymbol{u}_0 \cdot \nabla b - \boldsymbol{b}^{\boldsymbol{u}} \cdot \nabla r_0) - bU \tag{3.74}$$

Therefore, the ROM can realized by using the Galerkin method to project equation (3.74) onto the subspace spanned by $\{r_i(\boldsymbol{x})\}_{i=1}^N$. The resulting ROM reads

$$\dot{\boldsymbol{X}} = \hat{\boldsymbol{A}}\boldsymbol{X} + \hat{\boldsymbol{B}}U \tag{3.75}$$

where

$$\begin{aligned}
&\boldsymbol{X} = (a_1, a_2, \cdots, a_N, \mathcal{U}')^T \\
&\hat{A}_{ij} = (\alpha\nabla^2 r_j - \boldsymbol{u}_0\nabla r_j, r_i)_\Omega \quad i,j = 1, \cdots, N \\
&\hat{A}_{i(N+1)} = (\alpha\nabla^2 b - \boldsymbol{u}_0\nabla b - \boldsymbol{b}^{\boldsymbol{u}}\nabla r_0, r_i)_\Omega \quad i = 1, \cdots, N \\
&\hat{A}_{(N+1)j} = (0, 0, \cdots, 0) \\
&\hat{B}_i = -(b, r_i)_\Omega \quad i = 1, \cdots, N \\
&\hat{B}_{N+1} = 1
\end{aligned} \tag{3.76}$$

Note that $\hat{\boldsymbol{A}}$ is an $(N+1)\times(N+1)$ matrix, $\hat{\boldsymbol{B}}$ is a $1\times(N+1)$ matrix and U is the augmented state corresponds to the control $\mathcal{U}$.

3.7 Linear Quadratic Regulator

The goal of optimal control theory is to operate dynamic system to minimize a cost functional [Lewis and Syrmos, 1995]. For the case where the system dynamics are described by a set of linear differential equations and the cost function is described by a quadratic function, a Linear Quadratic Regulator (LQR) can be defined. LQR provides a way of calculating the gain $\boldsymbol{K}$ in the control law of $U = -\boldsymbol{K}\boldsymbol{X}$ in full state feedback. Moreover, the appropriate weighting factors for states and control need to be defined in the process of designing the controller[Athans and Falb, 1966]. Discretizing equations (3.75) by implicit Euler results in

$$\boldsymbol{X}_{k+1} = \boldsymbol{A}\boldsymbol{X}_k + \boldsymbol{B}U_k \tag{3.77}$$

where $\boldsymbol{A} = (\boldsymbol{I} - \hat{\boldsymbol{A}}\Delta t)^{-1}$, $\boldsymbol{B} = (\boldsymbol{I} - \hat{\boldsymbol{A}}\Delta t)^{-1}\hat{\boldsymbol{B}}\Delta t$, $U \equiv \partial \mathcal{U}'/\partial t$, with Δt being the time step of the simulation. Here the purpose of the controller is to try to drive the system to zero, its linearization point, which in this case is the desired density field $r_d = r_0$. Therefore the corresponding cost functional $g(r)$ is defined as follows

$$g(r) = \|r - r_d\|_\Omega^2 = \left\| \sum_{i=0}^{N} a_i r_i + b\mathcal{U}' - r_0 \right\|_\Omega^2 = \left\| \sum_{i=1}^{N} a_i r_i + b\mathcal{U}' \right\|_\Omega^2 \tag{3.78}$$

For notational convenience, we denote b by r_{N+1}. Rewriting the above equation into matrix form results in

$$\|r - r_d\|_\Omega^2 = \boldsymbol{X}^T\boldsymbol{Q}\boldsymbol{X} \tag{3.79}$$

where

$\boldsymbol{X} = (a_1, a_2, \cdots, a_N, \mathcal{U}')^T$

$\boldsymbol{Q}_{ij} = (r_i, r_j)_\Omega \quad i, j = 1, \cdots, N+1$

Here a quadratic discrete cost functional can be specified as follows

$$J = \sum_{k=0}^{\infty} (\|r - r_d\|_{\Omega}^2 + RU^2) = \sum_{k=0}^{\infty} (\boldsymbol{X}^T \boldsymbol{Q} \boldsymbol{X} + U^T RU) \tag{3.80}$$

According to the necessary condition of optimality, the feedback law reads

$$U_k = -\boldsymbol{K} \boldsymbol{X}_k \tag{3.81}$$

where

$$\boldsymbol{K} = (\boldsymbol{R} + \boldsymbol{B}^T \boldsymbol{P} \boldsymbol{B})^{-1} (\boldsymbol{B}^T \boldsymbol{P} \boldsymbol{A}) \tag{3.82}$$

and $\boldsymbol{P}$ is the unique positive definite solution to the discrete time algebraic Riccati equation (DARE) defined so that

$$\boldsymbol{P} = \boldsymbol{A}^T \boldsymbol{P} \boldsymbol{A} - (\boldsymbol{A}^T \boldsymbol{P} \boldsymbol{B})(\boldsymbol{R} + \boldsymbol{B}^T \boldsymbol{P} \boldsymbol{B})^{-1} (\boldsymbol{B}^T \boldsymbol{P} \boldsymbol{A}) \tag{3.83}$$

Here the feedback controller gain $\boldsymbol{K}$ is a constant vector and is calculated off-line in advance.

Chapter 4

Results and Discussion

4.1 Reduced order model and system identification

The reduced order model (ROM) is constructed by applying the POD/Galerkin approach discussed in section 3.5. In order to identify the system dynamics, the step function of figure 4.1, is used as the input $\mathcal{U}(t)$ because of its simple nature and broadband spectral content. The change of value of $\mathcal{U}(t)$ is not applied till after 500 s, which allows the system to achieve steady state.

Thereafter, the method of snapshots is applied and the snapshots are collected every second. Recall from section 2.5, that the time step of the simulation Δt is defined to be 0.004s, as a result of which, the snapshots are collected every 250 time

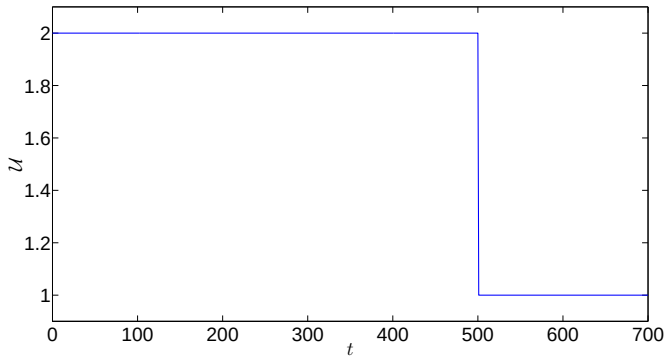

Figure 4.1: Input $\mathcal{U}(t)$ used for system identification.

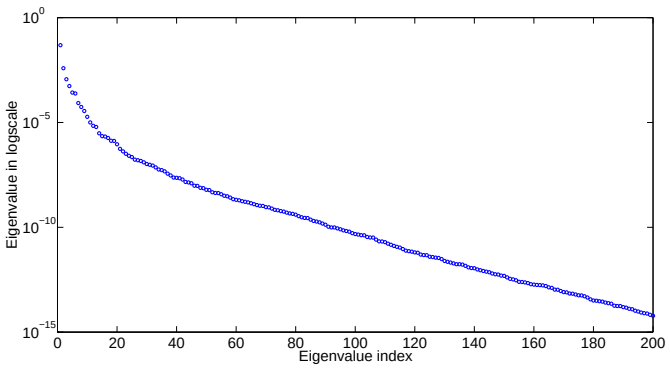

Figure 4.2: Eigenvalues corresponding to the input shown in figure 4.1

Table 4.1: Percent of energy captured as a function of POD mode number.

Number of POD modes N	2	4	6	8	10
Fraction of Energy (%)	72.17	93.92	99.01	99.63	99.88

steps. For the time interval between 500 s and 700 s, 200 snapshots are collected, therefore the eigenvalue spectrum consists of 200 eigenvalues of the density field r is shown in figure 4.2. The magnitude of the eigenvalues falls off quickly, suggesting that a limited number of modes are sufficient to capture the dominant dynamics of the flow.

The fraction of energy captured by the first N POD modes of the ROM can be calculated using equation (3.27); results corresponding to figure 4.2 are given in table 4.1. These data indicate that 99% of the energy can be captured using the first six POD modes. Considering there is only a 0.62% improvement in increasing the number of POD modes from six to eight whereas the computational cost of this increase is far greater, six POD modes are used hereafter. The density field associated with each of the first six POD modes are shown in figure 4.3. Superposed on top of density field is the velocity field, which is drawn in the form of a quiver plot.

