Numerical Methods for Hamilton-Jacobi-Bellman Equations with Applications

by

Yangang Chen

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Applied Mathematics

Waterloo, Ontario, Canada, 2019

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

Supervisor:	Justin W. L. Wan David R. Cheriton School of Computer Science, University of Waterloo
Internal Member:	Lilia Krivodonova Department of Applied Mathematics, University of Waterloo
	Sander Rhebergen Department of Applied Mathematics, University of Waterloo
Internal-External Member:	Yuying Li David R. Cheriton School of Computer Science, University of Waterloo
External Examiner:	Kenneth R. Jackson Department of Computer Science, University of Toronto

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Hamilton-Jacobi-Bellman (HJB) equations are nonlinear controlled partial differential equations (PDEs). In this thesis, we propose various numerical methods for HJB equations arising from three specific applications.

First, we study numerical methods for the HJB equation coupled with a Kolmogorov-Fokker-Planck (KFP) equation arising from mean field games. In order to solve the nonlinear discretized systems efficiently, we propose a multigrid method. The main novelty of our approach is that we subtract artificial viscosity from the direct discretization coarse grid operators, such that the coarse grid error estimations are more accurate. The convergence rate of the proposed multigrid method is mesh-independent and faster than the existing methods in the literature.

Next, we investigate numerical methods for the HJB formulation that arises from the mass transport image registration model. We convert the PDE of the model (a Monge-Ampère equation) to an equivalent HJB equation, propose a monotone mixed discretization, and prove that it is guaranteed to converge to the viscosity solution. Then we propose multigrid methods for the mixed discretization, where we set wide stencil points as coarse grid points, use injection at wide stencil points as the restriction, and achieve a mesh-independent convergence rate. Moreover, we propose a novel periodic boundary condition for the image registration PDE, such that when two images are related by a combination of a translation and a non-rigid deformation, the numerical scheme recovers the underlying transformation correctly.

Finally, we propose a deep neural network framework for the HJB equations emerging from the study of American options in high dimensions. We convert the HJB equation to an equivalent Backward Stochastic Differential Equation (BSDE), introduce the least squares residual of the BSDE as the loss function, and propose a new neural network architecture that utilizes the domain knowledge of American options. Our proposed framework yields American option prices and deltas on the entire spacetime, not only at a given point. The computational cost of the proposed approach is quadratic in dimension, which addresses the curse of dimensionality issue that state-of-the-art approaches suffer.

Acknowledgements

First and foremost, I thank my PhD advisor, Justin W.L. Wan, for his outstanding mentorship on my doctoral research, for his profound guidance on developing my intellectual and interpersonal skills, and for his unreserved support during some of the most difficult times in my life.

I thank my PhD thesis committee members, Lilia Krivodonova, Sander Rhebergen, Yuying Li, and Kenneth R. Jackson, for their time, their roles, and their insightful comments.

I thank my mentors outside my PhD program, including Eugene Wen, Dragos Daniel Capan and Guifre Vidal, for their enlightenment on my career path.

I thank Colin Boucher, for his mind of brilliance, empathy and beauty, and for his significant role in my life, which I hold dearest to my heart.

I thank Jason Pye, Alex Preciado, Yasaman Yazdi, Anton Van Niekerk, Mansour Karami, for their deeply cherished friendship.

I thank Kempf's lab for giving me so much sunshine. I thank my office-mates and Sander's group for being great fellows. I thank my curling team for winning the league championship together. I thank my CS 475 students for fulfilling my dream as a university lecturer.

I thank all the amazing people I have met during the past few years, for their supports, laughter and tears.

I thank the chapter of my Waterloo life, for making me proud of what I have accomplished and who I am.

Finally, I thank my family, especially my mom, the most loving and courageous woman, and my dad, the most diligent and righteous man, for their unconditional love.

Dedication

To my parents and my grandfather.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Hamilton-Jacobi-Bellman (HJB) Equations

The objective of this thesis is to propose numerical methods for solving Hamilton-Jacobi-Bellman (HJB) equations. An HJB equation is a nonlinear controlled partial differential equation (PDE). HJB equations usually arise from optimal control theory. More specifically, consider a dynamical system in a continuous spacetime, where a controller, starting at a **state** $(\boldsymbol{x}, t)$, controls the future evolution of the state through a **control variable** $\boldsymbol{c}$, and aims to optimize its cumulative **objective function** over the prospective trajectory. For instance, consider the dynamical system of a competitive smartphone market. Each company (controller) sets its smart phone price (control variable) based on its current capacity (state) in order to maximize its long-term profit (objective function).

In such a dynamical system, we are particularly interested in two quantities. One is the optimal objective function, called the **value function** $u(\boldsymbol{x}, t)$. The other is the control variable that optimizes the objective function, called the **optimal control** $\boldsymbol{c}^*(\boldsymbol{x}, t)$. An HJB equation models such dynamical system by coupling the value function $u(\boldsymbol{x}, t)$ and the optimal control $\boldsymbol{c}^*(\boldsymbol{x}, t)$ as follows:

$$\mathcal{L}_{\boldsymbol{c}^*(\boldsymbol{x},t)}\, u(\boldsymbol{x}, t) = 0, \tag{1.1}$$

$$\text{subject to} \quad \boldsymbol{c}^*(\boldsymbol{x}, t) \equiv \arg\max_{\boldsymbol{c}(\boldsymbol{x},t)} H(\boldsymbol{x}, t; \boldsymbol{c}(\boldsymbol{x}, t); u(\boldsymbol{x}, t)). \tag{1.2}$$

Here $\mathcal{L}$ is a second order differential operator on $u(\boldsymbol{x}, t)$ where $\boldsymbol{c}^*(\boldsymbol{x}, t)$ is treated as a

parameter (i.e., contained in coefficients), for example,

$$\mathcal{L}_{\boldsymbol{c}^*(\boldsymbol{x},t)}\, u(\boldsymbol{x},t) = u_t(\boldsymbol{x},t) - \Delta u(\boldsymbol{x},t) + \boldsymbol{c}^*(\boldsymbol{x},t)\cdot\nabla u(\boldsymbol{x},t) + ru(\boldsymbol{x},t) - \frac{\boldsymbol{c}^*(\boldsymbol{x},t)^2}{2} - f(\boldsymbol{x},t);$$

and H is called the Hamiltonian, where optimizing the Hamiltonian as in (1.2) is equivalent to optimizing the objective function under an optimization method called dynamic programming [18]. An HJB equation couples two sub-problems. One sub-problem is a PDE (1.1), where the solution is the value function $u(\boldsymbol{x},t)$ given the optimal control $\boldsymbol{c}^*(\boldsymbol{x},t)$; the other sub-problem is an optimization problem (1.2) with respect to the control variable $\boldsymbol{c}(\boldsymbol{x},t)$ given the value function $u(\boldsymbol{x},t)$. Solving an HJB equation would yield both the value function $u(\boldsymbol{x},t)$ and the optimal control $\boldsymbol{c}^*(\boldsymbol{x},t)$ simultaneously.

1.2 Applications

The study of HJB equations has been a field of great interest due to a wide range of applications [18, 70, 30, 109, 110, 111, 66, 67, 15, 106]. In this thesis, we focus on three important applications: mean field games, American options and image registration.

1.2.1 Mean field games

One of the most well-known applications of HJB equations is in game theory in continuous spacetime [70, 30]. The competitive smartphone market mentioned in Section 1.1 is a typical example of game theory in continuous spacetime.

In particular, this thesis is concerned with a subfield called mean field games [109, 110, 111]. To motivate mean field games, consider the competitive smartphone market with N companies. Each company is modelled by one HJB equation that solves for its value function. As a result, N companies can be modelled by a PDE system with N HJB equations. Since each company tries to win the competition by adjusting its own control in response to the other companies' controls, the N HJB equations are coupled. When N is large, the PDE system becomes extremely complicated.

Fortunately, when $N \to \infty$, the model can be simplified. Each company's impact on the entire system is negligible. Also, since it is difficult to keep track of every single opponent's control, companies respond to each other's control in a statistical sense. As a result, an N-player game can be reduced to a less complicated model, called a "mean field game". Mathematically, a mean field game is modelled by a system of nonlinear

PDEs that contains two equations. One equation is an **HJB equation** for the optimal value function of all the players. The other equation is a **Kolmogorov-Fokker-Planck (KFP) equation** for the distribution (or density function) of the players' states. It is shown in [111, 83, 42] that a mean field game model yields a good approximation of the original N-HJB model when N is large.

There are numerous applications of mean field games, including, but not limited to, micro or macro economics, sociology, engineering, urban planning, etc. [83, 56, 42] We refer interested readers to [109, 110, 111, 83, 42] for an extensive introduction to mean field games and the associated HJB equations.

1.2.2 American options

Another important application of HJB equations is to American option problems in finance. An American option [97, 58, 67, 65] is a financial contract that gives a contract holder the right, but not the obligation, to buy or sell underlying assets (such as stocks, commodities, foreign currencies) at a preset price (called the strike price) at any time before the expiry of the contract. The action of buying or selling the underlying assets at the preset price is called "exercising (the right of) an American option." An option holder would make the best exercise decision during the life of the option contract in order to maximize the gain. The expected maximal gain of an option holder is called the price of the American option. American options are among the most common contracts in financial markets.

Derived from the famous Black-Scholes model [97], there are multiple mathematical formulations for American options, including HJB formulations, linear complementarity problems, free boundary problems, Monte Carlo formulations, variational formulations, etc. We refer readers to [58, 139] for a substantial review of different formulations.

In this thesis, we study the **HJB formulation** [67, 66, 131]. Indeed, the HJB formulation is very suitable for modeling American options. More specifically, the price of an American option (i.e., the expected maximal gain) can be naturally formulated as the value function of an HJB equation; meanwhile, the optimal exercise decision that maximizes the gain can be naturally formulated as the optimal control of an HJB equation. The HJB formulation couples both the option price and the optimal exercise decision together, and solving it would yield both quantities simultaneously.

1.2.3 Image registration

In many practical applications, one has to compare two images, T (template) and R (reference), that display the same object, but the object inside the two images is not spatially aligned. An image registration problem is the task of finding a coordinate transformation ϕ that transforms the image T to another image T_ϕ, such that T_ϕ is close and thus comparable to the image R.

One important application of image registration is to compare medical images of the same patient, such as CT (computed tomography) and MRI (magnetic resonance imaging) images of a damaged brain, which gives guidance for diagnosis and surgery [119, 92]. Image registration can also be used for image fusion [98]. That is, multiple images of the same object are taken, registered and then merged together, such that the integrated image is clearer than the original ones. We refer readers to [81] for more discussion on applications of image registration.

In [85, 86, 123], the coordinate transformation between the template and reference images is obtained by solving a nonlinear PDE called the **Monge-Ampère equation**. Significantly, a Monge-Ampère equation is equivalent to an **HJB equation**, as established by [105] and [116]. Hence, we can solve the image registration problem by solving the HJB formulation of the Monge-Ampère equation. It turns out that solving the HJB formulation has several advantages over solving the original Monge-Ampère equation. One is that the Monge-Ampère equation contains a convexity constraint, which is difficult to handle. However, the convexity constraint can be removed in the equivalent HJB formulation. The other is that the differential operator of the HJB formulation turns out to be linear under a given control variable. These properties of the HJB formulation make numerical solution more manageable.

1.3 Numerical Methods

An HJB equation (1.1)-(1.2) is a second-order nonlinear PDE. Finding analytical solutions is usually impossible. Hence, HJB equations are typically solved by numerical methods. There exist numerous numerical methods for HJB equations, including but not limited to [66, 95, 55, 118, 25, 12, 11, 155]. However, there still exist a number of challenges for numerical solutions of HJB equations. In this thesis, we propose several numerical methods for addressing the existing challenges. Here we give an overview of these numerical methods.

1.3.1 Discretization

The standard approach to solving a PDE numerically is to discretize it using finite difference, finite volume, finite element methods, etc., and then to solve the discretized system. In this thesis, we consider finite difference discretization, since it is relatively easier to implement and analyze compared to the other approaches, and the spatial domains of the applications we consider are hypercubic.

Similar to other nonlinear PDEs, an HJB equation may have multiple weak solutions, among which only one solution, called the **viscosity solution** [53, 52], is the correct solution in practical applications. Finding the viscosity solution of an HJB equation numerically is challenging. The choice of the discretization scheme turns out to play a significant role in obtaining the viscosity solution. More specifically, it is desirable for a discretization to satisfy

- **consistency**: the truncation error of a discretized equation approaches zero as the mesh size $h \to 0$ (i.e., a discretized equation of a PDE approaches the PDE as $h \to 0$);
- **stability**: the solution of a discretized system is bounded;
- **monotonicity**: the discretized equation at a grid point $\boldsymbol{x}_i$ is non-decreasing in $u(\boldsymbol{x}_i)$ and non-increasing in $\{u(\boldsymbol{x}_j) \,|\, j \neq i\}$;
- **strong comparison principle**: a subsolution is no greater than a supersolution on the entire domain (including the boundary).

If all these properties are fulfilled, then the Barles-Souganidis Theorem [14] guarantees the **convergence** of the discrete solution to the continuous viscosity solution as $h \to 0$. In this thesis, we will design discretization schemes for HJB equations (especially for image registration problems) that satisfy these properties.

1.3.2 Policy iteration

Discretization of an HJB equation leads to a nonlinear discretized system. There are two major difficulties in solving such a system. One is that, in an HJB equation, the solution $u(\boldsymbol{x}, t)$ and the optimal control $\boldsymbol{c}^*(\boldsymbol{x}, t)$ are coupled in a non-trivial fashion. In order to compute the solution, both quantities must be obtained simultaneously. The other difficulty is that the nonlinearity of the discretized HJB equation requires nonlinear iterative solvers. In general, a nonlinear iterative solver is not necessarily convergent.

We can take advantage of the feature that an HJB equation consists of two sub-problems:

- fixing the control variables $\boldsymbol{c}^*$ and solving the PDE (1.1) for the solution u;
- fixing the solution u and solving the optimization problem (1.2) for the optimal control $\boldsymbol{c}^*$.

We note that these sub-problems can be solved numerically on a discretized grid. Iterating between solving these two sub-problems gives rise to a nonlinear solver for discretized HJB equations, called **policy iteration** (or Howard's algorithm) [95, 66].

To be more concrete, we put the discretized solution and discretized optimal control under lexicographic order into vectors, denoted as u_h and $\boldsymbol{c}_h^*$, respectively. Typically, the discretization of (1.1) and (1.2) can be written in the following matrix form:

$$A_h(\boldsymbol{c}_h^*)\, u_h = b_h(\boldsymbol{c}_h^*), \tag{1.3}$$

$$\text{subject to} \quad \boldsymbol{c}_h^* \equiv \arg\max_{\boldsymbol{c}_h} H(\boldsymbol{c}_h, u_h), \tag{1.4}$$

respectively. Here A_h and b_h are the matrix and the right hand side vector arising from the discretization of the differential operator $\mathcal{L}_{\boldsymbol{c}^*}$, and thus depend on $\boldsymbol{c}_h^*$. We note that when the discretized optimal control $\boldsymbol{c}_h^*$ is fixed, the sub-problem (1.3) is a linear system with respect to u_h. Then the policy iteration can be described as Algorithm 1.1:

Algorithm 1.1 Policy iteration for solving discretized HJB equation (1.3)-(1.4)

1: Start with an initial guess of the control variable $\boldsymbol{c}_h^{(0)}$.
2: **for** $k = 1, 2, \ldots$ until convergence **do**
3: Solve the linearized HJB equation $A_h(\boldsymbol{c}_h^{(k-1)})\, u_h^{(k)} = b_h(\boldsymbol{c}_h^{(k-1)})$ for the solution $u_h^{(k)}$.
4: Solve the optimization problem $\boldsymbol{c}_h^{(k)} \equiv \arg\max_{\boldsymbol{c}_h} H(\boldsymbol{c}_h, u_h^{(k)})$ for the control $\boldsymbol{c}_h^{(k)}$.
5: **end for**
6: Convergent solution: $u_h = u_h^{(k)}$, $\boldsymbol{c}_h^* = \boldsymbol{c}_h^{(k)}$.

Significantly, policy iteration is guaranteed to converge for any initial guess of the solution (or the control), if an HJB equation is discretized by a monotone scheme and the resulting discretized matrix A_h is an M-matrix under all admissible controls [25, 12, 11]. Once policy iteration converges, we obtain both the solution u and the optimal control $\boldsymbol{c}^*$ on the discretized grid simultaneously.

1.3.3 Multigrid methods

One of the major challenges of solving a discretized HJB system is the computational cost. Consider using policy iteration, where each iteration requires solving a linear system. One may consider using iterative solvers, such as the preconditioned conjugate gradient method or the Gauss-Seidel iteration. However, the convergence rate (i.e., number of iterations for convergence) grows as $h \to 0$, which is inefficient for problems with fine meshes.

Multigrid methods [153] are the optimally efficient iterative solvers for many elliptic and parabolic problems. More specifically, the convergence rates of multigrid methods are independent of the mesh size and remain approximately constant with mesh refinement. In general, HJB equations are elliptic or parabolic, which suggests that multigrid methods can potentially achieve mesh-independent convergence rates for HJB equations. Hence, in this thesis, we will develop multigrid methods for HJB equations.

To introduce the basic idea of multigrid methods, consider a linear system arising from discretization of an elliptic PDE

$$A_h u_h = b_h. \tag{1.5}$$

Consider solving the linear system using relaxation methods (i.e., Gauss-Seidel iterations, Jacobi iterations, etc.). In general, relaxation methods are slow to converge, or in other words, a few steps of relaxation is far from sufficient for solving the linear system. However, due to the ellipticity, a few steps of relaxation is sufficient to make the error of the approximate solution smooth. Significantly, such smoothness allows the error to be estimated on coarse grids accurately. Meanwhile, estimating the error on coarse grids can be substantially cheaper than on the original fine grid. Motivated by these observations, we can design an iterative solver for the linear system, where each iteration combines the substeps of **smoothing the error** and estimating error on coarse grids (called **coarse grid correction**). This iterative solver is called a multigrid method (or multigrid cycles, multigrid iterations). As a result of the above desirable properties, multigrid cycles can solve elliptic linear systems both efficiently and accurately.

Figure 1.1 illustrates an example of error evolution in one multigrid cycle. The linear problem is a two-dimensional Poisson equation. The process of smoothing the error is shown in Figure 1.1(i)–(ii), while the process of coarse grid correction is shown in Figure 1.1(iii)–(iv). More specifically, start with a random initial guess of the solution, where the error is shown in Figure 1.1(i). Figure 1.1(ii) shows that, after only 2 Gauss-Seidel iterations, the error immediately becomes smooth, even though the error reduction is slow. Next we estimate the error by solving the corresponding linear system on a coarse grid, which is computationally cheaper. The coarse grid estimated error is shown in Figure

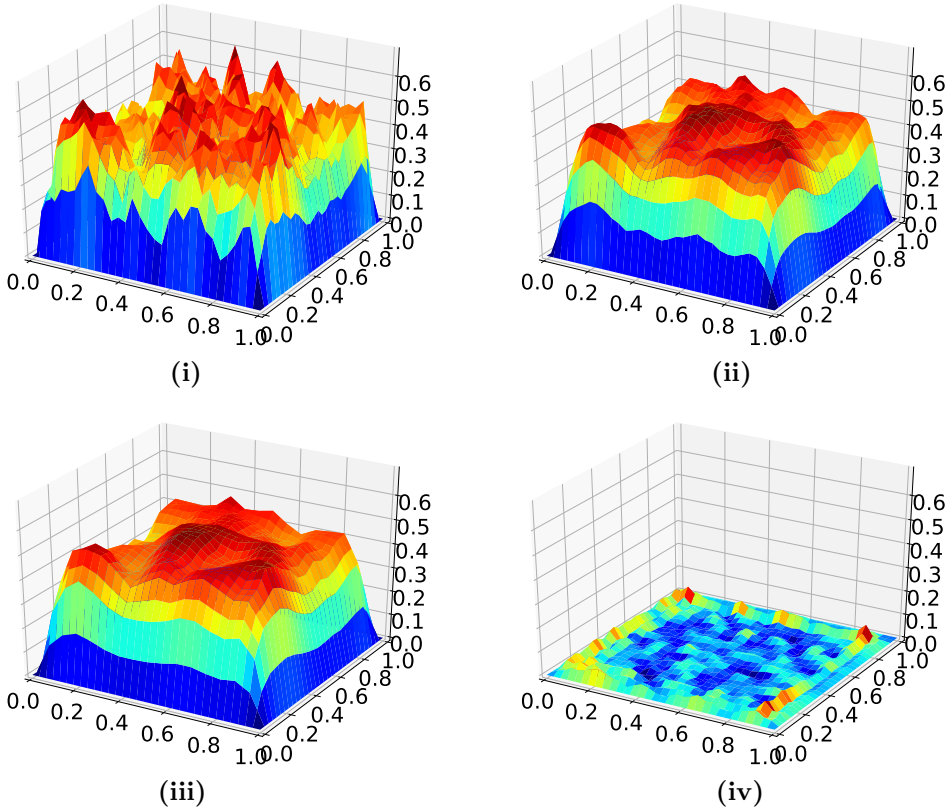


Figure 1.1: Evolution of error of in one multigrid cycle: **(i)** Initial error; **(ii)** Error after 2 Gauss-Seidel iterations; **(iii)** Estimated error on a coarse grid; **(iv)** Resulting error after subtracting (iii) from (ii).

1.1(iii), which is very similar to the fine grid error in Figure 1.1(ii). Then from the fine grid error we subtract the interpolated coarse grid estimated error, resulting in Figure 1.1(iv). As a result, the error is reduced very efficiently. By applying this multigrid cycle iteratively, we would expect the approximate solution to converge to the exact solution rapidly.

A fully-defined multigrid cycle consists of the following components: smoother, coarsening strategy, restriction, interpolation, and coarse grid problem. A **smoother** is a relaxation method that smooths the error. The default choice is Gauss-Seidel iteration. Other choices include damped Jacobi iteration, block Gauss-Seidel iteration, etc. A **coarsening strategy** is a definition of coarse grid configuration. Figure 1.2 shows an example of a fine

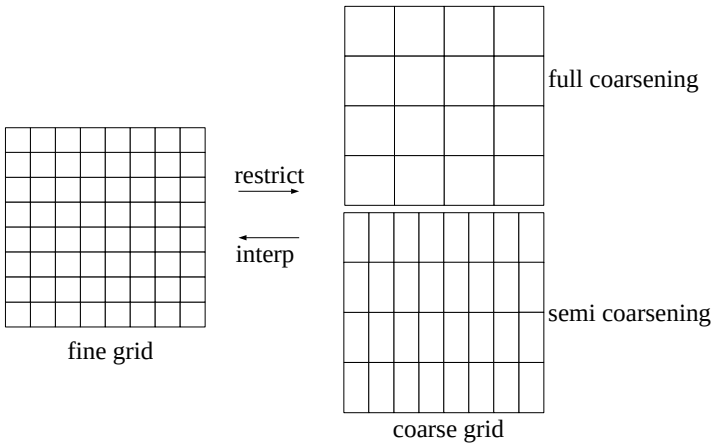


Figure 1.2: Fine grid, coarse grid, coarsening strategies (full-coarsening and semi-coarsening).

grid, a coarse grid, and coarsening strategies such as full-coarsening and semi-coarsening. The default choice is full-coarsening. An **interpolation** (or a **restriction**) is a linear operator that approximates a grid function from a coarse grid to a fine grid (or from a fine grid to a coarse grid, respectively). More specifically, the estimated error is interpolated from a coarse grid to a fine grid, where the default choice is the **linear (or bilinear, trilinear) interpolation**. Additionally, the residual of the approximate solution is restricted from a fine grid to a coarse grid, where the default choice is **full-weighting restriction** (the transpose of the linear interpolation). A **coarse grid problem** defines how the error on a fine grid is approximated on a coarse grid. The coarse grid matrix, denoted as A_{2h}, can be constructed in two ways. One is to use **direct discretization**, i.e., A_{2h} comes from exactly the same discretization scheme of the PDE as A_h, except that the mesh size h is replaced by $2h$. The other is to use **Petrov-Galerkin coarse grid operator**, defined as

$$A_{2h}^{PG} \equiv R_h A_h P_h, \tag{1.6}$$

where R_h and P_h are the restriction and interpolation matrices. The right hand side of the coarse grid problem is constructed from the restricted residual.

Eventually, each multigrid cycle assembles these components together in the following order:

- pre-smoothing using a smoother;
- coarse grid correction, including:

  - defining a coarse grid based on a coarsening strategy;
  - restricting the residual from the fine grid to the coarse grid;
  - constructing a coarse grid problem using either direct discretization or the Petrov-Galerkin coarse grid operator;
  - solving the coarse grid problem, noting that the coarse grid problem itself is also a linear system and can be recursively solved by the "smoothing – coarse grid correction" process on even coarser grids, until on the coarsest grid, where a linear system becomes very small, and can be solved easily by Gaussian elimination;
  - interpolating the estimated error from the coarse grid to the fine grid;
  - subtracting the interpolated error from the fine grid approximate solution;
- post-smoothing using the same smoother.

A multigrid method iterates through this cycle until convergence. We refer interested readers to Section 2.4 of [153] for further introduction to multigrid cycle and Algorithm A.1 in the appendices for the corresponding pseudo-code.

Designing effective multigrid methods for HJB equations is challenging. One of the main reasons is the nonlinearity of HJB equations. We note that multigrid methods were originally developed to solve linear systems. For nonlinear systems, including HJB equations, there are two major multigrid approaches. The first approach is called the **global linearization method** (or outer-inner linearization method) [8, 9]. The idea is to solve the nonlinear system using a nonlinear iteration, such as policy iteration, where each iteration requires solving a linearized system. Then one can apply multigrid cycles to solve each linearized system. This approach involves two layers of iterations, where the outer iteration is the nonlinear iteration, while the inner iteration is the multigrid cycles.

The second approach is called the **full approximation scheme (FAS)** [153, 28, 87, 93, 23]. FAS is a family of multigrid methods that extends and generalizes linear multigrid methods directly to nonlinear systems. Similar to the linear multigrid method, FAS also iterates between smoothing error and coarse grid correction. Each FAS cycle follows the same procedure as one linear multigrid cycle, except two specific multigrid components. One is the smoother. In particular, a linear multigrid cycle uses linear relaxation methods, whereas FAS uses nonlinear relaxation methods. The other difference is the coarse grid problem. More specifically, linear coarse grid problems can be defined by either direct discretization or the Petrov-Galerkin coarse grid operator. However, coarse grid problems

of FAS is nonlinear. The nonlinearity of FAS is in general incompatible with the Petrov-Galerkin coarse grid operator. Consequentially, coarse grid problems of FAS are (almost) only defined by direct discretization. We refer readers to Section 5.3 of [153] for a further introduction to FAS and Algorithm A.2 in the appendices for the corresponding pseudo-code.

We note that designing an efficient FAS is usually more difficult than global linearization methods, mainly because the choice of FAS coarse grid operators is limited to direct discretization only. However, if an efficient FAS can be developed, it usually takes fewer iterations to converge than global linearization methods with the same multigrid components. This is because FAS is directly developed upon nonlinear discretized systems, and involves only one layer of iterations. Conversely, global linearization methods carry not only outer nonlinear but also inner multigrid iterations. The total number of iterations, which is the product of outer and inner iteration counts, can be large.

Nonlinearity is not the only challenge for designing efficient multigrid methods for HJB equations. We note that HJB equations may contain anisotropy, convection, singularity, etc. If one simply chooses the standard multigrid components, then multigrid methods may become inefficient or even non-convergent. In this thesis, we devise appropriate multigrid components under either the global linearization framework or the FAS framework, such that our multigrid methods can address the above issues and achieve mesh-independent convergence rates for HJB equations.

1.3.4 Deep neural networks

In some practical applications of HJB equations (e.g., American options), the dimension of the computational domain (e.g., the number of underlying assets of an American option) could be as high as hundreds. The conventional approaches, including finite difference, finite volume and finite element methods, become impractical if the dimension is greater than 3. The reason, called "curse of dimensionality", is that the number of mesh points, and thus the complexity, grows exponentially with the dimensionality. Solving high-dimensional HJB equations requires new techniques.

Deep learning approaches, or deep neural networks [80], are one of the most sought-after and successful frameworks in the past decades. Deep neural networks have proved successful in applications where the dimensions are extremely high, such as Google's AlphaGo AI [141] and natural language processing [50, 51]. In this thesis, we will use deep neural networks to solve high-dimensional HJB equations.

Deep neural networks are a family of nonlinear parameterized functions. A simpler, basic type of network, is called a **one-layer feed-forward network**. Such a network is a function that maps a d-dimensional input $\boldsymbol{x} \in \mathbb{R}^d$ to a scalar output $y \in \mathbb{R}$ using a composition of the following transformations:

$$\text{parameterized linear transformation } \mathbb{R}^d \to \mathbb{R}^\chi : \qquad \boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}, \tag{1.7}$$

$$\text{nonlinear transformation } \mathbb{R}^\chi \to \mathbb{R}^\chi : \qquad \boldsymbol{h} = f(\boldsymbol{z}), \tag{1.8}$$

$$\text{parameterized linear transformation } \mathbb{R}^\chi \to \mathbb{R} : \qquad y = \boldsymbol{w}^T\boldsymbol{h} + b, \tag{1.9}$$

where $\boldsymbol{W} \in \mathbb{R}^{\chi \times d}$, $\boldsymbol{b} \in \mathbb{R}^\chi$, $\boldsymbol{\omega} \in \mathbb{R}^\chi$ and $b \in \mathbb{R}$ are parameters (or linear coefficients) in the nonlinear function, and f is an element-wise nonlinear mapping, such as $h_i = \max(z_i, 0)$ for $i = 1, \cdots, \chi$. The composition of these transformations yields a nonlinear parameterized function

$$y(\boldsymbol{x}; \Omega) \equiv \boldsymbol{w}^T f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b, \tag{1.10}$$

where $\Omega \equiv \{\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{w}, b\}$ is the set of parameters. In practical applications, it is common to repeat and stack the transformations (1.7)-(1.8) multiple times before applying the output transformation (1.9). This gives rise to a multi-layer feed-forward network, also known as a **deep feed-forward network**, where each composition of (1.7)-(1.8) is called one **layer** of the network. We note that deep neural networks are not limited to such feed-forward architecture. Numerous architectures, such as convolutional networks, recurrent and recursive networks, generative adversarial networks, have been proposed. We refer interested readers to [80] for a substantial introduction to the topic of deep neural networks.

In order to sketch how a deep neural network framework solves a high-dimensional PDE, consider a generic PDE

$$\mathcal{N}\, u(\boldsymbol{x}, t) = 0. \tag{1.11}$$

One approach to solving the PDE is to write the solution as a parameterized function $y(\boldsymbol{x}, t; \Omega)$, where Ω is the parameter set of the function. Then the goal is to find the parameter set that best fits the PDE, i.e., to find the optimal parameter set Ω^* that minimizes the residual of the PDE under the L_2 norm:

$$\Omega^* \equiv \arg\min_{\Omega} \| \mathcal{N}\, y(\boldsymbol{x}, t; \Omega) \|_{L_2}. \tag{1.12}$$

Then $y(\boldsymbol{x}, t; \Omega^*)$ is an approximation of the solution $u(\boldsymbol{x}, t)$. The main advantages of this formulation are two-fold. One is that it is mesh-free, as opposed to the conventional approaches. As a result, its complexity does not necessarily grow exponentially with the dimensionality. Instead, the complexity depends on the size of the parameter set, which does not usually grow exponentially with the dimensionality. The other desirable property

of this formulation is that a complicated high-dimensional PDE is converted to an optimization problem with respect to a set of parameters, which allows us to apply various well-developed optimization techniques, such as gradient descent, to solve the PDE.

Significantly, a deep neural network can be used as a parameterized solution of a PDE, and can yield an accurate solution without suffering from curse of dimensionality; see a very recent work [142] for a demonstration.

We note that deep neural networks are not the only choice for a parameterized solution. Indeed, the well-known spectral methods and finite element methods use similar ideas. More specifically, the solution is approximated by a linear combination of user-defined static basis functions $\{\varphi_k(\boldsymbol{x}) \,|\, k = 1, \cdots, K\}$:

$$y(\boldsymbol{x};\, \boldsymbol{\omega}) = \sum_{k=1}^{K} \omega_k \varphi_k(\boldsymbol{x}) = \boldsymbol{\omega}^T \boldsymbol{\varphi}(\boldsymbol{x}). \tag{1.13}$$

However, a set of user-defined static basis functions may not be the optimal choice for each specific PDE. Remarkably, a deep neural network is essentially a linear combination of *dynamical* basis functions. For instance, the one-layer network (1.10) is a linear combination of the dynamical basis parameterized by $\boldsymbol{\varphi}(\boldsymbol{x};\, \boldsymbol{W}, \boldsymbol{b}) \equiv f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$. The optimal basis functions are learned during the minimization of the PDE residual. As a result, deep neural networks can yield accurate solutions.

In this thesis, we study high-dimensional American option problems as an exemplary application of high-dimensional HJB equations, and propose a deep neural network framework. The HJB equations have the form:

$$\mathcal{N} u(\boldsymbol{x}, t) = 0, \tag{1.14}$$

where

$$\mathcal{N} u(\boldsymbol{x}, t) \equiv \min\left(\mathcal{L}\, u(\boldsymbol{x}, t),\, u(\boldsymbol{x}, t) - f(\boldsymbol{x})\right), \tag{1.15}$$

$\mathcal{L}$ is a second order linear differential operator, and f is a given function[1]. As deep neural networks are a fairly new framework for solving PDEs, many challenges still require investigation. For instance, the differential operator $\mathcal{L}\, u(\boldsymbol{x}, t)$ contains the Hessian of the solution $D^2 u(\boldsymbol{x}, t)$. As a result, a neural network with the loss function (1.12) requires computation of a Hessian tensor of size $O(Md^2)$, where M is the size of the dataset for the training and evaluation process of the neural network. When d is large, the quadratic size of the Hessian tensor could be cumbersome in both computational time and memory. Another challenge

[1] We will show in Section 6.2.2 that (1.14)-(1.15) can be written in the form of (1.1)-(1.2).

is that, in practical financial applications, it is crucial to solve not only the solution of an HJB equation (i.e., American option price), but also the gradient of the solution (i.e., American option delta). To the best of our knowledge, no literature in deep neural networks has been dedicated to solving both prices and deltas of high-dimensional American options accurately and efficiently. We will discuss how to address these challenges using our deep neural network formulation.

We remark that, in the literature, American option problems are more usually formulated as linear complementarity problems, i.e.,

$$\begin{aligned} &\mathcal{L}\, u(\boldsymbol{x}, t) \geq 0, \\ &u(\boldsymbol{x}, t) \geq f(\boldsymbol{x}), \\ &(u(\boldsymbol{x}, t) - f(\boldsymbol{x})) \cdot \mathcal{L}\, u(\boldsymbol{x}, t) = 0. \end{aligned} \tag{1.16}$$

One may consider using a neural network approach to solve the linear complementarity problems. However, it is unclear how to directly handle the inequalities in (1.16) using neural networks. Conversely, the HJB formulation (1.14)-(1.15) is an equation rather than an inequality problem, which allows us to design a neural network approach based on the paradigm (1.12).

1.4 Contributions

The central contribution of this thesis is that we propose various numerical methods for HJB equations as introduced in Section 1.3, and we demonstrate the accuracy and efficiency of these numerical methods by applying them to the three applications introduced in Section 1.2. More specifically:

(I) When the spatial dimension of an HJB equation is less than 3, where we study mean field games and image registration as exemplary problems, we propose general finite difference frameworks for solving HJB equations in a convergence sense and in an efficient manner. In particular:

- We develop finite difference schemes that satisfy consistency, stability, monotonicity and the strong comparison principle. As a result, a discrete solution is guaranteed to converge to the continuous viscosity solution as $h \to 0$. We support our claim with mathematical proofs.

- We solve discretized HJB systems using nonlinear iterative solvers, including policy iteration and the Levenberg-Marquardt algorithm [114, 120]. These iterative solvers are convergent.

- We speed up solving discretized HJB systems by devising efficient multigrid methods under either the global linearization framework or the FAS framework. In order to overcome difficulties such as nonlinearity, convection and singularity, we design the multigrid components (i.e., smoother, coarsening strategy, restriction, interpolation, and coarse grid problem) with special care. In particular, some multigrid components we propose are novel, such as subtracting artificial viscosity from coarse grid problems, and using injection as the restriction for wide stencil discretization. Significantly, our multigrid methods yield mesh-independent convergence rates, and achieve faster convergence rates than other existing solvers in the literature.

(II) When the spatial dimension of an HJB equation is as high as 200, where we study high-dimensional American option problems as an exemplary problem, we propose a deep neural network framework. Our deep neural network framework has a new architecture (sequence of neural networks with non-conventional connectivity). In addition, our framework uses a new loss function (least squares residual of backward stochastic differential equation), which avoids the costly evaluation and storage of Hessian tensors, and couples the solution $u(\boldsymbol{x}, t)$ and its gradient $\nabla u(\boldsymbol{x}, t)$ in a single loss function. As a result, our formulation yields not only accurate solutions, but also accurate gradients, on the entire spacetime, which cannot be computed correctly by the state-of-the-art approaches (e.g., [117]) in high dimensions. Moreover, our deep neural network framework addresses the curse of dimensionality issue, which is supported by the result that our approach solves the American option problems accurately in as high as 200 dimension, while the state-of-the-art approaches fail to solve the problems due to the out-of-memory error and the exponentially (or high-degree polynomially) increasing computational cost in above 20 dimension.

Our results appear in the following articles (or are submitted under review):

- Yangang Chen and Justin W. L. Wan. Multigrid methods for convergent mixed finite difference scheme for Monge-Ampère equation. *Computing and Visualization in Science*, pages 1–15, 2017

- Yangang Chen, Justin W. L. Wan, and Jessey Lin. Monotone mixed finite difference scheme for Monge-Ampère equation. *J. Sci. Comput.*, 76(3):1839–1867, 2018

- Yangang Chen and Justin W. L. Wan. Numerical method for image registration model based on optimal mass transport. *Inverse Probl. Imaging*, 12(2):401–432, 2018

- Yangang Chen and Justin W. L. Wan. Artificial viscosity joint spacetime multigrid method for Hamilton-Jacobi-Bellman and Kolmogorov-Fokker-Planck system arising from mean field games. *(submitted, under review)*, 2019

- Yangang Chen and Justin W. L. Wan. Deep neural network framework based on backward stochastic differential equations for pricing and hedging American options in high dimensions. *(submitted, under review)*, 2019

1.5 Outline

This thesis is organized by the three applications of HJB equations as follows:

First, in Chapter 2, we study numerical methods for HJB equations arising from mean field games. In this application, an HJB equation is coupled with a KFP equation, which forms a PDE system. By solving the HJB/KFP PDE system, we present a general pipeline of numerically solving HJB equations, which involves finite difference discretization, solvers for the discretized system, and moreover, efficient solvers for speeding up computation. The focus of this chapter is to propose an efficient multigrid method for solving the discretized HJB/KFP system, and to illustrate the mesh-independent convergence rates of the proposed multigrid method.

Next, in Chapters 3-5, we investigate numerical methods for HJB formulation associated with image registration problems. Considering the sophistication of image registration problems, we break down this topic into three chapters:

Chapter 3 considers a simplified image registration problem, i.e., a Monge-Ampère equation where the right hand side does not depend on the solution u. The focus of this chapter is to establish the equivalence between a Monge-Ampère equation and an HJB equation, and to propose a finite difference method for the equivalent HJB formulation that converges in the viscosity sense. This chapter serves as a foundation for solving the more complicated image registration problems. Chapter 4 proposes multigrid methods for the discretized HJB system considered in Chapter 3, and demonstrates that our multigrid methods achieve mesh-independent convergence rates. Chapter 5 comes back to the image registration problems, where we apply the discretization scheme introduced in Chapter 3. The new challenge of Chapter 5 is that, in order to address the issue of registration

quality in the literature, we propose imposing a novel periodic boundary condition, which leads to a singular matrix. In order to solve the singular system, we propose using the Levenberg-Marquardt algorithm.

Then, in Chapter 6, we study numerical methods for HJB equations arising from American option problems. Unlike the previous chapters, the HJB equation considered in this chapter is high-dimensional. In order to address the curse of dimensionality, we propose a deep neural network framework. We illustrate the accuracy and efficiency of the proposed approach.

Finally, we conclude the thesis in Chapter 7.

Chapter 2

Multigrid Method for HJB Equations Arising from Mean Field Games

2.1 Introduction

This chapter studies the numerical solution of HJB equations arising from mean field games. Let Ω be a space domain in $\mathbb{R}^d$. Let $\Omega \times [0,T]$ be a spacetime domain. Let $\boldsymbol{x} \in \Omega$ be the d-dimensional state variable of the controllers (players), and let $t \in [0,T]$ be the time. Let $u : \Omega \times [0,T] \to \mathbb{R}$ be the optimal value function of the players, and let $m : \Omega \times [0,T] \to \mathbb{R}$ be the distribution (or density function) of the players' state variable. Let $\boldsymbol{c} : \Omega \times [0,T] \to \mathbb{R}^d$ be a d-dimensional control parameter. In [111, 83], mean field games are formulated into a system of PDEs that contains two equations. One equation is the **HJB equation** for the value function $u(\boldsymbol{x},t)$:

$$\begin{aligned} -u_t(\boldsymbol{x},t) - \sigma\Delta u(\boldsymbol{x},t) + \boldsymbol{c}^*(\boldsymbol{x},t)\cdot\nabla u(\boldsymbol{x},t) - L(\boldsymbol{x},t;\boldsymbol{c}^*(\boldsymbol{x},t)) & \\ +ru(\boldsymbol{x},t) - \Phi(m(\boldsymbol{x},t)) = 0, \quad \text{in } \Omega\times[0,T), & \\ u(\boldsymbol{x},T) = u_T(\boldsymbol{x}), \quad \text{in } \Omega, & \end{aligned} \tag{2.1}$$

subject to the optimal control:

$$\boldsymbol{c}^*(\boldsymbol{x},t) \equiv \underset{\boldsymbol{c}(\boldsymbol{x},t)\in\mathbb{R}^d}{\arg\max} \left\{\boldsymbol{c}(\boldsymbol{x},t)\cdot\nabla u(\boldsymbol{x},t) - L(\boldsymbol{x},t;\boldsymbol{c}(\boldsymbol{x},t))\right\}. \tag{2.2}$$

Here r is the discount factor, σ is the diffusion factor, Φ is the local cost function and L is the Lagrangian. The function that (2.2) aims to maximize, denoted as

$$H(\boldsymbol{c};u) \equiv \boldsymbol{c}\cdot\nabla u - L(\boldsymbol{c}), \tag{2.3}$$

is called the Hamiltonian. We assume that, for any fixed u, the Hamiltonian is a concave function of $\boldsymbol{c}$, and the maximum can be achieved at the corresponding stationary point[1]. We note that the HJB equation is solved backward from the terminal time $t = T$ to the initial time $t = 0$. Hence, the HJB equation has a terminal condition $u(\boldsymbol{x}, T) = u_T(\boldsymbol{x})$ rather than an initial condition.

The other equation in the PDE system is the forward **Kolmogorov-Fokker-Planck (KFP) equation** for the distribution $m(\boldsymbol{x}, t)$:

$$\begin{aligned} m_t(\boldsymbol{x}, t) - \sigma \Delta m(\boldsymbol{x}, t) - \nabla \cdot (\boldsymbol{c}^*(\boldsymbol{x}, t) m(\boldsymbol{x}, t)) = 0, \quad & \text{in } \Omega \times (0, T], \\ m(\boldsymbol{x}, 0) = m_0(\boldsymbol{x}), \quad & \text{in } \Omega. \end{aligned} \tag{2.4}$$

For simplicity, unless otherwise specified, we assume periodic boundary conditions for both (2.1) and (2.4). We refer readers to [111, 2] for a discussion on the well posedness of the problem (2.1)-(2.4).

A unique feature of the HJB/KFP system is that, while the HJB equation (2.1)-(2.2) is backward from $t = T$ to $t = 0$, the KFP equation (2.4) is forward from $t = 0$ to $t = T$. Furthermore, the HJB equation is nonlinear, since the optimal control $\boldsymbol{c}^*$ that maximizes $\boldsymbol{c} \cdot \nabla u - L(\boldsymbol{c})$ is a functional of u. The two equations are coupled, because, in the HJB equation, the cost function $\Phi(m)$ depends on the solution of the KFP equation m; and, in the KFP equation, the optimal control $\boldsymbol{c}^*$ depends on the solution of the HJB equation u. As a result, the entire HJB/KFP system is nonlinear.

Numerical methods for the HJB/KFP system have been studied extensively in [2, 6, 10, 31, 3, 4, 5, 107, 40, 32]. We note that the nonlinearity and the size of the discretized system pose major challenges for the numerical solution for the HJB/KFP system. To address the challenges, effective and fast solvers are required.

However, only a few papers, such as [2, 6, 10, 31], have proposed effective and fast solvers for the nonlinear discretized HJB/KFP systems. These solvers have two common features. One is that they are all spacetime methods. Spacetime methods are numerical methods where the unknowns for all the timesteps are solved simultaneously in a single system, as opposed to timestepping where the unknowns are solved timestep by timestep. The reason to consider spacetime methods is that the HJB equation is backward and the KFP equation is forward, which makes it impossible to solve the system using the conventional forward timestepping. The other common feature is that all of these methods are global linearization (or outer-inner linearization) multigrid methods. More specifically, in order to solve the nonlinear discretized problems, [2, 6] propose spacetime Newton's

[1] In some applications, the Hamiltonian is convex in $\boldsymbol{c}$. Then "max" in (2.2) is replaced by "min".

iterations; [10] proposes ALG2 (i.e., Douglas-Rachford) iterations; [31] proposes primal-dual (i.e., Chambolle-Pock) iterations. Each nonlinear iteration requires solving a large spacetime linear system. In order to solve the linear system, they typically use BiCGStab iterations, and use multigrid cycles as preconditioners for each BiCGStab iteration. The common feature of these approaches is that they all involve two layers of iterations: outer nonlinear iterations, and inner multigrid iterations (or inner BiCGStab iterations with multigrid preconditioners) for each linearized system. As a result, the total number of iterations, which is approximately the product of the iteration counts of the outer and inner iterations, can be large. To be more concrete, the approach in [10] typically requires more than 1000 outer ALG2 iterations and 7 inner multigrid-preconditioned CG iterations per ALG2 iteration, i.e., more than 7000 iterations in total; the approach in [31] typically requires more than 20 outer Chambolle-Pock iterations and 4 inner multigrid-preconditioned BiCGStab iterations per outer iteration, i.e., more than 80 iterations in total.

To address the issues of these existing multigrid methods, in this chapter, we propose another spacetime multigrid solver for nonlinear discretized HJB/KFP systems. In particular, our multigrid method is a **full approximation scheme (FAS)** [153, 87]. Unlike the other multigrid methods that are applied iteratively to the inner linearized systems nested in outer nonlinear iterations, our FAS is directly applied to the nonlinear system itself and thus involves only one layer of iterations.

In our FAS multigrid method, we consider a hybrid of full and semi coarsenings to address the anisotropy arising from the time direction [94, 6]. We note that convections in the HJB/KFP system pose a major challenge for multigrid methods, as standard multigrid methods are ineffective for convection-diffusion equations [153]. In order to address this difficulty, we consider adapting a stable and efficient multigrid method proposed in [13], which uses a type of biased restriction, called a kernel preserving restriction, together with Petrov-Galerkin coarse grid operators. However, the multigrid method in [13] is designed for linear equations. It cannot be directly applied to the FAS framework, because the nonlinearity of FAS is incompatible with Petrov-Galerkin operators. Direct discretization, as the alternative of Petrov-Galerkin operators, is compatible with the FAS framework. However, it is well-known that when convection is non-negligible, the convergence rate for the direct discretization coarse grid operators is no better than 0.5 [27].

Our approach, which is the main novel component of this chapter, is to **subtract artificial viscosity** from the direct discretization coarse grid operators. Subtracting artificial viscosity allows us to design an effective FAS solver under the direct discretization operators when convection is non-negligible. Our Local Fourier Analysis proves that subtracting artificial viscosity improves the asymptotic convergence factor and the error reduction factor. We remark that a seemingly similar idea—adding artificial viscosity—is well-known

for stabilizing numerical solution for convection-diffusion equations [112, 96, 68]. However, we emphasize that our approach is to subtract (rather than add) artificial viscosity, and our purpose is to improve the coarse grid correction for multigrid methods. This is distinct from the existing artificial viscosity approaches in the literature.

Significantly, our numerical simulations illustrate that our FAS multigrid method with artificial viscosity subtraction yields mesh-independent convergence rates for the HJB/KFP system. In particular, our approach typically converges in less than 10 iterations *in total*, which is faster than the global linearization multigrid methods in [2, 6, 10, 31].

To the best of our knowledge, this chapter is the first proposal of a multigrid method with the following two features: it is an FAS directly designed for the nonlinear discretized HJB/KFP system (as opposed to the linearized version); and in particular, it subtracts artificial viscosity from the direct discretization coarse grid operators. These two features are critical for our proposed multigrid method to converge faster than the other methods.

To illustrate our proposed multigrid method, we first describe the finite difference discretization in Section 2.2. Section 2.3 introduces joint spacetime methods for solving the nonlinear discretized system. Section 2.4 describes our proposed spacetime FAS multigrid methods with artificial viscosity subtraction. Section 2.5 uses Local Fourier Analysis to demonstrate the efficiency of the proposed multigrid method. Section 2.6 presents numerical results. Section 2.7 is the conclusion.

2.2 Finite Difference Discretization

2.2.1 Notation

First, we introduce finite difference notation that will be used throughout the thesis. Without loss of generality, we assume that the spacetime is (2+1)-dimensional. Then the state and the control variables can be denoted as $(\boldsymbol{x}, t) \equiv (x, y, t)$ and $\boldsymbol{c}(\boldsymbol{x}, t) \equiv (c_1(\boldsymbol{x}, t), c_2(\boldsymbol{x}, t))$. Define an $n_x \times n_y \times n_t$ spacetime mesh $\{(\boldsymbol{x}_{i,j}, t_n) \equiv (x_i, y_j, t_n) \mid i = 1, ..., n_x;\ j = 1, ..., n_y;\ n = 0, ..., n_t\}$. Denote the corresponding mesh sizes as Δx, Δy, Δt. For simplicity, unless otherwise specified, we assume that $\Delta x = \Delta y$, in which case we can denote both Δx and Δy as h. Denote the grid function of a continuous function $f(\boldsymbol{x}, t)$ as $\{f^n_{i,j} \equiv f(\boldsymbol{x}_{i,j}, t_n) \,|\, 0 \le n \le n_t, 1 \le i \le n_x, 1 \le j \le n_y\}$. Our goal is to solve the set of the unknowns $\{u^n_{i,j}\}$, $\{m^n_{i,j}\}$ together with the optimal control $\{(\boldsymbol{c}^*)^n_{i,j} \equiv ((c^*_1)^n_{i,j}, (c^*_2)^n_{i,j})\}$.

The first derivatives of the unknown, $u_x(\boldsymbol{x}_{i,j}, t_n)$ and $u_y(\boldsymbol{x}_{i,j}, t_n)$, are usually discretized by either forward or backward differencing, which are represented by the following for-

ward/backward difference operators:

$$
\begin{aligned}
D_x^+ u_{i,j}^n &\equiv \frac{u_{i+1,j}^n - u_{i,j}^n}{\Delta x}, \quad D_x^- u_{i,j}^n \equiv \frac{u_{i,j}^n - u_{i-1,j}^n}{\Delta x}, \\
D_y^+ u_{i,j}^n &\equiv \frac{u_{i,j+1}^n - u_{i,j}^n}{\Delta y}, \quad D_y^- u_{i,j}^n \equiv \frac{u_{i,j}^n - u_{i,j-1}^n}{\Delta y}.
\end{aligned} \tag{2.5}
$$

The second derivatives of the unknown, $u_{xx}(\boldsymbol{x}_{i,j}, t_n)$ and $u_{yy}(\boldsymbol{x}_{i,j}, t_n)$, are usually discretized by the following standard central differencing:

$$
D_x^+ D_x^- u_{i,j}^n \equiv \frac{u_{i+1,j}^n - 2u_{i,j}^n + u_{i-1,j}^n}{\Delta x^2}, \quad D_y^+ D_y^- u_{i,j}^n \equiv \frac{u_{i,j+1}^n - 2u_{i,j}^n + u_{i,j-1}^n}{\Delta y^2}. \tag{2.6}
$$

2.2.2 Finite difference discretization

Consider numerical solution of (2.1)-(2.4). For the HJB equation (2.1)-(2.2), we use implicit timestepping for u_t, central differencing for Δu, and upwinding discretization for $\boldsymbol{c}^* \cdot \nabla u = c_1^* u_x + c_2^* u_y$. Define $c^+ = \max(c, 0)$, and $c^- = \min(c, 0)$. Then the discretization of HJB equation reads

$$
\begin{aligned}
-\frac{u_{i,j}^{n+1} - u_{i,j}^n}{\Delta t} - \sigma D_x^+ D_x^- u_{i,j}^n - \sigma D_y^+ D_y^- u_{i,j}^n + ((c_1^*)_{i,j}^n)^+ D_x^- u_{i,j}^n + ((c_1^*)_{i,j}^n)^- D_x^+ u_{i,j}^n \\
+((c_2^*)_{i,j}^n)^+ D_y^- u_{i,j}^n + ((c_2^*)_{i,j}^n)^- D_y^+ u_{i,j}^n - L((\boldsymbol{c}^*)_{i,j}^n) + r u_{i,j}^n - \Phi(m_{i,j}^n) = 0, \\
n = n_t - 1, \cdots, 0, \; i = 1, \cdots, n_x, \; j = 1, \cdots, n_y,
\end{aligned} \tag{2.7}
$$

subject to the optimal control:

$$
\begin{aligned}
(\boldsymbol{c}^*)_{i,j}^n \equiv \arg\max_{\boldsymbol{c}_{i,j}^n \in \mathbb{R}^2} \big\{ ((c_1)_{i,j}^n)^+ D_x^- u_{i,j}^n + ((c_1)_{i,j}^n)^- D_x^+ u_{i,j}^n \\
+((c_2)_{i,j}^n)^+ D_y^- u_{i,j}^n + ((c_2)_{i,j}^n)^- D_y^+ u_{i,j}^n - L((\boldsymbol{c}^*)_{i,j}^n) \big\}.
\end{aligned} \tag{2.8}
$$

We note that the HJB equation is backward in time, i.e., $n = n_t - 1, \cdots, 0$. Hence, the implicit timestepping is given by $(u_{i,j}^{n+1} - u_{i,j}^n)/\Delta t$ rather than the conventional $(u_{i,j}^n - u_{i,j}^{n-1})/\Delta t$.

