Brigham Young University
BYU ScholarsArchive

All Theses and Dissertations

2018-07-01

Data Assimilation in the Boussinesq Approximation for Mantle Convection

Shane Alexander McQuarrie
Brigham Young University

Follow this and additional works at: https://scholarsarchive.byu.edu/etd

Part of the Mathematics Commons

BYU ScholarsArchive Citation

McQuarrie, Shane Alexander, "Data Assimilation in the Boussinesq Approximation for Mantle Convection" (2018). *All Theses and Dissertations*. 6951.
https://scholarsarchive.byu.edu/etd/6951

This Thesis is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in All Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.

www.manaraa.com

Data Assimilation in the Boussinesq Approximation for Mantle Convection

Shane Alexander McQuarrie

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Jared Whitehead, Chair
Emily Evans
Mark Allen

Department of Mathematics

Brigham Young University

Copyright © 2018 Shane Alexander McQuarrie

All Rights Reserved

المنارة للاستشارات

www.manaraa.com

ABSTRACT

Data Assimilation in the Boussinesq Approximation for Mantle Convection

Shane Alexander McQuarrie
Department of Mathematics, BYU
Master of Science

Many highly developed physical models poorly approximate actual physical systems due to natural random noise. For example, convection in the earth's mantle—a fundamental process for understanding the geochemical makeup of the earth's crust and the geologic history of the earth—exhibits chaotic behavior, so it is difficult to model accurately. In addition, it is impossible to directly measure temperature and fluid viscosity in the mantle, and any indirect measurements are not guaranteed to be highly accurate.

Over the last 50 years, mathematicians have developed a rigorous framework for reconciling noisy observations with reasonable physical models, a technique called *data assimilation*. We apply data assimilation to the problem of mantle convection with the infinite-Prandtl Boussinesq approximation to the Navier-Stokes equations as the model, providing rigorous conditions that guarantee synchronization between the observational system and the model. We validate these rigorous results through numerical simulations powered by a flexible new Python package, Dedalus. This methodology, including the simulation and post-processing code, may be generalized to many other systems.

The numerical simulations show that the rigorous synchronization conditions are not sharp; that is, synchronization may occur even when the conditions are not met. These simulations also cast some light on the true relationships between the system parameters that are required in order to achieve synchronization. To conclude, we conduct experiments for two closely related data assimilation problems to further demonstrate the limitations of the rigorous results and to test the flexibility of data assimilation for mantle-like systems.

Keywords: Data Assimilation, Boussinesq Equations, Mantle Convection, Computational Fluid Dynamics, Dedalus

المنارة للاستشارات
www.manaraa.com

Acknowledgments

First of all, I am grateful to Jared Whitehead for his mentoring and friendship, and for introducing me to such a compelling and relevant area of computational mathematics. Thanks to our collaborators Nathan Glatt-Holtz, Vincent Martinez, and Aseel Farhat, who made a great deal of this work possible. Thank you to the BYU ACME professors, particularly Tyler Jarvis, Emily Evans, Jared Whitehead, and Jeffrey Humpherys, for their herculean efforts to create a thorough and modern undergraduate curriculum in applied and computational mathematics, and for inviting me to contribute to the development of the program. Thanks to my previous math teachers David Wright, Annette Bartlett, Anne Crosland, and Jessica Purcell, for being patient and effective with an energetic and distracted young student. To my tireless music teachers, especially Patricia Henderson, David Blackinton, and David Fullmer, thank you for teaching me to show up, dig in, and have fun.

I am also grateful to my parents-in-law Glen and Alicia Hilton, who are always interested to hear about my work, plans, and ideas. Thanks to my own sweet parents R. Scott and Melissa McQuarrie, for the kindness with which they raised me and for the countless hours of babysitting they provided while this work was being finished.

Finally, thank you to my lovely wife Laura. I'll always belong to you.

المنارة للاستشارات

www.manaraa.com

Contents

Contents iv

List of Tables vi

List of Figures vii

1 Introduction 1

1.1 Convection 1

1.2 The Boussinesq Approximation 6

1.3 Data Assimilation 8

1.4 Dedalus 11

1.5 Outline 13

2 Infinite Prandtl Assimilation 14

2.1 Functional Setting 15

2.2 Regularity 17

2.3 A Priori Estimates 19

2.4 Main Theorem 20

3 Numerical Simulations 25

3.1 Dimension Reduction 25

3.2 Minimal Simulation Code 27

3.3 Results 31

4 Other Assimilation Schemes 38

4.1 Finite Prandtl Assimilation 38

4.2 Hybrid Prandtl Assimilation 42

المنارة للاستشارات

www.manaraa.com

5 Conclusion **46**

5.1 Summary 46

5.2 Extensions 46

A Notation **48**

A.1 Vector Products 49

A.2 Derivatives 50

A.3 Spaces and Norms 56

B Analytical Tools **61**

B.1 Vector Calculus 61

B.2 Inequalities 62

C Simulation Code **63**

Bibliography **88**

المنارة للاستشارات

www.manaraa.com

List of Tables

1.1 Typical Prandtl numbers for several fluids. 3

2.1 The function spaces used in the rigorous analysis of the data assimilation problem. 15

3.1 Minimal values of μ that result in synchronization within about 0.005 units of simulation time for various Ra with $N = 32$. 35

3.2 Minimal values of N that result in synchronization within about 0.005 units of simulation time for the Ra and μ in Table 3.1. 36

4.1 Minimal values of Pr that result in synchronization within about 0.01 units of simulation time for the Ra and μ in Table 3.1, with $N = 32$. 42

www.manaraa.com

List of Figures

1.1 Typical two-dimensional Rayleigh-Bénard convection in a turbulent state. . . 5

1.2 The solution to a simple BVP, calculated with Dedalus. 12

3.1 Typical "before" and "after" snapshots of a successful numerical experiment. 32

3.2 Synchronization in various norms for a typical numerical experiment with $\Pr = \infty$. 33

3.3 Convergence (or lack thereof) with $\mu = 1$ and various Ra. 34

3.4 Synchronization with $\Pr = \infty$, $N = 32$, and various μ for two different values of Ra. 34

3.5 Quadratic (but nearly linear) least-squares fit for the data in Table 3.1. . . . 35

3.6 Synchronization for two different values of Ra, with μ as listed in Tables 3.1 and 3.2 and for various values of N. 36

4.1 Synchronization in various norms for $\mathrm{Ra} \approx 5.22 \times 10^7$, $\mu = 18,000$, $N = 32$, and $\Pr = 100$. 41

4.2 Synchronization with $N = 32$ and various Pr for two different values of Ra, with μ as given in Table 4.1. 41

4.3 Hybrid assimilation with $\mathrm{Ra} \approx 5.22 \times 10^7$, $\mu = 18,000$, $N = 32$, and $\Pr = 100$. The temperature and velocity differences remain almost constant, with no hint of convergence. 44

4.4 Lack of synchronization in hybrid assimilation with $\mathrm{Ra} \approx 8.89 \times 10^6$, $N = 32$, and $\Pr = 100$, with various μ values. 45

المنارة للاستشارات
www.manaraa.com

Chapter 1. Introduction

A substantial portion of engineering, physics, chemistry, biology, mathematics, and other scientific fields is dedicated to the study of *fluid dyamics*, the science of liquids and gasses in motion. For example, [11] discusses wind turbine designs for converting air flow into clean energy; in [18], biologists study intracellular fluid flow to understand cellular reproduction; and [20] discusses tsunami modeling to better predict and prepare for future disasters. While some simple fluid phenomena are well understood, there are many settings in which the exact dynamical behavior of a fluid is extremely complex. Even something as fundamental and simple as the wind blowing is difficult to accurately model and analyze. Since fluid dynamics describe many such fundamental processes, it is important to continue studying these difficult problems in order to make progress on weather models, airplane technology, materials manufacturing, and many other areas of science and technology.

1.1 Convection

Most fluids expand as they heat up and contract as they cool down. This expansion and contraction changes the density of the fluid: as a fluid is heated, it becomes less dense, and vice versa. *Natural convection*, also called *free heat convection*, is a form of heat transfer in which the resulting changes in density cause fluid motion; in other words, the fluid is moved by internal processes, not by external forces. Convective systems often exhibit unpredictable behavior and are notoriously difficult to analyze with classical techniques.

On geologic time scales, the mantle's motion is fluid and exhibits convective behavior. Convective processes thus play a role in tectonic motion and volcanic activity. In this thesis, we analyze a classical model for the earth's mantle as a fluid. To begin, we briefly introduce some common fluid dynamics notation.

المنارة للاستشارات
www.manaraa.com

1.1.1 Variables. Let $\Omega \subset \mathbb{R}^3$ be a bounded domain and $\mathbb{T} = [0, \infty)$. In fluid dynamics the following variables are standard.

$\mathbf{u} : \Omega \times \mathbb{T} \to \mathbb{R}^3$ is the *velocity field*, which describes the macroscopic motion of an infinitesimal unit of fluid. The behavior of $\mathbf{u}$ is of principal interest in most fluid dynamics problems.

$T : \Omega \times \mathbb{T} \to \mathbb{R}$ is the *absolute temperature* of an infinitesimal unit of fluid.

$p : \Omega \times \mathbb{T} \to \mathbb{R}$ is the *pressure* on the fluid, measured in pascals (Pa). Though it is often the most computationally difficult quantity to work with, it is an important component in many applications. For our numerical simulations, we will simplify our main equations in a way that eliminates p.

$\rho : \Omega \times \mathbb{T} \to \mathbb{R}$ is the *density* of the fluid, measured in kg/m^3. We will examine the special case when ρ is constant, in which case the fluid is called *incompressible*.

$\nu : \Omega \times \mathbb{T} \to \mathbb{R}$ is the *kinematic viscosity*, the ratio of the dynamic (shear) viscosity to the density. This quantity has units m^2/s; at 20°C, water has $\nu \approx 1.0023 \times 10^{-6} m^2/s$. Roughly speaking, the lower ν, the more freely the fluid moves in relation to itself. For instance, at the same temperature, tar has a greater ν than dish soap.

$\kappa : \Omega \times \mathbb{T} \to \mathbb{R}$ is the *thermal diffusivity*, the ratio of thermal conductivity (how strongly the fluid conducts heat) to the density. This is the same κ that appears in the heat equation $T_t = \kappa \Delta T$ and also has units m^2/s.

α is the *coefficient of thermal expansion*, which indicates how the volume of a fluid changes with respect to the temperature at constant pressure.

g is the gravitational constant, $g = 9.8\ m/s^2$.

These variables are also used in the two-dimensional setting where Ω is replaced by a bounded domain $\Psi \subset \mathbb{R}^2$ (in that case, $\mathbf{u} : \Psi \times \mathbb{T} \to \mathbb{R}^2$). See, for example, [9] for more details on these common variables.

المنارة للاستشارات
www.manaraa.com

Fluid	Pr
Mercury	0.015
Oxygen	0.63
Air	0.7
Water	7
Glycerol	1000
Earth's Mantle	1×10^{25}

Table 1.1: Typical Prandtl numbers for several fluids [7]. The mantle is an extreme outlier, which justifies the approximation $\text{Pr} = \infty$ for modeling mantle convection.

1.1.2 Dimensionless Parameters. The following dimensionless numbers encapsulate some important properties of a fluid.

The *Rayleigh number* Ra is the ratio of the buoyancy-driven effects (the convective driving force) to the viscous and diffusive effects in a fluid. The precise definition depends on the domain in question (see (1.3)). A higher Ra means heat transfer happens primarily by convection; a lower Ra indicates that the heat transfer is dominated by conduction.

The *Prandtl number* Pr is the ratio of momentum diffusivity to thermal diffusivity,

$$\text{Pr} = \frac{\nu}{\kappa}. \tag{1.1}$$

See Table 1.1 for the Prandtl numbers of a few fluids.

These dimensionless numbers are the main parameters of interest for our experiments. See [3, 15, 24] for more details on these (and other) dimensionless fluid numbers.

1.1.3 Rayleigh-Bénard Convection. In 1916, Lord Rayleigh established a rigorous framework for analyzing fluid experiments carried out by Henri Bénard [21]. The setting is simple: place a fluid between parallel surfaces and maintain those surfaces at constant temperatures. Though this is not that different from boiling water on the stove, the resulting flow is remarkably difficult to analyze. See [1] for an overview of current challenges in Rayleigh-Bénard convection.

المنارة للاستشارات
www.manaraa.com

To describe Rayleigh-Bénard convection in three dimensions, we couple a spatial domain $\Omega = [0, L]^2 \times [0, h]$ having coordinates $[x_1, x_2, x_3]$ with the Dirichlet boundary conditions

$$T(x_1, x_2, 0, t) = T_{\text{lower}}, \qquad T(x_1, x_2, h, t) = T_{\text{upper}},$$

where $\delta T = T_{\text{lower}} - T_{\text{upper}} > 0$. In addition, we impose periodic boundary conditions in the non-vertical directions x_1 and x_2:

$$\mathbf{u}(x_1 + L, x_2, x_3, t) = \mathbf{u}(x_1, x_2, x_3, t), \qquad \mathbf{u}(x_1, x_2 + L, x_3, t) = \mathbf{u}(x_1, x_2, x_3, t),$$
$$T(x_1 + L, x_2, x_3, t) = T(x_1, x_2, x_3, t), \qquad T(x_1, x_2 + L, x_3, t) = T(x_1, x_2, x_3, t).$$

Periodicity is physically relevant for mantle convection, since the mantle is bounded between two concentric spheres. The directions x_1 and x_2 correspond roughly to longitude and latitude, while x_3 measures depth. Thus h represents the distance from the core to the crust, and L serves as a rough measure of the mantle's circumference. See Figure 1.1.

For the velocity field, we adopt the *no-slip* conditions for rigid boundaries. That is, we assume that the fluid motion vanishes at the top and bottom of Ω:

$$\mathbf{u}(x_1, x_2, 0, t) = \mathbf{u}(x_1, x_2, h, t) = \mathbf{0}.$$

These conditions are helpful mathematically and experimentally validated for most fluid boundaries [24, 9]. On the other hand, the more intuitive *free-slip* conditions state that the fluid should not pass through the bounding plates, but allow for movement parallel to the plates:

$$\mathbf{u}_{x_3}(x_1, x_2, 0, t) = \mathbf{u}_{x_3}(x_1, x_2, h, t) = \mathbf{0}. \tag{1.2}$$

While we use the no-slip conditions in our rigorous analysis, we note that our numerical method can be easily adapted to either set of conditions.

المنارة للاستشارات
www.manaraa.com

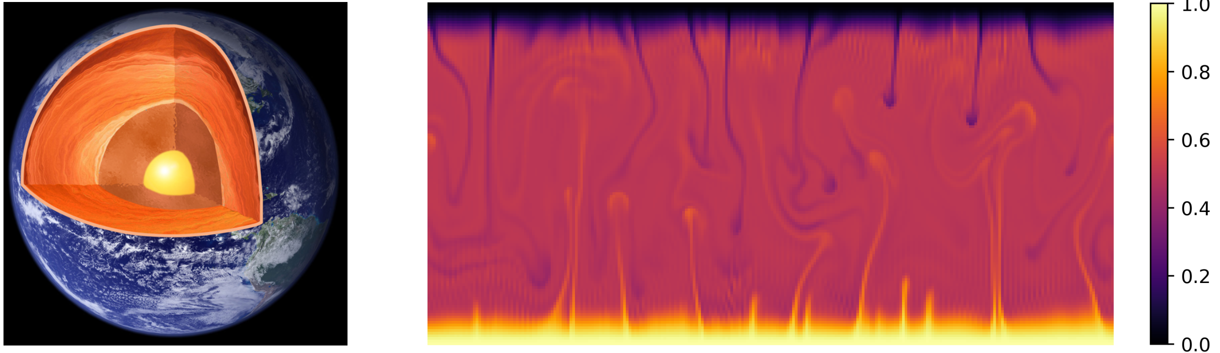

Figure 1.1: On the left, an illustration of the earth's layers. The mantle is the large orange section just beneath the outer crust. On the right, typical two-dimensional Rayleigh-Bénard convection in a turbulent state. Here $T_{\text{lower}} = 1$ and $T_{\text{upper}} = 0$. Note the periodicity in the horizontal direction and the small vertical boundary layers. Earth image taken from `http://newscenter.lbl.gov/2011/07/17/kamland-geoneutrinos/`.

The model can be further simplified by ignoring the horizontal direction x_2. While unnecessary for our rigorous results, reducing the dimensions of the domain greatly reduces the cost of numerical simulations (see Section 3.1). Let $\Psi = [0, L] \times [0, h]$ with coordinates $[x_1, x_3]$. In this setting we have the boundary conditions

$$T(x_1, 0, t) = T_{\text{lower}}, \qquad T(x_1, h, t) = T_{\text{upper}},$$
$$\mathbf{u}|_{x_3=0} = \mathbf{u}|_{x_3=h} = \mathbf{0},$$
$$\mathbf{u}, T \text{ are periodic in } x_1.$$

In either setting, we have a domain-specific definition for the Rayleigh number,

$$\text{Ra} = \frac{g\alpha}{\nu\kappa}(\delta T)h^3. \tag{1.3}$$

المنارة للاستشارات
www.manaraa.com

1.2 The Boussinesq Approximation

We now introduce a system of partial differential equations, subject to the boundary conditions discussed in the previous section, that describes Rayleigh-Bénard convection. The most general partial differential equations for viscous fluid flow are the Navier-Stokes equations; what follows is an augmentation of Navier-Stokes with a temperature equation, then simplified based on a few key assumptions.

1.2.1 Incompressibility. The law of conservation of mass leads to the *continuity equation* for fluids,

$$\partial_t \rho + \nabla \cdot (\rho \mathbf{u}) = 0. \tag{1.4}$$

An *incompressible fluid* is one that has constant density, i.e. $\rho \equiv \rho_0 > 0$. In this case, $\partial_t \rho = 0$ and the continuity equation becomes

$$\nabla \cdot \mathbf{u} = 0. \tag{1.5}$$

In other words, the fluid velocity field is *divergence free*. The assumption of constant density is reasonable for certain fluids; for example, water at 20°C has fairly constant density in relation to perturbations of temperature and pressure. On the other hand, changes in density throughout the mantle have a significant impact on mantle movement and the resulting tectonic behavior [25].

1.2.2 Gravitational Effects. Changes in density drive fluid flow in convection, so the assumption of incompressibility must be accompanied by an additional term that accounts for density evolution. The simplest way to account for density is to assume that it only affects the fluid when coupled with gravity. In other words, gravity is the only force strong enough to make density changes noticeably important.

المنارة للاستشارات
www.manaraa.com

Conservation of momentum, ignoring the effects of density except when coupled with gravity, gives rise to the *momentum equations*

$$\partial_t \mathbf{u} + (\mathbf{u} \cdot \nabla)\mathbf{u} - \nu \Delta \mathbf{u} = -\frac{1}{\rho_0} \nabla p + \alpha g \mathbf{e}_3 T. \tag{1.6}$$

Note that this is a vector equation, since $\mathbf{u}$ is a vector field.[1] We also have the following *temperature equation*, which arises from assuming a constant heat capacity per unit volume:

$$\partial_t T + \mathbf{u} \cdot \nabla T - \kappa \Delta T = 0. \tag{1.7}$$

See [5, 15, 24] for more rigorous derivations of the system up to this point.

1.2.3 Nondimensionalization. The system (1.6)–(1.7) can be non-dimensionalized to include the dimensionless parameters Ra and Pr. This is a common final step to make the model as simple as possible.

Set $T' = (T - T_{\text{high}})/\delta T$, shift $\Omega = [0, L]^2 \times [0, h]$ to the box $\Omega' = [0, L']^2 \times [0, 1]$, and scale time by $t' = \kappa t / h^2$. Then the new variables $\mathbf{u}'$ and T' satisfy

$$\frac{1}{\text{Pr}} [\partial_{t'} \mathbf{u}' + (\mathbf{u}' \cdot \nabla')\mathbf{u}'] - \Delta' \mathbf{u}' = -\nabla' p' + \text{Ra} \mathbf{e}_3 T', \tag{1.8}$$

$$\nabla' \cdot \mathbf{u}' = 0, \tag{1.9}$$

$$\partial_{t'} T' + \mathbf{u}' \cdot \nabla' T' - \Delta' T' = 0, \tag{1.10}$$

with boundary conditions

$$\mathbf{u}'|_{x_3'=0} = \mathbf{u}|_{x_3'=1} = \mathbf{0}, \tag{1.11}$$

$$T'|_{x_3'=0} = 1, \; T'|_{x_3'=1} = 0, \tag{1.12}$$

$$\mathbf{u}', T' \text{ are periodic in } x_1' \text{ and } x_2', \tag{1.13}$$

[1] See Appendix A for mathematical notation.

www.manaraa.com

and initial conditions $T'(\mathbf{x}',0) = T'_0(\mathbf{x}')$ and $\mathbf{u}'(\mathbf{x}',0) = \mathbf{u}'_0(\mathbf{x}')$. These are the non-dimensional *Boussinesq equations*. For convenience, we drop the $'$ notation from here on out.

In Chapter 2, we make one last simplification by formally setting $\mathrm{Pr} = \infty$, resulting in the system

$$-\Delta \mathbf{u} = -\nabla p + \mathrm{Ra}\mathbf{e}_3 T, \tag{1.14}$$

$$\nabla \cdot \mathbf{u} = 0, \tag{1.15}$$

$$\partial_t T + (\mathbf{u} \cdot \nabla) T - \Delta T = 0, \tag{1.16}$$

with the same auxiliary conditions

$$\mathbf{u}|_{x_3=0} = \mathbf{u}|_{x_3=1} = \mathbf{0},$$

$$T|_{x_3=0} = 1, \qquad T|_{x_3=1} = 0,$$

$$\mathbf{u}, T \text{ are periodic in } x_1 \text{ and } x_2,$$

$$T(\mathbf{x},0) = T_0(\mathbf{x}),$$

where now the initial velocity $\mathbf{u}_0$ is determined by T_0 and (1.14). In Chapter 4, we return to the formulation (1.8)–(1.10) where Pr is large, but finite.

Although there are several additional physical effects relevant to mantle convection that are omitted from this Rayleigh-Bénard model (e.g., compressibility and/or temperature-dependent viscosity), it is an appropriate "zeroth order" representation of mantle convection, and is the usual starting point and test base for mantle convection simulations [4, 22].

1.3 Data Assimilation

The critical question about (1.8)–(1.10) and (1.14)–(1.16), as with all mathematical models, is this: how well does the system reflect what happens in real life? The answer for these models (and most models) is "very well, in some aspects, most of the time." *Data assimila-*

المنارة للاستشارات
www.manaraa.com

tion is one way to overcome the shortcomings of an imperfect model by incorporating real world observations into the system. See [19] for a good review of data assimilation in general from a mathematical perspective.

Consider the model (1.14)–(1.16) with state variables $\tilde{\mathbf{u}}$, $\tilde{T}$, and $\tilde{p}$, but with a slight modification to the temperature equation:

$$-\Delta\tilde{\mathbf{u}} = -\nabla\tilde{p} + \mathrm{Ra}\mathbf{e}_3\tilde{T}, \tag{1.17}$$

$$\nabla\cdot\tilde{\mathbf{u}} = 0, \tag{1.18}$$

$$\partial_t\tilde{T} + (\tilde{\mathbf{u}}\cdot\nabla)\tilde{T} = \Delta\tilde{T} - \mu P_N(\tilde{T} - T). \tag{1.19}$$

We call this the *nudging* or *assimilating* system. The final term $\mu P_N(\tilde{T} - T)$ of the temperature equation is a Fourier projection of the difference between the model temperature $\tilde{T}$ and some real world temperature data given by T. That is, if $\{\phi_n\}_{n=1}^{\infty}$ is a Fourier basis for $H^2(\Omega)$ and $f \in H^2(\Omega)$,

$$f(\mathbf{x},t) = \sum_{n=1}^{\infty} f_n(t)\phi_n(\mathbf{x}) \qquad \Longrightarrow \qquad [P_N(f)](\mathbf{x},t) = \sum_{n=1}^{N} f_n(t)\phi_n(\mathbf{x}).$$

The integer N is the number of modes included in the truncation. The constant μ is a *relaxation parameter* that determines the strength of the coupling between $\tilde{T}$ and the "true" temperature T. The goal of Chapter 2 is to analyze the strength of the coupling (in particular, requirements on μ and N) needed to synchronize the assimilating variables $\tilde{\mathbf{u}}$ and $\tilde{T}$ with the data variables $\mathbf{u}$ and T.

Ideally, T is constructed from geological measurements. Unfortunately, obtaining real-world temperature and velocity data from the mantle is impractical for a few reasons.

(i) The mantle is buried beneath 3–20 km of crust and is mostly composed of materials that stay around 500–900°C at all times [25].

(ii) The mantle evolves on geologic time scales; any measurements taken within the same

المنارة للاستشارات
www.manaraa.com

decade correspond to virtually the same time t in the model.

(iii) The surface of the earth is only adjacent to the top layer of the mantle, so even if direct sampling were possible, we could only obtain measurements close to the upper boundary layer (near $x_3 = 1$ in the model).

Sparse temperature measurements are sometimes possible by observing volcanic activity [25], but observing the velocity field of the mantle is nigh impossible.

In the absence of data, we set up additional equations for the evolution of $\mathbf{u}$ and T. Thus the full data assimilation problem for $\mathrm{Pr} = \infty$ is

$$\begin{aligned} -\Delta \mathbf{u} &= -\nabla p + \mathrm{Ra}\mathbf{e}_3 T, \\ \nabla \cdot \mathbf{u} &= 0, \\ \partial_t T + (\mathbf{u} \cdot \nabla) T &= \Delta T, \end{aligned} \qquad \begin{aligned} &\mathbf{u}|_{x_3=0} = \mathbf{u}|_{x_3=1} = \mathbf{0}, \\ &T|_{x_3=0} = 1,\ T|_{x_3=1} = 0, \\ &\mathbf{u}, T \text{ are periodic in } x_1 \text{ and } x_2, \\ &T(\mathbf{x}, 0) = T_0(\mathbf{x}), \end{aligned}$$

$$\begin{aligned} -\Delta \tilde{\mathbf{u}} &= -\nabla \tilde{p} + \mathrm{Ra}\mathbf{e}_3 \tilde{T}, \\ \nabla \cdot \tilde{\mathbf{u}} &= 0, \\ \partial_t \tilde{T} + (\tilde{\mathbf{u}} \cdot \nabla) \tilde{T} &= \Delta \tilde{T} - \mu P_N(\tilde{T} - T), \end{aligned} \qquad \begin{aligned} &\tilde{\mathbf{u}}|_{x_3=0} = \tilde{\mathbf{u}}|_{x_3=1} = \mathbf{0}, \\ &\tilde{T}|_{x_3=0} = 1,\ \tilde{T}|_{x_3=1} = 0, \\ &\tilde{\mathbf{u}}, \tilde{T} \text{ are periodic in } x_1 \text{ and } x_2, \\ &\tilde{T}(\mathbf{x}, 0) = \tilde{T}_0(\mathbf{x}). \end{aligned}$$

The goal is to show that

$$\lim_{t \to \infty} \|\tilde{T}(\mathbf{x}, t) - T(\mathbf{x}, t)\| = \lim_{t \to \infty} \|\tilde{\mathbf{u}}(\mathbf{x}, t) - \mathbf{u}(\mathbf{x}, t)\| = 0 \quad \forall \mathbf{x} \in \Omega$$

with an appropriate norm, even when the initial temperature states $\tilde{T}_0$ and T_0 differ. This will require some conditions on Ra, μ, and N.

Previous studies in data assimilation for Rayleigh-Bénard convection, including [2, 13], assimilate the velocity field into the nudging equations as well as the temperature. The novelty of the method given above is that only the temperature T, not the velocity $\mathbf{u}$, is

المنارة للاستشارات
www.manaraa.com

present in the nudging equations. As we will see in Chapter 2, this is possible because $\mathrm{Pr} = \infty$.

1.4 Dedalus

Verifying our rigorous results with direct numerical simulation means running numerous fluid simulations. Numerical solvers for partial differential equations are usually very complex, requiring careful adaptation to each specific situation. *Dedalus* [6] is a new Python package that is being developed[2] specifically to overcome this problem: it supports symbolic entry for equations and boundary conditions, making it very easy to tweak and adapt to different problems. Dedalus uses pseudospectral methods to solve differential equations, meaning that the only real requirement is that the domain be "spectrally representable." Since our spatial domain Ω has a simple box geometry and some periodicity, our data assimilation problem is a perfect setting for Dedalus.

As a demonstration,[3] consider the following Poisson equation on the two-dimensional domain $[0, 2\pi] \times [0, 1]$, supplemented with boundary and periodicity conditions:

$$
\begin{gathered}
-\Delta u = 10 \sin(x/2)^2 (y - y^2), \qquad (1.20) \\
u(x_1, 0) = \sin(8x_1), \qquad u_{x_2}(x_1, 1) = \cos(x_1), \\
u(x_1 + 2\pi, x_2) = u(x_1, x_2).
\end{gathered}
$$

The setup and solver code is remarkably simple. The result is displayed in Figure 1.2.