In order to test the fidelity of the ROM, comparisons are made between the evolution history of the directly projected temporal coefficients and those determined via

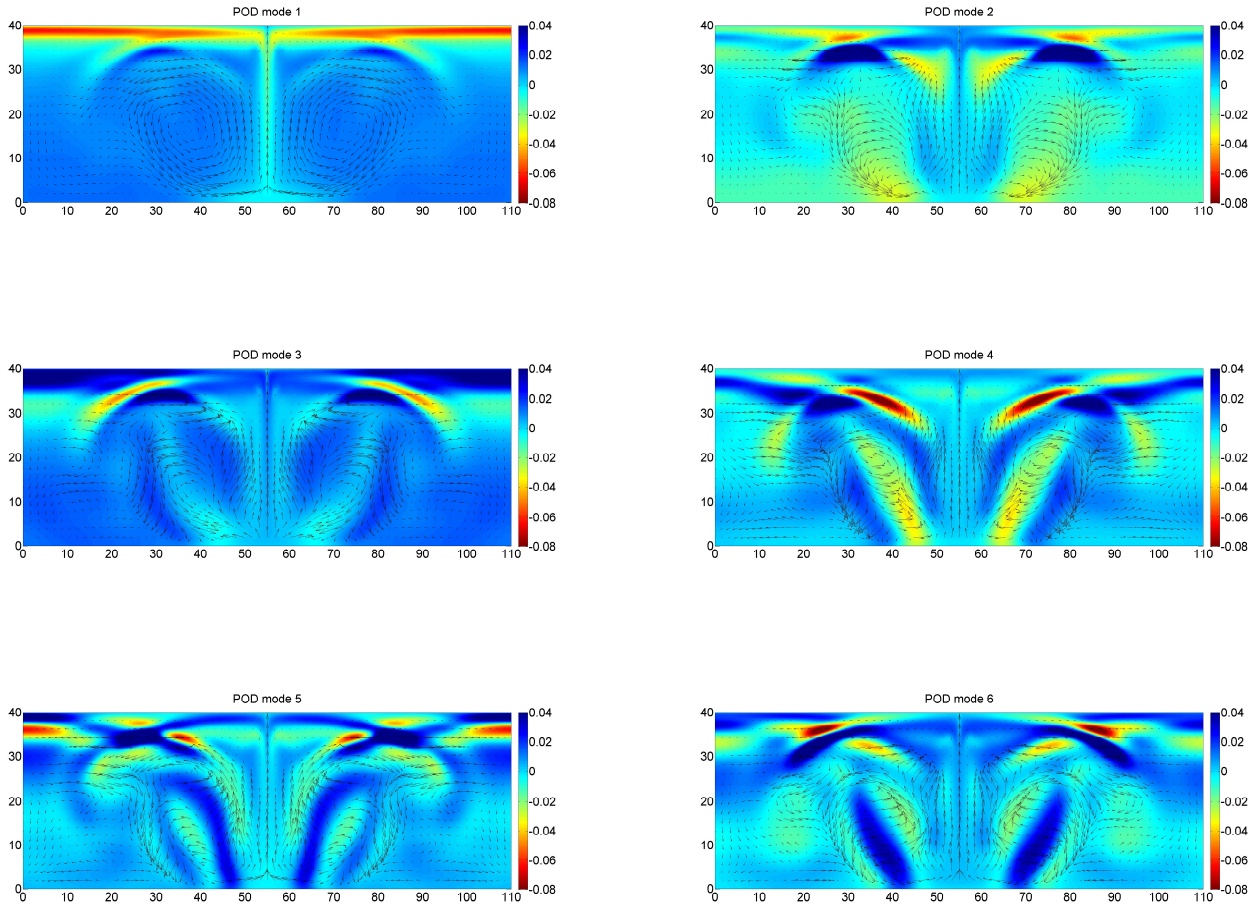

Figure 4.3: The density distribution (in colormap) and velocity profile (in quiverplot) of the first six POD modes corresponding to the step function input.

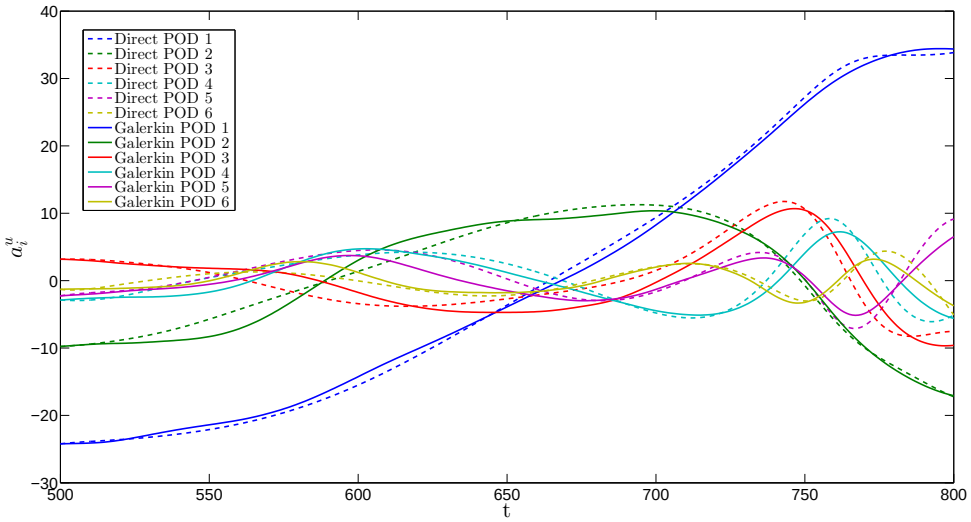

Figure 4.4: Comparison of the directly projected temporal coefficients (solid) and Galerkin integrated temporal coefficients (dashed) for velocity.

the Galerkin method. The former are generated by the projection of the numerical model (Diablo) onto the first six POD modes, and the latter are generated by integrating the governing equations (3.30) and (3.31) of ROM. The associated comparison is shown in figure 4.4 where strong agreement is noted for all values of POD mode number. The analogue comparison for the density temporal coefficient a_i^r is presented in figure 4.5. The agreement remains positive, but some slightly larger deviations are observed for the even-numbered POD modes when $550\text{s} < t < 650\text{s}$.

4.2 LQR controller design for buoyancy driven flow

Recall the definition of the steady state for density field $r_{\mathcal{U}}$ in section 3.6. The purpose of the LQR controller is to drive the flow from one steady state of the density field $r_{\mathcal{U}}$ to another. Since the linearization point r_0 (of our linearized model) is itself a steady state, the specific objective of our controller is to drive the flow from steady state $r_{\mathcal{U}}$ to the linearization point r_0 (desired final state). The control action is achieved

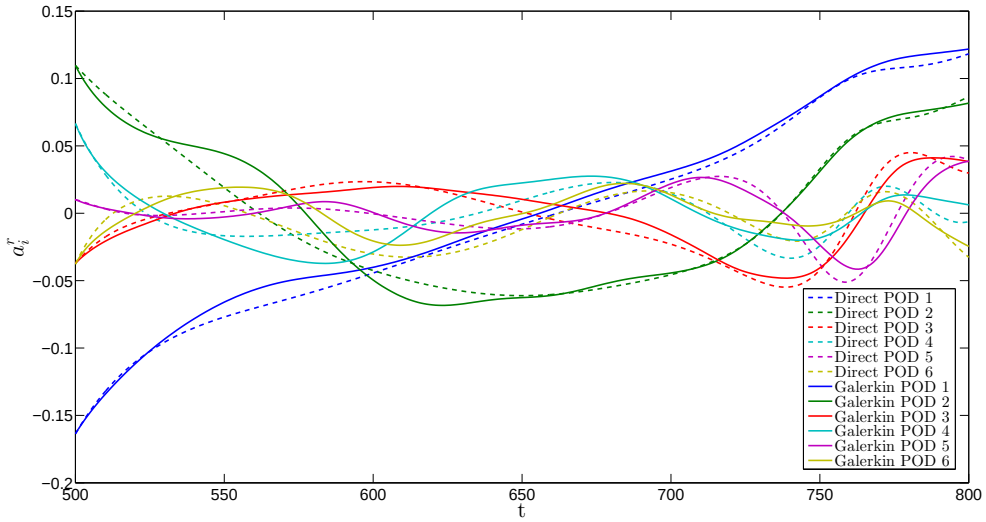

Figure 4.5: Comparison of the directly projected temporal coefficients (solid) and Galerkin integrated temporal coefficients (dashed) for density.

through the top boundary condition for density, which is expressed as

$$r(x, L_y, t) = f(x)\mathcal{U}(t) \qquad x \in [0, L_x] \tag{4.1}$$

where $\mathcal{U}(t)$ and $f(x)$ represent respectively the temporal dependence and spatial distribution. Due to the complexity of the flow, it is useful to have a performance index to provide a simple metric to evaluate the quality of the control. The choice of performance index to meet the control objective of driving the flow to the linearization point is non-trivial. A performance index of the form

$$\mathcal{J}(t) = \|r(t) - r_0\|_\Omega^2 \tag{4.2}$$

is chosen. Here r_0 is the linearization point, therefore the control purpose is the same as driving the value of the performance index $\mathcal{J}(t)$ to zero. Since the POD form of r is given by equation (3.64), the performance index $\mathcal{J}(t)$ can be rewritten in terms of

the POD modes of density $\{r_i\}_{i=1}^N$ in matrix form, i.e.