For the KFP equation (2.4), we also use implicit timestepping for m_t. Notice that the KFP equation is written into a conservation form [152]. A standard discretization for

conservation laws is to use a numerical flux for $f = \boldsymbol{c}^* m$. For instance, we can choose the Engquist-Osher flux [152]:

$$\begin{aligned} \hat{f}^n_{i+1/2,j} &= ((c_1^*)^n_{i,j})^- m^n_{i,j} + ((c_1^*)^n_{i+1,j})^+ m^n_{i+1,j}, \\ \hat{f}^n_{i,j+1/2} &= ((c_2^*)^n_{i,j})^- m^n_{i,j} + ((c_2^*)^n_{i,j+1})^+ m^n_{i,j+1}. \end{aligned} \tag{2.9}$$

This is essentially an upwind flux with an additional consideration for rarefactions and shocks. As a result, the finite difference discretization for the KFP equation is given by

$$\begin{aligned} \frac{m^n_{i,j} - m^{n-1}_{i,j}}{\Delta t} - \sigma D_x^+ D_x^- m^n_{i,j} - \sigma D_y^+ D_y^- m^n_{i,j} - \frac{\hat{f}^n_{i+1/2,j} - \hat{f}^n_{i-1/2,j}}{\Delta x} - \frac{\hat{f}^n_{i,j+1/2} - \hat{f}^n_{i,j-1/2}}{\Delta y} = 0, \\ n = 1, \cdots, n_t,\ i = 1, \cdots, n_x,\ j = 1, \cdots, n_y. \end{aligned} \tag{2.10}$$

Following the standard finite difference paradigm, we rewrite the nonlinear discrete system (2.7)-(2.10) in matrix forms. Define the vector $u^n_h \equiv (u^n_{1,1}, u^n_{1,2}, \cdots, u^n_{n_x,n_y})^T \in \mathbb{R}^{n_x n_y}$ as the vector of $\{u^n_{i,j} \,|\, 1 \le i \le n_x, 1 \le j \le n_y\}$ at the n-th timestep, where the elements are arranged in lexicographical order. Similarly, define the vectors $m^n_h \in \mathbb{R}^{n_x n_y}$, $(c_1)^n_h \in \mathbb{R}^{n_x n_y}$, $(c_2)^n_h \in \mathbb{R}^{n_x n_y}$ and $\boldsymbol{c}^n_h \equiv ((c_1)^n_h, (c_2)^n_h)^T \in \mathbb{R}^{2n_x n_y}$. Then the discretized HJB equation (2.7)-(2.8) can be rewritten as

$$\begin{aligned} &A^n_{HJB}((\boldsymbol{c}^*)^n_h) u^n_h = \frac{1}{\Delta t} \cdot u^{n+1}_h + L((\boldsymbol{c}^*)^n_h) + \Phi(m^n_h), \\ &\text{subject to} \quad (\boldsymbol{c}^*)^n_h \equiv \underset{\boldsymbol{c}^n_h \in \mathbb{R}^{2n_x n_y}}{\arg\max} \left\{ A^n_{HJB}(\boldsymbol{c}^n_h) u^n_h - L(\boldsymbol{c}^n_h) \right\}, \quad n = n_t - 1, \cdots, 0. \end{aligned} \tag{2.11}$$

Here $A^n_{HJB} \in \mathbb{R}^{n_x n_y \times n_x n_y}$ is a matrix that assembles the coefficients of $u^n_{i,j}$, $u^n_{i\pm1,j}$ and $u^n_{i,j\pm1}$ from (2.7). We have added the term $u^n_{i,j}/\Delta t - \sigma D_x^+ D_x^- u^n_{i,j} - \sigma D_y^+ D_y^- u^n_{i,j} + r u^n_{i,j}$, which does not depend on the control, to the constraint (2.8), such that both lines of (2.11) use the same matrix A^n_{HJB}. One can see from (2.7) that A^n_{HJB} depends on the control vector $(\boldsymbol{c}^*)^n_h$. Similarly, the discretized KFP equation (2.10) can be rewritten as

$$A^n_{KFP}\left((\boldsymbol{c}^*)^n_h\right) m^n_h = \frac{1}{\Delta t} \cdot m^{n-1}_h, \quad n = 1, \cdots, n_t, \tag{2.12}$$

where A^n_{KFP} is the corresponding coefficient matrix from (2.10).

2.3 Solving the Discretized System

Next we consider solving the nonlinear discretized HJB/KFP system (2.11)-(2.12). A standard approach to solving a time-dependent system is timestepping. For the HJB/KFP

system, timestepping needs to be implemented as a forward/backward fixed point iteration. We leave the discussion of this standard approach to Appendix A.2.

In this chapter, we will instead focus on another approach, called the spacetime formulation. The idea of the spacetime formulation is to treat the unknowns of the discretized HJB/KFP system for all the timesteps as one entity, and solve them simultaneously. Our motivation for using spacetime methods is to develop fast solvers for the HJB/KFP system. The idea is that, in addition to the spatial dimensions, spacetime methods allow computational speed-up in the time dimension as well. Some literature, such as [76, 77], has also considered using spacetime methods to speed up the computation for time dependent PDEs. These methods would not be more advantageous than the conventional timestepping unless they are implemented in a parallel manner. However, we will show that our approach achieves faster convergence than timestepping even without parallelization. Spacetime methods have also been seen in the numerical solution for time dependent PDEs on deforming domains [133], which is beyond the scope of this thesis.

Mathematically, introduce the spacetime unknown vectors $u_h \equiv (u_h^0, ..., u_h^{n_t-1})^T \in \mathbb{R}^{n_x n_y n_t}$, $m_h \equiv (m_h^1, ..., m_h^{n_t})^T \in \mathbb{R}^{n_x n_y n_t}$, and the corresponding spacetime control vector $\boldsymbol{c}_h = (\boldsymbol{c}_h^0, ..., \boldsymbol{c}_h^{n_t})^T \in \mathbb{R}^{2n_x n_y (n_t+1)}$. Then the HJB/KFP system (2.11)-(2.12) can be rewritten into the following spacetime matrix forms:

$$\begin{aligned} &A_{HJB}(\boldsymbol{c}_h^*)u_h = b_{HJB}(\boldsymbol{c}_h^*, m_h), \\ &\text{subject to} \quad \boldsymbol{c}_h^* = \arg\max_{\boldsymbol{c}_h} \left\{ A_{HJB}(\boldsymbol{c}_h)u_h - L(\boldsymbol{c}_h) \right\}, \end{aligned} \tag{2.13}$$

$$A_{KFP}(\boldsymbol{c}_h^*)m_h = b_{KFP}. \tag{2.14}$$

Here

$$A_{HJB} = \begin{pmatrix} A_{HJB}^0 & -\frac{1}{\Delta t}I & & \\ & A_{HJB}^1 & -\frac{1}{\Delta t}I & \\ & & \ddots & \ddots \\ & & & A_{HJB}^{n_t-1} \end{pmatrix}, A_{KFP} = \begin{pmatrix} A_{KFP}^1 & & & \\ -\frac{1}{\Delta t}I & A_{KFP}^2 & & \\ & \ddots & \ddots & \\ & & -\frac{1}{\Delta t}I & A_{KFP}^{n_t} \end{pmatrix}$$

are $n_x n_y n_t \times n_x n_y n_t$ matrices, and

$$b_{HJB} = \begin{pmatrix} L((\boldsymbol{c}^*)_h^0) + \Phi(m_h^0) \\ L((\boldsymbol{c}^*)_h^1) + \Phi(m_h^1) \\ \vdots \\ L((\boldsymbol{c}^*)_h^{n_t-1}) + \Phi(m_h^{n_t-1}) + \frac{1}{\Delta t}u_h^{n_t} \end{pmatrix}, \quad b_{KFP} = \begin{pmatrix} \frac{1}{\Delta t}m_h^0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

are $n_x n_y n_t$ vectors.

We can further rewrite the system (2.13)-(2.14) into a nonlinear discretized system for the joint unknown variable (u_h, m_h) on the entire spacetime domain:

$$\begin{pmatrix} A_{HJB}(\boldsymbol{c}_h^*) & \\ & A_{KFP}(\boldsymbol{c}_h^*) \end{pmatrix} \begin{pmatrix} u_h \\ m_h \end{pmatrix} = \begin{pmatrix} b_{HJB}(\boldsymbol{c}_h^*, m_h) \\ b_{KFP} \end{pmatrix}, \quad \text{subject to} \quad \boldsymbol{c}_h^* = \arg\max_{\boldsymbol{c}_h} \{A_{HJB}(\boldsymbol{c}_h) u_h - L(\boldsymbol{c}_h)\}. \tag{2.15}$$

To emphasize that (u_h, m_h) are solved together, we call (2.15) a *joint* spacetime formulation.

One may argue that spacetime methods are usually more demanding than conventional timestepping methods in terms of memory, as spacetime methods require storing solutions on the entire spacetime and also the spacetime matrix. However, since the HJB equation is backward and the KFP equation is forward, even the forward/backward timestepping method requires storing both solutions u_h and m_h on the entire spacetime [2]. We can avoid storing the spacetime matrix (2.15) by the full approximation scheme that will be proposed in Section 2.4. Indeed, Section 5.3 of [153] explains that a full approximation scheme does not require explicitly constructing matrices. Hence, for the HJB/KFP system (2.15), the joint spacetime method does not increase memory requirement.

Regarding solving the joint spacetime system (2.15), we can use policy iteration introduced in Algorithm 1.1. The only subtlety is that the right hand side of (2.15) contains m_h. Hence, merely fixing $\boldsymbol{c}_h^*$ would not make the matrix form linear. However, we can modify the policy iteration by fixing both $\boldsymbol{c}_h^*$ and the right-hand-side m_h simultaneously, which makes (2.15) linear. This modified policy iteration is described in Algorithm 2.1. Although there is no theoretical guarantee, Algorithm 2.1 converges in our numerical simulation. Line 5 of the algorithm involves solving an optimization problem. Since we have assumed that the maximum of the Hamiltonian (2.3) is achieved at the stationary point with respect to $\boldsymbol{c}$, solving the optimization problem is reduced to using the first derivative test to compute the stationary point of each discretized equation (2.8). The main challenge of Algorithm 2.1 is then to solve the spacetime linear systems (Lines 3-4), which is typically large and requires fast and effective solvers.

2.4 Multigrid Methods

In order to solve (2.15) efficiently, we will propose a multigrid method. In particular, the proposed multigrid method is developed on the entire spacetime. There exist multigrid

Algorithm 2.1 Modified policy iteration for solving the joint spacetime system (2.15)

1: Start with an initial guess of $\boldsymbol{c}_h^{(0)}$.
2: **for** $k = 1, 2, ...$ until convergence **do**
3: Solve the KFP block of (2.15), $A_{KFP}(\boldsymbol{c}_h^{(k-1)})\, m_h^{(k)} = b_{KFP}$, for the solution $m_h^{(k)}$.
4: Solve the HJB block of (2.15), $A_{HJB}(\boldsymbol{c}_h^{(k-1)})\, u_h^{(k)} = b_{HJB}(\boldsymbol{c}_h^{(k-1)}, m_h^{(k)})$, for the solution $u_h^{(k)}$.
5: Solve the optimization problem $\boldsymbol{c}_h^{(k)} \equiv \arg\max_{\boldsymbol{c}_h} \left\{ A_{HJB}(\boldsymbol{c}_h) u_h^{(k)} - L(\boldsymbol{c}_h) \right\}$ for the control $\boldsymbol{c}_h^{(k)}$.
6: **end for**
7: Convergent solution: $u_h = u_h^{(k)}$, $m_h = m_h^{(k)}$, $\boldsymbol{c}_h^* = \boldsymbol{c}_h^{(k)}$.

methods that are spacetime. For example, [77] considers spacetime multigrid methods for solving discretized linear parabolic PDEs. The authors propose using a block Jacobi iteration as smoother. We note that the block Jacobi smoother has two deficiencies. One is that it is a block smoother, which requires solving multiple blocks of linear systems at each smoothing step and is thus more expensive than standard pointwise smoothers. The other is that it is a Jacobi smoother, which is less effective than widely-used Gauss-Seidel smoothers in error smoothing [153]. In addition, the spacetime multigrid method in [77] is only developed for linear systems and cannot be directly applied to the nonlinear system (2.15).

Global linearization methods, such as [2, 6, 10, 31], have been proposed for (2.15). They have been discussed in Section 2.1. We remark that other than [2, 6, 10, 31], the modified policy iteration introduced in Algorithm 2.1 can also be a possible choice of the outer nonlinear iteration. We emphasize, however, that all these schemes require multiple layers of iterations: outer nonlinear iterations (Newton, ALG2, primal-dual, policy); and inner multigrid cycles (or inner BiCGStab iterations with multigrid preconditioners). Due to the outer-inner iterative structure of these algorithms, the total number of iterations can be large. We also note that these multigrid schemes are not fully nonlinear, because the inner multigrid cycles are applied to the linearization of (2.15), rather than (2.15) itself.

2.4.1 Full approximation scheme (FAS)

To address the issues of the global linearization multigrid methods, in this chapter, we propose a full approximation scheme [153, 28, 87]. As introduced in Section 1.3.3, FAS is

a family of multigrid schemes that is directly developed for nonlinear discretized systems, and involves only one layer of iterations. Here we give a brief review on how a coarse grid problem of FAS is constructed in general. Denote a nonlinear discretized system as

$$\mathcal{N}_h(u_h) = 0, \tag{2.16}$$

where $\mathcal{N}_h(u_h)$ is the direct discretization of a nonlinear PDE with mesh size h. Then the FAS coarse grid problem is defined by

$$\mathcal{N}_{2h}(u_{2h}) = b_{2h}. \tag{2.17}$$

Here the left hand side $\mathcal{N}_{2h}(u_{2h})$ is the direct discretization of the same PDE with mesh size $2h$; the right hand side is $b_{2h} \equiv \mathcal{N}_{2h}(\bar{u}_{2h}) + R_h r_h$, where R_h is the restriction operator, $\bar{u}_{2h}$ is the injection of the current approximate solution $\bar{u}_h$ from fine grid to coarse grid (i.e., directly copying values onto coarse grid points from the corresponding fine grid points), and r_h is the residual of $\bar{u}_h$. Solving (2.17) yields a coarse grid solution $\hat{u}_{2h}$. Then the coarse grid error estimate is given by $e_{2h} = \hat{u}_{2h} - \bar{u}_{2h}$.

Now we apply the above definition of FAS coarse grid problem to the HJB/KFP system (2.15). We rewrite (2.15) as

$$\mathcal{N}_h(u_h, m_h) = 0, \tag{2.18}$$

where

$$\begin{aligned} \mathcal{N}_h(u_h, m_h) \equiv & \begin{pmatrix} A_{HJB}(\boldsymbol{c}_h^*(u_h)) & \\ & A_{KFP}(\boldsymbol{c}_h^*(u_h)) \end{pmatrix} \begin{pmatrix} u_h \\ m_h \end{pmatrix} - \begin{pmatrix} b_{HJB}(\boldsymbol{c}_h^*(u_h), m_h) \\ b_{KFP} \end{pmatrix} \\ & \text{subject to} \quad \boldsymbol{c}_h^*(u_h) = \arg\max_{\boldsymbol{c}_h} \left\{ A_{HJB}(\boldsymbol{c}_h) u_h - L(\boldsymbol{c}_h) \right\}. \end{aligned} \tag{2.19}$$

If we replace h in (2.19) by $2h$, i.e., if we replace h in the left hand sides of (2.7) and (2.10) by $2h$, then we obtain the nonlinear operator on the coarse grid $\mathcal{N}_{2h}(u_{2h}, m_{2h})$, which defines the FAS coarse grid problem.

2.4.2 Nonlinear smoother

In order to apply the FAS algorithm on the HJB/KFP system, we still need to define a nonlinear smoother. We propose a nonlinear smoother based on the policy iteration. More specifically, in one step nonlinear smoothing, we first fix the control $\boldsymbol{c}_h^*$ in (2.15), obtain the linearized HJB/KFP system and perform smoothing for the linearized problem; then

we update the control by solving the nonlinear optimization problem in (2.15) under the latest u_h.

Regarding the smoother for the linearized problem of (2.15), we consider a spacetime pointwise Gauss-Seidel smoother. That is, we perform forward timestepping for the KFP equation (2.10) and backward timestepping for the HJB equation (2.7), where the pointwise Gauss-Seidel smoother is applied at each timestep. We note that both (2.7) and (2.10) are convection-diffusion problems. In some mean field games, where the convection is guaranteed to be non-negative (such as the demand function in [42] or production quantity in [42, 83], etc.), the smoother at each timestep is the downstream Gauss-Seidel smoother; in other mean field games, where the sign of the convection may change at different grid points, the smoother at each timestep is the four-direction Gauss-Seidel smoother [153]. We note that, if the four-direction smoother is applied, it is only applied in the spatial dimensions; in the time dimension, the smoother remains one-directional.

The pseudo-code for the proposed smoother is given in Algorithm 2.2.

Algorithm 2.2 Joint spacetime pointwise Gauss-Seidel smoother

subroutine $(\bar{u}_h, \bar{m}_h) = \text{SMOOTH}\ (u_h, m_h)$
for $n = 1, \cdots, n_t$ **do**
  Update the control: $\bar{\boldsymbol{c}}_h^n = \arg\max_{\boldsymbol{c}_h^n} \{A_{HJB}^n(\boldsymbol{c}_h^n) u_h^n - L(\boldsymbol{c}_h^n)\}$.
  Apply one step Gauss-Seidel smoother on the linearized KFP equation $A_{KFP}^n(\bar{\boldsymbol{c}}_h^n) m_h^n = \frac{1}{\Delta t} \cdot \bar{m}_h^{n-1}$, which updates the solution $m_h^n \to \bar{m}_h^n$.
end for
for $n = n_t - 1, \cdots, 0$ **do**
  Apply one step Gauss-Seidel smoother on the linearized HJB equation $A_{HJB}^n(\bar{\boldsymbol{c}}_h^n) u_h^n = L(\bar{\boldsymbol{c}}_h^n) + \Phi(\bar{m}_h^n) + \frac{1}{\Delta t} \cdot \bar{u}_h^{n+1}$, which updates the solution $u_h^n \to \bar{u}_h^n$.
end for

2.4.3 Issues

Consider an FAS scheme for the HJB/KFP system, where the smoother is defined in Section 2.4.2, the coarse grid problem is defined in Section 2.4.1, and standard full coarsening, trilinear interpolation, full-weighting restriction are used. It turns out that such a multigrid method does not converge in general. The failure is caused by the following issues:

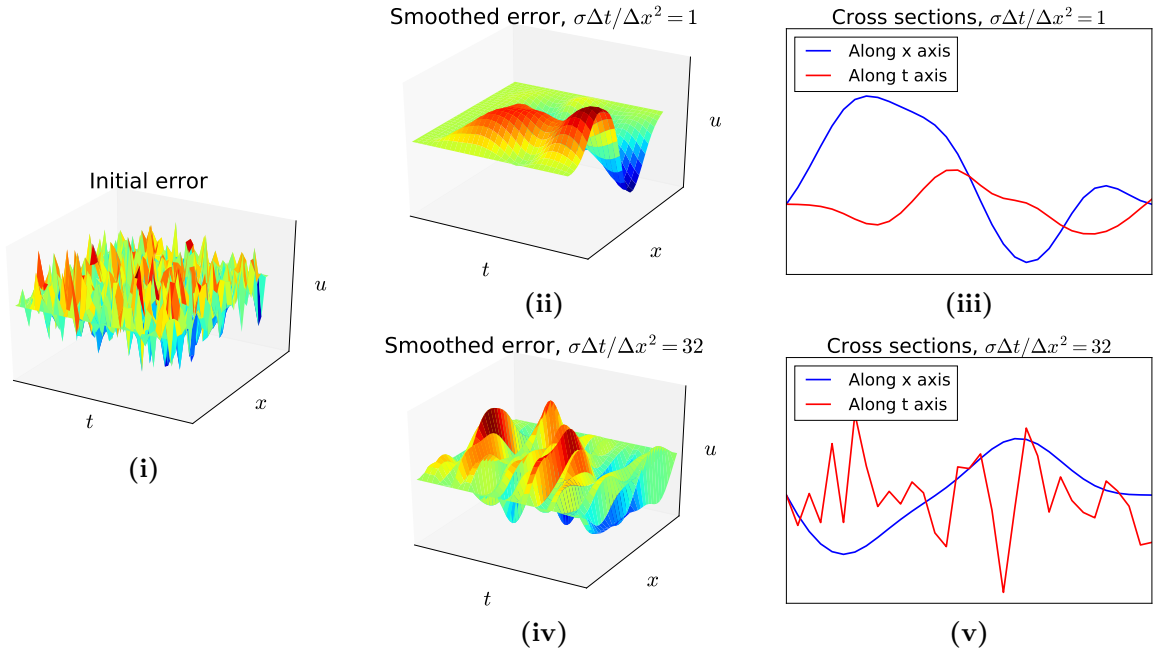


Figure 2.1: Error of the one-dimensional KFP equation with zero convection, i.e., $m_t - \sigma m_{xx} = 0$. **(i)** Initial error; **(ii)** Error after 10 Gauss-Seidel iterations, where $\sigma\Delta t/\Delta x^2 = 1$; **(iii)** Cross sections of the smoothed error (ii), where blue and red lines are the cross sections along x and t axes respectively; **(iv)** Error after 10 Gauss-Seidel iterations, where $\sigma\Delta t/\Delta x^2 = 32$; **(v)** Cross sections of the smoothed error (iv).

- The pointwise Gauss-Seidel smoother does not smooth the error in the time direction, if $\sigma\frac{\Delta t}{\Delta x^2} \gg 1$. Consider the one-dimensional KFP equation with zero convection, i.e., $m_t - \sigma m_{xx} = 0$. Figure 2.1 shows the error after 10 Gauss-Seidel iterations, where $\sigma\frac{\Delta t}{\Delta x^2} = 1$ and 32, respectively. We note that, when $\sigma\frac{\Delta t}{\Delta x^2} = 32$, the error is not smooth in the time dimension; see Figure 2.1(iv)-(v).
- The standard full-weighting restriction does not take into account the one-sided nature of the information propagation in the time dimension.
- The standard full-weighting restriction does not take into account the one-sided nature of the information propagation resulting from the convections in the spatial dimensions. It is well-known that convergence of multigrid deteriorates as the convections increase [153].
- The direct discretization coarse grid operator (2.19) results in a poor coarse grid estimated error. It is shown in [27] that, if direct discretization is used, when the

convection is not aligned with the grid, the convergence factor is no better than 0.5.

In the next few subsections, we will discuss how to address these issues in detail.

2.4.4 Hybrid coarsening

Section 2.4.3 has shown that the pointwise Gauss-Seidel smoother does not smooth the error in the time direction when $\sigma\frac{\Delta t}{\Delta x^2}$ is large. To explain this, consider again the KFP equation with zero convection, which is reduced to a heat equation $m_t - \sigma\Delta m = 0$. When $\sigma\frac{\Delta t}{\Delta x^2} \gg 1$, the discretized heat equation is highly anisotropic. That is, it is strongly connected in the spatial directions but weakly connected in the time direction. It is well-known that pointwise smoothers do not smooth errors in the weakly connected direction [153].

To address this issue, one may use block smoothers, where each block corresponds to the 2-dimensional sub-mesh at each time step [77]. However, each block is a linear system of size $(n_x n_y) \times (n_x n_y)$, and the cost of solving the linear system is as high as $O(\min(n_x^2, n_y^2) n_x n_y)$. The problem is more severe if the dimension of the space is greater than 2.

Alternatively, following the idea in [94, 6, 31], we stick to a pointwise smoother, where the cost is only $O(n_x n_y)$. However, in order to use a pointwise smoother, the coarsening strategy is changed to semi-coarsening. More specifically, the strongly connected dimensions, namely the spatial dimensions, are fully coarsened; the weakly connected dimension, namely the time dimension, remains uncoarsened.

We note that, if we perform semi-coarsening, then $\sigma\frac{\Delta t}{\Delta x^2}$ will decrease on the coarse grids. When $\sigma\frac{\Delta t}{\Delta x^2}$ is no longer large, the pointwise Gauss-Seidel smoother can effectively smooth the error in the time direction as well; see Figure 2.1(ii)-(iii). In this case, coarsening can also be applied in the time direction. As a result, we can combine these two coarsening strategies together. More specifically, when $\sigma\frac{\Delta t}{\Delta x^2}$ is larger than a threshold value (e.g., 1), we use semi-coarsening in the spatial dimensions only; otherwise, we use full-coarsening on the entire spacetime grid. This gives rise to a *hybrid full-semi coarsening scheme*. We remark that even though semi-coarsening has been proposed in [6, 31] for the HJB/KFP system, our strategy is a hybrid coarsening rather than a pure semi-coarsening.

2.4.5 Interpolation

Under the proposed hybrid coarsening strategy, the errors are smooth along the coarsened directions. As a result, to transfer the errors from coarse grids to fine grids, we use the standard trilinear interpolation for full-coarsening and the standard bilinear interpolation for semi-coarsening.

2.4.6 Restriction

We note that the KFP equation is forward in time, while the HJB equation is backward in time. Hence, following [94], we use forward and backward restrictions in the time direction for the residuals of KFP and HJB equations, respectively:

$$\begin{aligned} \text{KFP: } R_h r_h(x_i, y_j, t_n) &= \frac{1}{2} r_h(x_i, y_j, t_{n-1}) + \frac{1}{2} r_h(x_i, y_j, t_n), \\ \text{HJB: } R_h r_h(x_i, y_j, t_n) &= \frac{1}{2} r_h(x_i, y_j, t_n) + \frac{1}{2} r_h(x_i, y_j, t_{n+1}), \\ &\qquad \text{for any coarse grid point} \quad (x_i, y_j, t_n). \end{aligned} \tag{2.20}$$

In this thesis, we use the stencil notation introduced in Section 1.3.4 of [153], which arranges the coefficients of a sparse linear operator in a compact array. Under the stencil notation, the restriction operators are written as

$$\text{KFP: } R_h = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 \end{bmatrix}, \quad \text{HJB: } R_h = \begin{bmatrix} 0 & \frac{1}{2} & \frac{1}{2} \end{bmatrix}, \tag{2.21}$$

where the elements are the coefficients of $[r_h(x_i, y_j, t_{n-1}), r_h(x_i, y_j, t_n), r_h(x_i, y_j, t_{n+1})]$.

The restriction in the spatial dimensions must take into account of the one-sided convectional effect. Let us first consider the spatial restriction for the KFP equation. Following [13], we consider a kernel preserving biased restriction. The idea is to capture the hyperbolic nature of the PDE. The restriction weights are biased towards the upwind side and matched with the flow direction of the error. Such biased restriction may not be unique. A kernel preserving scheme is one type of biased restriction that preserves the kernels of the differential operator of the KFP equation, $-\sigma\Delta m - \nabla \cdot (\boldsymbol{c}m)$. In this case, the kernels are the arbitrary constant and the exponential function $\exp(-\sigma^{-1}\boldsymbol{c} \cdot \boldsymbol{x})$. To preserve the kernels, [13] proposes the following restriction operator when $c_1 c_2 > 0$

$$R_h(\boldsymbol{c}) = \frac{1}{4} \begin{bmatrix} 0 & \frac{1}{1+\exp(-\sigma^{-1}\boldsymbol{c}\cdot(0,h))} & \frac{1}{1+\exp(-\sigma^{-1}\boldsymbol{c}\cdot(h,h))} \\ \frac{1}{1+\exp(-\sigma^{-1}\boldsymbol{c}\cdot(-h,0))} & 1 & \frac{1}{1+\exp(-\sigma^{-1}\boldsymbol{c}\cdot(h,0))} \\ \frac{1}{1+\exp(-\sigma^{-1}\boldsymbol{c}\cdot(-h,-h))} & \frac{1}{1+\exp(-\sigma^{-1}\boldsymbol{c}\cdot(0,-h))} & 0 \end{bmatrix}, \tag{2.22}$$

where the elements are the restriction coefficients of

$$\begin{bmatrix} r_h(x_{i-1}, y_{j+1}, t_n) & r_h(x_i, y_{j+1}, t_n) & r_h(x_{i+1}, y_{j+1}, t_n) \\ r_h(x_{i-1}, y_j, t_n) & r_h(x_i, y_j, t_n) & r_h(x_{i+1}, y_j, t_n) \\ r_h(x_{i-1}, y_{j-1}, t_n) & r_h(x_i, y_{j-1}, t_n) & r_h(x_{i+1}, y_{j-1}, t_n) \end{bmatrix}.$$

Similar operator can be derived for $c_1 c_2 < 0$. We note that, when $c_1 = c_2 = 0$, the restriction operator (2.22) is reduced to a 7-point constant restriction, which is an alternative of the full-weighting restriction. Conversely, when $|c_1|$ and $|c_2|$ are very large, it becomes a pure upwind biased constant restriction. For the HJB equation, the convection coefficient has an opposite sign compared with the KFP equation. To obtain the restriction operator of the HJB equation, we simply replace $\boldsymbol{c}$ in (2.22) by $-\boldsymbol{c}$.

We note that, in [13], the kernel preserving biased restriction is combined with the Petrov-Galerkin coarse grid operator $A_{2h} \equiv R_h A_h P_h$, where R_h is the restriction (2.22) and P_h is the trilinear/bilinear interpolation.

As analyzed in [13], the kernel preserving biased restriction operator has several desirable properties. One is that it captures the one-sided nature of the convections and preserves the kernel of $-\sigma\Delta m - \nabla \cdot (\boldsymbol{c}m)$. In addition, the resulting Petrov-Galerkin coarse grid operators are nearly M-matrices, which is crucial for the stability of multigrid. Conversely, Galerkin coarse grid operators under the standard full-weighting restriction are not M-matrices [153, 13].

The restriction operator on the entire spacetime is the composition of the restriction operator in the spatial dimensions and the one in the time dimension.

2.4.7 Direct discretization and artificial viscosity subtraction

Despite the advantage of the kernel preserving restriction in [13], it combines with the Petrov-Galerkin coarse grid operator, which is incompatible with the nonlinearity of FAS. More specifically, under the FAS framework, the Petrov-Galerkin coarse grid operator is given by $A_{2h}(\boldsymbol{c}) \equiv R_h(\boldsymbol{c}) A_h(\boldsymbol{c}) P_h$, where each matrix entry of $A_{2h}(\boldsymbol{c})$ becomes a nonlinear function that depends on the control $\boldsymbol{c}$. This is much more complicated than the Petrov-Galerkin coarse grid operator for the linear problems considered in [13], where each matrix entry of A_{2h} is a number. As a result, constructing the Petrov-Galerkin coarse grid operator for nonlinear FAS is impractical.

In order to make FAS a practical approach, the FAS literature uses direct discretization as the coarse grid operator; see Section 2.4.1. However, if the direct discretization coarse

grid operator is used, then the kernel preserving restriction does not yield precise coarse grid estimated errors any more.

In this chapter, we propose to modify the direct discretization coarse grid operator, such that it becomes a good approximation to the Petrov-Galerkin coarse grid operator, and thus yields accurate coarse grid estimated errors.

In order to achieve this, we investigate the difference between the Petrov-Galerkin and the direct discretization coarse grid operators. For simplicity, we first consider the one dimensional steady-state linear convection-diffusion equation

$$-\sigma m_{xx} - c m_x = 0, \tag{2.23}$$

where c is a positive constant. Under the stencil notation, the finite difference stencil is given by

$$A_h(c) = \left[-\frac{\sigma}{h^2} \quad \frac{2\sigma}{h^2} + \frac{c}{h} \quad -\frac{\sigma}{h^2} - \frac{c}{h} \right], \tag{2.24}$$

where the elements are the stencil coefficients of $[m_h(x_{i-1}), m_h(x_i), m_h(x_{i+1})]$. If we replace h by $2h$, we obtain the direct discretization coarse grid operator:

$$A_{2h}^{DD}(c) = \left[-\frac{\sigma}{(2h)^2} \quad \frac{2\sigma}{(2h)^2} + \frac{c}{2h} \quad -\frac{\sigma}{(2h)^2} - \frac{c}{2h} \right]. \tag{2.25}$$

Meanwhile, under the standard linear interpolation P_h and the kernel preserving restriction

$$R_h(c) = \frac{1}{2} \left[\frac{1}{1+\exp(\sigma^{-1}ch)} \quad 1 \quad \frac{1}{1+\exp(-\sigma^{-1}ch)} \right], \tag{2.26}$$

the Petrov-Galerkin coarse grid operator is given by

$$\begin{aligned} A_{2h}^{PG}(c) &= R_h(c) A_h(c) P_h \\ &= \left[-\frac{\sigma}{(2h)^2} + \frac{(1-\eta)c}{8h} \quad \frac{2\sigma}{(2h)^2} + \frac{(1+\eta)c}{4h} \quad -\frac{\sigma}{(2h)^2} - \frac{(3+\eta)c}{8h} \right], \end{aligned} \tag{2.27}$$

where $\eta \equiv \tanh(\frac{ch}{2\sigma})$. Then the difference between the two coarse grid operators (2.27) and (2.25) is

$$A_{2h}^{PG}(c) - A_{2h}^{DD}(c) = \frac{1}{2}(1-\eta)ch \cdot \left[\frac{1}{(2h)^2} \quad -\frac{2}{(2h)^2} \quad \frac{1}{(2h)^2} \right]. \tag{2.28}$$

Significantly, this turns out to be the stencil for an $O(h)$ viscosity $\frac{1}{2}(1-\eta)ch\, m_{xx}$. Motivated by this fact, we consider subtracting this $O(h)$ "artificial viscosity" from the direct

discretization coarse grid operator $A_{2h}^{DD}(c)$. This yields the Petrov-Galerkin coarse grid operator $A_{2h}^{PG}(c)$ and thus a more precise coarse grid error estimation.

To summarize, our multigrid scheme for (2.23) is to construct the direct discretization coarse grid operator, but instead of using the original viscosity σ, we use the damped viscosity

$$\hat{\sigma} = \sigma - \frac{1}{2}(1-\eta)|c|h, \tag{2.29}$$

where

$$\eta \equiv \tanh(\frac{|c|h}{2\sigma}). \tag{2.30}$$

Here we put the absolute value on c to generalize the result from $c > 0$ to any c.

Next we consider the two dimensional linear convection-diffusion equation

$$-\sigma_1 m_{xx} - \sigma_2 m_{yy} - c_1 m_x - c_2 m_y = 0. \tag{2.31}$$

We obtain the difference between the two coarse grid operators:

$$\begin{aligned} A_{2h}^{PG}(c) - A_{2h}^{DD}(c) = \; & \frac{1}{4}h \Big[(2-\eta_{12}-\eta_1)|c_1|m_{xx} + (2-\eta_{12}-\eta_2)|c_2|m_{yy} \\ & -\text{sign}(c_1c_2)((\eta_{12}+\eta_2)|c_1| + (\eta_{12}+\eta_1)|c_2|)m_{xy} \\ & -\text{sign}(c_2)2\sigma_1\eta_2 m_{xxy} - \text{sign}(c_1)2\sigma_2\eta_1 m_{xyy} + O(h) \Big], \end{aligned} \tag{2.32}$$

where

$$\eta_1 \equiv \tanh(\frac{|c_1|h}{2\sigma_1}),\ \eta_2 \equiv \tanh(\frac{|c_2|h}{2\sigma_2}),\ \eta_{12} \equiv \tanh(\frac{|c_1|h}{2\sigma_1} + \frac{|c_2|h}{2\sigma_2}). \tag{2.33}$$

Notice that, when $h \to 0$, η_1, η_2 and η_{12} are also $O(h)$. If we assume that $\frac{|c_1|h}{2\sigma_1}$ and $\frac{|c_2|h}{2\sigma_2}$ are not much larger than 1, then (2.32) can be approximated by

$$A_{2h}^{PG}(c) - A_{2h}^{DD}(c) \approx \frac{1}{4}h \Big[(2-\eta_{12}-\eta_1)|c_1|m_{xx} + (2-\eta_{12}-\eta_2)|c_2|m_{yy} \Big], \tag{2.34}$$

where we only keep the viscosity terms in (2.32).

Similar to the one dimensional convection-diffusion equation, for the two dimensional case (2.31), our multigrid scheme is to construct the direct discretization coarse grid operator, where we subtract $O(h)$ artificial viscosity from the original σ_1 and σ_2 and damp them to

$$\hat{\sigma}_1 = \sigma_1 - \frac{1}{4}(2-\eta_{12}-\eta_1)|c_1|h, \quad \hat{\sigma}_2 = \sigma_2 - \frac{1}{4}(2-\eta_{12}-\eta_2)|c_2|h. \tag{2.35}$$

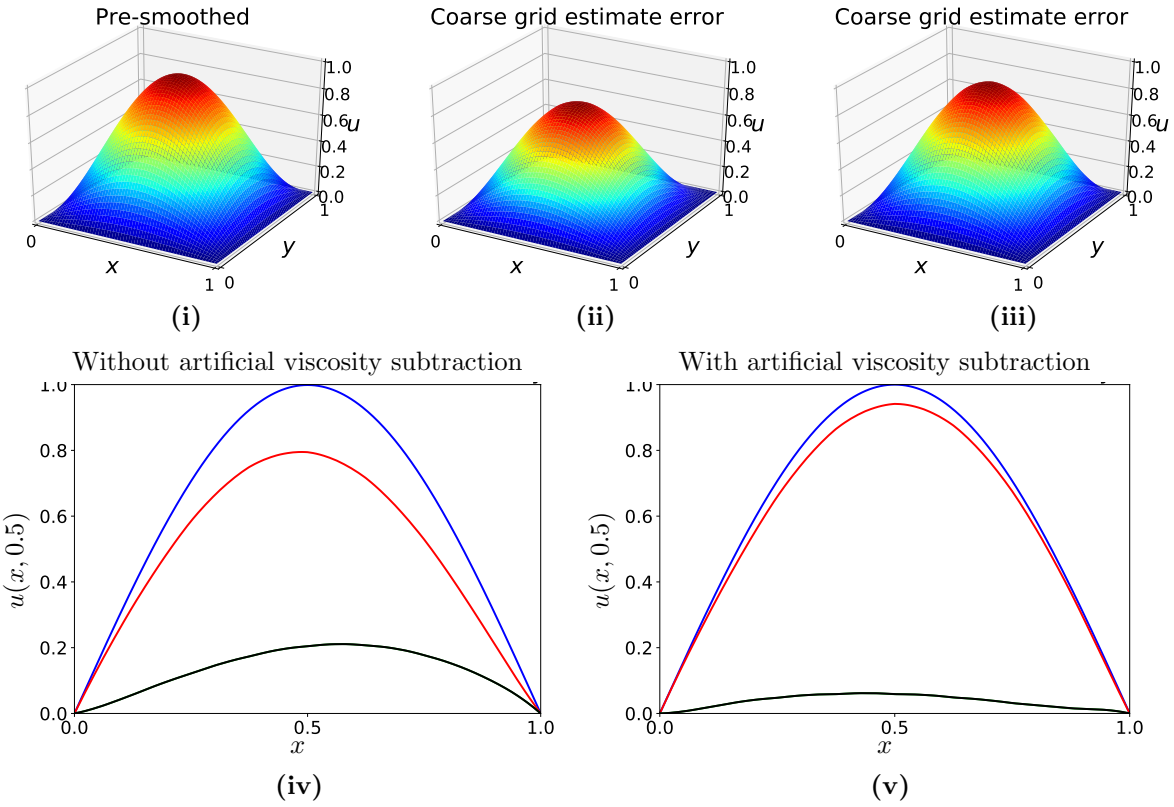


Figure 2.2: An example of the multigrid errors for the two-dimensional convection-diffusion equation (2.31). **(i)** Pre-smoothed error. **(ii)** Coarse grid estimated error *without* artificial viscosity subtraction. **(iii)** Coarse grid estimated error *with* artificial viscosity subtraction. **(iv)** Cross sections of the pre-smoothed error (blue), coarse grid estimated error (red) and post-smoothed error (black) along the x axis *without* artificial viscosity subtraction. **(v)** Cross sections of the corresponding errors along the x axis *with* artificial viscosity subtraction.

Figure 2.2 shows an example of the multigrid errors for the two-dimensional convection-diffusion equation (2.31). By comparing Figures 2.2(iv) and 2.2(v), we observe that with artificial viscosity subtraction, the coarse grid estimated error (red) becomes closer to the pre-smoothed error (blue), and the post-smoothed error (black) becomes smaller. In other words, subtracting artificial viscosity yields a more precise coarse grid estimated error and a more efficient multigrid error reduction.

In this chapter, we extend the proposed idea of artificial viscosity subtraction from the

linear convection-diffusion equations to the nonlinear HJB/KFP system (2.15). We will demonstrate in Section 2.6 that, by subtracting artificial viscosity from direct discretization coarse grid operators, we obtain a more efficient multigrid method for the nonlinear HJB/KFP system.

One may argue that subtracting (rather than adding) artificial viscosity may make the coarse grid problem unstable. However, straightforward algebra can show that the decreased viscosity (2.29) or (2.35) is always non-negative. As a result, the direct discretization coarse grid problem, which uses upwinding discretization, remains stable. In addition, we emphasize that this chapter is concerned with convection-diffusion equations (rather than purely convection equations), where shock waves are not a major concern. In such cases, the benefit of subtracting artificial viscosity, i.e., improving coarse grid correction for multigrid methods and ultimately the overall efficiency of the algorithm, becomes more prominent.

We summarize the proposed artificial viscosity joint spacetime FAS multigrid method in Algorithm 2.3.

2.5 Local Fourier Analysis

In this section, we use Local Fourier Analysis (LFA) [153, 156] to demonstrate the efficiency of the proposed multigrid method. Let us consider the linearized KFP equation

$$m_t - \sigma \Delta m + \nabla \cdot (\boldsymbol{c} m) = 0. \tag{2.36}$$

We assume that $\boldsymbol{c}$ is a positive constant.

2.5.1 Smoothing analysis

We first analyze the smoothing property of the joint spacetime Gauss-Seidel smoother proposed in Section 2.4.2. Since we assume that $\boldsymbol{c} \geq 0$, in the spatial dimensions, it is sufficient to consider the downstream (rather than four-direction) Gauss-Seidel smoother. Define the spacetime Fourier modes as $\varphi_{h,\Delta t}(\boldsymbol{\kappa}; \boldsymbol{x}, t) \equiv \exp\left[i\left(\frac{\kappa_1 x}{h} + \frac{\kappa_2 y}{h} + \frac{\kappa_0 t}{\Delta t}\right)\right]$, where $\boldsymbol{\kappa} = (\kappa_1, \kappa_2, \kappa_0) \in [-\pi, \pi)^3$. Significantly, Fourier modes are the eigenvectors of the smoothing operator, where the corresponding eigenvalues are called the **Fourier symbol** of the smoother. Following [153], we obtain the Fourier symbol of the smoother:

$$\tilde{S}_h(\boldsymbol{\kappa}) = \frac{\frac{\sigma \Delta t}{h^2}(e^{i\kappa_1} + e^{i\kappa_2})}{1 + \frac{\sigma \Delta t}{h^2}(4 + \frac{c_1 h}{\sigma} + \frac{c_2 h}{\sigma}) - \frac{\sigma \Delta t}{h^2}(1 + \frac{c_1 h}{\sigma})e^{-i\kappa_1} - \frac{\sigma \Delta t}{h^2}(1 + \frac{c_2 h}{\sigma})e^{-i\kappa_2} - e^{-i\kappa_0}}.$$

Algorithm 2.3 Artificial viscosity joint spacetime FAS multigrid method

Start with an initial guess $(u_h^{(0)}, m_h^{(0)})$.
for $k = 1, 2, ...$ until convergence **do**
  $(u_h^{(k+1)}, m_h^{(k+1)})$ = FASCYC $(u_h^{(k)}, m_h^{(k)}, 0, \sigma, \sigma, \gamma, \nu_1, \nu_2)$. See below for the subroutine "FASCYC".
end for

subroutine $(u_h^{(k+1)}, m_h^{(k+1)})$ = FASCYC $(u_h^{(k)}, m_h^{(k)}, b_h, (\sigma_1)_h, (\sigma_2)_h, \gamma, \nu_1, \nu_2)$
Construct the nonlinear operator (2.19), $\mathcal{N}_h$, using the viscosity $(\sigma_1)_h$ and $(\sigma_2)_h$.
Perform ν_1 smoothing steps (Algorithm 2.2) on $\mathcal{N}_h(u_h, m_h) = b_h$, which updates the solution $(u_h^{(k)}, m_h^{(k)}) \to (\bar{u}_h^{(k)}, \bar{m}_h^{(k)})$.
Compute the residual: $r_h = b_h - \mathcal{N}_h(\bar{u}_h^{(k)}, \bar{m}_h^{(k)})$.
Determine whether to use full-coarsening or semi-coarsening according to Section 2.4.4.

Inject the solution: $(\bar{u}_h^{(k)}, \bar{m}_h^{(k)}) \to (\bar{u}_{2h}^{(k)}, \bar{m}_{2h}^{(k)})$.
Restrict the residual: $r_{2h} = R_h r_h$, where R_h is the restriction in Section 2.4.6.
Inject the viscosity: $(\sigma_1)_h \to (\sigma_1)_{2h}$, $(\sigma_2)_h \to (\sigma_2)_{2h}$.
Subtract artificial viscosity: $(\sigma_1)_{2h} \leftarrow (\sigma_1)_{2h} - \frac{1}{4}(2 - \eta_{12} - \eta_1)|c_1|h$, $(\sigma_2)_{2h} \leftarrow (\sigma_2)_{2h} - \frac{1}{4}(2 - \eta_{12} - \eta_2)|c_2|h$, where η_1, η_2 and η_{12} are defined in (2.33).
Construct the coarse grid nonlinear operator $\mathcal{N}_{2h}$, using the viscosity $(\sigma_1)_{2h}$ and $(\sigma_2)_{2h}$.

Compute the right hand side: $b_{2h} = \mathcal{N}_{2h}(\bar{u}_{2h}^{(k)}, \bar{m}_{2h}^{(k)}) + R_h r_h$.
if on the coarsest grid **then**
  Solve $\mathcal{N}_{2h}(\hat{u}_{2h}^{(k)}, \hat{m}_{2h}^{(k)}) = b_{2h}$ for $(\hat{u}_{2h}^{(k)}, \hat{m}_{2h}^{(k)})$, using Algorithm 2.1 or Algorithm 2.2 repeatedly.
else
  Solve $\mathcal{N}_{2h}(\hat{u}_{2h}^{(k)}, \hat{m}_{2h}^{(k)}) = b_{2h}$ for $(\hat{u}_{2h}^{(k)}, \hat{m}_{2h}^{(k)})$ approximately by γ-time recursions of
    $(\hat{u}_{2h}^{(k)}, \hat{m}_{2h}^{(k)})$ = FASCYC $(\bar{u}_{2h}^{(k)}, \bar{m}_{2h}^{(k)}, b_{2h}, (\sigma_1)_{2h}, (\sigma_2)_{2h}, \gamma, \nu_1, \nu_2)$.
end if
Compute the coarse grid estimated error: $e_{2h} = (\hat{u}_{2h}^{(k)}, \hat{m}_{2h}^{(k)}) - (\bar{u}_{2h}^{(k)}, \bar{m}_{2h}^{(k)})$.
Interpolate the estimated error: $e_h = P_h e_{2h}$, where P_h is the trilinear or bilinear interpolation.
Correct the fine grid solution: $(\tilde{u}_h^{(k)}, \tilde{m}_h^{(k)}) = (\bar{u}_h^{(k)}, \bar{m}_h^{(k)}) + e_h$.
Perform ν_2 smoothing steps (Algorithm 2.2) on $\mathcal{N}_h(u_h, m_h) = b_h$, which updates the solution $(\tilde{u}_h^{(k)}, \tilde{m}_h^{(k)}) \to (u_h^{(k+1)}, m_h^{(k+1)})$.

(i) Full coarsening

	$(\frac{c_1 h}{\sigma}, \frac{c_2 h}{\sigma}) = (0.1, 0.1)$	$(\frac{c_1 h}{\sigma}, \frac{c_2 h}{\sigma}) = (1, 1)$	$(\frac{c_1 h}{\sigma}, \frac{c_2 h}{\sigma}) = (10, 10)$
$\frac{\sigma \Delta t}{h^2} = 10$	0.951	0.951	0.951
$\frac{\sigma \Delta t}{h^2} = 2$	0.792	0.793	0.784
$\frac{\sigma \Delta t}{h^2} = 0.4$	0.499	0.411	0.433

(ii) Semi coarsening

	$(\frac{c_1 h}{\sigma}, \frac{c_2 h}{\sigma}) = (0.1, 0.1)$	$(\frac{c_1 h}{\sigma}, \frac{c_2 h}{\sigma}) = (1, 1)$	$(\frac{c_1 h}{\sigma}, \frac{c_2 h}{\sigma}) = (10, 10)$
$\frac{\sigma \Delta t}{h^2} = 10$	0.478	0.334	0.083
$\frac{\sigma \Delta t}{h^2} = 2$	0.483	0.339	0.084
$\frac{\sigma \Delta t}{h^2} = 0.4$	0.499	0.354	0.086

Table 2.1: The smoothing factor μ_{loc} for different combinations of $\frac{\sigma \Delta t}{h^2}$ and $(\frac{c_1 h}{\sigma}, \frac{c_2 h}{\sigma})$ and for **(i)** full coarsening, and **(ii)** semi coarsening.

We define its **smoothing factor** as $\mu_{loc} \equiv \sup_{\boldsymbol{\kappa}} \left\{ |\tilde{S}_h(\boldsymbol{\kappa})| : \boldsymbol{\kappa} \in \text{high frequency mode} \right\}$. The smoothing factor evaluates how effectively a smoother can make errors smooth. In particular, a smoothing factor reaches its best value at 0 and worst at 1. One can see that the smoothing factor is determined by three ratios: $\frac{\sigma \Delta t}{h^2}$, $\frac{c_1 h}{\sigma}$ and $\frac{c_2 h}{\sigma}$.

Table 2.1 reports the smoothing factor μ_{loc} under different combinations of $\frac{\sigma \Delta t}{h^2}$ and $(\frac{c_1 h}{\sigma}, \frac{c_2 h}{\sigma})$. We can see that, if full-coarsening is used, then the smoothing factor depends on the ratio $\frac{\sigma \Delta t}{h^2}$. When $\frac{\sigma \Delta t}{h^2}$ is large, the smoothing factor is close to 1. When $\frac{\sigma \Delta t}{h^2}$ is small, the smoothing factor is much smaller than 1. This means that it is desirable to use full-coarsening if and only if $\frac{\sigma \Delta t}{h^2}$ is small. In addition, if semi-coarsening is applied, then the smoothing factor is basically determined by the ratios between convection and diffusion, $(\frac{c_1 h}{\sigma}, \frac{c_2 h}{\sigma})$. The smoothing factor decreases as the ratios increase. Indeed, when the ratios are infinity, or, when the linear problem (2.36) becomes purely hyperbolic, the smoothing factor becomes 0, which seems to suggest that purely hyperbolic problems can be solved by one single Gauss-Seidel iteration. However, this is only true for linear problems. The original KFP equation (2.4) would not be solved by one single Gauss-Seidel iteration due to the nonlinearity and the coupling with the HJB equation (2.1)-(2.2), even when the problem becomes hyperbolic.

2.5.2 Two-grid analysis

In this subsection, we follow [156] and consider a two-grid analysis for the full and semi coarsenings. We note that each individual Fourier mode is no longer an eigenvector of

the two-grid operator. However, given a low frequency mode $\boldsymbol{\kappa}^{000} \equiv \boldsymbol{\kappa} \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right)^3$, the 8-dimensional subspace of $\text{span}\{\, \varphi_{h,\Delta t}(\boldsymbol{\kappa}^{\boldsymbol{\alpha}}; \cdot) : \boldsymbol{\alpha} = (\alpha_1, \alpha_2, \alpha_0),\ \alpha_1, \alpha_2, \alpha_0 \in \{0, 1\} \,\}$, where $\boldsymbol{\kappa}^{\boldsymbol{\alpha}} \equiv \boldsymbol{\kappa}^{000} - (\alpha_1 \text{sign}(\kappa_1), \alpha_2 \text{sign}(\kappa_2), \alpha_0 \text{sign}(\kappa_0)) \cdot \pi$, is invariant under the two-grid operator. As a result, the Fourier symbol of the two grid operator, $M_h^{2h}(\boldsymbol{\kappa}) \in \mathbb{C}^{8\times 8}$, is not a scalar, but rather an 8×8 matrix. We refer readers to Appendix A.3 for technical details.

Based on $M_h^{2h}(\boldsymbol{\kappa})$, we define the **asymptotic convergence factor** and the **error reduction factor** as

$$\rho_{loc}(M_h^{2h}) \equiv \sup_{\boldsymbol{\kappa}} \left\{ \rho(M_h^{2h}(\boldsymbol{\kappa})) : \boldsymbol{\kappa} \in \text{low frequency mode} \right\},$$
$$\sigma_{loc}(M_h^{2h}) \equiv \sup_{\boldsymbol{\kappa}} \left\{ \|M_h^{2h}(\boldsymbol{\kappa})\|_2 : \boldsymbol{\kappa} \in \text{low frequency mode} \right\}.$$

As suggested by their names, these two factors evaluate how effective a two-grid method reduces errors, or equivalently, how fast the method converges to the solution. Similar to the smoothing factor, the asymptotic convergence factor and the error reduction factor reach their best values at 0 and worst at 1, and are determined by the three ratios $\frac{\sigma\Delta t}{h^2}$, $\frac{c_1 h}{\sigma}$ and $\frac{c_2 h}{\sigma}$.

Table 2.2 reports the two-grid convergence factor $\rho_{loc}(M_h^{2h})$ and error reduction factor $\sigma_{loc}(M_h^{2h})$ under different combinations of $\frac{\sigma\Delta t}{h^2}$ and $(\frac{c_1 h}{\sigma}, \frac{c_2 h}{\sigma})$. Table 2.2(i) shows that, if full-coarsening is used, then the factors depend on the ratio $\frac{\sigma\Delta t}{h^2}$. When $\frac{\sigma\Delta t}{h^2}$ is large, the

(i) Full coarsening

	$(\frac{c_1 h}{\sigma}, \frac{c_2 h}{\sigma}) = (0.1, 0.1)$		$(\frac{c_1 h}{\sigma}, \frac{c_2 h}{\sigma}) = (1, 1)$		$(\frac{c_1 h}{\sigma}, \frac{c_2 h}{\sigma}) = (10, 10)$	
	$\rho_{loc}(M_h^{2h})$	$\sigma_{loc}(M_h^{2h})$	$\rho_{loc}(M_h^{2h})$	$\sigma_{loc}(M_h^{2h})$	$\rho_{loc}(M_h^{2h})$	$\sigma_{loc}(M_h^{2h})$
$\frac{\sigma\Delta t}{h^2} = 10$	0.89	0.97	0.89	0.97	0.89	0.97
$\frac{\sigma\Delta t}{h^2} = 2$	0.54	0.67	0.59	0.67	0.51	0.67
$\frac{\sigma\Delta t}{h^2} = 0.4$	0.50	0.52	0.56	0.60	0.50	0.52

(ii) Semi coarsening

	$(\frac{c_1 h}{\sigma}, \frac{c_2 h}{\sigma}) = (0.1, 0.1)$		$(\frac{c_1 h}{\sigma}, \frac{c_2 h}{\sigma}) = (1, 1)$		$(\frac{c_1 h}{\sigma}, \frac{c_2 h}{\sigma}) = (10, 10)$	
	$\rho_{loc}(M_h^{2h})$	$\sigma_{loc}(M_h^{2h})$	$\rho_{loc}(M_h^{2h})$	$\sigma_{loc}(M_h^{2h})$	$\rho_{loc}(M_h^{2h})$	$\sigma_{loc}(M_h^{2h})$
$\frac{\sigma\Delta t}{h^2} = 10$	0.22	0.29	0.22	0.28	0.18	0.19
$\frac{\sigma\Delta t}{h^2} = 2$	0.23	0.29	0.22	0.28	0.18	0.19
$\frac{\sigma\Delta t}{h^2} = 0.4$	0.26	0.30	0.22	0.34	0.18	0.19

Table 2.2: The two-grid convergence factor $\rho_{loc}(M_h^{2h})$ and error reduction factor $\sigma_{loc}(M_h^{2h})$ for different combinations of $\frac{\sigma\Delta t}{h^2}$ and $(\frac{c_1 h}{\sigma}, \frac{c_2 h}{\sigma})$ and for **(i)** full coarsening, and **(ii)** semi coarsening.

factors are close to 1. When $\frac{\sigma\Delta t}{h^2}$ is small, the factors are much smaller than 1. Again, this suggests an efficient error reduction under full-coarsening if and only if $\frac{\sigma\Delta t}{h^2}$ is small. Table 2.2(ii) shows that, if semi-coarsening is applied, then the factors are mainly determined by the ratios $(\frac{c_1 h}{\sigma}, \frac{c_2 h}{\sigma})$. The factors decrease as $(\frac{c_1 h}{\sigma}, \frac{c_2 h}{\sigma})$ increase. However, similar to the discussion at the end of Section 2.5.1, this does not imply a more efficient error reduction in a convection-dominant regime than in a diffusion-dominant regime, due to the nonlinearity of the HJB/KFP system. Interested readers are referred to [69] for more discussions on the relation between spacetime multigrid convergence results and the corresponding LFA estimate.