[2]See `http://dedalus-project.org`.

[3]This example is from the documentation at `https://bitbucket.org/dedalus-project/dedalus/src`.

المنارة للاستشارات
www.manaraa.com

```python
import numpy as np
from matplotlib import pyplot as plt
from dedalus import public as de

# Set up the problem domain using a Fourier basis for the periodic direction.
x_basis = de.Fourier('x', 256, interval=(0, 2*np.pi))   # Periodic
y_basis = de.Chebyshev('y', 128, interval=(0, 1))       # Non-periodic
domain = de.Domain([x_basis, y_basis], grid_dtype=np.float64)
problem = de.LBVP(domain, variables=['u','uy'])

# Define the problem with symbolic entry.
problem.add_equation("uy - dy(u) = 0")                  # uy = du/dy.
problem.add_equation("dx(dx(u)) + dy(uy) = -10 * sin(x/2)**2 * (y - y**2)")

# Add boundary conditions.
problem.add_bc("left(u) = left(sin(8*x))")              # Dirichlet condition.
problem.add_bc("right(uy) = right(cos(x))")             # Neumann condition.

# Solve the problem.
solver = problem.build_solver()
solver.solve()

# Extract and visualize the solution.
u = solver.state['u']['g']                              # 'g' gets the real
fig, ax = plt.subplots(1,1)                             #  solution, not the
im = ax.pcolormesh(u.T, cmap="inferno")                 #  basis coefficients.
ax.axis("off")
fig.colorbar(im, ax=ax)
plt.show()
```

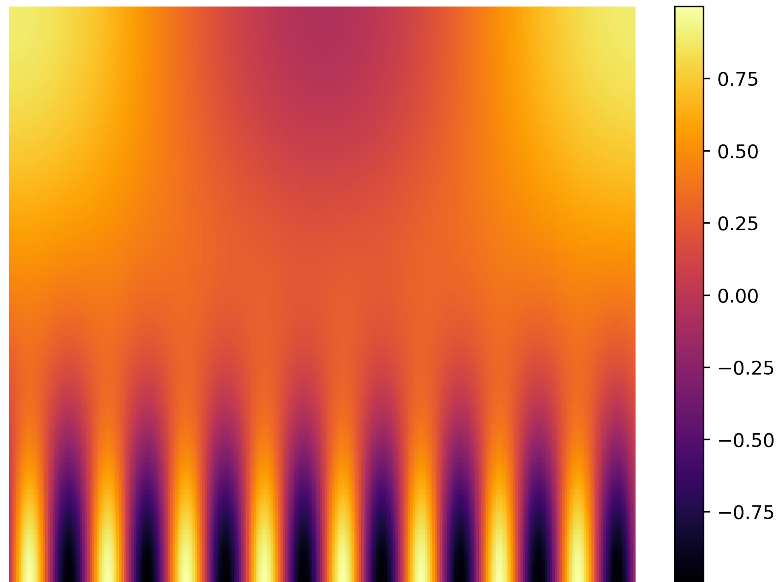

Figure 1.2: The solution of (1.20), calculated with Dedalus. Note how the solution is horizontally periodic and satisfies the vertical boundary conditions.

Note the following details about the example simulation code given above.

المنارة للاستشارات
www.manaraa.com

- The periodic direction (x) is represented with a Fourier basis, and the non-periodic direction (y) is represented with a Chebyshev basis. Dedalus only symbolically handles first order terms in non-periodic domains, so we explicitly define u_y with the line `problem.add_equation("uy - dy(u) = 0")`.

- When equations are entered symbolically, all linear terms must be on the left side of the equation. This is because Dedalus uses implicit-explicit timestepping; linear terms are evolved implicitly and nonlinear terms are treated explicitly.

- Since Dedalus uses pseudospectral techniques, state variables are represented in coefficient space. If `u = solver.state['u']` is one of the state variables, use `u['c']` to view the coefficient array and `u['g']` to view the real-valued array. This particular point is important in the implementation of P_N (see Section 3.2.1).

This extremely flexible, high-level system can be adapted to many problems. Dedalus implements a variety of time stepping schemes for initial value problems. We use an RK-443 method (`de.timesteppers.RK443`), but there are other options.

See `http://dedalus-project.readthedocs.io/en/latest/getting_started.html` for the official Dedalus tutorial and further examples.

1.5 Outline

We begin in Chapter 2 by examining the data assimilation problem presented in Section 1.3 from a mathematical standpoint. Theorem 2.6 presents, with a careful proof, the main result that the data and assimilating equations synchronize under suitable conditions. Chapter 3 validates Theorem 2.6 by presenting the results of corresponding numerical experiments. Finally, in Chapter 4, we explore extensions of the previous data assimilation model where $\mathrm{Pr} < \infty$, state the analogs to Theorem 2.6, and show corresponding numerical results.

المنارة للاستشارات
www.manaraa.com

In this chapter we examine the data assimilation system introduced in Section 1.3, the Boussinesq approximation where $\mathrm{Pr} = \infty$ and the accompanying nudging equations:

$$\begin{aligned} -\Delta \mathbf{u} &= -\nabla p + \mathrm{Ra}\mathbf{e}_3 T, \\ \nabla \cdot \mathbf{u} &= 0, \\ \partial_t T + \mathbf{u} \cdot \nabla T &= \Delta T, \end{aligned} \qquad \begin{aligned} \mathbf{u}|_{x_3=0} = \mathbf{u}|_{x_3=1} &= \mathbf{0}, \\ T|_{x_3=0} = 1,\ T|_{x_3=1} &= 0, \\ \mathbf{u}, T \text{ are periodic in } &x_1 \text{ and } x_2, \\ T(\mathbf{x}, 0) &= T_0(\mathbf{x}), \end{aligned} \tag{2.1}$$

$$\begin{aligned} -\Delta \tilde{\mathbf{u}} &= -\nabla \tilde{p} + \mathrm{Ra}\mathbf{e}_3 \tilde{T}, \\ \nabla \cdot \tilde{\mathbf{u}} &= 0, \\ \partial_t \tilde{T} + \tilde{\mathbf{u}} \cdot \nabla \tilde{T} &= \Delta \tilde{T} - \mu P_N(\tilde{T} - T), \end{aligned} \qquad \begin{aligned} \tilde{\mathbf{u}}|_{x_3=0} = \tilde{\mathbf{u}}|_{x_3=1} &= \mathbf{0}, \\ \tilde{T}|_{x_3=0} = 1,\ \tilde{T}|_{x_3=1} &= 0, \\ \tilde{\mathbf{u}}, \tilde{T} \text{ are periodic in } &x_1 \text{ and } x_2, \\ \tilde{T}(\mathbf{x}, 0) &= \tilde{T}_0(\mathbf{x}). \end{aligned} \tag{2.2}$$

Recall that (2.1) is called the *truth* or *data* equations, and (2.2) is called the *nudging* or *assimilating* equations. Defining $\mathbf{w} = \tilde{\mathbf{u}} - \mathbf{u}$, $S = \tilde{T} - T$, and $q = \tilde{p} - p$, the goal of the following analysis is to show that $S, q \to 0$ and $\mathbf{w} \to \mathbf{0}$ as $t \to \infty$ in the appropriate spaces, given specific conditions on the relaxation parameter μ and the number of projected modes N relative to Ra. First, we establish the appropriate functional setting. Then, after reviewing some well-known results pertaining to the regularity and energy of (2.1), we state and prove a theorem that specifies the strength of the coupling required to achieve synchronization between (2.1) and (2.2).

المنارة للاستشارات

www.manaraa.com

2.1 Functional Setting

2.1.1 Function Spaces. Let $\Omega = [0, L]^2 \times [0, 1]$. Following the notation of [23], we consider the function spaces

$$L^2_{per}(\Omega) := \{v \in L^2(\Omega) : v(x_1 + L, x_2, x_3, t) = v(x_1, x_2, x_3, t) \text{ for a.e. } x_1 \in [0, L], \quad (2.3)$$
$$v(x_1, x_2 + L, x_3, t) = v(x_1, x_2, x_3, t) \text{ for a.e. } x_2 \in [0, L]\},$$
$$H := \{\mathbf{v} \in L^2_{per}(\Omega)^3 : \nabla \cdot \mathbf{v} = 0, \mathbf{v}|_{x_3=0} = \mathbf{v}|_{x_3=1} = 0\}, \quad (2.4)$$
$$V := H \cap H^1(\Omega)^3, \quad (2.5)$$
$$\mathcal{V} := \{\mathbf{v} \in C^\infty_c(\Omega) : \nabla \cdot \mathbf{v} = 0,\ v(x_1 + L, x_2, x_3, t) = v(x_1, x_2, x_3, t), \quad (2.6)$$
$$v(x_1, x_2 + L, x_3, t) = v(x_1, x_2, x_3, t)\}.$$

Table 2.1 explains these function spaces more explicitly. See Appendix A.3 for descriptions of the common spaces $L^2(\Omega)$, $H^1(\Omega)$, $H^1_0(\Omega)$, and $C^\infty_c(\Omega)$.

Space	Description
$L^2_{per}(\Omega)$	The square-integrable functions on Ω that are L-periodic in x_1 and x_2, except perhaps on a set of measure zero.
H	Divergence free vector fields where each component is a member of $L^2_{per}(\Omega)$ and vanishes on the vertical boundaries.
V	Vector fields in H where each component has first-order, square-integrable weak derivatives.
$\mathcal{V}$	Smooth functions with compact support over Ω that are divergence free and L-periodic in x_1 and x_2.

Table 2.1: Descriptions of the spaces (2.3)–(2.6).

2.1.2 Projection Operators. Let $\{(\lambda_n, \phi_n)\}_{n=1}^\infty$ denote the eigenpairs of $-\Delta$, meaning $-\Delta\phi_n = \lambda_n\phi_n$. Then each $f \in H^2(\Omega)$ can be expressed in terms of the eigenfunctions as

$$f(\mathbf{x}, t) = \sum_{n=1}^{\infty} f_n(t)\phi_n(\mathbf{x}), \qquad f_n(t) = \langle f(t), \phi_n\rangle = \int_\Omega f(\mathbf{x}, t)\phi_n(\mathbf{x})\, d\mathbf{x}.$$

www.manaraa.com

In addition, let each ϕ_n be scaled so that

$$\int_{\Omega} \phi_i(\mathbf{x})\phi_j(\mathbf{x})\, d\mathbf{x} = \delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j. \end{cases}$$

That is, we assume that the set $\{\phi_n\}_{n=1}^{\infty}$ is orthonormal.

For each $N \geq 0$, define the projections P_N, Q_N by

$$P_N(f) = \sum_{n=1}^{N} f_n \phi_n, \qquad Q_N(f) = [I - P_N](f) = \sum_{n=N+1}^{\infty} f_n \phi_n,$$

where I is the identity operator. In other words, P_N is a truncation of the eigenfunction expansion, and Q_N is its orthogonal complement. Using the orthogonality of the eigenbasis, we have

$$\int_{\Omega} P_N(f) Q_N(f)\, d\mathbf{x} = \int_{\Omega} \left(\sum_{i=1}^{N} f_i \phi_i \right) \left(\sum_{j=N+1}^{\infty} f_j \phi_j \right) d\mathbf{x} = \sum_{\substack{i \leq N \\ j > N}} f_i f_j \int_{\Omega} \phi_i \phi_j\, d\mathbf{x} = 0, \tag{2.7}$$

and

$$\begin{aligned} \|P_N(f)\|_{L^2(\Omega)}^2 + \|Q_N(f)\|_{L^2(\Omega)}^2 &= \int_{\Omega} \left(\sum_{i=1}^{N} f_i \phi_i \right)^2 + \left(\sum_{j=N+1}^{\infty} f_j \phi_j \right)^2 d\mathbf{x} \\ &= \int_{\Omega} \sum_{i=1}^{N} f_i^2 \phi_i^2 + \sum_{j=N+1}^{\infty} f_j^2 \phi_j^2\, d\mathbf{x} \\ &= \int_{\Omega} \left(\sum_{k=1}^{\infty} f_i \phi_i \right)^2 d\mathbf{x} \\ &= \|f\|_{L^2(\Omega)}^2. \end{aligned} \tag{2.8}$$

المنارة للاستشارات
www.manaraa.com

2.2 Regularity

For the analysis of the data assimilation system (2.1)–(2.2), we wish to treat each of the variables $\mathbf{u}$, T, $\tilde{\mathbf{u}}$, and $\tilde{T}$ as functions with a degree of smoothness. We appeal to [26] to show the eventual regularity of suitable weak solutions and existence of a global attractor. First, we define what it means to be a weak solution to the original Boussinesq system (1.8)–(1.13) with $\mathrm{Pr} < \infty$.

Definition 2.1. A pair $(\mathbf{u}, T)$ is a *weak solution* to (1.8)–(1.10) with boundary conditions (1.11)–(1.13) and initial conditions $(\mathbf{u}_0, T_0)$ on the time interval $[0, T^*]$ if

$$\begin{aligned}
&\mathbf{u} \in L^\infty(0, T^*; H) \cap L^2(0, T^*; V) \cap C_w([0, T^*]; H),\\
&\mathbf{u}' \in L^{4/3}(0, T^*; V'),\\
&T \in L^\infty(0, T^*; L^2_{per}(\Omega)) \cap L^2(0, T^*; H^1_0(\Omega)) \cap C_w([0, T^*]; L^2_{per}(\Omega)),\\
&T' \in L^{4/3}(0, T^*; H^{-1}(\Omega))),\\
&\mathbf{u}(0) = \mathbf{u}_0, \qquad T(0) = T_0,
\end{aligned}$$

collectively satisfy

$$\frac{1}{\mathrm{Pr}}\left(\frac{d}{dt}\langle \mathbf{u}, \mathbf{v}\rangle + \langle (\mathbf{u}\cdot\nabla)\mathbf{u}, \mathbf{v}\rangle\right) + \langle \nabla\mathbf{u}, \nabla\mathbf{v}\rangle = \mathrm{Ra}\langle T, v_3\rangle \qquad \forall \mathbf{v} \in V, \tag{2.9}$$

$$\frac{d}{dt}\langle T, \eta\rangle + \langle \mathbf{u}\cdot\nabla T, \eta\rangle + \langle \nabla T, \nabla\eta\rangle = \langle u_3, \eta\rangle \qquad \forall \eta \in H^1_0(\Omega), \tag{2.10}$$

as well as the energy inequality and maximum principle:

$$\frac{1}{\mathrm{Pr}}\|\mathbf{u}(t)\|^2_{L^2(\Omega)} + 2\int_0^t \|\nabla\mathbf{u}(s)\|^2_{L^2(\Omega)} ds \le \frac{1}{\mathrm{Pr}}\|\mathbf{u}_0\|^2_{L^2(\Omega)} + 2\mathrm{Ra}\int_0^t \langle T(s), u_3(s)\rangle\, ds, \tag{2.11}$$

$$\|(T-1)^+(t)\|^2_{L^2(\Omega)} + 2\int_0^t \|\nabla(T-1)^+(s)\|^2_{L^2(\Omega)}\, ds \le \|(T_0-1)^+\|^2_{L^2(\Omega)}, \tag{2.12}$$

$$\|T^-(t)\|^2_{L^2(\Omega)} + 2\int_0^t \|\nabla T^-(s)\|^2_{L^2(\Omega)}\, ds \le \|(T_0)^-\|^2_{L^2(\Omega)}. \tag{2.13}$$

المنارة للاستشارات
www.manaraa.com

We then have the following theorem from [26].

Proposition 2.2. *Let* $(\mathbf{u}_0, T_0) \in H \times L^2_{per}(\Omega)$*. There exists a suitable weak solution* $(\mathbf{u}, T)$ *of* (1.8)–(1.13) *corresponding to* $(\mathbf{u}_0, T_0)$*. Moreover, there exists an absolute constant* $K_0 > 0$ *such that if* $PrRa^{-1} \geq K_0$*, then there exists a time* $T^* > 0$ *for which all suitable weak solutions corresponding to* $(\mathbf{u}_0, T_0)$ *are regular for all* $t > T^* > 0$*. Lastly, there exists an absolute constant* $K_1 > 0$ *such that if* $PrRa^{-1} \geq K_1$*, then* (1.8)–(1.13) *has a global attractor in* $V \times H^1_0(\Omega)$*. In particular,* (1.8)–(1.13) *possesses an absorbing ball in* $V \times H^1_0(\Omega)$*; that is, there exist* $\rho_1, \rho_2 > 0$ *such that for any* $R_1, R_2 > 0$*, there is a constant* $t_1(R_1, R_2) > 0$ *such that*

$$(\mathbf{u}(t), T(t)) \in \mathcal{B}_V(\rho_1) \times \mathcal{B}_{H^1(\Omega)}(\rho_2), \quad t \geq t_1(R_1, R_2), \tag{2.14}$$

for all $(\mathbf{u}_0, T_0) \in \mathcal{B}_V(R_1) \times \mathcal{B}_{H^1(\Omega)}(R_2)$*, where*

$$\mathcal{B}(\rho_1, \rho_2) = \mathcal{B}_V(\rho_1) \times \mathcal{B}_{H^1(\Omega)}(\rho_2).$$

Here $\mathcal{B}_X(\rho)$ represents a ball of radius ρ in the space X.

The application of the maximum principle is crucial to obtaining Proposition 2.2. However, it is not immediately clear how to properly adapt the argument for (2.1)–(2.2) [23, 26] to establish a maximum principle (see Lemma 2.5) for the assimilating variable $\tilde{T}$ due to the additional term $-\mu P_N(\tilde{T} - T)$ in (2.2).[1] However, since we are assuming $(\mathbf{u}, T)$ is a regular solution to (1.8)–(1.13), it is straightforward to verify global existence and uniqueness for the associated nudged system by considering instead the corresponding system for the difference $(\mathbf{w}, S)$, where $\mathbf{w} = \tilde{\mathbf{u}} - \mathbf{u}$ and $S = \tilde{T} - T$. In this setting, we need only appeal to a maximum principle for T, rather than for $\tilde{T}$. The well-posedness then follows in a standard fashion [14], which we state as the following result.

[1] See [17] for a case where a maximum principle is necessarily developed in spite of the projection term.

المنارة للاستشارات
www.manaraa.com

Proposition 2.3. *Let $\rho = (\rho_1, \rho_2)$ be as in Proposition 2.2 and $(\mathbf{u}, T)$ satisfy (2.14) for all $t \geq 0$. Suppose $(\tilde{\mathbf{u}}_0, \tilde{T}_0) \in B_{R_1}^{H^1}(\mathbf{0}) \times B_{R_2}^{H^2}(0)$. Then there exists a unique global regular solution $(\tilde{\mathbf{u}}, \tilde{T})$ to (2.1).*

Going forward, we assume that $(\mathbf{u}, T)$ and $(\tilde{\mathbf{u}}, \tilde{T})$ are regular solutions to (2.1)–(2.2) such that $(\mathbf{u}, T)$ belongs to the absorbing ball described by (2.14) in Proposition 2.2. In other words, $(\mathbf{u}, T)$ and $(\tilde{\mathbf{u}}, \tilde{T})$ are as smooth as we need them to be.

2.3 A Priori Estimates

The following are well-known a priori estimates for Stokes' equations [10] and the drift-diffusion equation, where the advecting velocity field is divergence free [23, 26].

Lemma 2.4. *Let $n \geq 2$ and $\Omega \subset \mathbb{R}^n$ be an open, bounded set with boundary $\partial\Omega \in C^2$. For any $\mathbf{f} \in L^2(\Omega)^n$, there exists a unique $\mathbf{u} \in H^2(\Omega)^n \cap V$ and $p \in H^1(\Omega)$ (up to constants) such that*

$$-\Delta \mathbf{u} = -\nabla p + \mathbf{f}, \qquad \nabla \cdot \mathbf{u} = 0, \qquad \mathbf{u}|_{\partial\Omega} = \mathbf{0}.$$

Moreover, there exists an absolute constant $C = C(\Omega) > 0$ such that

$$\|\mathbf{u}(t)\|_{H^2(\Omega)} + \|p(t)\|_{H^1(\Omega)} \leq C\|\mathbf{f}(t)\|_{L^2(\Omega)}.$$

Lemma 2.5. *Let $\mathbf{u}$ be a smooth divergence-free vector field over Ω with boundary conditions $\mathbf{u}|_{x_3=0} = \mathbf{u}|_{x_3=1} = \mathbf{0}$, and let T be a smooth solution to the temperature equation*

$$\partial_t T + \mathbf{u} \cdot \nabla T - \Delta T = 0,$$
$$T(0) = T_0, \qquad T|_{x_3=0} = 1, \qquad T|_{x_3=1} = 0.$$

المنارة للاستشارات
www.manaraa.com

Then there exist $\eta, \bar{T}$ *such that*

$$T = \bar{T} + \eta, \qquad \eta = (T-1)_+ - T_-,$$

and

$$0 \leq \bar{T}(t) \leq 1, \qquad \|\eta(t)\|_{L^2(\Omega)} \leq C\left(\|T_0\|_{L^2(\Omega)} + 1\right) e^{-t},$$

for some absolute constant $C > 0$*. In particular,*

$$\|T(t)\|_{L^2(\Omega)} \leq C_0\left(\|T_0\|_{L^2(\Omega)} + 1\right), \quad t \geq 0.$$

2.4 Main Theorem

We now state and prove the main theorem of this chapter, showing that (2.1) and (2.2) synchronize under certain conditions.

Theorem 2.6 (Infinite Prandtl Synchronization)**.** *Let* $(\mathbf{u}, T)$ *and* $(\tilde{\mathbf{u}}, \tilde{T})$ *satisfy* (2.1)–(2.2)*, and suppose that* $\|T_0\|_{L^2(\Omega)}, \|\tilde{T}_0\|_{L^2(\Omega)} \leq M$ *for some* $M > 0$*. Then there exists an absolute constant* $C_0 > 0$ *such that if*

$$\frac{1}{2}\lambda_N \geq \mu \quad \text{and} \quad \mu \geq \frac{C_0}{2} Ra^2,$$

then

$$\|(\tilde{T} - T)(t)\|^2_{L^2(\Omega)} + Ra^{-2}\|(\tilde{\mathbf{u}} - \mathbf{u})(t)\|^2_{H^2(\Omega)} \in O(e^{-\mu t})$$

for all $t \geq 0$*.*

Proof. Define $\mathbf{w} = \tilde{\mathbf{u}} - \mathbf{u}$, $S = \tilde{T} - T$, and $q = \tilde{p} - p$ and subtract the data momentum

www.manaraa.com

equation in (2.1) from the assimilating momentum equation in (2.2)

$$\begin{aligned}
-\Delta\tilde{\mathbf{u}} + \Delta\mathbf{u} &= -\nabla\tilde{p} + \mathrm{Ra}\mathbf{e}_3\tilde{T} + \nabla p - \mathrm{Ra}\mathbf{e}_3 T \\
-\Delta(\tilde{\mathbf{u}} - \mathbf{u}) &= -\nabla(\tilde{p} - p) + \mathrm{Ra}\mathbf{e}_3(\tilde{T} - T) \\
-\Delta\mathbf{w} &= -\nabla q + \mathrm{Ra}\mathbf{e}_3 S.
\end{aligned}$$

The incompressibility conditions give $\nabla \cdot \mathbf{w} = 0$. For the temperature equations, note this identity that follows from the linearity of the advection operator (see Appendix A.2.6)

$$\begin{aligned}
(\tilde{\mathbf{u}} \cdot \nabla)\tilde{T} - (\mathbf{u} \cdot \nabla)T &= (\tilde{\mathbf{u}} \cdot \nabla)\tilde{T} - (\tilde{\mathbf{u}} \cdot \nabla)T + (\tilde{\mathbf{u}} \cdot \nabla)T - (\mathbf{u} \cdot \nabla)T \\
&= (\tilde{\mathbf{u}} \cdot \nabla)(\tilde{T} - T) + ((\tilde{\mathbf{u}} - \mathbf{u}) \cdot \nabla)T \\
&= (\tilde{\mathbf{u}} \cdot \nabla)S + (\mathbf{w} \cdot \nabla)T.
\end{aligned}$$

Now subtract the temperature equations

$$\begin{aligned}
\partial_t\tilde{T} + (\tilde{\mathbf{u}} \cdot \nabla)\tilde{T} - \partial_t T - (\mathbf{u} \cdot \nabla)T &= \Delta\tilde{T} - \mu P_N(\tilde{T} - T) - \Delta T \\
\partial_t(\tilde{T} - T) + (\tilde{\mathbf{u}} \cdot \nabla)\tilde{T} - (\mathbf{u} \cdot \nabla)T &= \Delta(\tilde{T} - T) - \mu P_N(\tilde{T} - T) \\
\partial_t S + (\tilde{\mathbf{u}} \cdot \nabla)S + (\mathbf{w} \cdot \nabla)T &= \Delta S - \mu P_N(S).
\end{aligned}$$

We therefore arrive at the system

$$-\Delta\mathbf{w} = -\nabla q + \mathrm{Ra}\mathbf{e}_3 S, \tag{2.15}$$

$$\nabla \cdot \mathbf{w} = \mathbf{0}, \tag{2.16}$$

$$\partial_t S + \tilde{\mathbf{u}} \cdot \nabla S + \mathbf{w} \cdot \nabla T = \Delta S - \mu P_N(S), \tag{2.17}$$

$$\mathbf{w}|_{x_3=0} = \mathbf{w}|_{x_3=1} = \mathbf{0}, \qquad S|_{x_3=0} = S|_{x_3=1} = 0, \tag{2.18}$$

$$\mathbf{w}, S \text{ are periodic in } x_1 \text{ and } x_2, \tag{2.19}$$

$$S(\mathbf{x}, 0) = \tilde{T}_0(\mathbf{x}) - T_0(\mathbf{x}).$$

المنارة للاستشارات
www.manaraa.com

Note that (2.15) satisfies Lemma 2.4 with $\mathbf{u} = \mathbf{w}$, $p = q$, and $\mathbf{f} = \mathrm{Ra}\mathbf{e}_3 S$. That is,

$$\|\mathbf{w}\|_{H^2(\Omega)} + \|q\|_{H^1(\Omega)} \leq C\|\mathrm{Ra}\mathbf{e}_3 S\|_{L^2(\Omega)}. \tag{2.20}$$

Therefore, it is sufficient to show that $S \to 0$ in $L^2(\Omega)$. In other words, as long as the temperature fields synchronize, the velocity fields will synchronize as well.