$$\mathcal{J}(t) = \boldsymbol{X}^T \boldsymbol{Q} \boldsymbol{X} \tag{4.3}$$

According to the orthonormality of $\{r_i\}_{i=1}^N$, $(r_i, r_j)_\Omega = \delta_{ij}$. Applying this result, we have

$$\boldsymbol{Q}_{ii} = (r_i, r_i)_\Omega = 1 \quad i = 1, \cdots, N$$
$$\boldsymbol{Q}_{ij} = (r_i, r_j)_\Omega = 0 \quad i, j = 1, \cdots, N \quad i \neq j$$

Note, however, that the value of $(r_i, r_{N+1})_\Omega = (r_i, b)_\Omega$ still needs to be calculated. Since b is defined in (3.60) and the first six POD modes are used, $\boldsymbol{Q}$ reads

$$\boldsymbol{Q} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 2.66 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0.68 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1.54 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0.25 \\ 0 & 0 & 0 & 0 & 1 & 0 & -0.50 \\ 0 & 0 & 0 & 0 & 0 & 1 & -0.51 \\ 2.66 & 0.68 & -1.54 & 0.25 & -0.50 & -0.51 & 20.2 \end{bmatrix}$$

Here $\boldsymbol{Q}$ is positive definite. Therefore the LQR cost functional of the infinite-horizon LQR problem is

$$J(\boldsymbol{X}, U) = \sum_{k=0}^{\infty} (\|r - r_d\|_\Omega^2 + RU^2) = \sum_{k=0}^{\infty} (\boldsymbol{X}^T \boldsymbol{Q} \boldsymbol{X} + U^T R U) \tag{4.4}$$

The task of the LQR controller is to compute $\mathcal{U}(t)$ such that the cost functional $J(\boldsymbol{X}, U)$ is minimized subject to the constraint that the flow fields satisfy the system

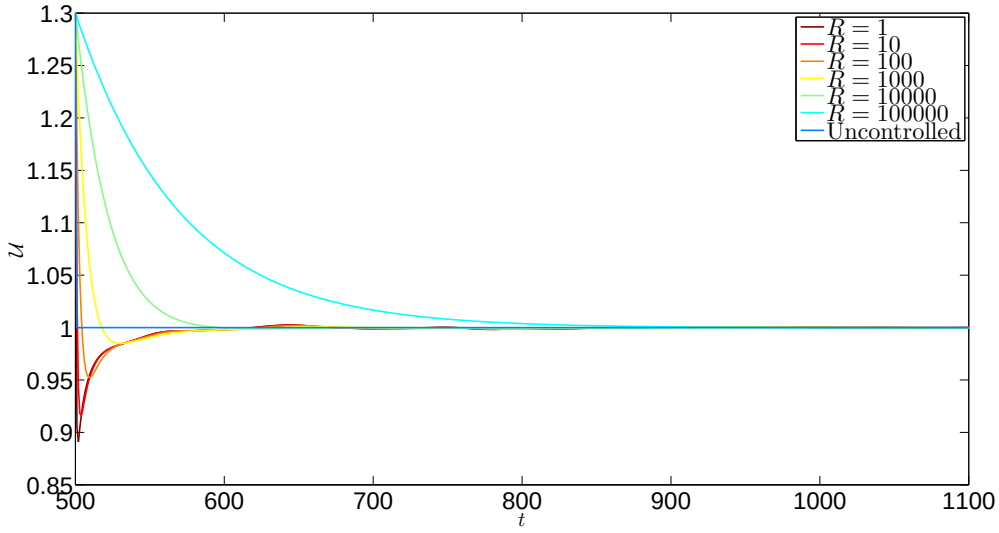

Figure 4.6: Evolution curves of the LQR controller input $\mathcal{U}(t)$ with the starting steady state $r_{\mathcal{U}}$ fixed at $\mathcal{U} = 1.3$ for different penalty parameters R.

dynamics described by equation (3.75). By including a term involving the augmented state U ($\equiv \partial\mathcal{U}'/\partial t$) in (4.4), one minimizes the rate of change associated with the forcing by the upper boundary condition $f(x)(\mathcal{U}_0 + \mathcal{U}'(t))$. The penalty parameter $R > 0$ adjusts the relative weight of the two terms in the functional.

In order to determine an appropriate value for the penalty parameter R, which was found to play a critical role in the controller design, the starting steady state is fixed at $r_{\mathcal{U}=1.3}$. $T_0 = 500\,\mathrm{s}$ is the time instant where the controller "kicks" in and $T_f = 1100\,\mathrm{s}$ is the final time instant for the simulation. As shown in figure 4.6, when $R \leq 10^3$, the control input initially undershoots. As R increases the amplitude of the undershoot is observed to decrease. So that when $R = 10^4$, a monotone control input is realized. Of course, similar comments apply to $R \geq 10^5$. Here, however, the controller is slower than with $R = 10^4$. For comparison purposes, an uncontrolled case that uses an open-loop step down input is also included which is described in more detail below.

Note that, $\mathcal{J}(T_f)$ represents the value of the performance index at the end point

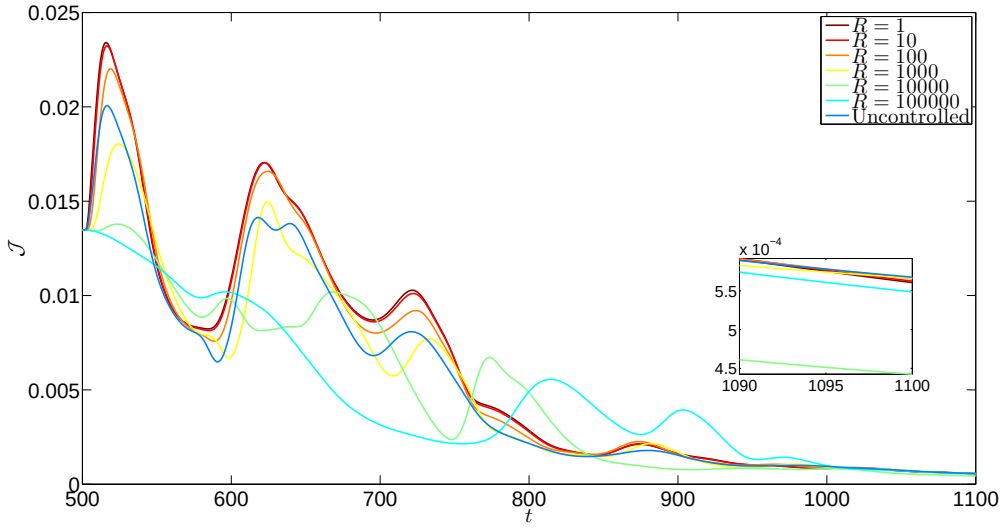

Figure 4.7: Evolution curves of the performance index $\mathcal{J}(t)$ with the starting steady state $r_{\mathcal{U}}$ fixed at $\mathcal{U} = 1.3$ for different penalty parameters R.

Table 4.2: Performance index $\mathcal{J}(t)$ at T_f and its time average in the time interval $T_0 < t < T_f$ as a function of the penalty parameter R ($T_0 = 200s$).

R	$\mathcal{J}(T_f)$	$\bar{\mathcal{J}}(T_0, T_f)$
1	5.607×10^{-4}	0.0062
10	5.624×10^{-4}	0.0061
10^2	5.657×10^{-4}	0.0059
10^3	5.663×10^{-4}	0.0053
10^4	4.425×10^{-4}	0.0048
10^5	5.487×10^{-4}	0.0049
uncontrolled	5.676×10^{-4}	0.0054

$t = T_f$; the smaller the value of $\mathcal{J}(T_f)$, the closer the density field is to the linearization point r_0. Moreover, let us define

$$\bar{\mathcal{J}}(T_0, T_f) = \left(\int_{T_0}^{T_f} \mathcal{J}(t)dt \right) / (T_f - T_0) \tag{4.5}$$

where $\bar{\mathcal{J}}(T_0, T_f)$ represents the time average of performance index in the interval $T_0 < t < T_f$. The smaller $\bar{\mathcal{J}}(T_0, T_f)$ is, the closer the density field is to the linearization point r_0 during the entire transition stage. As shown in table 4.2 and figure 4.7, when $R = 10^4$, the smallest value of both $\mathcal{J}(T_f)$ and $\bar{\mathcal{J}}(T_0, T_f)$ are achieved. This result indicates that $R = 10^4$ gives the best overall performance in terms of driving the system to its linearization point. As such, 10^4 will serve as the benchmark value of R in much of the discussion below.

4.3 Controller performance for different starting steady states

Figures 4.6 and 4.7 consider the system dynamics for variable R but fixed starting steady state $r_{\mathcal{U}=1.3}$. Having identified $R = 10^4$ as an approximately optimum value which minimize the performance index $\mathcal{J}$ defined in (4.2), the starting steady state $r_{\mathcal{U}}$ and the corresponding $\mathcal{U}$ is now varied at $t = T_0$, as shown in figure 4.8. Here T_0 and T_f remain unchanged compared to section 4.2.