2.5.3 The effect of subtracting artificial viscosity

Our proposed multigrid method considers subtracting artificial viscosity. For simplicity, here we consider the steady-state version of (2.36), or equivalently, $-\sigma\Delta m + \nabla\cdot(\boldsymbol{c}m) = 0$. In this subsection, we will use multigrid (i.e., three-grid, four-grid, etc) analysis to illustrate the improvement due to subtracting artificial viscosity. An introduction to multigrid analysis can be found in [156].

Table 2.3 reports the multigrid asymptotic convergence factors and error reduction factors for the linearized KFP equation. Figure 2.3 illustrates the multigrid asymptotic convergence factors $\rho_{loc}(M)$ versus the convection coefficient $\boldsymbol{c}$. We compare the results with and without artificial viscosity subtraction. To summarize, the improvements of subtracting artificial viscosity include the following:

(i) $h = \frac{1}{64}$, four-grid

	$\boldsymbol{c} = (20, 20)^T$		$\boldsymbol{c} = (30, 30)^T$	
	$\rho_{loc}(M_h^{8h})$	$\sigma_{loc}(M_h^{8h})$	$\rho_{loc}(M_h^{8h})$	$\sigma_{loc}(M_h^{8h})$
Without artificial viscosity subtraction	0.34	0.40	0.43	0.47
With artificial viscosity subtraction	**0.20**	**0.32**	**0.33**	**0.39**

(ii) $h = \frac{1}{128}$, five-grid

	$\boldsymbol{c} = (20, 20)^T$		$\boldsymbol{c} = (30, 30)^T$	
	$\rho_{loc}(M_h^{16h})$	$\sigma_{loc}(M_h^{16h})$	$\rho_{loc}(M_h^{16h})$	$\sigma_{loc}(M_h^{16h})$
Without artificial viscosity subtraction	0.39	0.44	0.48	0.51
With artificial viscosity subtraction	**0.23**	**0.35**	**0.38**	**0.43**

Table 2.3: Multigrid asymptotic convergence factors and error reduction factors for $-\sigma\Delta m + \nabla\cdot(\boldsymbol{c}m) = 0$, where $\sigma = 1$.

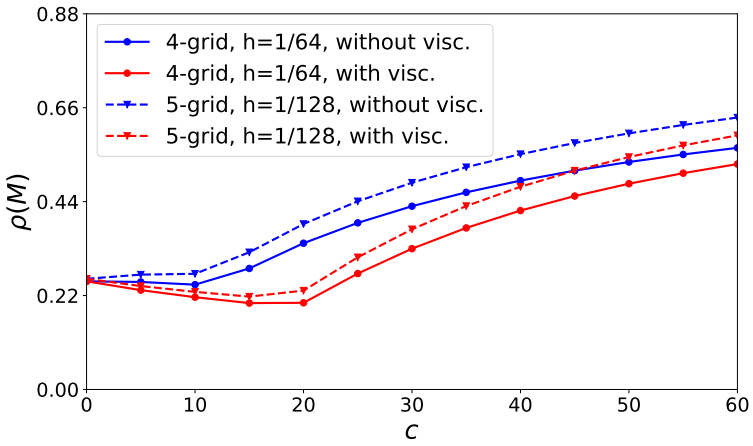


Figure 2.3: The multigrid asymptotic convergence factors $\rho_{loc}(M)$ versus the convection coefficient $\boldsymbol{c}$ for the two-dimensional convection-diffusion equation $-\sigma\Delta m + \nabla\cdot(\boldsymbol{c}m) = 0$, where $\sigma = 1$ and $\boldsymbol{c} = (0,0), (5,5), (10,10), \cdots$. The blue lines are the convergence factors *without* artificial viscosity subtraction, while the red lines are the corresponding convergence factors *with* artificial viscosity subtraction.

- The significant improvement of the multigrid convergence factors occurs approximately between $\boldsymbol{c} = (10,10)^T$ and $(30,30)^T$. By subtracting artificial viscosity, the multigrid factors are significantly reduced by 20%-40%.
- When $\boldsymbol{c}$ is close to $(0,0)^T$, or when the problem is diffusion dominant, subtracting artificial viscosity does not have a significant impact on the multigrid factors, because the original direct discretization already yields good error estimations.
- When $\boldsymbol{c}$ is greater than $(40,40)^T$, or when the problem is convection dominant, the multigrid factors have modest improvements if artificial viscosity is subtracted. We note that as the convection increases, the ratios $(\frac{|c_1|h}{\sigma}, \frac{|c_2|h}{\sigma})$ also increase. The terms dropped out of (2.32) are no longer negligible.
- For $h = \frac{1}{64}$ and $h = \frac{1}{128}$, subtracting artificial viscosity yields approximately the same amount of improvement on the convergence factors.

The LFA provides an estimate on how many iterations are saved by subtracting artificial viscosity. Denote the convergence (or error reduction) factors with and without artificial viscosity subtraction as ρ_{yes} and ρ_{no} (or σ_{yes} and σ_{no}), respectively. Using the reported numbers in Table 2.3, the ratio of the numbers of iterations with and without artificial viscosity subtraction is given by $\log\rho_{\text{no}}/\log\rho_{\text{yes}} \approx 70\%$ (or $\log\sigma_{\text{no}}/\log\sigma_{\text{yes}} \approx 80\%$). That

is, subtracting artificial viscosity may save 2-3 iterations per 10 iterations, or 20%-30% of computational cost. We will show in Section 2.6 that the numerical simulation agrees with this LFA estimate.

2.6 Numerical Results

In this section, we apply our proposed spacetime multigrid method to the joint HJB/KFP system (2.1)-(2.4), or equivalently, its discretization (2.15). We illustrate the fast and mesh-independent convergence rates.

Unless otherwise specified, we use the V(1,1)-cycle [153]. That is, we choose $\nu_1 = \nu_2 = \gamma = 1$ in Algorithm 2.3, or equivalently, we perform one pre and post smoothings respectively, and perform multigrid recursion only once on each coarse grid. We terminate the multigrid iterations at the residual norm $\|r_h\| \leq 10^{-6}$. The initial guesses for the grid size (n_x, n_y, n_t) are the trilinear interpolation of the solutions from the grid size $(n_x/2, n_y/2, n_t/2)$.

In our numerical examples, we compare the following multigrid schemes:

Scheme I (our proposed scheme) is the spacetime FAS scheme for the HJB/KFP system (2.15), where artificial viscosity is subtracted from the direct discretization coarse grid operators. The number of iterations is counted.

Scheme II is the same as Scheme I, except that *no* artificial viscosity is subtracted from the direct discretization coarse grid operators.

Scheme III is the spacetime FAS scheme, where we use the multigrid components proposed in [77]. More specifically, we apply the block Jacobi smoother and the coarsening strategy proposed in [77], with the full-weighting restriction and the trilinear interpolation. We note that the scheme in [77] is developed for linear problems and cannot be directly applied to nonlinear problems. We use FAS with direct discretization operators to adapt this scheme for the nonlinear HJB/KFP system. The number of iterations is counted.

Scheme IV is the global linearization spacetime multigrid method, where the outer loop is the modified policy iteration described in Algorithm 2.1, and the inner loop is multigrid V(1,1)-cycle for each linearized problem. We use the proposed multigrid components described in Section 2.4. The only exception is that we use Petrov-Galerkin coarse grid operator. The reason is that the inner loop is linear, which allows us to use the Petrov-Galerkin operator, and it is generally accurate. The number of iterations is defined as the sum of the numbers of inner loops.

Scheme V is the forward/backward timestepping fixed point iteration described in Algorithm A.3. Each timestep (Lines 6 and 9) is solved by the FAS scheme using our proposed multigrid components described in Section 2.4. We note that the FAS here is applied on each timestep rather than on the entire spacetime domain. In order to make a fair comparison between Scheme V and the spacetime Schemes I-IV, we define the number of iterations for Scheme V as the average number of iterations per timestep, namely, $1/n_t\times$ sum of the FAS V-cycle counts over all the n_t timesteps.

For all these five multigrid schemes, we note that the computational costs per iteration are dominated by the costs of pre and post smoothings. In other words, the computational costs of restrictions and interpolations are negligible. Hence, the complexities per iteration are the same for Schemes I, II, IV and V. As a result, the number of iterations is a good measure for the complexity of each multigrid scheme. We note that, for Scheme III, since a block smoother is used, the complexity per iteration is greater than the other schemes.

Example 2.1. Consider solving the (2+1)-dimensional mean field games in Section 5 of [83]. That is, we solve (2.1)-(2.4), where the spacetime domain is $T = 1$, $\Omega = [0, 1]^2$, the cost function is $\Phi(m) = \ln(m)$, and the Lagrangian is $L(\boldsymbol{c}) = -\frac{\|\boldsymbol{c}\|^2}{2}$. Section 5 of [83] derives the following exact solution:

$$u = -a(x^2 + y^2) + b, \quad m = \frac{a}{\pi\sigma} \exp\left(-\frac{a(x^2 + y^2)}{\sigma}\right), \tag{2.37}$$

where $a = \frac{1}{2\sigma} - \frac{r}{2}$, $b = \frac{1}{r}\left(\ln\frac{a}{\pi\sigma} - 4a\sigma\right)$. Here we impose Dirichlet boundary conditions for both u and m, and let the terminal, initial and boundary conditions be given by (2.37). We set the discount factor as $r = 0.1$. We note that the convection, given by $\boldsymbol{c}^* = -\nabla u = 2a(x, y)^T$, is proportional to a. We test the following two cases.

Case 1: $(\sigma, a) = (1, 0.45)$, which is diffusion dominant. The first two columns of Table 2.4 report the convergence rates of the numerical solutions towards the exact solution (2.37). The convergence rates of $\|u - u_h\|$ and $\|m - m_h\|$ are first order, namely, $O(h)$.

We then investigate the convergence rates of the five multigrid schemes; see the last five columns of Table 2.4. Scheme I takes only 4 iterations to converge. In addition, the CPU time for Scheme I is approximately linear in the grid size, namely $O(n_x n_y n_t)$. The convergence rates of Scheme II are basically the same as Scheme I. The reason is that the problem is diffusion dominant, and thus direct discretization without artificial viscosity subtraction yields sufficiently good coarse grid error estimations. Scheme I converges faster than Scheme III. Figure 2.5 explains the reason. Starting with the same initial error

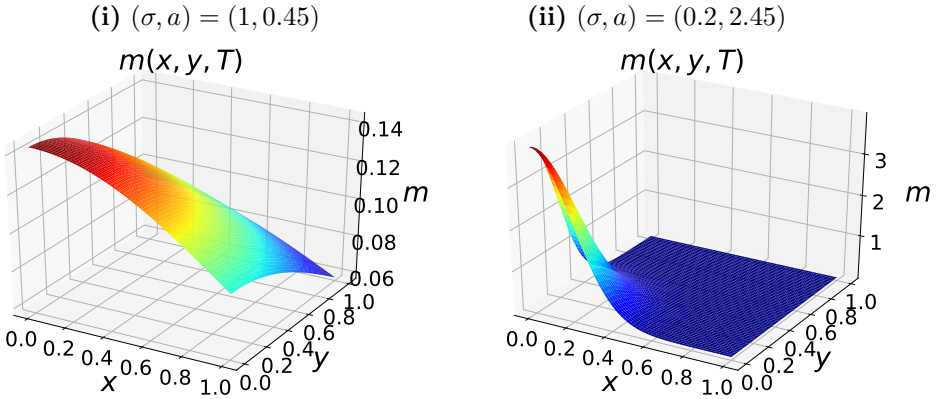


Figure 2.4: Example 2.1: Numerical solutions $m_h(x, y, T)$. **(i)** $(\sigma, a) = (1, 0.45)$. **(ii)** $(\sigma, a) = (0.2, 2.45)$.

$(\sigma, a) = (1, 0.45)$

$n_x \times n_y \times n_t$	Error of HJB/KFP		Number of iterations (CPU time)				
	$\|u - u_h\|$	$\|m - m_h\|$	**Scheme I**	Scheme II	Scheme III	Scheme IV	Scheme V
$16 \times 16 \times 16$	1.11×10^{-3}	2.88×10^{-4}	**4 (3.0s)**	4 (2.9s)	12 (18s)	28 (8.3s)	21 (8.4s)
$32 \times 32 \times 32$	5.73×10^{-4}	1.49×10^{-4}	**4 (23s)**	4 (24s)	11 (219s)	28 (74s)	21 (42s)
$64 \times 64 \times 64$	2.91×10^{-4}	7.61×10^{-5}	**4 (193s)**	4 (188s)	11 (3404s)	21 (491s)	19 (245s)
$128 \times 128 \times 128$	1.47×10^{-4}	3.84×10^{-5}	**3 (1104s)**	4 (1686s)	11 (39616s)	15 (3507s)	19 (2125s)

Table 2.4: Example 2.1: Convergence of the five multigrid schemes. $(\sigma, a) = (1, 0.45)$.

(yellow lines) for both schemes, we observe that Scheme I's pre-smoothed error (blue lines) is smoother than Scheme III's, which leads to a more precise coarse grid estimate (red lines), a smaller post-smoothed error (purple lines), a more efficient error reduction, and eventually a faster convergence. In addition, the cost of Scheme I's pointwise smoother is only $O(n_x n_y n_t)$, while the cost of Scheme III's block smoother is $O(\min(n_x^2, n_y^2) n_x n_y n_t)$. We note that Schemes IV and V are both outer-inner iterations, where the total iteration count is the number of outer iterations times the average number of inner iterations; conversely, Scheme I is a single-layer iteration, where the total iteration count is the FAS iteration count itself. The inner iteration counts of Schemes IV and V are approximately equal to the iteration count of Scheme I. As a result, the total iteration counts of Schemes IV and V are much higher than Scheme I. Comparing with Schemes III-V, we conclude that the proposed Scheme I has the fastest convergence rates.

Case 2: $(\sigma, a) = (0.2, 2.45)$, which is convection dominant. The first two columns of Table 2.5 show that the numerical solutions converge to the exact solution as $h \to 0$. We

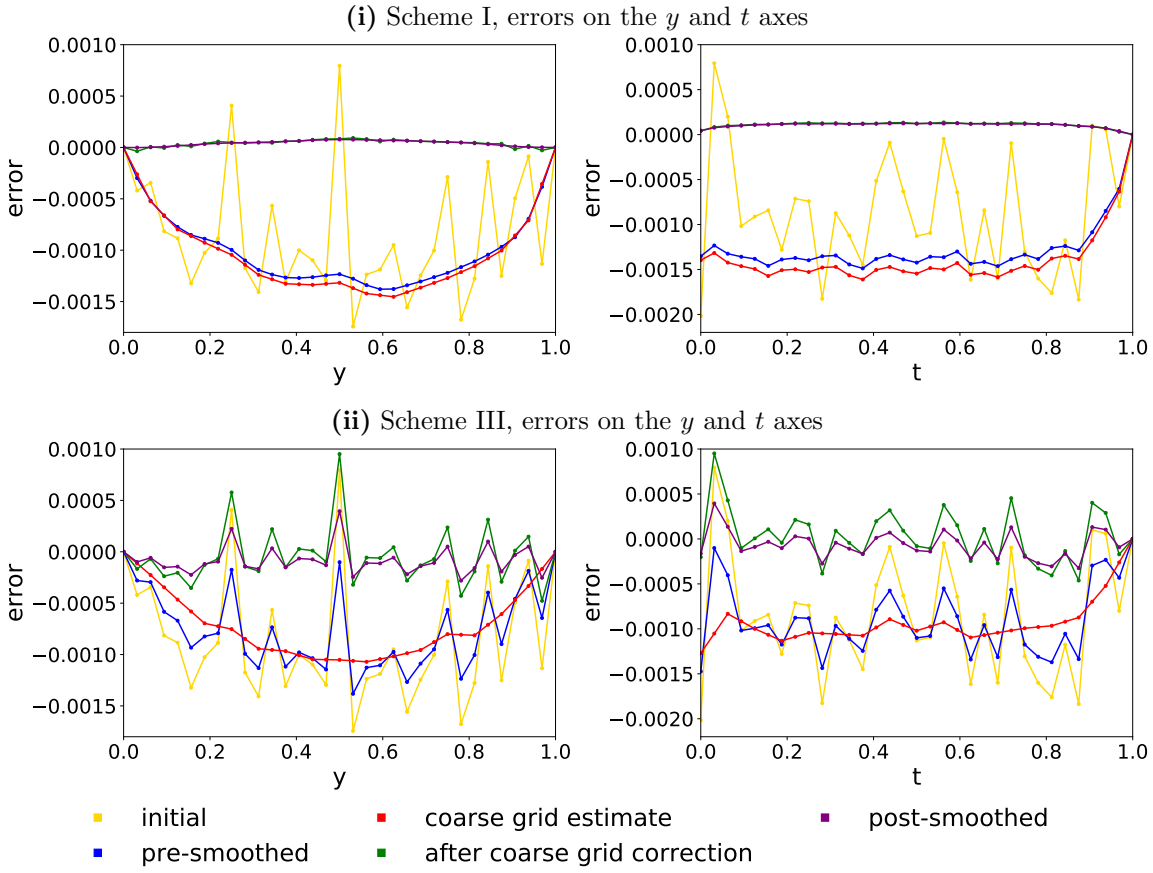


Figure 2.5: Comparison of the error reductions between Scheme I and Scheme III. Here we consider the errors of the HJB equation in Example 2.1 under one multigrid V-cycle, and plot the cross sections of the errors along the y and t axes.

$(\sigma, a) = (0.2, 2.45)$

$n_x \times n_y \times n_t$	Error of HJB/KFP		Number of iterations (CPU time)				
	$\|u - u_h\|$	$\|m - m_h\|$	**Scheme I**	Scheme II	Scheme III	Scheme IV	Scheme V
$16 \times 16 \times 16$	3.68×10^{-2}	4.17×10^{-2}	**6 (5.0s)**	7 (5.9s)	18 (29s)	64 (32s)	69 (24s)
$32 \times 32 \times 32$	2.65×10^{-2}	2.46×10^{-2}	**7 (45s)**	10 (66s)	23 (474s)	75 (340s)	78 (137s)
$64 \times 64 \times 64$	1.70×10^{-2}	1.35×10^{-2}	**7 (349s)**	∞	∞	86 (3145s)	80 (1032s)
$128 \times 128 \times 128$	9.73×10^{-3}	7.04×10^{-3}	**7 (2554s)**	∞	∞	96 (29381s)	77 (9400s)

Table 2.5: Example 2.1: Convergence of the five multigrid schemes. $(\sigma, a) = (0.2, 2.45)$.

note that $m(1,1,t) \approx 0$; see Figure 2.4(ii). Hence in (2.1), $\ln(m(1,1,t)) \approx -\infty$. Due to this singularity, the convergence rates of $\|u - u_h\|$ and $\|m - m_h\|$ are slightly slower than $O(h)$.

The last five columns of Table 2.5 report the convergence rates of the five multigrid schemes. The numbers of iterations for $(\sigma, a) = (0.2, 2.45)$ are larger than those for $(\sigma, a) = (1, 0.45)$. The reason is that multigrid is usually less efficient when the problem becomes more hyperbolic. Significantly, Scheme I converges in 6-7 iterations, independent of the mesh size. The total complexity (reflected by the total CPU time) is linear in the grid size, namely $O(n_x n_y n_t)$. By comparing Schemes I and II, we see that subtracting artificial viscosity saves 14% and 30% of the iterations when the mesh size is $16 \times 16 \times 16$ and $32 \times 32 \times 32$, respectively. This agrees with the LFA estimate, which is around 20%-30% (see the end of Section 2.5.3). More importantly, Scheme II fails to converge when the mesh size is larger than $64 \times 64 \times 64$. However, Scheme I successfully converges in only 7 iterations, and the convergence rate is mesh-independent. Scheme III also fails to converge. The reason, which has been explained in Figure 2.5, is that Scheme III's errors are oscillatory. The oscillation grows quickly as the iteration proceeds. Due to the outer-inner structure of the iterations, the iteration counts for Schemes IV and V are above 64, much larger than Scheme I. In addition, the number of iterations for Scheme IV grows as the grid becomes finer.

Example 2.2. In this example, we compare the numerical results given by our proposed multigrid method with the results given by the numerical scheme in [2]. Consider solving the HJB/KFP system in Section 4.2 of [2]. That is, we solve (2.1)-(2.4), where the spacetime domain is $T = 1$, $\Omega = [-0.5, 0.5]^2$, the cost function is $\Phi(m) = m$, the terminal condition is $u(x, y, T) = 0$, the initial condition is $m(x, y, 0) = 1$, the boundary conditions for both u and m are periodic, and the Lagrangian is $L(\boldsymbol{c}) = \frac{2\sqrt{3}}{9}\|\boldsymbol{c}\|^{3/2} - \sin(2\pi x) - \sin(2\pi y) - \cos(4\pi x)$. Straightforward algebra can show that under this Lagrangian, the optimal control is $\boldsymbol{c}^* = 3\|\nabla u\|\nabla u$. This yields the same Hamiltonian $H(\boldsymbol{x}, \nabla u) = \|\nabla u\|^3 + \sin(2\pi x) + \sin(2\pi y) + \cos(4\pi x)$ as the one in [2].

We test our proposed multigrid method under different diffusion parameters; see Table 2.6. When $\sigma = 0.12, 0.046$, the problem is convection dominant and poses a challenge to multigrid methods. When the spatial grid is coarser than 8×8, i.e., $h \geq \frac{1}{8}$, the convection-diffusion ratios $(\frac{|c_1|h}{\sigma}, \frac{|c_2|h}{\sigma})$ are much larger than 1. As discussed in Section 2.4.7, subtracting artificial viscosity is efficient under the assumption that $(\frac{|c_1|h}{\sigma}, \frac{|c_2|h}{\sigma})$ are

$n_x \times n_y \times n_t$	Total number of iterations				
	$\sigma = 0.6$	$\sigma = 0.36$	$\sigma = 0.2$	$\sigma = 0.12$	$\sigma = 0.046$
$32 \times 32 \times 32$	**4**	**4**	**5**	**6**	**6**
$64 \times 64 \times 64$	**4**	**5**	**5**	**7**	**8**
$128 \times 128 \times 128$	**4**	**5**	**5**	**7**	**8**

Table 2.6: Example 2.2: Convergence of our proposed multigrid schemes. Total number of iterations is counted.

$n_x \times n_y \times n_t$	Average numbers of BiCGStab iterations per Newton loop				
	$\sigma = 0.6$	$\sigma = 0.36$	$\sigma = 0.2$	$\sigma = 0.12$	$\sigma = 0.046$
$32 \times 32 \times 32$	2	2	3.5	6	10
$64 \times 64 \times 64$	2	2	3.5	6	10
$128 \times 128 \times 64$	2	2	4	6.1	10

Table 2.7: Average (on the Newton loop) numbers of BiCGStab iterations given by the algorithm in [2]. We note that the total numbers of iterations of the algorithm = the numbers of Newton loops $\times$ the average numbers of BiCGStab iterations per Newton loop, which can be much higher than the numbers listed in the table.

not much larger than 1. To further improve the error estimations, we consider using W-cycles (i.e., $\gamma = 2$ in Algorithm 2.3) on the coarsest grids where $n_x \times n_y$ is coarser than 8×8. W-cycles are more expensive than V-cycles. However, we emphasize that, for the grids where $n_x \times n_y$ is finer than 8×8, we still apply V-cycles. Since W-cycles are only applied on the very coarsest grids, the extra computational cost is negligible.

Table 2.6 shows that our proposed multigrid method converges in around 4-8 iterations in total; the convergence rates are approximately mesh-independent.

Table 2.7 shows the numbers of inner multigrid preconditioned BiCGStab iterations per outer Newton's iteration, as reported in [2]. However, the number of Newton's iterations is not reported in [2]. We note that the total number of iterations, which is the product of the outer Newton's iteration counts and the inner BiCGStab iteration counts, can be much higher than the numbers listed in the table. If the number of Newton's iterations is greater than 2, then the total number of iterations by the method in [2] can be higher than our proposed method. Another observation of Table 2.7 is that as the diffusion decreases, the convergence rates of the algorithm in [2] deteriorate quickly from 2 to 10. However, using our proposed multigrid method (Table 2.6), the increase of iteration counts is modest, from 4 to 8.

Example 2.3. We note that (2.1)-(2.4) only involves local coupling. In this example, we extend our method to an HJB/KFP system with nonlocal coupling, which arises from the Bertrand mean field games proposed in [42]. To give a quick review on the mathematical formulation, consider again a competitive market with a large number of companies. Each company, distinguished by its capacity $x \in [0, \infty)$, evolves over time $t \in [0, T]$. Let $u(x, t)$ be the optimal expected profit (value function) of each company over the period $[t, T]$. Let $p(x, t)$ be the price of each company's product (control). Let $m(x, t)$ be the distribution of the companies. The HJB equation reads

$$\begin{aligned} -u_t - \sigma u_{xx} - D(p^*, \bar{p}(m), \eta(m))(p^* - u_x) + ru = 0, \\ u(x, T) = 0, \quad u(0, t) = 0, \quad u_x(\infty, t) = 0, \end{aligned} \tag{2.38}$$

subject to the optimal control

$$p^* = \arg\max_{p \geq 0} \{D(p, \bar{p}, \eta)(p - u_x)\}. \tag{2.39}$$

Here r is the interest rate, and σ is the randomness of the demand in the market. In this HJB equation, the total number of companies $\eta : t \to \eta(m)(t)$, the market average price $\bar{p} : t \to \bar{p}(m)(t)$ and the demand function $D : (x, t) \to D(p, \bar{p}(m), \eta(m))(x, t)$ are given by

$$\begin{aligned} \eta(t) = \int_0^\infty m(x, t)dx, \quad \bar{p}(t) = \frac{1}{\eta(t)} \int_0^\infty p^*(x, t)m(x, t)dx, \\ D(x, t) = s\left(\frac{1}{1 + \epsilon\eta(t)} - p(x, t) + \frac{\epsilon\eta(t)}{1 + \epsilon\eta(t)}\bar{p}(t)\right), \end{aligned} \tag{2.40}$$

where ϵ and s are constants. The corresponding KFP equation reads

$$\begin{aligned} m_t - \sigma m_{xx} - [D(p^*, \bar{p}(m), \eta(m))m]_x = 0, \\ m(x, 0) = m_0(x), \quad m_x(0, t) = 0, \quad m(\infty, t) = 0, \end{aligned} \tag{2.41}$$

where $m_0(x)$ is the initial distribution (e.g. $m_0(x) = 1 - B(x; 2, 4)$, where B is the cumulative beta distribution function). We refer readers to [82] for the well posedness of the problem (2.38)-(2.41). We note that the coupling between the HJB equation (2.38)-(2.39) and the KFP equation (2.41) is more complicated than Equations (2.1)-(2.4). More specifically, (2.1)-(2.4) only involves local couplings, i.e., the convection coefficient is the local optimal control $\boldsymbol{c}^*$ and the cost function Φ is also local. However, in (2.38)-(2.41), the convection coefficient D is a functional of both p^* and m in the form of nonlocal integrals.

We consider numerically solving (2.38)-(2.41). We first verify the correctness of our nonlinear solver by a simulation under the parameters[2] in [42], i.e., $r = 0.2$, $s = 1$, $\sigma = 0$

[2] $\sigma = 0.005$ in Equation (2.38)-(2.41) corresponds to $\sigma = 0.1$ (namely $\sigma^2/2 = 0.005$) in [42].

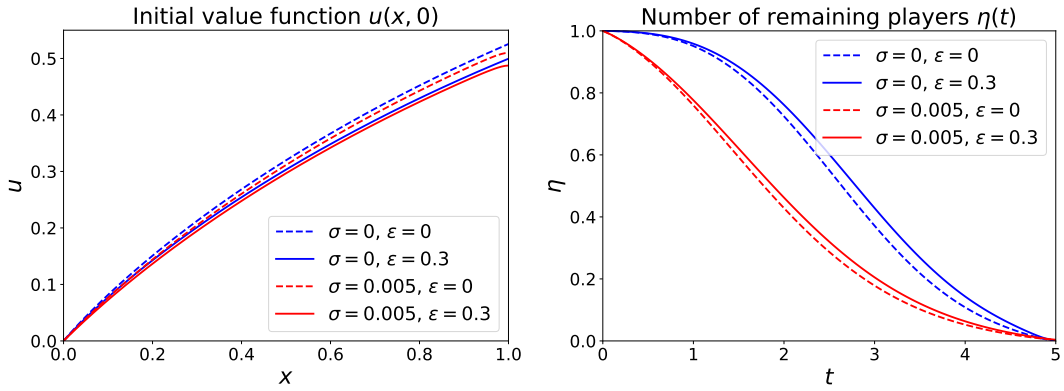


Figure 2.6: Example 2.3: Initial value function $u(x, 0)$ and number of remaining players $\eta(t)$ for different (σ, ϵ).

(i) $(r, s) = (0.2, 1)$

$n_x \times n_t$	Number of iterations				
	Scheme I	Scheme II	Scheme III	Scheme IV	Scheme V
64×64	**6**	7	12	29	17
128×128	**7**	7	12	35	19
256×256	**7**	8	13	40	20
512×512	**7**	8	14	46	20

(ii) $(r, s) = (0.02, 10)$

$n_x \times n_t$	Number of iterations				
	Scheme I	Scheme II	Scheme III	Scheme IV	Scheme V
64×64	**9**	11	∞ (11)	38	19
128×128	**9**	11	∞ (11)	44	21
256×256	**9**	12	∞ (11)	50	21
512×512	**9**	13	∞ (11)	54	22

Table 2.8: Example 2.3: Convergence of the five multigrid schemes. $\sigma = 0.005$, $\epsilon = 0.3$, $T = 5$. **(i)** $(r, s) = (0.2, 1)$. **(ii)** $(r, s) = (0.02, 10)$. Note that, for Scheme III, the numbers in the parentheses are the iteration counts if the numbers of pre and post smoothings are increased from $(1, 1)$ to $(2, 2)$.

or 0.005, $\epsilon = 0$ or 0.3, $T = 5$. Figure 2.6 shows the plots of the initial value function $u(x, 0)$ and the number of remaining players $\eta(t)$. The plots given by our numerical results are the same as those in [42].

Reference [42] does not discuss fast and efficient solvers for (2.38)-(2.41). Here we test

the five multigrid schemes. We restrict the choice of parameters to $\sigma = 0.005$, $\epsilon = 0.3$, $T = 5$. As the convection coefficient D is approximately proportional to s, the size of s indicates the amount of convection. We test the cases of $(r, s) = (0.2, 1)$ and $(r, s) = (0.02, 10)$, which are relatively diffusive and convective, respectively. We also note that since the demand function D is always non-negative, we simply use the low-cost one-direction downstream Gauss-Seidel smoother rather than multi-direction smoother for our proposed Scheme I. The convergence rates are reported in Table 2.8. All the methods have mesh-independent convergence rates. However, our proposed Scheme I has the fastest convergence rates, which is around 7 iterations for the diffusive case and 9 iterations for the convective case. The comparison between Schemes I and II in Table 2.8(ii) shows that subtracting artificial viscosity saves 2-4 iterations, which is consistent with the LFA estimate in Section 2.5.3. We note that, to make Scheme III converge for the convective case, we change the numbers of pre and post smoothings from $(1, 1)$ to $(2, 2)$, as shown in Table 2.8(ii). However, this doubles the computational cost per iteration and yet the convergence rate is still slower than Scheme I.

2.7 Conclusion

We propose a joint spacetime multigrid method for the HJB and KFP system arising from mean field games. We propose a nonlinear FAS scheme, which requires only one layer of iteration (as opposed to outer-inner iterations). We use a hybrid full-semi coarsening and kernel preserving biased restriction operator to treat the anisotropy in time and the convection in space properly. We propose subtracting artificial viscosity to improve the precision of the coarse grid error estimation using direct discretization. These properties are supported by our Fourier analysis. The resulting multigrid method converges at the mesh-independent rate and at a faster rate than the other approaches considered in Section 2.6.

Chapter 3

Finite Difference Method for HJB Formulation of Monge-Ampère Equation

3.1 Introduction

In Chapters 3-5, we will study numerical methods for HJB formulation of image registration problems. As introduced in Section 1.2.3, an image registration problem requires solving a PDE called the Monge-Ampère equation, which is equivalent to an HJB equation. Since image registration problems are non-trivial, in this chapter, we start with a simplified problem, i.e., the following two-dimensional **Monge-Ampère equation** with a Dirichlet boundary condition and a convexity constraint:

$$\det[D^2u(\boldsymbol{x})] = f(\boldsymbol{x}), \quad \text{in } \Omega, \tag{3.1}$$

$$u(\boldsymbol{x}) = g(\boldsymbol{x}), \quad \text{on } \partial\Omega, \tag{3.2}$$

$$u \text{ is convex}, \tag{3.3}$$

where Ω is a bounded convex open set in $\mathbb{R}^2$, $\partial\Omega$ is its boundary, $\overline{\Omega} = \Omega \cup \partial\Omega$, $\boldsymbol{x} = (x, y) \in \overline{\Omega}$ is the spatial coordinates, $u : \overline{\Omega} \to \mathbb{R}$ is the unknown function, D^2u is the Hessian of u, $f : \Omega \to \mathbb{R}$ is a given non-negative right hand side, and $g : \partial\Omega \to \mathbb{R}$ is a given Dirichlet boundary condition. We remark that the difference between (3.1)-(3.3) and the image registration problem is that, while the right hand side of (3.1) does not depend on the solution u, the right hand side of the image registration Monge-Ampère equation depends on u, which will be shown in Chapter 5.

The Monge-Ampère equation is a nonlinear PDE, since the left hand side, $\det(D^2u) = u_{xx}u_{yy} - u_{xy}^2$, consists of products of the second derivatives. As a result, it may have multiple weak solutions. Among all these weak solutions, we are interested in computing the viscosity solution [53, 52], since it is often considered the correct one [72] in practical applications, including the application of image registration that will be studied in Chapter 5. In particular, the viscosity solution of the Monge-Ampère equation is globally convex and satisfies the convexity constraint (3.3), while the other solutions may not be convex [72].

The sufficient conditions for a numerical scheme to converge to the viscosity solution are consistency, stability, monotonicity and the strong comparison principle [14]. Due to the nonlinearity of the Monge-Ampère equation with the additional convexity constraint, it is challenging to design a numerical scheme that satisfies these conditions.

Numerical schemes have been proposed for the Monge-Ampère equation. We first review finite difference methods. For instance, [21] uses the standard central differencing to discretize u_{xy}, and is thus not monotone. Few existing finite difference schemes are monotone and thus convergent in the viscosity sense. For instance, [127] achieves monotonicity by exploiting the geometrical interpretation of the Monge-Ampère equation, but their grid structure is more complicated than rectangular or triangular. Another scheme, proposed in [126, 72] and further improved in [73, 74], uses wide stencils to achieve monotonicity. However, in order for the scheme to converge, the number of the stencil points must increase towards infinity when the mesh size h decreases towards 0, thus resulting in high computational costs for solving problems on fine grids. Recently, [20] improves on the previous wide stencil approach so that the number of stencil points does not grow to infinity as $h \to 0$, but it still grows and can reach as high as 48. There also exist Galerkin-type methods, including [24, 29, 108, 54, 63], etc. A common issue for these Galerkin-type methods is that convergence to the viscosity solutions remains unclear.

Our approach, which is distinct from many of the existing methods, is to first convert (3.1)-(3.3) into an **equivalent HJB equation** [105, 116], and then numerically solve the equivalent HJB equation. Applying the HJB formulation to the numerical computation of the Monge-Ampère equation was first investigated by my coauthors in [115], and further investigated in [62]. There are some important benefits of using the HJB formulation. One is that the differential operator of the HJB equation under fixed control parameters is linear. Another benefit is that the convexity constraint (3.3) is automatically satisfied by the solution of the HJB equation. In other words, there is no need to impose the convexity constraint in the HJB formulation [62]. In addition, many convergent numerical schemes for HJB equations or HJB differential operators have been developed, such as [66, 95, 55, 118, 25, 12, 155]. As a result, it is more tractable to use the equivalent HJB

formulation to design a numerical scheme that converges in the viscosity sense.

It turns out that the cross derivative u_{xy} is still present in the HJB equation, and the standard central differencing or the standard 7-point stencil discretization for u_{xy} may be non-monotone. In order to achieve monotonicity, reference [62] follows the idea in [55, 118] and applies a **semi-Lagrangian scheme**[1], which is a central differencing with a stencil length $\sqrt{h}$ on the locally rotated coordinates where the cross derivative vanishes. This discretization results in *at most 17 stencil points for any h*. However, monotonicity is achieved at the expense of a large truncation error and slow convergence. In particular, the convergence rate is no better than $O(h)$. In addition, reference [62] applies the semi-Lagrangian scheme to the entire computational domain, which may not be necessary.

In order to improve the accuracy and also *strictly* maintain monotonicity, our approach is to apply a **mixed standard 7-point stencil and a semi-Lagrangian wide stencil discretization** to the equivalent HJB equation. More specifically, the standard 7-point stencil discretization, which is second order accurate, is applied to discretize u_{xy} at a grid point if monotonicity is fulfilled. Otherwise, the semi-Lagrangian wide stencil scheme, which is less accurate but guaranteed to be monotone, is implemented. We emphasize that our discretization scheme is designed such that consistency, stability, monotonicity and the strong comparison principle are fulfilled on the entire computational domain. As a result, our numerical scheme is guaranteed to converge to the viscosity solution of the Monge-Ampère equation [14]. Meanwhile, by maximal use of the standard 7-point stencil discretization, the discretization error of the numerical solution is significantly reduced, compared to using the pure semi-Lagrangian wide stencil scheme in [62]. Moreover, our numerical scheme yields a convergence rate of $O(h^2)$ whenever the standard 7-point stencil discretization can be applied monotonically on the entire computation domain, and up to $O(h)$ otherwise. The second order convergence rate in the optimal cases is another significant improvement over the numerical scheme in [62].

To solve the resulting nonlinear discretized system, we use policy iteration. One of the most expensive steps is to optimize two control parameters at every grid point. Reference [62] does not discuss the computational cost of the optimization problem. Typically a bilinear search is implemented on an $m \times m$ discretized control set, resulting in $O(m^2)$ computational complexity. We propose an approach that reduces the computational cost for the optimization problem to $O(1)$ whenever the standard 7-point stencil discretization is applied, and at most $O(m)$ otherwise.

Finally, we want to emphasize that our method is the only method that fulfills all the

[1]In the literature, such a semi-Lagrangian scheme is sometimes called a wide stencil scheme, but this should not be confused with the previously mentioned wide stencil scheme in [126, 72, 73, 74].

following properties: monotone and thus convergent to the viscosity solution, second order accurate in the optimal cases, and having at most 17 stencil points independent of the mesh size h. None of the references in this chapter have all these properties.

To illustrate our numerical scheme, we review the notion of viscosity solution in Section 3.2. In Section 3.3, we establish the equivalent HJB formulation for the Monge-Ampère equation. Section 3.4 describes our mixed standard 7-point stencil and semi-Lagrangian wide stencil finite difference discretization. Section 3.5 discusses solving the nonlinear discretized system using policy iteration, and speeding up the computation for the optimization problem. Section 3.6 proves that our numerical scheme satisfies consistency, stability, monotonicity and the strong comparison principle, and thus converges to the viscosity solution. Section 3.7 shows numerical results. Section 3.8 is the conclusion.

3.2 Viscosity Solution

The objective of this chapter is to compute the viscosity solution of the Monge-Ampère equation (3.1)-(3.3). An overview on the topic of the viscosity solution can be found in [53, 52].

Before defining the viscosity solution, we rewrite (3.1)-(3.2) as

$$\mathcal{N}\left(\boldsymbol{x}, u(\boldsymbol{x}), D^2u(\boldsymbol{x})\right) = 0, \tag{3.4}$$

where $\mathcal{N}$ represents the Monge-Ampère differential operator, defined as

$$\mathcal{N}\left(\boldsymbol{x}, u(\boldsymbol{x}), D^2u(\boldsymbol{x})\right) \equiv \begin{cases} -\det\left[D^2u(\boldsymbol{x})\right] + f(\boldsymbol{x}), & \boldsymbol{x} \in \Omega, \\ u(\boldsymbol{x}) - g(\boldsymbol{x}), & \boldsymbol{x} \in \partial\Omega. \end{cases} \tag{3.5}$$

We also rewrite (3.3) as

$$u \text{ is convex} \quad \Rightarrow \quad D^2u(\boldsymbol{x}) \text{ is positive semi-definite.} \tag{3.6}$$

We also define the upper (respectively lower) semi-continuous envelope of a function $z : C \to \mathbb{R}$ on a closed set C as

$$z^*(x) \equiv \limsup_{y \to x,\, y \in C} z(y) \quad \left(\text{ respectively } z_*(x) \equiv \liminf_{y \to x,\, y \in C} z(y) \right). \tag{3.7}$$

Definition 3.1 (Viscosity solution)**.** A convex upper (respectively lower) semi-continuous function $u : \overline{\Omega} \to \mathbb{R}$ is a viscosity subsolution (respectively supersolution) of the Monge-Ampère equation $\mathcal{N}\left(\boldsymbol{x}, u(\boldsymbol{x}), D^2u(\boldsymbol{x})\right) = 0$, if for all smooth test functions $\varphi(\boldsymbol{x}) \in C^2(\overline{\Omega})$

and all $\boldsymbol{x} \in \overline{\Omega}$, such that $u^* - \varphi$ (respectively $u_* - \varphi$) has a local maximum (respectively minimum) at $\boldsymbol{x}$, we have

$$\mathcal{N}_*(\boldsymbol{x}, u^*(\boldsymbol{x}), D^2\varphi(\boldsymbol{x})) \leq 0 \quad \left(\text{ respectively } \mathcal{N}^*(\boldsymbol{x}, u_*(\boldsymbol{x}), D^2\varphi(\boldsymbol{x})) \geq 0 \right). \tag{3.8}$$

Furthermore, the function u is a viscosity solution if it is both a viscosity sub-solution and super-solution.

We note that Definition 3.1 can be used to define the viscosity solution of not only the Monge-Ampère equation, but also a generic scalar nonlinear PDE (with some adaptations). We also note that (3.6) implies the degenerate ellipticity of the differential operator (3.5). Degenerate ellipticity, plus $\overline{\Omega}$ being bounded and convex, ensures the existence and uniqueness of the viscosity solution of (3.4). We refer readers to [52, 84] for more details.

3.3 HJB Formulation

Our approach to solving the Monge-Ampère equation is to convert it into an equivalent HJB equation. The equivalence of the two PDEs was first established in [105] and [116] for classical solutions. Recently, reference [62] extended the equivalence to viscosity solutions. Here we state the equivalence of the two PDEs as the following theorem:

Theorem 3.1 (HJB formulation)**.** Let Ω be a convex open set in $\mathbb{R}^2$. Let $f \in C(\Omega)$ be a non-negative function. Let a convex function u be the viscosity solution of the following HJB equation,

$$\begin{aligned} &- \operatorname{tr}\left[\boldsymbol{\Sigma}^*(\boldsymbol{x}) D^2 u(\boldsymbol{x})\right] + 2\sqrt{\det(\boldsymbol{\Sigma}^*(\boldsymbol{x}))\, f(\boldsymbol{x})} = 0, \\ &\text{subject to } \boldsymbol{\Sigma}^*(\boldsymbol{x}) \equiv \underset{\boldsymbol{\Sigma}(\boldsymbol{x}) \in S_1^+}{\arg\max} \left\{ - \operatorname{tr}\left[\boldsymbol{\Sigma}(\boldsymbol{x}) D^2 u(\boldsymbol{x})\right] + 2\sqrt{\det(\boldsymbol{\Sigma}(\boldsymbol{x}))\, f(\boldsymbol{x})} \right\}, \end{aligned} \tag{3.9}$$

where $S_1^+ \equiv \{\boldsymbol{\Sigma}(\boldsymbol{x}) \in \mathbb{R}^{2\times 2} : \boldsymbol{\Sigma}(\boldsymbol{x}) \text{ is symmetric positive semi-definite, } \operatorname{tr}(\boldsymbol{\Sigma}(\boldsymbol{x})) = 1\}$. Then u is the viscosity solution of the Monge-Ampère equation (3.4)-(3.6).

Proof. We refer interested readers to the proof in [143] when u is a classical solution, and the proof in [62] for the extension to the viscosity solution. □

We notice that due to the symmetric positive semi-definite property of the matrix $\boldsymbol{\Sigma}(\boldsymbol{x})$, it can be diagonalized by a 2×2 orthogonal matrix as follows:

$$\begin{aligned} \boldsymbol{\Sigma}(\boldsymbol{x}) = \begin{pmatrix} \cos\theta(\boldsymbol{x}) & \sin\theta(\boldsymbol{x}) \\ -\sin\theta(\boldsymbol{x}) & \cos\theta(\boldsymbol{x}) \end{pmatrix} \begin{pmatrix} c(\boldsymbol{x}) & 0 \\ 0 & 1 - c(\boldsymbol{x}) \end{pmatrix} \begin{pmatrix} \cos\theta(\boldsymbol{x}) & -\sin\theta(\boldsymbol{x}) \\ \sin\theta(\boldsymbol{x}) & \cos\theta(\boldsymbol{x}) \end{pmatrix}, \\ c(\boldsymbol{x}) \in [0, 1],\ \theta(\boldsymbol{x}) \in [-\pi, \pi). \end{aligned} \tag{3.10}$$

This parameterization yields an HJB equation with more concrete coefficients and control variables, as follows:

Corollary 3.1 (HJB formulation)**.** Under the parameterization (3.10), the HJB equation (3.9) becomes

$$\mathcal{L}_{c^*(\boldsymbol{x}),\theta^*(\boldsymbol{x})}\, u(\boldsymbol{x}) = 0, \tag{3.11}$$

$$\text{subject to } (c^*(\boldsymbol{x}),\theta^*(\boldsymbol{x})) \equiv \underset{(c(\boldsymbol{x}),\theta(\boldsymbol{x}))\in\Gamma}{\arg\max}\ \mathcal{L}_{c(\boldsymbol{x}),\theta(\boldsymbol{x})}\, u(\boldsymbol{x}), \tag{3.12}$$

where $(c(\boldsymbol{x}),\theta(\boldsymbol{x}))$ is the pair of control at point $\boldsymbol{x}$, $\Gamma = [0,1]\times\left[-\frac{\pi}{4},\frac{\pi}{4}\right)$[2] is the set of admissible control, and

$$\begin{aligned}\mathcal{L}_{c(\boldsymbol{x}),\theta(\boldsymbol{x})}\, u(\boldsymbol{x}) \equiv\ & -\sigma_{11}(c(\boldsymbol{x}),\theta(\boldsymbol{x}))u_{xx}(\boldsymbol{x}) - 2\sigma_{12}(c(\boldsymbol{x}),\theta(\boldsymbol{x}))u_{xy}(\boldsymbol{x}) \\ & -\sigma_{22}(c(\boldsymbol{x}),\theta(\boldsymbol{x}))u_{yy}(\boldsymbol{x}) + 2\sqrt{c(\boldsymbol{x})(1-c(\boldsymbol{x}))f(\boldsymbol{x})}\end{aligned} \tag{3.13}$$

is a linear differential operator with the coefficients

$$\begin{aligned}\sigma_{11}(c(\boldsymbol{x}),\theta(\boldsymbol{x})) &= \frac{1}{2}[1-(1-2c(\boldsymbol{x}))\cos 2\theta(\boldsymbol{x})], \\ \sigma_{22}(c(\boldsymbol{x}),\theta(\boldsymbol{x})) &= \frac{1}{2}[1+(1-2c(\boldsymbol{x}))\cos 2\theta(\boldsymbol{x})], \\ \sigma_{12}(c(\boldsymbol{x}),\theta(\boldsymbol{x})) &= \frac{1}{2}(1-2c(\boldsymbol{x}))\sin 2\theta(\boldsymbol{x}).\end{aligned} \tag{3.14}$$

As discussed in Section 3.1, the HJB formulation introduces some favorable properties over the Monge-Ampère equation. We first note that the convexity constraint of the Monge-Ampère equation poses a major difficulty in designing a convergent numerical scheme. However, the convexity constraint is already implicitly enforced in the equivalent HJB formulation (3.11)-(3.12), and thus can be removed from the HJB formulation [72, 62], which makes the numerical computation more manageable. Another useful property of the HJB formulation is that, for a fixed given control (c,θ), the differential operator (3.13) is linear, which allows us to develop finite difference schemes based on numerical methods for linear PDEs. In addition, many papers have been devoted to convergent numerical schemes for HJB equations [66, 95, 55, 118, 25, 12, 155]. Considering these advantages, our approach is to solve the HJB equation (3.11)-(3.12) instead of the Monge-Ampère equation (3.4)-(3.6).

[2] Although (3.10) defines the admissible control set to be in the range of $[0,1]\times[-\pi,\pi)$, the optimal control (c^*,θ^*) that maximizes (3.12) may not be unique in $[0,1]\times[-\pi,\pi)$. Since $\mathcal{L}_{c,\theta}\, u = \mathcal{L}_{c,\theta+\pi}\, u$, and $\mathcal{L}_{c,\theta}\, u = \mathcal{L}_{1-c,\theta+\frac{\pi}{2}}\, u$, the admissible control set can be reduced to $[0,1]\times[-\frac{\pi}{4},\frac{\pi}{4})$. Such removal of the redundancy ensures the uniqueness of the optimal control (c^*,θ^*) in Γ, except when $c^* = \frac{1}{2}$ or when $f = 0$.

3.4 Finite Difference Discretization

In this section, we will propose a finite difference discretization for the HJB equation (3.11)-(3.12). In particular, our goal is to make sure that the discretization is monotone, since [14] proves that monotonicity, together with consistency and stability, are sufficient conditions for a numerical scheme to converge to the viscosity solution. The finite difference notation follows Section 2.2.1. The only exception is that there is no time coordinate. Also, under the Dirichlet boundary condition, $\boldsymbol{x}_{i,j} \in \Omega$ when $1 \le i \le n_x$, $1 \le j \le n_y$, and $\boldsymbol{x}_{i,j} \in \partial\Omega$ when $i = 0$ or $i = n_x + 1$ or $j = 0$ or $j = n_y + 1$.

3.4.1 Standard 7-point stencil discretization

Consider discretizing the differential operator (3.13) at a grid point $\boldsymbol{x}_{i,j}$. We use the standard central differencing (2.6) to approximate $u_{xx}(\boldsymbol{x}_{i,j})$ and $u_{yy}(\boldsymbol{x}_{i,j})$. Regarding the cross derivative $u_{xy}(\boldsymbol{x}_{i,j})$, it can be shown that the standard 7-point stencil discretization leads to a monotone scheme in the following two cases:

- **Case 1.** When the coefficients (3.14) at a grid point $\boldsymbol{x}_{i,j}$ satisfy

  $$\sigma_{11}(c_{i,j},\theta_{i,j}) \ge |\sigma_{12}(c_{i,j},\theta_{i,j})|,\ \sigma_{22}(c_{i,j},\theta_{i,j}) \ge |\sigma_{12}(c_{i,j},\theta_{i,j})|,\ \sigma_{12}(c_{i,j},\theta_{i,j}) \ge 0, \quad (3.15)$$

  we approximate $u_{xy}(\boldsymbol{x}_{i,j})$ using

  $$\frac{1}{2}(D_x^+D_y^+ + D_x^-D_y^-)u_{i,j} \equiv \frac{2u_{i,j} + u_{i+1,j+1} + u_{i-1,j-1} - u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1}}{2h^2}. \quad (3.16)$$

- **Case 2.** When the coefficients (3.14) at a grid point $\boldsymbol{x}_{i,j}$ satisfy

  $$\sigma_{11}(c_{i,j},\theta_{i,j}) \ge |\sigma_{12}(c_{i,j},\theta_{i,j})|,\ \sigma_{22}(c_{i,j},\theta_{i,j}) \ge |\sigma_{12}(c_{i,j},\theta_{i,j})|,\ \sigma_{12}(c_{i,j},\theta_{i,j}) \le 0, \quad (3.17)$$

  we approximate $u_{xy}(\boldsymbol{x}_{i,j})$ using

  $$\frac{1}{2}(D_x^+D_y^- + D_x^-D_y^+)u_{i,j} \equiv \frac{-2u_{i,j} - u_{i+1,j-1} - u_{i-1,j+1} + u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1}}{2h^2}. \quad (3.18)$$

Figure 3.1 shows the stencil points of the 7-point stencil discretizations (3.16) and (3.18).

As a result, the discretization of the differential operator (3.13) at $\boldsymbol{x}_{i,j}$ reads

$$\begin{aligned}\mathcal{L}_{i,j}(c_{i,j},\theta_{i,j};u_h) \equiv\ & -\sigma_{11}(c_{i,j},\theta_{i,j})D_x^+D_x^-u_{i,j} - \sigma_{12}(c_{i,j},\theta_{i,j})(D_x^+D_y^\pm + D_x^-D_y^\mp)u_{i,j} \\ & -\sigma_{22}(c_{i,j},\theta_{i,j})D_y^+D_y^-u_{i,j} + 2\sqrt{c_{i,j}(1-c_{i,j})f_{i,j}}.\end{aligned} \quad (3.19)$$

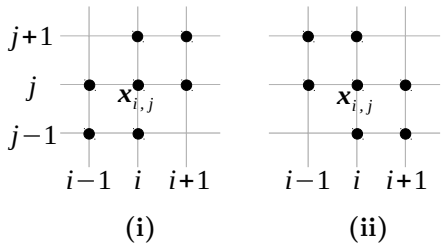


Figure 3.1: **(i)** 7-point stencil of (3.16); **(ii)** 7-point stencil of (3.18).