Next, integrating by parts (Proposition B.3) results in the identit.

$$\int_\Omega -S\Delta S \, d\mathbf{x} = \int_\Omega \nabla S \cdot \nabla S \, d\mathbf{x} - \int_{\partial\Omega} S(\nabla S \cdot \boldsymbol{\nu}) \, ds = \int_\Omega \nabla S \cdot \nabla S \, d\mathbf{x}. \tag{2.21}$$

The boundary integral vanishes due to the boundary and periodicity conditions on S in (2.18)–(2.19). We also have, as a result of Corollary B.2 (with $g = S$, $\mathbf{F} = S\tilde{\mathbf{u}}$),

$$\begin{aligned} \int_\Omega S(\tilde{\mathbf{u}} \cdot \nabla)S \, d\mathbf{x} &= -\int_\Omega (\nabla \cdot S\tilde{\mathbf{u}})S \, d\mathbf{x} + \int_{\partial\Omega} S(S\tilde{\mathbf{u}} \cdot \boldsymbol{\nu}) \, d\sigma \\ &= -\int_\Omega (\nabla S \cdot \tilde{\mathbf{u}})S + (\nabla \cdot \tilde{\mathbf{u}})S \, d\mathbf{x} = -\int_\Omega S(\tilde{\mathbf{u}} \cdot \nabla S) \, d\mathbf{x} \\ &\implies \int_\Omega S(\tilde{\mathbf{u}} \cdot \nabla S) \, d\mathbf{x} = 0. \end{aligned} \tag{2.22}$$

In addition, due to (2.7),

$$\begin{aligned} \int_\Omega SP_N(S) \, d\mathbf{x} &= \int_\Omega (Q_N(S) + P_N(S))P_N(S) \, d\mathbf{x} \\ &= \int_\Omega Q_N(S)P_N(S) + P_N(S)^2 \, d\mathbf{x} = \|P_N(S)\|^2_{L^2(\Omega)}. \end{aligned} \tag{2.23}$$

Now multiply (2.17) by S on both sides, integrate over Ω, and use (2.21)–(2.23) to simplify

المنارة للاستشارات
www.manaraa.com

the resulting equation.

$$\begin{aligned}
\int_\Omega SS_t + S(\tilde{\mathbf{u}}\cdot\nabla)S + S(\mathbf{w}\cdot\nabla)T\,d\mathbf{x} &= \int_\Omega S\Delta S - \mu SP_N(S)\,d\mathbf{x} \\
\int_\Omega \frac{d}{dt}\left[\frac{1}{2}S^2\right] - S\Delta S + \mu SP_N(S)\,d\mathbf{x} &= -\int_\Omega S(\tilde{\mathbf{u}}\cdot\nabla)S + S(\mathbf{w}\cdot\nabla)T\,d\mathbf{x} \\
\frac{1}{2}\frac{d}{dt}\int_\Omega S^2\,d\mathbf{x} + \int_\Omega \nabla S\cdot\nabla S\,d\mathbf{x} + \mu\int_\Omega SP_N(S)\,d\mathbf{x} &= -\int_\Omega S(\mathbf{w}\cdot\nabla)T\,d\mathbf{x} \\
\frac{1}{2}\frac{d}{dt}\|S\|^2_{L^2(\Omega)} + \|\nabla S\|^2_{L^2(\Omega)} + \mu\|P_N(S)\|^2_{L^2(\Omega)} &= -\int_\Omega (\mathbf{w}\cdot\nabla T)S\,d\mathbf{x}. \qquad (2.24)
\end{aligned}$$

Assume that $N > 0$ is chosen sufficiently large so that

$$\frac{1}{2}\lambda_N \geq \mu \quad \text{and} \quad \mu \geq \frac{C_0}{2}\mathrm{Ra}^2, \qquad (2.25)$$

where $C_0 > 0$ is, as of yet, unspecified. After an integration by parts, an application of the Sobolev embedding $H^2(\Omega) \hookrightarrow L^\infty(\Omega)$, Lemma 2.4, and Lemma 2.5 imply that there exists a function η satisfying $\|\eta(t)\|_{L^2(\Omega)} \in O(e^{-t})$ such that

$$\begin{aligned}
\left|\int_\Omega (\mathbf{w}\cdot\nabla T)S\,d\mathbf{x}\right| &\leq \|\mathbf{w}(t)\|_{L^\infty(\Omega)}\|T(t)\|_{L^2(\Omega)}\|\nabla S(t)\|_{L^2(\Omega)} \\
&\leq C\mathrm{Ra}\|S(t)\|_{L^2(\Omega)}\|T(t)\|_{L^2(\Omega)}\|\nabla S(t)\|_{L^2(\Omega)} \\
&\leq \frac{1}{2}\|\nabla S(t)\|^2_{L^2(\Omega)} + C\mathrm{Ra}^2\|T(t)\|^2_{L^2(\Omega)}\|S(t)\|^2_{L^2(\Omega)} \\
&\leq \frac{1}{2}\|\nabla S(t)\|^2_{L^2(\Omega)} + C\mathrm{Ra}^2(1 + \|\eta(t)\|^2_{L^2(\Omega)})\|S(t)\|^2_{L^2(\Omega)}. \qquad (2.26)
\end{aligned}$$

On the other hand, due to (2.8) and the inverse Poincaré inequality (Proposition B.5), and since $N > 0$ satisfies the first condition in (2.25) (so that $\frac{1}{2} - \frac{\mu}{\lambda_N} \geq 0$), it follows that

$$\begin{aligned}
\frac{1}{2}\|\nabla S\|^2_{L^2(\Omega)} + \mu\|P_N(S)\|^2_{L^2(\Omega)} &= \frac{1}{2}\|\nabla S\|^2_{L^2(\Omega)} - \mu\|Q_N(S)\|^2_{L^2(\Omega)} + \mu\|S\|^2 \\
&\geq \left(\frac{1}{2} - \frac{\mu}{\lambda_N}\right)\|\nabla S\|^2_{L^2(\Omega)} + \mu\|S\|^2_{L^2(\Omega)} \\
&\geq \mu\|S\|^2_{L^2(\Omega)}. \qquad (2.27)
\end{aligned}$$

المنارة للاستشارات
www.manaraa.com

This last inequality (2.27) can be combined with (2.24) and (2.26) by adding a few terms:

$$\begin{aligned}
\mu\|S\|^2_{L^2(\Omega)} &+ \frac{1}{2}\frac{d}{dt}\|S(t)\|^2_{L^2(\Omega)} + \frac{1}{2}\|\nabla S(t)\|^2_{L^2(\Omega)} \\
&\leq \frac{1}{2}\frac{d}{dt}\|S(t)\|^2_{L^2(\Omega)} + \|\nabla S(t)\|^2_{L^2(\Omega)} + \mu\|P_N(S)\|^2_{L^2(\Omega)} \\
&= -\int_\Omega (\mathbf{w}\cdot\nabla T)S\,d\mathbf{x} \\
&\leq \left|\int_\Omega (\mathbf{w}\cdot\nabla T)S\,d\mathbf{x}\right| \\
&\leq \frac{1}{2}\|\nabla S(t)\|^2_{L^2(\Omega)} + C\mathrm{Ra}^2(1+\|\eta(t)\|^2_{L^2(\Omega)})\|S(t)\|^2_{L^2(\Omega)}.
\end{aligned}$$

Combining like terms, we arrive at

$$\frac{d}{dt}\|S(t)\|^2_{L^2(\Omega)} + 2\left(\mu - C\mathrm{Ra}^2(1+\|\eta(t)\|^2_{L^2(\Omega)})\right)\|S(t)\|^2_{L^2(\Omega)} \leq 0.$$

This satisfies Gronwall's inequality (Proposition B.4 with $f(t) = \|S(t)\|^2_{L^2(\Omega)}$), by which we obtain

$$\|S(t)\|^2 \leq \|S(0)\|^2_{L^2(\Omega)} \exp\left(-2\mu t + C\mathrm{Ra}^2\int_0^t (1+\|\eta(s)\|^2_{L^2(\Omega)})\,ds\right).$$

Finally, by Lemma 2.5 and the second condition in (2.25), we deduce

$$\begin{aligned}
\|S(t)\|^2 &\leq \|S(0)\|^2_{L^2(\Omega)} \exp\left(-2\mu t + C\mathrm{Ra}^2\int_0^t (1+\|\eta(s)\|^2_{L^2(\Omega)})ds\right) \\
&\leq \|S(0)\|^2_{L^2(\Omega)} \exp\left(C_0\mathrm{Ra}^2\left(\|T_0\|^2_{L^2(\Omega)}+1\right)\right)\exp\left(-\mu t\right),
\end{aligned} \tag{2.28}$$

as desired. □

Theorem 2.6 shows that, given Ra and reasonable boundary conditions T_0 and $\tilde{T}_0$, we can always choose μ and N large enough so that $(\mathbf{u}, T)$ and $(\tilde{\mathbf{u}}, \tilde{T})$ eventually match. In the next chapter, we use numerical simulations to verify that this is indeed the case.

المنارة للاستشارات
www.manaraa.com

In this chapter we continue examining the data assimilation problem (2.1)–(2.2), but from a numerical perspective. The objective is to validate the results of Theorem 2.6 and probe the sharpness of the inequality condition

$$\mu \geq \frac{C_0}{2}\mathrm{Ra}^2 \tag{3.1}$$

by strategically varying Ra, μ, and N.

3.1 Dimension Reduction

Numerical fluid simulations are computationally expensive in three dimensions, but the analysis of the previous chapter also applies to the two-dimensional setting where Ω is replaced with $\Psi = [0, L] \times [0, 1]$ (with coordinates $[x_1, x_3]$). In two dimensions, there are a few preliminary simplifications that make the numerical work more straightforward.

Let $\mathbf{u} : \Psi \times \mathbb{T} \to \mathbb{R}^2$ have components $[u_1, u_3]^\mathsf{T}$. The incompressibility condition becomes

$$0 = \nabla \cdot \mathbf{u} = \partial_{x_1} u_1 + \partial_{x_3} u_3,$$

which can always be satisfied by introducing a *stream function* $\psi : \Psi \to \mathbb{R}$ [24, 15] such that

$$u_1 = -\partial_{x_3}\psi, \qquad u_3 = \partial_{x_1}\psi.$$

Defining $\zeta = \Delta\psi$, we have the identities

$$\mathbf{u} = \begin{bmatrix} u_1 \\ u_3 \end{bmatrix} = \begin{bmatrix} -\psi_{x_3} \\ \psi_{x_1} \end{bmatrix}, \qquad \begin{aligned} \nabla \cdot \mathbf{u} &= -\psi_{x_3 x_1} + \psi_{x_1 x_3} = 0, \\ \nabla \times \mathbf{u} &= \partial_{x_1} u_3 - \partial_{x_3} u_1 = \psi_{x_1 x_1} + \psi_{x_3 x_3} = \Delta\psi = \zeta, \\ \nabla \times ((\mathbf{u} \cdot \nabla)\mathbf{u}) &= (\mathbf{u} \cdot \nabla)(\nabla \times \mathbf{u}). \end{aligned}$$

المنارة للاستشارات
www.manaraa.com

The last equation is a consequence of Lemma A.1 because

$$\partial_{x_1} u_1 = -\psi_{x_3 x_1} = -\psi_{x_1 x_3} = -\partial_{x_3} u_3,$$

and because $\mathbf{u}$ has only two components.

Now the momentum equation can be simplified by taking the curl of both sides:

$$\begin{aligned}
\nabla \times (-\Delta \mathbf{u}) &= \nabla \times (-\nabla p + \mathrm{Ra}\mathbf{e}_3 T) \\
-\Delta(\nabla \times \mathbf{u}) &= -(\nabla \times \nabla p) + \mathrm{Ra}(\nabla \times \mathbf{e}_3 T) \\
-\Delta(\Delta \psi) &= \mathrm{Ra}(T_{x_1} - 0) \\
-\Delta \zeta &= \mathrm{Ra} T_{x_1}. \qquad (3.2)
\end{aligned}$$

The main advantage of (3.2) is that the pressure p drops out from (2.1); it also reduces the momentum equation to a scalar equation that can be written without using $\mathbf{u}$ explicitly. Repeating this reduction process for the assimilating system (2.2), we arrive at the simplified data assimilation problem

$$\begin{aligned}
-\Delta \zeta &= \mathrm{Ra} T_{x_1}, & \partial_{x_i}\psi|_{x_3=0} = \partial_{x_i}\psi|_{x_3=1} = 0,\ i = 1, 3, \\
\Delta \psi &= \zeta, & T|_{x_3=0} = 1,\ T|_{x_3=1} = 0, \\
\partial_t T - \psi_{x_3} T_{x_1} + \psi_{x_1} T_{x_3} &= \Delta T, & \psi_{x_1}, \psi_{x_3}, T \text{ are periodic in } x_1, \\
& & T(\mathbf{x}, 0) = T_0(\mathbf{x}),
\end{aligned} \qquad (3.3)$$

$$\begin{aligned}
-\Delta \tilde{\zeta} &= \mathrm{Ra} \tilde{T}_{x_1}, & \partial_{x_i}\tilde{\psi}|_{x_3=0} = \partial_{x_i}\tilde{\psi}|_{x_3=1} = 0,\ i = 1, 3, \\
\Delta \tilde{\psi} &= \tilde{\zeta}, & \tilde{T}|_{x_3=0} = 1,\ \tilde{T}|_{x_3=1} = 0, \\
\partial_t \tilde{T} - \tilde{\psi}_{x_3} \tilde{T}_{x_1} + \tilde{\psi}_{x_1} \tilde{T}_{x_3} &= \Delta \tilde{T} - \mu P_N(\tilde{T} - T), & \tilde{\psi}_{x_1}, \tilde{\psi}_{x_3}, \tilde{T} \text{ are periodic in } x_1, \\
& & \tilde{T}(\mathbf{x}, 0) = \tilde{T}_0(\mathbf{x}).
\end{aligned} \qquad (3.4)$$

The implementation of (3.3)–(3.4) with Dedalus is fairly straightforward, if a bit lengthy.

المنارة للاستشارات
www.manaraa.com

3.2 Minimal Simulation Code

3.2.1 Projection Operator. The most non-conventional part of (3.3)–(3.4) is the projection operator P_N. Since Dedalus represents state variables in coefficient space, P_N can be implemented by directly setting certain entries of a state variable's coefficient array to zero. For simplicity, we alter the coefficients in both the x_1 and x_3 directions.

```
import numpy as np

def P_N(F, N, scale=False):
    """Calculate the Fourier mode projection of F with N terms."""
    # Set the spectral coefficients to 0 wherever n > N (in both axes).
    X,Y = np.indices(F['c'].shape)
    F['c'][(X >= N) | (Y >= N)] = 0

    if scale:
        F.set_scales(1)
    return F['g']
```

3.2.2 Problem Definition. The following code translates (3.3)–(3.4) into a Dedalus system where $\Psi = [0,4] \times [0,1]$, $\mathrm{Ra} = 10000$, $\mu = 1$, and $N = 32$. In the code, x_1 and x_3 are called `x` and `z`, and the components u_1 and u_3 of $\mathbf{u}$ are called `v` and `w`, respectively. Assimilating variables are denoted with a trailing underscore: `T` represents T, `T_` represents $\tilde{T}$, and so on.

```
from dedalus import public as de
from dedalus.extras import flow_tools
from dedalus.core.operators import GeneralFunction


# System Variables -------------------------------------------------------------
L = 4
xsize = 256
zsize = 128
Rayleigh=10000
mu = 1
N = 32

# Bases and Domain -------------------------------------------------------------
x_basis = de.Fourier('x', xsize, interval=(0, L), dealias=3/2)
z_basis = de.Chebyshev('z', zsize, interval=(0, 1), dealias=3/2)
domain = de.Domain([x_basis, z_basis], grid_dtype=np.float64)

# Initialize the problem as an IVP and add variables --------------------------
problem = de.IVP(domain, variables=['T', 'T_',      # Temperature
                                    'Tz', 'Tz_',
                                    'psi', 'psi_',  # Stream function
                                    'psiz', 'psiz_',
                                    'zeta', 'zeta_',# Laplace of stream
                                    'zetaz', 'zetaz_'])

driving = GeneralFunction(domain, 'g', P_N, args=[])
```


المنارة للاستشارات
www.manaraa.com

```
# System parameters (these are saved to a JSON file).
problem.parameters['L'] = L                          # Domain parameters
problem.parameters['xsize'] = xsize
problem.parameters['zsize'] = zsize

problem.parameters['Ra'] = Rayleigh                  # Rayleigh number
problem.parameters["N"] = N                          # Assimilation parameters
problem.parameters["mu"] = mu
problem.parameters["driving"] = driving

# Stream function substitutions: u = [v, w] = [-psi_z, psi_w]
problem.substitutions['v']  = "-dz(psi)"
problem.substitutions['v_'] = "-dz(psi_)"
problem.substitutions['w']  =  "dx(psi)"
problem.substitutions['w_'] =  "dx(psi_)"

# Relate higher-order z derivatives (b/c Chebyshev).
problem.add_equation("psiz  - dz(psi)  = 0")
problem.add_equation("psiz_ - dz(psi_) = 0")
problem.add_equation("zetaz  - dz(zeta)  = 0")
problem.add_equation("zetaz_ - dz(zeta_) = 0")
problem.add_equation("Tz  - dz(T)  = 0")
problem.add_equation("Tz_ - dz(T_) = 0")

# zeta = laplace(psi)
problem.add_equation("zeta  - dx(dx(psi))  - dz(psiz)  = 0")
problem.add_equation("zeta_ - dx(dx(psi_)) - dz(psiz_) = 0")

# 2D Boussinesq Equations -----------------------------------------------------
# Ra T_x + laplace(zeta) = 0 (Infinite Prandtl number)
problem.add_equation("Ra*dx(T)  + dx(dx(zeta))  + dz(zetaz)  = 0")
problem.add_equation("Ra*dx(T_) + dx(dx(zeta_)) + dz(zetaz_) = 0")
# T_t - laplace(T) = -u . grad(T) (+ driving Fourier projection)
problem.add_equation("dt(T)  - dx(dx(T))  - dz(Tz) = -v*dx(T)   - w*Tz")
problem.add_equation("dt(T_) - dx(dx(T_)) - dz(Tz_) "
                                    "= -v_*dx(T_) - w_*Tz_ - mu*driving")

# Boundary Conditions ---------------------------------------------------------
# Temperature heating from 'left' (bottom), cooling from 'right' (top).
problem.add_bc("left(T)  = 1")                       # T(z=0) = 1
problem.add_bc("left(T_) = 1")
problem.add_bc("right(T)  = 0")                      # T(z=1) = 0
problem.add_bc("right(T_) = 0")

# Velocity field boundary conditions: no-slip.
# u(z=0) = 0 --> v(z=0) = 0 = w(z=0)
problem.add_bc("left(psiz)  = 0")
problem.add_bc("left(psiz_) = 0")
problem.add_bc("left(psi)  = 0")
problem.add_bc("left(psi_) = 0")

# u(z=1) = 0 --> v(z=1) = 0 = w(z=1)
problem.add_bc("right(psiz)  = 0")
problem.add_bc("right(psiz_) = 0")
problem.add_bc("right(psi)  = 0")
problem.add_bc("right(psi_) = 0")

# Construct Solver ------------------------------------------------------------
# Build the solver object for the problem.
solver = problem.build_solver(de.timesteppers.RK443)
```


المنارة للاستشارات
www.manaraa.com

3.2.3 Initial Conditions. The Dedalus example in Section 1.4 is linear a boundary value problem with no time variable, so no initial conditions are required. In this situation, however, we must specify the initial states T_0 and $\tilde{T}_0$. For T_0, there are two options.

- Set $T_0(x_1, x_3) = 1 - x_3 + \varepsilon(x_1, x_3)$ for some small perturbation function $\varepsilon : \Psi \to \mathbb{R}$. This begins the simulation almost "from rest" which—though not physically relevant—is the necessary starting point for initial experiments.

- Load T_0 from the final state of a previous simulation with similar parameters. If said simulation has run long enough, T will thus begin in a realistic convective state.

The correct choice for $\tilde{T}_0$ is a little less obvious. Setting $\tilde{T}_0(x_1, x_3) = 1 - x_3$ assumes no prior knowledge of T, but this strategy often results in an overly stiff scheme due to the strength of the initial temperature difference $\|\tilde{T}_0 - T_0\|^2_{L^2(\Omega)}$, and is therefore numerically impractical. We could set $\tilde{T}_0 = P_N(T_0)$, since we assume that we can regularly observe the first N modes of T at any given time. However, this often results in a very weak difference $\|\tilde{T}_0 - T_0\|^2_{L^2(\Omega)}$ so that (3.3) and (3.4) appear essentially synchronized at $t = 0$. Instead, we set $\tilde{T}_0$ as a low-mode projection of T_0 such as $\tilde{T}_0 = P_4(T_0)$. This is the "happy medium" where $\|\tilde{T}_0 - T_0\|^2_{L^2(\Omega)}$ is small enough to avoid numerical stiffness, but large enough to observe significant synchronization.

The following code sets $T_0(x_1, x_3) = 1 - x_3 + \varepsilon \sin(kx_1)\sin(2\pi x_3)$ where $\varepsilon = .0001$ and $k = 3.117$ and initializes $\tilde{T}_0 = P_4(T_0)$.

```
# Initial conditions ----------------------------------------------------------
eps = 1e-4
k = 3.117
x,z = problem.domain.grids(scales=1)

# Start T from rest plus a small perturbation.
T, T_ = solver.state['T'], solver.state['T_']
T['g']  = 1 - z + eps*np.sin(k*x)*np.sin(2*np.pi*z)
T.differentiate('z', out=solver.state['Tz'])

# Start T_ as a low-mode projection of T (N = 4).
G = problem.domain.new_field()
G['c'] = solver.state['T']['c'].copy()
T_['g'] = P_N(G, 4, True)
T_.differentiate('z', out=solver.state['Tz_'])
```


المنارة للاستشارات
www.manaraa.com

3.2.4 Iterative Solution. Having chosen initial conditions, the solver computes the state variables at discrete time steps until reaching some stopping point T^*. This requires some care: it is possible for the solver to mishandle the computation due to numerical error, so at each iteration we check that the quantities

$$\frac{1}{\text{Ra}}\|\mathbf{u}(t)\|_2 \qquad \text{and} \qquad \frac{1}{\text{Ra}}\|\tilde{\mathbf{u}}(t)\|_2$$

are finite.

Dedalus makes it easy to save other useful quantities like $\|(\tilde{T} - T)(t)\|_{L^2(\Omega)}$ at each time step. This version of the code only saves the state variables at each iteration, but the full code (see Appendix C) saves each of the normed quantities used in the next section.

```
import time

# Driving / projection function (P_N) arguments ------------------------------
dT = solver.state['T_'] - solver.state['T']
problem.parameters["driving"].args = [dT, N]
problem.parameters["driving"].original_args = [dT, N]

# Stopping Parameters ---------------------------------------------------------
solver.stop_sim_time = 2                        # Length of simulation.
solver.stop_wall_time = np.inf                  # Real time allowed to compute.
solver.stop_iteration = np.inf                  # Maximum iterations allowed.

# State snapshots -------------------------------------------------------------
# Save the temperature measurements in states/ files.
snaps = solver.evaluator.add_file_handler("states", sim_dt=.05,
                                          max_writes=5000, mode="append")
snaps.add_task("T")
snaps.add_task("T_")
snaps.add_task("driving", name="P_N")

# Control Flow ----------------------------------------------------------------
cfl = flow_tools.CFL(solver, initial_dt=1e-4, cadence=1, safety=1,
                                            max_change=1.5, min_change=0.01,
                                            max_dt=0.01,    min_dt=1e-10)
cfl.add_velocities(('v',  'w' ))
cfl.add_velocities(('v_', 'w_'))

# Flow properties (print during run; not recorded in the records files)
flow = flow_tools.GlobalFlowProperty(solver, cadence=1)
flow.add_property("sqrt(v **2 + w **2) / Ra", name='Re' )
flow.add_property("sqrt(v_**2 + w_**2) / Ra", name='Re_')

# MAIN COMPUTATION LOOP -------------------------------------------------------
try:
    print("Starting main loop")
    start_time = time.time()
    while solver.ok:
        # Recompute time step and iterate.
        dt = solver.step(cfl.compute_dt())
```


المنارة للاستشارات
www.manaraa.com

```
        # Print info to the screen every 10 iterations.
        if solver.iteration % 10 == 0:
            info = "Iteration {:>5d}, Time: {:.7f}, dt: {:.2e}".format(
                                solver.iteration, solver.sim_time, dt)
            Re  = flow.max("Re")
            Re_ = flow.max("Re_")
            info += ", Max Re = {:f}".format(Re)
            info += ", Max Re_ = {:f}".format(Re_)
            print(info)
            # Make sure the simulation hasn't blown up.
            if np.isnan(Re) or np.isnan(Re_):
                raise ValueError("Reynolds number went to infinity!!"
                                 "\nRe = {}, Re_ = {}".format(Re, Re_))
except BaseException as e:
    print("Exception raised, triggering end of main loop.")
    raise
finally:
    total_time = time.time() - start_time
    print("Iterations: {:d}".format(solver.iteration))
    print("Sim end time: {:.3e}".format(solver.sim_time))
    print("Run time: {:.3e} sec".format(total_time))
    print("END OF SIMULATION\n")
```

See Appendix C for the full code, including implementations for loading T_0 from an existing simulation, visualizing results, and managing data files.

3.3 Results

Values of Ra that are of interest for mantle convection are typically between 10^7 and 10^8 [25]. However, the greater Ra, the more computationally expensive the simulation, and it is numerically unstable to start T in an unnatural state such as $T_0 \approx 0$ at large Ra. We therefore select logarithmically spaced Ra values from 10^4 to 10^9 and run a simulation for each Ra, one after another. For the very first simulation (Ra $= 10^4$), we set $T_0(x_1, x_3) = 1 - x_3 + \varepsilon(x_1, x_3)$ as in Section 3.2.3 and evolve the systems forward until reaching a nearly steady state. Thereafter, we set T_0 as the final T from the previous simulation and increment Ra. This gives us realistic initial conditions for any given Ra.

For the assimilating system, we use the initial condition $\tilde{T}_0 = P_4(T_0)$ as discussed in Section 3.2.3. Figure 3.1 shows an example of T_0 and $\tilde{T}_0$, together with subsequent end states of T and $\tilde{T}$ for Ra $\approx 5.22 \times 10^7$.

www.manaraa.com

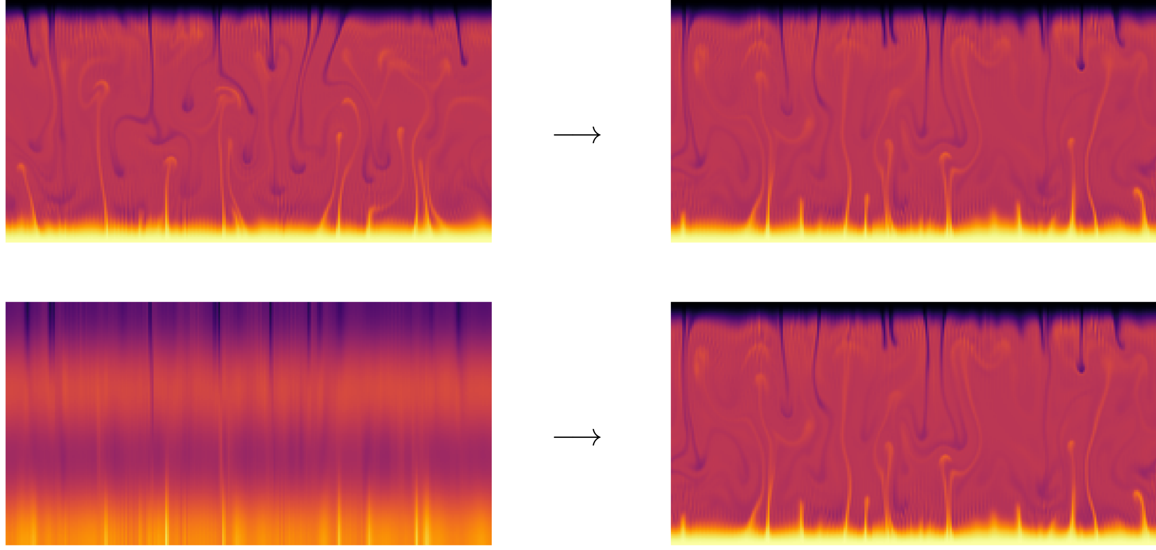

Figure 3.1: Typical "before" and "after" snapshots of a successful numerical experiment. On the top, the initial condition T_0 compared with T at the end of a simulation. On the bottom, the initial projected condition $\tilde{T}_0 = P_4(T_0)$ and the eventual $\tilde{T}$ at the end of the simulation. Note that the post-simulation T and $\tilde{T}$ appear identical to the naked eye. This particular simulation uses $\mathrm{Ra} \approx 5.22 \times 10^7$, $\mu = 12,700$, and $N = 32$.

To measure the synchronization of (3.3) and (3.4), we keep track of the norms

$$\|(\tilde{T} - T)(t)\|_{L^2(\Omega)}, \qquad \|(\tilde{\mathbf{u}} - \mathbf{u})(t)\|_{L^2(\Omega)}, \tag{3.5}$$

$$\|(\nabla\tilde{T} - \nabla T)(t)\|_{L^2(\Omega)}, \qquad \|(\nabla\tilde{\mathbf{u}} - \nabla\mathbf{u})(t)\|_{L^2(\Omega)}, \tag{3.6}$$

$$\|(\tilde{T} - T)(t)\|_{H^2(\Omega)}, \qquad \|(\tilde{\mathbf{u}} - \mathbf{u})(t)\|_{H^2(\Omega)}, \tag{3.7}$$

throughout the simulation. The $H^2(\Omega)$ norms in (3.7) are calculated with the equivalent formulations

$$\|T\|_{H^2(\Omega)} = \left(\int_\Omega (\partial_{x_1}^2 T)^2 + (\partial_{x_1}\partial_{x_3} T)^2 + (\partial_{x_3}^2 T)^2 \, d\mathbf{x}\right)^2,$$

$$\|\mathbf{u}\|_{H^2(\Omega)} = \left(\int_\Omega (\partial_{x_1}^2 u_1)^2 + (\partial_{x_1}\partial_{x_3} u_1)^2 + (\partial_{x_3}^2 u_1)^2 + (\partial_{x_1}^2 u_3)^2 + (\partial_{x_1}\partial_{x_3} u_3)^2 + (\partial_{x_3}^2 u_3)^2 \, d\mathbf{x}\right)^2,$$

and each error is normalized by dividing by the norms of the data variables. For example,

المنارة للاستشارات
www.manaraa.com

to measure the temperature difference in $L^2(\Omega)$, we compute

$$\frac{\|(\tilde{T} - T)(t)\|_{L^2(\Omega)}}{\|T(t)\|_{L^2(\Omega)}}$$

at each simulation time t. For simplicity, we write these errors without the denominators.

Figure 3.2 shows how the error norms (3.5)–(3.7) decrease in the simulation from Figure 3.1. Though the analysis indicates that these norms should converge to $\varepsilon_{\text{machine}} \approx 10^{-16}$, seeing the errors decrease to about 10^{-9} is numerically satisfactory in this setting.