When $\mathcal{U}(T_0) \leq 1.4$, the evolution curves of the performance index $\mathcal{J}$ plateau just above zero after 1000 s, which means that the LQR controller manage to drive the flow from the starting steady state $r_{\mathcal{U}}$ to the linearization point r_0 (desired final state). As for the comparison between the controlled (solid curves) and uncontrolled (dashed curves) cases, note that the uncontrolled case exhibits a more significant transient overshoot for t just larger than 500 s. Although both sets of curves show some oscillatory behavior, the dashed curves (uncontrolled cases) are associated with larger oscillation amplitudes. However, for $\mathcal{U}(T_0) = 1.5$, the situation reverses. Although

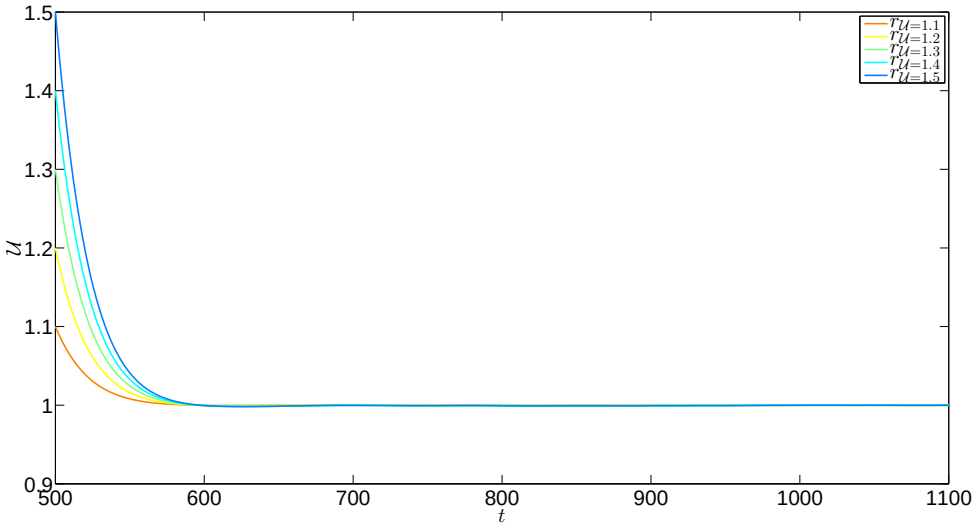

Figure 4.8: Evolution curves of the LQR controller input $\mathcal{U}(t)$ with different starting steady states $r_{\mathcal{U}}$.

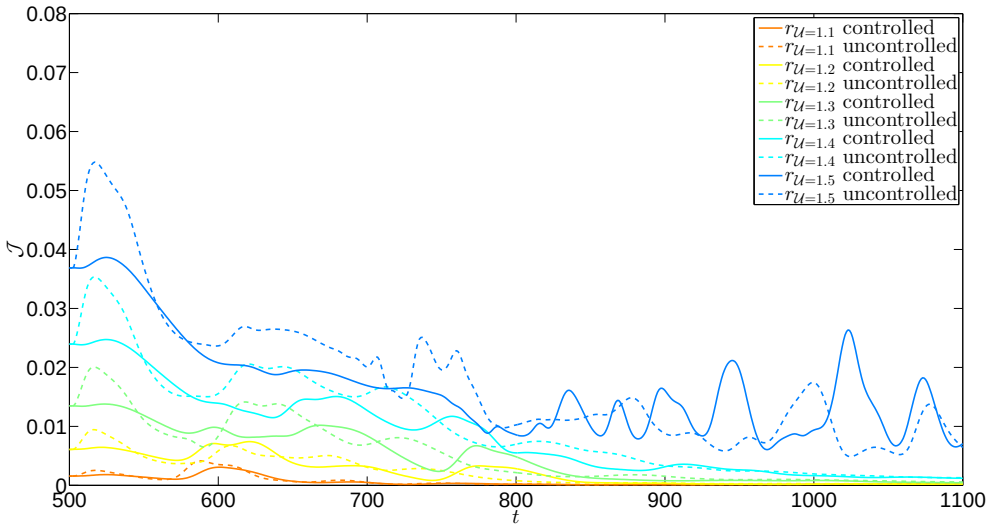

Figure 4.9: Evolution curves of the performance index $\mathcal{J}(t)$ with different starting steady states $r_{\mathcal{U}}$. The solid curves, representing the controlled cases (LQR) are plotted against those of the uncontrolled cases (open-loop) in the same color for comparison. The insert shows the comparison amongst $\mathcal{J}(T_f)$ with different value for R.

Table 4.3: Performance index for controlled (LQR) and uncontrolled (open-loop) cases at T_0 and T_f for starting steady states $r_{\mathcal{U}}$ corresponding to different $\mathcal{U}$ ($T_0 = 500s$).

$\mathcal{U}(T_0)$	$\mathcal{J}(T_0)$	$\mathcal{J}_c(T_f)$	$\mathcal{J}_u(T_f)$
1.1	0.0016	1.71×10^{-4}	1.94×10^{-4}
1.2	0.0061	2.26×10^{-4}	2.42×10^{-4}
1.3	0.0135	4.42×10^{-4}	5.68×10^{-4}
1.4	0.0240	0.0012	0.0013
1.5	0.0369	–	–

the dashed curve exhibits a larger overshoot at the beginning, the solid curve shows more pronounced oscillations thereafter. This is not an unexpected result and is attributed to the ROM being linearized about r_0 with $\mathcal{U}(T_0) = 1.0$. By contrast with $\mathcal{U}(T_0) = 1.5$, non-linear effects, ignored in the model development, are likely strong enough to compromise the performance of the controller. Also, as suggested by figure 4.10, the oscillations realized after 800 s for the dark blue curve of figure 4.9 may be due to a qualitative change in the nature of the plume and a transition from symmetric to asymmetric flow. Once the asymmetry within the plume is strong enough, there is no way to drive the system back to a symmetric state using the current (symmetric) upper boundary condition. Consistent with these observations, figure 4.9 indicates that the evolution curve for $\mathcal{J}$ never plateau for the cases where $\mathcal{U}(T_0) = 1.5$.

Table 4.3 shows the performance index $\mathcal{J}$ of both controlled and uncontrolled cases at T_0 and T_f for different $\mathcal{U}(T_0)$ values. Here $\mathcal{J}_c(T_f)$ and $\mathcal{J}_u(T_f)$ are the performance indices for the controlled and uncontrolled cases respectively at the final time T_f. The entries for $\mathcal{J}_c(T_f)$ and $\mathcal{J}_u(T_f)$ at $\mathcal{U}(T_0) = 1.5$ are left empty because of the oscillatory behavior noted for the dark blue curves of figure 4.9. Since controlled cases and uncontrolled cases start from the same point in time, $\mathcal{J}_c(T_0) = \mathcal{J}_u(T_0) = \mathcal{J}(T_0)$. Based on the data provided in table 4.3, the effectiveness of the controller can be evaluated