3.4.2 Semi-Lagrangian wide stencil discretization

However, if neither of Conditions (3.15) and (3.17) is fulfilled at the grid point $\boldsymbol{x}_{i,j}$, then it is unclear how to directly discretize the cross derivative $u_{xy}(\boldsymbol{x}_{i,j})$ in (3.13) monotonically. Our approach is to consider a **semi-Lagrangian wide stencil discretization** [55, 118]. Figure 3.2 illustrates the discretization process. More specifically, we consider eliminating the cross derivative $u_{xy}(\boldsymbol{x}_{i,j})$ by a local coordinate transformation. Let $\{(\boldsymbol{e}_z)_{i,j}, (\boldsymbol{e}_w)_{i,j}\}$ be a local orthogonal basis obtained by a clockwise rotation of the standard axes $\{(\boldsymbol{e}_x)_{i,j}, (\boldsymbol{e}_y)_{i,j}\}$, as represented by the grey axes in Figure 3.2. By straightforward algebra, one can show that if the rotation angle is

$$\frac{1}{2}\arctan\frac{2\sigma_{12}\left(c_{i,j},\theta_{i,j}\right)}{\sigma_{22}\left(c_{i,j},\theta_{i,j}\right)-\sigma_{11}\left(c_{i,j},\theta_{i,j}\right)}=\theta_{i,j},$$

then the cross derivative vanishes under the basis $\{(\boldsymbol{e}_z)_{i,j}, (\boldsymbol{e}_w)_{i,j}\}$. As a result, (3.13) becomes

$$-c_{i,j}\,u_{zz}(\boldsymbol{x}_{i,j})-(1-c_{i,j})\;u_{ww}(\boldsymbol{x}_{i,j})+2\sqrt{c_{i,j}\left(1-c_{i,j}\right)f_{i,j}} \tag{3.20}$$

Here $u_{zz}(\boldsymbol{x}_{i,j})$ and $u_{ww}(\boldsymbol{x}_{i,j})$ are the directional derivatives along the basis $(\boldsymbol{e}_z)_{i,j}$ and $(\boldsymbol{e}_w)_{i,j}$. We note that (3.20) still depends on $\theta_{i,j}$, as the basis $(\boldsymbol{e}_z)_{i,j}$ and $(\boldsymbol{e}_w)_{i,j}$ depend on $\theta_{i,j}$.

To discretize (3.20), one may consider applying the standard central differencing to $u_{zz}(\boldsymbol{x}_{i,j})$ and $u_{ww}(\boldsymbol{x}_{i,j})$. For instance, we approximate $u_{zz}(\boldsymbol{x}_{i,j})$ by

$$\frac{1}{h^2}\left[u(\boldsymbol{x}_{i,j}+h(\boldsymbol{e}_z)_{i,j})-2u_{i,j}+u(\boldsymbol{x}_{i,j}-h(\boldsymbol{e}_z)_{i,j})\right]. \tag{3.21}$$

However, since the stencil is rotated, the stencil points $\boldsymbol{x}_{i,j}\pm h(\boldsymbol{e}_z)_{i,j}$ may no longer coincide with any grid points. In such cases, we consider approximating $u(\boldsymbol{x}_{i,j}\pm h(\boldsymbol{e}_z)_{i,j})$ using bilinear interpolation from the neighboring grid points. However, a consequence of the

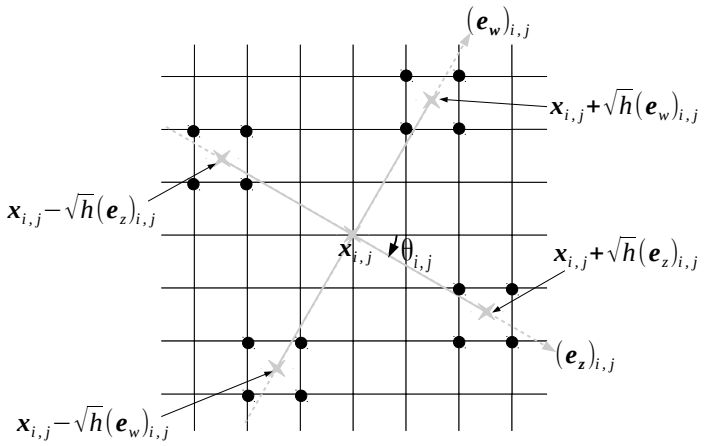


Figure 3.2: Semi-Lagrangian wide stencil discretization at a grid point $\boldsymbol{x}_{i,j}$ inside the computational domain.

bilinear interpolation is that the truncation error of (3.21) turns out to be $O(1)$, which is not consistent. In order to maintain consistency, we choose the stencil length $\sqrt{h}$, which yields $O(h)$ truncation error. We note that the stencil length $\sqrt{h}$ is greater than h, which gives rise to a "wide" stencil.

Under the stencil length $\sqrt{h}$, the new stencil points are $\boldsymbol{x}_{i,j} \pm \sqrt{h}(\boldsymbol{e}_z)_{i,j}$ and $\boldsymbol{x}_{i,j} \pm \sqrt{h}(\boldsymbol{e}_w)_{i,j}$, as represented by the grey stars in Figure 3.2. The unknown values at these stencil points are approximated by the bilinear interpolation from their neighboring points, as represented by the black dots in Figure 3.2. We denote these interpolated unknown values as $\mathcal{I}_h u|_{\boldsymbol{x}_{i,j}\pm\sqrt{h}(\boldsymbol{e}_z)_{i,j}}$ and $\mathcal{I}_h u|_{\boldsymbol{x}_{i,j}\pm\sqrt{h}(\boldsymbol{e}_w)_{i,j}}$. As a result, the finite difference discretizations for $u_{zz}(\boldsymbol{x}_{i,j})$ and $u_{ww}(\boldsymbol{x}_{i,j})$ are given by

$$D_z^+ D_z^- u_{i,j} \equiv \frac{\mathcal{I}_h u|_{\boldsymbol{x}_{i,j}+\sqrt{h}(\boldsymbol{e}_z)_{i,j}} - 2u_{i,j} + \mathcal{I}_h u|_{\boldsymbol{x}_{i,j}-\sqrt{h}(\boldsymbol{e}_z)_{i,j}}}{h}, \tag{3.22}$$

$$D_w^+ D_w^- u_{i,j} \equiv \frac{\mathcal{I}_h u|_{\boldsymbol{x}_{i,j}+\sqrt{h}(\boldsymbol{e}_w)_{i,j}} - 2u_{i,j} + \mathcal{I}_h u|_{\boldsymbol{x}_{i,j}-\sqrt{h}(\boldsymbol{e}_w)_{i,j}}}{h}. \tag{3.23}$$

As a result, the discretization of the differential operator (3.13) at $\boldsymbol{x}_{i,j}$ reads

$$\mathcal{L}_{i,j}(c_{i,j}, \theta_{i,j}; u_h) \equiv -c_{i,j}\, D_z^+ D_z^- u_{i,j} - (1 - c_{i,j})\; D_w^+ D_w^- u_{i,j} + 2\sqrt{c_{i,j}(1 - c_{i,j}) f_{i,j}}. \tag{3.24}$$

We remark that here we have only discussed the scenario where $\boldsymbol{x}_{i,j}$ is well inside the computational domain. In Appendix A.4, we also discuss the scenario where $\boldsymbol{x}_{i,j}$ is near

the boundary. In addition, since each bilinear interpolation introduces 4 stencil points (or 4 unknowns), there are at most 17 stencil points (or 17 unknowns) in (3.24). For interested readers, we also show this result explicitly in Appendix A.4.

3.4.3 Mixed discretization

The advantage of the semi-Lagrangian wide stencil discretization (3.24) is that it is unconditionally monotone. Reference [62] applies the semi-Lagrangian wide stencil discretization at every grid point. However, we will prove in Section 3.6 that this is only first order accurate. Conversely, the standard 7-point stencil discretization is second order accurate. In order to combine the advantages of both discretization schemes, we will only apply the semi-Lagrangian wide stencil discretization at the grid points where neither (3.15) nor (3.17) is satisfied. Otherwise, the standard 7-point stencil discretization is applied. Hence, we propose discretizing the HJB equation at each grid point $\boldsymbol{x}_{i,j}$ by the following **mixed discretization**:

Algorithm 3.1 Mixed discretization for the HJB equation (3.11)-(3.12)

for $i = 1, , ..., n_x$ **do**
  for $j = 1, , ..., n_y$ **do**
    if $(c_{i,j}, \theta_{i,j})$ satisfies Conditions (3.15) or (3.17) **then**
      The discrete equation at (i, j), $\mathcal{L}_{i,j}(c_{i,j}, \theta_{i,j}; u_h)$, is given by the **standard 7-point stencil discretization** (3.19)
    else
      The discrete equation at (i, j), $\mathcal{L}_{i,j}(c_{i,j}, \theta_{i,j}; u_h)$, is given by the **semi-Lagrangian wide stencil discretization** (3.24)
    end if
  end for
end for

The significance of this mixed discretization is that monotonicity is strictly maintained at every grid point, and meanwhile, by using the standard 7-point stencil discretization as much as possible, the numerical scheme is as accurate as possible.

The mixed discretization scheme gives rise to a nonlinear discrete system that contains $n_x n_y$ discrete equations. Similar to Section 2.2.2, we can write the entire nonlinear discrete

system in the following matrix form:

$$A_h(c_h^*, \theta_h^*)\, u_h = b_h(c_h^*, \theta_h^*), \tag{3.25}$$

$$\text{subject to} \quad (c_h^*, \theta_h^*) \equiv \arg\max_{(c_h,\theta_h)\in\Gamma} \left\{ A_h(c_h, \theta_h)\, u_h - b_h(c_h, \theta_h) \right\}, \tag{3.26}$$

where the matrix $A_h \in \mathbb{R}^{n_x n_y \times n_x n_y}$ and the vectors $u_h, c_h, \theta_h, b_h \in \mathbb{R}^{n_x n_y}$. We note that A_h (and b_h respectively) are obtained by splitting (3.19) and (3.24) into the sections that contain (and do not contain, respectively) the unknowns u, i.e.,

$$\mathcal{L}_{i,j}(c_{i,j}, \theta_{i,j}; u_h) \to \left[A_h(c_h, \theta_h)\, u_h - b_h(c_h, \theta_h) \right]_{\langle i,j \rangle}, \quad 1 \le i \le n_x,\ 1 \le j \le n_y. \tag{3.27}$$

where

$$\langle i, j \rangle \equiv n_y \cdot (i-1) + j \tag{3.28}$$

maps a grid point (i, j) to its corresponding lexicographical matrix/vector index.

3.5 Solving the Discretized System

After setting up the complete nonlinear discrete system (3.25)-(3.26), the next objective is to solve it. We apply policy iteration. For concreteness, we write down the policy iteration for (3.25)-(3.26) in Algorithm 3.2.

Algorithm 3.2 consists of two sub-steps. One sub-step is to solve the linear system under a given control; see Line 3. In this chapter, we use Krylov subspace methods, such as the GMRES with the incomplete LU preconditioner. We will leave the discussion of speeding up this sub-step using multigrid methods to Chapter 4. The other sub-step is to solve the optimization problem at each grid point $\boldsymbol{x}_{i,j}$; see Line 6. Here we focus our discussion on speeding up the computation of the optimization problem.

Consider solving the optimization problem at a grid point $\boldsymbol{x}_{i,j}$ (Line 6) where the semi-Lagrangian wide stencil discretization (3.24) is applied. As the discretization of $D_z^+ D_z^- u_{i,j}$ and $D_w^+ D_w^- u_{i,j}$ depends on the control $\theta_{i,j}$, there is no simple closed-form formula to evaluate the optimal $(c_{i,j}^{(k)}, \theta_{i,j}^{(k)})$ directly. In this case, one typical approach is to use a bilinear search algorithm for the optimization problem. More specifically, we discretize the continuous admissible control set $\Gamma = [0, 1] \times [-\frac{\pi}{4}, \frac{\pi}{4})$ into an $m \times m$ discrete set, denoted as Γ^h. The discretization of the admissible control set introduces additional truncation error. In order to maintain consistency, we must let $m \to \infty$ as $h \to 0$. A typical choice of m is $m = \sqrt{n_x n_y}$ (or simply $m = n_x$ if we assume $n_x = n_y$). Then we compute the $m \times m$

Algorithm 3.2 Policy iteration for solving the discretized HJB equation (3.25)-(3.26)

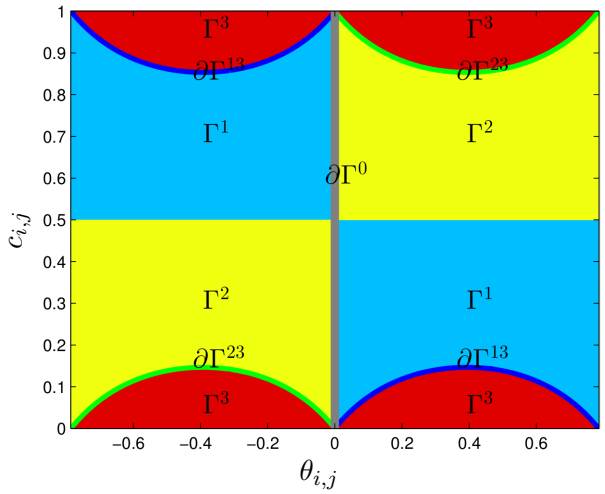
Start with an initial guess of the control variable $(c_h^{(0)}, \theta_h^{(0)})$.
for $k = 1, 2, ...$ until convergence **do**
  Solve the linearized HJB equation $A_h(c_h^{(k-1)}, \theta_h^{(k-1)})\, u_h^{(k)} = b_h(c_h^{(k-1)}, \theta_h^{(k-1)})$ for the solution $u_h^{(k)}$.
  for $i = 1, , ..., n_x$ **do**
    for $j = 1, , ..., n_y$ **do**
      Solve the optimization problem $(c_{i,j}^{(k)}, \theta_{i,j}^{(k)}) \equiv \underset{(c_{i,j},\theta_{i,j})\in\Gamma}{\arg\max}\, \mathcal{L}_{i,j}(c_{i,j}, \theta_{i,j}; u_h^{(k)})$ for the control $(c_{i,j}^{(k)}, \theta_{i,j}^{(k)})$.
    end for
  end for
end for
Convergent solution: $u_h = u_h^{(k)}$, $c_h^* = c_h^{(k)}$, $\theta_h^* = \theta_h^{(k)}$.

values of the objective function $\{\mathcal{L}_{i,j}(c_{i,j}, \theta_{i,j}; u_h^{(k)}) \,|\, (c_{i,j}, \theta_{i,j}) \in \Gamma^h\}$ and then find the global maximal value, which gives the optimal $(c_{i,j}^{(k)}, \theta_{i,j}^{(k)})$. However, the computational cost of the bilinear search is $O(m^2)$. Furthermore, if we consider applying such bilinear search on all the grid points $\{\boldsymbol{x}_{i,j} \,|\, 1 \le i \le n_x, 1 \le j \le n_y\}$, then the total computational cost on the entire computational domain is as high as $O(m^2 n_x n_y)$, or $O((n_x n_y)^2)$.

In order to speed up the computation for the optimal control, we divide the continuous admissible control set $\Gamma = [0, 1] \times [-\frac{\pi}{4}, \frac{\pi}{4})$ into six regions, as shown in Figure 3.3[3]. The six regions are identified by whether the control $(c_{i,j}, \theta_{i,j})$ satisfies (3.15), or (3.17), or neither. Our approach is to first find the optimal control within each of the six regions. Regarding how to solve the optimization problem within each region, Appendix A.5 describes the technical details, and Figure 3.3 provides a table summarizing the approach. Once we obtain the six regional optimal controls, we search within them for the global optimal control. This approach enables us to make full use of the analytical property of each region, and to improve the optimization algorithm within each region and eventually on the entire admissible control set Γ.

Using our approach, the computational cost of solving the optimization problem on Γ can be significantly reduced. More specifically, if the standard 7-point stencil discretization

[3]It is unnecessary to consider the line $c_{i,j} = \frac{1}{2}$, since the objective function is a constant on this line. Also it is unnecessary to consider the line $\theta_{i,j} = \pm\frac{\pi}{4}$, since $\mathcal{L}_{c,\theta}\, u = \mathcal{L}_{1-c,\theta+\frac{\pi}{2}}\, u$ indicates that $\theta_{i,j} = \pm\frac{\pi}{4}$ is indeed an interior part of Γ^1 and Γ^2.



Region	Definition	Discretization	Optimization algorithm in each region	Cost	Extra truncation error introduced?
Γ^1	The region where (3.15) is satisfied	Standard 7-point stencil with (3.16)	Closed-form formula from the first derivative test	$O(1)$	No
Γ^2	The region where (3.17) is satisfied	Standard 7-point stencil with (3.18)			
Γ^3	The region where neither (3.15) nor (3.17) is satisfied	Semi-Lagrangian wide stencil	Linear search over a single control $\theta_{i,j} \in [-\frac{\pi}{4}, \frac{\pi}{4})$	$O(m)$	Yes
$\partial\Gamma^0$	The line $\theta_{i,j} = 0$	Standard 7-point stencil with (3.16) or (3.18)	Closed-form formula from the first derivative test	$O(1)$	No
$\partial\Gamma^{13}$	The boundary between Γ^1 and Γ^3	Standard 7-point stencil with (3.16)			
$\partial\Gamma^{23}$	The boundary between Γ^2 and Γ^3	Standard 7-point stencil with (3.18)			

Figure 3.3: Division of the admissible control set $\Gamma = [0, 1] \times [-\frac{\pi}{4}, \frac{\pi}{4})$ into regions.

can be applied monotonically on all or most of the grid points, then the computational cost is $O(1)$ per grid point and $O(n_x n_y)$ on the entire computational domain. In general, the computational cost is at most $O(m)$ per grid point and at most $O(m n_x n_y)$ on the entire computational domain. For the typical choice $m = \sqrt{n_x n_y}$, the total computational cost

of solving the optimization problem is $O((n_x n_y)^{3/2})$.

As a side remark, in Section 8 of [62], the authors discretize θ with $m = 64$ different angles, regardless of the mesh size h. Indeed, if m is constant, then the numerical scheme in [62] is no longer consistent in theory. This is different from our scheme, where θ is discretized with $m = \sqrt{n_x n_y}$ angles, such that consistency is still maintained.

3.6 Convergence Analysis

As proved by Barles and Souganidis [14], consistency, stability, monotonicity and the strong comparison principle are the four sufficient conditions for the numerical scheme of a nonlinear PDE to converge in the viscosity sense. In this section, we will prove that our numerical scheme fulfills all the four requirements and is therefore guaranteed to converge to the viscosity solution of (3.4)-(3.6).

Before moving on to the analysis, we introduce some notation. First, we consider rewriting the HJB equation (3.11)-(3.12) as $\mathcal{N}\left(\boldsymbol{x}, u(\boldsymbol{x}), D^2 u(\boldsymbol{x})\right) = 0$, where the complete nonlinear differential operator is

$$\begin{aligned} \mathcal{N}\left(\boldsymbol{x}, u(\boldsymbol{x}), D^2 u(\boldsymbol{x})\right) \equiv \quad & \mathcal{L}_{c^*(\boldsymbol{x}),\theta^*(\boldsymbol{x})}\, u(\boldsymbol{x}), \\ & \text{subject to} \quad (c^*(\boldsymbol{x}), \theta^*(\boldsymbol{x})) \equiv \underset{c(\boldsymbol{x}),\theta(\boldsymbol{x})\in\Gamma}{\arg\max}\ \mathcal{L}_{c(\boldsymbol{x}),\theta(\boldsymbol{x})}\, u(\boldsymbol{x}). \end{aligned} \tag{3.29}$$

For compactness, we can rewrite it as

$$\mathcal{N}\left(\boldsymbol{x}, u(\boldsymbol{x}), D^2 u(\boldsymbol{x})\right) \equiv \max_{c(\boldsymbol{x}),\theta(\boldsymbol{x})\in\Gamma} \mathcal{L}_{c(\boldsymbol{x}),\theta(\boldsymbol{x})}\, u(\boldsymbol{x}). \tag{3.30}$$

We note that here we use the same symbol $\mathcal{N}$ as the one for the Monge-Ampère equation (3.5), since the HJB equation and the Monge-Ampère equation are equivalent.

Similarly, we denote the correspondingly nonlinear discretized system (3.25)-(3.26) as $\mathcal{N}_h(u_h) = 0$, where

$$\mathcal{N}_h(u_h) \equiv \max_{(c_h,\theta_h)\in\Gamma} \mathcal{L}_h(c_h, \theta_h; u_h) \tag{3.31}$$

is the discrete nonlinear operator, while

$$\mathcal{L}_h(c_h, \theta_h; u_h) \equiv A_h(c_h, \theta_h)\, u_h - b_h(c_h, \theta_h), \tag{3.32}$$

is the discrete linear operator. We note that (3.32) uses the notation established in (3.27). As a remark, here we choose the symbols $\mathcal{N}_h$ and $\mathcal{L}_h$, as they are essentially the discretization of $\mathcal{N}$ and $\mathcal{L}$, respectively.

3.6.1 Consistency

In this subsection, we prove that our numerical scheme is consistent in the viscosity sense.

Lemma 3.1 (Consistency)**.** For the Monge-Ampère equation (or equivalently, the HJB equation) $\mathcal{N}(\boldsymbol{x}, u(\boldsymbol{x}), D^2u(\boldsymbol{x})) = 0$, the numerical scheme $\mathcal{N}_h(\boldsymbol{x}_{i,j}, u_h) = 0$ given in (3.25)-(3.26) is consistent in the viscosity sense. More specifically, for any smooth test function $\varphi(\boldsymbol{x}) \in C^\infty(\overline{\Omega})$ with $\varphi_{i,j} \equiv \varphi(\boldsymbol{x}_{i,j})$ and $\varphi_h \equiv (\varphi_{1,1}, \varphi_{1,2}, \cdots, \varphi_{n_x,n_y})^T \in \mathbb{R}^{n_x n_y}$, for any $\hat{\boldsymbol{x}} \in \overline{\Omega}$, and for h and ξ that are arbitrary small constants independent of $\boldsymbol{x}$, we have

$$\limsup_{\substack{h\to 0,\,\xi\to 0\\ \boldsymbol{x}_{i,j}\to\hat{\boldsymbol{x}}}} \mathcal{N}_h(\boldsymbol{x}_{i,j}, \varphi_h + \xi) \le \mathcal{N}^*(\hat{\boldsymbol{x}}, \varphi(\hat{\boldsymbol{x}}), D^2\varphi(\hat{\boldsymbol{x}})), \tag{3.33}$$

$$\liminf_{\substack{h\to 0,\,\xi\to 0\\ \boldsymbol{x}_{i,j}\to\hat{\boldsymbol{x}}}} \mathcal{N}_h(\boldsymbol{x}_{i,j}, \varphi_h + \xi) \ge \mathcal{N}_*(\hat{\boldsymbol{x}}, \varphi(\hat{\boldsymbol{x}}), D^2\varphi(\hat{\boldsymbol{x}})). \tag{3.34}$$

In practice, we prove a sufficient condition called local consistency, as follows:

Lemma 3.2 (Local consistency)**.** Under the assumptions in Lemma 3.1, we have

$$\begin{aligned}
&\mathcal{N}(\boldsymbol{x}_{i,j}, \varphi(\boldsymbol{x}_{i,j}), D^2\varphi(\boldsymbol{x}_{i,j})) - \mathcal{N}_h(\boldsymbol{x}_{i,j}, \varphi_h + \xi)\\
&= \begin{cases}
O(h^2) + O(\xi), & \text{standard 7-point stencil,}\\
O(h) + O(\xi), & \text{semi-Lagrangian wide stencil, with all the 4}\\
& \text{wide stencil points } \in \Omega,\\
O(\sqrt{h}) + O(\xi), & \text{semi-Lagrangian wide stencil, otherwise.}
\end{cases}
\end{aligned} \tag{3.35}$$

Proof. We note that the proof with $\xi = 0$ is equivalent to the proof with a general ξ. Such equivalence can be easily verified if we substitute φ by $\varphi + \xi$ in the following proof. Hence, we will only prove the case where $\xi = 0$.

Case 1. *Truncation error of the standard 7-point stencil discretization.* Suppose the standard 7-point stencil discretization is applied at $\boldsymbol{x}_{i,j}$. Using Taylor expansion, we obtain the following truncation errors:

$$\begin{aligned}
&|\varphi_{xx}(\boldsymbol{x}_{i,j}) - D_x^+ D_x^- \varphi_{i,j}| \le C^{xx} h^2, \quad |\varphi_{yy}(\boldsymbol{x}_{i,j}) - D_y^+ D_y^- \varphi_{i,j}| \le C^{yy} h^2,\\
&|\varphi_{xy}(\boldsymbol{x}_{i,j}) - \tfrac{1}{2}(D_x^+ D_y^\pm + D_x^- D_y^\mp)\varphi_{i,j}| \le C^{xy} h^2,
\end{aligned}$$

where the coefficients C^{xx}, C^{yy} and C^{xy} are uniformly bounded. Here "uniformly bounded" means that the coefficients are bounded for any h and for any $(c_{i,j}, \theta_{i,j}) \in \Gamma$. To see this

explicitly, we check the following:

$$\begin{aligned}
&\varphi_{xy}(\boldsymbol{x}_{i,j}) - \tfrac{1}{2}(D_x^+ D_y^+ + D_x^- D_y^-)\varphi_{i,j} \\
&= \tfrac{1}{24}h^2[\,(\varphi_{xxxx})_{i+r_1,j} - (\varphi_{xxxx})_{i+r_2,j+s_2} + (\varphi_{yyyy})_{i,j+s_3} - (\varphi_{yyyy})_{i+r_4,j+s_4} \\
&\qquad -4(\varphi_{xxxy})_{i+r_5,j+s_5} - 6(\varphi_{xxyy})_{i+r_6,j+s_6} - 4(\varphi_{xyyy})_{i+r_7,j+s_7}\,],
\end{aligned}$$

where $\varphi_{i+r,j+s} = \varphi(x+rh, y+sh)$ with $r, s \in [-1, 1]$. Hence,

$$|\,\varphi_{xy}(\boldsymbol{x}_{i,j}) - \frac{1}{2}(D_x^+ D_y^+ + D_x^- D_y^-)\varphi_{i,j}\,| \le C^{xy}h^2,$$

where the coefficient

$$\begin{aligned}
C^{xy} \equiv \tfrac{1}{12} \quad &\big(\max_{\boldsymbol{x}\in\Omega}|\varphi_{xxxx}(\boldsymbol{x})| + \max_{\boldsymbol{x}\in\Omega}|\varphi_{yyyy}(\boldsymbol{x})| \\
&+2\max_{\boldsymbol{x}\in\Omega}|\varphi_{xxxy}(\boldsymbol{x})| + 3\max_{\boldsymbol{x}\in\Omega}|\varphi_{xxyy}(\boldsymbol{x})| + 2\max_{\boldsymbol{x}\in\Omega}|\varphi_{xyyy}(\boldsymbol{x})| \;\big)
\end{aligned}$$

is bounded independent of h, since the test function $\varphi \in \mathbb{C}^\infty(\Omega)$.

Then, the local truncation error of the discrete linear operator at $\boldsymbol{x}_{i,j}$ is

$$|\,\mathcal{L}_{c(\boldsymbol{x}_{i,j}),\theta(\boldsymbol{x}_{i,j})}\varphi(\boldsymbol{x}_{i,j}) - \mathcal{L}_h(\boldsymbol{x}_{i,j}; c_{i,j}, \theta_{i,j}; \varphi_h)\,| \le C(c_{i,j}, \theta_{i,j})h^2,$$

where the coefficient is

$$C(c_{i,j}, \theta_{i,j}) \equiv |\sigma_{11}(c_{i,j}, \theta_{i,j})|\,C^{xx} + 2|\sigma_{12}(c_{i,j}, \theta_{i,j})|\,C^{xy} + |\sigma_{22}(c_{i,j}, \theta_{i,j})|\,C^{yy}.$$

We note that σ_{11}, σ_{22} and σ_{12} are uniformly bounded, and in particular, the bounds are independent of $(c_{i,j}, \theta_{i,j}) \in \Gamma$. As a result, C is uniformly bounded.

Then, the local truncation error of the discrete nonlinear operator at $\boldsymbol{x}_{i,j}$ is

$$\begin{aligned}
&|\,\mathcal{N}(\boldsymbol{x}_{i,j}, \varphi(\boldsymbol{x}_{i,j}), D^2\varphi(\boldsymbol{x}_{i,j})) - \mathcal{N}_h(\boldsymbol{x}_{i,j}, \varphi_h)\,| \\
&= \Big| \max_{(c(\boldsymbol{x}_{i,j}),\theta(\boldsymbol{x}_{i,j}))\in\Gamma} \mathcal{L}_{c(\boldsymbol{x}_{i,j}),\theta(\boldsymbol{x}_{i,j})}\varphi(\boldsymbol{x}_{i,j}) - \max_{(c_{i,j},\theta_{i,j})\in\Gamma} \mathcal{L}_h(\boldsymbol{x}_{i,j}; c_{i,j}, \theta_{i,j}; \varphi_h) \Big| \\
&\le \max_{(c_{i,j},\theta_{i,j})\in\Gamma} \big|\,\mathcal{L}_{c(\boldsymbol{x}_{i,j}),\theta(\boldsymbol{x}_{i,j})}\varphi(\boldsymbol{x}_{i,j}) - \mathcal{L}_h(\boldsymbol{x}_{i,j}; c_{i,j}, \theta_{i,j}; \varphi_h)\,\big| \\
&= \max_{(c_{i,j},\theta_{i,j})\in\Gamma} C(c_{i,j}, \theta_{i,j})h^2,
\end{aligned} \tag{3.36}$$

where we have applied the inequality $\Big|\max_x f(x) - \max_x g(x)\Big| \le \max_x |f(x) - g(x)|$. Due to the uniform boundedness of $C(c_{i,j}, \theta_{i,j})$ for any $(c_{i,j}, \theta_{i,j}) \in \Gamma$, we conclude that

$$\big|\,\mathcal{N}(\boldsymbol{x}_{i,j}, \varphi(\boldsymbol{x}_{i,j}), D^2\varphi(\boldsymbol{x}_{i,j})) - \mathcal{N}_h(\boldsymbol{x}_{i,j}, \varphi_h)\,\big| = O(h^2). \tag{3.37}$$

Case 2. *Truncation error of the semi-Lagrangian wide stencil discretization.* Suppose the semi-Lagrangian wide stencil discretization is applied at $\boldsymbol{x}_{i,j}$. Before going into details, we note that the truncation error of the linear interpolation at a point $\boldsymbol{x}$ is given by

$$\varphi(\boldsymbol{x}) - \mathcal{I}_h\varphi|_{\boldsymbol{x}} = C_{\boldsymbol{x}}^{interp} h^2,$$

where $C_{\boldsymbol{x}}^{interp}$ is uniformly bounded, as all the derivatives of φ are uniformly bounded.

We focus on the truncation error for $D_z^+ D_z^- \varphi_{i,j}$ only, and analyze three cases. The first case is that both stencil points of $D_z^+ D_z^- \varphi_{i,j}$ are in the computational domain. The expression for $D_z^+ D_z^- \varphi_{i,j}$ is given by (3.22). The truncation error for $D_z^+ D_z^- \varphi_{i,j}$ is then

$$\begin{aligned}
&\varphi_{zz}(\boldsymbol{x}_{i,j}) - D_z^+ D_z^- \varphi_{i,j} \\
&= \varphi_{zz}(\boldsymbol{x}_{i,j}) - \frac{\mathcal{I}_h\varphi|_{\boldsymbol{x}_{i,j}+\sqrt{h}(\boldsymbol{e}_z)_{i,j}} - 2\varphi_{i,j} + \mathcal{I}_h\varphi|_{\boldsymbol{x}_{i,j}-\sqrt{h}(\boldsymbol{e}_z)_{i,j}}}{h} \\
&= \varphi_{zz}(\boldsymbol{x}_{i,j}) - \frac{1}{h}\Big[\Big(\varphi(\boldsymbol{x}_{i,j} + \sqrt{h}(\boldsymbol{e}_z)_{i,j}) + C^{interp}_{\boldsymbol{x}_{i,j}+\sqrt{h}(\boldsymbol{e}_z)_{i,j}} h^2\Big) - 2\varphi(\boldsymbol{x}_{i,j}) \\
&\qquad\qquad + \Big(\varphi(\boldsymbol{x}_{i,j} - \sqrt{h}(\boldsymbol{e}_z)_{i,j}) + C^{interp}_{\boldsymbol{x}_{i,j}-\sqrt{h}(\boldsymbol{e}_z)_{i,j}} h^2\Big)\Big] \\
&= -\left(\frac{1}{12} u_{zzzz}(\boldsymbol{x}_{i,j} + s\sqrt{h}(\boldsymbol{e}_z)_{i,j}) + C^{interp}_{\boldsymbol{x}_{i,j}+\sqrt{h}(\boldsymbol{e}_z)_{i,j}} + C^{interp}_{\boldsymbol{x}_{i,j}-\sqrt{h}(\boldsymbol{e}_z)_{i,j}}\right) h,
\end{aligned}$$

where $s \in [-1, 1]$. Hence,

$$|\varphi_{zz}(\boldsymbol{x}_{i,j}) - D_z^+ D_z^- \varphi_{i,j}| \le C^{zz} h,$$

where the coefficient

$$C^{zz} \equiv \frac{1}{12} \max_{\boldsymbol{x}\in\Omega, \boldsymbol{e}_z} |u_{zzzz}(\boldsymbol{x})| + 2 \max_{\boldsymbol{x}\in\Omega} C_{\boldsymbol{x}}^{interp}$$

is uniformly bounded, independent of h and $(c_{i,j}, \theta_{i,j}) \in \Gamma$. In other words, the truncation error for $D_z^+ D_z^- \varphi_{i,j}$ is $O(h)$.

Now we consider another case, where one of the stencil points of $D_z^+ D_z^- \varphi_{i,j}$ falls outside the computational domain and is thus relocated; see Appendix A.4. Without loss of generality, let us assume again that $\boldsymbol{x}_{i,j} + \eta_1(\boldsymbol{e}_z)_{i,j} \in \partial\Omega$ is the relocated point. The expression for $D_z^+ D_z^- \varphi_{i,j}$ is given by (A.3). The truncation error for $D_z^+ D_z^- \varphi_{i,j}$ is then

$$\begin{aligned}
&\varphi_{zz}(\boldsymbol{x}_{i,j}) - D_z^+ D_z^- \varphi_{i,j} \\
&= \varphi_{zz}(\boldsymbol{x}_{i,j}) - \frac{\frac{\varphi(\boldsymbol{x}_{i,j}+\eta_1(\boldsymbol{e}_z)_{i,j}) - \varphi_{i,j}}{\eta_1} - \frac{\varphi_{i,j} - \mathcal{I}_h\varphi|_{\boldsymbol{x}_{i,j}-\sqrt{h}(\boldsymbol{e}_z)_{i,j}}}{\sqrt{h}}}{\frac{\eta_1+\sqrt{h}}{2}}
\end{aligned}$$

$$= \varphi_{zz}(\boldsymbol{x}_{i,j}) - \frac{\frac{\varphi(\boldsymbol{x}_{i,j}+\eta_1(\boldsymbol{e}_z)_{i,j})-\varphi(\boldsymbol{x}_{i,j})}{\eta_1} - \frac{\varphi(\boldsymbol{x}_{i,j})-\left(\varphi(\boldsymbol{x}_{i,j}-\sqrt{h}(\boldsymbol{e}_z)_{i,j})+C^{interp}_{\boldsymbol{x}_{i,j}-\sqrt{h}(\boldsymbol{e}_z)_{i,j}}h^2\right)}{\sqrt{h}}}{\frac{\eta_1+\sqrt{h}}{2}}$$

$$= \tfrac{1}{3}(\sqrt{h}-\eta_1)u_{zzz}(\boldsymbol{x}_{i,j}) + \tfrac{\eta_1^3}{12(\eta_1+\sqrt{h})}u_{zzzz}(\boldsymbol{x}_{i,j}+s_1\sqrt{h}(\boldsymbol{e}_z)_{i,j}) + \tfrac{h^{3/2}}{12(\eta_1+\sqrt{h})}u_{zzzz}(\boldsymbol{x}_{i,j}-s_2\sqrt{h}(\boldsymbol{e}_z)_{i,j})$$
$$+ \tfrac{2h^{3/2}}{\eta_1+\sqrt{h}}C^{interp}_{\boldsymbol{x}_{i,j}-\sqrt{h}(\boldsymbol{e}_z)_{i,j}},$$

where $s_1, s_2 \in [0,1]$. Hence,

$$|\varphi_{zz}(\boldsymbol{x}_{i,j}) - D_z^+ D_z^- \varphi_{i,j}| \leq \hat{C}^{zz}\sqrt{h} + \hat{D}^{zz}h,$$

where the coefficients

$$\hat{C}^{zz} \equiv \frac{1}{3}\max_{\boldsymbol{x}\in\Omega,\boldsymbol{e}_z}|u_{zzz}(\boldsymbol{x})|, \quad \hat{D}^{zz} \equiv \frac{1}{6}\max_{\boldsymbol{x}\in\Omega,\boldsymbol{e}_z}|u_{zzzz}(\boldsymbol{x})| + 2\max_{\boldsymbol{x}\in\Omega}C^{interp}_{\boldsymbol{x}},$$

are uniformly bounded, independent of h and $(c_{i,j},\theta_{i,j}) \in \Gamma$. In other words, the truncation error for $D_z^+D_z^-\varphi_{i,j}$ is $O(\sqrt{h})$.

There is one more case, where $\boldsymbol{x}_{i,j}+\eta_1(\boldsymbol{e}_z)_{i,j} \in \partial\Omega$ and $\boldsymbol{x}_{i,j}-\eta_2(\boldsymbol{e}_z)_{i,j} \in \partial\Omega$ are both relocated points. Using the similar argument, one can show that the truncation error for $D_z^+D_z^-\varphi_{i,j}$ is again $O(\sqrt{h})$.

Then, similar to Case 1, one can show that the local truncation error of the finite difference scheme at $\boldsymbol{x}_{i,j}$, where the semi-Lagrangian wide stencil discretization is applied, is given by

$$\begin{aligned}&|\,\mathcal{N}(\boldsymbol{x}_{i,j},\varphi(\boldsymbol{x}_{i,j}),D^2\varphi(\boldsymbol{x}_{i,j})) - \mathcal{N}_h(\boldsymbol{x}_{i,j},\varphi_h)\,| \\ &\quad = \begin{cases} O(h), & \text{semi-Lagrangian wide stencil, with all the 4} \\ & \text{wide stencil points } \in \Omega, \\ O(\sqrt{h}), & \text{semi-Lagrangian wide stencil, otherwise.}\end{cases}\end{aligned} \tag{3.38}$$

Finally, we note that the previous proof has assumed that the optimal control is solved exactly, or does not introduce additional truncation error. In Section 3.5, we mentioned that using linear search for the optimal control under the semi-Lagrangian wide stencil discretization introduces truncation error. If we choose $m = O(\sqrt{n_x n_y})$, then $O(h)$ truncation error is introduced [155]. As a result, (3.38) holds. □

3.6.2 Stability

Stability means that the discrete system has a bounded solution u_h. The stability condition is very closely related to the matrix $A_h(c_h, \theta_h)$ being an M-matrix [135]. We will prove that $A_h(c_h, \theta_h)$ is indeed an M-matrix. For convenience, given vectors u_h and v_h, we use $u_h \geq 0$ and $u_h \geq v_h$ to denote $(u_h)_i \geq 0$ and $(u_h)_i \geq (v_h)_i$ for all i. Similarly, given a matrix A, we use $A \geq 0$ to denote $A_{ij} \geq 0$ for all i, j. In other words, the inequalities for vectors and matrices hold for all the elements.

Lemma 3.3 (M-matrix)**.** Suppose an $n \times n$ matrix A satisfies the following:

1. A is an L-matrix: $A_{ii} > 0$ for all i, and $A_{ij} \leq 0$ for all $i \neq j$;
2. A is weakly diagonally dominant: $|A_{ii}| \geq \sum_{j \neq i} |A_{ij}|$; and
3. A has the following connectivity property: Let $\mathcal{G}(A) = \left\{ i \,\middle|\, |A_{ii}| > \sum_{j \neq i} |A_{ij}| \right\} \neq \emptyset$ be the set of rows where strict inequality is achieved. For any $i \notin \mathcal{G}(A)$, there exists a sequence $i_0, i_1, \cdots, i_k$ with $A_{i_r, i_{r+1}} \neq 0$, $0 \leq r \leq k-1$, such that $i_0 = i$ and $i_k \in \mathcal{G}(A)$.

Then A is an M-matrix, and has the following properties:

1. A is non-singular; and
2. $A^{-1} \geq 0$, namely, $(A^{-1})_{ij} \geq 0$ for all i, j.

Proof. We refer the readers to [140, 12, 135]. □

Lemma 3.4. The matrix $A_h(c_h, \theta_h)$, defined in (3.25), is an M-matrix, given an arbitrary control under the admissible control set, i.e., $(c_h, \theta_h) \in \Gamma$.

Proof. The L-matrix condition and the weakly diagonal dominance condition for the matrix $A_h(c_h, \theta_h)$ can be easily verified. For instance, one can verify these conditions for the wide stencil discretization (A.5), using the fact that the neighboring bilinear interpolation weights must sum up to 1 (i.e., $\sum_{k=0}^{1} \sum_{k'=0}^{1} p_{r+k,s+k'} = 1$). The strictly diagonally dominant rows correspond to the grid points near the boundary $\partial\Omega$, while the weakly diagonally dominant rows correspond to those inside the computation domain Ω.

The connectivity property of $A_h(c_h, \theta_h)$ is yet to be verified. For the grid points $\boldsymbol{x}_{i,j}$ that are near the boundary, the lexicographical index $\langle i, j \rangle$, as defined in (3.28), satisfies

$\langle i,j\rangle \in \mathcal{G}(A_h)$. For those points that are inside the computational domain, or $\langle i,j\rangle \notin \mathcal{G}(A_h)$, there must exist non-zero entries $A_{\langle i,j\rangle,\langle i',j'\rangle} \neq 0$, where $i' \geq i$, $j' \geq j$, with at lease one strict inequality satisfied. Hence, given any $\boldsymbol{x}_{i_0,j_0}$, where $\langle i_0,j_0\rangle \notin \mathcal{G}(A_h)$, there exist monotonically increasing sequences $i_0 \leq i_1 \leq ... \leq i_k \leq n_x$ and $j_0 \leq j_1 \leq ... \leq j_k \leq n_y$, such that $\langle i_k,j_k\rangle \in \mathcal{G}(A_h)$. □

Before investigating the stability for the nonlinear problem (3.25)-(3.26), we first prove the stability for the corresponding linear problem (3.25).

Lemma 3.5. Define a circle $B_R(0) : \{(x,y)|x^2+y^2 \leq R^2\}$, where the radius $R = \max_{(x,y)\in\overline{\Omega}} \sqrt{x^2+y^2}$, such that $B_R(0)$ covers the entire computational domain $\overline{\Omega}$. Let $\varphi(\boldsymbol{x}) \equiv -\frac{1}{2}\|\sqrt{f}\|_\infty(R^2-x^2-y^2)$ be a lower-bound estimate function that is smooth and non-positive in $\overline{\Omega}$. Denote its corresponding grid function as $\varphi_h \in \mathbb{R}^{n_x n_y}$. Then the vector $A_h\varphi_h \in \mathbb{R}^{n_x n_y}$ satisfies

$$A_h\varphi_h \leq -\|\sqrt{f}\|_\infty, \text{ for all } h. \tag{3.39}$$

Proof. Without loss of generality, let us consider a grid point $\boldsymbol{x}_{i,j}$ where the semi-Lagrangian wide stencil discretization is applied and boundary terms occur with $\boldsymbol{x}_{i,j}+\sqrt{h}(\boldsymbol{e}_z)_{i,j}$ relocated to $\boldsymbol{x}_{i,j}+\eta_1(\boldsymbol{e}_z)_{i,j}$; see Appendix A.4. Then

$$\begin{aligned}
(A_h\varphi_h)_{\langle i,j\rangle} =\ & 2\left(\frac{c_{i,j}}{\eta_1\sqrt{h}}+\frac{1-c_{i,j}}{h}\right)\varphi(\boldsymbol{x}_{i,j}) - \frac{c_{i,j}}{\sqrt{h}\frac{\eta_1+\sqrt{h}}{2}}\,\mathcal{I}_h\varphi|_{\boldsymbol{x}_{i,j}-\sqrt{h}(\boldsymbol{e}_z)_{i,j}} \\
& - \frac{1-c_{i,j}}{h}\,\mathcal{I}_h\varphi|_{\boldsymbol{x}_{i,j}+\sqrt{h}(\boldsymbol{e}_w)_{i,j}} - \frac{1-c_{i,j}}{h}\,\mathcal{I}_h\varphi|_{\boldsymbol{x}_{i,j}-\sqrt{h}(\boldsymbol{e}_w)_{i,j}} \\
\leq\ & 2\left(\frac{c_{i,j}}{\eta_1\sqrt{h}}+\frac{1-c_{i,j}}{h}\right)\varphi(\boldsymbol{x}_{i,j}) - \frac{c_{i,j}}{\eta_1\frac{\eta_1+\sqrt{h}}{2}}\varphi(\boldsymbol{x}_{i,j}+\eta_1(\boldsymbol{e}_z)_{i,j}) \\
& - \frac{c_{i,j}}{\sqrt{h}\frac{\eta_1+\sqrt{h}}{2}}\varphi(\boldsymbol{x}_{i,j}-\sqrt{h}(\boldsymbol{e}_z)_{i,j}) - \frac{1-c_{i,j}}{h}\varphi(\boldsymbol{x}_{i,j}+\sqrt{h}(\boldsymbol{e}_w)_{i,j}) \\
& - \frac{1-c_{i,j}}{h}\varphi(\boldsymbol{x}_{i,j}-\sqrt{h}(\boldsymbol{e}_w)_{i,j}) \\
=\ & -\|\sqrt{f}\|_\infty,
\end{aligned}$$

where we have used $\varphi(\boldsymbol{x}_{i,j}+\eta_1(\boldsymbol{e}_z)_{i,j}) \leq 0$, and $\mathcal{I}_h\varphi|_{\boldsymbol{x}_{i,j}-\sqrt{h}(\boldsymbol{e}_z)_{i,j}} \geq \varphi(\boldsymbol{x}_{i,j}-\sqrt{h}(\boldsymbol{e}_z)_{i,j})$ and similarly for the other stencil points. Interested readers can prove the other cases in the same fashion. □

Lemma 3.6 (Stability for the linear problem)**.** Assume that an arbitrary control under the admissible control set, i.e., $(c_h, \theta_h) \in \Gamma$, is given for the linearized problem (3.25). Then the solution u_h is bounded as follows:

1. If $g = 0$ (homogeneous boundary condition) and $f \geq 0$ is a bounded function,

$$-\frac{1}{2}\|\sqrt{f}\|_\infty R^2 \leq u_h \leq 0, \text{ independent of } h. \tag{3.40}$$

2. If $f = 0$ (homogeneous PDE) and g is a bounded function,

$$\|u_h\|_\infty \leq \|g\|_\infty, \text{ independent of } h. \tag{3.41}$$

3. In general, if $f \geq 0$ and g are bounded functions,

$$\|u_h\|_\infty \leq \frac{1}{2}\|\sqrt{f}\|_\infty R^2 + \|g\|_\infty, \text{ independent of } h. \tag{3.42}$$

Proof. 1. The proof follows the idea in [136]. In this case, the vector b_h is simply given by $b_{\langle i,j\rangle} = -2\sqrt{c_{i,j}(1-c_{i,j})f_{i,j}}$. Since $c_{i,j} \in [0,1]$, we have $-\|\sqrt{f}\|_\infty \leq b_h \leq 0$.

Lemma 3.4 has proved that A_h is an M-matrix, and thus $A_h^{-1} \geq 0$. Also, we note that $b_h \leq 0$. Hence, the upper bound of u_h is given by $u_h = A_h^{-1} b_h \leq 0$.

Lemma 3.5 has proved that $A_h \varphi_h \leq -\|\sqrt{f}\|_\infty$. Since $-\|\sqrt{f}\|_\infty \leq b_h = A_h u_h$, we have $A_h \varphi_h \leq A_h u_h$. Since $A_h^{-1} \geq 0$, we have $\varphi_h \leq u_h$. Hence, the lower bound of u_h is given by $u_h \geq \varphi_h \geq -\|\varphi\|_\infty = -\frac{1}{2}\|\sqrt{f}\|_\infty R^2$.

2. By Lemma 3.4, A_h is an M-matrix. Then following the proof in [49], the solution u_h under the M-matrix discretization satisfies the discrete comparison principle, and furthermore, (3.41).

3. This can be obtained by applying the linear superposition principle on 1 and 2. □

Now, we come back to our original nonlinear problem (3.25)-(3.26).

Lemma 3.7 (Stability for the nonlinear problem)**.** Assume that f and g are bounded in the L_∞ norm. The solution of the discrete system (3.25)-(3.26), u_h, is bounded by

$$\|u_h\|_\infty \leq \frac{1}{2}\|\sqrt{f}\|_\infty R^2 + \|g\|_\infty, \tag{3.43}$$

where the bound is independent of the mesh size h and the control (c_h, θ_h).

Proof. Given that the linear stability (Lemma 3.6) is proved, the proof of nonlinear stability follows [62]. We note that (3.26) suggests that (c_h^*, θ_h^*) depends on u_h. However, regardless of u_h, (c_h^*, θ_h^*) must fall into the admissible control set $\Gamma = [0,1] \times [-\frac{\pi}{4}, \frac{\pi}{4})$, i.e., each component $(c_{\langle i,j\rangle}^*, \theta_{\langle i,j\rangle}^*)$ must fall into Γ. Hence, (c_h^*, θ_h^*) is always bounded by Γ; the bound of (c_h^*, θ_h^*) is independent of u_h; and, whether u_h is bounded or not is irrelevant to the boundedness of (c_h^*, θ_h^*).

By Lemma 3.6, given any (c_h, θ_h) in the admissible control set Γ, we have $\|u_h\|_\infty \leq \frac{1}{2}\|\sqrt{f}\|_\infty R^2 + \|g\|_\infty$, where the bound is independent of the mesh size h and the control (c_h, θ_h). The solution of the nonlinear problem (3.25)-(3.26) is equivalent to the solution of a linearized problem where the control (c_h^*, θ_h^*) is optimal. Since the optimal control (c_h^*, θ_h^*) must fall into the admissible control set Γ, the same bound for $\|u_h\|_\infty$ applies to the solution of the nonlinear problem. □

3.6.3 Monotonicity

Monotonicity of our numerical scheme (3.25)-(3.26) is inherited from the M-matrix property (or more precisely, L-matrix property) defined in Lemma 3.3.

Lemma 3.8 (Monotonicity)**.** The mixed discretization

$$\mathcal{N}_h(\boldsymbol{x}_{i,j}, u_h) = \mathcal{N}_h(\boldsymbol{x}_{i,j}, u_{i,j}, \{u_{p,q}|_{(p,q)\neq(i,j)}\}) = 0,$$

given in (3.25)-(3.26), is monotone. More specifically, for all $u_h \leq v_h$, we have

$$\begin{aligned} &\mathcal{N}_h(\boldsymbol{x}_{i,j}, u_{i,j}, \{u_{p,q}|_{(p,q)\neq(i,j)}\}) \leq \mathcal{N}_h(\boldsymbol{x}_{i,j}, v_{i,j}, \{u_{p,q}|_{(p,q)\neq(i,j)}\}), \\ &\mathcal{N}_h(\boldsymbol{x}_{i,j}, u_{i,j}, \{u_{p,q}|_{(p,q)\neq(i,j)}\}) \geq \mathcal{N}_h(\boldsymbol{x}_{i,j}, u_{i,j}, \{v_{p,q}|_{(p,q)\neq(i,j)}\}). \end{aligned} \tag{3.44}$$

Proof. The proof follows [66]. Without loss of generality, let us analyze one example: $u_h \leq v_h$ with $u_{i,j} = v_{i,j}$. Then

$$\begin{aligned} &\mathcal{N}_h(\boldsymbol{x}_{i,j}, u_{i,j}, \{u_{p,q}|_{(p,q)\neq(i,j)}\}) - \mathcal{N}_h(\boldsymbol{x}_{i,j}, u_{i,j}, \{v_{p,q}|_{(p,q)\neq(i,j)}\}) \\ &= \max_{(c_{i,j},\theta_{i,j})\in\Gamma} \left\{ (A_h(c_{i,j},\theta_{i,j})\, u_h)_{\langle i,j\rangle} - b_{\langle i,j\rangle}(c_{i,j},\theta_{i,j}) \right\} \\ &\qquad\qquad - \max_{(c_{i,j},\theta_{i,j})\in\Gamma} \left\{ (A_h(c_{i,j},\theta_{i,j})\, v_h)_{\langle i,j\rangle} - b_{\langle i,j\rangle}(c_{i,j},\theta_{i,j}) \right\} \\ &\geq \min_{(c_{i,j},\theta_{i,j})\in\Gamma} \{A_h(c_{i,j},\theta_{i,j})(u_h - v_h)\}_{\langle i,j\rangle} \;\geq\; 0, \end{aligned}$$

where the first inequality uses $\max_x f(x) - \max_x g(x) \geq \min_x [f(x) - g(x)]$, and the last inequality considers that $u_h - v_h \leq 0$ and that all the off-diagonal entries of A_h are non-positive under all admissible control (i.e., A_h is an L-matrix). □

3.6.4 Strong comparison principle

The strong comparison principle holds if the boundary condition is satisfied in the viscosity sense. Unfortunately, there is no proof in the literature that this necessarily holds for the Dirichlet problem (3.4)-(3.6). Hence, we provide a proof in the setting of our proposed numerical scheme.

Lemma 3.9. Let $\zeta(\boldsymbol{x};\boldsymbol{p}) \equiv \frac{1}{2}\|\sqrt{f}\|_\infty \|\boldsymbol{x}-\boldsymbol{p}\|_2^2$, where $\boldsymbol{p}$ is an arbitrary vector in $\mathbb{R}^2$. Let $\hat{u}(\boldsymbol{x}) : \{\boldsymbol{x}_{i,j} \in \Omega\} \cup \partial\Omega \to \mathbb{R}$, where $\hat{u}(\boldsymbol{x}) \equiv \begin{cases} u_h(\boldsymbol{x}_{i,j}), & \text{if } \boldsymbol{x} \in \{\boldsymbol{x}_{i,j} \in \Omega\}, \\ g(\boldsymbol{x}), & \text{if } \boldsymbol{x} \in \partial\Omega. \end{cases}$ Then $\mathcal{I}_h\zeta \pm \hat{u}$ achieves its maximum on $\partial\Omega$.