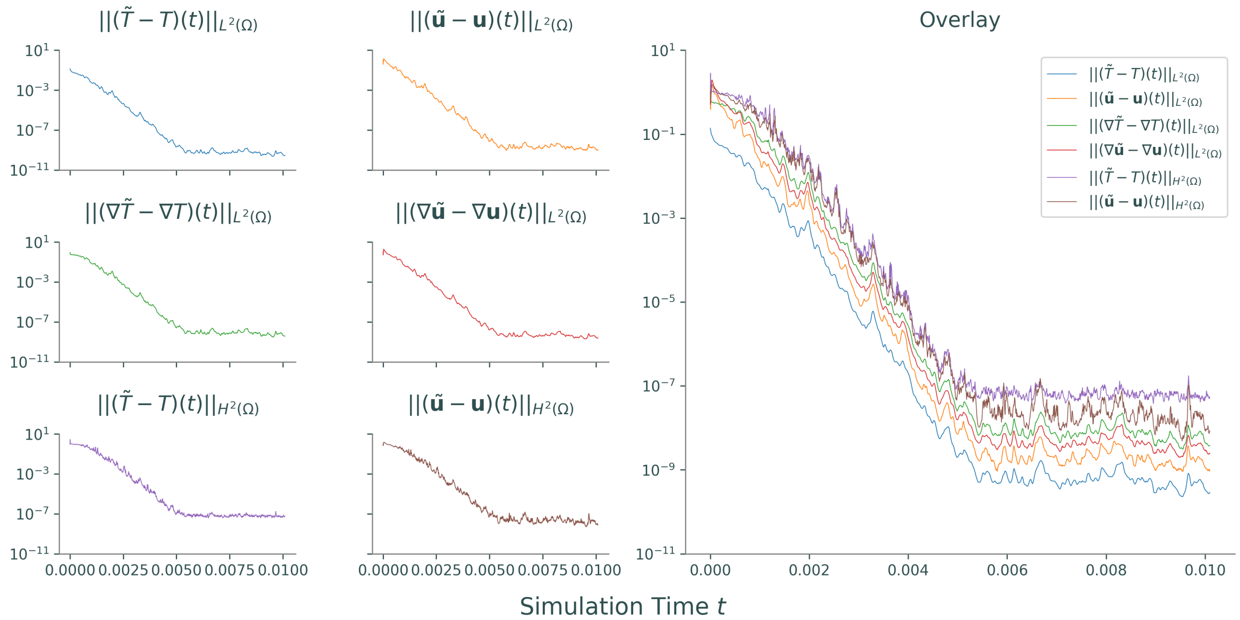

Figure 3.2: Synchronization in various norms for Ra $\approx 5.22 \times 10^7$, $\mu = 12,700$, and $N = 32$ (the same conditions used in Figure 3.1). The temperature and velocity differences decrease exponentially until flattening out at sufficiently small values. The temperature difference is the smallest, a consequence of using only temperature observations for the assimilation.

3.3.1 Failure of the Usual Approach.

Other studies of data assimilation for Rayleigh-Bénard convection usually fix $\mu = 1$ for all experiments and instead focus on the effects of another parameter, such as the number of projection modes [2, 13]. However, Theorem 2.6 requires that μ be proportional to Ra^2, so we do not expect synchronization with $\mu = 1$ as Ra increases. Indeed, Figure 3.3 shows that for Ra in the range of 4×10^7 to 2×10^8, using $\mu = 1$ results in either extremely slow convergence, or in no convergence at all.

المنارة للاستشارات
www.manaraa.com

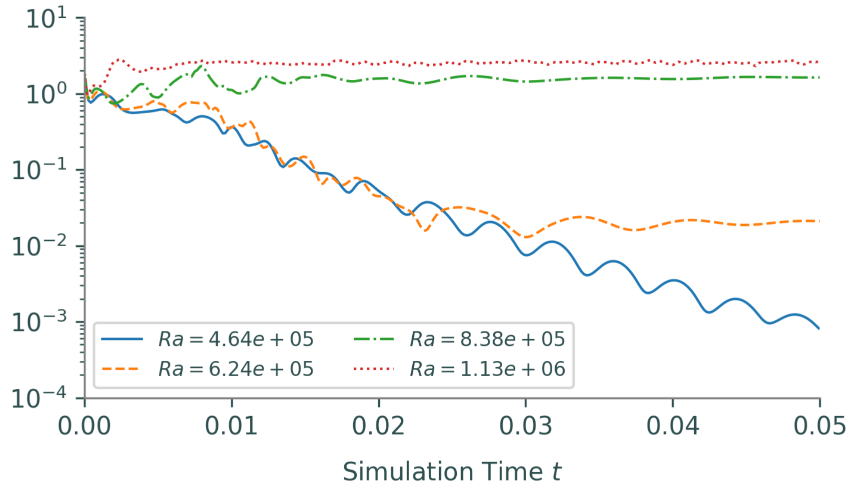

Figure 3.3: Convergence (or lack thereof) of $\|(\tilde{\mathbf{u}}-\mathbf{u})(t)\|_{H^2(\Omega)}+\|(\tilde{T}-T)(t)\|_{H^2(\Omega)}$ with $\mu=1$ and $N=32$ for various values of Ra. For lower Ra, setting $\mu=1$ is sufficient to very slowly induce synchronization; however, as Ra increases, $\mu=1$ quickly becomes too weak to nudge the assimilating system toward the data at all.

3.3.2 Varying the Relaxation Parameter.

To explore the relationship between Ra, μ, and N needed for synchronization to occur, we fix two of the parameters at a time and run several simulations with various values for the remaining parameter. First, we fix Ra, set $N=32$, and vary μ until synchronization can be observed within 0.005 units of simulation time.[1] Table 3.1 records the smallest value of μ where synchronization was observed; Figure 3.4 shows the convergence rates for a few different μ for Ra $\approx 3.89\times10^7$ and Ra $\approx 7.02\times10^7$.

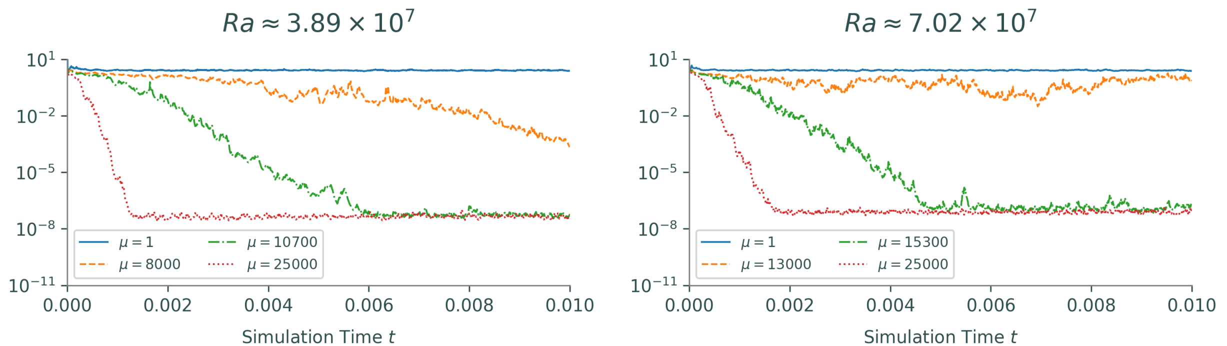

Figure 3.4: $\|(\tilde{\mathbf{u}}-\mathbf{u})(t)\|_{H^2(\Omega)}+\|(\tilde{T}-T)(t)\|_{H^2(\Omega)}$ with $N=32$ and various μ for two different values of Ra. We begin to see satisfactory convergence around $\mu=10,700$ and $\mu=15,300$ for Ra $\approx 3.89\times10^7$ and Ra $\approx 7.02\times10^7$, respectively.

[1] A simulation of this length still usually requires several thousand iterations.

المنارة للاستشارات
www.manaraa.com

Ra	μ
1.1937766×10^7	$6,300$
1.6037187×10^7	$6,500$
2.1544346×10^7	$7,900$
2.8942661×10^7	$9,400$
3.8881551×10^7	$10,700$
5.2233450×10^7	$12,700$
7.0170382×10^7	$15,300$
9.4266845×10^7	$19,400$
1.26638017×10^8	$24,900$
1.70125427×10^8	$29,900$

Table 3.1: Minimal values of μ that result in synchronization within about 0.005 units of simulation time for the given Ra with $N = 32$. These values represent the edge of what works when $N = 32$: a lower μ may not stimulate convergence, but a larger μ will. See Figure 3.5 for a quadratic least-squares fit.

Given the requirement (3.1) from Theorem 2.6, it is surprising to discover that—based on Table 3.1—the relationship between μ and Ra is more linear than quadratic. Indeed, a least squares fit of the data to a general quadratic of the form $f(x) = a + bx + cx^2$ yields coefficients of $a \approx 4.03 \times 10^3$, $b \approx 1.77 \times 10^{-4}$, and $c \approx -1.37 \times 10^{-13}$ (essentially $c = 0$). See Figure 3.5.

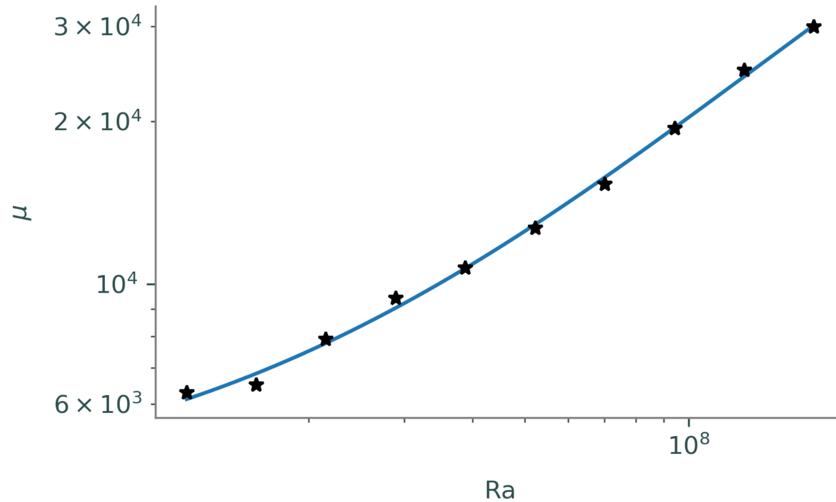

Figure 3.5: Quadratic (but nearly linear) least-squares fit for the data in Table 3.1.

www.manaraa.com

3.3.3 Varying the Projection.

The relaxation parameter μ is a system parameter without much physical interpretation. The number of projected Fourier modes N, on the other hand, indicates the amount of data that is "visible" to the assimilating system. Therefore, a lower bound on N represents how much data is required in order to maintain an accurate model. Fixing μ as given by Table 3.1, we decrease N to see how it affects synchronization. See Figure 3.6.

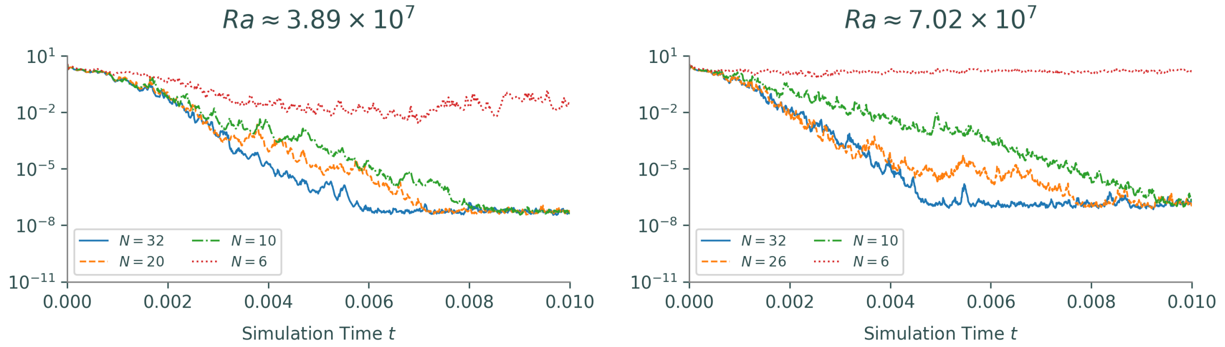

Figure 3.6: $\|(\tilde{\mathbf{u}} - \mathbf{u})(t)\|_{H^2(\Omega)} + \|(\tilde{T} - T)(t)\|_{H^2(\Omega)}$ for two different values of Ra, with μ as listed in Tables 3.1 and 3.2 and for various values of N. For both Ra $\approx 3.89 \times 10^7$ and Ra $\approx 7.02 \times 10^7$, $N = 10$ appears to be the least number of modes that results in synchronization.

Ra	μ	N
1.1937766×10^7	$6,300$	6
1.6037187×10^7	$6,500$	8
2.1544346×10^7	$7,900$	8
2.8942661×10^7	$9,400$	8
3.8881551×10^7	$10,700$	10
5.2233450×10^7	$12,700$	10
7.0170382×10^7	$15,300$	10
9.4266845×10^7	$19,400$	10
1.26638017×10^8	$24,900$	12
1.70125427×10^8	$29,900$	14

Table 3.2: Minimal values of N that result in synchronization within about 0.01 units of simulation time for the given Ra and μ. In general, as long as an appropriate μ is chosen, $N = 16$ is sufficient for physically relevant values of Ra.

3.3.4 Summary.

The numerical simulations show that for relevant Ra, we can pick μ large enough to nudge (3.4) toward (3.3); in particular, the experiments confirm Theorem

المنارة للاستشارات
www.manaraa.com

2.6 and show that the conditions stated therein are stronger than necessary. Furthermore, assimilation can be obtained with a relatively low number of modes even for large Ra, as long as μ is chosen large enough. We took the approach of varying μ first and then N, but it is just as feasible to vary μ for a fixed Ra and N.

المنارة للاستشارات
www.manaraa.com

In this chapter we return to the original Boussinesq approximation (1.8)–(1.13) with finite Prandtl number $\mathrm{Pr} < \infty$. We give an overview of two data assimilation schemes: one where the truth and nudging systems uses a momentum equation with $\mathrm{Pr} < \infty$, and one where the truth system uses a $\mathrm{Pr} < \infty$ momentum equation but the nudging system uses the $\mathrm{Pr} = \infty$ formulation. The technical details are similar to those presented in Chapter 2, so here we simplify the presentation of the analysis.

4.1 Finite Prandtl Assimilation

Consider the data assimilation problem

$$\begin{aligned} &\tfrac{1}{\mathrm{Pr}}\left[\partial_t \mathbf{u} + (\mathbf{u}\cdot\nabla)\mathbf{u}\right] - \Delta\mathbf{u} = -\nabla p + \mathrm{Ra}\mathbf{e}_3 T, \\ &\nabla\cdot\mathbf{u} = 0, \\ &\partial_t T + \mathbf{u}\cdot\nabla T = \Delta T, \end{aligned} \qquad \begin{aligned} &\mathbf{u}|_{x_3=0} = \mathbf{u}|_{x_3=1} = \mathbf{0}, \\ &T|_{x_3=0} = 1,\ T|_{x_3=1} = 0, \\ &\mathbf{u}, T \text{ are periodic in } x_1 \text{ and } x_2, \\ &T(\mathbf{x},0) = T_0(\mathbf{x}),\ \mathbf{u}(\mathbf{x},0) = \mathbf{u}_0(\mathbf{x}), \end{aligned} \tag{4.1}$$

$$\begin{aligned} &\tfrac{1}{\mathrm{Pr}}\left[\partial_t \tilde{\mathbf{u}} + (\tilde{\mathbf{u}}\cdot\nabla)\tilde{\mathbf{u}}\right] - \Delta\hat{\mathbf{u}} = -\nabla \hat{p} + \mathrm{Ra}\mathbf{e}_3 \hat{T}, \\ &\nabla\cdot\hat{\mathbf{u}} = 0, \\ &\partial_t \hat{T} + \hat{\mathbf{u}}\cdot\nabla \hat{T} = \Delta \hat{T} - \mu P_N(\hat{T} - T), \end{aligned} \qquad \begin{aligned} &\hat{\mathbf{u}}|_{x_3=0} = \hat{\mathbf{u}}|_{x_3=1} = \mathbf{0}, \\ &\hat{T}|_{x_3=0} = 1,\ \hat{T}|_{x_3=1} = 0, \\ &\hat{\mathbf{u}}, \hat{T} \text{ are periodic in } x_1 \text{ and } x_2, \\ &\hat{T}(\mathbf{x},0) = \hat{T}_0(\mathbf{x}),\ \hat{\mathbf{u}}(\mathbf{x},0) = \hat{\mathbf{u}}_0(\mathbf{x}). \end{aligned} \tag{4.2}$$

In this setting, $\mathbf{u}$ and T are the data variables and $\hat{\mathbf{u}}$ and $\hat{T}$ are the assimilating variables.

4.1.1 Rigorous Results. Using the same functional setting and setup as in Chapter 2 leads to the following result, the analog of Theorem 2.6 for $0 \ll \mathrm{Pr} < \infty$. We do not provide the proof here, but the ideas are the same as in the proof of Theorem 2.6.

www.manaraa.com

Theorem 4.1. *Let* $(\mathbf{u}, T)$ *and* $(\hat{\mathbf{u}}, \hat{T})$ *satisfy* (4.1)–(4.2) *with* $(\mathbf{u}_0, T_0), (\hat{\mathbf{u}}_0, \hat{T}_0) \in \mathcal{B}(\rho_1, \rho_2)$, *respectively. There exist absolute constants* $C_0, C_1, C_2 > 0$ *such that if*

$$\frac{1}{2}\lambda_N \geq \mu, \quad \mu \geq \left(\frac{1}{2} + Ra^2\right) Pr, \tag{4.3}$$

and

$$Pr \geq \max\{1, 4C_0^{1/2} \max\{C_2^{1/2}, C_1^{1/2}\}(1 + Ra)^5\}, \tag{4.4}$$

then

$$\|(\hat{\mathbf{u}} - \mathbf{u})(t)\|_{L^2(\Omega)}^2 + \|(\hat{T} - T)(t)\|_{L^2(\Omega)}^2 \leq O(e^{-(Pr/2)t})$$

holds for $t \geq 0$.

Note that the results of Theorem 4.1 are less optimistic than those of Theorem 2.6 in the sense that the inequality conditions on μ are much harsher, not to mention the strong requirements on Pr in relation to Ra. Still, we expect to see some synchronization for reasonably large values of μ and Pr.

4.1.2 Numerical Results. The dimension reduction process is the same as in Section 3.1, except now the momentum equation has the following extra terms:

$$\begin{aligned}
\nabla \times \left(\frac{1}{\mathrm{Pr}}\left[\partial_t \mathbf{u} + (\mathbf{u} \cdot \nabla)\mathbf{u}\right]\right) &= \frac{1}{\mathrm{Pr}}\left[(\nabla \times \partial_t \mathbf{u}) + \nabla \times ((\mathbf{u} \cdot \nabla)\mathbf{u})\right] \\
&= \frac{1}{\mathrm{Pr}}\left[\partial_t(\nabla \times \mathbf{u}) + (\mathbf{u} \cdot \nabla)(\nabla \times \mathbf{u})\right] \\
&= \frac{1}{\mathrm{Pr}}\left[\zeta_t + (\mathbf{u} \cdot \nabla\zeta)\right] \\
&= \frac{1}{\mathrm{Pr}}\left[\zeta_t - \psi_{x_3}\zeta_{x_1} + \psi_{x_1}\zeta_{x_3}\right].
\end{aligned}$$

المنارة للاستشارات
www.manaraa.com

Thus we examine the following data assimilation problem (compare this to (3.3)–(3.4)):

$$\begin{gathered} \frac{1}{\mathrm{Pr}}\left[\zeta_t - \psi_{x_3}\zeta_{x_1} + \psi_{x_1}\zeta_{x_3}\right] - \Delta\zeta = \mathrm{Ra}T_{x_1}, \\ \Delta\psi = \zeta, \\ \partial_t T - \psi_{x_3}T_{x_1} + \psi_{x_1}T_{x_3} = \Delta T, \end{gathered} \qquad \begin{gathered} \partial_{x_i}\psi|_{x_3=0} = \partial_{x_i}\psi|_{x_3=1} = 0,\ i = 1,3, \\ T|_{x_3=0} = 1,\ T|_{x_3=1} = 0, \\ \psi_{x_1}, \psi_{x_3}, T \text{ are periodic in } x_1, \\ T(\mathbf{x},0) = T_0(\mathbf{x}),\ \zeta(\mathbf{x},0) = \zeta_0(\mathbf{x}), \end{gathered} \tag{4.5}$$

$$\begin{gathered} \frac{1}{\mathrm{Pr}}\left[\hat{\zeta}_t - \hat{\psi}_{x_3}\hat{\zeta}_{x_1} + \hat{\psi}_{x_1}\hat{\zeta}_{x_3}\right] - \Delta\hat{\zeta} = \mathrm{Ra}\hat{T}_{x_1}, \\ \Delta\hat{\psi} = \hat{\zeta}, \\ \partial_t \hat{T} - \hat{\psi}_{x_3}\hat{T}_{x_1} + \hat{\psi}_{x_1}\hat{T}_{x_3} = \Delta\hat{T} - \mu P_N(\hat{T} - T), \end{gathered} \qquad \begin{gathered} \partial_{x_i}\hat{\psi}|_{x_3=0} = \partial_{x_i}\hat{\psi}|_{x_3=1} = 0,\ i = 1,3, \\ \hat{T}|_{x_3=0} = 1,\ \hat{T}|_{x_3=1} = 0, \\ \hat{\psi}_{x_1}, \hat{\psi}_{x_3}, \hat{T} \text{ are periodic in } x_1, \\ \hat{T}(\mathbf{x},0) = \hat{T}_0(\mathbf{x}),\ \zeta(\mathbf{x},0) = \zeta_0(\mathbf{x}). \end{gathered} \tag{4.6}$$

The code for (4.5)–(4.6) is the same as for (3.3)–(3.4), but with an adjustment for the momentum equations.

```
# Add the Prandtl number as a problem parameter (before defining equations).
Prandtl = 100
problem.parameters['Pr'] = Prandtl                  # Prandtl number

# ...

# Pr(Ra T_x + laplace(zeta)) - zeta_t = u.grad(zeta)
problem.add_equation("Pr*(Ra*dx(T)  + dx(dx(zeta))  + dz(zetaz))  - dt(zeta)"
                     " = v*dx(zeta)    + w*zetaz")
problem.add_equation("Pr*(Ra*dx(T_) + dx(dx(zeta_)) + dz(zetaz_)) - dt(zeta_)"
                     " = v_*dx(zeta_) + w_*zetaz_")
```

The numerical results are similar to those presented in Section 3.3, but now we must also consider the additional parameter Pr. First, we check that synchronization still occurs for reasonable choices of Ra, μ, and N given a finite Pr, say $\mathrm{Pr} = 100$. See Figure 4.1.

المنارة للاستشارات
www.manaraa.com

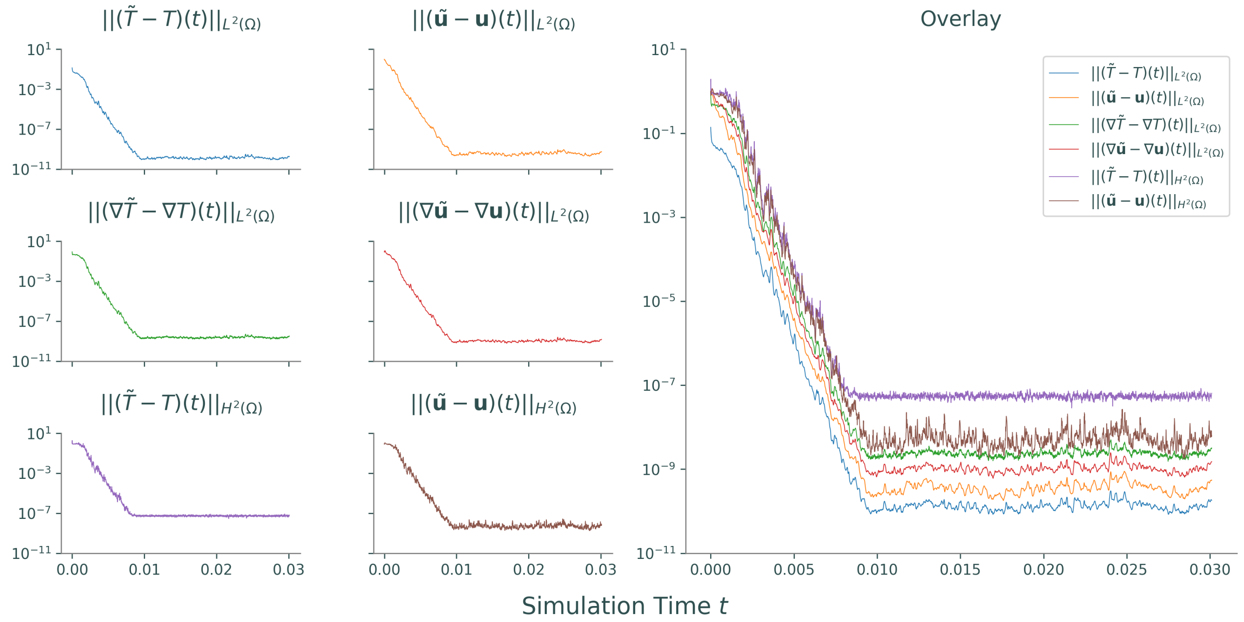

Figure 4.1: Synchronization in various norms for Ra $\approx 5.22 \times 10^7$, $\mu = 18,000$, $N = 32$, and Pr $= 100$. The temperature and velocity differences still decrease exponentially to zero, despite the fact that Pr $< \infty$. However, the convergence is slow compared to the Pr $= \infty$ case (see Figure 4.2).

To simplify our exploration, for each value of Ra listed in Tables 3.1 and 3.2, we pick μ so that the systems synchronize quickly. Then, with $N = 32$, we run simulations for logarithmically spaced values of Pr from 1 to 100. Even with these larger-than-necessary choices of μ, convergence is lost for small enough Pr. See Figure 4.2 for a few additional examples and Table 4.1 for the chosen μ and the lowest Pr where synchronization still occurs.

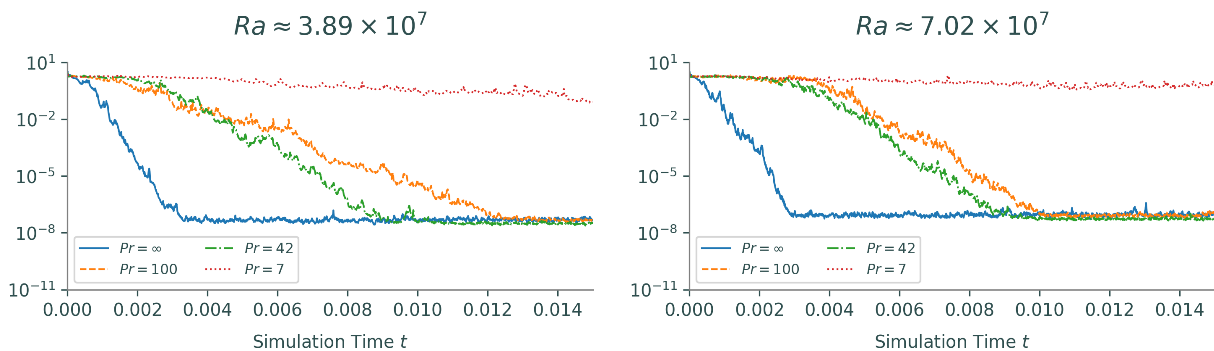

Figure 4.2: $\|(\hat{\mathbf{u}}-\mathbf{u})(t)\|_{H^2(\Omega)}+\|(\hat{T}-T)(t)\|_{H^2(\Omega)}$ with $N = 32$ and various Pr for two different values of Ra, with μ as given in Table 4.1. The convergence is generally slower the lower that Pr gets, until eventually there is no synchronization if Pr is too small.

المنارة للاستشارات
www.manaraa.com

Ra	μ	Pr
1.1937766×10^7	$10,000$	75
1.6037187×10^7	$11,000$	56
2.1544346×10^7	$12,000$	56
2.8942661×10^7	$13,000$	100
3.8881551×10^7	$14,000$	42
5.2233450×10^7	$18,000$	42
7.0170382×10^7	$20,000$	42
9.4266845×10^7	$25,000$	31
1.26638017×10^8	$32,000$	31
1.70125427×10^8	$40,000$	31

Table 4.1: Minimal values of Pr that result in synchronization within about 0.01 units of simulation time for the given Ra and μ with $N = 32$. The relationship here is much less precise than those described in Tables 3.1 and 3.2 (for example, Ra $\approx 2.89 \times 10^7$ is a bit of an outlier), but it is interesting to note that lower Ra seem to require larger Pr. Whether or not this is an effect of Ra increasing or μ increasing, however, is unclear.

There are two main points to take away from these results.