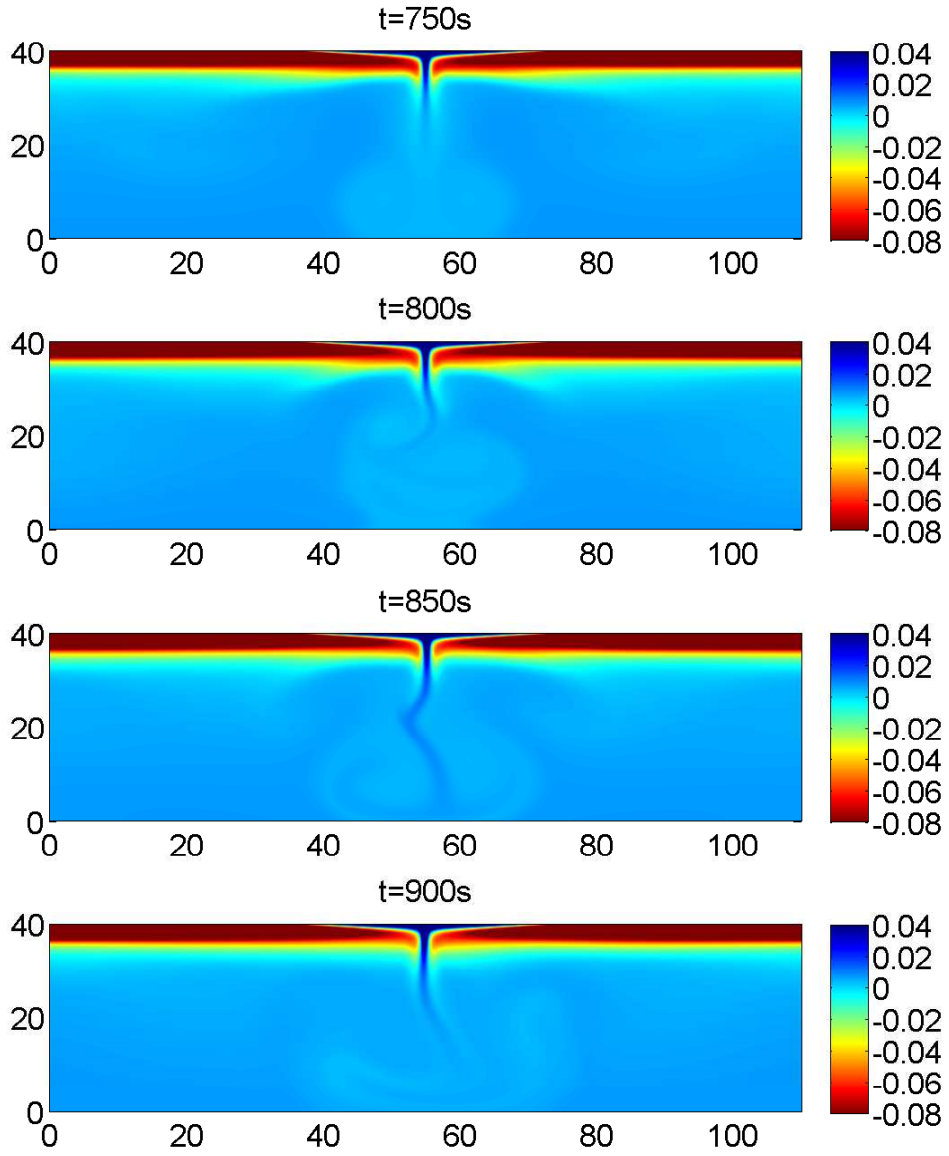

Figure 4.10: Density field for the controlled case with the starting steady state $r_{\mathcal{U}}$ at $\mathcal{U} = 1.5$ for four different time instants $t = 750s$, $t = 800s$, $t = 850s$ and $t = 900s$. Note that the plume develops an asymmetric character between 750 s and 800 s.

Table 4.4: Absolute and relative effectiveness of the LQR controller for $T_0 = 500s$.

$\mathcal{U}(T_0)$	η_a	η_r
1.1	89.3%	11.7%
1.2	96.3%	6.6%
1.3	96.7%	22.2%
1.4	95.0%	7.7%

in two ways.

First, recall that the control objective is to reduce the value of the performance index $\mathcal{J}$ to zero. Therefore the effectiveness of the controller can be evaluated based on its ability to reduce the magnitude of $\mathcal{J}$. In this spirit, define

$$\eta_a = [\mathcal{J}(T_0) - \mathcal{J}_c(T_f)]/\mathcal{J}(T_0) \tag{4.6}$$

η_a, referred to as the absolute effectiveness of the LQR controller, shows how much closer the density field is to the linearization point r_0, the desired final state, at $t = T_f$ compared with $t = T_0$. Secondly, the effectiveness of the controller can be evaluated based on the comparison between the controlled cases and uncontrolled cases. Therefore, define

$$\eta_r = [\mathcal{I}_u(T_f) - \mathcal{I}_c(T_f)]/\mathcal{I}_u(T_f) \tag{4.7}$$

η_r, referred to as the relative effectiveness of the LQR controller, shows how much better the LQR controller is compared with the open-loop controller.

Based on the data from table 4.3, η_a and η_r for different $\mathcal{U}(T_0)$ are calculated and listed in table 4.4. From table 4.4, we can conclude that when $\mathcal{U}(T_0) \leq 1.4$, the LQR controller manages to reduce the performance index by 89% or more, and the LQR controller is always better than the open-loop controller because η_r is always positive.

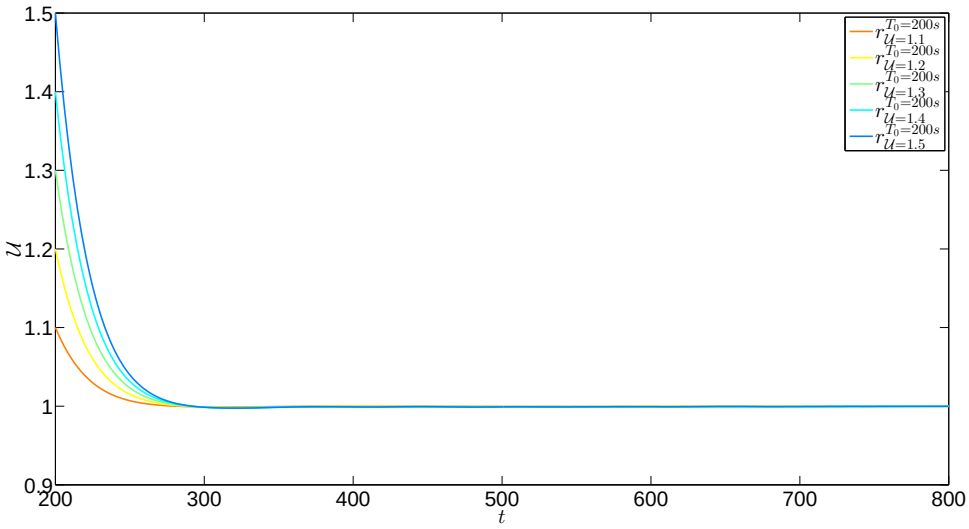

Figure 4.11: Evolution curves of the LQR controller input $\mathcal{U}(t)$ starting from transient states $r_{\mathcal{U}}^{T_0}$ with fixed $T_0 = 200s$ and different $\mathcal{U}$.

Recall the penalty parameter $R = 10^4$ is designed for $\mathcal{U}(T_0) = 1.3$, so it is to be expected that the highest values for both η_a and η_r are achieved at $\mathcal{U}(T_0) = 1.3$. This indicates that possible improvements can be achieved for other $\mathcal{U}(T_0)$ by specifically tailoring the penalty parameter R.

4.4 Controller performance for transient flow

In all cases considered thus far, the controller is activated after 500 s and the flow is the steady state denoted by $r_{\mathcal{U}}$. In this section, the control is applied earlier and before steady state conditions have been realized. The transient state corresponding to a constant $\mathcal{U}$ and a starting time instant T_0 ($T_0 < 500s$) is denoted by $r_{\mathcal{U}}^{T_0}$. As shown in figure 4.11, the starting transient state $r_{\mathcal{U}}^{T_0}$ and the corresponding $\mathcal{U}$ is varied at $t = T_0$. The control is turned on at $T_0 = 200$ s and final time instant $T_f = 800$ s so that the duration over which the control is applied remains 600 s.

In this case, the LQR control is more attractive compared to steady state case because the transient flow have more energy which is actively damped by the con-

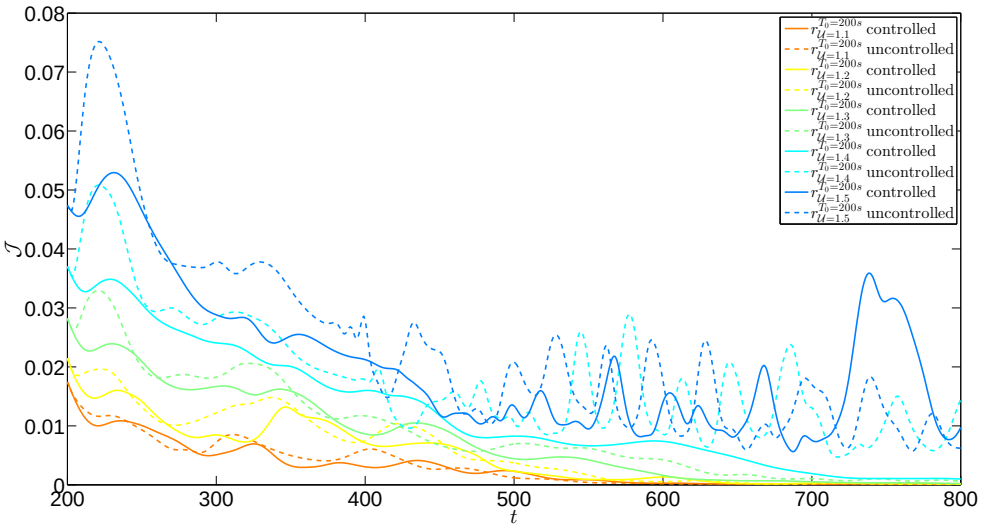

Figure 4.12: Evolution curves of the performance index $\mathcal{J}(t)$ starting from transient states $r_{\mathcal{U}}^{T_0}$ with fixed $T_0 = 200s$ and different $\mathcal{U}$. The solid curves, representing the controlled cases (LQR) are plotted against those of the uncontrolled cases (open-loop) in the same color for comparison.

troller. When $\mathcal{U}(T_0) \leq 1.3$, the evolution curves of $\mathcal{J}$ plateau just above zero about 700 s – see figure 4.9. As before, the controller is effective in driving the flow from the initial to the desired final (steady) state. As for the case where $\mathcal{U}(T_0) = 1.4$, the LQR controller still manages to drive the flow to the desired final steady state without oscillation. By contrast, for the corresponding uncontrolled case indicated by the teal dashed curve, large oscillations are observed, indicating that the open loop controller is no longer effective when $\mathcal{U}(T_0) = 1.4$. Finally when $\mathcal{U}(T_0) = 1.5$, the results are consistent with analogue curves from figure 4.9 and confirm that the model linearization assumption does not extend to values of $\mathcal{U}(T_0)$ as large as 1.5.