Proof. Without loss of generality, let us consider again a grid point $\boldsymbol{x}_{i,j} \notin \partial\Omega$ where the semi-Lagrangian wide stencil discretization is applied and the boundary terms occur with $\boldsymbol{x}_{i,j} + \sqrt{h}(\boldsymbol{e}_z)_{i,j}$ relocated to $\boldsymbol{x}_{i,j} + \eta_1(\boldsymbol{e}_z)_{i,j}$. Assume that the control is fixed. Define a linear stencil operator on an arbitrary function u at $\boldsymbol{x}_{i,j}$ as

$$\begin{aligned}\mathcal{S}[u](\boldsymbol{x}_{i,j}) \;\equiv\; & 2\left(\tfrac{c_{i,j}}{\eta_1\sqrt{h}} + \tfrac{1-c_{i,j}}{h}\right) u|_{\boldsymbol{x}_{i,j}} - \tfrac{c_{i,j}}{\sqrt{h}\frac{\eta_1+\sqrt{h}}{2}} u|_{\boldsymbol{x}_{i,j}-\sqrt{h}(\boldsymbol{e}_z)_{i,j}} \\ & - \tfrac{c_{i,j}}{\eta_1\frac{\eta_1+\sqrt{h}}{2}} u|_{\boldsymbol{x}_{i,j}+\eta_1(\boldsymbol{e}_z)_{i,j}} - \tfrac{1-c_{i,j}}{h} u|_{\boldsymbol{x}_{i,j}+\sqrt{h}(\boldsymbol{e}_w)_{i,j}} - \tfrac{1-c_{i,j}}{h} u|_{\boldsymbol{x}_{i,j}-\sqrt{h}(\boldsymbol{e}_w)_{i,j}}.\end{aligned}$$

We note that the relocated stencil point is also included in the operator. Then we have $\mathcal{S}[\mathcal{I}_h\zeta](\boldsymbol{x}_{i,j}) \le \mathcal{S}[\zeta](\boldsymbol{x}_{i,j}) = -\|\sqrt{f}\|_\infty$, and $\mathcal{S}[\hat{u}](\boldsymbol{x}_{i,j}) = -2\sqrt{c_{i,j}(1-c_{i,j})f_{i,j}}$. As a result, we have $\mathcal{S}[\mathcal{I}_h\zeta \pm \hat{u}](\boldsymbol{x}_{i,j}) = -\|\sqrt{f}\|_\infty \pm 2\sqrt{c_{i,j}(1-c_{i,j})f_{i,j}} \le 0$.

Now assume that $\mathcal{I}_h\zeta \pm \hat{u}$ achieves its maximum at this grid point $\boldsymbol{x}_{i,j}$. Next we prove that $(\mathcal{I}_h\zeta \pm \hat{u})|_{\boldsymbol{y}} = (\mathcal{I}_h\zeta \pm \hat{u})|_{\boldsymbol{x}_{i,j}}$ for any stencil point $\boldsymbol{y}$ connected to $\boldsymbol{x}_{i,j}$, namely, for any $\boldsymbol{y} \in \{\boldsymbol{x}_{i,j}+\eta_1(\boldsymbol{e}_z)_{i,j}, \boldsymbol{x}_{i,j}-\sqrt{h}(\boldsymbol{e}_z)_{i,j}, \boldsymbol{x}_{i,j}\pm\sqrt{h}(\boldsymbol{e}_w)_{i,j}\}$. This can be proved by contradiction. Assume that there exists at least one stencil point where the strict inequality holds, namely, $(\mathcal{I}_h\zeta \pm \hat{u})|_{\boldsymbol{y}} < (\mathcal{I}_h\zeta \pm \hat{u})|_{\boldsymbol{x}_{i,j}}$. Then

$$\begin{aligned}\mathcal{S}[\mathcal{I}_h\zeta \pm \hat{u}](\boldsymbol{x}_{i,j}) \;>\; & \left[2\left(\tfrac{c_{i,j}}{\eta_1\sqrt{h}} + \tfrac{1-c_{i,j}}{h}\right)\right. \\ & \left. - \tfrac{c_{i,j}}{\sqrt{h}\frac{\eta_1+\sqrt{h}}{2}} - \tfrac{c_{i,j}}{\eta_1\frac{\eta_1+\sqrt{h}}{2}} - \tfrac{1-c_{i,j}}{h} - \tfrac{1-c_{i,j}}{h}\right] (\mathcal{I}_h\zeta \pm \hat{u})|_{\boldsymbol{x}_{i,j}} = 0,\end{aligned}$$

which contradicts with $\mathcal{S}[\mathcal{I}_h\zeta \pm \hat{u}](\boldsymbol{x}_{i,j}) \le 0$. The key point of this result is that $(\mathcal{I}_h\zeta \pm \hat{u})|_{\boldsymbol{x}_{i,j}+\eta_1(\boldsymbol{e}_z)_{i,j}} = (\mathcal{I}_h\zeta \pm \hat{u})|_{\boldsymbol{x}_{i,j}}$. That is, $\mathcal{I}_h\zeta \pm \hat{u}$ achieves its maximum at the boundary point $\boldsymbol{x}_{i,j} + \eta_1(\boldsymbol{e}_z)_{i,j} \in \partial\Omega$.

In general, consider any grid point $\boldsymbol{x}_{i,j} \notin \partial\Omega$. Assume that $\mathcal{I}_h\zeta \pm \hat{u}$ achieves its maximum at $\boldsymbol{x}_{i,j}$. One can prove in the same fashion that $(\mathcal{I}_h\zeta \pm \hat{u})|_{\boldsymbol{y}} = (\mathcal{I}_h\zeta \pm \hat{u})|_{\boldsymbol{x}_{i,j}}$ for any stencil point $\boldsymbol{y}$ connected to $\boldsymbol{x}_{i,j}$. Then by the connectivity property (see the proof of Lemma 3.4), there exists a boundary point $\boldsymbol{z} \in \partial\Omega$, such that $(\mathcal{I}_h\zeta \pm \hat{u})|_{\boldsymbol{z}} = (\mathcal{I}_h\zeta \pm \hat{u})|_{\boldsymbol{x}_{i,j}}$. Hence, $\mathcal{I}_h\zeta \pm \hat{u}$ achieves its maximum at the boundary point $\boldsymbol{z} \in \partial\Omega$. □

Lemma 3.10. Let Ω be a strictly convex domain. Assume that Lemma 3.9 holds. Define

$$\overline{u}(\boldsymbol{x}) \equiv \limsup_{h\to 0,\, \boldsymbol{y}\to\boldsymbol{x}} u_h(\boldsymbol{y}), \quad \underline{u}(\boldsymbol{x}) \equiv \liminf_{h\to 0,\, \boldsymbol{y}\to\boldsymbol{x}} u_h(\boldsymbol{y}).$$

Then $\overline{u}(\boldsymbol{x}) = \underline{u}(\boldsymbol{x}) = g(\boldsymbol{x})$ for all $\boldsymbol{x} \in \partial\Omega$.

Proof. Given Lemma 3.9, the proof follows Lemma 6.4 in [62]. □

Lemma 3.10 is essentially the comparison result on the boundary $\partial\Omega$. Now we are ready to extend the comparison result to the entire computational domain $\overline{\Omega}$.

Lemma 3.11. Given that the mixed discretization (3.25)-(3.26) satisfies consistency, stability and monotonicity, $\overline{u}(\boldsymbol{x})$ and $\underline{u}(\boldsymbol{x})$ are respectively the viscosity subsolution and supersolution of the Dirichlet problem (3.4)-(3.6).

Proof. See the proof of Theorem 2.1 in [14]. □

Lemma 3.12 (Strong comparison principle)**.** Let Ω be a strictly convex domain. Then the mixed discretization (3.25)-(3.26) satisfies $\overline{u} \le \underline{u}$ in $\overline{\Omega}$.

Proof. Since $\overline{u}$ and $\underline{u}$ are respectively the viscosity subsolution and supersolution (Lemma 3.11), and $\overline{u} \le \underline{u}$ on $\partial\Omega$ (Lemma 3.10), by Theorem 3.3 in [52], we conclude that $\overline{u} \le \underline{u}$ in $\overline{\Omega}$. □

3.6.5 Convergence

Once consistency, stability, monotonicity and the strong comparison principle are proved, the Barles-Souganidis theorem [14] guarantees the convergence of the numerical solution to the viscosity solution.

Theorem 3.2 (Barles-Souganidis theorem)**.** Let Ω be a strictly convex domain. Given that the mixed discretization (3.25)-(3.26) satisfies consistency, stability, monotonicity and the strong comparison principle, the numerical solution converges to the viscosity solution of the Dirichlet problem (3.4)-(3.6).

Proof. See Barles and Souganidis's proof of Theorem 2.1 in [14]. □

3.7 Numerical Results

In this section, we will present numerical results for the Monge-Ampère equation (3.1)-(3.3), or equivalently, its HJB formulation (3.11)-(3.12), using our proposed mixed standard 7-point stencil and semi-Lagrangian wide stencil scheme. The examples in this section come from [72, 21]. We set the tolerance of the residual for the policy iteration (Algorithm 3.2) as 10^{-6}. We let the initial guess of Algorithm 3.2 be $(c_h^{(0)}, \theta_h^{(0)}) = (\frac{1}{2}, 0)$. In other words, we let the initial guess of the numerical solution of (3.1)-(3.3) be the solution of

$$\begin{aligned} u_{xx} + u_{yy} &= 2\sqrt{f}, && \text{in } \Omega, \\ u &= g, && \text{on } \partial\Omega. \end{aligned}$$

We choose the grid size $n_x = n_y = 32, 64, \cdots, 512$, and define the numerical convergence rate as $\log_2 \frac{\|u - u_h(n_x/2, n_y/2)\|}{\|u - u_h(n_x, n_y)\|}$, where $u_h(n_x, n_y)$ is the numerical solution on an $n_x \times n_y$ grid.

Example 3.1. Consider solving (3.1)-(3.3), where

$$f(x,y) = (1 + x^2 + y^2)e^{x^2+y^2}, \qquad g(x,y) = e^{\frac{1}{2}(x^2+y^2)}, \qquad \overline{\Omega} = [-1,1] \times [-1,1].$$

The exact solution $u(x,y) = e^{\frac{1}{2}(x^2+y^2)}$ is smooth. For this example, it turns out that the standard 7-point stencil discretization can be applied on the entire computational domain and still results in a monotone scheme. Consequentially, the numerical solution converges

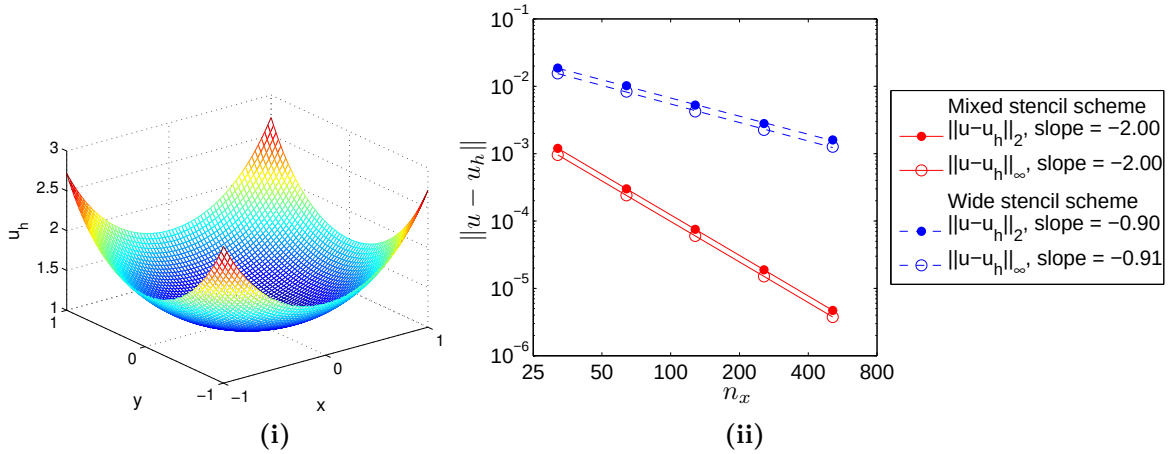


Figure 3.4: Example 3.1, where the exact solution is $u(x,y) = e^{\frac{1}{2}(x^2+y^2)}$. **(i)** Numerical solution. **(ii)** Norms of the errors $\|u - u_h\|$.

(i) Proposed mixed stencil scheme

$n_x \times n_y$	$\|u-u_h\|_2$	Numerical convergence rate	$\|u-u_h\|_\infty$	Numerical convergence rate	Number of policy iterations
32×32	1.201×10^{-3}		9.598×10^{-4}		3
64×64	3.009×10^{-4}	2.00	2.404×10^{-4}	2.00	3
128×128	7.526×10^{-5}	2.00	6.013×10^{-5}	2.00	3
256×256	1.882×10^{-5}	2.00	1.504×10^{-5}	2.00	3
512×512	4.705×10^{-6}	2.00	3.759×10^{-6}	2.00	3

(ii) Pure semi-Lagrangian wide stencil scheme

$n_x \times n_y$	$\|u-u_h\|_2$	Numerical convergence rate	$\|u-u_h\|_\infty$	Numerical convergence rate	Number of policy iterations
32×32	1.868×10^{-2}		1.557×10^{-2}		5
64×64	1.020×10^{-2}	0.87	8.364×10^{-3}	0.90	5
128×128	5.263×10^{-3}	0.95	4.240×10^{-3}	0.98	6
256×256	2.801×10^{-3}	0.91	2.259×10^{-3}	0.91	5
512×512	1.600×10^{-3}	0.81	1.268×10^{-3}	0.83	5

Table 3.1: Numerical results of Example 3.1, where the exact solution is $u(x,y) = e^{\frac{1}{2}(x^2+y^2)}$. **(i)** Proposed mixed stencil scheme. **(ii)** Pure semi-Lagrangian wide stencil scheme.

at the optimal theoretical convergence rate $O(h^2)$. More specifically, the red-solid lines in Figure 3.4(ii) show that, for the proposed mixed stencil scheme, the convergence rates, indicated by the slopes, are $O(h^2)$ in both the L_2 and L_∞ norms. Table 3.1(i) demonstrates the same second order convergence rates. In addition, we observe that the computation is efficient, in the sense that the number of policy iterations remains a small constant 3 as n_x increases.

We compare the proposed mixed scheme with the pure semi-Lagrangian wide stencil scheme in [62], where the wide stencils are applied on the entire computation domain. The blue-dashed lines in Figure 3.4(ii) show that, for the pure semi-Lagrangian wide stencil scheme, the convergence rates are approximately $O(h)$ in both the L_2 and L_∞ norms. Table 3.1(ii) also indicates such first order convergence rates. We note that order one is the optimal theoretical convergence rate for the pure wide stencil scheme; see Lemma 3.2. The convergence rate using the proposed mixed scheme is significantly faster than the rate using the pure semi-Lagrangian wide stencil scheme.

Example 3.2. Consider

$$f(x,y) = \frac{2}{(2-x^2-y^2)^2}, \qquad g(x,y) = -\sqrt{2-x^2-y^2}, \qquad \overline{\Omega} = [0,1]\times[0,1],$$

where f is singular at $(1,1)$. The exact solution is $u(x,y) = -\sqrt{2-x^2-y^2}$. Similar to Example 3.1, we can apply the standard 7-point stencil discretization monotonically on

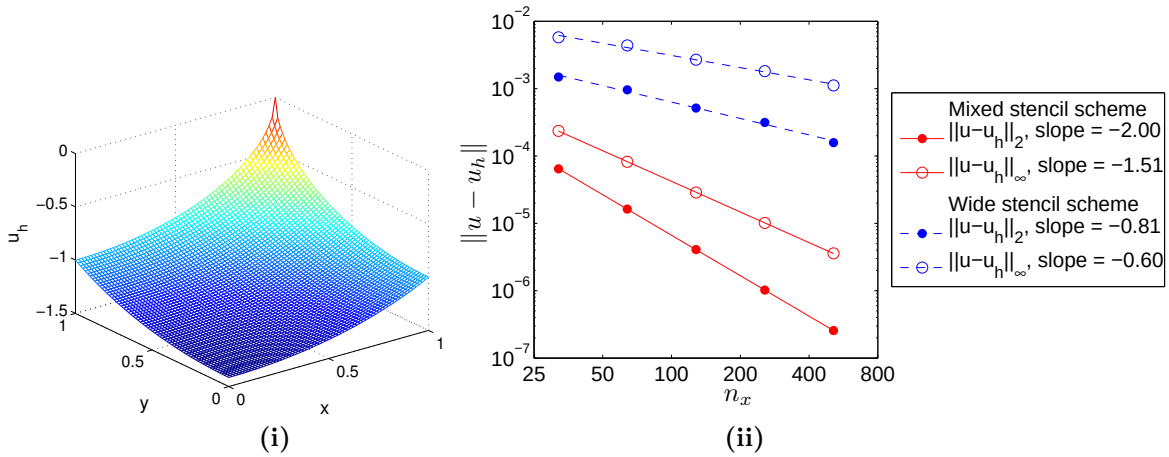


Figure 3.5: Example 3.2, where the exact solution is $u(x,y) = -\sqrt{2-x^2-y^2}$. **(i)** Numerical solution. **(ii)** Norms of the errors $\|u-u_h\|$.

(i) Proposed mixed stencil scheme

$n_x \times n_y$	$\|u-u_h\|_2$	Numerical convergence rate	$\|u-u_h\|_\infty$	Numerical convergence rate	Number of policy iterations
32×32	6.450×10^{-5}		2.359×10^{-4}		4
64×64	1.628×10^{-5}	1.99	8.211×10^{-5}	1.52	5
128×128	4.084×10^{-6}	2.00	2.882×10^{-5}	1.51	5
256×256	1.022×10^{-6}	2.00	1.015×10^{-5}	1.51	5
512×512	2.557×10^{-7}	2.00	3.583×10^{-6}	1.50	5

(ii) Pure semi-Lagrangian wide stencil scheme

$n_x \times n_y$	$\|u-u_h\|_2$	Numerical convergence rate	$\|u-u_h\|_\infty$	Numerical convergence rate	Number of policy iterations
32×32	1.493×10^{-3}		5.799×10^{-3}		5
64×64	9.634×10^{-4}	0.63	4.394×10^{-3}	0.40	4
128×128	5.166×10^{-4}	0.90	2.697×10^{-3}	0.70	5
256×256	3.153×10^{-4}	0.71	1.824×10^{-3}	0.56	5
512×512	1.583×10^{-4}	0.99	1.120×10^{-3}	0.70	5

Table 3.2: Numerical results of Example 3.2, where the exact solution is $u(x,y) = -\sqrt{2-x^2-y^2}$. **(i)** Proposed mixed stencil scheme. **(ii)** Pure semi-Lagrangian wide stencil scheme.

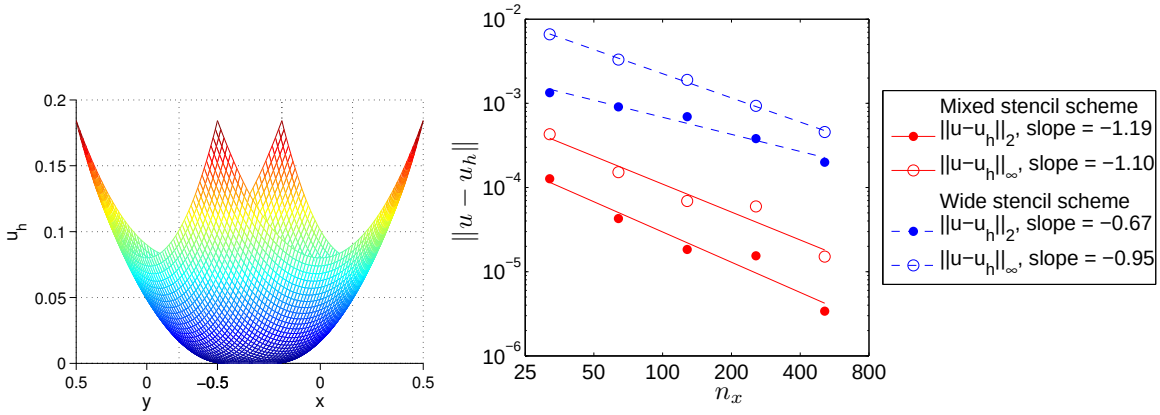
the entire Ω. Both the red-solid lines in Figure 3.5(ii) and the reported numbers in Table 3.2(i) show that, for the proposed mixed stencil scheme, the convergence rates are $O(h^2)$ in the L_2 norm and $O(h^{1.5})$ in the L_∞ norm, respectively. As a comparison, if we applied the pure semi-Lagrangian wide stencil scheme, then the convergence rates are worse than $O(h)$; see the blue-dashed lines in Figure 3.5(ii) and the reported numbers in Table 3.2(ii).

Example 3.3. Consider

$$f(x,y) = \max\left(1 - \frac{0.1}{\sqrt{x^2+y^2}}, 0\right), \qquad g(x,y) = \frac{1}{2}(\sqrt{x^2+y^2} - 0.1)^2,$$
$$\overline{\Omega} = [-0.5, 0.5] \times [-0.5, 0.5].$$

The exact solution is given by $u(x,y) = \frac{1}{2}\max\left(\sqrt{x^2+y^2} - 0.1, 0\right)^2$. This is a C^1 function where the singularity occurs at the ring $x^2 + y^2 = 0.1^2$.

First we consider the proposed mixed scheme. Semi-Lagrangian wide stencils need to be applied near the ring $x^2 + y^2 = 0.1^2$. The red-solid lines in Figure 3.6(ii) show that the convergence rates are approximately $O(h)$ in both the L_2 and L_∞ norms. Table 3.3(i) also reports such convergence rates. We note that the error reduction rates for the sequence of $n_x = 32, 64, \cdots, 512$ do not look as regular as the previous examples. The reason is that wide stencils introduce interpolation errors, which fluctuate as n_x increases, despite converging towards 0. However, a clear error reduction, and thus convergence, can be observed.



(i) **(ii)**

Figure 3.6: Example 3.3, where the exact solution is $\frac{1}{2}\max(\sqrt{x^2+y^2} - 0.1, 0)^2$. **(i)** Numerical solution. **(ii)** Norms of the error $\|u - u_h\|$.

(i) Proposed mixed stencil scheme

$n_x \times n_y$	$\|u-u_h\|_2$	Numerical convergence rate	$\|u-u_h\|_\infty$	Numerical convergence rate	Number of policy iterations
32×32	1.270×10^{-4}		4.298×10^{-4}		4
64×64	4.273×10^{-5}	1.57	1.520×10^{-4}	1.50	6
128×128	1.835×10^{-5}	1.22	6.907×10^{-5}	1.14	7
256×256	1.544×10^{-5}	0.25	5.959×10^{-5}	0.21	9
512×512	3.396×10^{-6}	2.18	1.513×10^{-5}	1.98	20

(ii) Pure semi-Lagrangian wide stencil scheme

$n_x \times n_y$	$\|u-u_h\|_2$	Numerical convergence rate	$\|u-u_h\|_\infty$	Numerical convergence rate	Number of policy iterations
32×32	1.337×10^{-3}		6.604×10^{-3}		5
64×64	9.084×10^{-4}	0.56	3.304×10^{-3}	1.00	6
128×128	6.940×10^{-4}	0.39	1.901×10^{-3}	0.80	7
256×256	3.815×10^{-4}	0.86	9.335×10^{-4}	1.03	7
512×512	1.998×10^{-4}	0.93	4.563×10^{-4}	1.03	9

Table 3.3: Numerical results for Example 3.3, where the exact solution is $\frac{1}{2}\max(\sqrt{x^2+y^2}-0.1,0)^2$. **(i)** Proposed mixed stencil scheme. **(ii)** Pure semi-Lagrangian wide stencil scheme.

For comparison, we also test the pure semi-Lagrangian wide stencil scheme. The blue-dashed lines in Figure 3.6(ii) and the numbers in Table 3.3(ii) show that the convergence rates are worse than $O(h)$ in both the L_2 and L_∞ norms. In addition, the errors $\|u-u_h\|$ by the pure semi-Lagrangian wide stencil scheme (Table 3.3(ii)) are larger than the corresponding errors by our mixed scheme (Table 3.3(i)). Hence, our proposed mixed scheme performs better than the pure wide stencil scheme, in the sense that the errors $\|u-u_h\|$ by the proposed mixed scheme are significantly smaller, and the convergence rates are faster.

Example 3.4. In practice, our numerical scheme can converge to not only viscosity solutions, but also a type of more general weak solutions, called Aleksandrov solutions [84]. In this example, the corresponding f is a delta function at the origin and is zero elsewhere:

$$f(x,y) = \pi\delta(0,0), \quad g(x,y) = \sqrt{x^2+y^2}, \quad \overline{\Omega} = [-0.5,0.5]\times[-0.5,0.5].$$

The exact solution $u(x,y) = \sqrt{x^2+y^2}$ is an Aleksandrov solution. It is a C^0 function and is singular at the origin. Figure 3.7(i) shows that our proposed mixed scheme converges to the cone-shaped Aleksandrov solution. Conversely, Figure 3.7(ii) shows that the pure semi-Lagrangian wide stencil scheme in [62] fails to converge to the cone-shaped Aleksandrov solution. Figure 3.7(iii) and Table 3.4 report the convergence results by the proposed

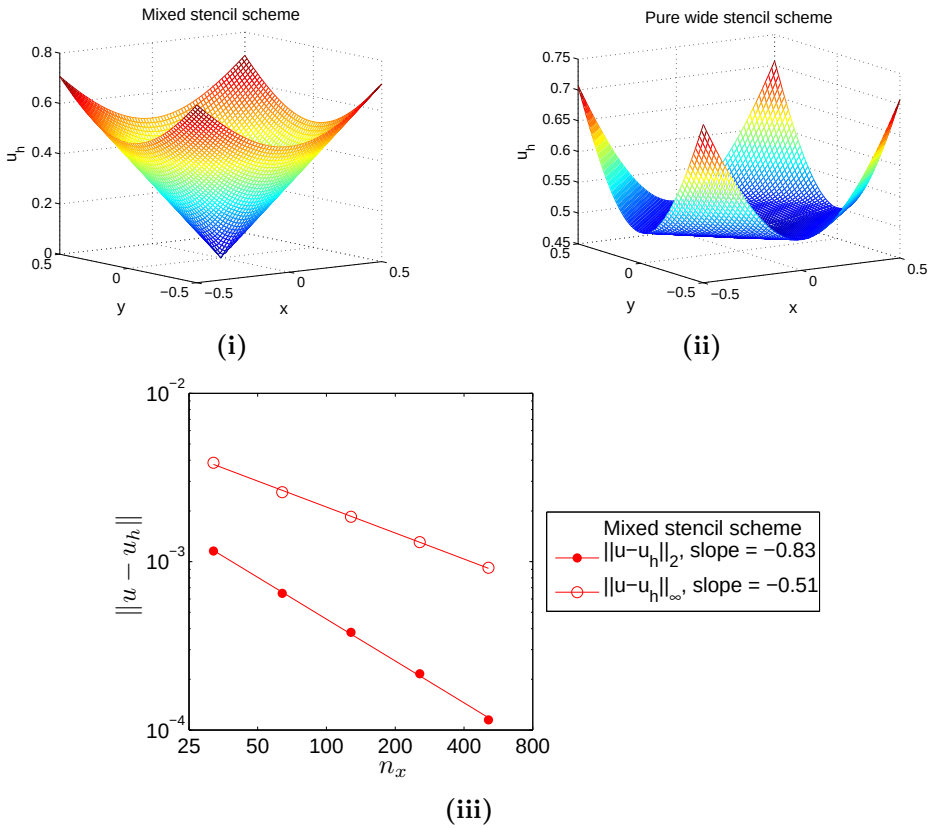


Figure 3.7: Example 3.4, where the exact solution is $u(x,y) = \sqrt{x^2+y^2}$. **(i)** Numerical solution by the proposed mixed stencil scheme. **(ii)** Numerical solution by the pure semi-Lagrangian wide stencil scheme. **(iii)** Norms of the error $\|u - u_h\|$ for the proposed mixed stencil scheme.

mixed scheme. The convergence rates are around $O(h^{0.8})$ in the L_2 norm and $O(h^{0.5})$ in the L_∞ norm, respectively.

Example 3.5. In order to make a case for designing a monotone numerical scheme that converges to the viscosity solution (which is convex), we show explicitly that a non-monotone numerical scheme may converge to a non-viscosity solution (which may be non-convex). More analysis on this issue can be found in [72, 21]. We consider

$$f(x,y) = 1, \qquad g(x,y) = 0, \qquad \overline{\Omega} = [-0.5, 0.5] \times [-0.5, 0.5].$$

For this example, the exact solution u is not smooth near $\partial\Omega$ [21]. Since a closed-form expression for u is not available, we follow [21] and study the convergence behavior of

Proposed mixed stencil scheme

$n_x \times n_y$	$\|u - u_h\|_2$	Numerical convergence rate	$\|u - u_h\|_\infty$	Numerical convergence rate	Number of policy iterations
32×32	1.156×10^{-3}		3.868×10^{-3}		9
64×64	6.484×10^{-4}	0.83	2.583×10^{-3}	0.58	15
128×128	3.803×10^{-4}	0.77	1.848×10^{-3}	0.48	17
256×256	2.159×10^{-4}	0.82	1.305×10^{-3}	0.50	23
512×512	1.148×10^{-4}	0.91	9.203×10^{-4}	0.50	27

Table 3.4: Numerical results of Example 3.4, where the exact solution is $u(x, y) = \sqrt{x^2 + y^2}$. The proposed mixed stencil scheme is used.

u_h towards u by checking the values of $u_h(0, 0)$ as $h \to 0$. The numerical solution using our monotone mixed scheme converges to the convex viscosity solution as $h \to 0$; see Figure 3.8(i) and the left column of Table 3.5. Alternatively, we consider a possible non-monotone discretization for $u_{xx}u_{yy} - u_{xy}^2 = f$, which is the direct application of the standard central differencing on u_{xx}, u_{yy} and the standard 4-point central differencing on u_{xy}. In our numerical experiment, the numerical solution under the non-monotone discretization converges to a concave function as $h \to 0$; see Figure 3.8(ii) and the right column of Table 3.5. We note that [21] considered the same example using a non-monotone discretization, and obtained another non-viscosity solution that is non-convex near $\partial\Omega$.

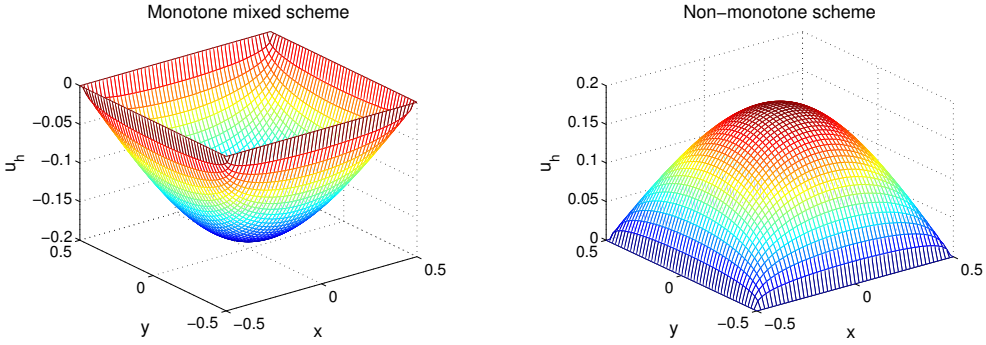


Figure 3.8: Example 3.5: **(i)** The solution given by the monotone mixed scheme, which is convex and is convergent in the viscosity sense. **(ii)** One possible solution given by a non-monotone scheme, which is concave and is not a viscosity solution.

$n_x \times n_y$	$u_h(0,0)$ by monotone scheme	$u_h(0,0)$ by non-monotone scheme
32×32	-0.18380	0.18063
64×64	-0.18444	0.18312
128×128	-0.18461	0.18436
256×256	-0.18485	0.18499
512×512	-0.18507	0.18530

Table 3.5: Example 3.5: The numerical solution at $(0,0)$, i.e., $u_h(0,0)$, given by the proposed monotone mixed scheme and a non-monotone scheme, respectively.

3.8 Conclusion

In this chapter, we convert the Monge-Ampère equation into the equivalent HJB equation, and propose a mixed finite difference discretization for solving the equivalent HJB equation. The discretization satisfies consistency, stability, monotonicity and the strong comparison principle, and hence converges to the viscosity solution. Numerical results show that our proposed mixed scheme can achieve a second order convergence rate whenever the standard 7-point stencils can be applied monotonically on the entire computational domain, and up to an order one convergence rate otherwise. Our proposed mixed scheme significantly improves the accuracy over the pure semi-Lagrangian scheme in [62], in the sense that our proposed mixed scheme yields a smaller discretization error $\|u - u_h\|$ and a faster convergence rate.

Chapter 4

Multigrid Method for HJB Formulation of Monge-Ampère Equation

4.1 Introduction

In Chapter 3, we propose a mixed finite difference method for solving the HJB formulation of a Monge-Ampère equation, which gives rise to a nonlinear discretized system (3.25)-(3.26). In this chapter, we consider multigrid methods for speeding up the solution of the discretized system. As explained in Chapters 1 and 2, either global linearization (i.e., outer-inner linearization) multigrid methods or FAS multigrid methods can be considered for solving the nonlinear discretized system (3.25)-(3.26).

We first develop FAS for the mixed discretization where the standard 7-point stencil discretization can be applied monotonically on the entire computational domain. The major issue is the anisotropy along various directions. Standard pointwise smoothers fail to smooth errors along weakly connected directions. To address this, we propose using an alternating line smoother along with a 7-point restriction and interpolation. We show that such FAS yields a more effective multigrid solver than the corresponding global linearization method with the same multigrid components.

However, when the semi-Lagrangian wide stencil discretization is also applied in the mixed discretization, designing an efficient multigrid method becomes very challenging. One challenge is that standard pointwise smoothers and line smoothers do not smooth

errors at the grid points where wide stencils are applied. In other words, after smoothing, the errors are still locally oscillatory around wide stencils.

The other challenge associated with the wide stencil discretization is that the resulting matrix A_h turns out to be non-symmetric. We note that most multigrid theories are developed for symmetric matrices. The standard multigrid methods may not be effective for non-symmetric matrices. The existing literature of multigrid methods for non-symmetric matrices are mostly restricted to convection-diffusion equations [13, 27, 22, 99]. Only a few investigations, such as [138, 130], are related to non-symmetric matrices beyond convection-diffusion equations. In particular, to the best of our knowledge, [130] is the only reference that investigates multigrid methods in the context of a semi-Lagrangian discretization.

Both [138] and [130] use existing algebraic multigrid (AMG) methods [134, 147, 146, 125, 124] as preconditioners. We refer readers to [153, 146, 157] for substantial reviews of AMG. The basic idea of AMG is to come up with the multigrid components based on the matrix entries $A_{i,j}$. More specifically, one performs coarsening along the strongly connected grid points, where the strength of connections are proportional to the magnitude of the corresponding matrix entries; then one defines interpolation weights, which are again proportional to the magnitude of the corresponding matrix entries. Unfortunately, AMG has a few drawbacks. One is that the interpolation matrices, and furthermore the coarse grid matrices, can be dense, and the density keeps growing as the grid coarsens. Hence, it is expensive to both construct and solve the matrices, and the computational cost per MG iteration may be higher than linear in the system size, i.e., higher than $O(n_x n_y)$. The other issue is that geometric information (such as square grid structure) may be overlooked or even destroyed by AMG. The consequence is that the AMG coarsening strategy and restriction/interpolation are not the optimal choice for the mixed discretization on a square grid. In addition, as pointed out in [130], although AMG methods used as preconditioners give approximately mesh-independent convergence, AMG methods themselves are not efficient if used as stand-alone solvers.

Our contribution is that we propose a fast stand-alone multigrid method for mixed discretization that involves semi-Lagrangian wide stencils. The proposed multigrid method is a global linearization method. To address the challenge from the error smoothing, we propose setting wide stencil points as coarse grid points. The idea is to directly use the coarse grid points to capture the oscillations that cannot be eliminated by smoothing. We note that the resulting grid structure remains approximately square. To address the asymmetry of the matrix A_h, we propose a restriction that is different from the transpose of interpolation. In particular, we propose using injection as the restriction, which yields sparse coarse grid matrices, and hence a more efficient multigrid method than AMG. In our numerical experiments, we illustrate that the proposed multigrid method has a mesh-

independent convergence rate even as a standalone solver.

This chapter is organized as follows. In Sections 4.2 and 4.3, we propose multigrid methods for the standard 7-point stencil discretization and the more general mixed discretization separately. Section 4.4 includes smoothing analysis on the four-direction alternating line smoother. Section 4.5 shows that the proposed multigrid method as a standalone solver can achieve mesh-independent convergence. Section 4.6 is the conclusion.

4.2 Multigrid Methods for Standard 7-point Stencil Discretization

We start with designing FAS multigrid methods for the standard 7-point stencil discretization, or more precisely, the case where the standard 7-point stencil discretization can be applied on the entire computational domain and still results in a monotone scheme.

For FAS, the discretized nonlinear operator $\mathcal{N}_h$ is given by (3.31)-(3.32), where the corresponding linear operator $\mathcal{L}_h$ is given by the standard 7-point stencil discretization (3.19). To obtain $\mathcal{N}_{2h}$, one can use direct discretization, i.e., simply replacing h by $2h$ when performing the discretization (3.19). Then the construction of the coarse grid problem follows (2.17).

4.2.1 Nonlinear smoother

Next we discuss smoothers. First consider the linearized HJB equation (3.11). It turns out that (3.11) may become anisotropic. For instance, when $c^* = \epsilon$ is a small constant close to 0 and $\theta^* = 0$, Equation (3.11) becomes

$$-\epsilon u_{xx} - (1-\epsilon)u_{yy} + 2\sqrt{\epsilon(1-\epsilon)f} = 0,$$

which is an anisotropic Poisson equation. It is well-known that when solving anisotropic equations, the standard pointwise smoothers do not smooth errors along the weakly connected axis, which causes poor convergence rates [153].

To address anisotropy, we follow Section 5.1.3 of [153] and consider using line smoothers. More specifically, instead of updating the unknowns point by point, we update strongly-connected grid points collectively. In general, the strongly-connected direction of the 7-point discretization can change alignment to either the x-axis, or the y-axis, or the diagonal axes, in different parts of the computational domain. Considering this, we apply

four-direction alternating Gauss-Seidel line smoother. More specifically, the line smoother is applied four times: along the x-axis (left to right), the y-axis (top to bottom), the diagonal axis (top left to bottom right) and the transpose diagonal axis (top right to bottom left).

So far we have only described a linear smoother. FAS scheme requires a nonlinear smoother that can directly solve the nonlinear system. Similar to Section 2.4.2, we propose a nonlinear smoother based on policy iteration, where the step of solving the linear system is replaced be applying a one-step linear smoother. We summarize the nonlinear smoother in Algorithm 4.1.

Algorithm 4.1 Nonlinear four-direction alternating Gauss-Seidel line smoother

1: **subroutine** $\bar{u}_h = \text{SMOOTH}\ (u_h)$
2: **for** $i = 1, , ..., n_x$ **do**
3: **for** $j = 1, , ..., n_y$ **do**
4: Update the control: $(\bar{c}_{i,j}, \bar{\theta}_{i,j}) = \underset{(c_{i,j},\theta_{i,j})\in\Gamma}{\arg\max}\ \mathcal{L}_{i,j}(c_{i,j}, \theta_{i,j}; u_h)$.
5: **end for**
6: **end for**
7: Apply the one-step four-direction alternating Gauss-Seidel line smoother to the linearized system $A_h(\bar{c}_h, \bar{\theta}_h)\, u_h = b_h(\bar{c}_h, \bar{\theta}_h)$, which updates the solution $u_h \to \bar{u}_h$.

4.2.2 Restriction and interpolation

Once the error becomes smooth along the x, y and diagonal axes after using the four-direction alternating line smoother, the standard full-coarsening can be applied. In order to capture the directional feature of the 7-point discretization, we follow [153] and apply 7-point restriction operators to (3.19). If we use again the stencil notation introduced in Section 1.3.4 of [153], or equivalently, the stencil notation for (2.22), then under the Conditions (3.15) and (3.17), the corresponding 7-point restriction operators are given by

$$R^{[1]} = \frac{1}{8}\begin{bmatrix} 0 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 0 \end{bmatrix}, \quad R^{[2]} = \frac{1}{8}\begin{bmatrix} 1 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \tag{4.1}$$

respectively. The interpolation operator is the scaled transpose of the restriction operator:

$$P = 4R^T. \tag{4.2}$$

4.3 Multigrid Methods for Mixed Discretization with Wide Stencils

In this section, we will discuss multigrid methods for the more general mixed discretization, where the semi-Lagrangian wide stencil discretization is applied to part of the computational domain. We will propose global linearization multigrid methods instead of FAS methods. One reason is that mixed discretization with wide stencils is a more difficult problem than the pure standard 7-point stencil discretization. We would like to use the Petrov-Galerkin coarse grid operators, which is more robust in terms of the accuracy of the error estimate but is incompatible with the nonlinearity of FAS. Another reason, which will be shown, is that the coarse grids of our proposed approach are no longer square grids, which poses difficulties in defining an FAS coarse grid problem using direct discretization.

4.3.1 Issues

We apply a global linearization method with the components described in Section 4.2 to the mixed discretization. To start with a simple scenario, we consider solving the mixed discretization of the following linearized HJB equation:

$$\begin{aligned} \frac{1}{2}u_{xx} + \frac{1}{2}u_{yy} &= \sqrt{f}, \quad \text{in } \Omega\backslash\{(0,0)\}, \\ \frac{2+\sqrt{2}}{4}u_{xx} + \frac{2-\sqrt{2}}{4}u_{yy} + \frac{1}{\sqrt{2}}u_{xy} &= 0, \quad \text{at } (0,0), \\ u &= g, \quad \text{on } \partial\Omega. \end{aligned} \tag{4.3}$$

In other words, we assume that the control is given as $(c^*, \theta^*) = (\frac{1}{2}, 0)$ on the entire computational domain Ω, where the standard 7-point stencil discretization is applied, except that the control is $(c^*, \theta^*) = (1, \frac{\pi}{8})$ at the origin (the center of Ω), where wide stencil discretization is applied. Figure 4.1(ii) shows the error after applying the four-direction alternating line smoother. In particular, the cross section of the smoothed error shows that a kink appears at the origin (0,0). In general, wherever the wide stencil discretization is applied at a grid point, a kink appears in a smoothed error. Unfortunately, such kinks cannot be eliminated by the other types of smoothers either.

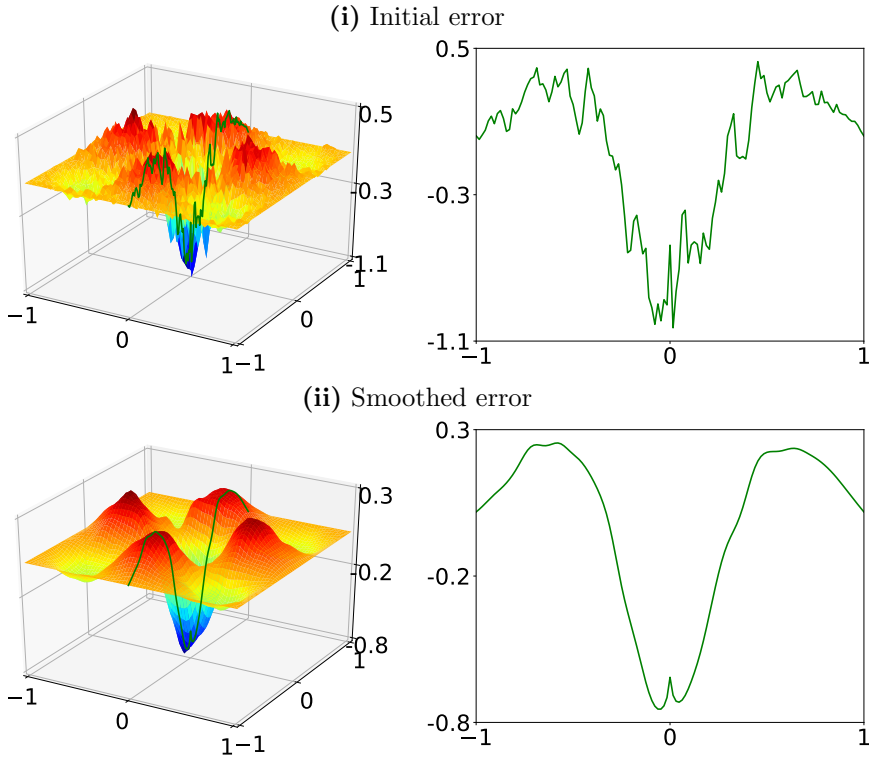


Figure 4.1: The error after one step four-direction alternating Gauss-Seidel line smoothing. **(i)** Initial error and its cross section along the x-axis. **(ii)** Smoothed error and its cross section along the x-axis. A kink appears at the origin (0,0).

4.3.2 Coarsening strategy

Despite kink(s), Figure 4.1(ii) shows that, after smoothing, kink(s) are restricted to the wide stencil point(s), and the error at the other grid points (i.e., the standard 7-point stencil points) is still smooth. This motivates us to apply full-coarsening to the standard 7-point stencil points, and consider a special type of coarsening strategy at the wide stencil points.

To motivate our coarsening strategy for wide stencils, we define a C-point as a fine grid point that is kept in its corresponding coarse grid; and an F-point otherwise. Let us first consider a one-dimensional cross section of a smoothed error; see Figure 4.2(i). Black dots are C-points, while hollow dots are F-points. Assume that the standard full-coarsening assigns a wide stencil point (indicated by the red arrow) as an F-point. Let the black curves represent the underlying fine grid error. On the coarse grid, let its estimated error

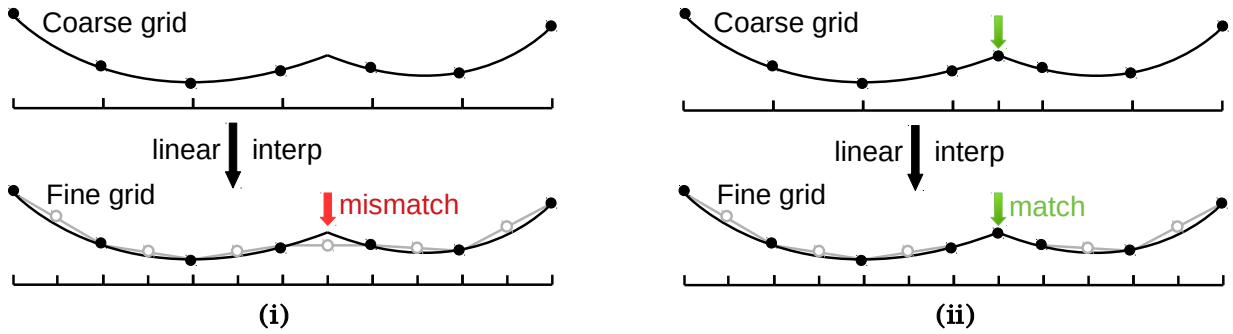


Figure 4.2: Coarsening strategy at a wide stencil point. **(i)** Standard coarsening with linear interpolation at a wide stencil F-point (red arrow). **(ii)** Setting the wide stencil point as a coarse grid C-point (green arrow).

match the underlying fine grid error exactly, i.e., let the values of the black dots sit on the black curve. After linear interpolation of the coarse grid error, we obtain the interpolated error (grey curve) on the fine grid. Ideally, the interpolated error (grey curve) should match the underlying fine grid error (black curve) as closely as possible. However, since the underlying fine grid error has a kink at the wide stencil point, the resulting interpolated error turns out to have a mismatch, as indicated by the red arrow. In other words, if the wide stencil point is an F-point, a linearly interpolated error will fail to capture the kink accurately.

Instead, our approach is simply setting the wide stencil F-point as a coarse grid point, i.e., a C-point; see Figure 4.2(ii). As a result, interpolation at the wide stencil point is no longer needed. The error at the wide stencil point is simply copied from the coarse grid to the fine grid. This yields a more accurate fine grid estimated error, as indicated by the green arrow.

The proposed coarsening strategy can be extended to two dimensions. Figure 4.3 illustrates the coarsening process. On the fine grid, the black dots are selected as C-points, and the hollow dots are selected as F-points. Suppose wide stencils are applied to the three red dots. Then these three dots are all assigned as C-points. The resulting first coarse grid is a combination of a square grid that comes from geometric coarsening, and some additional coarse grid points that come from wide stencils. We can continue to coarsen the square sub-grid and meanwhile keep all the wide stencil points as C-points, which generates the second coarse grid. Such a coarsening strategy can be applied recursively until the coarsest level.

One may argue that by setting all the wide stencil points as coarse grid points, the num-

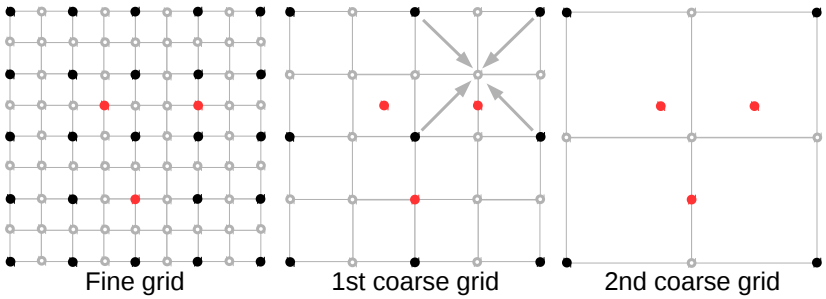


Figure 4.3: Proposed coarsening strategy. Wide stencil grid points (red) are kept as C-points as the grid is coarsened from a fine grid to a coarse grid.

ber of coarse grid points, and thus the computational complexity, will increase. However, it is observed in numerical simulations that wide stencils typically account for a negligible proportion of the total grid points in practical applications (such as image registration). Setting wide stencil points as coarse grid points would not result in a significant increase of the number of coarse grid points, and would still approximately maintain the square grid structure as the grid coarsens.

4.3.3 Interpolation

Under the proposed coarsening strategy, all the wide stencil points are excluded from the set of F-points. In other words, F-points must be the standard 7-point stencils. Hence, the 7-point interpolation, as described in Section 4.2.2, can be used for interpolating the errors at these F-points.

We note that the coarse grids are no longer square grids; see Figure 4.3. However, each of these coarse grids can be seen as a combination of a square grid and some additional wide-stencil C-points. Then all the F-points can still be interpolated from the C-points on the square grid. The arrows in Figure 4.3 show how an F-point can be interpolated.

4.3.4 Restriction

In both the standard geometric and algebraic multigrid methods, restriction is simply the transpose of interpolation. However, it does not result in mesh-independent convergence rates for the non-symmetric matrices A_h arising from the mixed discretization. We will show such poor convergence in Section 4.5.2. Instead, we propose a restriction operator R that is different from the transpose of the interpolation P.

Our approach is simply to use **injection** on wide stencil points. To motivate the use of injection, let us simplify our problem and start with the one-dimensional Poisson equation

$$-u_{xx} = 0, \quad x \in [-0.5, 0.5]. \tag{4.4}$$

We apply the wide stencil discretization at $x = 0$ and the standard finite difference discretization on the rest of the computational domain. Figure 4.4 shows that under our coarsening strategy (which in this case is the same as the standard full coarsening), the fine grid points with even indices are C-points (black points), and the ones with odd indices are F-points (hollow points). The wide stencil point is $i = 0$. A naive choice of restriction at $i = 0$ would be the transpose of the linear interpolation, i.e., the standard full-weighting restriction:

$$r_0^H = \frac{1}{4}r_{-1} + \frac{1}{2}r_0 + \frac{1}{4}r_1, \tag{4.5}$$

where r_{-1}, r_0, r_1 are the fine grid residuals at $i = -1, 0, 1$, respectively, and r_0^H is the restricted residual at the coarse grid point. However, this leads to a poor coarse grid estimated error. In order to find a better restriction, we investigate two cases.

Case 1: $h = \frac{1}{36}$ and $\sqrt{h} = 6h$. Figure 4.4(i) shows that on the fine grid, the stencil points of $i = 0$ fall onto $i = \pm 6$. In this case, the wide stencil discretization at $i = 0$ reads

$$\frac{-u_{-6} + 2u_0 - u_6}{(6h)^2} = 0. \tag{4.6}$$

The residual at $i = 0$ is then given by

$$r_0 = \frac{-e_{-6} + 2e_0 - e_6}{(6h)^2}. \tag{4.7}$$

We notice that $i = 0$, $i = -6$ and $i = 6$ are all C-points. Then a natural construction of the coarse grid problem at $i = 0$ is to discretized the Poisson equation using these three

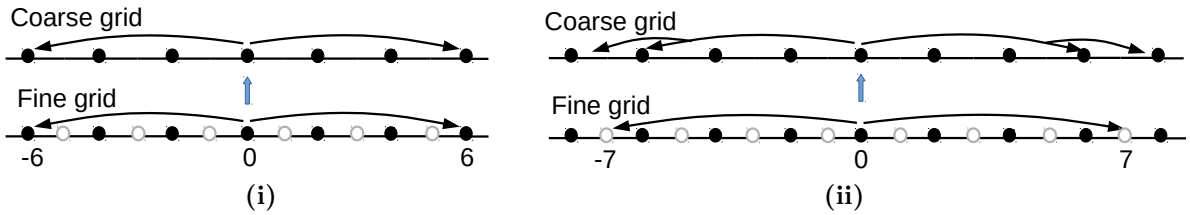


Figure 4.4: Restriction for one-dimensional Poisson equation. **(i)** $h = \frac{1}{36}$ and $\sqrt{h} = 6h$. **(ii)** $h = \frac{1}{49}$ and $\sqrt{h} = 7h$.

points, or more precisely,

$$\frac{-e^H_{-6} + 2e^H_0 - e^H_6}{(6h)^2} = r^H_0\,, \tag{4.8}$$

where the left hand side is a discretization of the Poisson equation on the coarse grid with the stencil length $6h$, and the right hand side is the coarse grid residual r^H_0. Comparing (4.7) and (4.8), we can see that the restriction at $i = 0$ is a simple injection:

$$r^H_0 \equiv r_0. \tag{4.9}$$

Case 2: $h = \frac{1}{49}$ and $\sqrt{h} = 7h$. Figure 4.4(ii) shows that on the fine grid, the stencil points of $i = 0$ fall onto $i = \pm 7$. Unlike the previous case, here the two points $i = \pm 7$ are both F-points. To discretize the Poisson equation on the coarse grid, we interpolate the errors at $i = 7$ and $i = -7$ from their neighboring C-points, which gives

$$\frac{-\frac{1}{2}(e^H_{-8} + e^H_{-6}) + 2e^H_0 - \frac{1}{2}(e^H_6 + e^H_8)}{(7h)^2} = r^H_0\,. \tag{4.10}$$

We want to find a restriction, i.e., to rewrite r^H_0 as a linear combination of fine grid residuals, such that it matches the left hand side of (4.10). One scheme is to use the linear combination of the following fine grid residuals:

$$r_0 = \frac{-e_{-7} + 2e_0 - e_7}{(7h)^2}, \quad r_7 = \frac{-e_6 + 2e_7 - e_8}{h^2}, \quad r_{-7} = \frac{-e_{-6} + 2e_{-7} - e_{-8}}{h^2}. \tag{4.11}$$

If we combine r_0, r_7 and r_{-7} as follows

$$r_0 + \frac{1}{98}r_7 + \frac{1}{98}r_{-7} = \frac{-\frac{1}{2}(e_{-8} + e_{-6}) + 2e_0 - \frac{1}{2}(e_6 + e_8)}{(7h)^2}, \tag{4.12}$$

then (4.12) matches the left hand side of (4.10) in the exact sense. Equation (4.12) defines a possible restriction, i.e.,

$$r^H_0 \equiv r_0 + \frac{1}{98}r_7 + \frac{1}{98}r_{-7}. \tag{4.13}$$

We note that the restriction (4.12) makes use of the residuals r_7 and r_{-7}, which are the points that the wide stencil point $i = 0$ connects to. This is different from the standard full weighting restriction (4.5), which uses the neighboring points r_1 and r_{-1}. Since the coefficients of r_7 and r_{-7} are small, we simply drop them from (4.13) and yield again an injection:

$$r^H_0 \equiv r_0. \tag{4.14}$$

More generally, given a wide stencil C-point $i \in C$ with a stencil length $\sqrt{h}$, the non-zero restriction weights occur at the set of the F-points that it connects to, denoted as $\{j \,|\, j \in F, A_{i,j} \neq 0\}$. We can show that the restriction weights are

$$w_{i,j} = -\frac{A_{i,j}}{A_{j,j}} = -\frac{-\frac{1}{(\sqrt{h})^2}}{\frac{2}{h^2}} = \frac{h}{2}. \tag{4.15}$$

When h is small, the restriction (4.15) can be left out. In other words, injection is sufficient for a good coarse grid problem.