- Finite-but-large Prandtl data assimilation through only temperature measurements is possible, though it is slow compared to the infinite Prandtl version in Chapters 2–3.
- The relationship between Ra, μ, N, and Pr remains unclear and would require further measurements to precisely quantify. However, the fact that the assimilation works at all with reasonably small Pr indicates that the inequality constraints in Theorem 4.1 are strongly overstated, or the constants in (4.4) are extremely small.

4.2 Hybrid Prandtl Assimilation

We conclude by briefly mentioning a data assimilation scheme where, unlike the schemes we have considered so far, the momentum equations are **not** identical for the data and assimilating variables: we use (4.1) as the truth and (2.2) for the nudging equations. That is, we consider the problem

المنارة للاستشارات
www.manaraa.com

$$
\begin{aligned}
\tfrac{1}{\mathrm{Pr}}\left[\partial_t \mathbf{u} + (\mathbf{u}\cdot\nabla)\mathbf{u}\right] - \Delta\mathbf{u} &= -\nabla p + \mathrm{Ra}\mathbf{e}_3 T, \\
\nabla\cdot\mathbf{u} &= 0, \\
\partial_t T + \mathbf{u}\cdot\nabla T &= \Delta T,
\end{aligned}
\qquad
\begin{aligned}
&\mathbf{u}|_{x_3=0} = \mathbf{u}|_{x_3=1} = \mathbf{0}, \\
&T|_{x_3=0} = 1,\ T|_{x_3=1} = 0, \\
&\mathbf{u}, T \text{ are periodic in } x_1 \text{ and } x_2, \\
&T(\mathbf{x},0) = T_0(\mathbf{x}),\ \mathbf{u}(\mathbf{x},0) = \mathbf{u}_0(\mathbf{x}),
\end{aligned}
\tag{4.7}
$$

$$
\begin{aligned}
-\Delta\tilde{\mathbf{u}} &= -\nabla\tilde{p} + \mathrm{Ra}\mathbf{e}_3\tilde{T}, \\
\nabla\cdot\tilde{\mathbf{u}} &= 0, \\
\partial_t\tilde{T} + \tilde{\mathbf{u}}\cdot\nabla\tilde{T} &= \Delta\tilde{T} - \mu P_N(\tilde{T} - T),
\end{aligned}
\qquad
\begin{aligned}
&\tilde{\mathbf{u}}|_{x_3=0} = \tilde{\mathbf{u}}|_{x_3=1} = \mathbf{0}, \\
&\tilde{T}|_{x_3=0} = 1,\ \tilde{T}|_{x_3=1} = 0, \\
&\tilde{\mathbf{u}}, \tilde{T} \text{ are periodic in } x_1 \text{ and } x_2, \\
&\tilde{T}(\mathbf{x},0) = \tilde{T}_0(\mathbf{x}).
\end{aligned}
\tag{4.8}
$$

This system is more realistic in the sense that $\mathrm{Pr} < \infty$ for actual mantle fluids (the "real-world data"), but it also takes advantage of the more computationally tractable $\mathrm{Pr} = \infty$ situation for the assimilation. The analysis is similar to that of (4.1)–(4.2); here we state but do not prove the analog of 4.1.

Theorem 4.2. *Let* $(\mathbf{u}, T)$, $(\tilde{\mathbf{u}}, \tilde{T})$ *be a regular solution of* (4.7)–(4.8) *with initial conditions* $(\mathbf{u}_0, T_0) \in \mathcal{B}_V(\rho_1) \times \mathcal{B}_{H^1(\Omega)}(\rho_2)$ *and* $\tilde{T}_0 \in \mathcal{B}_{H^1(\Omega)}(\rho_2)$. *There exists an absolute constant* $C_0 > 0$ *such that if*

$$
\frac{\lambda_N}{2} \geq \mu \quad \text{and} \quad \mu \geq \frac{4}{3} C_0^2 \left(\rho_2 + 1\right)^2 Ra^2, \tag{4.9}
$$

then for any given fixed time interval $[0, t]$,

$$
Ra^{-1}\|\tilde{\mathbf{u}}(\tau) - \mathbf{u}(\tau)\|_{H^2(\Omega)} + \|\tilde{T}(\tau) - T(\tau)\|^2_{L^2(\Omega)} \leq O(e^{-\mu t}) + O(Pr^{-1}), \quad 0 \leq \tau \leq t. \tag{4.10}
$$

4.2.1 Numerical Results. The hybridization of the data assimilation problem is clever, but the numerical results are disappointing. Take, for example, a simulation with $\mathrm{Ra} \approx$

المنارة للاستشارات
www.manaraa.com

5.22×10^7, $\mu = 18,000$ $N = 32$, and $\Pr = 100$. For the finite Prandtl model in Section 4.1, this was completely successful (see Figure 4.1). In this situation, however, this set of parameters fails spectacularly.

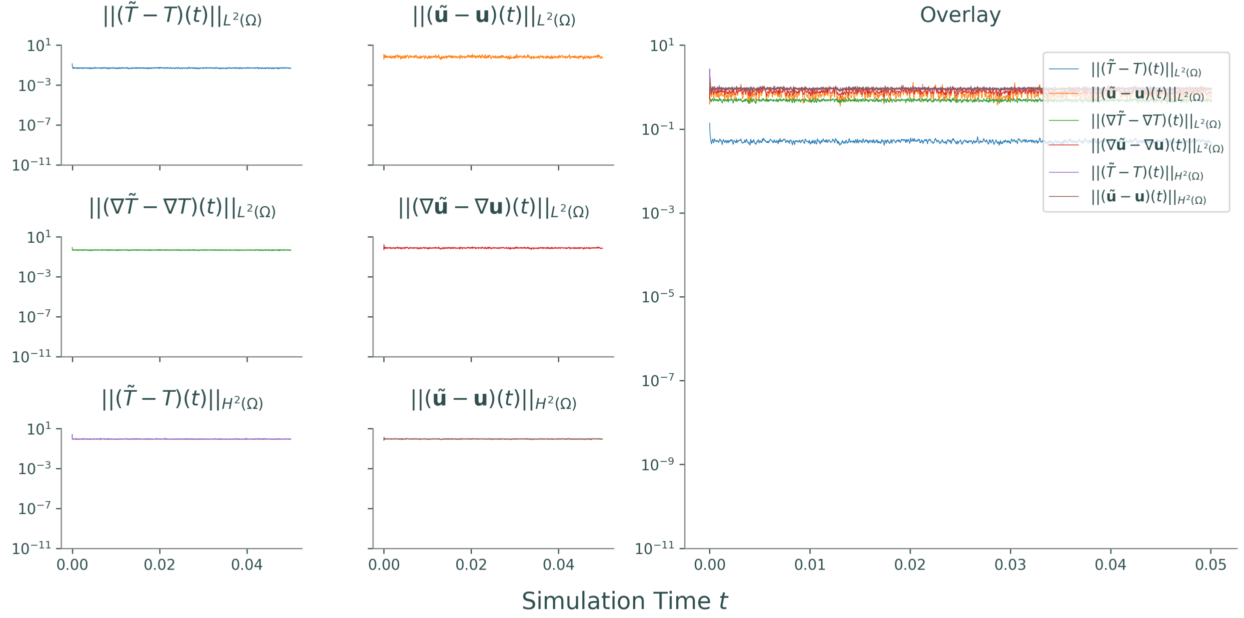

Figure 4.3: Hybrid assimilation with $\text{Ra} \approx 5.22 \times 10^7$, $\mu = 18,000$, $N = 32$, and $\Pr = 100$. The temperature and velocity differences remain almost constant, with no hint of convergence.

Figure 4.3 suggests that $\mu = 18,000$ is too small for $\text{Ra} \approx 5.22 \times 10^7$ in this setting. In an attempt to achieve synchronization in an easier setting, we select a lower Rayleigh value of $\text{Ra} \approx 8.89 \times 10^6$ and raise μ drastically. However, the results are also underwhelming.

المنارة للاستشارات
www.manaraa.com

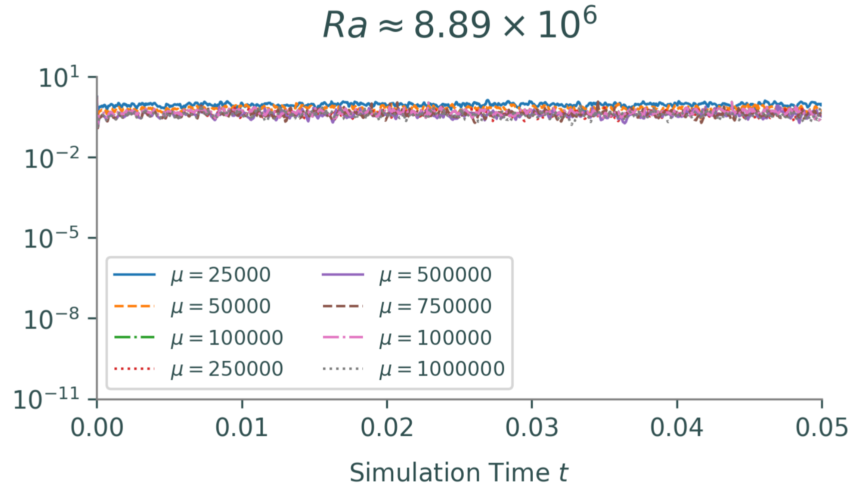

Figure 4.4: Lack of synchronization in hybrid assimilation with $\mathrm{Ra} \approx 8.89 \times 10^6$, $N = 32$, and $\mathrm{Pr} = 100$, with various μ values. Raising μ further results in numerical problems as $\|\mathbf{u}(t)\|_2$ and/or $\|\tilde{\mathbf{u}}(t)\|_2$ blow up in finite time.

This apparent failure is not a contradiction to Theorem 4.2; rather, it implies that the conditions of the theorem require values of μ that are not numerically feasible. In other words, though the analysis demonstrates the theoretical possibility of choosing μ so large that synchronization occurs, the numerical failure of the hybrid system suggests that the practical success of data assimilation is dependent on the data coming from the same model as the simulated system, an unrealistically stringent restriction. Recall that the Boussinesq approximation is effectively a "zeroth order" approximation for the mantle, meaning the true physical system has several complicated secondary effects (some of which are unknown) that are not included in the model. This numerical experimental failure in such a simple setting suggests that data assimilation in general is not adaptable to settings where the exact model is not known.

المنارة للاستشارات
www.manaraa.com

5.1 Summary

In Chapter 2, we examined a data assimilation scheme for the Boussinesq system with $\mathrm{Pr} = \infty$ and showed rigorously that synchronization occurs between the data and assimilating equations under certain conditions on the relaxation parameter μ and the number of projected modes N relative to the Rayleigh number Ra. That is, as long as there is enough data (i.e., N is not too small), μ can be chosen large enough to guarantee synchronization. Though this is a satisfying theoretical result, the numerical experiments in Chapter 3 show that synchronization often occurs under much weaker conditions on μ and N than Theorem 2.6 requires. In addition, the numerical results also demonstrate situations in which the assimilation fails, namely when μ and/or N are not large enough.

In Chapter 4, we examined two similar Boussinesq data assimilation problems where Pr is large, but finite. When both the data and assimilating equations use $\mathrm{Pr} < \infty$, the theoretical and numerical results are comparable to those of Chapters 2 and 3, though the synchronization occurs more slowly and the relationship between Ra, Pr, μ, and N remains somewhat ambiguous. On the other hand, when the data equations use $\mathrm{Pr} < \infty$ but the assimilating equations use $\mathrm{Pr} = \infty$, the numerical results fail to converge even though the corresponding rigorous result provides theoretical conditions for synchronization. In short, the numerical experiments show that the requirements in Theorems 2.6, 4.1, and 4.2 are not sharp, and suggest that structural similarity in the truth and nudging equations is the key factor that allows the data assimilation process to be successful and computationally feasible.

5.2 Extensions

We conclude by discussing a few possible ways to extend the work presented in this thesis.

المنارة للاستشارات
www.manaraa.com

5.2.1 Sampling Strategies. From an industrial perspective, one of the main problems with the data assimilation schemes presented in Chapters 2–4 is the projection operator P_N, which allows the assimilating system to "see" the data T everywhere in the domain, but with low precision. A more likely scenario is that T is perfectly observable at part of the domain and not at all in the rest. In a weather model, for example, data can be gathered with high accuracy, but only at weather stations with the proper equipment. The notion of complete observables over only part of the domain is also valid for mantle convection, since volcanic activity can result in somewhat accurate readings for parts of the mantle [25]. Such an observation scheme could be implemented in Dedalus with some careful array masking.

5.2.2 Plume Identification. Theorem 2.6 and its analogs prove synchronization in a limiting sense, meaning T and $\tilde{T}$ match exactly in the limit as $t \to \infty$. Having the data and assimilating variable match completely is desirable, but from a geologic and fluid-dynamical perspective, the most important features to capture in the assimilation are the *coherent structures* (the hot and cold plumes) that arise in this setting. Once the assimilating system is sufficiently close to the data, it can be used to statistically analyze these structures. For example, a clustering algorithm such as the k-means algorithm could identify the plumes based on the assimilated temperature and fluid velocity. Successful synchronization may then be defined as convergence of plume statistics.

5.2.3 Stochastic Data. Using a Boussinesq system in the place of real-world data is the only practical way to currently approach this problem. However, in order to reflect the noise present in nature and the error from experimental measurements, the truth equations in each data assimilation scheme could be supplemented with a stochastic term (a random variable).

المنارة للاستشارات
www.manaraa.com

Appendix A. Notation

Most of the notation presented here is standard in vector calculus and applied mathematics [16]. However, some familiar-looking notation in the Boussinesq equations (1.8)–(1.10) have non-familiar meanings, and some definitions have different interpretations depending on the dimension of the underlying space. While many of the notions here can be generalized to arbitrary dimensions, we detail only the relevant 3- and 2-dimensional versions.

Let $\Omega \subset \mathbb{R}^3$ and $\Psi \subset \mathbb{R}^2$ be spatial domains with boundaries $\partial\Omega$ and $\partial\Psi$, respectively, and let $\mathbb{T} = [0, \infty)$ be the generic time domain. Spatial coordinates are denoted as the vectors $\mathbf{x} = [x_1, x_2, x_3]^\mathsf{T} \in \Omega$ or $\mathbf{x} = [x_1, x_2]^\mathsf{T} \in \Psi$, and time is denoted $t \in \mathbb{T}$. The standard basis vectors in $\mathbb{R}^3$ are written $\mathbf{e}_1 = [1, 0, 0]^\mathsf{T}$, $\mathbf{e}_2 = [0, 1, 0]^\mathsf{T}$, and $\mathbf{e}_3 = [0, 0, 1]^\mathsf{T}$.

For this appendix, bold capital letters denote 3-dimensional vector fields, bold lower-case letters denote 2-dimensional vector fields, and non-bold lower case letters denote scalar-valued functions. For example, $\mathbf{F} : \Omega \times \mathbb{T} \to \mathbb{R}^3$ is a 3-dimensional vector field $\mathbf{F} = [f_1, f_2, f_3]^\mathsf{T}$ with smooth[1] components $f_i : \Omega \times \mathbb{T} \to \mathbb{R}$ for $i = 1, 2, 3$; likewise, $\mathbf{f} : \Psi \times \mathbb{T} \to \mathbb{R}^2$ is a 2-dimensional vector field $\mathbf{f} = [f_1, f_2]^\mathsf{T}$ with smooth components $f_i : \Psi \times \mathbb{T} \to \mathbb{R}$ for $i = 1, 2$; and $f : \Omega \times \mathbb{T} \to \mathbb{R}$ is a smooth scalar-valued function.

[1] In this context, *smooth* typically means infinitely continuously differentiable, but we also use the term loosely to mean "as differentiable as needed."

المنارة للاستشارات
www.manaraa.com

A.1 Vector Products

A.1.1 Dot Product. The usual Euclidean inner product of two vectors $\mathbf{x}$ and $\mathbf{y}$ of the same length is denoted $\mathbf{x} \cdot \mathbf{y}$ and is the sum of the component-wise product,

$$\mathbf{x} \cdot \mathbf{y} = [x_1, x_2, x_3]^{\mathsf{T}} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \sum_{i=1}^{3} x_i y_i.$$

Beware that the dot $\cdot$ is also used in divergence $\nabla \cdot \mathbf{F}$ (A.2) and the advection operator $\mathbf{F} \cdot \nabla$ (A.10), a slight, but common abuse of notation.

A.1.2 Cross Product. The *cross product* of two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^3$ is defined as

$$\mathbf{x} \times \mathbf{y} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \times \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{vmatrix} \hat{\mathbf{i}} & \hat{\mathbf{j}} & \hat{\mathbf{k}} \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{vmatrix} = \begin{bmatrix} x_2 y_3 - x_3 y_2 \\ x_3 y_1 - x_1 y_3 \\ x_1 y_2 - x_2 y_1 \end{bmatrix}.$$

The somewhat odd determinant notation in the previous equation is standard. In the special case that $\mathbf{x}$ and $\mathbf{y}$ are "two-dimensional" in the sense that $x_3 = 0 = y_3$, we have

$$\mathbf{x} \times \mathbf{y} = \begin{bmatrix} x_1 \\ x_2 \\ 0 \end{bmatrix} \times \begin{bmatrix} y_1 \\ y_2 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ x_1 y_2 - x_2 y_1 \end{bmatrix}.$$

With this motivation, we define the *2-dimensional cross product* **as a scalar**:

$$\mathbf{x} \times \mathbf{y} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \times \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix} = x_1 y_2 - x_2 y_1. \tag{A.1}$$

المنارة للاستشارات
www.manaraa.com

A.2 Derivatives

Partial derivatives are notated by

$$\frac{\partial f}{\partial s} = \partial_s f = f_s.$$

Which style is used depends purely on which is clearest in context. Here s is an independent (dummy) variable representing time or a spatial dimension.

A.2.1 Gradients.

Scalar Gradient. The *gradient* of a scalar-valued function is a vector-valued function where the ith component is the ith spatial derivative of the original function:

$$\nabla f = \begin{bmatrix} \partial_{x_1} f \\ \partial_{x_2} f \\ \partial_{x_3} f \end{bmatrix}.$$

We consider ∇f a column vector. Note that the gradient only takes derivatives in spatial dimensions ($\partial_t f$ is ignored).

Gradient Tensor. The *vector gradient* or *gradient tensor* of a vector field is the transpose of its Jacobian matrix:

$$\nabla \mathbf{F} = \begin{bmatrix} \partial_{x_1} f_1 & \partial_{x_1} f_2 & \partial_{x_1} f_3 \\ \partial_{x_2} f_1 & \partial_{x_2} f_2 & \partial_{x_2} f_3 \\ \partial_{x_3} f_1 & \partial_{x_3} f_2 & \partial_{x_3} f_3 \end{bmatrix} = \left[\begin{array}{c|c|c} \nabla f_1 & \nabla f_2 & \nabla f_3 \end{array} \right].$$

As with the scalar gradient, the time dimension is ignored.

المنارة للاستشارات
www.manaraa.com

A.2.2 Divergence. The *divergence operator* transforms a vector field into a scalar-valued function by summing the ith partial derivative of each ith component:

$$\nabla \cdot \mathbf{F} = \operatorname{div}(\mathbf{F}) = \sum_{i=1}^{3} \partial_{x_i} f_i. \tag{A.2}$$

A.2.3 Curl. The *curl* of a 3-dimensional vector field is defined as

$$\nabla \times \mathbf{F} = \nabla \times \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} = \begin{vmatrix} \hat{\mathbf{i}} & \hat{\mathbf{j}} & \hat{\mathbf{k}} \\ \partial_{x_1} & \partial_{x_2} & \partial_{x_3} \\ f_1 & f_2 & f_3 \end{vmatrix} = \begin{bmatrix} \partial_{x_2} f_3 - \partial_{x_3} f_2 \\ \partial_{x_3} f_1 - \partial_{x_1} f_3 \\ \partial_{x_1} f_2 - \partial_{x_2} f_1 \end{bmatrix}.$$

As with the definition of the cross product, the determinant notation is standard.

If $\mathbf{F}$ is 2-dimensional in the sense that $f_3 = 0$ and $\partial_{x_3} f_1 = 0 = \partial_{x_3} f_2$, then we have

$$\nabla \times \mathbf{F} = \begin{bmatrix} 0 \\ 0 \\ \partial_{x_1} f_2 - \partial_{x_2} f_1 \end{bmatrix}.$$

We therefore define the *curl for a 2-dimensional vector field* **as a scalar-valued function**:

$$\nabla \times \mathbf{f} = \nabla \times \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = \begin{vmatrix} \partial_{x_1} & \partial_{x_2} \\ f_1 & f_2 \end{vmatrix} = \partial_{x_1} f_2 - \partial_{x_2} f_1. \tag{A.3}$$

Compare (A.3) with (A.1).

A.2.4 Curl Identities. The following are well-known identities for the regular 3-dimensional curl, which we cite without proof.

المنارة للاستشارات

www.manaraa.com

Nullity. The curl of any gradient is the zero vector. Conversely, the divergence of any curl is zero. That is,

$$\nabla \times (\nabla f) = \mathbf{0}, \tag{A.4}$$

$$\nabla \cdot (\nabla \times \mathbf{F}) = 0. \tag{A.5}$$

Distributivity. The curl operator distributes over vector addition:

$$\nabla \times (\mathbf{F} + \mathbf{G}) = (\nabla \times \mathbf{F}) + (\nabla \times \mathbf{G}). \tag{A.6}$$

Linearity. The curl operator is linear with respect to vector addition and multiplication by a scalar. That is, for any $\alpha, \beta \in \mathbb{R}$,

$$\nabla \times (\alpha\mathbf{F} + \beta\mathbf{G}) = \alpha(\nabla \times \mathbf{F}) + \beta(\nabla \times \mathbf{G}). \tag{A.7}$$

Commutativity. The curl operator commutes with partial derivatives, i.e.,

$$\partial_s(\nabla \times \mathbf{F}) = \nabla \times (\partial_s \mathbf{F}). \tag{A.8}$$

Product Rule. The curl operator follows a product rule involving the gradient:

$$\nabla \times (g\mathbf{F}) = g(\nabla \times \mathbf{F}) + (\nabla g) \times \mathbf{F}. \tag{A.9}$$

That (A.7) and (A.8) hold for the 2-dimensional curl defined in (A.3) is obvious, and (A.5) no longer makes sense in two dimensions, but we verify directly that the other properties listed above hold as well. The key difference is that each of the vector equations listed above are now scalar equations.

www.manaraa.com

Nullity. Compare to (A.4). In this case, the curl of any gradient is the **scalar** 0:

$$\nabla \times (\nabla f) = \partial_{x_2}(\partial x_1 f) - \partial_{x_1}(\partial x_2 f) = 0.$$

Distributivity. Compare to (A.6). We have

$$\begin{aligned}\nabla \times (\mathbf{f} + \mathbf{g}) &= \partial_{x_1}(f_2 + g_2) - \partial_{x_2}(f_1 + g_1) \\ &= (\partial_{x_1} f_2 - \partial_{x_2} f_1) + (\partial_{x_1} g_2 - \partial_{x_2} g_1) = (\nabla \times \mathbf{f}) + (\nabla \times \mathbf{g})\end{aligned}$$

Product Rule. Compare to (A.9). We have

$$\begin{aligned}\nabla \times (g\mathbf{f}) &= \partial_{x_1}(g f_2) - \partial_{x_2}(g f_1) \\ &= (\partial_{x_1} g) f_2 + g(\partial_{x_1} f_2) - (\partial_{x_2} g) f_1 - g(\partial_{x_2} f_1) \\ &= g(\partial_{x_1} f_2 - \partial_{x_2} f_1) + ((\partial_{x_1} g) f_2 - (\partial_{x_2} g) f_1) = g(\nabla \times \mathbf{f}) + (\nabla g) \times \mathbf{f}\end{aligned}$$

A.2.5 Laplacians.

Scalar Laplacian. The *Laplacian* of a scalar-valued function is the divergence of its gradient,

$$\Delta f = \nabla^2 f = \nabla \cdot (\nabla f) = \sum_{i=1}^{n} \partial_{x_i x_i} f.$$

We use the Δf notation in this thesis. Note that Δf is a scalar-valued function.

Vector Laplacian. The *vector Laplacian* or *tensor Laplacian* of a vector field is defined by the equation

$$\Delta \mathbf{F} = \nabla(\nabla \cdot \mathbf{F}) - \nabla \times (\nabla \times \mathbf{F}).$$

www.manaraa.com

In Cartesian coordinates, this simplifies to

$$\Delta \mathbf{F} = \begin{bmatrix} \Delta f_1 \\ \Delta f_2 \\ \Delta f_3 \end{bmatrix}.$$

Note that $\Delta \mathbf{F}$ is a vector-valued function.

Due to (A.7) and (A.8), the curl operator commutes with the Laplacian operator:

$$\nabla \times (\Delta \mathbf{F}) = \Delta(\nabla \times \mathbf{F}).$$

A.2.6 Advection Operator. The *advection operator* with respect to a vector field $\mathbf{F}$ is defined as

$$\mathbf{F} \cdot \nabla = \sum_{i=1}^{3} f_i \partial_{x_i}. \tag{A.10}$$

The operator may act on scalar functions and/or vector fields. In the case of a scalar function, the notation is consistent with the usual gradient and dot operators:

$$(\mathbf{F} \cdot \nabla) g = \sum_{i=1}^{3} f_i \partial_{x_i} g = [f_1, f_2, f_3]^{\mathsf{T}} \begin{bmatrix} g_{x_1} \\ g_{x_2} \\ g_{x_3} \end{bmatrix} = \mathbf{F} \cdot (\nabla g).$$

The definition is motivated by the fluid velocity vector $\mathbf{u}$: Let $\mathbf{x}(t) \in \Omega$ be a fixed fluid particle at time $t \in \mathbb{T}$ so that $\partial_t x_i = u_i$ for $i = 1, 2, 3$. Then for any vector field $\mathbf{G} = \mathbf{G}(\mathbf{x}(t), t)$,

$$\frac{d}{dt} \mathbf{G}(\mathbf{x}(t), t) = \partial_t \mathbf{G} + \sum_{i=1}^{3} \partial_t x_i \partial_{x_i} \mathbf{G} = \partial_t \mathbf{G} + \sum_{i=1}^{3} u_i \partial_{x_i} \mathbf{G} = \partial_t \mathbf{G} + (\mathbf{u} \cdot \nabla) \mathbf{G}.$$

The operator $\frac{D}{Dt} = \partial_t + \mathbf{u} \cdot \nabla$ is sometimes called the *material derivative* [9, 8].

المنارة للاستشارات
www.manaraa.com

Note that the vector Laplacian is the divergence of the vector gradient, i.e.,

$$\nabla \cdot (\nabla \mathbf{F}) = (\nabla \cdot \nabla)\mathbf{F} = \sum_{i=1}^{3} \partial_{x_i}\partial_{x_i}\mathbf{F} = \Delta \mathbf{F}.$$

The advection operator is linear in that the mapping $\mathbf{u} \mapsto (\mathbf{F} \cdot \nabla)\mathbf{u}$ is linear, and the mapping $\mathbf{F} \mapsto \mathbf{F} \cdot \nabla$ is linear:

$$\begin{aligned}
(\mathbf{F} \cdot \nabla)(\alpha \mathbf{u} + \beta \mathbf{v}) &= \sum_{i=1}^{n} f_i (\alpha \mathbf{u} + \beta \mathbf{v})_{x_i} \\
&= \alpha \sum_{i=1}^{n} f_i \mathbf{u}_{x_i} + \beta \sum_{i=1}^{n} f_i \mathbf{v}_{x_i} = \alpha (\mathbf{F} \cdot \nabla)\mathbf{u} + \beta (\mathbf{F} \cdot \nabla)\mathbf{v}, \\
((\alpha \mathbf{F} + \beta \mathbf{G}) \cdot \nabla)\mathbf{u} &= \sum_{i=1}^{n} (\alpha f_i + \beta g_i)\mathbf{u}_{x_i} \\
&= \alpha \sum_{i=1}^{n} f_i \mathbf{u}_{x_i} + \beta \sum_{i=1}^{n} g_i \mathbf{u}_{x_i} = \alpha (\mathbf{F} \cdot \nabla)\mathbf{u} + \beta (\mathbf{G} \cdot \nabla)\mathbf{u}.
\end{aligned}$$

Lastly, the advection operator commutes with the curl operator under certain conditions in two dimensions. We state this more precisely as a lemma.

Lemma A.1. *Let* $\mathbf{u} : \mathbb{R}^2 \times \mathbb{T} \to \mathbb{R}^2$ *be smooth with* $\mathbf{u} = [u^1, u^2]^\mathsf{T}$. *If* $u^1_{x_1} = -u^2_{x_2}$, *then* $\nabla \times ((\mathbf{u} \cdot \nabla)\mathbf{u}) = (\mathbf{u} \cdot \nabla)(\nabla \times \mathbf{u})$.