Table 4.5 shows the performance index $\mathcal{J}$ of both controlled and uncontrolled cases at T_0 and T_f for different $\mathcal{U}(T_0)$ values for the transient flow. Blank entries are for the same reason as table 4.3. The absolute and relative effectiveness for the transient case are calculated based on the data in table 4.5, and the results are shown in table 4.6. When $\mathcal{U}(T_0) \leq 1.4$, the LQR controller manages to reduce the performance

Table 4.5: Performance index for controlled (LQR) and uncontrolled (open-loop) cases at T_0 and T_f for starting steady states $r_{\mathcal{U}}$ corresponding to different $\mathcal{U}$ ($T_0 = 200s$).

$\mathcal{U}(T_0)$	$\mathcal{J}(T_0)$	$\mathcal{J}_c(T_f)$	$\mathcal{J}_u(T_f)$
1.1	0.0175	1.23×10^{-4}	1.43×10^{-4}
1.2	0.0215	3.24×10^{-4}	3.97×10^{-4}
1.3	0.0282	5.21×10^{-4}	0.0010
1.4	0.0370	0.0018	–
1.5	0.0474	–	–

Table 4.6: Absolute and relative effectiveness of the LQR controller for $T_0 = 200s$.

$\mathcal{U}(T_0)$	η_a	η_r
1.1	99.3%	16.3%
1.2	98.5%	18.4%
1.3	98.1%	47.9%
1.4	95.1%	–

index by over 95% and even up to 99%. η_r at $\mathcal{U}(T_0) = 1.4$ is left blank because meaningful comparisons cannot be made when the LQR controller manages to drive the system to the desired final state but the open-loop controller fails to do so (see figure 4.12). When $\mathcal{U}(T_0) \leq 1.3$, η_r at $\mathcal{U}(T_0) = 1.3$ is almost three times the value of η_r at $\mathcal{U}(T_0) = 1.1$ and $\mathcal{U}(T_0) = 1.2$. This indicates that the penalty parameter deigned for $\mathcal{U} = 1.3$ is sub optimal for other values of $\mathcal{U}$.

Comparing figure 4.9 (steady case) with figure 4.12 (transient case), the most significant difference observed is for the teal dashed curve representing the uncontrolled case for $\mathcal{U}(T_0) = 1.4$. In figure 4.9, the teal dashed curve plateaus after 1000 s just above zero whereas large oscillations are observed for the teal dashed curve after 450 s in figure 4.12. This result indicates that the open-loop controller works reasonably well when $T_0 = 500$ s (flow in steady state), but its not effective when $T_0 = 200$ s (flow in transient state). To further investigate this phenomenon, the performance of the

LQR controller is compared to the open loop controller for five starting times within time interval $200s < T_0 < 500s$. These different starting times effectively represent different starting state $r^{T_0}_{\mathcal{U}=1.4}$.

Additional numerical simulations are conducted at $T_0 = 275\,\mathrm{s}$, $T_0 = 350\,\mathrm{s}$ and $T_0 = 425\,\mathrm{s}$ for $\mathcal{U}(T_0) = 1.4$. As shown in figure 4.13, despite the difference in T_0, the LQR controller stabilizes the flow in all three cases. For the uncontrolled cases with open-loop step-down input, the flow does not monotonically decay to the steady state depending on the starting time T_0. For the dashed curve representing the uncontrolled case, when $T_0 = 275\,\mathrm{s}$, the number of oscillations decrease but the amplitude of the oscillations, especially the last few, increase compared with the $T_0 = 200\,\mathrm{s}$ case. When $T_0 = 350\,\mathrm{s}$, $T_0 = 425\,\mathrm{s}$ and $T_0 = 500\,\mathrm{s}$, the curves are almost identical and both solid and dashed curves plateau just above zero within the 600 s of control. In addition, the starting value of $\mathcal{J}$ decreases as T_0 increases. In other words, the later the controller "kicks" in, the closer the starting density field is to the final desired density field. In conclusion, when $\mathcal{U}(T_0) = 1.4$, the LQR controller is always effective. But the stabilizing effect of open-loop controller breaks down at some point in the interval $275s < T_0 < 350s$, indicating the non-linear nature of this flow.

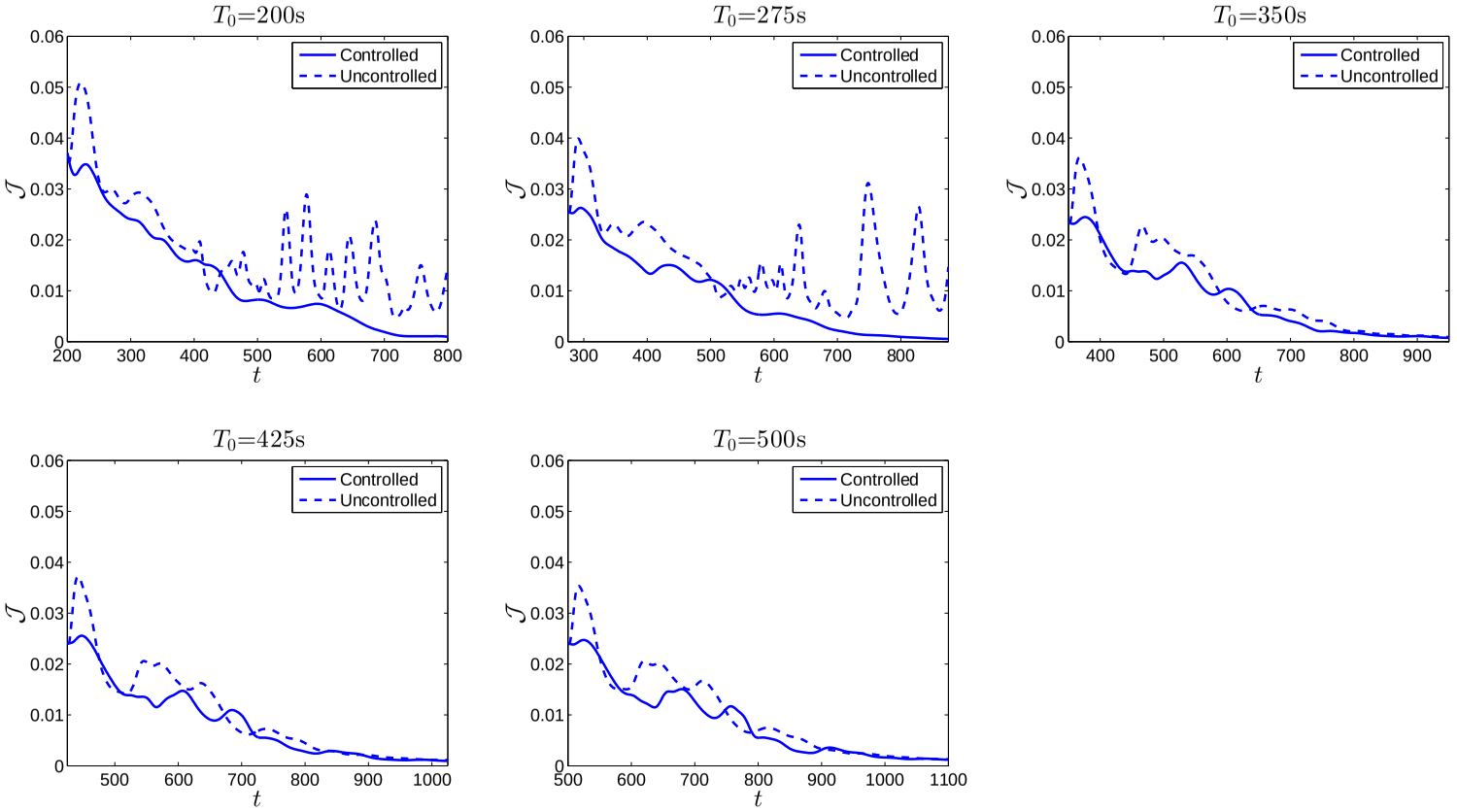

Figure 4.13: Evolution curves of the performance index $\mathcal{J}(t)$ starting from transient states $r_{\mathcal{U}}^{T_0}$ with fixed $\mathcal{U} = 1.4$ and different T_0. The solid curve, representing the controlled cases (LQR) is plotted against that of the uncontrolled cases (open-loop), dashed curve for comparison.

Chapter 5

Conclusions and future work

5.1 Conclusions

A reduced order modeling method suitable for linear quadratic regulator (LQR) boundary control of incompressible, density stratified, Boussinesq flow is developed using a proper orthogonal decomposition (POD)/Galerkin approach. Major contributions and results stemming from this thesis are summarized below

1. Updated and modified the DNS Fortran solver Diablo to make it suitable for developing control in numerical simulation

- The existing spectral method library of Diablo is updated from FFTW2 to FFTW3.
- Various simulations using different boundary conditions are conducted to gain physical insights into the flow. Parameters suitable for subsequent flow control are found.
- Time varying boundary conditions are implemented on the density field to allow for boundary control.
- The simulation provides snapshots for generating POD modes.