We extend the proposed injection at wide stencil C-points from the one-dimensional Poisson equation to the two-dimensional HJB equation. Once the restriction operator is specified, we construct the Petrov-Galerkin coarse grid operator by

$$A_{2h} \equiv R_h A_h P_h. \tag{4.16}$$

The benefits of injection at wide stencil C-points are mainly two-fold. One is that the resulting restriction operator and Petrov-Galerkin operator (4.16) are significantly sparser than their counterparts if other types of restriction operators are used (such as AMG restriction). This reduces the computational complexity. The other benefit is that such restriction would lead to an accurate coarse grid error estimate and eventually a mesh-independent convergence rate for the proposed multigrid method, which will be shown in Section 4.5.2.

4.4 Local Fourier Analysis

In this section, we use Local Fourier Analysis (LFA) to analyze the smoothing property of the four-direction alternating line smoother for the standard 7-point stencil discretization. First consider the x-line smoother. For simplicity, we assume that the control (c, θ) is constant on the entire computational domain and satisfies Condition (3.15). Following the notation in Section 2.5 and the analysis in [153], we obtain the Fourier symbol of the x-line smoother as

$$\tilde{S}_h(c, \theta; \boldsymbol{\kappa}) = \frac{(\sigma_{22} - \sigma_{12})e^{i\kappa_2} + \sigma_{12}e^{i(\kappa_1+\kappa_2)}}{2(\sigma_{11} + \sigma_{22} - \sigma_{12}) - (\sigma_{11} - \sigma_{12})(e^{-i\kappa_1} + e^{i\kappa_1}) - (\sigma_{22} - \sigma_{12})e^{-i\kappa_2} - \sigma_{12}e^{-i(\kappa_1+\kappa_2)}}.$$

One can also derive the symbol when the control (c, θ) satisfies Condition (3.17) instead. As defined in Section 2.5, the corresponding smoothing factor is $\mu_{loc}(c, \theta) \equiv$

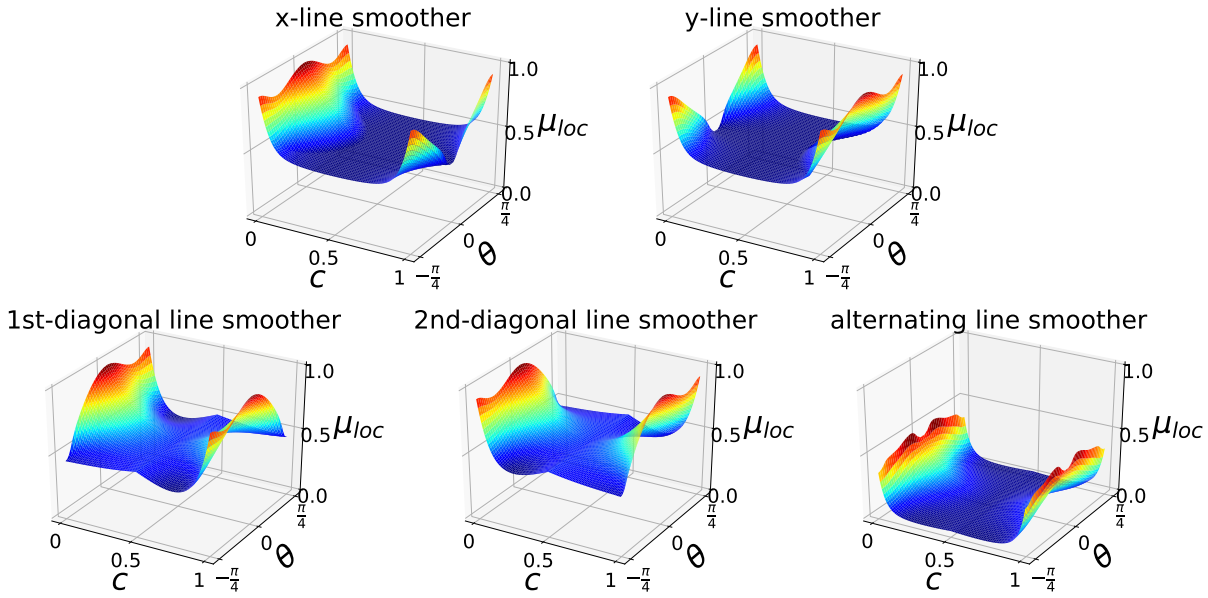


Figure 4.5: Smoothing factor $\mu_{loc}(c,\theta)$ for **(i)** x-line smoother, **(ii)** y-line smoother, **(iii)** first diagonal line smoother, **(iv)** second diagonal line smoother, **(v)** four-direction alternating line smoother.

$\sup_{\boldsymbol{\kappa}} \left\{ |\tilde{S}_h(c,\theta;\boldsymbol{\kappa})| : \boldsymbol{\kappa} \in \text{high frequency mode} \right\}$. A smoother is effective if $\mu_{loc}(c,\theta) \ll 1$ and ineffective if $\mu_{loc}(c,\theta) \approx 1$.

We evaluate the smoothing factors of different line smoothers. Figure 4.5 shows the following:

- *x-line smoother* is effective except at $c \approx 0$, or at $(c,\theta) \approx (1,\frac{\pi}{4}), (1,-\frac{\pi}{4})$.
- *y-line smoother* is effective except at $c \approx 1$, or at $(c,\theta) \approx (0,\frac{\pi}{4}), (0,-\frac{\pi}{4})$.
- *First-diagonal line smoother* (sweeping from top left to bottom right) is ineffective at $c \approx 0$ or $c \approx 1$, but effective at $(c,\theta) \approx (0,-\frac{\pi}{4}), (1,\frac{\pi}{4})$.
- *Second-diagonal line smoother* (sweeping from top right to bottom left) is ineffective at $c \approx 0$ or $c \approx 1$, but effective at $(c,\theta) \approx (0,\frac{\pi}{4}), (1,-\frac{\pi}{4})$.

Four-direction alternating line smoother combines these four smoothers together so that the ineffective part of each individual smoother can be compensated by the others. Figure 4.5 shows that the smoothing factor of the combined smoother is bounded by 0.5 for any $(c,\theta) \in [0,1] \times \left[-\frac{\pi}{4},\frac{\pi}{4}\right)$. Hence, the combined smoother has a good smoothing property.

4.5 Numerical Results

In this section, we demonstrate the mesh-independent convergence rates of the proposed multigrid methods for solving the discretized system (3.25)-(3.26) that arises from the Dirichlet problem (3.1)-(3.3).

4.5.1 Multigrid for standard 7-point stencil discretization

In Examples 4.1–4.2, the standard 7-point stencil discretization can be applied monotonically on the entire computational domain. We compare the performance of two families of multigrid methods - global linearization methods and FAS. For *global linearization methods*, the residual tolerances for the outer policy iteration and the inner multigrid V-cycle are 10^{-6} and 10^{-7}, respectively. The Gauss-Seidel smoother, the standard full coarsening and the 7-point restriction and interpolation described in Section 4.2 are applied. The Petrov-Galerkin coarse grid operators are used to construct coarse grid problems. For *FAS*, the multigrid components are the same as the global linearization methods, except that we use the nonlinear version of the smoothers and direct discretization coarse grid operators.

Example 4.1. We consider again Example 3.1. This example is isotropic, so it suffices to apply the less expensive pointwise Gauss-Seidel smoother. First we show the convergence rates of the global linearization method; see the first and second columns of Table 4.1. To understand the reported numbers, we take the grid size of 32×32 as an example. The numbers "8, 7, 2" mean that it takes 3 policy iterations to converge to the solution of the nonlinear problem, where the 1st policy iteration takes 8 V-cycles to converge to the solution of the linearized problem, the 2nd policy iteration takes 7 V-cycles, and the 3rd policy iteration takes 2 V-cycles. The table shows that the number of multigrid V-cycles within each policy iteration ranges from 2-9. The total number of multigrid V-cycles for solving the nonlinear problem is 17-19, independent of mesh size. As a side remark, we use

$n_x \times n_y$	Global linearization method		FAS
	Number of multigrid V-cycles within each policy iteration	Total number of multigrid V-cycles	Total number of multigrid V-cycles
32×32	8, 7, 2	17	8
64×64	9, 7, 3	19	8
128×128	9, 7, 3	19	9
256×256	9, 7, 3	19	9

Table 4.1: Convergence of the global linearization method and the FAS for Example 4.1.

the solution of the k-th policy iteration, $u_h^{(k)}$, as the initial guess of the multigrid V-cycles at the $(k+1)$-th policy iteration. Hence, as policy iteration converges, the initial guess of multigrid V-cycles becomes more and more precise, and the number of multigrid V-cycles within each policy iteration decreases.

We compare the global linearization method with the FAS iteration. The last column of Table 4.1 shows that the total number of the FAS iterations is 8-9 and is independent of mesh size. We note that for both the global linearization method and the FAS iteration, the computational cost per multigrid iteration is approximately the same. Hence, the FAS iteration is less expensive and converges faster.

Example 4.2. We consider multigrid method for solving (3.1)-(3.3), where

$$f(x,y) = 1 + 24(x+y)^2, \quad g(x,y) = \frac{1}{2}(x^2+y^2) + (x+y)^4, \quad \overline{\Omega} = [-1,1] \times [-1,1].$$

The exact solution is $u(x,y) = \frac{1}{2}(x^2+y^2) + (x+y)^4$. Table 4.2 reports the convergence of the global linearization method using alternating line smoother and pointwise smoother. The multigrid V-cycle with the alternating line smoother converges at 20-32 iterations in total, which is approximately independent of mesh size. Conversely, the multigrid V-cycle with a pointwise smoother converges with more than 70 iterations, and the number

$n_x \times n_y$	MG with alternating line smoother		MG with pointwise smoother
	Number of multigrid V-cycles within each policy iteration	Total number of multigrid V-cycles	Total number of multigrid V-cycles
32×32	5,5,5,3,2	20	73
64×64	5,6,6,4,2,1	24	94
128×128	6,6,7,5,3,1	28	129
256×256	7,7,7,6,3,1	32	161

Table 4.2: Convergence of the global linearization method for Example 4.2 using alternating line smoother and pointwise smoother.

$n_x \times n_y$	Global linearization method	FAS
32×32	20	5
64×64	24	6
128×128	28	6
256×256	32	6

Table 4.3: Total number of multigrid V-cycles of the global linearization method and the FAS for Example 4.2 using the alternating line smoother.

of iterations is more than doubled as n_x increases from 32 to 256. This is because the example is anisotropic, and a pointwise smoother is not efficient in smoothing errors along weakly connected directions.

Similar to Example 4.1, we also compare the total numbers of multigrid V-cycles given by the global linearization method with the numbers given by the FAS. The alternating line smoother is used. Table 4.3 shows that the global linearization method converges in 20-32 iterations, whereas the FAS converges in 5-6 iterations, which is significantly faster.

4.5.2 Multigrid for mixed discretization with wide stencils

In this section, we illustrate the multigrid convergence rates for the mixed discretization. The examples are solved by the global linearization methods introduced in Section 4.3. More specifically, we apply four-direction alternating line smoother. At standard 7-point stencil points, we apply the standard full coarsening and the 7-point restriction and interpolation. At wide stencil points, we set them as coarse grid points, and use injection as the restriction. The Petrov-Galerkin coarse grid operators are used for constructing coarse grid problems.

Example 4.3. We consider solving the linearized HJB equation (4.3), where f and g are the same as in Example 3.1. Consider applying the wide stencil at the origin and the standard 5-point stencil discretization everywhere else. We compare the performance of the proposed multigrid method (Scheme I), the standard multigrid with four-direction alternating line smoother (Scheme II), and the standard multigrid with pointwise Gauss-Seidel smoother (Scheme III). For this example, the only difference between Schemes I and II is that injection is applied at the wide stencil point for Scheme I, while full-weighting restriction is applied at the same point for Scheme II. Table 4.4 shows that Scheme III has poor convergence. Scheme II converges in less than 20 iterations, but the convergence rate grows as n_x increases. Our proposed Scheme I converges in 5-6 iterations, and the convergence rate is independent of mesh size.

Figure 4.6 explains the convergence observed in Table 4.4 by examining the evolution of errors during one two-grid cycle. Only the cross sections along the x-axis are plotted. Start with the same initial error (green lines) for both the proposed and the standard schemes. The pre-smoothed error (blue lines) is smooth everywhere, except that a kink appears at the wide stencil point $x = 0$. Figure 4.6(i) uses the proposed algorithm, where injection is applied at the wide stencil point $x = 0$. The resulting coarse grid problem yields an accurate coarse grid estimated error, i.e., the red line matches the blue line well. Such accurate coarse grid estimate eliminates the error effectively, and yields a small

$n_x \times n_y$	**Scheme I: Proposed MG**	Scheme II: Standard MG with alternating line smoother	Scheme III: Standard MG with pointwise smoother
32×32	**5**	7	23
64×64	**5**	9	46
128×128	**6**	12	198
256×256	**6**	17	more than 200

Table 4.4: Convergence of linear multigrid V-cycles for Example 4.3.

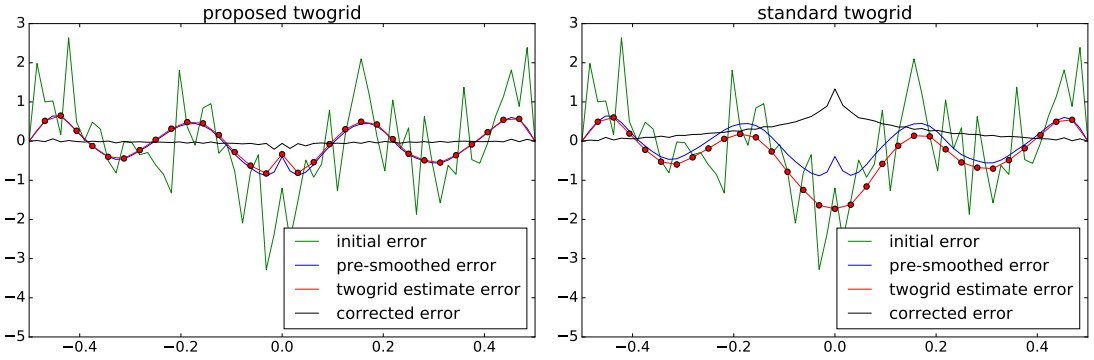


Figure 4.6: Cross sections of errors along the x-axis. **(i)** Proposed algorithm, where injection is used at the wide stencil point $x = 0$. **(ii)** Standard algorithm, where full-weighting restriction is used.

post-corrected error (black line). Conversely, under the same smoother, if the standard full-weighting is used at the wide stencil, then Figure 4.6(ii) shows that the coarse grid estimated error (red line) is no longer a good approximation of the pre-smoothed error (blue line).

Example 4.4. We use the proposed global linearization method to solve (3.1)-(3.3), where

$$f(x,y) = \max\left(1 - \frac{0.15}{\sqrt{x^2+y^2}}, 0\right), \quad g(x,y) = \frac{1}{2}(\sqrt{x^2+y^2} - 0.15)^2$$

on $\overline{\Omega} = [-0.5, 0.5] \times [-0.5, 0.5]$. The viscosity solution is given by $u(x,y) = \frac{1}{2}\max(\sqrt{x^2+y^2} - 0.15, 0)^2$. This is a C^1 function where the solution is not smooth at the ring $x^2+y^2 = 0.15^2$. Semi-Lagrangian wide stencils are applied near the ring.

Table 4.5 reports the convergence of the global linearization method. The number of outer policy iterations increases from 5 to 10 as n_x increases from 32 to 256. Such increase of outer iteration is related to nonlinearity and the singularity on the ring.

$n_x \times n_y$	Number of multigrid V-cycles within each policy iteration	Average number of multigrid V-cycles per policy iteration
32×32	4,5,3,2,1	3.0
64×64	4,6,3,2,1	3.2
128×128	5,6,4,3,3,2	3.8
256×256	6,6,6,6,5,4,3,3,2,1	4.2

Table 4.5: Convergence of the global linearization multigrid method for Example 4.4.

$n_x \times n_y$	Number of multigrid V-cycles within each policy iteration	Average number of multigrid V-cycles per policy iteration
32×32	4,5,5,4,4,4,2	4.0
64×64	5,6,6,7,5,4,4,3,2,1	4.3
128×128	5,7,8,7,7,6,5,5,4,2,1,1	4.8
256×256	6,7,8,8,8,9,8,7,6,5,5,4,4,4,2,1,1	5.5

Table 4.6: Convergence of the global linearization multigrid method for Example 4.5.

To compare the number of multigrid V-cycles across different mesh sizes fairly, we compute the average number of multigrid V-cycles per policy iteration. Table 4.5 shows that the average V-cycle count is approximately a constant ranging from 3.0 to 4.2 as n_x increases from 32 to 256. Hence, the inner multigrid V-cycle for solving linearized systems is nearly mesh-independent.

Example 4.5. We use the proposed global linearization method to solve Example 3.5. Table 4.6 shows that the average number of multigrid V-cycles per policy iteration is approximately a constant ranging from 4.0 to 5.5 as n_x increases from 32 to 256, which is approximately mesh-independent.

4.6 Conclusion

We propose multigrid methods for solving the mixed discretization of the Monge-Ampère equation. We investigate two scenarios. One scenario is when the standard 7-point stencil discretization is applied on the entire computational domain. FAS gives the optimal mesh-independent convergence. The other scenario is the general mixed discretization. Global linearization method is used. We set all wide stencil points as coarse grid points and propose injection of residuals at wide stencil points. The resulting multigrid methods converge at mesh-independent rates.

Chapter 5

Numerical Method for HJB Formulation of Image Registration Model

5.1 Introduction

In Chapters 3-4, we have studied the HJB formulation of the Monge-Ampère equation that is a simplified version of an image registration problem. In this chapter, we return to the image registration problem, and propose a numerical scheme for its HJB formulation. **Optimal mass transport image registration model** (also known as Monge-Kantorovich image registration model) [85, 86, 71, 137, 43, 123] is a non-rigid image registration method. The model treats two images R and T as two mass densities. The goal is to find a mapping which transforms one mass density T to the other R with mass conservation. Such a transformation is non-unique. By defining a transformation-dependent cost function and minimizing it, we can obtain a unique optimal[1] transformation. This optimal transformation has desirable properties. For instance, it is usually diffeomorphic and does not introduce foldings and crossings.

It turns out this image registration model gives rise to a **Monge-Ampère equation**, where the left hand side is the determinant of the Hessian $\det(D^2u)$, and the right hand side depends on the solution u. The Monge-Ampère equation may have multiple weak

[1] Although the optimality of "optimal transformation" may have various meanings, in this chapter, unless otherwise specified, optimality refers to the minimization of the cost function in the mass transport.

solutions, among which there exists a unique globally convex solution, called the scalar potential. The gradient of this solution yields the optimal transformation between R and T [85, 102].

Numerical methods have been proposed for the optimal mass transport model. The first numerical approach, proposed in [85] and [86], is not based on the Monge-Ampère equation. It involves many intermediate steps and requires solving multiple nonlinear PDEs. The nonlinear PDEs in [85] and [86] may give multiple solutions, which correspond to multiple transformations between R and T. To the best of our knowledge, it is unclear whether the numerical scheme in [85] and [86] gives the optimal transformation.

Some authors have investigated numerical schemes for the Monge-Ampère equation arising from the image registration model [71, 137, 43, 36, 37]. In [71], the author proposes a finite difference scheme for the Monge-Ampère equation, and proves that the resulting transformation between R and T is optimal. However, in order to guarantee the optimality, the computational cost *per pixel* must increase to infinity as the image size increases [61], which is not practical for large images. Other numerical schemes based on the Monge-Ampère equation can be found in [137, 43, 36, 37]. Whether the transformations given in [137, 43, 36, 37] are optimal remains an open question.

Another issue in the existing literature is the choice of boundary conditions. We note that the transformation on the boundary of R and T cannot be determined from the mass transport model and thus needs to be specified by model users. Since the transformation on the boundary influences the transformation inside the images, it is crucial to specify an appropriate boundary condition, such that the resulting transformation reflects the physical movement of the object inside R and T. For instance, if the object inside R and T is related by a translation, then the resulting transformation should be able to recover the underlying translation.

A common boundary condition considered in the literature, such as in [71], is a Neumann boundary condition. Alternatively, the authors in [137] use a periodic boundary condition. However, when R and T are related by a translation, or by a combination of a translation and a non-rigid deformation, the resulting transformation under either of these boundary conditions may not be equal to the underlying transformation. To the best of our knowledge, no existing methods using the optimal mass transport model have registered translated images.

In this chapter, we develop a numerical scheme for the Monge-Ampère equation arising from the optimal mass transport image registration model. Our main contributions are the following:

- We propose a new boundary condition where *the gradient of the solution* is periodic[2]. In contrast to the commonly used Neumann boundary condition, our periodic boundary condition can recover the underlying transformation for the images that are related by a translation or by a combination of a translation and a non-rigid deformation.

- We ensure that our numerical scheme yields the optimal transformation between R and T. In order to achieve this, we follow Chapter 3 and design a finite difference scheme based on the **HJB formulation** of the Monge-Ampère equation. As proved in Chapter 3, our numerical solution is guaranteed to converge to the globally convex viscosity solution. Notably, the globally convex solution corresponds to the optimal transformation between R and T [71, 72]. Hence, the resulting transformation computed by our numerical scheme is the optimal one.

- In addition, our numerical scheme can automatically decompose the transformation between R and T into translation and non-rigid deformation components. This extra information is useful in visualizing and understanding the underlying transformation.

This chapter is organized as follows. In Section 5.2, we describe the image registration model based on optimal mass transport. In particular, we have a detailed discussion of boundary conditions. Section 5.3 describes our finite difference discretization based on the HJB formulation. In addition, we propose a modified Levenberg-Marquardt algorithm to solve the discretized system. Experimental results in Section 5.4 show that our numerical scheme gives the optimal transformations, and the results under our periodic boundary condition outperform those under the commonly used Neumann boundary condition. Section 5.5 concludes the chapter.

5.2 Optimal Mass Transport Image Registration Model

In this chapter, we use the following notation:

T	Template image.
R	Reference image.
Ω^T	Domain of template image. For simplicity, assume that $\Omega^T = [0,1] \times [0,1]$.
Ω^R	Domain of reference image. For simplicity, assume that $\Omega^R = [0,1] \times [0,1]$.

[2]This is different from the previously mentioned [137] where periodicity is imposed on the solution itself.

ρ^T	Intensity of template image T on its domain $\Omega^T \subset \mathbb{R}^2$. Must be positive and bounded.
ρ^R	Intensity of reference image R on its domain $\Omega^R \subset \mathbb{R}^2$. Must be positive and bounded.
ϕ	Coordinate transformation $\phi : \Omega^R \to \Omega^T$.
ϕ^*	Optimal coordinate transformation.
T_ϕ	Transformed image under the transformation ϕ.
ρ^{T_ϕ}	Intensity of transformed image T_ϕ on domain Ω^R.
u	Convex scalar potential. $u \in C(\Omega^R)$.

5.2.1 Optimal mass transport image registration model

Consider registering two images T and R. If we view them as two piles of soil with the densities ρ^T and ρ^R, then an image registration problem can be interpreted as a mass transport problem [85, 86, 123]. That is, we consider two piles of soil ρ^T and ρ^R with the same total mass:

$$\int_{\hat{\boldsymbol{x}} \in \Omega^T} \rho^T(\hat{\boldsymbol{x}}) d^2\hat{\boldsymbol{x}} = \int_{\boldsymbol{x} \in \Omega^R} \rho^R(\boldsymbol{x}) d^2\boldsymbol{x}. \tag{5.1}$$

Our goal is to find a coordinate transformation $\phi : \Omega^R \to \Omega^T$, or $\hat{\boldsymbol{x}} = \phi(\boldsymbol{x}) \in \mathbb{R}^2$, such that ρ^T is transformed to ρ^R while the total mass is conserved:

$$\int_{\boldsymbol{x} \in \Omega^R} \rho^T(\phi(\boldsymbol{x})) d^2\phi(\boldsymbol{x}) = \int_{\boldsymbol{x} \in \Omega^R} \rho^R(\boldsymbol{x}) d^2\boldsymbol{x}, \tag{5.2}$$

or equivalently,

$$\rho^T(\phi(\boldsymbol{x})) \det[D\phi(\boldsymbol{x})] = \rho^R(\boldsymbol{x}), \tag{5.3}$$

where $D\phi(\boldsymbol{x}) \in \mathbb{R}^{2\times 2}$ is the Jacobian of the transformation $\phi(\boldsymbol{x})$.

Under the transformation ϕ, define the intensity of the transformed image T_ϕ as

$$\rho^{T_\phi}(\boldsymbol{x}) \equiv \rho^T(\phi(\boldsymbol{x})) \det[D\phi(\boldsymbol{x})]. \tag{5.4}$$

Then the transformed template image is equal to the reference image:

$$\rho^{T_\phi}(\boldsymbol{x}) = \rho^R(\boldsymbol{x}). \tag{5.5}$$

As a result, the mass transport model can transform any template image T to any reference image R [123].

According to (5.4), the mass transport image registration consists of two components. One component is the movement of pixels from $\boldsymbol{x}$ to $\phi(\boldsymbol{x})$, which transforms the image to $\rho^T(\phi(\boldsymbol{x}))$. The other component, called the morphing effect, changes the intensity at

each moved pixel $\phi(\boldsymbol{x})$ by a factor $\det[D\phi(\boldsymbol{x})]$, or more specifically, changes the intensity from $\rho^T(\phi(\boldsymbol{x}))$ to $\det[D\phi(\boldsymbol{x})]\rho^T(\phi(\boldsymbol{x}))$. We note that the morphing effect is an essential part of the mass transport model for two reasons. One is that if pixels can be treated as masses, then after a movement of masses, the accumulation/dissipation of mass in certain regions will inevitably cause an increase/decrease of the intensity. The other reason is that, although the movement of pixels alone makes T close to R, it is the morphing effect that further makes T_ϕ equal R.

The mass transport registration (5.4) is ill-posed. More specifically, there exist multiple transformations that move the soil ρ^T to ρ^R. Among all possible transformations, one of them requires the "least cost", which is desirable. Following [85, 86, 19], we aim to find the optimal transformation $\phi^*(\boldsymbol{x})$ that minimizes the following cost function:

$$\phi^*(\boldsymbol{x}) \equiv \arg\min_{\phi(\boldsymbol{x})} \int_{\mathbb{R}^2} \|\boldsymbol{x} - \phi(\boldsymbol{x})\|^2 \rho^R(\boldsymbol{x}) d^2\boldsymbol{x}, \tag{5.6}$$

which is the weighted least squares displacement of the mass. In essence, (5.6) regularizes the mass transport registration and makes the transformation between ρ^T and ρ^R unique.

5.2.2 Monge-Ampère equation

It has been proved in [102] that the optimal transformation that minimizes the cost function (5.6) can be written as

$$\phi^*(\boldsymbol{x}) = \nabla u(\boldsymbol{x}), \tag{5.7}$$

where $u \in C(\Omega^R)$ is a strictly convex scalar potential field, and its gradient ∇u generates the optimal transformation ϕ^*. Substituting (5.7) into (5.3), we have

$$\det[D^2 u(\boldsymbol{x})] = \frac{\rho^R(\boldsymbol{x})}{\rho^T(\nabla u(\boldsymbol{x}))}, \tag{5.8}$$

$$u \text{ is strictly convex.} \tag{5.9}$$

Equations (5.8)-(5.9) is a **Monge-Ampère equation**. We note that (5.8) is the same as (3.1), except that the right hand side contains the gradient of the solution ∇u.

Due to the nonlinearity, the equation (5.8) itself, without the convexity constraint (5.9), can have multiple solutions [61, 21]. However, the solution of (5.8) that satisfies the convexity constraint (5.9) is unique [71], which we will denote as u^* whenever we need to distinguish it from the other solutions. We emphasize that the convexity of u^* is equivalent to the optimality of the transformation $\phi^* = \nabla u^*$ [85, 71].

The convexity of u^* implies two desirable properties for the optimal transformation ϕ^*:

- ϕ^* does not introduce foldings or invert the order of pixels.
- ϕ^* is diffeomorphic under the assumption that ρ^T, ρ^R are α-Hölder continuous and ϕ^* is bijective.

Here we provide an explanation (rather than a proof) of these two properties. Regarding the first property, take one-dimensional images as an example. The solution u^* being convex means $\phi^*_x = u^*_{xx} > 0$ on the entire computational domain. Then for any $x_1 < x_2$, their corresponding new coordinates satisfy $\phi^*(x_1) < \phi^*(x_2)$. This implies no foldings or inversion of pixel order.

Regarding the second property, since u^* is strictly convex, the Jacobian of the transformation $D\phi^* = D^2u^*$ is positive definite and non-singular on the entire computational domain. Hence, ϕ^* is invertible. Meanwhile, if ρ^T, ρ^R are α-Hölder continuous, then $u^* \in C^2(\Omega)$[71, 39], which implies that $\phi^* = \nabla u^*$ is differentiable. As a result, ϕ^* is diffeomorphic.

5.2.3 Boundary conditions

Up to this point, the image registration model is still incomplete. A boundary condition for the Monge-Ampère equation (5.8)-(5.9) cannot be derived from the mass transport formulation, and is yet to be specified by model users. A natural attempt is to specify a Dirichlet boundary condition. However, in the image registration context, u is a scalar potential, and it is not clear what value of scalar potential to specify on the boundary.

Alternatively, one can specify the value of the transformation ϕ^* on the boundary. For instance, [71] assumes that

$$\text{for any } \boldsymbol{x} \in \partial\Omega^R, \text{ the new coordinate } \phi^*(\boldsymbol{x}) \in \partial\Omega^T,$$

where $\partial\Omega^R$ and $\partial\Omega^T$ are the boundaries of Ω^R and Ω^T, respectively. This means that any pixels that are on the boundary of R must stay on the boundary of T under the mapping. Since $\phi^* = \nabla u$, and we have assumed that $\Omega^R = \Omega^T = [0,1] \times [0,1]$, this boundary condition can be rewritten as

$$u_x|_{x=0} = 0,\ u_x|_{x=1} = 1,\ u_y|_{y=0} = 0,\ u_y|_{y=1} = 1. \tag{5.10}$$

Equation (5.10) is a Neumann boundary condition.

However, under the Neumann boundary condition (5.10), the quality of the resulting transformation ϕ^* may not be good. More specifically, let R be a translation of T; see Figure

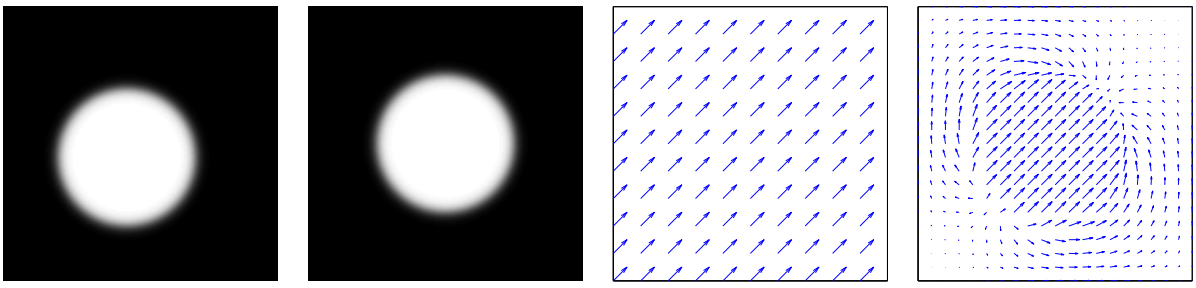
5.1(i)-(ii). We expect the resulting ϕ^* to equal the underlying translation, as shown in Figure 5.1(iii). However, Figure 5.1(iv) shows that in numerical experiments, the resulting ϕ^* is not a translation. The reason is that under a translation, the boundary of $\Omega^R = [0,1] \times [0,1]$ cannot be the boundary of $\Omega^T = [0,1] \times [0,1]$, which violates the assumption of (5.10). More generally, when R and T are related by a combination of a translation and a non-rigid deformation, the resulting ϕ^* under the Neumann boundary condition may not reflect the underlying transformation, which will be shown in Section 5.4.

It seems that specifying non-zero values in (5.10) might recover the underlying transformation between T and R. However, feeding correct non-zero values to (5.10) requires knowing in advance what the underlying transformation is, and normally we do not know the answer (this is exactly what image registration tries to find).

We remark that [137] considers another boundary condition "$u(\boldsymbol{x}) - \frac{1}{2}|\boldsymbol{x}|^2$ is periodic on $\partial\Omega$". Similar to (5.10), this boundary condition does not recover the underlying transformations mentioned in the previous paragraphs.

Our goal is to impose an appropriate boundary condition that will recover the underlying transformation when R and T are related by a translation or by a combination of a translation and a non-rigid deformation. We note that the backgrounds of many images, especially medical images, display only one single color (black or white). Such images can be viewed as periodic density functions, if the domain of the images are extended to $\mathbb{R}^2$. Motivated by this fact and a brief discussion in [85] and [86], we assume that

$$\text{the displacement of a pixel, } \phi^*(\boldsymbol{x}) - \boldsymbol{x}, \text{ is periodic on } \partial\Omega.$$



(i) **(ii)** **(iii)** **(iv)**

Figure 5.1: An example of image registration using the Neumann boundary condition. **(i)** Template image T. **(ii)** Reference image R. **(iii)** Underlying transformation between T and R, which is a pure translation. **(iv)** Transformation given by the Neumann boundary condition.

To take one step further, we substitute $\phi^* = \nabla u$ and obtain

$$\begin{aligned} (u_x - x)_{x=0} &= (u_x - x)_{x=1}, \quad (u_x - x)_{y=0} = (u_x - x)_{y=1}, \\ (u_y - y)_{x=0} &= (u_y - y)_{x=1}, \quad (u_y - y)_{y=0} = (u_y - y)_{y=1}. \end{aligned} \tag{5.11}$$

Equation (5.11) is a periodic boundary condition with respect to *the gradient* of u. To the best of our knowledge, this is the first investigation of gradient-like periodic boundary condition in the context of solving the Monge-Ampère equation for the image registration model.

5.3 Numerical Scheme

In Chapter 3, we have proposed a numerical scheme for a Monge-Ampère equation where the discrete solution is guaranteed to converge to the globally convex viscosity solution. For image registration, the globally convex solution corresponds to the optimal transformation. Hence, it is desirable to extend the numerical scheme proposed in Chapter 3 to the image registration problem (5.7)-(5.9). More specifically, our approach is:

- *Step 1*: converting the Monge-Ampère equation (5.8)-(5.9) into an equivalent HJB equation;
- *Step 2*: performing a mixed standard 7-point stencil and wide stencil finite difference discretization on the equivalent HJB equation; and
- *Step 3*: solving the discretized system.

Steps 1-2 follow Chapter 3. However, Step 3 requires significant modifications. The reason is that, in Chapter 3, the right hand side of (3.1) depends only on $\boldsymbol{x}$ and the boundary condition is Dirichlet; however, for the image registration problem, the right hand side of (5.8) depends on both $\boldsymbol{x}$ and $\nabla u(\boldsymbol{x})$, and the boundary condition is periodic in ∇u. Next we describe the three steps of our numerical scheme.

5.3.1 HJB formulation

Start with Corollary 3.1. We replace $f(\boldsymbol{x})$ in the differential operator (3.13) by $\frac{\rho^R(\boldsymbol{x})}{\rho^T(\nabla u(\boldsymbol{x}))}$. This converts the Monge-Ampère equation (5.8)-(5.9) into an equivalent HJB equation:

Corollary 5.1 (HJB formulation)**.** Let $u \in C^2(\Omega^R)$ be convex, and let $\rho^T \in C(\Omega^T)$ and $\rho^R \in C(\Omega^R)$ be two positive functions. Then the Monge-Ampère equation (5.8)-(5.9) is equivalent to the following HJB equation

$$\hat{\mathcal{L}}_{c^*(\boldsymbol{x}),\theta^*(\boldsymbol{x})}\, u(\boldsymbol{x}) = 0, \tag{5.12}$$

$$\text{subject to } (c^*(\boldsymbol{x}), \theta^*(\boldsymbol{x})) \equiv \underset{(c(\boldsymbol{x}),\theta(\boldsymbol{x}))\in\Gamma}{\arg\max}\ \hat{\mathcal{L}}_{c(\boldsymbol{x}),\theta(\boldsymbol{x})}\, u(\boldsymbol{x}), \tag{5.13}$$

where the differential operator is

$$\begin{aligned}\hat{\mathcal{L}}_{c(\boldsymbol{x}),\theta(\boldsymbol{x})}\, u(\boldsymbol{x}) \equiv\ & -\sigma_{11}(c(\boldsymbol{x}),\theta(\boldsymbol{x}))u_{xx}(\boldsymbol{x}) - 2\sigma_{12}(c(\boldsymbol{x}),\theta(\boldsymbol{x}))u_{xy}(\boldsymbol{x}) \\ & -\sigma_{22}(c(\boldsymbol{x}),\theta(\boldsymbol{x}))u_{yy}(\boldsymbol{x}) + 2\sqrt{c(\boldsymbol{x})(1-c(\boldsymbol{x}))\frac{\rho^R(\boldsymbol{x})}{\rho^T(\nabla u(\boldsymbol{x}))}},\end{aligned} \tag{5.14}$$

and $(c(\boldsymbol{x}), \theta(\boldsymbol{x}))$, Γ, σ_{11}, σ_{22} and σ_{12} are defined in Corollary 3.1.

We remark that unlike the differential operator (3.13), the revised differential operator (5.14) is no longer linear, because the revised term $2\sqrt{c(\boldsymbol{x})(1-c(\boldsymbol{x}))\frac{\rho^R(\boldsymbol{x})}{\rho^T(\nabla u(\boldsymbol{x}))}}$ depends on ∇u.

5.3.2 Finite difference discretization

Next, we discretize the HJB equation (5.12)-(5.14) using the mixed discretization described in Section 3.4. Regarding the discretization of the gradient term $\rho^T(u_x(\boldsymbol{x}_{i,j}), u_y(\boldsymbol{x}_{i,j}))$, we consider using the standard upwinding scheme [152], or more generally, the Godunov scheme [79, 152, 128], which is monotone.

A complete finite difference discretization of (5.12)-(5.14) gives rise to a discrete system. Following Section 3.4.3, we can write the discretized system in the following matrix form:

$$A_h(c_h^*, \theta_h^*)\, u_h = b_h(c_h^*, \theta_h^*;\, u_h), \tag{5.15}$$

$$\text{subject to } \quad (c_h^*, \theta_h^*) \equiv \underset{(c_h,\theta_h)\in\Gamma}{\arg\max} \left\{ A_h(c_h, \theta_h)\, u_h - b_h(c_h, \theta_h;\, u_h) \right\}. \tag{5.16}$$

Compared to (3.25)-(3.26), the major difference in (5.15)-(5.16) is that the right hand side vector b_h not only depends on the control (c_h, θ_h), but also depends on the solution u_h. This is because b_h comes from the discretization of $2\sqrt{c(\boldsymbol{x}_{i,j})(1-c(\boldsymbol{x}_{i,j}))\frac{\rho^R(\boldsymbol{x}_{i,j})}{\rho^T(\nabla u(\boldsymbol{x}_{i,j}))}}$, which depends on ∇u. As a result, (5.15) is not a linear system.

5.3.3 Solving the discretized system

Next, we solve the nonlinear discretized system (5.15)-(5.16). The nonlinearity of (5.15) poses difficulty in using policy iteration. Instead, we consider using more general iterative methods for solving (5.15)-(5.16).

To motivate a general iterative method, we denote the nonlinear discretized system as $\mathcal{N}_h(u_h) = 0$, where

$$\mathcal{N}_h(u_h) \equiv \max_{(c_h,\theta_h)\in\Gamma} \left\{A_h(c_h,\theta_h)\, u_h - b_h(c_h,\theta_h;\, u_h)\right\}. \tag{5.17}$$

Denote the approximate solution at the k-th iteration as $u_h^{(k)}$. Denote the corresponding optimal control as

$$(c_h^{(k)},\theta_h^{(k)}) \equiv \arg\max_{(c_h,\theta_h)\in\Gamma} \left\{A_h(c_h,\theta_h)\, u_h^{(k)} - b_h(c_h,\theta_h;\, u_h^{(k)})\right\}. \tag{5.18}$$

Then the k-th nonlinear discrete operator can be written as

$$\mathcal{N}_h^{(k)} \equiv \mathcal{N}_h(u_h^{(k)}) = A_h(c_h^{(k)},\theta_h^{(k)})\, u_h^{(k)} - b_h(c_h^{(k)},\theta_h^{(k)};\, u_h^{(k)}). \tag{5.19}$$

Furthermore, we obtain the Jacobian matrix of the nonlinear discrete operator as

$$J_h^{(k)} \equiv J_h(u_h^{(k)}) = \frac{d\mathcal{N}_h^{(k)}}{du_h^{(k)}} = \frac{\partial \mathcal{N}_h^{(k)}}{\partial u_h^{(k)}} + \frac{\partial \mathcal{N}_h^{(k)}}{\partial c_h^{(k)}}\frac{\partial c_h^{(k)}}{\partial u_h^{(k)}} + \frac{\partial \mathcal{N}_h^{(k)}}{\partial \theta_h^{(k)}}\frac{\partial \theta_h^{(k)}}{\partial u_h^{(k)}}. \tag{5.20}$$

Since $(c_h^{(k)},\theta_h^{(k)})$ are the optimal control with respect to $u_h^{(k)}$, which implies that $\frac{\partial \mathcal{N}_h^{(k)}}{\partial c_h^{(k)}} = \frac{\partial \mathcal{N}_h^{(k)}}{\partial \theta_h^{(k)}} = 0$, we can simplify the Jacobian as

$$J_h^{(k)} = \frac{\partial \mathcal{N}_h^{(k)}}{\partial u_h^{(k)}} = A_h(c_h^{(k)},\theta_h^{(k)}) - \delta_u b_h(c_h^{(k)},\theta_h^{(k)};u_h^{(k)}), \tag{5.21}$$

where $\delta_u b_h(c_h^{(k)},\theta_h^{(k)};u_h^{(k)})$ represents the Fréchet derivative of b_h with respect to the vector $u_h^{(k)}$, given the fixed control $(c_h^{(k)},\theta_h^{(k)})$.

A standard iterative method for solving (5.15)-(5.16) is to apply Newton's method, which reads

$$\begin{aligned} &e_h^{(k)} \equiv -(J_h^{(k)})^{-1}\mathcal{N}_h^{(k)}, \text{ i.e., solve } J_h^{(k)} e_h^{(k)} = -\mathcal{N}_h^{(k)} \text{ for } e_h^{(k)}, \\ &u_h^{(k+1)} = u_h^{(k)} + e_h^{(k)}. \end{aligned} \tag{5.22}$$

It turns out that Newton's method fails to converge, since the Jacobian is singular.

To explain the reason behind the singular Jacobian, we note that the differential operator $-\sigma_{11}u_{xx} - 2\sigma_{12}u_{xy} - \sigma_{22}u_{yy}$ under the periodic boundary condition (5.11) has two linearly-independent zero kernels $\bar{u}^1(x,y) \equiv x$ and $\bar{u}^2(x,y) \equiv y$, where $\nabla\bar{u}^1 = (1,0)^T$ and $\nabla\bar{u}^2 = (0,1)^T$ correspond to the translations along x and y directions respectively. These two translation kernels are desirable, since they allow translation to be a solution of the image registration problem when R and T are related by a translation. Consequentially, on the discrete level, the matrix A_h inherits two zero kernels $\bar{u}_h^1$ and $\bar{u}_h^2$, where

$$(\bar{u}_h^1)_{\langle i,j\rangle} \equiv x_i, \quad (\bar{u}_h^2)_{\langle i,j\rangle} \equiv y_j, \tag{5.23}$$

and where $\langle i,j\rangle \equiv n_y(i-1)+j$ is the lexicographical index of the vectors. As suggested by (5.21), the Jacobian J_h is a perturbation of A_h by the Fréchet derivative of b_h. It turns out that J_h also has two zero kernels, which are small perturbation of $\bar{u}_h^1$ and $\bar{u}_h^2$. As a result, the Jacobian is singular.

In the event of a singular Jacobian, one may consider replacing the inverse of the Jacobian in (5.22) by its pseudo-inverse. Then the iteration becomes

$$\begin{aligned} &\text{Solve } (J_h^{(k)})^T J_h^{(k)} e_h^{(k)} = -(J_h^{(k)})^T \mathcal{N}_h^{(k)} \text{ for } e_h^{(k)}, \\ &u_h^{(k+1)} = u_h^{(k)} + e_h^{(k)}. \end{aligned} \tag{5.24}$$

More generally, we may consider introducing a regularizer:

$$\begin{aligned} &\text{Solve } [\lambda I + (J_h^{(k)})^T J_h^{(k)}] e_h^{(k)} = -(J_h^{(k)})^T \mathcal{N}^{(k)} \text{ for } e_h^{(k)}, \\ &u_h^{(k+1)} = u_h^{(k)} + e_h^{(k)}, \end{aligned} \tag{5.25}$$

where λ is a non-negative number specified by users. This indeed leads to a known algorithm, called the **Levenberg-Marquardt algorithm** [114, 120].

One advantage of using the Levenberg-Marquardt algorithm (5.25) is that convergence of this iterative solver has been proved [120]. In practice, the parameter λ is changed dynamically to make the algorithm not only convergent, but also more efficient. More specifically, when the approximate solution $u_h^{(k)}$ is not close to the exact solution, λ can be increased, such that the algorithm behaves like a gradient descent and is less likely to diverge. Conversely, when $u_h^{(k)}$ is close to the exact solution, λ can be decreased, such that the algorithm behaves like Newton's method and converges rapidly [120]. In particular, the Levenberg-Marquardt algorithm is able to converge even when the Jacobian becomes singular.

Despite the fact that the Levenberg-Marquardt algorithm (5.25) converges, it may not converge to the solution of the discretized system (5.15)-(5.16). To explain this, we note that in practice, the algorithm is essentially solving the following nonlinear least squares problem

$$u_h = \arg\min_{w_h} \|\mathcal{N}_h(w_h)\|^2. \tag{5.26}$$

Its global minimum, which satisfies $\mathcal{N}_h(u_h) = 0$, is the solution of (5.15)-(5.16). However, it is known that the Levenberg-Marquardt algorithm may converge to a local minimum rather than the global minimum.

The local minimum issue is observed in image registration. For instance, when the two images are related by a translation, the algorithm may be stuck in a local minimum solution that does not correspond to the underlying translation. To be more precise, define the "translation components" of a vector u_h as the projections of u_h to the two translation kernels $\bar{u}_h^1$ and $\bar{u}_h^2$. An initial guess $u_h^{(0)}$ and the exact solution u_h usually have different translation components. The Levenberg-Marquardt algorithm may get stuck in a local minimum before it can fully correct the translation components of $u_h^{(0)}$ to the translation components of u_h.

In order to address the local minimum issue, we add an additional step before each Levenberg-Marquardt iteration. In this additional step, we explicitly correct the translation components of the approximate solution $u_h^{(k)}$. The amount of correction is added such that the corrected solution, denoted as $u_h^{(k+\frac{1}{2})}$, minimizes the residual $\mathcal{N}_h(u_h)$. This gives rise to Algorithm 5.1, which is our final algorithm for solving the discretized system (5.15)-(5.16).

Algorithm 5.1 can be split into two parts: (i) corrections of translation kernels (Lines 4–7), and (ii) the primary nonlinear solver (Lines 8–12). We note that $(\epsilon_1^{(k)}, \epsilon_2^{(k)})$ is the amount of corrected translation components along the x and y directions. We can use simple search to solve Line 5. That is, we discretize $[-\frac{1}{2}, \frac{1}{2}] \times [-\frac{1}{2}, \frac{1}{2}]$ into a discrete set and then directly search for the minimum on the discrete set. Also, whenever $(\epsilon_1^{(k)}, \epsilon_2^{(k)}) = (0, 0)$ at the k-th iteration, which means that the correction for the translation components of the approximate solution is completed, then Lines 4–7 can be skipped for future iterations. It turns out that Lines 4–7 only need to be executed for very few iterations (typically, around 1-3 iterations in our experiments).

An additional benefit of Algorithm 5.1 is that it decomposes the resulting transformation $\phi^* = \nabla u$ into a pure translation component and a non-rigid deformation component. The pure translation component is given by the accumulation of the corrections of the

Algorithm 5.1 Modified Levenberg-Marquardt algorithm for solving (5.15)-(5.16)

Start with an initial guess $u_h^{(0)} = \frac{1}{2}(x^2 + y^2)$.
Set $(\epsilon_1^{(-1)}, \epsilon_2^{(-1)}) = (\infty, \infty)$.
for $k = 0, 1, ...$ until convergence **do**
  if $(\epsilon_1^{(k-1)}, \epsilon_2^{(k-1)}) \neq (0, 0)$ **then**
    $(\epsilon_1^{(k)}, \epsilon_2^{(k)}) = \underset{(\epsilon_1,\epsilon_2)\in[-\frac{1}{2},\frac{1}{2}]\times[-\frac{1}{2},\frac{1}{2}]}{\arg\min} \|\mathcal{N}_h(u_h^{(k)} + \epsilon_1 \bar{u}_h^1 + \epsilon_2 \bar{u}_h^2)\|$.
    $u_h^{(k+\frac{1}{2})} = u_h^{(k)} + \epsilon_1^{(k)} \bar{u}_h^1 + \epsilon_2^{(k)} \bar{u}_h^2$.
  end if
  Solve the optimization (5.18) for $(c^{(k+\frac{1}{2})}, \theta^{(k+\frac{1}{2})})$ based on Section 3.5.
  Compute $\mathcal{N}_h^{(k+\frac{1}{2})} \equiv \mathcal{N}_h(u_h^{(k+\frac{1}{2})})$ by (5.19).
  Compute $J_h^{(k+\frac{1}{2})} \equiv J_h(u_h^{(k+\frac{1}{2})})$ by (5.21).
  Solve
$$[\lambda I + (J_h^{(k+\frac{1}{2})})^T J_h^{(k+\frac{1}{2})}] e_h^{(k+\frac{1}{2})} = -(J_h^{(k+\frac{1}{2})})^T \mathcal{N}_h^{(k+\frac{1}{2})}.$$
  for $e_h^{(k+\frac{1}{2})}$.
  $u_h^{(k+1)} = u_h^{(k+\frac{1}{2})} + e_h^{(k+\frac{1}{2})}$.
end for

translation kernels, or more precisely, the accumulation of $(\epsilon_1^{(k)}, \epsilon_2^{(k)})$:

$$\phi^{tran} \equiv \sum_{k\geq 0} (\epsilon_1^{(k)}, \epsilon_2^{(k)})^T. \tag{5.27}$$

Subtracting ϕ^{tran} from the resulting transformation ϕ^* yields the remaining non-rigid deformation component.

Eventually, to link the theory with the experiments, we summarize the complete implementation of our numerical scheme in Algorithm 5.2.

5.4 Numerical Results

In this section, we present experimental results for image registration using our proposed Algorithm 5.2. One criteria of evaluating the quality of registration is the quality of the resulting transformation ϕ^*. We expect ϕ^* to reflect the underlying transformation

Algorithm 5.2 Numerical solution of image registration problem (5.7)-(5.9)

1: Normalize ρ^R and ρ^T to $\|\rho^R\| = \|\rho^T\| = 1$, which satisfies (5.1).
2: Discretize (5.8)-(5.9) as described in Sections 5.3.1–5.3.2, yielding the discrete system (5.15)-(5.16).
3: Solve (5.15)-(5.16) by Algorithm 5.1, yielding the convex scalar potential u.
4: Compute the transformation $\phi^* = \nabla u$.
5: Decompose the transformation ϕ^* into the translation component ϕ^{tran} and the non-rigid deformation component.
6: Compute the transformed image (5.4).

between the two images T and R, especially when they are related by a translation, or by a combination of a translation and a non-rigid deformation. We will evaluate the resulting transformations qualitatively in Sections 5.4.1–5.4.3, and quantitatively in Section 5.4.4.

An additional criteria to consider is the morphing effect, which is the change of intensity at each moved pixel. We refer readers to Section 5.2.1 for a description of the morphing effect. Although the morphing effect is an essential component of the mass transport registration, in some image registration applications, where the physical object inside the two images is (nearly) incompressible, the morphing effect may be undesirable [123]. Considering this, it is good to suppress the morphing effect under the framework of mass transport. We will see that this can be achieved by imposing our periodic boundary condition (5.11).

To quantify the morphing effect, we define the *morphing magnitude*:

$$\mu(\boldsymbol{x}) \equiv \log_{10} \det[D\phi^*(\boldsymbol{x})], \tag{5.28}$$

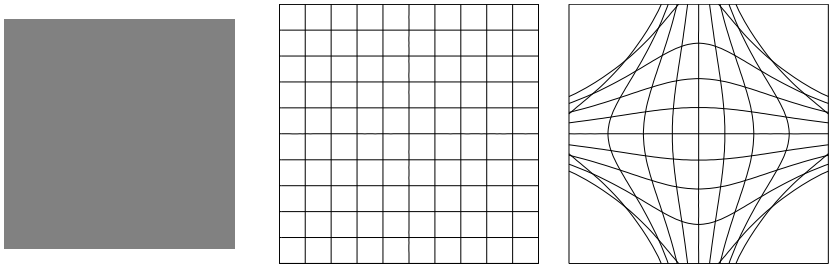
which is the (logarithmic) ratio of the intensity before and after morphing at a moved pixel $\boldsymbol{x} \to \phi^*(\boldsymbol{x})$. We will visualize the morphing magnitude $\mu(\boldsymbol{x})$ using color scale in the deformed mesh images in Figures 5.3–5.9. We note that $\mu(\boldsymbol{x}) = 0$, i.e., white color at a local pixel $\boldsymbol{x}$, means no morphing effect; the larger $|\mu(\boldsymbol{x})|$ is, i.e., the more intense the red/blue color is, the more severe the morphing effect is. Table 5.1 summarizes how to interpret the morphing magnitude. Note that "area of pixel" is represented by the area of each tiny mesh element in the deformed mesh images.