Proof. To begin, compute

$$\begin{aligned}
(\nabla u^1) \times \mathbf{u}_{x_1} + (\nabla u^2) \times \mathbf{u}_{x_2} &= \begin{bmatrix} u^1_{x_1} \\ u^1_{x_2} \end{bmatrix} \times \begin{bmatrix} u^1_{x_1} \\ u^2_{x_1} \end{bmatrix} + \begin{bmatrix} u^2_{x_1} \\ u^2_{x_2} \end{bmatrix} \times \begin{bmatrix} u^1_{x_2} \\ u^2_{x_2} \end{bmatrix} \\
&= u^1_{x_1} u^2_{x_1} - u^1_{x_1} u^1_{x_2} + u^2_{x_1} u^2_{x_2} - u^1_{x_2} u^2_{x_2} \\
&= -u^2_{x_2} u^2_{x_1} + u^2_{x_2} u^1_{x_2} + u^2_{x_1} u^2_{x_2} - u^1_{x_2} u^2_{x_2} = 0.
\end{aligned}$$

Note that these computations use the two-dimensional curl definition in (A.3). This identity,

المنارة للاستشارات
www.manaraa.com

together with (A.6), (A.9), and (A.8), yields

$$\begin{aligned}
\nabla \times (\mathbf{u} \cdot \nabla)\mathbf{u} &= \nabla \times (u^1 \mathbf{u}_{x_1} + u^2 \mathbf{u}_{x_2}) \\
&= \nabla \times (u^1 \mathbf{u}_{x_1}) + \nabla \times (u^2 \mathbf{u}_{x_2}) \\
&= u^1(\nabla \times \mathbf{u}_{x_1}) + (\nabla u^1) \times \mathbf{u}_{x_1} + u^2(\nabla \times \mathbf{u}_{x_2}) + (\nabla u^2) \times \mathbf{u}_{x_2} \\
&= u^1 \partial_{x_1}(\nabla \times \mathbf{u}) + u^2 \partial_{x_2}(\nabla \times \mathbf{u}) + \left[(\nabla u^1) \times \mathbf{u}_{x_1} + (\nabla u^2) \times \mathbf{u}_{x_2}\right] \\
&= (\mathbf{u} \cdot \nabla)(\nabla \times \mathbf{u}),
\end{aligned}$$

which is the desired result. □

A.3 Spaces and Norms

In this section, we largely follow the notation of [12].

Let $0 < p < \infty$. For a point $\mathbf{x} = [x_1, x_2, \ldots, x_n]^\mathsf{T} \in \mathbb{R}^n$, the *p-norm* of $\mathbf{x}$ is

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}, \tag{A.11}$$

and the ∞-*norm* (sometimes called the "sup norm") is $\|\mathbf{x}\|_\infty = \max_{i \in \{1,\ldots,n\}} |x_i|$. Given another point $\mathbf{y} = [y_1, y_2, \ldots, y_n]^\mathsf{T} \in \mathbb{R}^n$, the Euclidean *inner product* of $\mathbf{x}$ and $\mathbf{y}$ is given by

$$\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^{n} x_i y_i.$$

We also make frequent use of the following function norms and the corresponding spaces.

A.3.1 L^p Spaces. Let $f : \Omega \to \mathbb{R}$ and $0 < p < \infty$. The L^p *norm* of f over Ω is

$$\|f\|_{L^p(\Omega)} = \left(\int_\Omega f(\mathbf{x})^p \, d\mathbf{x} \right)^{1/p},$$

المنارة للاستشارات
www.manaraa.com

and the space $L^p(\Omega)$ is the set of all functions over Ω with finite L^p norm,

$$L^p(\Omega) = \left\{ f : \Omega \to \mathbb{R} \ \mid \ \|f\|_{L^p(\Omega)} < \infty \right\}.$$

The L^∞ *norm* ($p = \infty$) of f over Ω is

$$\|f\|_{L^\infty(\Omega)} = \operatorname*{ess\,sup}_{\mathbf{x}\in\Omega} |f(\mathbf{x})| = \inf \{ k \in \mathbb{R} \ \mid \ \mu(\{\mathbf{x} \in \Omega \ \mid \ |f(\mathbf{x})| > k\}) = 0 \},$$

where μ is the usual Lebesgue measure. As with the $0 < p < \infty$ case,

$$L^\infty(\Omega) = \left\{ f : \Omega \to \mathbb{R} \ \mid \ \|f\|_{L^\infty(\Omega)} < \infty \right\}.$$

For $p = 1$, we also define the space of *locally integrable* functions on Ω as

$$L^1_{loc}(\Omega) = \left\{ f : \Omega \to \mathbb{R} \ \mid \ f \in L^1(K) \text{ for all compact subsets } K \subset \Omega \right\}.$$

For a function $\mathbf{f} : \Omega \to \mathbb{R}^n$, we define the norms

$$\|\mathbf{f}\|_{L^p(\Omega)^n} = \left(\int_\Omega \|\mathbf{f}(\mathbf{x})\|_p^p \, d\mathbf{x} \right)^{1/p}, \qquad \|\mathbf{f}\|_{L^\infty(\Omega)^n} = \operatorname*{ess\,sup}_{\mathbf{x}\in\Omega} \|\mathbf{f}(\mathbf{x})\|_\infty,$$

where the norms within the integrand are the usual p norms for vectors given in (A.11). For $0 < p \le \infty$, the corresponding space is

$$L^p(\Omega)^n = \left\{ \mathbf{f} : \Omega \to \mathbb{R}^n \ \mid \ \|\mathbf{f}\|_{L^p(\Omega)^n} < \infty \right\} = \{ \mathbf{f} : \Omega \to \mathbb{R}^n \ \mid \ f_i \in L^p(\Omega), \ i = 1, 2, \ldots, n \}.$$

In the special case that $p = 2$, we associate $L^2(\Omega)$ with the inner product

$$\langle f, g \rangle_{L^2(\Omega)} = \int_\Omega f(\mathbf{x}) g(\mathbf{x}) \, d\mathbf{x}$$

المنارة للاستشارات
www.manaraa.com

for $f, g \in L^2(\Omega)$, and

$$\langle \mathbf{f}, \mathbf{g} \rangle_{L^2(\Omega)^n} = \int_\Omega \langle \mathbf{f}(\mathbf{x}), \mathbf{g}(\mathbf{x}) \rangle \, d\mathbf{x}$$

for $\mathbf{f}, \mathbf{g} \in L^2(\Omega)^n$. We use $\langle \cdot, \cdot \rangle$ to denote the inner product on $\mathbb{R}^n$, $L^p(\Omega)$, or $L^p(\Omega)^n$, depending on the context.

A.3.2 Sobolev Spaces. While the L^p spaces describe integrable functions, *Sobolev spaces* describe functions that are differentiable in a weak sense. For this section, let $\alpha = (\alpha_1, \alpha_2, \alpha_3)$ where each α_i is a nonnegative integer. We write a general differential operator as

$$D^\alpha = \frac{\partial^{\alpha_1}}{\partial x_1^{\alpha_1}} \frac{\partial^{\alpha_2}}{\partial x_2^{\alpha_2}} \frac{\partial^{\alpha_3}}{\partial x_3^{\alpha_3}}.$$

Let $C^\infty(\Omega)$ be the space of infinitely continuously differentiable functions on Ω and

$$C_c^\infty(\Omega) = \{ f \in C^\infty(\Omega) \mid f \text{ has compact support in } \Omega \}.$$

In other words, $f \in C_c^\infty(\Omega)$ is smooth and vanishes on $\partial\Omega$. An integrable function $f \in L^1_{loc}(\Omega)$ has a *weak derivative* $D_w^\alpha f$ if there exists a function $g \in L^1_{loc}(\Omega)$ such that

$$\int_\Omega g(\mathbf{x}) \phi(\mathbf{x}) \, d\mathbf{x} = (-1)^{|\alpha|} \int_\Omega f(\mathbf{x}) D^\alpha \phi(\mathbf{x}) \, d\mathbf{x}$$

for all $\phi \in C_c^\infty(\Omega)$. If such a g exists, we define $D_w^\alpha f = g$.

The Sobolev space $W_p^k(\Omega)$ is the set of all functions $f : \Omega \to \mathbb{R}$ such that for all $|\alpha| \leq k$, $D_w^\alpha f$ exists and is a member of $L^p(\Omega)$. For $p = 2$, we write $H^k(\Omega) = W_2^k(\Omega)$. The norm on $W_p^k(\Omega)$ is defined as

$$\|f\|_{W_p^k(\Omega)} = \left(\sum_{|\alpha| \leq k} \|D_w^\alpha f\|_{L^p(\Omega)}^p \right)^{1/p}.$$

المنارة للاستشارات
www.manaraa.com

To obtain functions with desirable boundary conditions, we also define

$$H_0^1(\Omega) = \{f \in H^1(\Omega) \ \mid \ f \text{ has compact support in } \Omega\}.$$

As with the L^p case, for vector fields we define the Sobolev space

$$W_p^k(\Omega)^n = \left\{\mathbf{f} : \Omega \to \mathbb{R}^n \ \mid \ f_i \in W_p^k(\Omega),\ i = 1, 2, \ldots, n\right\}.$$

Spaces Involving Time. Let $f : \Omega \times \mathbb{T} \to \mathbb{R}$. We say $f \in L^p(\Omega)$ if, for each $t \in \mathbb{T}$, the map $\mathbf{x} \mapsto f(\mathbf{x}, t)$ is in $L^p(\Omega)$. In this case the L^p norm of f is written $\|f(t)\|_{L^p(\Omega)}$. Similarly, for $\mathbf{f} : \Omega \times \mathbb{T} \to \mathbb{R}^n$, we say $\mathbf{f} \in L^p(\Omega)^n$ if the map $\mathbf{x} \mapsto f(\mathbf{x}, t)$ is in $L^p(\Omega)^n$ for all $t \in \mathbb{T}$. Note that these norms are still spatial in the sense that they involve an integration over Ω; for integration in the time domain, we use the following spaces.

Let X be a Banach space with norm $\|\cdot\|$ (for instance, $X = L^p(\Omega)$ or $X = W_p^k(\Omega)$) and let $T^* > 0$. The space $L^p(0, T^*; X)$ is the set of all strongly measurable functions $\mathbf{u} : [0, T^*] \to X$ such that

$$\|\mathbf{u}\|_{L^p(0,T^*;X)} = \left(\int_0^{T^*} \|\mathbf{u}(t)\|^p \, dt\right)^{1/p} < \infty$$

for $1 \leq p < \infty$, and

$$\|\mathbf{u}\|_{L^\infty(0,T^*;X)} = \operatorname*{ess\,sup}_{0 \leq t \leq T^*} \|\mathbf{u}(t)\| < \infty.$$

Similarly, $C([0, T^*]; X)$ is the set of all continuous functions $\mathbf{u} : [0, T^*] \to X$ satisfying

$$\|\mathbf{u}\|_{C([0,T^*];X)} = \max_{0 \leq t \leq T^*} \|\mathbf{u}(t)\| < \infty.$$

Finally, a function $\mathbf{u} \in L^1(0, T^*; X)$ has a *weak derivative* $\mathbf{u}'$ if there exists a function

المنارة للاستشارات
www.manaraa.com

$\mathbf{v} \in L^1(0, T^*; X)$ such that

$$\int_0^{T^*} \phi'(t)\mathbf{u}(t)\, dt = -\int_0^{T^*} \phi(t)\mathbf{v}(t)\, dt$$

for all $\phi \in C_c^\infty([0, T^*])$.

المنارة للاستشارات

www.manaraa.com

Appendix B. Analytical Tools

Here we review some fundamental tools for analyzing partial differential equations, including the divergence theorem, integration by parts, and several important inequalities. Most of these are used in the proof of Theorem 2.6.

For this appendix, let $\Omega \subset \mathbb{R}^n$ be bounded, and let the boundary $\partial\Omega$ be Lipschitz with outward-pointing normal $\boldsymbol{\nu}$.

B.1 Vector Calculus

Proposition B.1 (Divergence Theorem). *Given* $\mathbf{u} \in W_1^1(\Omega)^n$,

$$\int_\Omega \nabla \cdot \mathbf{u}\, d\mathbf{x} = \int_{\partial\Omega} \mathbf{u} \cdot \boldsymbol{\nu}\, d\sigma.$$

Corollary B.2. *If* $\mathbf{F} \in W_1^1(\Omega)^n$ *and* $g \in W_1^1(\Omega)$,

$$\int_\Omega \mathbf{F} \cdot \nabla g + g(\nabla \cdot \mathbf{F})\, d\mathbf{x} = \int_{\partial\Omega} g\mathbf{F} \cdot \boldsymbol{\nu}\, d\sigma.$$

Proof. The divergence operator obeys the product rule

$$\nabla \cdot (g\mathbf{F}) = \sum_{i=1}^{n} \partial_{x_i}(gf_i) = \sum_{i=1}^{n} (\partial_{x_i} g) f_i + g(\partial_{x_i} f_i) = \nabla g \cdot \mathbf{F} + g(\nabla \cdot \mathbf{F}).$$

The result then follows by setting $\mathbf{u} = g\mathbf{F}$ in Theorem B.1. □

Proposition B.3 (Integration by Parts). *If* $u \in H^2(\Omega)$, $v \in H^1(\Omega)$,

$$\int_\Omega (-\Delta u) v\, d\mathbf{x} = \int_\Omega \nabla u \cdot \nabla v\, d\mathbf{x} + \int_{\partial\Omega} v \nabla u \cdot \boldsymbol{\nu}\, d\sigma.$$

المنارة للاستشارات
www.manaraa.com

B.2 Inequalities

Proposition B.4 (Gronwall's Inequality)**.** *Let $f : [0, T^*] \to [0, \infty)$ be absolutely continuous and $\phi : [0, T^*] \to [0, \infty)$ be integrable. If the differential inequality*

$$f'(t) - \phi(t)f(t) \leq 0$$

holds for almost every $t \in [0, T^*]$, then for $0 \leq t \leq T^*$ we have

$$f(t) \leq f(0)e^{\int_0^t \phi(s)\, ds}.$$

Proposition B.5 (Inverse Poincaré Inequality)**.** *Let $\{(\lambda_k, \phi_k)\}_{k=1}^{\infty}$ be the eigenpairs of $-\Delta$, ordered so that $0 < \lambda_1 \leq \lambda_2 \leq \cdots$. Suppose that $f \in L^2(\Omega)$ can be expressed as a linear combination of the first N basis functions:*

$$f(\mathbf{x}, t) = \sum_{k=1}^{N} f_k(t)\phi_k(\mathbf{x}).$$

If $\int_{\partial\Omega} f\nabla f \cdot \boldsymbol{\nu}\, d\sigma = 0$, then we have

$$\|\nabla f(t)\|_{L^2(\Omega)}^2 \leq \lambda_N \|f(t)\|_{L^2(\Omega)}^2.$$

Proof. To begin, observe

$$-\Delta f = -\Delta \sum_{k=1}^{N} f_k\phi_k = \sum_{k=1}^{N} f_k(-\Delta\phi_k) = \sum_{k=1}^{N} f_k\lambda_k\phi_k \leq \lambda_N \sum_{k=1}^{N} f_k\phi_k = \lambda_N f.$$

Integration by parts then yields the desired inequality:

$$\|\nabla f\|_{L^2(\Omega)}^2 = \int_\Omega \nabla f \cdot \nabla f\, d\mathbf{x} = \int_\Omega (-\Delta f)f\, d\mathbf{x} + \int_{\partial\Omega} f\nabla f \cdot \boldsymbol{\nu}\, d\sigma \leq \lambda_N \int_\Omega f^2\, d\mathbf{x} = \lambda_N \|f\|_{L^2(\Omega)}^2.$$

□

المنارة للاستشارات
www.manaraa.com

The following files are coded in Python 3.6, the most recent version of Python at the time of this writing. Most of the modules and packages that are used are part of the Python standard library; those that are not may be installed with `pip install <package>`. See [6] or `http://dedalus-project.org/` for more on the third-party `dedalus` package, including installation.

`base_simulator.py`. Defines a class, `BaseSimulator`, that takes care of record keeping and data management for Dedalus simulations.

```
# base_simulator.py
"""Base class / variables for Dedalus simulations with good file management.

Author: Shane McQuarrie
"""

import os
import re
import json
import time
import logging
from glob import glob
from mpi4py import MPI

from dedalus.tools import post

# Global variables ============================================================

RANK = MPI.COMM_WORLD.rank        # Which process this is running on
SIZE = MPI.COMM_WORLD.size        # How many processes are running

# Log file formatting
LOG_FORMAT = "(%(asctime)s, {:0>2}/{:0>2}) %(levelname)s: %(message)s".format(
                                                                RANK+1, SIZE)
LOG_FILE = "process{:0>2}.log".format(RANK)
logging.basicConfig(format=LOG_FORMAT)

# Regex for finding data files
FILE_INDEX = re.compile(r"_s(\d+?)(?:_p\d)?\.h5$")

# Parameter storage filename
PARAMS = "params.json"

# Simulation Class =============================================================

class BaseSimulator:
    def __init__(self, records_dir=None, log=True):
        """Store variables for record keeping.

        Parameters:
            records_dir (str): the directory in which all simulation data
                is stored or retrieved. If None (default), the directory is
```


www.manaraa.com

```
                B2D__date_mm_dd_yyyy__time_hh_mm/, which depends on the day
                and time of creation.
        """
        self._name = type(self).__name__
        if not records_dir:             # Default: directory name by timestamp.
            records_dir = self._name + time.strftime("__%m_%d_%Y__%H_%M")
        records_dir = records_dir.strip(os.sep)
        if os.path.isdir(records_dir):  # Notify user if directory exists.
            print("Connecting to existing directory {}".format(records_dir))
        else:                           # Default: make new directory.
            try:
                os.mkdir(records_dir)
                print("Created new directory {}".format(records_dir))
            except FileExistsError:     # Other process might create it first.
                pass

        # Log information to a logfile in the records directory.
        logger = logging.getLogger(records_dir + str(RANK))
        logger.setLevel(logging.DEBUG)
        if log and len(logger.handlers) == 0:
            logfile = logging.FileHandler(os.path.join(records_dir, LOG_FILE))
            logfile.setFormatter(logging.Formatter(LOG_FORMAT))
            logfile.setLevel(logging.DEBUG)
            logger.addHandler(logfile)

        # Store variables.
        self.records_dir = records_dir
        self.params_file = os.path.join(records_dir, PARAMS)
        self.logger = logger
        self.problem = None

        # If the parameter file already exists, load and print it.
        if os.path.isfile(self.params_file) and RANK == 0:
            self.logger.info("Previous simulation parameters:")
            for key, value in self._load_params(self.records_dir).items():
                self.logger.info("\t'{}': {}".format(key, value))

    def setup(*args, **kwargs):
        """Define the dedalus problem (de.IVP, de.EVP, de.LBVP, or de.NLBVP),
        including variables, substitutions, equations, and boundary conditions.

        IMPORTANT: Save the dedalus problem object as 'self.problem'.
        """
        raise NotImplementedError("setup() must be implemented in subclasses")

    def simulate(*args, **kwargs):
        """Create a solver (self.problem.build_solver(scheme)), set initial
        conditions on the problem (if appropriate), set up analyses, and run
        the actual simulation.
        """
        raise NotImplementedError("simulate() must be implemented in"
                                                              " subclasses")

    def _save_params(self):
        """Save a dictionary of problem parameters to a readable JSON file.
        If such a file already exists, compare the current problem parameters
        to the old parameters, log any differences, and overwrite the file.
        """
        if not self.problem:
            raise NotImplementedError("Problem not initialized (call setup())")

        def JSONish(x):
            """Return True if x is JSON serializable, False otherwise."""
            try:
                json.dumps(x)
                return True
```


www.manaraa.com

```
        except TypeError:
            return False

    params = {k:v for k,v in self.problem.parameters.items() if JSONish(v)}
    self.logger.info("Writing parameters to '{}'".format(self.params_file))
    for key, value in params.items():
        self.logger.info("\t'{}': {}".format(key, value))

    # If there is already a params file, compare old and new parameters.
    if os.path.isfile(self.params_file):
        old_params = self._load_params(self.records_dir)
        if old_params != params:
            self.logger.info("Saved parameters updated")
        old_keys, new_keys = set(old_params.keys()), set(params.keys())
        # Report missing or new parameter keys.
        for key in old_keys - new_keys:
            self.logger.info("\tOld param '{}' removed (was {})".format(
                                                key, old_keys[key]))
        for key in new_keys - old_keys:
            self.logger.info("\tNew param '{}'".format(key))
        # Report changed parameters.
        for key in old_keys & new_keys:
            old, new = old_params[key], params[key]
            if old != new:
                self.logger.info("\tParam '{}' changed: {} -> {}".format(
                                                        key, old, new))

    with open(self.params_file, 'w') as outfile:
        json.dump(params, outfile)

@staticmethod
def _load_params(records_dir):
    """Load a dictionary of problem parameters from a given directory."""
    # Make sure the directory exists.
    if not os.path.isdir(records_dir):
        raise NotADirectoryError(records_dir)

    # Load the parameters.
    with open(os.path.join(records_dir, PARAMS), 'r') as infile:
        return json.load(infile)

@staticmethod
def _file_index(f):
    """Get the index of a dedalus h5 file. For example,
    states_s1.h5 -> 1, analysis_s10.h5 -> 10, etc.
    """
    out = FILE_INDEX.findall(f)
    try:
        return int(out[0])
    except IndexError:
        return -1

def get_files(self, label):
    """Return a sorted list of the merged h5 data files.

    Parameters:
        label (str): The name of the subdirectory containing merged .h5
            files. For example, if label="states", there should be a folder
            self.records_dir/states/ containing at least one .h5 file.

    Raises:
        NotADirectoryError: if the specified subdirectory does not exist.
        FileNotFoundError: if the specified subdirectory exists but there
            are no matching files.
    """
    # Check that the relevant folder exists.
```


المنارة للاستشارات
www.manaraa.com

```
        subdir = os.path.join(self.records_dir, label)
        if not os.path.isdir(subdir):
            raise NotADirectoryError(subdir)

        # Get the list of files.
        out = sorted(glob(os.path.join(subdir, "*.h5")), key=self._file_index)
        if len(out) == 0:
            raise FileNotFoundError("no {} files found".format(label))

        return out

    def merge_results(self, label, full_merge=False, force=False):
        """Merge the different process result files together.

        Parameters:
            label (str): The name of the subdirectory containing folders where
                each process computes results. For example, if label="states",
                then self.records_dir/states/ should exist and contain at least
                one subfolder named states_s1/ (or similar), which in turn
                contains .h5 files from each process (states_s1_p0.h5, etc.).
            full_merge (bool): If true, merge the process files AND merge
                the resulting files into one large file. For example,
                states_s1_p0.h5 and states_s1_p1.h5 are merged into
                states_s1.h5 like usual, and then states_s1.h5 and states_s2.h5
                are merged into one large states.h5.
            force (bool): If true, merge the files even if there is an existing
                merged file.
        """
        # Check that the relevant folder exists.
        subdir = os.path.join(self.records_dir, label)
        if not os.path.isdir(subdir):
            raise NotADirectoryError(subdir)

        # Check for existing folders and missing files before merging.
        work_todo = False
        if full_merge:
            work_todo = not os.path.isfile(os.path.join(subdir, label+".h5"))
        else:
            for d in os.listdir(subdir):
                target = os.path.join(subdir, d)
                if os.path.isdir(target) and not os.path.isfile(target+".h5"):
                    work_todo = True
                    break
        if work_todo or force:
            self.logger.info("Merging {} files...".format(label))
            post.merge_process_files(subdir,cleanup=False,comm=MPI.COMM_WORLD)
            if full_merge:
                # Wait for other processes to finish.
                MPI.COMM_WORLD.Barrier()
                # Do the last merge.
                set_paths = glob(os.path.join(subdir, label+"_s*.h5"))
                post.merge_sets(os.path.join(subdir, label+".h5"),
                                set_paths, cleanup=True, comm=MPI.COMM_WORLD)
            self.logger.info("\t{} files now {}merged".format(label,
                                        "fully " if full_merge else ""))

    def __str__(self):
        """String representation: the raw equations, boundary conditions, and
        parameters of the dedalus problem object.
        """
        if not self.problem:
            return "Problem not initialized (call setup())"
        out = self._name + " System\n\nEquations:\n\t"
        out += "\n\t".join([q["raw_equation"] for q in self.problem.equations])
        out += "\nBoundary Conditions:\n\t"
        out += "\n\t".join([q["raw_equation"] for q in self.problem.bcs])
```


المنارة للاستشارات
www.manaraa.com

```
        out += "\nParameters:\n\t"
        out += "\n\t".join("{}: {}".format(key,value)
                          for key,value in self.problem.parameters.items())
        return out


def all_experiment_params():
    """Get a dictionary mapping foldername to parameter dictionaries, as given
    by BaseSimulator._load_params().
    """
    return {f: BaseSimulator._load_params(f) for f in os.listdir()
              if os.path.isdir(f) and os.path.isfile(os.path.join(f, PARAMS))}
```

`boussinesq2d.py`. Defines simulation classes that inherit from `BaseSimulator`. These subclasses define the data and assimilating equations (`setup()`), the projection operator P_N (`P_N()`), and run the corresponding simulations (`simulate()`). They also include methods for visualizing simulation results (`plot_convergence()`, `animate_temperature()`, and so on). The only difference between the two simulation subclasses is that the first, `BoussinesqDataAssimilation2D`, records norms like $\|(\tilde{T} - T)(t)\|_{L(\Omega)}$ (for creating plots like Figure 3.2), while the second, `BoussinesqDataAssimilation2Dmovie`, records the state variables T, $\tilde{T}$, and so on (for creating animations and plots like Figure 3.1).

```
# boussinesq2d.py
"""Dedalus script for data assimilation in the Boussinesq equations.

Authors: Shane McQuarrie, Jared Whitehead
"""

import os
import re
import h5py
import time
import numpy as np
from scipy.integrate import simps
from matplotlib import pyplot as plt
from matplotlib.animation import writers as mplwriters
try:
    from tqdm import tqdm
except ImportError:
    print("Recommended: install tqdm (pip install tqdm)")
    tqdm = lambda x: x

from dedalus import public as de
from dedalus.extras import flow_tools
from dedalus.core.operators import GeneralFunction

from base_simulator import BaseSimulator, RANK, SIZE

# Simulation Classes ==========================================================

class BoussinesqDataAssimilation2D(BaseSimulator):
    """Manager for dedalus simulations of the 2D Boussinesq system.
```


المنارة للاستشارات
www.manaraa.com

```
    Let Psi = [0,L]x[0,1] with coordinates (x,z). Defining
    u = [v, w] = [-psi_z, psi_x] and zeta = laplace(psi),
    the Boussinesq equations can be written as follows.

    Pr [Ra T_x + laplace(zeta)] - zeta_t = u.grad(zeta)
                        laplace(T) - T_t = u.grad(T)
    subject to
        u(z=0) = 0 = u(z=1)
        T(z=0) = 1, T(z=1) = 0
        u, T periodic in (x,y) (use a Fourier basis)

    Variables:
        u:R2xR -> R2: the fluid velocity vector field.
        T:R2xR -> R: the fluid temperature.
        p:R2xR -> R: the pressure.
        Ra: the Rayleigh number.
        Pr: the Prandtl number

    If the Prandtl number is infinite, the first equations can be simplified.

    - zeta_t = u.grad(zeta)
    """
    @staticmethod
    def P_N(F, N, scale=False):
        """Calculate the Fourier mode projection of F with N terms."""
        # Set the c_n to zero wherever n > N (in both axes).
        X,Y = np.indices(F['c'].shape)
        F['c'][(X >= N) | (Y >= N)] = 0

        if scale:
            F.set_scales(1)
        return F['g']

    def setup(self, L=4, xsize=256, zsize=128, Prandtl=None, Rayleigh=10000,
                                        mu=1, N=32, BCs="no-slip"):
        """Set up the systems of equations as a dedalus Initial Value Problem,
        without providing initial conditions yet.

        Parameters:
            L (float): the length of the x domain. In x and z, the domain is
                therefore [0,L]x[0,1].
            xsize (int): the number of points to discretize in the x direction.
            zsize (int): the number of points to discretize in the z direction.
            Prandtl (None or float): the ratio of momentum diffusivity to
                thermal diffusivity of the fluid. If None (default), then
                the system is set up as if Prandtl = infinity.
            Rayleigh (float): measures the amount of heat transfer due to
                convection, as opposed to conduction.
            mu (float): constant on the Fourier projection in the
                Data Assimilation system.
            N (int): the number of modes to keep in the Fourier projection.
            BCs (str): if 'no-slip', use the no-slip BCs u(z=0,1) = 0.
                If 'free-slip', use the free-slip BCs u_z(z=0,1) = 0.
        """
        # Validate BCs parameter.
        if BCs not in {"no-slip", "free-slip"}:
            raise ValueError("'BCs' must be 'no-slip' or 'free-slip'")

        # Validate N parameter.
        minsize = min(xsize, zsize)
        if not 0 <= N <= minsize:
            raise ValueError("0 <= N <= {} is required".format(minsize))

        # Bases and Domain ---------------------------------------------------
        x_basis = de.Fourier('x', xsize, interval=(0, L), dealias=3/2)
```