2. Obtained reduced order model (ROM) by the applying POD/Galerkin approach

   - The control system is excited using a step function because of its simple nature and broadband spectral content.
   - The first six POD modes are found to contain more than 99% of the flow energy (as shown in table 4.1).
   - The ROM is obtained by projecting the POD modes through the governing equations using Galerkin projection.
   - The ROM with the boundary controller is obtained by transferring the boundary condition into a forcing term in the governing equation through Duhamel's principle.
   - The fidelity of the ROM is validated by the strong agreement between the time evolution of the directly projected temporal coefficients and Galerkin integrated temporal coefficients (as shown in figures 4.4 and 4.5).

3. Developed and tested a LQR boundary controller

   - $U(t)$, the time derivative of the input $\mathcal{U}(t)$, is introduced into the ROM as an augmented state.
   - The cost functional which is minimized in the LQR controller design is defined to be the squared norm of the difference between the actual density field and the desired density field.
   - The penalty parameter R in the cost functional is found to play a critical role in the process of controller design.
   - Two metrics η_a and η_r are introduced to quantify the performance of the controller. η_a denotes the absolute effectiveness of the LQR controller and is a

metric of the controller's ability to drive the system to the final desired state. Conversely η_r denotes the relative effectiveness of the LQR controller and is a metric of the improvement of the LQR controller compared to a open-loop controller.

- For the control cases tested, the LQR controller is found to have $\eta_a = 96.7\%$ and $\eta_r = 22.2\%$ in the best (as shown in table 4.4)) case when the initial state corresponds to steady conditions, and $\eta_a = 99.3\%$ and $\eta_r = 47.9\%$ (as shown in table 4.6) when the initial state corresponds to transient conditions. η_r is more than doubled when comparing the transient and steady cases, indicating that the LQR controller is able to reject complex transient flow much better than the open-loop controller.

In summary, the reduced order LQR controller designed within this thesis provides effective control for moderate disturbance about a base case. This is remarkable given the non-linear nature of the governing equations of this flow and the fact that control is implemented using a single, spatially-symmetric boundary condition whose amplitude is modulated by the controller.

5.2 Future work

The research summarized above can be extended in numerous ways. In particular

- Different spatial distributions for the upper boundary condition could be considered. Also a different type of boundary conditions (i.e. Neumann or Robin vs. Dirichlet) could be incorporated to simulate different scenarios.

- More than one input could be added to the control. This would allow for more effective control when the flow is asymmetric in the horizontal direction.

- Although the reduced order model is designed for both the Naiver-Stokes and convection-diffusion equations, control is only applied to the density field r and the design of the LQR controller is only based on the convection-diffusion equation. Therefore extending the control to include the Navier-Stokes equations explicitly (rather than implicitly) could result in better control performance.

References

Antoulas, A. C. (2005). *Approximation of large-scale dynamical systems*, volume 6. SIAM.

Athans, M. and Falb, P. L. (1966). *Optimal control; an introduction to the theory and its applications*. Lincoln Laboratory publications. New York, McGraw-Hill.

Bewley, T. (Accessed Jan. 14, 2014). *Numerical Renaissance*. http://numerical-renaissance.com/NR.pdf.

Canuto, C., Hussaini, M. Y., Quarteroni, A., and Zang, T. A. (1988). *Spectral methods in fluid dynamics*. Springer-Verlag, New York.

Christofides, P. D. (2001). *Robust control of hyperbolic PDE systems*. Springer.

Courant, R., Friedrichs, K., and Lewy, H. (1967). On the partial difference equations of mathematical physics. *IBM journal*, 11(2):215–234.

Fletcher, C. (2012). *Computational techniques for fluid dynamics 2: Specific techniques for different flow categories*. Springer Science & Business Media.

Flynn, M. R. (2006). *Buoyancy and stratification in Boussinesq flow with applications to natural ventilation and intrusive gravity currents*. PhD thesis, University of California, San Diego.

Frigo, M. and Johnson, S. G. (2003). FFTW users manual for version 2.1.5. *http://www.fftw.org/fftw2.pdf*.

Frigo, M. and Johnson, S. G. (2012). FFTW users manual for version 3.3.3. *http://www.fftw.org/fftw3.pdf*.

Holmes, P., Lumley, J. L., and Berkooz, G. (1998). *Turbulence, coherent structures, dynamical systems and symmetry*. Cambridge University Press.

John, F. (1982). *Partial differential equations, volume 1 of Applied Mathematical Sciences*. Springer-Verlag, New York,.

Karhunen, K. (1947). *Über lineare Methoden in der Wahrscheinlichkeitsrechnung*, volume 37. Universitat Helsinki.

Krstic, M. and Smyshlyaev, A. (2008). *Boundary control of PDEs: A course on backstepping designs*, volume 16. SIAM.

Kundu, P. K., Cohen, I. M., and Dowling, D. R. (2011). *Fluid mechanics. [electronic resource].* London : Academic, 2011.

Lewis, F. L. and Syrmos, V. L. (1995). *Optimal control*. John Wiley & Sons.

Loeve, M. (1978). Probability theory, vol. ii. *Graduate texts in mathematics*, 46:0–387.

Luchtenburg, D., Noack, B., and Schlegel, M. (2009). An introduction to the pod galerkin method for fluid flows with analytical examples and matlab source codes. *Berlin Institute of Technology MB1, Muller-Breslau-Strabe*, 11.

Moin, P. (2010). *Fundamentals of engineering numerical analysis*. Cambridge University Press.

Noack, B. R., Papas, P., and Monkewitz, P. A. (2005). The need for a pressure-term representation in empirical galerkin models of incompressible shear flows. *Journal of Fluid Mechanics*, 523:339–365.

Oppenheim, A. V., Schafer, R. W., Buck, J. R., et al. (1989). *Discrete-time signal processing*, volume 2. Prentice hall Englewood Cliffs, NJ.

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

Sirovich, L. (1987). Turbulence and the dynamics of coherent structures. part i: Coherent structures. *Quarterly of applied mathematics*, 45(3):561–571.

Sutherland, B., Flynn, M., and Dohan, K. (2004). Internal wave excitation from a collapsing mixed region. *Deep Sea Research Part II: Topical Studies in Oceanography*, 51(25):2889–2904.

Taylor, J. R. (2008). *Numerical simulations of the stratified oceanic bottom boundary layer*. PhD thesis, University of California, San Diego.

Thomas, L. H. (1949). Elliptic problems in linear differential equations over a network: Watson scientific computing laboratory. *Columbia Univversity, New York*.

Appendix A

Upgrading from FFTW2 to FFTW3

In the original version of Diablo, discrete Fourier transforms (DFT) are calculated using FFTW version 2. It is advantageous to upgrade FFTW to its latest official release version 3.3.4 due to the introduction of numerous new features into the FFTW framework. In this chapter, we summarize the process for adapting the codes that are designed for the older FFTW2 to work with FFTW3. Due to the fact that the interface for FFTW3 is not backward-compatible with the interface for version 2 or earlier, the codes designed to use version 2 or earlier cannot be made to link with the library of FFTW3. However, the upgrading process should still be straightforward due to the fact that the data formats between different versions are identical. Detailed information regarding FFTW2 and FFTW3 can be found in [Frigo and Johnson, 2003] and [Frigo and Johnson, 2012] respectively.

Although FFTW is originally designed as a C subroutine library for computing DFT, the interface for legacy Fortran, the language in which Diablo is written is included in FFTW library by default. Greater care is needed while using this interface, however, since it is not type-checked. Besides that, the legacy Fortran interface differs from its C counterpart in the prefix of the subroutine. More specifically in FFTW3, '`dfftw_`' is used instead of '`fftw_`' for double precision calculations.

The main difference between FFTW2 and FFTW3 is in the division of work

between planning and execution. In FFTW2, plans are generated only based on the type and size of the transform, then they can be executed on arrays with any multiplicity and stride parameters. In FFTW3, by contrast, information about both the array and the transform is needed in the process of generating plans, and these plans can, in turn, be executed on those specific arrays. Therefore, more specifically, the difference between FFTW2 and FFTW3 is that the information about arrays that was formerly specified at execution time in FFTW2 is now specified at planning time in FFTW3. FFTW calls with the highest level of flexibility are used in Diablo and they will be used as examples to demonstrate the upgrading process in detail.