5.4.1 Optimal versus non-optimal transformations

Example 5.1 (Figure 5.2)**.** In this example, we assume that the template and reference images are the same constant image, where $\rho^T = \rho^R = 1$ on the entire domain $\Omega =$

morphing magnitude	color scale	net flow of masses	area of pixel	intensity of pixel
$\mu(\boldsymbol{x}) = 0$	white	zero	invariant	invariant
$\mu(\boldsymbol{x}) > 0$	red	inflow	compressed	increased
$\mu(\boldsymbol{x}) < 0$	blue	outflow	expanded	decreased

Table 5.1: Interpretation of the morphing magnitude at a pixel $\boldsymbol{x}$.



(i) **(ii)** **(iii)**

Figure 5.2: Optimal versus non-optimal transformations. **(i)** Constant images R and T. **(ii)** The optimal transformation ϕ^* obtained by our monotone scheme, which is the identity mapping. **(iii)** A non-optimal transformation ϕ obtained by a non-monotone scheme.

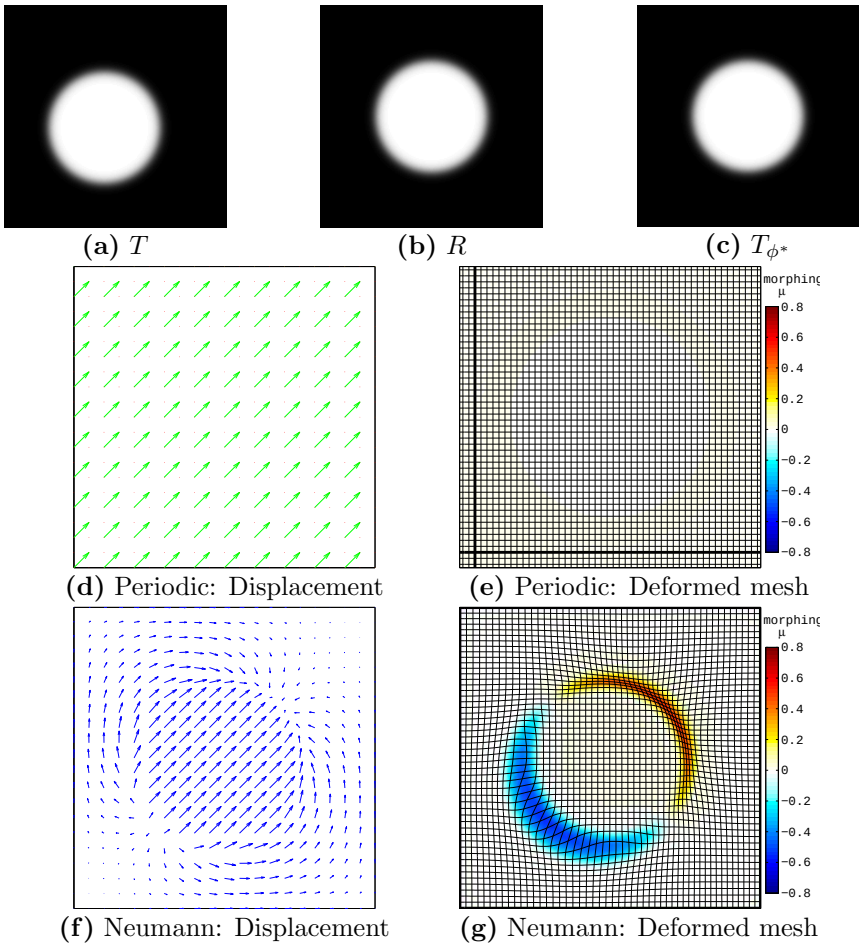
$[0,1]\times[0,1]$; see Figure 5.2(i). We compute a transformation using our monotone numerical scheme, and another transformation using the non-monotone finite difference scheme in [21]. To visualize the two computed transformations, we apply them to a square mesh and obtain two deformed meshes, as plotted in Figure 5.2(ii)–(iii). Figure 5.2(ii) shows that the transformation given by our monotone numerical scheme is the identity mapping $\phi^*(\boldsymbol{x}) = \boldsymbol{x}$ and maps a square mesh to itself. The identity mapping is indeed the optimal transformation for this example. Conversely, Figure 5.2(iii) shows that the non-monotone scheme gives a non-optimal transformation, which severely deforms a square mesh.

5.4.2 Periodic versus Neumann boundary conditions

Example 5.2 (Figure 5.3)**.** We revisit the case where R is a translation of T. Here we let the true underlying translation ϕ^*_{true} be

$$(\phi^*_{true})^{-1}(\boldsymbol{x}) \equiv \boldsymbol{x} + (0.05, 0.05)^T,\ \boldsymbol{x} \in \Omega^T.$$

The reason to define ϕ^*_{true} in terms of $(\phi^*_{true})^{-1}$ is that it is more intuitive to express the mapping from Ω^T to Ω^R, noting that ϕ is originally defined as a mapping from Ω^R to Ω^T.



(a) T **(b)** R **(c)** T_{ϕ^*}

(d) Periodic: Displacement **(e)** Periodic: Deformed mesh

(f) Neumann: Displacement **(g)** Neumann: Deformed mesh

Figure 5.3: Example 5.2: **(a)** Template image T. **(b)** Reference image R, where T and R are related by a translation. **(c)** Transformed image T_{ϕ^*} under the *periodic boundary condition*. **(d)** Displacement of pixels from T to T_{ϕ^*} under the *periodic boundary condition*, which is a pure translation. **(e)** A deformed mesh obtained by applying the transformation $(\phi^*)^{-1}$ on a square mesh. $(\phi^*)^{-1}$ is computed under the *periodic boundary condition*. The thick black lines show where the boundary of $\Omega = [0,1] \times [0,1]$ is moved to under $(\phi^*)^{-1}$. The color bar is the morphing magnitude μ. The intensity of the color shows the degree of morphing effect under $(\phi^*)^{-1}$. **(f)** Displacement of pixels from T to T_{ϕ^*} under the *Neumann boundary condition*. **(g)** A deformed mesh obtained by applying the transformation $(\phi^*)^{-1}$ on a square mesh. $(\phi^*)^{-1}$ is computed under the *Neumann boundary condition*.

As expected, the *periodic boundary condition* (5.11) yields a constant translation on the entire image; see Figure 5.3(d). Figure 5.3(e) shows that a square mesh is translated under

$(\phi^*)^{-1}$ and remains a square mesh. In particular, we observe a translation of the original boundary lines $x = 0$ and $y = 0$ to the thick vertical line $x = 0.05$ and the thick horizontal line $y = 0.05$, respectively. These suggest that the resulting ϕ^* is a pure translation.

On the contrary, under the *Neumann boundary condition* (5.10), the resulting transformation is not a constant translation; see Figure 5.3(f). In Figure 5.3(g), we observe that in some regions, red/blue color is intense and parts of the mesh are severely compressed/expanded. This indicates that the registration under the Neumann boundary condition relies heavily on the morphing effect.

Example 5.3 (Figure 5.4)**.** We consider two images R and T that are related by a combination of a translation and a dilation. We specify the true underlying transformation as

$$(\phi^*_{true})^{-1}(\boldsymbol{x}) \equiv \lambda \left[\boldsymbol{x} - \begin{pmatrix} 0.45 \\ 0.45 \end{pmatrix} \right] + \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}, \ \boldsymbol{x} \in \Omega^T,$$

with the scaling (or dilating) factor

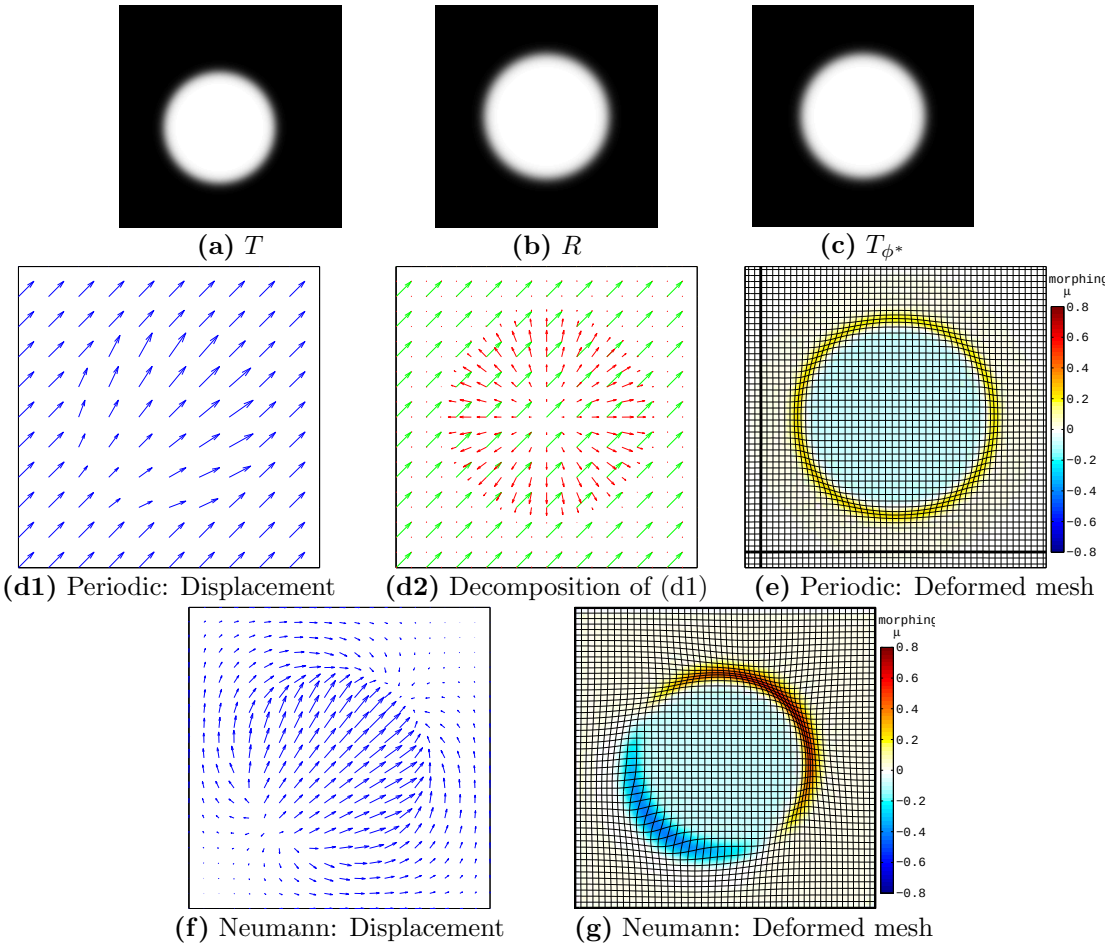
$$\lambda \equiv \begin{cases} 1.12, & \text{if } \|(\phi^*_{true})^{-1}(\boldsymbol{x}) - (0.5, 0.5)^T\|^2 \leq 0.33^2, \\ 1, & \text{otherwise.} \end{cases}$$

Under the *periodic boundary condition*, the resulting transformation is shown in Figure 5.4(d1). To better understand this resulting transformation, we use (5.27) to decompose it into translation and non-rigid deformation components, represented by green and red arrows respectively in Figure 5.4(d2). We observe that the non-rigid deformation component (red arrows) is clearly a dilation. Figure 5.4(e) shows once again that the boundary lines $x = 0$ and $y = 0$ are translated under $(\phi^*)^{-1}$ to $x = 0.05$ and $y = 0.05$. Also, the deformed mesh is a symmetric and isotropic dilation with respect to the center of the circle.

However, if we use the *Neumann boundary condition*, then we cannot identify the underlying combination of a translation and a dilation in Figures 5.4(f) and 5.4(g). In addition, by a comparison between Figures 5.4(e) and 5.4(g), we observe that the morphing effect under the periodic boundary condition is mild compared to the Neumann boundary condition.

Example 5.4 (Figure 5.5)**.** We consider two images R and T where the true underlying transformation

$$(\phi^*_{true})^{-1}(\boldsymbol{x}) \equiv \begin{pmatrix} \cos\frac{\pi}{20} & \sin\frac{\pi}{20} \\ -\sin\frac{\pi}{20} & \cos\frac{\pi}{20} \end{pmatrix} \left[\boldsymbol{x} - \begin{pmatrix} 0.45 \\ 0.45 \end{pmatrix} \right] + \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}$$

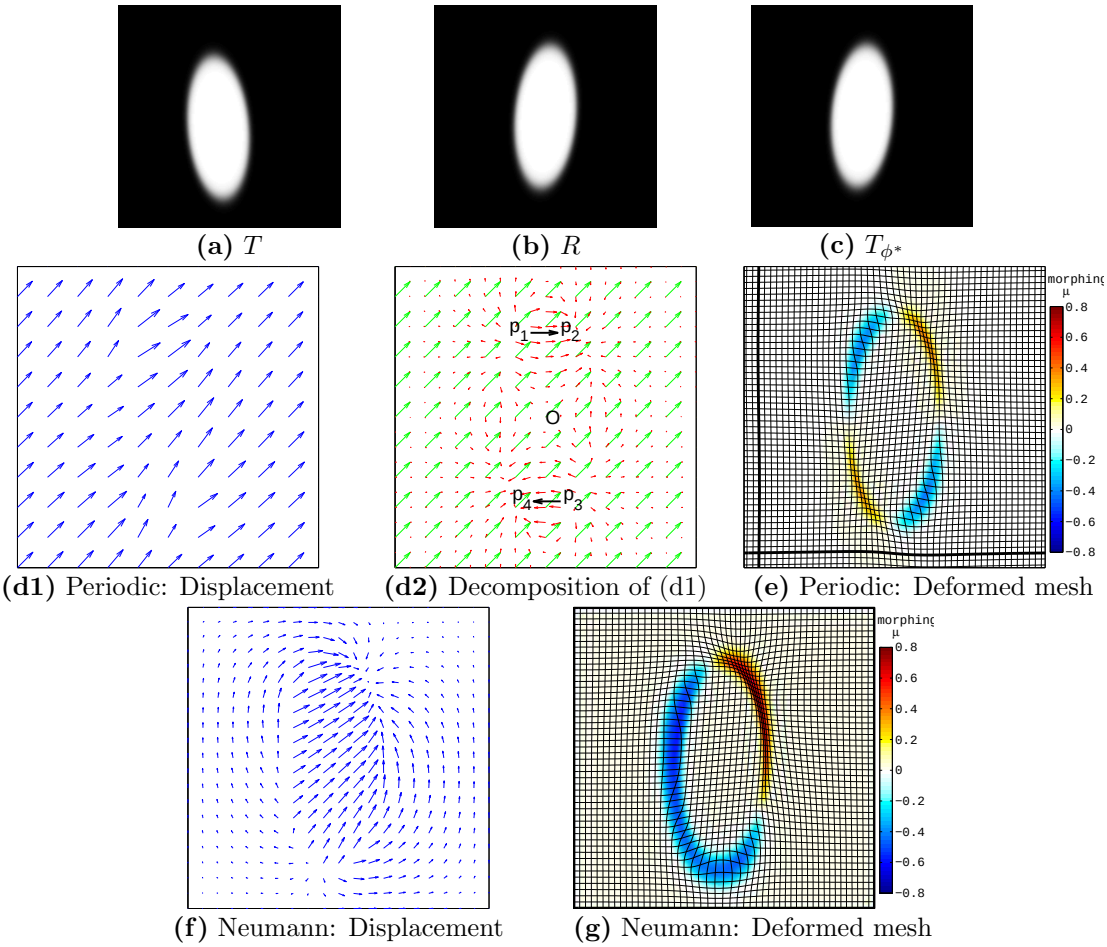


(a) T **(b)** R **(c)** T_{ϕ^*}

(d1) Periodic: Displacement **(d2)** Decomposition of (d1) **(e)** Periodic: Deformed mesh

(f) Neumann: Displacement **(g)** Neumann: Deformed mesh

Figure 5.4: Example 5.3: **(a)** Template image T. **(b)** Reference image R, where T and R are related by a combination of a translation and a dilation. **(c)** Transformed image T_{ϕ^*} under the *periodic boundary condition*. **(d1)** Displacement of pixels from T to T_{ϕ^*} under the *periodic boundary condition*. **(d2)** Decomposition of the displacement into a combination of the translation component (green) and the dilation component (red). **(e)** A deformed mesh obtained by applying the transformation $(\phi^*)^{-1}$ on a square mesh. $(\phi^*)^{-1}$ is computed under the *periodic boundary condition*. The thick black lines show where the boundary of $\Omega = [0,1] \times [0,1]$ is moved to under $(\phi^*)^{-1}$. The color bar is the morphing magnitude μ. The intensity of the color shows the degree of morphing effect under $(\phi^*)^{-1}$. **(f)** Displacement of pixels from T to T_{ϕ^*} under the *Neumann boundary condition*. **(g)** A deformed mesh obtained by applying the transformation $(\phi^*)^{-1}$ on a square mesh. $(\phi^*)^{-1}$ is computed under the *Neumann boundary condition*.



(a) T **(b)** R **(c)** T_{ϕ^*}

(d1) Periodic: Displacement **(d2)** Decomposition of (d1) **(e)** Periodic: Deformed mesh

(f) Neumann: Displacement **(g)** Neumann: Deformed mesh

Figure 5.5: Example 5.4: **(a)** Template image T. **(b)** Reference image R, where T and R are related by a combination of a translation and a rotation. **(c)** Transformed image T_{ϕ^*} under the *periodic boundary condition*. **(d1)** Displacement of pixels from T to T_{ϕ^*} under the *periodic boundary condition*. **(d2)** Decomposition of the displacement into a combination of the translation component (green) and the local rotation component (red). **(e)** A deformed mesh obtained by applying the transformation $(\phi^*)^{-1}$ on a square mesh. $(\phi^*)^{-1}$ is computed under the *periodic boundary condition*. The thick black lines show where the boundary of $\Omega = [0,1] \times [0,1]$ is moved to under $(\phi^*)^{-1}$. The color bar is the morphing magnitude μ. The intensity of the color shows the degree of morphing effect under $(\phi^*)^{-1}$. **(f)** Displacement of pixels from T to T_{ϕ^*} under the *Neumann boundary condition*. **(g)** A deformed mesh obtained by applying the transformation $(\phi^*)^{-1}$ on a square mesh. $(\phi^*)^{-1}$ is computed under the *Neumann boundary condition*.

is a combination of a translation and a *global rotation*.

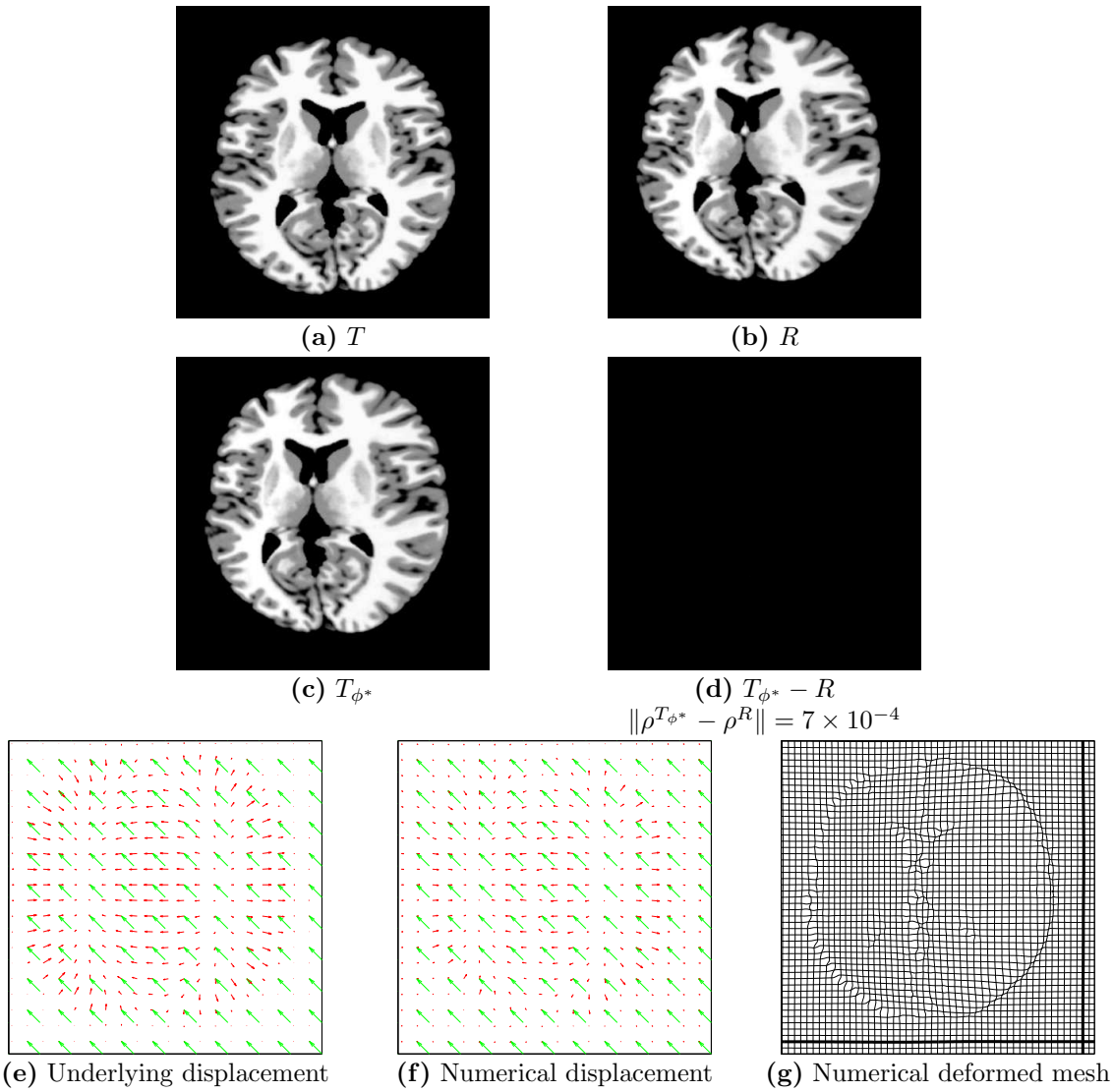
Start again with the *periodic boundary condition*. Figure 5.5(d2) decomposes the resulting transformation into translation and non-rigid deformation components. The non-rigid deformation (red arrows) is a *local rotation*. More explicitly, these red arrows rotate the pixels from p_1 to p_2 and from p_3 to p_4 around the center of rotation O. We note that the non-rigid deformation component is not a global rotation. More precisely, rotation does not occur in the black background area. Since the mass accumulates at p_2 and p_4, the intensity will further increase after the pixels are moved toward them. Conversely, the intensity at p_1 and p_3 will further decrease after the pixels are moved away from them. The color in Figure 5.5(e) illustrates this morphing effect.

As a comparison, for the *Neumann boundary condition*, Figures 5.5(f) and 5.5(g) do not reflect the underlying combination of a translation and a rotation. Once again, the morphing effect under the periodic boundary condition is much less severe than the Neumann boundary condition.

5.4.3 Mass transport registration versus empirical two-step registration

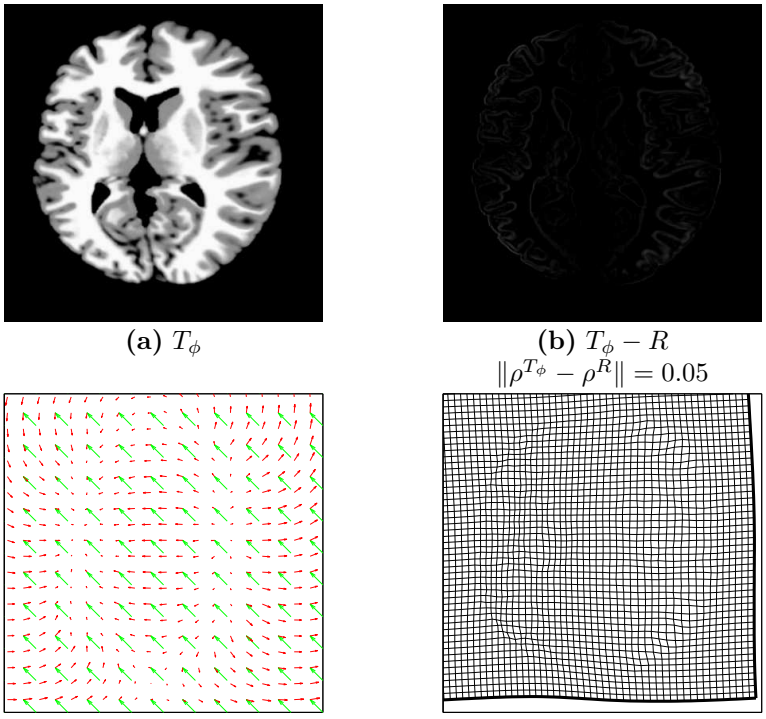
In this section, we apply mass transport image registration on medical images and compare it with the empirical two-step registration, which consists of a pre-registration (rigid transformation) followed by an elastic registration.

Example 5.5 (Figure 5.6)**.** To evaluate the performance of our approach, we perform the following test: We first specify a certain underlying transformation ϕ^*_{true}; see Figure 5.6(e). Then we apply this underlying transformation to an image T (Figure 5.6(a)), which generates the reference image R (Figure 5.6(b)). Once obtaining T and R, we register these two images using mass transport registration with the periodic boundary condition. We expect the numerical scheme to produce a transformed image T_{ϕ^*} that equals R and recover the pre-specified underlying transformation. As expected, the resulting transformed image T_{ϕ^*} almost equals R, where the magnitude of the error is 10^{-4}; see Figure 5.6(c)-(d). We remark that the error is not exactly zero since we let the Levenberg-Marquardt algorithm (Algorithm 5.1) stop at a tolerance of 10^{-4}. Moreover, the resulting transformation given by our numerical scheme, as shown in Figure 5.6(f), is approximately the same as the pre-specified underlying transformation. Figure 5.6(g) visualizes the resulting transformation again on a deformed mesh, which shows the compression and expansion at the right and left semi-circles due to the mass inflow and outflow, respectively. We note that some mesh



(a) T **(b)** R **(c)** T_{ϕ^*} **(d)** $T_{\phi^*} - R$ $\|\rho^{T_{\phi^*}} - \rho^R\| = 7 \times 10^{-4}$ **(e)** Underlying displacement **(f)** Numerical displacement **(g)** Numerical deformed mesh

Figure 5.6: Example 5.5: mass transport registration under *periodic boundary condition*. **(a)** Template image T. **(b)** Reference image R. **(c)** Transformed image T_{ϕ^*}. **(d)** Difference between the transformed image T_{ϕ^*} and the reference R. **(e)** Pre-specified underlying transformation between T and R. **(f)** Transformation given by the numerical scheme, which is a good approximation to the pre-specified underlying transformation in (e). **(g)** A deformed mesh obtained by applying the transformation $(\phi^*)^{-1}$ on a square mesh.



(a) T_ϕ **(b)** $T_\phi - R$ $\|\rho^{T_\phi} - \rho^R\| = 0.05$ **(c)** Numerical displacement **(d)** Deformed mesh

Figure 5.7: Example 5.5: empirical two-step registration implemented by the FAIR package [121], where T and R are the same as Figure 5.6(a)-(b). **(a)** Transformed image T_ϕ. **(b)** Difference between the transformed image T_ϕ and the reference R. **(c)** Transformation given by the empirical approach, consisting of a rigid pre-registration (green arrows) and a non-rigid elastic deformation (red arrows). **(d)** A deformed mesh obtained by applying the transformation ϕ^{-1} on a square mesh.

lines appear squiggly due to the limited numerical resolution. However, the deformed mesh is overall diffeomorphic and does not have foldings.

We also register T and R using the empirical two-step registration. The experiment is implemented by the FAIR package [121]. The results are shown in Figure 5.7. The error between the transformed image T_ϕ and the reference image R by the empirical approach (Figure 5.7(b)) is 0.05, which is larger than the mass transport error. The transformation computed by the empirical approach (Figure 5.7(c)) appears slightly different from the underlying transformation (Figure 5.6(e)), especially for the non-rigid component. One possible explanation is that the elastic registration used by the empirical approach may be too restrictive to recover large localized non-rigid deformations; conversely, the mass

transport registration is generally able to capture large localized non-rigid deformations [123].

Compared to the empirical approach, our approach does not require a two-step process. In addition, our approach makes the non-rigid model more universal (capable of handling both translation and non-rigid deformation) than the individual method employed in each step of the empirical approach.

5.4.4 Quantitative evaluation of the transformations

To quantitatively measure the accuracy of the resulting transformations, we compute

$$\|\phi^*(\boldsymbol{x}) - \phi^*_{true}(\boldsymbol{x})\|_{L_2(\Omega)}, \tag{5.29}$$

which is the difference between the computed transformation ϕ^* and the true underlying transformation ϕ^*_{true} in the L_2 norm. We note that the evaluation of (5.29) requires a prior knowledge of the true underlying transformation, which is unavailable in practice. However, in Examples 5.2–5.5, we pre-specify the underlying transformations between T and R, which allows us to perform such an evaluation. Table 5.2 reports the errors of the transformations (5.29). In each example, the error given by the mass transport registration under the periodic boundary condition is compared against the error either under the Neumann boundary condition or by the two-step empirical registration. Table 5.2 shows that the errors of the mass transport registration under the periodic boundary condition are smaller. We note that the error for Example 5.4 is relatively large even under the periodic boundary condition, since the mass transport model only recovers rotation locally rather than globally.

5.4.5 More examples

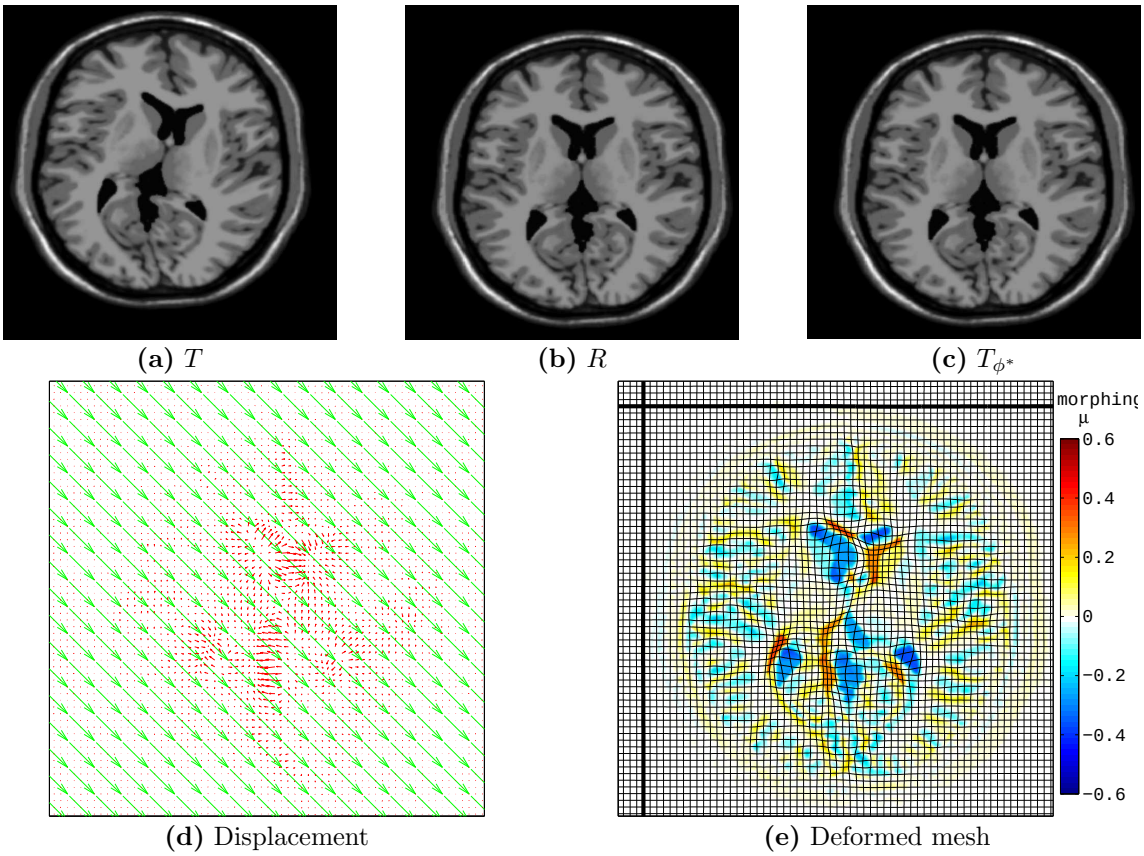
Example 5.6 (Figure 5.8)**.** We register two images taken from a brain of a patient under the *periodic boundary condition*. The transformed image T_{ϕ^*} (Figure 5.8(c)) is the same as

Example 5.2	Example 5.3	Example 5.4	Example 5.5
Periodic: 0	**Periodic:** 0.0053	**Periodic:** 0.066	**Mass transport, periodic:** 0.0055
Neumann: 0.056	Neumann: 0.056	Neumann: 0.088	Two-step empirical: 0.011

Table 5.2: The errors of the motion fields (5.29) for Examples 5.2–5.5.

the reference image R (Figure 5.8(b)). Regarding the resulting transformation, the numerical scheme can automatically translate T toward R, followed by a non-rigid deformation; see Figures 5.8(d)-(e). The two figures also indicate that around the pixels where T and R look different, the non-rigid deformation and the morphing effect are relatively large.

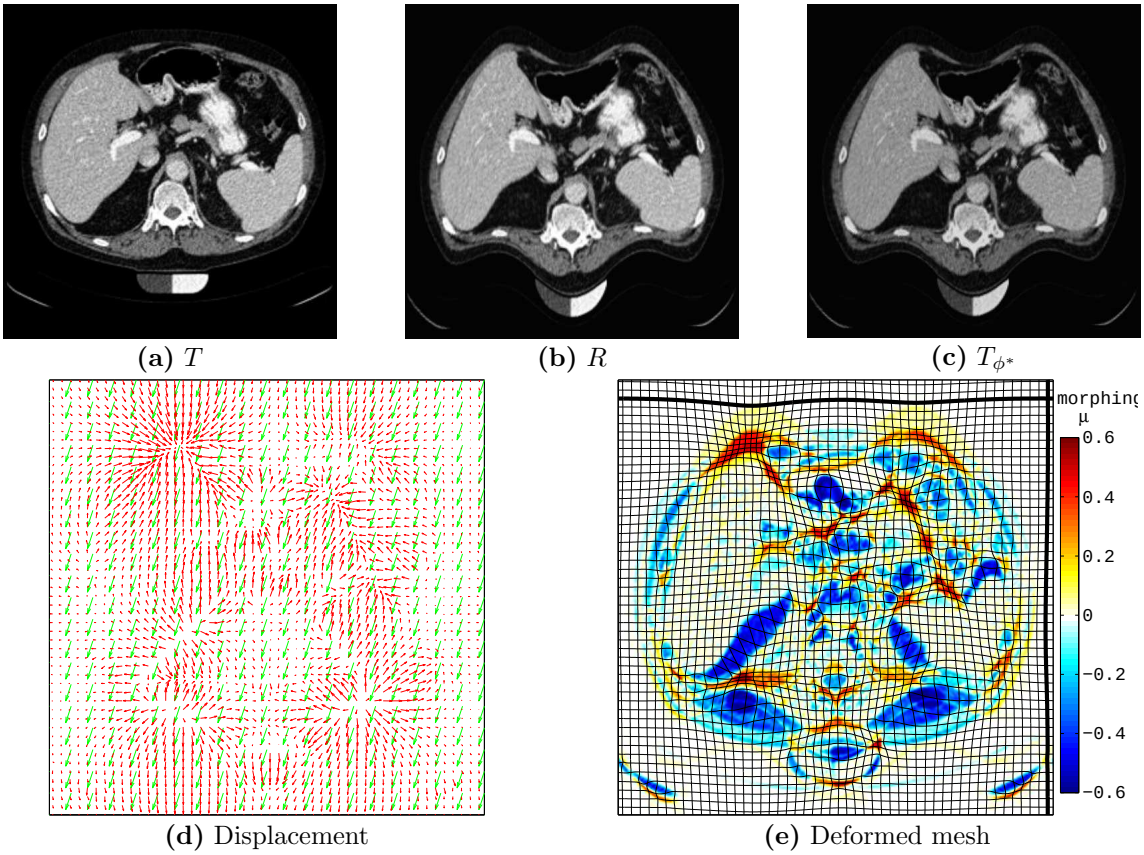
Example 5.7 (Figure 5.9)**.** We register two distinct medical images under the *periodic boundary condition*. The registration between them requires a large non-rigid deformation and a large morphing effect. This can be seen in Figures 5.9(d)-(e). We note that under the



(a) T **(b)** R **(c)** T_{ϕ^*}

(d) Displacement **(e)** Deformed mesh

Figure 5.8: Example 5.6: medical image registration under the *periodic boundary condition*. **(a)** Template image T. **(b)** Reference image R. **(c)** Transformed image T_{ϕ^*}. **(d)** Decomposition of the displacement into a combination of the translation component (green) and the non-rigid deformation component (red). **(e)** A deformed mesh obtained by applying the transformation $(\phi^*)^{-1}$ on a square mesh.



(a) T **(b)** R **(c)** T_{ϕ^*}

(d) Displacement **(e)** Deformed mesh

Figure 5.9: Example 5.7: medical image registration under the *periodic boundary condition*. **(a)** Template image T. **(b)** Reference image R. **(c)** Transformed image T_{ϕ^*}. **(d)** Decomposition of the displacement into a combination of the translation component (green) and the non-rigid deformation component (red). **(e)** A deformed mesh obtained by applying the transformation $(\phi^*)^{-1}$ on a square mesh.

transformation, the boundary lines of the domain, marked by the thick black lines in Figure 5.9(e), are not only translated, but also curved. Such curvature allows the transformation between the actual objects inside R and T free from the influence of the dark background. This is different from the Neumann boundary condition, where the transformation between the actual objects is affected by the transformation imposed artificially on the boundary.

5.5 Conclusion

In this chapter, we propose a numerical scheme together with a boundary condition for the mass transport registration model. The main contributions include the following:

- Our periodic boundary condition incorporates both translation and non-rigid deformation into the mass transport model. We note that the standard mass transport model using the conventional boundary conditions can only do non-rigid registration, whereas our method can also provide the translation component as a by-product.
- Our mixed finite difference discretization gives a convex solution and hence an optimal transformation for the mass transport image registration.
- To handle singular Jacobians, we propose a modified Levenberg-Marquardt algorithm where translation kernels are explicitly corrected.

In the numerical results, we first show that our numerical scheme yields the optimal transformation, whereas a non-monotone finite difference scheme gives a non-optimal transformation. In addition, the deformed mesh images across all the examples (i.e., Figures 5.3(e), 5.4(e), 5.5(e), 5.6(g), 5.8(e) and 5.9(e)) are smooth and do not contain foldings or crossings, which provides evidences that the computed transformations are optimal (see the discussion in Section 5.2.2). We also show that under the periodic boundary condition, ϕ^* recovers the underlying combination of a translation and a non-rigid deformation. The periodic boundary condition outperforms the Neumann boundary condition.

Regarding the efficiency of the numerical scheme, we attach the computational cost of the simulations in Appendix A.6. It would be desirable to speed up the computation by multilevel approaches [150, 121] or multigrid algorithms [153], which we leave as future work.

Chapter 6

Deep Neural Network Framework for HJB Equations Arising from American Option Problems

6.1 Introduction

In Chapters 2-5, we have investigated HJB equations where the spatial dimension is less than three. In this chapter, we study HJB equations where the spatial dimension is higher than three (e.g., as high as 200). Such high-dimensional HJB equations appear in the application of **American options**, where the spatial coordinate $\boldsymbol{x} \in \mathbb{R}^d$ contains the prices of the underlying assets, the dimension d is the number of the underlying assets, and the solution $u(\boldsymbol{x}, t)$ is an **American option price**[1]. In practical applications of hedging, we are required to compute not only an American option price $u(\boldsymbol{x}, t)$, but also the derivatives of the price with respect to the underlying asset prices $\nabla u(\boldsymbol{x}, t) \equiv \left(\frac{\partial u}{\partial x_1}(\boldsymbol{x}, t), \cdots \frac{\partial u}{\partial x_d}(\boldsymbol{x}, t)\right)^T$, called **American option delta** [97]. In addition, hedging requires computing their values on the entire spacetime, not only at a particular point $(\boldsymbol{x}, t)$; see [97, 90, 100] for explanations and concrete examples. Hence, our objective is to solve for both the price $u(\boldsymbol{x}, t)$ and the delta $\nabla u(\boldsymbol{x}, t)$ on the entire spacetime accurately.

As mentioned in Section 1.2.2, in addition to HJB equations, American option problems have other various formulations [58, 139, 67, 66, 131]. Numerous approaches have been

[1] Unless otherwise specified, "price" in this chapter refers to American option price, rather than underlying asset price.

proposed for solving American option problems under all these different formulations; see [97, 7, 58, 67, 131, 117, 154, 103, 35, 38, 132, 113] for typical approaches. When the dimension d is greater than 3, numerical solution of HJB equations (or other equivalent PDEs) using conventional discretization (e.g., finite difference, finite element) becomes infeasible, as its complexity grows exponentially with the dimension. Approaches that mitigate (although do not address) the curse of dimensionality issue include regression-based methods [117, 154, 103], stochastic mesh methods [35], sparse grids methods [38, 132, 113], etc. In particular, when the dimension d is moderate (e.g., $d \leq 20$), the regression-based **Longstaff-Schwartz method** [117] based on a Monte Carlo formulation is widely considered as the state-of-the-art approach for computing option prices. In addition, one can combine the Longstaff-Schwartz method with the methods proposed in [33, 26, 151] to compute corresponding option deltas. We note that these approaches only compute option prices and deltas at a given point (e.g., $t = 0$)[2]. However, we emphasize that a complete hedging process requires computing prices and deltas on the entire spacetime. Furthermore, for the Longstaff-Schwartz method, a set of χ-th degree polynomials is normally used as the basis for regression, which leads to χ-th degree complexity (rather than exponential complexity). However, χ is required to go to infinity for convergence [117, 145], which still results in a high complexity.

In this chapter, we propose a **deep neural network** framework for solving high-dimensional American option problems. The major contributions of the proposed neural network framework are summarized as follows:

- We convert the HJB equation into an equivalent **backward stochastic differential equation (BSDE)**. Furthermore, we introduce the least squares residual of the BSDE as the loss function of neural networks. BSDE couples prices and deltas in one single equation, and thus evaluates both prices and deltas accurately. Moreover, as discussed in Section 1.3.4, constructing Hessian tensors for an HJB equation is expensive in both computational time and memory. Unlike the HJB equation, the equivalent BSDE does not contain a Hessian, and thus gives rise to a less expensive neural network formulation.

- Our neural network architecture is new. Assuming that there are N discrete timesteps,

[2] Although one may consider using the Longstaff-Schwartz regressed values as an estimate of the spacetime prices, Figure 1 in [26] shows that using such regressed values as the spacetime solution is inaccurate. Alternatively, one may consider applying the Longstaff-Schwartz method repeatedly on all the spacetime points, where *every* point requires $M \to \infty$ samples. However, this is expensive.

we design a sequence of N recursively-defined feedforward neural networks[3], where each network extracts the difference between the price functions of adjacent timesteps.

- Our neural network formulation utilizes domain knowledge of American options, including smoothing the payoff at $t = T$, adding the payoff and the previous continuation price as features, etc.

- Our proposed approach can evaluate both option prices and option deltas on the entire spacetime, not only at a given point $(\boldsymbol{x}, t)$.

- The computational cost of the proposed neural network framework grows quadratically with the dimension d, in contrast to exponential growth as in the Longstaff-Schwartz method. In particular, our approach outperforms the Longstaff-Schwartz method when $d \geq 20$, in the sense that our proposed approach solves American option prices and deltas in as high as 200 dimension, while the Longstaff-Schwartz method fails to solve the problems due to the out-of-memory error and the worse-than-quadratic cost.

We note that our proposed approach is not the only neural network framework for American option problems. Early research of neural networks in American options can be found in [104, 89]. They consider using one-hidden-layer feedforward neural networks for option pricing. However, the highest dimension considered in their numerical simulations is 10. Very recently, deep neural network approaches were proposed in [142, 59, 16, 88, 75, 129, 17]. They suggest that increasing the depth of neural networks is important in pushing the solutions to higher dimensions. Similar to these approaches, our proposed framework is also a deep neural network approach. However, we emphasize that there are a few key differences between our proposed approach and the other deep neural network approaches.

- *Different computed quantities:* Our approach computes American option prices and deltas on the entire spacetime. The approach in [142] computes prices but not deltas. The approaches in [59, 16, 88, 129] only consider European option prices, noting that European options are easier to price than American options. Although [75, 17] extends their methods to American options, the authors only compute the price at a given point. In particular, we emphasize that only our approach discusses and simulates hedging options, which is beyond merely pricing options.

[3]Here the proposed "recursively-defined" feedforward network is not the same as the Recurrent Neural Network (RNN) in the literature, which will be explained in Section 6.4.1.

- *Different network architectures:* Our network architecture is a chain of recursively-defined networks that learn the difference of the price functions between adjacent timesteps; the approach in [142] uses a long short-term neural network that learns the price function itself; the approaches in [59, 16, 88, 75] consider a chain of isolated, independent feedforward networks.

- *Different loss functions:* The approach in [142] defines the loss function by the residual of the HJB equation. It involves computing the Hessian of the output price function, which is expensive and difficult to implement. Our framework uses the residual of one single BSDE as the loss function, which avoids computing the Hessian. The approaches in [59, 16, 88, 75] involve the integral form of multiple BSDEs, which is redundant for option pricing. In addition, their BSDEs are not used as loss functions.

The chapter is organized as follows. Section 6.2 introduces the American option problems and the corresponding HJB formulation. Section 6.3 introduces the BSDE formation and the least squares residual loss function. Section 6.4 describes the architecture and components of the proposed neural network model. Section 6.5 discusses the techniques that improve the performance of the framework. Section 6.6 analyzes the computational cost. In Section 6.7, we present numerical solutions of option prices and deltas to illustrate the advantage of our deep neural network framework. Section 6.8 concludes the chapter.

6.2 American Options

6.2.1 American options

Section 1.2.2 introduced American options. In this chapter, we first review the mathematical formulation of American option problems. Suppose an American option contains a basket of d underlying assets (such as stocks, commodities, foreign currencies). Let $\boldsymbol{X} = (X_1, \cdots, X_d)^T \in \mathbb{R}^d$ be the prices of the underlying assets. Note that $\boldsymbol{X}$ is a random variable. In order to distinguish random and deterministic variables, we will use capital and lowercase letters respectively. Let $t \in [0, T]$ be the time up to the expiry date T of the option contract. Let r be the interest rate. Let δ_i and σ_i $(i = 1, \cdots, d)$ be the dividend and volatility of each underlying asset. Let $\rho \in \mathbb{R}^{d\times d}$ be a correlation matrix where all the diagonal entries are 1. Define d correlated random variables

$$dW_i(t) = \sum_{j=1}^{d} L_{ij}\phi_j(t)\sqrt{dt}, \tag{6.1}$$

where $\phi_i(t) \sim \mathcal{N}(0,1)$ are independent standard normal random variables, and L is the Cholesky factorization of the correlation matrix, i.e., $\rho = LL^T$. Given an initial state $\boldsymbol{x}^0 \in \mathbb{R}^d$, the prices of the underlying assets $\boldsymbol{X}$ evolve under the following stochastic differential equations (SDEs):

$$\begin{aligned} &dX_i(t) = (r-\delta_i)X_i(t)dt + \sigma_i X_i(t)dW_i(t), \quad i = 1, \cdots, d, \\ &\boldsymbol{X}(0) = \boldsymbol{x}^0. \end{aligned} \tag{6.2}$$

Let $f(\boldsymbol{x})$ be the payoff function of the option at the state $\boldsymbol{x}$, which usually takes the form of

$$f(\boldsymbol{x}) = \max(g(\boldsymbol{x}), 0). \tag{6.3}$$

For instance, the commonly-seen "max call options" have the payoff function of

$$f(\boldsymbol{x}) = \max\left(\max_{i=1,\ldots,d}(x_i) - K,\, 0\right), \tag{6.4}$$

where K is the strike price (i.e., the preset price at which an option holder can buy/sell underlying assets). Let $q(\boldsymbol{x}, t)$ be the "continuation price" of an American option, i.e., the discounted option payoff provided that the option is not exercised at time t and state $\boldsymbol{x}$:

$$q(\boldsymbol{x}, t) = \max_{\tau \in [t,T]} \mathbb{E}\left[\, e^{-r(\tau - t)} f(\boldsymbol{X}(\tau)) \,\middle|\, \boldsymbol{X}(t) = \boldsymbol{x} \,\right], \tag{6.5}$$

where τ is the stopping time. Then the American option price $u(\boldsymbol{x}, t)$ is defined as

$$\begin{aligned} u(\boldsymbol{x}, t) &= \max\left[q(\boldsymbol{x}, t), f(\boldsymbol{x})\right] \\ &= \begin{cases} q(\boldsymbol{x}, t), & \text{if } q(\boldsymbol{x}, t) > f(\boldsymbol{x}), \text{ i.e., the option is continued at } (\boldsymbol{x}, t), \\ f(\boldsymbol{x}), & \text{if } q(\boldsymbol{x}, t) \le f(\boldsymbol{x}), \text{ i.e., the option is exercised at } (\boldsymbol{x}, t). \end{cases} \end{aligned} \tag{6.6}$$

6.2.2 HJB formulation

An equivalent formulation of the American option problem (6.2)-(6.6) is the HJB formulation [131, 67, 66]. Introduce the differential operator arising from the famous Black-Scholes model [97]:

$$\mathcal{L}\, u(\boldsymbol{x}, t) \equiv -\frac{\partial u}{\partial t}(\boldsymbol{x}, t) - \frac{1}{2}\sum_{i,j=1}^{d} \sigma_i \sigma_j \rho_{ij} x_i x_j \frac{\partial^2 u}{\partial x_i \partial x_j}(\boldsymbol{x}, t) - \sum_{i=1}^{d} (r - \delta_i) x_i \frac{\partial u}{\partial x_i}(\boldsymbol{x}, t) + r u(\boldsymbol{x}, t). \tag{6.7}$$

Then the HJB equation for the American option problem reads:

$$\min\left(\mathcal{L}\,u(\boldsymbol{x},t),\ u(\boldsymbol{x},t)-f(\boldsymbol{x})\right)=0, \qquad \text{in } \Omega\times[0,T); \tag{6.8}$$

$$u(\boldsymbol{x},T)=f(\boldsymbol{x}), \qquad \text{in } \Omega. \tag{6.9}$$

Similar to (2.1), the HJB equation (6.7)-(6.9) is backward. We note that (6.8) can be written in an equivalent form that is consistent with the rest of the thesis:

$$(1-c^*(\boldsymbol{x},t))\ \mathcal{L}\,u(\boldsymbol{x},t)+c^*(\boldsymbol{x},t)\,(u(\boldsymbol{x},t)-f(\boldsymbol{x}))=0, \tag{6.10}$$

$$\text{subject to } c^*(\boldsymbol{x},t)\equiv \underset{c(\boldsymbol{x},t)\in\{0,1\}}{\arg\min}\ \{(1-c(\boldsymbol{x},t))\ \mathcal{L}\,u(\boldsymbol{x},t)+c(\boldsymbol{x},t)\,(u(\boldsymbol{x},t)-f(\boldsymbol{x}))\}\,. \tag{6.11}$$

Unlike the previous chapters where the control set is continuous, here the control set $\{0,1\}$ is discrete, and more precisely, binary. Indeed, the control $c(\boldsymbol{x},t)=1$ (or 0) means that an option holder decides to exercise (or not to exercise) the American option at the state $(\boldsymbol{x},t)$. Correspondingly, the optimal control $c^*(\boldsymbol{x},t)$ means the best decision at $(\boldsymbol{x},t)$ that will maximize the option holder's gain.

When the dimension is less than 3, computing the solution of the HJB formulation using conventional discretization yields as accurate as $O(h^2)$ prices and deltas [131, 67, 66]. However, when the dimension is high, such as 200, conventional discretization is infeasible. As introduced in Section 1.3.4, one may consider using a neural network formulation to solve high dimensional HJB equations, such as the approach in [142]. Since the HJB equation (6.7)-(6.9) contains the Hessian $D^2u(\boldsymbol{x},t)$, i.e., $\{\frac{\partial^2 u}{\partial x_i\partial x_j}\,|\,\forall i,j\}$, the corresponding neural network approach involves computing Hessian tensors. Unfortunately, a Hessian tensor is an $O(Md^2)$ tensor, where M is the number of samples for a neural network. When d is high, a Hessian tensor can be expensive to compute and store. In addition, given a neural network, the automatic differentiation of a Hessian is nearly impossible to derive, which makes it difficult to implement using existing deep learning libraries.

6.3 Backward Stochastic Differential Equation (BSDE) Formulation

6.3.1 BSDE formulation

Instead of solving the HJB formulation (6.7)-(6.9), our approach is to first convert the HJB formulation into an equivalent BSDE using the following theorem:

Theorem 6.1 (BSDE formulation)**.** A linearized HJB equation where the option is not exercised at the time t, i.e.,

$$\mathcal{L}\, q(\boldsymbol{x}, t) = 0, \tag{6.12}$$

is equivalent to the following BSDE:

$$dq(\boldsymbol{X}, t) = rq(\boldsymbol{X}, t)dt + \sum_{i=1}^{d} \sigma_i X_i(t) \frac{\partial q}{\partial x_i}(\boldsymbol{X}, t) dW_i(t), \tag{6.13}$$

where $\boldsymbol{X}$ satisfies the SDE (6.2), and r, σ_i and $dW_i(t)$ are the same as in (6.2).

Proof. We refer interested readers to the proof in [60, 113], which uses Ito's lemma. □

We note that, although the BSDE formulation (6.13) only applies to the linearized HJB equation, we will soon discuss how to incorporate the nonlinearity and the terminal condition into the formulation. The significance of the BSDE formulation (6.13) is two-fold. One is that it correlates the price $q(\boldsymbol{x}, t)$ with the delta $\nabla q(\boldsymbol{x}, t)$. If the price is solved correctly, then (6.13) simultaneously yields the correct delta. A simultaneously correct evaluation of the price and the delta is essential for performing a complete hedging process. The other significance is that, unlike the HJB formulation, the BSDE formulation (6.13) does not contain the Hessian, which avoids the computation and storage of Hessian tensors. Instead, it only requires computing price tensors of size $O(M)$ and delta tensors of size $O(Md)$. In addition, delta tensors can be easily evaluated by the built-in automatic differentiation of Tensorflow [1], i.e., "tf.gradients".