المنارة للاستشارات
www.manaraa.com

```
z_basis = de.Chebyshev('z', zsize, interval=(0, 1), dealias=3/2)
domain = de.Domain([x_basis, z_basis], grid_dtype=np.float64)

# Initialize the problem as an IVP and add variables ------------------
problem = de.IVP(domain, variables=['T', 'T_',      # Temperature
                                    'Tz', 'Tz_',
                                    'psi', 'psi_',  # Stream function
                                    'psiz', 'psiz_',
                                    'zeta', 'zeta_',# Laplace of stream
                                    'zetaz', 'zetaz_'])

driving = GeneralFunction(domain, 'g',
                          BoussinesqDataAssimilation2D.P_N, args=[])

# System parameters (these are saved to a JSON file).
problem.parameters['L'] = L                 # Domain parameters
problem.parameters['xsize'] = xsize
problem.parameters['zsize'] = zsize

problem.parameters['Ra'] = Rayleigh         # Fluid parameters
if Prandtl:
    problem.parameters['Pr'] = Prandtl

problem.parameters["N"] = N                 # Assimilation parameters
problem.parameters["mu"] = mu
problem.parameters["driving"] = driving

# Stream function substitutions: u = [v, w] = [-psi_z, psi_w]
problem.substitutions['v']  = "-dz(psi)"
problem.substitutions['v_'] = "-dz(psi_)"
problem.substitutions['w']  =  "dx(psi)"
problem.substitutions['w_'] =  "dx(psi_)"

# Relate higher-order z derivatives (b/c Chebyshev).
problem.add_equation("psiz  - dz(psi)  = 0")
problem.add_equation("psiz_ - dz(psi_) = 0")
problem.add_equation("zetaz  - dz(zeta)  = 0")
problem.add_equation("zetaz_ - dz(zeta_) = 0")
problem.add_equation("Tz  - dz(T)  = 0")
problem.add_equation("Tz_ - dz(T_) = 0")

# zeta = laplace(psi)
problem.add_equation("zeta  - dx(dx(psi))  - dz(psiz)  = 0")
problem.add_equation("zeta_ - dx(dx(psi_)) - dz(psiz_) = 0")

# 2D Boussinesq Equations -------------------------------------------
if Prandtl is None: # Ra T_x + laplace(zeta) = 0
    problem.add_equation("Ra*dx(T)  + dx(dx(zeta))  + dz(zetaz)  = 0")
    problem.add_equation("Ra*dx(T_) + dx(dx(zeta_)) + dz(zetaz_) = 0")
else:               # Pr(Ra T_x + laplace(zeta)) - zeta_t = u.grad(zeta)
    problem.add_equation("Pr*(Ra*dx(T)  + dx(dx(zeta))  + dz(zetaz))"
                         " - dt(zeta)  = v*dx(zeta)   + w*zetaz")
    problem.add_equation("Pr*(Ra*dx(T_) + dx(dx(zeta_)) + dz(zetaz_))"
                         " - dt(zeta_) = v_*dx(zeta_) + w_*zetaz_")

# T_t - laplace(T) = -u . grad(T) (+ driving Fourier projection)
problem.add_equation("dt(T)  - dx(dx(T))  - dz(Tz)  "
                     "= -v*dx(T)   - w*Tz")
problem.add_equation("dt(T_) - dx(dx(T_)) - dz(Tz_) "
                     "= -v_*dx(T_) - w_*Tz_ - mu*driving")

# Boundary Conditions -----------------------------------------------
# Temperature heating from 'left' (bottom), cooling from 'right' (top).
problem.add_bc("left(T)  = 1")          # T(z=0) = 1
problem.add_bc("left(T_) = 1")
problem.add_bc("right(T)  = 0")         # T(z=1) = 0
```


www.manaraa.com

```
        problem.add_bc("right(T_) = 0")

        # Velocity field boundary conditions: no-slip or free-slip.
        # w(z=1) = w(z=0) = 0 (part of no-slip and free-slip)
        problem.add_bc("left(psi)  = 0")
        problem.add_bc("left(psi_) = 0")
        problem.add_bc("right(psi)  = 0")
        problem.add_bc("right(psi_) = 0")

        if BCs == "no-slip":
            # u(z=0) = 0 --> v(z=0) = 0 = w(z=0)
            problem.add_bc("left(psiz)  = 0")
            problem.add_bc("left(psiz_) = 0")

            # u(z=1) = 0 --> v(z=1) = 0 = w(z=1)
            problem.add_bc("right(psiz)  = 0")
            problem.add_bc("right(psiz_) = 0")

        elif BCs == "free-slip":
            # u'(z=0) = 0 --> v'(z=0) = 0 = w(z=0)
            problem.add_bc("left(dz(psiz))  = 0")
            problem.add_bc("left(dz(psiz_)) = 0")

            # u'(z=1) = 0 --> v'(z=1) = 0 = w(z=1)
            problem.add_bc("right(dz(psiz))  = 0")
            problem.add_bc("right(dz(psiz_)) = 0")

        self.problem = problem
        self.logger.info("Problem constructed")

        # Save system parameters in JSON format.
        if RANK == 0:
            self._save_params()

    def simulate(self, initial_conditions=None, scheme=de.timesteppers.RK443,
                       sim_time=2, wall_time=np.inf, tight=False, save=.05,
                       analysis=True):
        """Load initial conditions, run the simulation, and merge results.

        Parameters:
            initial_conditions (None, str): determines from what source to
                draw the initial conditions. Valid options are as follows:
                - None: use trivial conditions (T_ = 1 - z, T = 1 - z + eps).
                - 'resume': use the most recent state file in the
                    records directory (load both model and DA system).
                - An .h5 filename: load state variables for the model and
                    reset the data assimilation state variables to zero.
            scheme (de.timesteppers): The kind of solver to use. Options are
                RK443 (de.timesteppers.RK443), RK111, RK222, RKSMR, etc.
            sim_time (float): The maximum amount of simulation time allowed
                (in seconds) before ending the simulation.
            wall_time (float): The maximum amound of computing time allowed
                (in seconds) before ending the simulation.
            tight (bool): If True, set a low cadence and min_dt for refined
                simulation. If False, set a higher cadence and min_dt for a
                more coarse (but faster) simulation.
            save (float): The number of simulation seconds that pass between
                saving the state files. Higher save result in smaller data
                files, but lower numbers result in better animations.
                Set to 0 to disable saving state files.
            analysis (bool): Whether or not to track convergence measurements.
                Disable for faster simulations (less message passing via MPI)
                when convergence estimates are not needed (i.e. movie only).
        """
        if not self.problem:
            raise TypeError("problem not initialized (run setup())")
```


المنارة للاستشارات
www.manaraa.com

```
        self.logger.debug("\n")
        self.logger.debug("NEW SIMULATION")
        solver = self.problem.build_solver(scheme)
        self.logger.info("Solver built")

        N = int(self.problem.parameters['N'])

        # Initial conditions ----------------------------------------------------
        if initial_conditions is None:              # "Trivial" conditions.
            eps = 1e-4
            k = 3.117
            dt = 1e-4

            x,z = self.problem.domain.grids(scales=1)
            T, T_ = solver.state['T'], solver.state['T_']
            # Start T from rest plus a small perturbation.
            T['g']  = 1 - z + eps*np.sin(k*x)*np.sin(2*np.pi*z)
            T.differentiate('z', out=solver.state['Tz'])
            # Start T_ from rest.
            T_['g'] = 1 - z
            T_.differentiate('z', out=solver.state['Tz_'])
            self.logger.info("Using trivial initial conditions")

        elif isinstance(initial_conditions, str):   # Load data from a file.
            # Resume: load the state of the last (merged) state file.
            resume = initial_conditions == "resume"
            if resume:
                initial_conditions = self._get_merged_file("states")
            if not initial_conditions.endswith(".h5"):
                raise ValueError("'{}' is not an h5 file".format(
                                                initial_conditions))
            # Load the data from the specified h5 file into the system.
            self.logger.info("Loading initial conditions from {}".format(
                                                initial_conditions))

            with h5py.File(initial_conditions, 'r') as infile:
                dt = infile["scales/timestep"][-1] * .01    # initial dt
                errs = []
                tasks = ["T", "Tz", "psi", "psiz", "zeta", "zetaz"]
                if resume:      # Only load assimilating variables to resume.
                    tasks += ["T_", "Tz_", "psi_", "psiz_", "zeta_", "zetaz_"]
                    solver.sim_time = infile["scales/sim_time"][-1]
                    niters = infile["scales/iteration"][-1]
                    solver.initial_iteration = niters
                    solver.iteration = niters
                for name in tasks:
                    # Get task data from the h5 file (recording failures).
                    try:
                        data = infile["tasks/"+name][-1,:,:]
                    except KeyError as e:
                        errs.append("tasks/"+name)
                        continue
                    # Determine the chunk belonging to this process.
                    chunk = data.shape[1] // SIZE
                    subset = data[:,RANK*chunk:(RANK+1)*chunk]
                    # Change the corresponding state variable.
                    scale = solver.state[name]['g'].shape[0] / \
                                    self.problem.parameters["xsize"]
                    solver.state[name].set_scales(1)
                    solver.state[name]['g'] = subset
                    solver.state[name].set_scales(scale)
                if errs:
                    raise KeyError("Missing keys in '{}': '{}'".format(
                                initial_conditions, "', '".join(errs)))
            # Initial conditions for assimilating system: T_0 = P_4(T0).
```


المنارة للاستشارات
www.manaraa.com

```
        if not resume:
            G = self.problem.domain.new_field()
            G['c'] = solver.state['T']['c'].copy()
            solver.state['T_']['g'] = BoussinesqDataAssimilation2D.P_N(
                                                            G, 4, True)
            solver.state['T_'].differentiate('z', out=solver.state['Tz_'])

    # Driving / projection function arguments -----------------------------

    dT = solver.state['T_'] - solver.state['T']
    self.problem.parameters["driving"].args = [dT, N]
    self.problem.parameters["driving"].original_args = [dT, N]

    # Stopping Parameters -------------------------------------------------

    solver.stop_sim_time = sim_time         # Length of simulation.
    solver.stop_wall_time = wall_time       # Real time allowed to compute.
    solver.stop_iteration = np.inf          # Maximum iterations allowed.

    # State snapshots -----------------------------------------------------
    if save:
        # Save the temperature measurements in states/ files. Use sim_dt.
        snaps = solver.evaluator.add_file_handler(
                                os.path.join(self.records_dir, "states"),
                                sim_dt=save, max_writes=5000,mode="append")
                                # Set save=0.005 or lower for more writes.
        snaps.add_task("T")
        snaps.add_task("T_")
        snaps.add_task("driving", name="P_N")

    # Convergence analysis ------------------------------------------------
    if analysis:
        # Save specific tasks in analysis/ files every few iterations.
        annals = solver.evaluator.add_file_handler(
                                os.path.join(self.records_dir, "analysis"),
                                iter=20, max_writes=73600, mode="append")

        # Nusselt Number measurements - - - - - - - - - - - - - - - - - - -
        # 1 + int(wT)/L
        annals.add_task("1 + integ(w *T , 'x','z')/L", name="Nu_1")
        annals.add_task("1 + integ(w_*T_, 'x','z')/L", name="Nu_1_da")
        # int(grad(T)^2)/L
        annals.add_task("integ(dx(T )**2 + Tz **2, 'x','z')/L",
                                                    name="Nu_2")
        annals.add_task("integ(dx(T_)**2 + Tz_**2, 'x','z')/L",
                                                    name="Nu_2_da")
        # 1 + int(grad(u)^2)/(Ra L)
        annals.add_task("1 + "
            "integ(dx(v )**2 + dz(v )**2 + dx(w )**2 + dz(w )**2,'x','z')"
                                        "/(Ra*L)", name="Nu_3")
        annals.add_task("1 + "
            "integ(dx(v_)**2 + dz(v_)**2 + dx(w_)**2 + dz(w_)**2,'x','z')"
                                        "/(Ra*L)", name="Nu_3_da")

        # Convergence estimates - - - - - - - - - - - - - - - - - - - - - -
        # ||T - T_||_2
        annals.add_task("sqrt( integ((T - T_)**2, 'x','z')"
                                "/integ(T**2, 'x', 'z') )", name="T_err")
        # ||grad(T) - grad(T_)||_2
        # Could use dz(T-T_) or dz(T)-dz(T_) or Tz-Tz_
        annals.add_task("sqrt( integ(dx(T-T_)**2 + dz(T-T_)**2, 'x','z')"
                                "/integ(dx(T)**2 + dz(T)**2, 'x','z') )",
                                                    name="gradT_err")
        # ||u - u_||_2
        annals.add_task("sqrt( integ((v-v_)**2 + (w-w_)**2, 'x','z')"
                                "/integ(v**2 + w**2, 'x','z') )",
```


المنارة للاستشارات
www.manaraa.com

```
                                                      name="u_err")
    # ||grad(u - u_)||_2
    annals.add_task("sqrt( integ(dx(v-v_)**2 + dz(v-v_)**2"
                             " + dx(w-w_)**2 + dz(w-w_)**2, 'x','z')"
                         "/integ(dx(v)**2 + dz(v)**2"
                             " + dx(w)**2 + dz(w)**2, 'x','z') )",
                                                      name="gradu_err")
    # ||T - T_||_H2
    annals.add_task("sqrt( integ(dx(dx(T-T_))**2 + dx(dz(T-T_))**2 "
                                      "+ dz(dz(T-T_))**2, 'x','z')"
                      "/integ(dx(dx(T))**2 + dx(dz(T))**2 "
                                      "+ dz(dz(T))**2, 'x','z') )",
                                                      name="T_h2_err")
    # ||u - u_||_H2
    annals.add_task("sqrt("
                   "integ( dx(dx(v-v_))**2 + dz(dz(v-v_))**2"
                     " + dx(dz(v-v_))**2 + dx(dz(w-w_))**2"
                     " + dx(dx(w-w_))**2 + dz(dz(w-w_))**2, 'x','z')"
                  "/integ( dx(dx(v))**2 + dz(dz(v))**2"
                     " + dx(dz(v))**2 + dx(dz(w))**2"
                     " + dx(dx(w))**2 + dz(dz(w))**2, 'x','z') )",
                                                      name="u_h2_err")

# Control Flow --------------------------------------------------------
if tight:                   # Tighter control flow (slower but safer).
    cfl = flow_tools.CFL(solver, initial_dt=dt, cadence=1, safety=1,
                                max_change=1.5, min_change=0.01,
                                max_dt=0.01,    min_dt=1e-10)
else:                       # Looser control flow (faster but risky).
    cfl = flow_tools.CFL(solver, initial_dt=dt, cadence=10, safety=1,
                                max_change=1.5, min_change=0.5,
                                max_dt=0.01,    min_dt=1e-6)
cfl.add_velocities(('v',  'w' ))
cfl.add_velocities(('v_', 'w_'))

# Flow properties (print during run; not recorded in the records files)
flow = flow_tools.GlobalFlowProperty(solver, cadence=1)
flow.add_property("sqrt(v **2 + w **2) / Ra", name='Re' )
flow.add_property("sqrt(v_**2 + w_**2) / Ra", name='Re_')

# MAIN COMPUTATION LOOP -----------------------------------------------
try:
    self.logger.info("Starting main loop")
    start_time = time.time()
    while solver.ok:
        # Recompute time step and iterate.
        dt = cfl.compute_dt()
        dt = solver.step(dt) #, trim=True)

        # Print info to the screen every 10 iterations.
        if solver.iteration % 10 == 0:
            info = "Iteration {:>5d}, Time: {:.7f}, dt: {:.2e}".format(
                                solver.iteration, solver.sim_time, dt)
            Re  = flow.max("Re")
            Re_ = flow.max("Re_")
            info += ", Max Re = {:f}".format(Re)
            info += ", Max Re_ = {:f}".format(Re_)
            self.logger.info(info)
            # Make sure the simulation hasn't blown up.
            if np.isnan(Re) or np.isnan(Re_):
                raise ValueError("Reynolds number went to infinity!!"
                                 "\nRe = {}, Re_ = {}".format(Re, Re_))
except BaseException as e:
    self.logger.error("Exception raised, triggering end of main loop.")
    raise
finally:
```


المنارة للاستشارات
www.manaraa.com

```
            total_time = time.time() - start_time
            cpu_hr = total_time / 60 / 60 * SIZE
            self.logger.info("Iterations: {:d}".format(solver.iteration))
            self.logger.info("Sim end time: {:.3e}".format(solver.sim_time))
            self.logger.info("Run time: {:.3e} sec".format(total_time))
            self.logger.info("Run time: {:.3e} cpu-hr".format(cpu_hr))
            self.logger.debug("END OF SIMULATION\n")

    def _get_merged_file(self, label):
        """Return the name of the oldest merged (full or partial) h5 file with
        the specified label.
        """
        if label not in {"states", "analysis"}:
            raise ValueError("label must be 'states' or 'analysis'")
        out = self.get_files(label)
        if out[0].endswith("{}.h5".format(label)):
            return out[0]
        return out[-1]

    @staticmethod
    def _get_fully_merged_state_file(records_dir):
        """Return the name of the fully merged h5 state file, without doing
        any file merges if the file does not exist.

        Parameters:
            records_dir (str): The base folder containing the simulation files.

        Raises:
            NotADirectoryError: if the states/ subdirectory does not exist.
            FileNotFoundError: if the states/states.h5 file does not exist.
        """
        subdir = os.path.join(records_dir, "states")
        if not os.path.isdir(subdir):
            raise NotADirectoryError(subdir)

        target = os.path.join(records_dir, "states", "states.h5")
        if not os.path.isfile(target):
            raise FileNotFoundError(target)

        return target

    def merge_results(self, force=False):
        """Merge the different process state and analysis files together."""
        for label in ["analysis", "states"]:
            # Check that the folder exists and is nonempty.
            folder = os.path.join(self.records_dir, label)
            if os.path.isdir(folder) and os.listdir(folder):
                # Call the parent merge function.
                BaseSimulator.merge_results(self, label, True, force=force)
            else:
                # Inform the user that merge files were not found.
                self.logger.info("No {} files to merge".format(label))

    def plot_convergence(self, savefig=True):
        """Plot the six measures of convergence over time."""
        # self.merge_results()
        datafile = self._get_merged_file("analysis")
        self.logger.info("Plotting convergence estimates from '{}'...".format(
                                                                    datafile))
        # Gather data from the source file.
        with h5py.File(datafile, 'r') as data:
            times = list(data["scales/sim_time"])
            T_err = data["tasks/T_err"][:,0,0]
            gradT_err = data["tasks/gradT_err"][:,0,0]
            u_err = data["tasks/u_err"][:,0,0]
            gradu_err = data["tasks/gradu_err"][:,0,0]
```


المنارة للاستشارات
www.manaraa.com

```
    T_h2_err = data["tasks/T_h2_err"][:,0,0]
    u_h2_err = data["tasks/u_h2_err"][:,0,0]

with plt.style.context(".mplstyle"):
    # Make subplots and a big plot for an overlay.
    fig = plt.figure(figsize=(12,6))
    ax1 = plt.subplot2grid((3,4), (0,0))
    ax2 = plt.subplot2grid((3,4), (0,1))
    ax3 = plt.subplot2grid((3,4), (1,0))
    ax4 = plt.subplot2grid((3,4), (1,1))
    ax5 = plt.subplot2grid((3,4), (2,0))
    ax6 = plt.subplot2grid((3,4), (2,1))
    axbig = plt.subplot2grid((3,4), (0,2), rowspan=3, colspan=2)

    # Plot the data.
    ax1.semilogy(times, T_err, 'C0', lw=.5)
    ax2.semilogy(times, u_err, 'C1', lw=.5)
    ax3.semilogy(times, gradT_err, 'C2', lw=.5)
    ax4.semilogy(times, gradu_err, 'C3', lw=.5)
    ax5.semilogy(times, T_h2_err, 'C4', lw=.5)
    ax6.semilogy(times, u_h2_err, 'C5', lw=.5)
    axbig.semilogy(times, T_err, 'C0', lw=.5,
                   label=r"$||(\tilde{T} - T)(t)||_{L^2(\Omega)}$")
    axbig.semilogy(times, u_err, 'C1', lw=.5,
                   label=r"$||(\tilde{\mathbf{u}} - \mathbf{u})(t)||"
                         r"_{L^2(\Omega)}$")
    axbig.semilogy(times, gradT_err, 'C2', lw=.5,
                   label=r"$||(\nabla\tilde{T} - \nabla T)(t)||"
                         r"_{L^2(\Omega)}$")
    axbig.semilogy(times, gradu_err, 'C3', lw=.5,
                   label=r"$||(\nabla\tilde{\mathbf{u}} - \nabla"
                         r"\mathbf{u})(t)||_{L^2(\Omega)}$")
    axbig.semilogy(times, T_h2_err, 'C4', lw=.5,
                   label=r"$||(\tilde{T} - T)(t)||_{H^2(\Omega)}$")
    axbig.semilogy(times, u_h2_err, 'C5', lw=.5,
                   label=r"$||(\tilde{\mathbf{u}} - \mathbf{u})(t)||"
                         r"_{H^2(\Omega)}$")
    axbig.legend(loc="upper right")

    # Set minimal axis and tick labels.
    for ax in [ax1, ax2, ax3, ax4]:
        ax.set_xticklabels([])
    for ax in [ax2, ax4, ax6]:
        ax.set_yticklabels([])
    ax5.set_xlabel("Simulation Time", color="white")
    ax6.set_xlabel("Simulation Time", color="white")
    axbig.set_xlabel("Simulation Time", color="white")
    fig.text(0.5, 0.01, r"Simulation Time $t$", ha="center",
             fontsize=16)
    ax1.set_title(r"$||(\tilde{T} - T)(t)||_{L^2(\Omega)}$")
    ax2.set_title(r"$||(\tilde{\mathbf{u}} - \mathbf{u})(t)||"
                  r"_{L^2(\Omega)}$")
    ax3.set_title(r"$||(\nabla\tilde{T} - \nabla T)(t)||"
                  r"_{L^2(\Omega)}$")
    ax4.set_title(r"$||(\nabla\tilde{\mathbf{u}} - \nabla"
                  r"\mathbf{u})(t)||_{L^2(\Omega)}$")
    ax5.set_title(r"$||(\tilde{T} - T)(t)||_{H^2(\Omega)}$")
    ax6.set_title(r"$||(\tilde{\mathbf{u}} - \mathbf{u})(t)||"
                  r"_{H^2(\Omega)}$")
    axbig.set_title("Overlay")

    # Make the axes uniform and use tight spacing.
    xlims = axbig.get_xlim()
    for ax in [ax1, ax2, ax3, ax4, ax5, ax6, axbig]:
        ax.set_xlim(xlims)
        ax.set_ylim(1e-11, 1e1)
```


المنارة للاستشارات
www.manaraa.com

```
            plt.tight_layout()

            # Save or show the figure.
            if savefig:
                outfile = os.path.join(self.records_dir, "convergence.pdf")
                plt.savefig(outfile, dpi=300, bbox_inches="tight")
                self.logger.info("\tFigure saved as '{}'".format(outfile))
            else:
                plt.show()
            plt.close()

    def plot_nusselt(self, savefig=True):
        """Plot the three measures of the Nusselt number over time for the
        base and DA systems.
        """
        # self.merge_results()
        datafile = self._get_merged_file("analysis")
        self.logger.info("Plotting Nusselt number from '{}'...".format(
                                                                datafile))
        # Gather data from the source file.
        times = []
        nusselt = [[] for _ in range(6)]
        with h5py.File(datafile, 'r') as data:
            times = list(data["scales/sim_time"])
            for i in range(1,4):
                label = "tasks/Nu_{}".format(i)
                nusselt[i-1] = data[label][:,0,0]
                nusselt[i+2] = data[label+"_da"][:,0,0]
        t, nusselt = np.array(times), np.array(nusselt)

        # Calculate time averages (integrate using Simpson's rule).
        nuss_avg = np.array([[simps(nu[:n], t[:n]) for n in range(1,len(t)+1)]
                                                    for nu in nusselt])
        nuss_avg[:,1:] /= t[1:]

        with plt.style.context(".mplstyle"):
            # Plot results in 4 subplots (raw nusselt vs time avg, nonDA vs DA)
            fig = plt.figure(figsize=(12,6))
            ax1 = plt.subplot2grid((2,4), (0,0))
            ax2 = plt.subplot2grid((2,4), (0,1), sharey=ax1)
            ax3 = plt.subplot2grid((2,4), (1,0))
            ax4 = plt.subplot2grid((2,4), (1,1), sharey=ax3)
            axbig = plt.subplot2grid((2,4), (0,2), rowspan=2, colspan=2)
            for i in [0,1,2]:
                ax1.plot(t[1:], nusselt[i,1:])
                ax3.plot(t[1:], nuss_avg[i,1:])
                ax2.plot(t[1:], nusselt[i+3,1:])
                ax4.plot(t[1:], nuss_avg[i+3,1:])
            axbig.plot(t[1:], nuss_avg[:3,1:].mean(axis=0),
                        label='Data ("Truth")')
            axbig.plot(t[1:], nuss_avg[3:,1:].mean(axis=0),
                        label="Assimilating System")
            ax1.set_title("Raw Nusselt", fontsize=8)
            ax3.set_title("Time Average", fontsize=8)
            ax2.set_title("DA Raw Nusselt", fontsize=8)
            ax4.set_title("DA Time Average", fontsize=8)
            axbig.set_title("Overlay of Mean Time Averages", fontsize=8)
            axbig.legend(loc="lower right")
            plt.tight_layout()

            if savefig:
                outfile = os.path.join(self.records_dir, "nusselt.pdf")
                plt.savefig(outfile, dpi=300, bbox_inches="tight")
                self.logger.info("\tFigure saved as '{}'".format(outfile))
            else:
                plt.show()
```


المنارة للاستشارات
www.manaraa.com

```python
        plt.close()

def animate_temperature(self, max_frames=np.inf):
    """Animate the temperature results of the simulation (model and DA
    system) and save it to an mp4 file called 'temperature.mp4'.
    """
    # self.merge_results()
    state_file = self._get_merged_file("states")
    self.logger.info("Creating temperature animation from '{}'...".format(
                                                        state_file))

    # Set up the figure / movie writer.
    fig = plt.figure(figsize=(12,6))
    ax1 = plt.subplot2grid((2,2), (0,0))
    ax2 = plt.subplot2grid((2,2), (0,1))
    ax4 = plt.subplot2grid((2,2), (1,0), colspan=2)
    # fig, [[ax1, ax3], [ax2, ax4]] = plt.subplots(2, 2)
    ax1.axis("off"); ax2.axis("off") #; ax3.axis("off")
    ax1.set_title('Data ("Truth")')
    ax2.set_title("Assimilating System")
    # ax3.set_title("Projected Temperature Difference", fontsize=8)
    writer = mplwriters["ffmpeg"](fps=250) # frames per second, sets speed.

    # Rename the old animation if it exists (it will be deleted later).
    outfile = os.path.join(self.records_dir, "temperature.mp4")
    oldfile = os.path.join(self.records_dir, "old_temperature.mp4")
    if os.path.isfile(outfile):
        self.logger.info("\tRenaming old animation '{}' -> '{}'".format(
                                                    outfile, oldfile))
        os.rename(outfile, oldfile)

    # Write the movie at 200 DPI (resolution).
    with writer.saving(fig,outfile,200), h5py.File(state_file,'r') as data:
        print("Extracting data...", end='', flush=True)
        T = data["tasks/T"]
        T_ = data["tasks/T_"]
        # dT = data["tasks/P_N"]
        times = list(data["scales/sim_time"])
        assert len(times) == len(T) == len(T_), "mismatched dimensions"
        print("done")

        # Plot ||T_ - T||_L^infinity.
        print("Calculating / plotting ||T_ - T||_L^infty(Omega)...",
              end='', flush=True)
        L_inf = np.max(np.abs(T_[:] - T[:]), axis=(1,2))
        ax4.semilogy(times, L_inf, lw=1)
        ax4_line = plt.axvline(x=times[0], color='r', lw=.5)
        _, ylims = ax4_line.get_data()
        ax4.set_xlim(times[0], times[-1])
        ax4.set_ylim(1e-11, 1e1)
        ax4.set_title(r"$||\tilde{T} - T||_{L^\infty(\Omega)} =$" \
                      + "{:.2e}".format(L_inf[0]))
        ax4.spines["right"].set_visible(False)
        ax4.spines["top"].set_visible(False)
        ax4.set_xlabel(r"Simulation Time $t$")
        print("done")

        # Set up color maps for each temperature layer.
        im1 = ax1.imshow( T[0].T, animated=True, cmap="inferno",
                          vmin=0, vmax=1)
        im2 = ax2.imshow(T_[0].T, animated=True, cmap="inferno",
                          vmin=0, vmax=1)
        # im3 = ax3.imshow(dT[0].T, animated=True, cmap="RdBu_r",
        #                    vmin=-.05, vmax=.05)
                          # norm=SymLogNorm(linthresh=1e-10, vmin=-1, vmax=1))
        # im3 = ax3.imshow(np.log(np.abs(T[0] - T_[0]) + 1e-16).T,
```