A.1 Forward transform (physical space to Fourier space)

For FFTW2

```
CALL RFFTWND_F77_CREATE_PLAN_(INTEGER PLAN, INTEGER RANK, CONST INTEGER N[],
                              FFTW_FORWARD, FFTW_MEASURE + FFTW_IN_PLACE)
CALL RFFTWND_F77_REAL_TO_COMPLEX_(INTEGER PLAN, INTEGER HOWMANY,
                                  REAL IN, INTEGER ISTRIDE,
                                  INTEGER IDIST,
                                  COMPLEX OUT, INTEGER OSTRIDE,
                                  INTEGER ODIST)
```

For FFTW3

```
CALL dfftw_plan_many_dft_r2c_(INTEGER PLAN, INTEGER RANK, CONST INTEGER N[],
                              INTEGER HOWMANY,
                              REAL IN, CONST INTEGER INEMBED[],
                              INTEGER ISTRIDE, INTEGER IDIST,
                              COMPLEX OUT, CONST INTEGER ONEMBED[],
```

```
                                    INTEGER OSTRIDE, INTEGER ODIST,
                                    FFTW_ESTIMATE)
CALL dfftw_execute_dft_r2c_(INTEGER PLAN, REAL IN, COMPLEX OUT)
!CALL dfftw_execute_(INTEGER PLAN)
```

In the code section, the first command is the planning command, and the second command is the execution command. Besides, for FFTW3, since `PLAN` contains all the information needed for computing the transform, usually the execution subroutine does not need to include any parameter except for `PLAN` just like the last line of code that is commented out. However, for legacy Fortran, because the input/output arrays are not passed as explicit arguments in `dfftw_execute`, the semantics of legacy Fortran allow the compiler to assume that the input/output arrays are not changed by `dfftw_execute`. As a consequence, certain compilers will optimize out `dfftw_execute` assuming it does nothing. That is the reason why `dfftw_execute_dft_r2c` is used as the execution call instead.

Arguments

- `PLAN` contains all the necessary information to compute the transform including both the information about the transform itself and the information about input and output arrays for FFTW3. Whereas for FFTW2, information about arrays are unknown to the `PLAN` variable.

- `RANK` is the dimensionality of the transform.

- `N[RANK]` gives the physical size of the transform dimensions. `N[1]`, `N[2]` and `N[n]` equals the size of the transform in first, second and `n`-th dimension respectively.

- `IN` and `OUT` are the starting positions of the input and output arrays respectively. Since the concept of the pointer does not exist in Fortran, `IN` and `OUT` are just

the elements inside the arrays that specify where the input and output start.

- {I,O}NEMBED[RANK] must be element-wise greater than or equal to N[RANK]. {I,O}NEMBED[RANK] enable input and output arrays to be column major sub-arrays with larger size on each dimensions. {I,O}NEMBED[RANK] and N[RANK] are set to be the same in our case.

- HOWMANY is the number of transforms to compute. Plans generated in this way are often faster then calling plans that are generated for a single transform multiple times. If HOWMANY$>$ 1, the input of the n-th transform is at location IN+n*IDIST and the output is at location OUT+n*ODIST.

- {I,O}STRIDE is the distance between adjacent elements inside input and output arrays, respectively. If the data points that we want to perform transformations on are adjacent to each other, {I,O}STRIDE$=$ 1.

- FFTW_IN_PLACE is a FFTW2 only planning option. For FFTW3, input and output arrays are already specified at the planning phase. In place transforms can be performed by setting the input and output arrays to possess the same memory.

- FFTW_MEASURE tells FFTW to find an optimized plan by actually computing several transforms on input and output arrays and comparing their execution time. With this option, input and output arrays will be overwritten during the planning phase.

- FFTW_ESTIMATE tells FFTW to use a simple heuristic to pick a plan (probably sub-optimal) quickly. With this flag, input and output arrays will not be overwritten during the planning phase.

- FFTW_FORWARD represents forward transform.

A.2 Backward transform (Fourier space to physical space)

For FFTW2

```
CALL RFFTWND_F77_CREATE_PLAN_(INTEGER PLAN, INTEGER RANK, CONST INTEGER N[],
                              FFTW_BACKWARD, FFTW_MEASURE + FFTW_IN_PLACE)
CALL RFFTWND_F77_COMPLEX_TO_REAL_(INTEGER PLAN, INTEGER HOWMANY,
                                  COMPLEX IN, INTEGER ISTRIDE,
                                  INTEGER IDIST,
                                  REAL OUT, INTEGER OSTRIDE,
                                  INTEGER ODIST)
```

For FFTW3

```
CALL dfftw_plan_many_dft_c2r_(INTEGER PLAN, INTEGER RANK, CONST INTEGER N[],
                              INTEGER HOWMANY,
                              COMPLEX IN, CONST INTEGER INEMBED[],
                              INTEGER ISTRIDE, INTEGER IDIST,
                              REAL OUT, CONST INTEGER ONEMBED[],
                              INTEGER OSTRIDE, INTEGER ODIST,
                              FFTW_ESTIMATE)
CALL dfftw_execute_dft_c2r_(INTEGER PLAN, COMPLEX IN, REAL OUT)
```

Arguments for the backward transformation are as described above.

Appendix B

Output data in binary format from Diablo

In Diablo, the resulting velocity and density fields are saved in double precision. In outputting the data in Matlab-readable format, there are two options being ASCII format and binary format. Although binary files cannot be viewed in a text editor, data saved in binary is done so with much greater efficiency (and no loss of precision) as compared to ASCII. A further advantage of using binary is that Matlab loads binary files output by Diablo approximately four times as fast as compared to the corresponding ASCII files. With binary format being the obvious winner, the method of saving the resulting data from Diablo in Matlab-readable binary format is outlined in this chapter. Here, the `OPEN` and `WRITE` commands in Fortran are used to achieve this goal.

```
OPEN(UNIT=number, FILE=filename, RECL=length,
     STATUS="...", FORM="...", ACCESS="...")
```

- `number` must be a positive integer. Low unit numbers such as 0 and 1 are reserved for special units like `INPUT_UNIT` and `OUTPUT_UNIT`. Usually 10 is the lowest unit number available; values increase from 10 if more than one file needs to be open at a time.

- `filename` is the name of the file to be opened. The extension ".bin" is used

here since we want this file to be in binary format.

- `STATUS="UNKNOWN"` indicates that the existence of the file is unknown. This is the default setting.
- `FORM="FORMATTED"` indicates the file will be saved in ASCII format.
- `FORM="UNFORMATTED"` indicates the file will be saved in binary format.
- `ACCESS="SEQUENTIAL"` indicates the file will be accessed sequentially meaning without skipping. Although binary files can be generated in this way, addition information will be added prior to and after each piece of data. These additional information prevents Matlab from reading the data effectively as a result of which these additional details are useless and memory wasting.
- `ACCESS="DIRECT"` indicates the file could be accessed at any specified location. If this option is used, the value of `length` needs to be specified as well. Here `length` represents the number of characters associated with each file entry. Since data from Diablo is stored in double precision, `length` is set to be eight.

Based on the above information, our `OPEN` command looks as follows:

```
OPEN(UNIT=10, FILE="filename.bin", RECL=8,
     STATUS="UNKNOWN", FORM="UNFORMATTED", ACCESS="DIRECT")
```

The corresponding `WRITE` command is given by

```
DO I=1,N
  WRITE(UNIT=10; REC=I) U(I)
END DO
```

Here `REC` is the record counter. `U` is the array to be outputted to `"filename.bin"`.

Appendix C

Restarting Diablo from a specific time instant

Before applying our controller, the flow is allowed to develop over the time interval $0 < t < T_0$. As such, and so as to test the impact of different control schemes, it is desirable if our simulation can be restarted from a specific time instant from another, already completed, simulation. The method for accomplishing this is outlined as follows.

In order to restart a Diablo simulation, the velocity field $\boldsymbol{u}$, density field r and pressure distribution p are needed. For all our simulations conducted thus far, the velocity field $\boldsymbol{u}$ and density field r are always saved in Matlab-readable binary format – see appendix B for details. If the pressure p is the only unknown in equations (2.2), it can be solved for by substituting in the numerical values for $\boldsymbol{u}$ and r. Luckily, this functionality is already buried inside an existing Diablo subroutine `POISSON_P_CHAN`. I have already written the wrapper on both the Diablo side and Matlab side. Therefore without elaborating upon the tedious coding details, the steps to be followed are:

- 1. On the Matlab side, change the path parameter in the file restart.m to the path where the reference simulation is located. Then change the t parameter to the time instant of interest.

- 2. Run restart.m. Then three files, namely U.txt, V.txt, D.txt which respec-

tively store the horizontal velocity, the vertical velocity and the density at the time instant of choice are generated.

- 3. Copy those three files to the directory /diablo/case/.
- 4. Under the same directory, change the `IC_TYPE` parameter to 3 in the file input.dat. Then change the parameter `N_TIME_STEPS` to the remaining time steps that you would like to run in the same file. Recall that the number of time steps can be converted to real time by multiplying by `DELTA_T`.
- 5. Run Diablo by going to the directory where diablo.f is located and type `./go` in the command window.

Appendix D

Postprocess files

The following table provides a brief description of the files used in the post-processing of Diablo output.

File Name	Description
diablo.m	Compute POD using the snapshot method and Galerkin system based on the velocity and density data generated by Diablo
diablo_q.m	Compute convection term q_ijk for NS-equation
diablo_l.m	Compute dissipation term l_ij for NS-equation
diablo_f.m	Compute body force term f_ij for NS-equation
diablo_cv.m	Compute convection term cv_ijk for C-D equation
diablo_df.m	Compute diffusion term df_ij for C-D equation
diablo_b.m	Compute boundary control term b_i
diablo_gs.m	function handle for ODE45 to compute the Galerkin projection
diablo_k.m	compute the gain for LQR controller
restart.m	generate velocity and density field data at specific time instant in order to restart Diablo

Table D.1: Computer Program Summary