المنارة للاستشارات
www.manaraa.com

```
            #                   animated=True, cmap="viridis") # log difference
            # fig.colorbar(im3, ax=ax3, fraction=0.023)
            ax1.invert_yaxis() # Flip the images right-side up.
            ax2.invert_yaxis()
            # ax3.invert_yaxis()

            # Save a frame for each layer of task data.
            for j in tqdm(range(min(T.shape[0], max_frames))):
                im1.set_array( T[j].T)     # Truth
                im2.set_array(T_[j].T)     # Approximation
                # im3.set_array(dT[j].T)     # Difference
                # im3.set_array(np.log(np.abs(T[j] - T_[j]) + 1e-16).T)

                # Moving line for ||T - T_||_L^infty error plot.
                t = times[j]
                ax4_line.set_data([[t,t], ylims])
                ax4.set_title(r"$||(\tilde{T}-T)(t)||_{L^\infty(\Omega)} =$" \
                              + "{:.2e}".format(L_inf[j]))
                writer.grab_frame()
        self.logger.info("\tAnimation saved as '{}'".format(outfile))
        plt.close()

        # Delete the old animation.
        if os.path.isfile(oldfile):
            self.logger.info("\tDeleting old animation '{}'".format(oldfile))
            os.remove(oldfile)

    def process_results(self):
        """Call all post-processing methods."""
        self.merge_results()
        if RANK == 0:
            self.plot_convergence()
            self.plot_nusselt()
            self.animate_temperature()


class BoussinesqDataAssimilation2Dmovie(BoussinesqDataAssimilation2D):
    """Same as BoussinesqDataAssimilation2D, but simulate() never saves
    analysis files and always saves the temperature fields at every iteration.
    """
    def simulate(self, initial_conditions=None, scheme=de.timesteppers.RK443,
                 sim_time=2, wall_time=np.inf, tight=False):
        """Load initial conditions, run the simulation, and merge results.

        Parameters:
            initial_conditions (None, str): determines from what source to
                draw the initial conditions. Valid options are as follows:
                - None: use trivial conditions (T_ = 1 - z, T = 1 - z + eps).
                - 'resume': use the most recent state file in the
                    records directory (load both model and DA system).
                - An .h5 filename: load state variables for the model and
                    reset the data assimilation state variables to zero.
            scheme (de.timesteppers): The kind of solver to use. Options are
                RK443 (de.timesteppers.RK443), RK111, RK222, RKSMR, etc.
            sim_time (float): The maximum amount of simulation time allowed
                (in seconds) before ending the simulation.
            wall_time (float): The maximum amound of computing time allowed
                (in seconds) before ending the simulation.
            tight (bool): If True, set a low cadence and min_dt for refined
                simulation. If False, set a higher cadence and min_dt for a
                more coarse (but faster) simulation.
        """
        if not self.problem:
            raise TypeError("problem not initialized (run setup())")

        self.logger.debug("\n")
```


www.manaraa.com

```
        self.logger.debug("NEW SIMULATION")
        solver = self.problem.build_solver(scheme)
        self.logger.info("Solver built")

        N = int(self.problem.parameters['N'])

        # Initial conditions --------------------------------------------------
        if initial_conditions is None:              # "Trivial" conditions.
            dt = 1e-4
            eps = 1e-4
            k = 3.117

            x,z = self.problem.domain.grids(scales=1)
            T, T_ = solver.state['T'], solver.state['T_']
            # Start T from rest plus a small perturbation.
            T['g']  = 1 - z + eps*np.sin(k*x)*np.sin(2*np.pi*z)
            T.differentiate('z', out=solver.state['Tz'])
            # Start T_ from rest.
            T_['g'] = 1 - z
            T_.differentiate('z', out=solver.state['Tz_'])
            self.logger.info("Using trivial initial conditions")

        elif isinstance(initial_conditions, str):   # Load data from a file.
            # Resume: load the state of the last (merged) state file.
            resume = initial_conditions == "resume"
            if resume:
                initial_conditions = self._get_merged_file("states")
            if not initial_conditions.endswith(".h5"):
                raise ValueError("'{}' is not an h5 file".format(
                                                    initial_conditions))
            # Load the data from the specified h5 file into the system.
            self.logger.info("Loading initial conditions from {}".format(
                                                    initial_conditions))

            with h5py.File(initial_conditions, 'r') as infile:
                dt = infile["scales/timestep"][-1] * .01    # initial dt
                errs = []
                tasks = ["T", "Tz", "psi", "psiz", "zeta", "zetaz"]
                if resume:      # Only load assimilating variables to resume.
                    tasks += ["T_", "Tz_", "psi_", "psiz_", "zeta_", "zetaz_"]
                    solver.sim_time = infile["scales/sim_time"][-1]
                    niters = infile["scales/iteration"][-1]
                    solver.initial_iteration = niters
                    solver.iteration = niters
                for name in tasks:
                    # Get task data from the h5 file (recording failures).
                    try:
                        data = infile["tasks/"+name][-1,:,:]
                    except KeyError as e:
                        errs.append("tasks/"+name)
                        continue
                    # Determine the chunk belonging to this process.
                    chunk = data.shape[1] // SIZE
                    subset = data[:,RANK*chunk:(RANK+1)*chunk]
                    # Change the corresponding state variable.
                    scale = solver.state[name]['g'].shape[0] / \
                                        self.problem.parameters["xsize"]
                    solver.state[name].set_scales(1)
                    solver.state[name]['g'] = subset
                    solver.state[name].set_scales(scale)
                if errs:
                    raise KeyError("Missing keys in '{}': '{}'".format(
                                    initial_conditions, "', '".join(errs)))
            # Initial conditions for assimilating system: T_0 = P_4(T0).
            if not resume:
                G = self.problem.domain.new_field()
```


المنارة للاستشارات
www.manaraa.com

```
        G['c'] = solver.state['T']['c'].copy()
        solver.state['T_']['g'] = BoussinesqDataAssimilation2D.P_N(
                                                          G, 4, True)
        solver.state['T_'].differentiate('z', out=solver.state['Tz_'])

# Driving / projection function arguments ----------------------------

dT = solver.state['T_'] - solver.state['T']
self.problem.parameters["driving"].args = [dT, N]
self.problem.parameters["driving"].original_args = [dT, N]

# Stopping Parameters -------------------------------------------------

solver.stop_sim_time = sim_time         # Length of simulation.
solver.stop_wall_time = wall_time       # Real time allowed to compute.
solver.stop_iteration = np.inf          # Maximum iterations allowed.

# State snapshots -----------------------------------------------------

# Save the entire state in states/ files. USE iter, NOT sim_dt.
# NOTE: This is where BoussinesqDataAssimilation2Dmovie differs.
snaps = solver.evaluator.add_file_handler(
                        os.path.join(self.records_dir, "states"),
                        iter=1, max_writes=5000, mode="append")
snaps.add_task("T")
snaps.add_task("T_")
snaps.add_task("driving", name="P_N")

# Control Flow --------------------------------------------------------
if tight:                   # Tighter control flow (slower but safer).
    cfl = flow_tools.CFL(solver, initial_dt=dt, cadence=1, safety=1,
                            max_change=1.5, min_change=0.01,
                            max_dt=0.01,    min_dt=1e-10)
else:                       # Looser control flow (faster but risky).
    cfl = flow_tools.CFL(solver, initial_dt=dt, cadence=10, safety=1,
                            max_change=1.5, min_change=0.5,
                            max_dt=0.01,    min_dt=1e-6)
cfl.add_velocities(('v',  'w' ))
cfl.add_velocities(('v_', 'w_'))

# Flow properties (print during run; not recorded in the records files)
flow = flow_tools.GlobalFlowProperty(solver, cadence=1)
flow.add_property("sqrt(v **2 + w **2) / Ra", name='Re' )
flow.add_property("sqrt(v_**2 + w_**2) / Ra", name='Re_')

# MAIN COMPUTATION LOOP -----------------------------------------------
try:
    self.logger.info("Starting main loop")
    start_time = time.time()
    while solver.ok:
        # Recompute time step and iterate.
        dt = cfl.compute_dt()
        dt = solver.step(dt)

        # Print info to the screen every 10 iterations.
        if solver.iteration % 10 == 0:
            info = "Iteration {:>5d}, Time: {:.7f}, dt: {:.2e}".format(
                                solver.iteration, solver.sim_time, dt)
            Re  = flow.max("Re")
            Re_ = flow.max("Re_")
            info += ", Max Re = {:f}".format(Re)
            info += ", Max Re_ = {:f}".format(Re_)
            self.logger.info(info)
            # Make sure the simulation hasn't blown up.
            if np.isnan(Re) or np.isnan(Re_):
                raise ValueError("Reynolds number went to infinity!!"
```


المنارة للاستشارات
www.manaraa.com

```
                                    "\nRe = {}, Re_ = {}".format(Re, Re_))
        except BaseException as e:
            self.logger.error("Exception raised, triggering end of main loop.")
            raise
        finally:
            total_time = time.time() - start_time
            cpu_hr = total_time / 60 / 60 * SIZE
            self.logger.info("Iterations: {:d}".format(solver.iteration))
            self.logger.info("Sim end time: {:.3e}".format(solver.sim_time))
            self.logger.info("Run time: {:.3e} sec".format(total_time))
            self.logger.info("Run time: {:.3e} cpu-hr".format(cpu_hr))
            self.logger.debug("END OF SIMULATION\n")
```

`data_assimilation.py`. Parses command line arguments and runs the corresponding simulation. This is the main file that the user runs to do numerical experiments.

```
# data_assimilation.py
"""Class-based Dedalus script on Data Assimilation for Mantle Convection.

usage: data_assimilation.py -h
       data_assimilation.py --test
       data_assimilation.py --main [-Ra RAYLEIGH] [-Pr PRANDTL] [-N N]
                            [-m MU] [-t TIME] [-init INITIAL] [-s SAVE]
                            [--tight] [--no-anlaysis]
       data_assimilation.py --main --movie [-Ra RAYLEIGH] [-Pr PRANDTL] [-N N]
                            [-m MU] [-t TIME] [-init INITIAL] [--tight]

Run a Rayleigh-Benard convection simulation in 2D using the Boussinesq
equations; assimilate the data from that system into another system that
starts from a Fourier projection of the initial data.

optional arguments:
  -h, --help            show this help message and exit
  --main                run a full simulation with specified parameters
  --test                run a short test with default parameters
  -Ra RAYLEIGH, --Rayleigh RAYLEIGH
                        Rayleigh number (conduction vs convection)
  -Pr PRANDTL, --Prandtl PRANDTL
                        Prandtl number (viscous vs thermal diffusion)
  -N N, --N N           number of Fourier/Chebyshev projection modes
  -m MU, --mu MU        relaxation parameter for projection coupling
  -t TIME, --time TIME  simulation time
  -init INITIAL, --initial INITIAL
                        data file to use as initial conditions
  -s SAVE, --save SAVE  interval at which to save the simulation state
  --tight               simulate with smaller time steps than usual
  --no-analysis         simulate without convergence measurements
  --movie               indicate that this is a movie-making experiment

Author: Shane McQuarrie
"""

import argparse
import numpy as np

from boussinesq2d import (BoussinesqDataAssimilation2D,
                          BoussinesqDataAssimilation2Dmovie)

# Global variables ------------------------------------------------------------
```


المنارة للاستشارات
www.manaraa.com

```
DEFAULT_N = 32
DEFAULT_MU = 20000
TEST_DIR = "__TEST__"

# Folder management tools -----------------------------------------------------

def folder_name(Rayleigh, mu=None, N=None, Prandtl=None, movie=False):
    """Get the name of the folder for the specified simulation parameters."""
    # Always label with Rayleigh number.
    label = "RA_{:0>10}".format(Rayleigh)

    # Label by mu, N, Prandlt, and movie next.
    if N and N != DEFAULT_N:
        label += "_N_{:0>2}".format(N)
    if mu and mu != DEFAULT_MU:
        label += "_{:0>6}".format(mu)
    if Prandtl:
        label += "_PR_{:0>3}".format(Prandtl)
    if movie:
        label += "_movie"

    return label

# Command line argument parser ------------------------------------------------

parser = argparse.ArgumentParser(description="Run a Rayleigh-Benard "
                "convection simulation in 2D using the Boussinesq equations; "
                "assimilate the data from that system into another system "
                "that starts from a Fourier projection of the initial data.")
parser.usage = """data_assimilation.py -h
       data_assimilation.py --test
       data_assimilation.py --main [-Ra RAYLEIGH] [-Pr PRANDTL] [-N N]
                            [-m MU] [-t TIME] [-init INITIAL] [-s SAVE]
                            [--tight] [--no-anlaysis]
       data_assimilation.py --main --movie [-Ra RAYLEIGH] [-Pr PRANDTL] [-N N]
                            [-m MU] [-t TIME] [-init INITIAL] [--tight]
"""

# Mutually exclusive group: --main and --test
group1 = parser.add_mutually_exclusive_group()
group1.add_argument("--main", action="store_true",
                    help="run a full simulation with specified parameters")
group1.add_argument("--test", action="store_true",
                    help="run a short test with default parameters")

parser.add_argument("-Ra", "--Rayleigh", type=int, default=10000,
                    help="Rayleigh number (conduction vs convection)")
parser.add_argument("-Pr", "--Prandtl", type=int,
                    help="Prandtl number (viscous vs thermal diffusion)")
parser.add_argument("-N", "--N", type=int, default=DEFAULT_N,
                    help="number of Fourier/Chebyshev projection modes")
parser.add_argument("-m", "--mu", type=int, default=DEFAULT_MU,
                    help="relaxation parameter for projection coupling")
parser.add_argument("-t", "--time", type=float, default=1.0,
                    help="simulation time")
parser.add_argument("-init", "--initial",
                    help="data file to use as initial conditions")

# Mutually exclusive group: --movie and --save
group2 = parser.add_mutually_exclusive_group()
group2.add_argument("-s", "--save", type=float, default=.05,
                    help="interval at which to save the simulation state")
parser.add_argument("--tight", action="store_true",
                    help="simulate with smaller time steps than usual")
parser.add_argument("--no-analysis", action="store_true", default=False,
                    help="simulate without convergence measurements")
```


www.manaraa.com

```python
group2.add_argument("--movie", action="store_true", default=False,
                    help="indicate that this is a movie-making experiment")

# Main Routine ================================================================

if __name__ == "__main__":
    args = parser.parse_args()

    # Run a full simulation ---------------------------------------------------
    if args.main:
        label = folder_name(args.Rayleigh, args.mu,
                            args.N, args.Prandtl, args.movie)
        if args.movie:
            b2d = BoussinesqDataAssimilation2Dmovie(label)
            b2d.setup(Rayleigh=args.Rayleigh, mu=args.mu, N=args.N,
                      Prandtl=args.Prandtl)
            b2d.simulate(initial_conditions=args.initial,
                         sim_time=args.time, tight=args.tight)
        else:
            b2d = BoussinesqDataAssimilation2D(label)
            b2d.setup(Rayleigh=args.Rayleigh, mu=args.mu, N=args.N,
                      Prandtl=args.Prandtl)
            b2d.simulate(initial_conditions=args.initial,
                         sim_time=args.time, tight=args.tight,
                         save=args.save, analysis=not args.no_analysis)

    # Run a short test with default parameters --------------------------------
    elif args.test:
        b2d = BoussinesqDataAssimilation2D(TEST_DIR)
        b2d.setup(N=8)
        try:
            b2d.simulate(initial_conditions="default.h5", sim_time=.1501,
                         save=.0001)
        except KeyboardInterrupt:
            pass
        b2d.process_results()
```

`merge.py`. Parses command line arguments and merges the data generated by a completed experiment. This script also triggers visualization routines for the data.

```python
# merge.py
"""Merge existing analysis and state files for a single simulation directory.

usage: merge.py -h
       merge.py -d DIRECTORY
       merge.py -Ra RAYLEIGH [-m MU] [-Pr PRANDTL] [-N N] [--movie]

optional arguments:
  -h, --help            show this help message and exit
  -d DIRECTORY, --directory DIRECTORY
                        folder containing simulation files to merge
  -Ra RAYLEIGH, --Rayleigh RAYLEIGH
                        Rayleigh number (conduction vs convection)
  -m MU, --mu MU        relaxation parameter for projection coupling
  -Pr PRANDTL, --Prandtl PRANDTL
                        Prandtl number (viscous vs thermal diffusion)
  -N N, --N N           number of Fourier/Chebyshev projection modes
  --movie               indicate that this is a movie-making experiment
"""
```


www.manaraa.com

```python
import os
import argparse

from boussinesq2d import BoussinesqDataAssimilation2D
from data_assimilation import folder_name
from base_simulator import RANK


# Parse command line arguments -----------------------------------------------

parser = argparse.ArgumentParser(description="Merge existing analysis and "
                            "state files for a single simulation directory")
parser.usage = """merge.py -h
       merge.py -d DIRECTORY
       merge.py -Ra RAYLEIGH [-m MU] [-Pr PRANDTL] [-N N] [--movie]
"""

# mutually exclusive group: --directory, -Ra
group = parser.add_mutually_exclusive_group()
group.add_argument("-d", "--directory",
                    help="folder containing simulation files to merge")
group.add_argument("-Ra", "--Rayleigh", type=int, default=10000,
                    help="Rayleigh number (conduction vs convection)")
# Other arguments used if --directory is not specified
parser.add_argument("-m", "--mu", type=int,
                    help="relaxation parameter for projection coupling")
parser.add_argument("-Pr", "--Prandtl", type=int,
                    help="Prandtl number (viscous vs thermal diffusion)")
parser.add_argument("-N", "--N", type=int,
                    help="number of Fourier/Chebyshev projection modes")
parser.add_argument("--movie", action="store_true", default=False,
                    help="indicate that this is a movie-making experiment")

args = parser.parse_args()

# Get the name of the directory to merge -------------------------------------
if args.directory:
    label = args.directory
else:
    label = folder_name(Rayleigh=args.Rayleigh, mu=args.mu, N=args.N,
                        Prandtl=args.Prandtl, movie=args.movie)

if not os.path.isdir(label):
    raise NotADirectoryError(label)

# Run the actual merge.
b2d = BoussinesqDataAssimilation2D(label)
b2d.merge_results(force=True)

# Process the merged data (only on the root process).
if RANK == 0:
    if os.path.isfile(os.path.join(label, "analysis", "analysis.h5")):
        b2d.plot_convergence()
        b2d.plot_nusselt()
    if os.path.isfile(os.path.join(label, "states", "states.h5")):
        b2d.animate_temperature()
        b2d.animate_clusters()
```

`plot_tools.py`. Parses command line arguments and creates plots like Figures 3.4, 3.6, and 4.2.

المنارة للاستشارات
www.manaraa.com

```python
# plot_tools.py
"""Functions for extracting data to plot, etc."""

import os
import h5py
import argparse
import numpy as np
from itertools import cycle
from matplotlib import pyplot as plt

from base_simulator import BaseSimulator

plt.style.use(".mplstyle")

# Helper Functions ============================================================

def get_data(dirname, task):
    """Get the specified taks from <dirname>/analysis/analysis.h5

    Returns:
        ((n,) ndarray): sim_time
        ((n,) ndarray): task
    """
    if task == "sum_norm":
        return get_sum_norm(dirname)
    # Make sure the directory and corresponding analysis file exist.
    if not os.path.isdir(dirname):
        raise NotADirectoryError(dirname)
    target = os.path.join(dirname, "analysis", "analysis.h5")
    if not os.path.isfile(target):
        raise FileNotFoundError(target)
    try:
        with h5py.File(target, 'r') as datafile:
            return np.array(datafile["scales/sim_time"]), datafile[task][:,0,0]
    except KeyError:
        print("\nMissing data in {}: {}".format(dirname, task), end='\n\n')
        raise

def get_label(dirname, label):
    """Strip the label of interest out of a directory name. For example, if
    dirname == "RA_123_N_08_456",
        label=="RA" -> return 123
        label=="N"  -> return 8
        label=="mu" -> return 456
    For label=="RA", also put it in scientific notation.
    """
    try:
        out = BaseSimulator._load_params(dirname)[label]
    except KeyError:
        if label == "Pr":
            out = "\infty"
        else:
            raise
    out = "{:.2e}".format(out) if label == "Ra" else out
    return r"${} = {}$".format(r"\mu" if label == "mu" else label, out)

def get_sum_norm(dirname):
    """Extract ||u_ - u||_H2 + ||T_ - T||H2."""
    simtime1, T_h2_err = get_data(dirname, "tasks/T_h2_err")
    simtime2, u_h2_err = get_data(dirname, "tasks/u_h2_err")
    assert np.allclose(simtime1, simtime2)
    assert T_h2_err.shape == u_h2_err.shape
    return simtime1, T_h2_err + u_h2_err

# Main Routine ================================================================
```


www.manaraa.com

```python
def make_plot(dirs, task, variable, outfile, xmax=.01, ymin=1e-11, title=None):
    """Plot the task from the given directories.

    Parameters:
        dirs (list(str)): directories to pull data from.
        task (str): which item to search for in each analysis.h5 file.
        variable (str): the kind of labels to use in the plot legend.
        outfile (str): the name of the resulting figure file.
        xmax (float): upper bound for the x-axis of the plot.
        ymin (float): lower bound for the y-axis of the plot.
        title (str): title for the plot.
    """
    print("Plotting {} with label {} from".format(task, variable))
    print('\t' + "\n\t".join(dirs))
    fig,ax = plt.subplots(1, 1)
    data, missing = [], []

    # Get the data from each directory and record errors.
    for d in dirs:
        try:
            data.append(get_data(d, task))
        except FileNotFoundError:
            missing.append(d)
            dirs.remove(d)
    if not dirs:
        raise FileNotFoundError(missing)
    elif missing:
        print("MISSING FILES:\n\t", "\n\t".join(missing))

    # Plot the data over [0, xmax].
    for [t,y],d,style in zip(data, dirs, cycle(['-', '--', '-.', ':'])):
        ax.semilogy(t, y, style, lw=1, label=get_label(d, variable))
    ax.set_xlim(0, xmax)
    ax.set_ylim(ymin, 1e1)
    ax.set_xlabel(r"Simulation Time $t$")
    ax.legend(loc="lower left", ncol=2, fontsize=8)
    if title:
        ax.set_title(r"${}$".format(title))
    plt.tight_layout()
    plt.savefig(outfile, dpi=300)
    print("Figure saved as", outfile)


if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("TASK",
                        help="which task to plot (e.g. tasks/T_err, sum_norm)")
    parser.add_argument("LABEL",
                        help="the variable to use for a label (Ra, mu, or N)")
    parser.add_argument("DIR", nargs='+',
                        help="directories to pull data from")
    parser.add_argument("--outfile", default="TEST.png",
                        help="name of the file to save")
    parser.add_argument("--xmax", type=float, default=.01,
                        help="upper bound of x-axis on plot")
    parser.add_argument("--ymin", type=float, default=1e-11,
                        help="lower bound of y-axis on plot")
    parser.add_argument("--title", default=None,
                        help="the figure title (if any)")
    args = parser.parse_args()

    make_plot(args.DIR, args.TASK, args.LABEL,
              args.outfile, args.xmax, args.ymin, args.title)
```


المنارة للاستشارات
www.manaraa.com

`.mplstyle.` Style for `matplotlib` plots; used in the visualization routines in `boussinesq2d.py` and `plot_tools.py`.

```
### Styles for plots.

## Text
text.color : DarkSlateGrey

## Axes
axes.edgecolor : grey
axes.spines.top : False
axes.spines.right : False
axes.labelcolor : DarkSlateGrey
axes.labelpad : 8.0
axes.titlesize : x-large
axes.titlepad : 16.0

## Ticks
xtick.color : DarkSlateGrey
ytick.color : DarkSlateGrey

## Figures
figure.titlesize : x-large
figure.figsize : 5, 3

savefig.dpi : 300
savefig.format : pdf
savefig.bbox : tight
```


المنارة للاستشارات
www.manaraa.com

Bibliography

[1] G. Ahlers, S. Grossmann, and D. Lohse, *Heat transfer and large scale dynamics in turbulent Rayleigh-Bénard convection*, Reviews of modern physics, 81 (2009), p. 503.

[2] M. Altaf, E. Titi, T. Gebrael, O. Knio, L. Zhao, M. McCabe, and I. Hoteit, *Downscaling the 2d Bénard convection equations using continuous data assimilation*, Computational Geosciences, 21 (2017), pp. 393–410.

[3] P. S. Bernard, *Fluid Dynamics*, Cambridge University Press, 2015.

[4] B. Blankenbach, F. Busse, U. Christensen, L. Cserepes, D. Gunkel, U. Hansen, H. Harder, G. Jarvis, M. Koch, G. Marquart, et al., *A benchmark comparison for mantle convection codes*, Geophysical Journal International, 98 (1989), pp. 23–38.

[5] J. Boussinesq, *Théorie de L'écoulement Tourbillonnant et Tumultueux des Liquides Dans Les Lits Rectilignes à Grande Section*, vol. 1, Gauthier-Villars, 1897.

[6] K. J. Burns, G. M. Vasil, J. S. Oishi, D. Lecoanet, and B. Brown, *Dedalus: Flexible framework for spectrally solving differential equations.* Astrophysics Source Code Library, Mar. 2016.

[7] R. P. Chhabra and J. F. Richardson, *Non-Newtonian Flow: Fundamentals and Engineering Applications*, Butterworth-Heinemann, 1999.

[8] S. Childress, *An Introduction to Theoretical Fluid Mechanics*, vol. 19, American Mathematical Society, 2009.

[9] A. J. Chorin and J. E. Marsden, *A Mathematical Introduction to Fluid Mechanics*, vol. 4, Springer-Verkag, 2013.

[10] P. Constantin and C. Foias, *Navier-Stokes equations*, Chicago Lectures in Mathematics, University of Chicago Press, Chicago, IL, 1988.

[11] T. Dhert, T. Ashuri, and J. R. Martins, *Aerodynamic shape optimization of wind turbine blades using a Reynolds-averaged Navier–Stokes model and an adjoint method*, Wind Energy, 20 (2017), pp. 909–926.

[12] L. C. Evans, *Partial Differential Equations*, vol. 19 of Graduate Studies in Mathematics, American Mathematical Society, 2010.

[13] A. Farhat, H. Johnston, M. S. Jolly, and E. S. Titi, *Assimilation of nearly turbulent Rayleigh-Bénard flow through vorticity or local circulation measurements: a computational study*, arXiv preprint arXiv:1709.02417, (2017).

[14] A. Farhat, M. Jolly, and E. Titi, *Continuous data assimilation for the 2d Bénard convection through velocity measurements alone*, Phys. D, 303 (2015), pp. 59–66.

[15] W. Graebel, *Advanced Fluid Mechanics*, Academic Press, 2007.

المنارة للاستشارات
www.manaraa.com

[16] J. Humpherys, T. J. Jarvis, and E. J. Evans, *Foundations of Applied Mathematics Volume 1: Mathematical Analysis*, Society for Industrial and Applied Mathematics, 2017.

[17] M. Jolly, V. Martinez, S. Sadigov, and E. Titi, *A determining form for the subcritical surface quasi-geostrophic equation*, J. Dyn. Differ. Equations, (2018), pp. 1–38.

[18] L. Kondic, P. Palffy-Muhoray, and M. J. Shelley, *Models of non-newtonian Hele-Shaw flow*, Physical Review E, 54 (1996), p. R4536.

[19] K. Law, A. Stuart, and K. Zygalakis, *Data Assimilation: A Mathematical Introduction*, vol. 62, Springer, 2015.

[20] R. J. LeVeque, D. L. George, and M. J. Berger, *Tsunami modelling with adaptively refined finite volume methods*, Acta Numerica, 20 (2011), pp. 211–289.

[21] L. Rayleigh, *On convection currents in a horizontal layer of fluid, when the higher temperature is on the under side*, The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, 32 (1916), pp. 529–546.

[22] C. Sotin and S. Labrosse, *Three-dimensional thermal convection in an iso-viscous, infinite Prandtl number fluid heated from within and from below: applications to the transfer of heat through planetary mantles*, Physics of the Earth and Planetary Interiors, 112 (1999), pp. 171–190.

[23] R. Temam, *Infinite-Dimensional Dynamical Systems in Mechanics and Physics*, vol. 68 of Applied Mathematical Sciences, Springer-Verlag, New York, 1997.

[24] D. Tritton, *Physical Fluid Dynamics*, Clarendon Press, 1988.

[25] D. Turcotte and G. Schubert, *Geodynamics*, Cambridge University Press, 2014.

[26] X. Wang, *A note on long time behavior of solutions to the Boussinesq system at large Prandtl number*, Contemp. Math., 371 (2005), pp. 315–323.

المنارة للاستشارات
www.manaraa.com