In this chapter, we use an Euler timestepping Monte Carlo method to simulate the SDEs (6.2) and the BSDE (6.13). Let $m = 1, \cdots, M$ be the indices of simulation paths, $n = 0, \cdots, N$ be the indices of discrete timesteps from 0 to T, $\Delta t = \frac{T}{N}$, $t^n = n\Delta t$ be the timesteps, and $(\Delta W_i)_m^n = \sum_{j=1}^{d} L_{ij} (\phi_j)_m^n \sqrt{\Delta t}$. We discretize (6.2) as

$$(X_i)_m^0 = x_i^0, \qquad m = 1, ..., M, \quad i = 1, ..., d; \tag{6.14}$$

$$\begin{aligned}(X_i)_m^{n+1} = (1 + (r - \delta_i)\Delta t)(X_i)_m^n + \sigma_i (X_i)_m^n (\Delta W_i)_m^n, \\ m = 1, ..., M, \quad i = 1, ..., d, \quad n = 0, ..., N - 1.\end{aligned} \tag{6.15}$$

We also discretize (6.13) as

$$\begin{aligned}q(\boldsymbol{X}_m^{n+1}, t^{n+1}) = (1 + r\Delta t) q(\boldsymbol{X}_m^n, t^n) + \sum_{i=1}^{d} \sigma_i (X_i)_m^n \frac{\partial q}{\partial x_i}(\boldsymbol{X}_m^n, t^n)(\Delta W_i)_m^n, \\ m = 1, ..., M, \quad n = N - 1, ..., 0.\end{aligned} \tag{6.16}$$

Next we incorporate the nonlinearity of the HJB equation into the BSDE formulation. More specifically, if we allow the option to be exercised at any time after t^n, then we can replace $q(\boldsymbol{X}_m^{n+1}, t^{n+1})$ on the left hand side of (6.16) by $u(\boldsymbol{X}_m^{n+1}, t^{n+1}) = \max\left[q(\boldsymbol{X}_m^{n+1}, t^{n+1}), f(\boldsymbol{X}_m^{n+1})\right]$. This incorporation of the nonlinearity is essentially the "implicit timestepping using an explicit evaluation of the American constraint" described in [66]. In addition, we add the terminal condition (6.9) into the discretization. This yields a complete discretized system for the BSDE:

$$u(\boldsymbol{X}_m^N, t^N) = f(\boldsymbol{X}_m^N), \qquad m = 1, ..., M. \tag{6.17}$$

$$\text{Solve } (1 + r\Delta t)q(\boldsymbol{X}_m^n, t^n) + \sum_{i=1}^{d} \sigma_i(X_i)_m^n \frac{\partial q}{\partial x_i}(\boldsymbol{X}_m^n, t^n)(\Delta W_i)_m^n = u(\boldsymbol{X}_m^{n+1}, t^{n+1}) \quad \text{for } q(\boldsymbol{X}_m^n, t^n), \tag{6.18}$$

$$\text{and then compute } u(\boldsymbol{X}_m^n, t^n) = \max\left[q(\boldsymbol{X}_m^n, t^n), f(\boldsymbol{X}_m^n)\right], \quad m = 1, ..., M, \quad n = N-1, ..., 0. \tag{6.19}$$

To sketch the idea of solving the discretized BSDE, let (6.14)-(6.15) generate samples of underlying asset prices $\{\boldsymbol{X}_m^n\}$ for all n's and m's. Then one starts with $n = N$, computes the terminal condition (6.17), and then performs backward timestepping from $n = N-1$ to $n = 0$ using (6.18)-(6.19), which yields $\{u(\boldsymbol{X}_m^n, t^n)\}$ for all n's and m's. Eventually, at $n = 0$, noting that $\boldsymbol{X}_m^0 = \boldsymbol{x}^0$ for all m's by (6.14), we obtain the option price $u(\boldsymbol{x}^0, 0)$ and the option delta $\nabla u(\boldsymbol{x}^0, 0)$.

6.3.2 Least squares solution for the discretized BSDE

Consider only the n-th timestep t^n, and introduce a short notation for the corresponding price and delta functions as $u^n(\boldsymbol{x}) \equiv u(\boldsymbol{x}, t^n)$ and $\nabla u^n(\boldsymbol{x}) \equiv \nabla u(\boldsymbol{x}, t^n)$. Solving (6.18) requires finding a d-dimensional function $q^n(\boldsymbol{x})$ where both the function $q^n(\boldsymbol{x})$ itself and its derivative $\nabla q^n(\boldsymbol{x})$ satisfy (6.18), which is challenging.

In this chapter, we consider finding an approximation of the continuous price function. We let the approximation satisfy (6.18) in a least squares sense. More specifically, define the residual of (6.18) as the difference between the left and right hand sides:

$$\mathcal{R}[q^n]_m \equiv (1 + r\Delta t)q^n(\boldsymbol{X}_m^n) + \sum_{i=1}^{d} \sigma_i(X_i)_m^n \frac{\partial q^n}{\partial x_i}(\boldsymbol{X}_m^n)(\Delta W_i)_m^n - u^{n+1}(\boldsymbol{X}_m^{n+1}), \quad m = 1, \cdots, M. \tag{6.20}$$

Then our goal is to find an approximation y^n to the actual continuation function q^n which minimizes the least squares residual:

$$q^n \approx (y^n)^* \equiv \arg\min_{y^n} \left(\sum_{m=1}^{M} \mathcal{R}[y^n]_m^2 \right). \tag{6.21}$$

6.4 Neural Network Formulation

Finding the optimal approximate function in the least squares sense (6.21) is non-trivial. One approach is to use a parameterized function to represent the approximate function y^n. Then the optimization problem in terms of function space is converted to the optimization problem in terms of parameter space, which is more manageable.

6.4.1 Sequence of neural networks

Our approach is to use neural networks to represent the approximate continuation price function y^n. As introduced in Section 1.3.4, a neural network is a nonlinear parameterization where the basis is dynamic, i.e., the optimal basis is found during the optimization process, or in the language of deep learning community, the optimal basis is learned during the training process [80]. The main advantage of neural network formulation is that the complexity does not grow exponentially with the dimension d.

There exist many neural network architectures, such as feedforward, convolutional, or recurrent networks. We refer interested readers to [80] for a review of these standard network architectures. However, these standard networks are not designed for solving American option problems.

In this chapter, we propose a sequence of N networks $\{y^n(\boldsymbol{x};\Omega^n) \,|\, n = N-1, \cdots, 1, 0\}$, where Ω^n is the trainable parameter set of the n-th network. Each individual network $y^n(\boldsymbol{x};\Omega^n)$ approximates the price function at the n-th timestep $q^n(\boldsymbol{x})$. The design of each individual network is motivated by the fact that the approximate function of the n-th timestep, $y^n(\boldsymbol{x};\Omega^n)$, should differ from $y^{n+1}(\boldsymbol{x};\Omega^{n+1})$ by a function of magnitude $O(\Delta t)$. Mathematically, it means that

$$y^N(\boldsymbol{x}) = f(\boldsymbol{x}), \qquad n = N; \tag{6.22}$$

$$y^n(\boldsymbol{x};\Omega^n) = y^{n+1}(\boldsymbol{x};\Omega^{n+1}) + \Delta t \cdot \mathcal{F}(\boldsymbol{x};\Omega^n), \qquad n = N-1, \cdots, 0; \tag{6.23}$$

where $\mathcal{F}(\boldsymbol{x};\Omega^n)$ is the difference between the approximate functions at the two adjacent timesteps, or the "remainder" that we aim to find. We note that the sequence of networks (6.23) is defined in a recursive sense. In addition, the sequence of networks is backward in time, i.e., the timestep n decreases from $N-1$ to 0. Hence, in this chapter, we use the "previous", "current" and "next" timesteps to refer to the $(n+1)$-th, n-th and $(n-1)$-th timesteps, respectively.

Regarding each remainder function $\mathcal{F}(\boldsymbol{x};\Omega^n)$, we parameterize it by an L-layer feedforward network with batch normalizations. In the following part, we drop the timestep index n temporarily, and use superscript with square brackets for the layer index $l = 0, \cdots, L$. Let the dimensions of the layers be $\{d^{[l]} \,|\, l = 0, \cdots, L\}$. Let the input of the neural network be $\boldsymbol{x}^{[0]} = \boldsymbol{x} \in \mathbb{R}^{d^{[0]}}$, where the input dimension is $d^{[0]} = d$. Then we construct an L-layer feedforward neural network as follows:

- For the hidden layers, $l = 1, \cdots, L$:

$$\text{linear transformation: } \boldsymbol{z}^{[l]} = \boldsymbol{W}^{[l]} \cdot \boldsymbol{x}^{[l-1]}, \tag{6.24}$$

$$\text{batch normalization: } \boldsymbol{h}^{[l]} = \text{bnorm}(\boldsymbol{z}^{[l]}; \boldsymbol{\beta}^{[l]}, \boldsymbol{\gamma}^{[l]}, \boldsymbol{\mu}^{[l]}, \boldsymbol{\sigma}^{[l]}), \tag{6.25}$$

$$\text{rectified linear unit activation: } \boldsymbol{x}^{[l]} = \max(\boldsymbol{h}^{[l]}, 0), \tag{6.26}$$

  where

$$\text{bnorm}(\boldsymbol{x}; \boldsymbol{\beta}, \boldsymbol{\gamma}, \boldsymbol{\mu}, \boldsymbol{\sigma}) \equiv \boldsymbol{\gamma} \cdot \frac{\boldsymbol{x} - \boldsymbol{\mu}}{\boldsymbol{\sigma}} + \boldsymbol{\beta} \tag{6.27}$$

  is the batch normalization operator, $\boldsymbol{x}^{[l]}, \boldsymbol{z}^{[l]}, \boldsymbol{h}^{[l]} \in \mathbb{R}^{d^{[l]}}$ are hidden layer variables, $\boldsymbol{W}^{[l]} \in \mathbb{R}^{d^{[l]} \times d^{[l-1]}}$ are trainable weights, $\boldsymbol{\mu}^{[l]}, \boldsymbol{\sigma}^{[l]} \in \mathbb{R}^{d^{[l]}}$ are moving averages of batch means and standard deviations, and $\boldsymbol{\gamma}^{[l]}, \boldsymbol{\beta}^{[l]} \in \mathbb{R}^{d^{[l]}}$ are trainable scales and offsets. The operations in (6.25)-(6.27) are evaluated element-wise. For instance, (6.26) means $x_i^{[l]} = \max(h_i^{[l]}, 0)$ for all $i = 1, \cdots, d^{[l]}$.

- For the output layer:

$$\mathcal{F}(\boldsymbol{x};\Omega^n) = \boldsymbol{\omega} \cdot \boldsymbol{x}^{[L]} + b, \tag{6.28}$$

  where $\boldsymbol{\omega} \in \mathbb{R}^{d^{[L]}}$, $b \in \mathbb{R}$ are trainable weight and bias.

In addition, we propose adding a scaling parameter α^n to each neural network and revise (6.23) as

$$y^n(\boldsymbol{x};\,\Omega^n) = \alpha^n \left[y^{n+1}(\boldsymbol{x};\,\Omega^{n+1}) + \Delta t \cdot \mathcal{F}(\boldsymbol{x};\Omega^n) \right], \quad n = N-1, \cdots, 0. \tag{6.29}$$

We let α^n be trainable, or equivalently, $\alpha^n \in \Omega^n$. α^n is initialized as 1 before training, and is close to 1 during and after training. Introducing the trainable parameter α^n expands the function space the neural network can represent. A neural network with a larger function space is less likely to underfit, and thus more likely to have an accurate training result [80].

We remark that our proposed recursive architecture (6.29) is different from the other architectures in the literature, particularly [142], where one single neural network is used to represent the spacetime price function. To justify our choice of the recursive architecture, we note that the true price functions $q^{n+1}(\boldsymbol{x})$ and $q^n(\boldsymbol{x})$ differ by a function of magnitude $O(\Delta t)$. In (6.29), if we let $y^{n+1}(\boldsymbol{x};\Omega^{n+1}) \approx q^{n+1}(\boldsymbol{x})$ and $\alpha^n \approx 1$, then regardless of the value of $\mathcal{F}(\boldsymbol{x};\Omega^n)$, $y^n(\boldsymbol{x};\Omega^n)$ will only differ from the true price function $q^n(\boldsymbol{x})$ by a magnitude of $O(\Delta t)$. Hence, before training starts, $y^n(\boldsymbol{x};\Omega^n)$ is already a good approximation of $q^n(\boldsymbol{x})$. This makes it more likely for the training to find the optimal solution that (almost) equals $q^n(\boldsymbol{x})$. Therefore, the recursive architecture is critical to the accuracy of the resulting prices and deltas.

6.4.2 Computation of derivatives

Using the BSDE formulation (6.20)-(6.21) requires computation of the derivatives $\nabla q^n(\boldsymbol{x})$, which is approximated by the gradient of the neural network output, i.e., $\nabla y^n(\boldsymbol{x};\Omega^n)$. We use Tensorflow [1] to implement the neural network model. The gradient $\nabla y^n(\boldsymbol{x};\Omega^n)$ can be easily evaluated by Tensorflow's built-in automatic differentiation, "tf.gradients". Later we will describe how to train the neural network using both $y^n(\boldsymbol{x};\Omega^n)$ and $\nabla y^n(\boldsymbol{x};\Omega^n)$ under the BSDE loss function (6.20)-(6.21).

6.4.3 Smoothing payoff functions

We note that most of the payoff functions in practical applications have the form of (6.3), which is not differentiable at $g(\boldsymbol{x}) = 0$. In other words, $y^N(\boldsymbol{x})$ in (6.22) is not differentiable. However, $y^{N-1}(\boldsymbol{x};\Omega^{N-1})$ as an approximation of the continuation price function is differentiable. Consequentially, the left and right hand sides of

$$y^{N-1}(\boldsymbol{x};\,\Omega^{N-1}) = \alpha^{N-1}\left[y^N(\boldsymbol{x}) + \Delta t \cdot \mathcal{F}(\boldsymbol{x};\Omega^{N-1})\right], \tag{6.30}$$

are inconsistent in terms of differentiability. Such inconsistency makes it difficult to learn an accurate $\mathcal{F}(\boldsymbol{x};\Omega^{N-1})$, which negatively affects the accuracy of the trained $y^{N-1}(\boldsymbol{x};\Omega^{N-1})$,

and furthermore, the accuracy of the trained $y^n(\boldsymbol{x};\Omega^n)$ in the subsequent timesteps. In this chapter, we propose smoothing the function $y^N(\boldsymbol{x})$ in (6.22) as follows:

$$y^N(\boldsymbol{x}) = f_\kappa(\boldsymbol{x}) \equiv \frac{1}{\kappa}\ln\left(1+e^{\kappa g(\boldsymbol{x})}\right), \tag{6.31}$$

where κ is a user-defined parameter. The operations in (6.31) are evaluated element-wise. $f_\kappa(\boldsymbol{x})$ converges to $f(\boldsymbol{x})$ when $\kappa \to \infty$, and is a good approximation of $f(\boldsymbol{x})$ when κ is large. The significance of (6.31) is that $f_\kappa(\boldsymbol{x})$ is differentiable, which makes it easier to train an accurate $\mathcal{F}(\boldsymbol{x};\Omega^{N-1})$. In practice, we choose $\kappa = \frac{2}{\Delta t}$. We note that smoothing payoff functions is a standard technique in the literature of binomial trees for option pricing [91]. However, to the best of our knowledge, this is the first proposal of smoothing payoff functions among the literature of neural networks for option pricing.

6.4.4 Feature selection

Feature selection, i.e., choosing the correct input features based on domain knowledge, has a great impact on the accuracy of neural network models [80]. Naively one can simply set the input as the underlying asset prices $\boldsymbol{x}^{[0]} = \boldsymbol{x}$. In this chapter, we consider adding two new features.

One new feature is the payoff function. It is suggested in [103, 64] that including the payoff in the nonlinear basis can improve the accuracy of the regression-based algorithms.

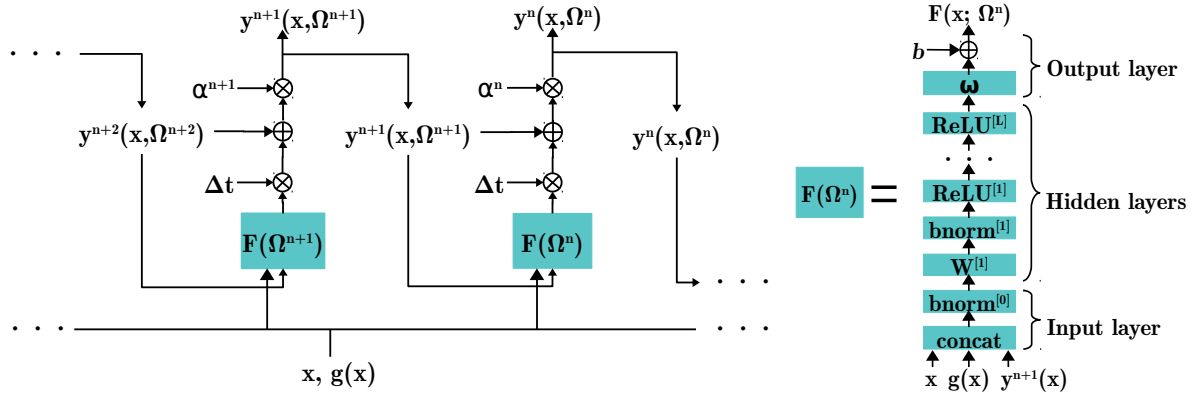


Figure 6.1: The architecture of the proposed neural network framework defined by (6.29) and (6.31), where the remainder network at each timestep $\mathcal{F}(\boldsymbol{x};\Omega^n)$ is defined by the input layer (6.32)-(6.33), the hidden layers (6.24)-(6.26) and the output layer (6.28). The symbols $\otimes$ and $\oplus$ represent multiplication and addition, respectively.

In this chapter, we consider using $g(\boldsymbol{x})$ in (6.3) as an input feature. The reason of using $g(\boldsymbol{x})$ rather than $f(\boldsymbol{x})$ is that the maximum operator in (6.3) is irreversible. In other words, $f(\boldsymbol{x})$ can be computed by $g(\boldsymbol{x})$ but not conversely. Hence, using $g(\boldsymbol{x})$ as the input contains more information than $f(\boldsymbol{x})$. The additional maximum operator in (6.3) can be learned by the activation function (6.26) in the network.

The other new feature we consider adding is the output price function from the previous timestep, i.e., $y^{n+1}(\boldsymbol{x};\,\Omega^{n+1})$ in (6.29). The intuition is that the solution at the n-th step should look similar to the solution at the $(n+1)$-th step.

The accuracy of neural network models can be further improved by input normalization [144]. Effectively, we can combine the implementation of feature selection and input normalization by adding the following "input layer" (denoted as $l=0$) before the hidden layer $l=1$:

$$\text{feature concatenation: } \boldsymbol{z}^{[0]} = \big(\boldsymbol{x},\, g(\boldsymbol{x}),\, y^{n+1}(\boldsymbol{x};\,\Omega^{n+1})\big)^T \in \mathbb{R}^{d^{[0]}}, \tag{6.32}$$

$$\text{input normalization: } \boldsymbol{x}^{[0]} = \text{bnorm}(\boldsymbol{z}^{[0]}; \boldsymbol{\beta}^{[0]}, \boldsymbol{\gamma}^{[0]}, \boldsymbol{\mu}^{[0]}, \boldsymbol{\sigma}^{[0]}), \tag{6.33}$$

where the input dimension is changed to $d^{[0]} = d+2$ after the concatenation. We note that $\boldsymbol{\mu}^{[0]}$ and $\boldsymbol{\sigma}^{[0]}$ can be pre-computed from the entire training dataset, unlike $\boldsymbol{\mu}^{[l]}$ and $\boldsymbol{\sigma}^{[l]}$ in the hidden layers that are computed by moving averages of training batches.

To summarize Sections 6.4.1-6.4.4, the architecture of the proposed neural network framework is defined by (6.29) and (6.31), where the remainder network at each timestep $\mathcal{F}(\boldsymbol{x};\Omega^n)$ is defined by the input layer (6.32)-(6.33), the hidden layers (6.24)-(6.26) and the output layer (6.28). The trainable parameters of the neural network framework are $\{\Omega^n \,|\, n = N-1, \cdots, 0\}$, where

$$\Omega^n \equiv \{(\boldsymbol{W}^{[l]})^n, (\boldsymbol{\gamma}^{[l]})^n, (\boldsymbol{\beta}^{[l]})^n, (\boldsymbol{\gamma}^{[0]})^n, (\boldsymbol{\beta}^{[0]})^n, \boldsymbol{\omega}^n, b^n, \alpha^n \,|\, L = 1, \cdots, L\}. \tag{6.34}$$

Figure 6.1 illustrates the architecture of the proposed neural network framework.

6.4.5 More efficient neural network sequence

We discussed the advantage of the recursive architecture (6.29) at the end of Section 6.4.1. However, the recursive architecture is expensive when N is large. More specifically, consider the 0-th timestep, and consider using the sequence of the neural networks to compute the value of $y^0(\boldsymbol{x})$. By applying the recursive relation (6.29), we have

$$y^0(\boldsymbol{x}) = y^N(\boldsymbol{x}) + \Delta t \cdot \sum_{\nu=1}^{N} \mathcal{F}(\boldsymbol{x};\Omega^{N-\nu}), \tag{6.35}$$

where for simplicity we set $\alpha^n = 1$ for all timesteps. Equation (6.35) shows that the computation of $y^0(\boldsymbol{x})$ requires going through N feedforward networks.

Here we propose a modified neural network architecture to reduce the computational cost. In Section 6.4.1, we motivate the recursive relation (6.29) based on the fact that the outputs of the two adjacent timesteps, $y^n(\boldsymbol{x})$ and $y^{n+1}(\boldsymbol{x})$, should differ by a function of magnitude $O(\Delta t)$. In fact, we can generalize this relation to any two timesteps n and $n+j$ where $j \ll N$. That is, the outputs $y^n(\boldsymbol{x})$ and $y^{n+j}(\boldsymbol{x})$ should differ by a function of magnitude $O(\Delta t)$. Similar to (6.29), we formulate this idea into the following recursive relation:

$$y^n(\boldsymbol{x};\, \Omega^n) = \alpha^n \left[y^{n+j}(\boldsymbol{x};\, \Omega^{n+j}) + j\Delta t \cdot \mathcal{F}(\boldsymbol{x}; \Omega^n) \right]. \tag{6.36}$$

This generalization allows us to recur the feedforward networks at every few timesteps, rather than at every single timestep, and thus reduces the computational cost.

To be more precise, if we recur the feedforward networks at every J timesteps ($J \ll N$), then we modify the sequence of the neural networks (6.29) as follows:

$$\begin{aligned} y^n(\boldsymbol{x};\, \Omega^n) &= \alpha^n \left[y^{n+\eta}(\boldsymbol{x};\, \Omega^{n+\eta}) + \eta\Delta t \cdot \mathcal{F}(\boldsymbol{x}; \Omega^n) \right], \\ &\text{where } \eta \equiv [(N-n-1) \bmod J] + 1, \quad n = N-1, \cdots, 0. \end{aligned} \tag{6.37}$$

We remark that (6.29) is simply a special case of (6.37) with $J = 1$. Figure 6.2 illustrates the modified architecture with $J = 3$. Readers can generalize the idea of Figure 6.2 to any $J \ll N$.

Regarding the choice of J, smaller J yields more precise trained y^n with higher computational cost; larger J is computationally cheaper but the trained y^n is less precise. In our numerical simulations, we choose $N = 100$ and $J = 4$.

To give an example on how the modified architecture reduces the computational cost, let us reconsider evaluating $y^0(\boldsymbol{x})$. By applying the recursive relation (6.37), we have

$$y^0(\boldsymbol{x}) = y^N(\boldsymbol{x}) + J\Delta t \cdot \sum_{\nu=1}^{\lfloor N/J \rfloor} \mathcal{F}(\boldsymbol{x}; \Omega^{N-\nu J}) + (N \bmod J)\Delta t \cdot \mathcal{F}(\boldsymbol{x}; \Omega^0), \tag{6.38}$$

where for simplicity we set $\alpha^n = 1$ for all timesteps. Compared to (6.35), using (6.38) to compute $y^0(\boldsymbol{x})$ only requires going through $\lceil N/J \rceil$ feedforward networks. In other words, the computation is J times cheaper.

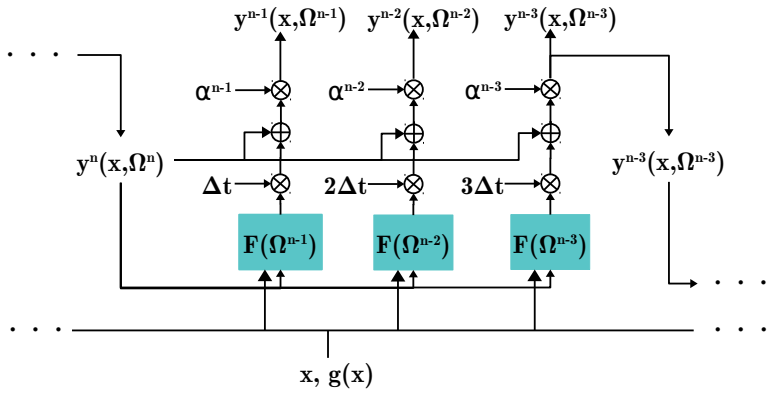


Figure 6.2: The modified architecture of the proposed neural network framework defined by (6.31) and (6.37), where $J = 3$. Similar to Figure 6.1, the remainder network at each timestep $\mathcal{F}(\boldsymbol{x}; \Omega^n)$ is defined by the input layer (6.32)-(6.33), the hidden layers (6.24)-(6.26) and the output layer (6.28).

6.4.6 Training neural networks

Consider training the network at the n-th timestep for solving (6.20)-(6.21). The training inputs are

$$\{\boldsymbol{X}_m^n,\ \Delta \boldsymbol{W}_m^n,\ u^{n+1}(\boldsymbol{X}_m^{n+1}), g(\boldsymbol{X}_m^n),\ y^{n+\eta}(\boldsymbol{X}_m^n; (\Omega^{n+\eta})^*),\ \nabla y^{n+\eta}(\boldsymbol{X}_m^n; (\Omega^{n+\eta})^*) \,|\, m = 1, \cdots, M\}, \tag{6.39}$$

where the blue inputs are the required inputs of (6.20), the red inputs are the features introduced in Section 6.4.4, $y^{n+\eta}$ is defined in (6.37), and $(\Omega^{n+\eta})^*$ is the trained parameters from the previous timestep $n + \eta$. The training output is $\{y^n(\boldsymbol{X}_m^n; \Omega^n),\ \nabla y^n(\boldsymbol{X}_m^n; \Omega^n) \,|\, \forall m\}$. The loss function of the network is given by (6.20)-(6.21), i.e., the least squares BSDE residual, which we rewrite as a function of the trainable parameters Ω^n:

$$\begin{aligned} \mathcal{L}[\Omega^n] \equiv \sum_{m=1}^{M} \Bigg[& (1 + r\Delta t) y^n(\boldsymbol{X}_m^n; \Omega^n) \\ & + \sum_{i=1}^{d} \sigma_i (X_i)_m^n \frac{\partial y^n}{\partial x_i}(\boldsymbol{X}_m^n; \Omega^n)(\Delta W_i)_m^n - u^{n+1}(\boldsymbol{X}_m^{n+1}) \Bigg]^2 . \end{aligned} \tag{6.40}$$

We consider using the popular Adam optimizer [101] to minimize the loss function (6.40), which yields the set of optimal trainable parameters

$$(\Omega^n)^* \equiv \arg\min_{\Omega^n} \mathcal{L}[\Omega^n]. \tag{6.41}$$

Then, using the trained neural network, we can compute the estimated option price $y^n(\boldsymbol{x}; (\Omega^n)^*)$ and delta $\nabla y^n(\boldsymbol{x}; (\Omega^n)^*)$. In addition, we use the estimated option price to determine the exercise boundary as

$$c^n(\boldsymbol{x}) = \begin{cases} 1 \text{ (exercised)}, & \text{if } y^n(\boldsymbol{x}; (\Omega^n)^*) \leq f(\boldsymbol{x}), \\ 0 \text{ (continued)}, & \text{otherwise.} \end{cases} \tag{6.42}$$

In order to ensure the accuracy of training, we follow suggested good practices in the deep learning community [80]. For instance, mini-batch optimization is used; the learning rate of the Adam optimizer is decayed to ensure convergence; gradient clipping is applied to avoid exploding gradients. In particular, we let the number of training steps be 600. At the s-th training step ($0 \leq s \leq 600$), we let the moving average rate for $\boldsymbol{\mu}^{[l]}$ and $\boldsymbol{\sigma}^{[l]}$ in (6.25) be $\frac{1}{0.99}(0.01^{\max(\min(s/350,1),0)} - 0.01)$, and let the learning rate for the Adam optimizer be $0.01 \times 0.001^{\max(\min((s-150)/350,1),0)}$.

At this point, it becomes clear that our proposed framework indeed fits into the general framework introduced in Section 1.3.4. There are two major differences between our proposed framework and the general framework. One is that the network architecture is different. The other distinction is that, while Section 1.3.4 minimizes the residual norm of an HJB equation, this chapter minimizes the residual norm of the equivalent BSDE, such that the computation and storage of Hessian tensors can be avoided.

6.5 Improving the Algorithm

Sections 6.3-6.4 describe the foundation of our algorithm. This section introduces a few techniques that improve the accuracy of resulting prices and deltas and the efficiency of the algorithm.

6.5.1 The training input u^{n+1}

Consider the n-th timestep. The definition of $u^{n+1}(\boldsymbol{X}_m^{n+1})$ in the training input (6.39) turns out to play a significant role in the accuracy of the trained continuation price y^n. More specifically, if the training input $u^{n+1}(\boldsymbol{X}_m^{n+1})$ is incorrectly defined, which means that we feed incorrect values to the right hand side of (6.18), then the trained network y^n would not represent the correct q^n.

Finding the correct definition of $u^{n+1}(\boldsymbol{X}_m^{n+1})$ turns out to be non-trivial. One natural way of defining $u^{n+1}(\boldsymbol{X}_m^{n+1})$ is to use the output prices of the trained network. More specifically, suppose $y^{n+1}(\boldsymbol{x}; (\Omega^{n+1})^*)$ is already trained. Then

$$u^{n+1}(\boldsymbol{X}_m^{n+1}) = \begin{cases} y^{n+1}(\boldsymbol{X}_m^{n+1}; (\Omega^{n+1})^*), & \text{if } c^{n+1}(\boldsymbol{X}_m^{n+1}) = 0 \text{ (continued)}, \\ f(\boldsymbol{X}_m^{n+1}), & \text{if } c^{n+1}(\boldsymbol{X}_m^{n+1}) = 1 \text{ (exercised)}, \end{cases} \tag{6.43}$$

where c^{n+1} is defined in (6.42). However, in practice, due to the finite number of samples and training steps, training error in the network y^{n+1} is inevitable, which means that $u^{n+1}(\boldsymbol{X}_m^{n+1})$ might contain error after applying (6.43). Consequentially, the error of the training input $u^{n+1}(\boldsymbol{X}_m^{n+1})$ will propagate into y^n after training the n-th network, and propagate into $u^n(\boldsymbol{X}_m^n)$ after applying (6.43) again, and propagate into y^{n-1}, $u^{n-1}(\boldsymbol{X}_m^{n-1})$, y^{n-2}, ..., after further backward timestepping. In other words, (6.43) is not robust against the accumulation of training errors over timesteps and may result in bias.

In fact, such bias can be quantified using the following lemma:

Lemma 6.1 (Quantifying bias)**.** Assume that $\{\boldsymbol{X}_m^\nu \,|\, 0 \le \nu \le n, \forall m\}$ are known and fixed, i.e., assume that the stochastic process $\{\boldsymbol{X}^n\}$ is adapted to the filtration $\{\mathcal{F}_n\}$. Let $\{\boldsymbol{X}_m^{n+1} \,|\, \forall m\}$ be another set generated under (6.15). Then the prices q^n and u^{n+1} must satisfy

$$q^n(\boldsymbol{X}_m^n) = \mathbb{E}[e^{-r\Delta t} u^{n+1}(\boldsymbol{X}_m^{n+1}) \,|\, \boldsymbol{X}_m^n] + O(\sqrt{\Delta t}), \tag{6.44}$$

where Δt is the same as in (6.18).

Proof. Consider taking the conditional expectation of (6.18):

$$(1+r\Delta t) q^n(\boldsymbol{X}_m^n) + \sum_{i=1}^{d} \sigma_i(X_i)_m^n \frac{\partial q^n}{\partial x_i}(\boldsymbol{X}_m^n) \mathbb{E}[(\Delta W_i)_m^n \,|\, \boldsymbol{X}_m^n] = \mathbb{E}[u^{n+1}(\boldsymbol{X}_m^{n+1}) \,|\, \boldsymbol{X}_m^n] + O(\sqrt{\Delta t}),$$

where we add the term $O(\sqrt{\Delta t})$ at the end of the equation to reflect the discretization error of (6.18). We note that $\{\boldsymbol{X}_m^n\}$, $\{q^n(\boldsymbol{X}_m^n)\}$ and $\{\frac{\partial q^n}{\partial x_i}(\boldsymbol{X}_m^n)\}$ are not random variables due to the filtration, and hence the only random variables are $\{\boldsymbol{X}_m^{n+1}\}$ and $\{(\Delta W_i)_m^n\}$. Since $\mathbb{E}[(\Delta W_i)_m^n \,|\, \boldsymbol{X}_m^n] = 0$ and $1 + r\Delta t = e^{r\Delta t} + O(\Delta t^2)$, we have $e^{r\Delta t} q^n(\boldsymbol{X}_m^n) + O(\Delta t^2) = \mathbb{E}[u^{n+1}(\boldsymbol{X}_m^{n+1}) \,|\, \boldsymbol{X}_m^n] + O(\sqrt{\Delta t})$, which gives (6.44). □

Lemma 6.1 indicates that if $u^{n+1}(\boldsymbol{X}_m^{n+1})$ is correctly evaluated, then $\mathbb{E}[e^{-r\Delta t} u^{n+1}(\boldsymbol{X}_m^{n+1})]$ should match the true underlying continuation function $q^n(\boldsymbol{X}_m^n)$. After a few timesteps, if $\mathbb{E}[e^{-r\Delta t} u^{n+1}(\boldsymbol{X}_m^{n+1})]$ deviates from $q^n(\boldsymbol{X}_m^n)$, then it indicates an accumulation of training errors from the previous timesteps.

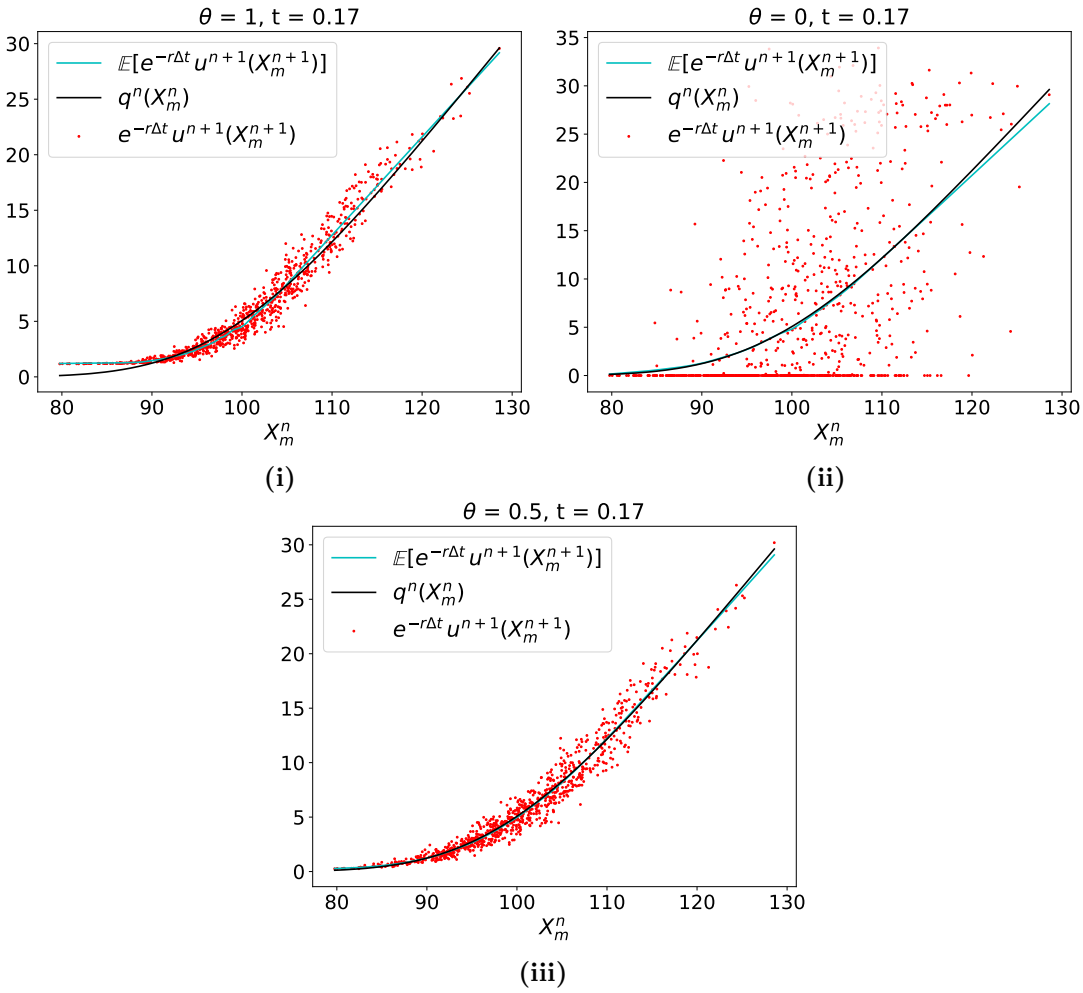


Figure 6.3: The values of $q^n(\boldsymbol{X}_m^n)$ (black line), $e^{-r\Delta t}u^{n+1}(\boldsymbol{X}_m^{n+1})$ (red dots) and $\mathbb{E}[e^{-r\Delta t}u^{n+1}(\boldsymbol{X}_m^{n+1})]$ (blue line) under different definitions of $u^{n+1}(\boldsymbol{X}_m^{n+1})$. **(i)** The values under the definition of (6.43), which shows a bias; **(ii)** The values under the definition of (6.45), which shows a variance; **(iii)** The values under the definition of (6.46) with $\theta = 0.5$, where both bias and variance are reduced.

Figure 6.3(i) shows a concrete example of the bias. Consider a simulation of a one-dimensional American option, where $T = 0.5$, $N = 100$ and the true continuation function q^n can be computed by finite difference methods. Consider using (6.43) to define the training input $u^{n+1}(\boldsymbol{X}_m^{n+1})$ at every timestep. As shown in Figure 6.3(i), when the simulation

proceeds to $n = 33$, there is a clear deviation of $\mathbb{E}[e^{-r\Delta t}u^{n+1}(\boldsymbol{X}_m^{n+1})]$ (blue line)[4] from $q^n(\boldsymbol{X}_m^n)$ (black line) around $\boldsymbol{X}_m^n = 80$.

In fact, we can use the relation (6.44) to avoid the bias caused by the definition (6.43). More specifically, let $\boldsymbol{X}_m^{n+1}$ be a continued point. Then $u^{n+1}(\boldsymbol{X}_m^{n+1}) = q^{n+1}(\boldsymbol{X}_m^{n+1}) = \mathbb{E}[e^{-r\Delta t}u^{n+2}(\boldsymbol{X}_m^{n+2})]$. This motivates us to redefine the training input $u^{n+1}(\boldsymbol{X}_m^{n+1})$ as follows:

$$u^{n+1}(\boldsymbol{X}_m^{n+1}) = \begin{cases} e^{-r\Delta t}\, u^{n+2}(\boldsymbol{X}_m^{n+2}), & \text{if } c^{n+1}(\boldsymbol{X}_m^{n+1}) = 0 \text{ (continued)}, \\ f(\boldsymbol{X}_m^{n+1}), & \text{if } c^{n+1}(\boldsymbol{X}_m^{n+1}) = 1 \text{ (exercised)}. \end{cases} \tag{6.45}$$

We note that (6.45) is actually the "discounted payoffs" used in Longstaff and Schwartz's approach [117]. They use (6.45) as target prices for regression.

Figure 6.3(ii) considers again the same simulation, where the definition of $u^{n+1}(\boldsymbol{X}_m^{n+1})$ is changed to (6.45). The deviation of $\mathbb{E}[e^{-r\Delta t}u^{n+1}(\boldsymbol{X}_m^{n+1})]$ (blue line) from $q^n(\boldsymbol{X}_m^n)$ (black line) around $\boldsymbol{X}_m^n = 80$ disappears. The blue and black lines agrees well with each other. This shows that using the definition (6.45) does not introduce bias as does the definition (6.43). However, the noisy red dots show that using the definition (6.45) results in a big variance of $e^{-r\Delta t}u^{n+1}(\boldsymbol{X}_m^{n+1})$. This poses a risk for the model to fit the noise, which may still result in an inaccurate trained y^n.

In this chapter, we define $u^{n+1}(\boldsymbol{X}_m^{n+1})$ as the linear combination of the two definitions (6.43) and (6.45):

$$u^{n+1}(\boldsymbol{X}_m^{n+1}) = \begin{cases} \theta\, y^{n+1}(\boldsymbol{X}_m^{n+1};\, (\Omega^{n+1})^*) + (1-\theta)e^{-r\Delta t}\, u^{n+2}(\boldsymbol{X}_m^{n+2}), & \\ & \text{if } c^{n+1}(\boldsymbol{X}_m^{n+1}) = 0 \text{ (continued)}, \\ f(\boldsymbol{X}_m^{n+1}), & \text{if } c^{n+1}(\boldsymbol{X}_m^{n+1}) = 1 \text{ (exercised)}, \end{cases} \tag{6.46}$$

where $\theta \in [0, 1]$ is a user-defined hyperparameter. This linear combination mitigates both the bias caused by the definition (6.43) and the variance caused by the definition (6.45). That is, the resulting $u^{n+1}(\boldsymbol{X}_m^{n+1})$ would accumulate less training error over multiple timesteps, and meanwhile contain less noise. Figure 6.3(iii) considers the same simulation, where the definition of $u^{n+1}(\boldsymbol{X}_m^{n+1})$ is (6.46) with $\theta = 0.5$. We observe almost no deviation of $\mathbb{E}[e^{-r\Delta t}u^{n+1}(\boldsymbol{X}_m^{n+1})]$ (blue line) from $q^n(\boldsymbol{X}_m^n)$ (black line), and a small variance of $e^{-r\Delta t}u^{n+1}(\boldsymbol{X}_m^{n+1})$ (red dots), as expected. Hence, the definition (6.46) can improve the accuracy of the trained networks.

[4] To assess $\mathbb{E}[e^{-r\Delta t}u^{n+1}(\boldsymbol{X}_m^{n+1})]$, we start with a fixed set of $\{\boldsymbol{X}_m^n\}$. For each point of $\boldsymbol{X}_m^n$, we generate multiple $\boldsymbol{X}_m^{n+1}$'s by (6.15), denoted as $\{\boldsymbol{X}_{m;m'}^{n+1} | m' = 1, \cdots, M'\}$; compute $\{u(\boldsymbol{X}_{m;m'}^{n+1})\}$; and then compute the imperial average: $\mathbb{E}[e^{-r\Delta t}u^{n+1}(\boldsymbol{X}_m^{n+1})] \approx e^{-r\Delta t}\frac{1}{M'}\sum_{m'} u(\boldsymbol{X}_{m;m'}^{n+1})$.

6.5.2 Weight reuse

The trainable parameters Ω^n need to be initialized for each individual network from $n = N-1$ to $n = 0$. Starting from the network at $n = N-1$, we initialize $(\boldsymbol{\beta}^{[l]})^{N-1}$ and b^{N-1} by zeros; $(\boldsymbol{\gamma}^{[l]})^{N-1}$ and α^{N-1} by ones; and $(\boldsymbol{W}^{[l]})^{N-1}$ and $\boldsymbol{\omega}^{N-1}$ by uniformly distributed random numbers in $(-1/\sqrt{d^{[l]}+d^{[l-1]}}, 1/\sqrt{d^{[l]}+d^{[l-1]}})$, as suggested in [80]. Move on to the consecutive networks at $n < N-1$. One can use the same idea to initialize their trainable parameters. However, we notice that when Δt is sufficiently small, the networks at the n-th and $(n+1)$-th timesteps should be close. In other words, their optimal trainable parameters should be close, i.e., $(\Omega^{n+1})^* \approx (\Omega^n)^*$. We can take advantage of this fact and use the values of the trained parameters $(\Omega^{n+1})^*$ as the initial values of the corresponding trainable parameters Ω^n. Such "weight reuse" provides a good initial guess before the training starts at the n-th timestep. Hence, the training results will be more accurate.

Figure 6.4 demonstrates a concrete example on how weight reuse improves the training accuracy. Consider again a simulation of a one-dimensional American option with $T = 0.5$, $N = 50$. Consider a particular timestep $n = 47$. We computed the delta $\frac{dy^n}{dx}(X_m^n)$ of 180000 sample points. Figure 6.4(i) shows the evolution of the L_1 norm error of the computed delta over 600 training steps. The error with weight reuse (red line) is significantly lower than the error without weight reuse (blue line). Figure 6.4(ii) shows that after 600 training steps, the computed delta with weight reuse (red dots) agrees with the exact delta (black

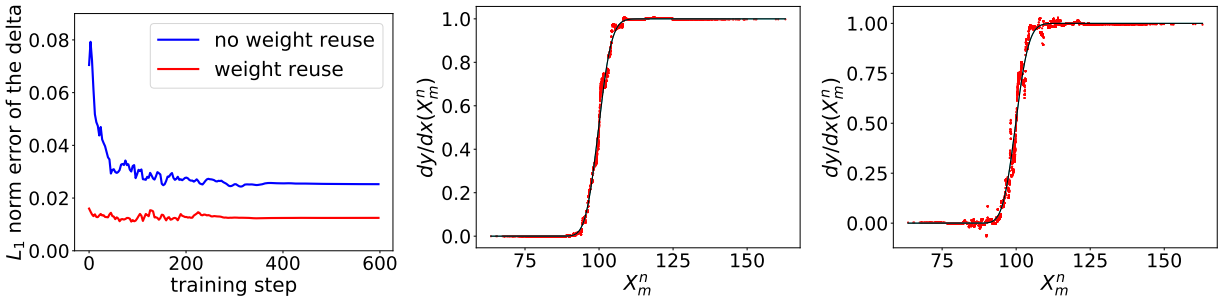


Figure 6.4: Example of the computed deltas with or without weight reuse. **(i)** The L_1 norm error of the computed delta over 600 training steps. Blue: the error with no weight reuse. Red: the error with weight reuse. **(ii)** The computed delta with weight reuse after 600 training steps. Black line: the exact delta computed by finite difference. Red dots: the sample values of the delta obtained from the neural network y^n. **(iii)** The computed delta without weigh reuse after 600 training steps.

line). As a comparison, Figure 6.4(iii) shows that after 600 training steps, the computed delta without weight reuse (red dots) does not match the exact delta (black line) well.

6.5.3 Final algorithm

The final version of the proposed algorithm is summarized in Algorithm 6.1. We remark that the algorithm uses "ensemble of neural networks", which is a combination and average of multiple neural networks. The reason is that ensemble learning usually outperforms individual models [80]. Interested readers are referred to Appendix A.7 for details. We also note that in Algorithm 6.1, we store $\{y^n(\boldsymbol{X}_m^n), \nabla y^n(\boldsymbol{X}_m^n) \,|\, \forall n, \forall m\}$ on the entire spacetime (i.e., for all m's and n's). The reason is that we are interested in a complete delta hedging simulation, which requires sample values of both prices and deltas on the entire spacetime. The implementation of Algorithm 6.1 uses an overwriting strategy for more efficient memory. We note, however, that if an algorithm user does not need sample values from the entire spacetime, then only the storage of the training outputs $\{y^n(\boldsymbol{X}_m^n), \nabla y^n(\boldsymbol{X}_m^n) \,|\, \forall m\}$ and the training inputs $\{y^{n+\eta}(\boldsymbol{X}_m^n), \nabla y^{n+\eta}(\boldsymbol{X}_m^n) \,|\, \forall m\}$ at the current timestep (i.e., for all m's and for a given n) is necessary.

6.6 Computational Cost and Errors

In this section, we analyze the computational cost and the errors of the proposed algorithm, and make a comparison with the Longstaff-Schwartz algorithm. For the Longstaff-Schwartz algorithm, consider degree-χ monomial basis [117, 103]

$$\varphi_\chi(\boldsymbol{x}) \equiv \{x_1^{a_1} x_2^{a_2} \cdots x_d^{a_d} \,|\, a_1 + a_2 + \cdots + a_d \leq \chi\}. \tag{6.47}$$

In practice, we choose $\chi \ll d$. Then the number of the monomial basis is $\binom{d+\chi}{d} \approx \frac{1}{\chi!} d^\chi$.

6.6.1 Memory

The proposed algorithm requires storing

- the underlying asset prices $\{\boldsymbol{X}_m^n \,|\, \forall n, \forall m\}$ on the entire spacetime, requiring NMd floating point numbers;
- the training outputs $\{y^n(\boldsymbol{X}_m^n), \nabla y^n(\boldsymbol{X}_m^n) \,|\, \forall m\}$ and the training inputs $\{y^{n+\eta}(\boldsymbol{X}_m^n), \nabla y^{n+\eta}(\boldsymbol{X}_m^n) \,|\, \forall m\}$ at the current timestep, requiring $2(M + Md)$ floating point numbers.

Algorithm 6.1 Neural network pricing and hedging under BSDE formulation

Parameters
  C: the number of networks in a network ensemble
  M: the number of samples per ensemble
  N: the number of timesteps
  J: the number of timesteps between the network recurrence

Initialize the underlying asset prices $\{\boldsymbol{X}_m^0 \equiv \boldsymbol{x}^0 \,|\, \forall m \text{ (i.e., } m = 1, \cdots, CM)\}$.
for $n = 1, \cdots, N$ **do**
  Use (6.14)-(6.15) to generate CM trajectories of the underlying asset prices $\{\boldsymbol{X}_m^n \,|\, \forall m\}$.
end for

Use (6.31) to compute the expiry option prices and option deltas
  $\{Y_m^\nu = y^N(\boldsymbol{X}_m^\nu) \,|\, 0 \le \nu \le N, \forall m\}$ and $\{\boldsymbol{Z}_m^\nu = \nabla y^N(\boldsymbol{X}_m^\nu) \,|\, 0 \le \nu \le N, \forall m\}$.
Initialize $\{u^N(\boldsymbol{X}_m^N) = f(\boldsymbol{X}_m^N) \,|\, \forall m\}$ by (6.17).
for $n = N-1, \cdots, 0$ **do**
  for $c = 1, \cdots, C$ **do**
    Initialize the neural network $y^n(\boldsymbol{x}; \Omega_c^n)$ defined by (6.37), where the input layer is (6.32)-(6.33), the hidden layers are (6.24)-(6.26) and the output layer is (6.28).
    Training: minimize the least squares residual (6.40)-(6.41), using the training input (6.39).
    Result: the trained neural network $y^n(\boldsymbol{x}; (\Omega_c^n)^*)$.
  end for

  if $(N-n) \bmod J = 0$ **then**
    Ensemble evaluation (all future timesteps): overwrite the option prices and deltas
      $\{Y_m^\nu = \frac{1}{C}\sum_{c=1}^C y^n(\boldsymbol{X}_m^\nu; (\Omega_c^n)^*) \,|\, 0 \le \nu \le n, \forall m\}$,
      $\{\boldsymbol{Z}_m^\nu = \frac{1}{C}\sum_{c=1}^C \nabla y^n(\boldsymbol{X}_m^\nu; (\Omega_c^n)^*) \,|\, 0 \le \nu \le n, \forall m\}$.
  else
    Ensemble evaluation (current timestep): overwrite the option prices and deltas
      $\{Y_m^n = \frac{1}{C}\sum_{c=1}^C y^n(\boldsymbol{X}_m^n; (\Omega_c^n)^*) \,|\, \forall m\}$,
      $\{\boldsymbol{Z}_m^n = \frac{1}{C}\sum_{c=1}^C \nabla y^n(\boldsymbol{X}_m^n; (\Omega_c^n)^*) \,|\, \forall m\}$.
  end if
  Determine whether $\boldsymbol{X}_m^n$ is continued or exercised using (6.42) for all m's.
  Update $\{u^n(\boldsymbol{X}_m^n) \,|\, \forall m\}$ by (6.46).
end for

Result: samples of option price and delta functions on the entire spacetime
  $\{Y_m^n \leftarrow \max(Y_m^n, f(\boldsymbol{X}_m^n)) \,|\, \forall n, \forall m\}$ and $\{\boldsymbol{Z}_m^n \,|\, \forall n, \forall m\}$.
Optional: Recompute the option price and the option delta at $t = 0$ using (A.12) and (A.13); see Appendix A.8 for details.

Hence, the entire process requires a total memory of $NMd + 2(M + Md) \approx NMd$ floating point numbers. As a comparison, the Longstaff-Schwartz method requires storing $\{\boldsymbol{X}_m^n \,|\, \forall n, \forall m\}$ on the entire spacetime and storing $\{\varphi_\chi(\boldsymbol{X}_m^n), y^n(\boldsymbol{X}_m^n) \,|\, \forall m\}$ at the current timestep. This requires a total memory of $NMd + M \cdot \frac{1}{\chi!} d^\chi + M \approx NMd + \frac{1}{\chi!} M d^\chi$ floating point numbers. We remind readers that convergence of the Longstaff-Schwartz method to the exact American option prices requires $\chi \to \infty$. As a result, the proposed neural network method is more memory efficient than the Longstaff-Schwartz method.

6.6.2 Time

Consider a given timestep n. The computational time is dominated by two stages:

- Stage 1: Computing the training inputs (6.39), in particular, $\{y^{n+\eta}(\boldsymbol{X}_m^n; (\Omega^{n+\eta})^*), \nabla y^{n+\eta}(\boldsymbol{X}_m^n; (\Omega^{n+\eta})^*) \,|\, \forall m\}$, using the trained networks $\{(\Omega^\nu)^* \,|\, \nu \geq n + \eta\}$.
- Stage 2: Training, using the training inputs (6.39).

To derive the computational time of each stage, denote the maximal width of the L-layer neural network $\mathcal{F}$ as $d_{max} \equiv \max_{l=0,\cdots,L} d^{[l]}$. We note that matrix multiplication is the dominant operation in (6.24)-(6.26). Hence, for each stage, the computational time *per neural network* is given by $c_1 MLd_{max}^2$ and $c_2 MLd_{max}^2$, where c_1 and c_2 are constants. Typically $c_1 \ll c_2$, because Stage 1 only involves computing the outputs of neural networks, while Stage 2 involves training. This seems to suggest that Stage 2 dominates Stage 1. However, we note that Stage 2 involves only one single network (i.e., the n-th network), while Stage 1 involves multiple networks from the previous timesteps. More specifically, following the same analysis as (6.38), one can show that the computation of training input $y^{n+\eta}(\boldsymbol{X}_m^n; (\Omega^{n+\eta})^*)$, given by

$$y^{n+\eta}(\boldsymbol{x}) = y^N(\boldsymbol{x}) + J\Delta t \cdot \sum_{\nu=1}^{(N-n-\eta)/J} \mathcal{F}(\boldsymbol{x}; \Omega^{N-\nu J}), \tag{6.48}$$

requires going through $(N - n - \eta)/J \approx (N - n)/J$ feedforward networks. As a result, the actual computational time for Stage 1 is $c_1 MLd_{max}^2 \cdot \frac{N-n}{J}$.

Furthermore, if we consider all the N timesteps, then the total computational time is

$$\begin{aligned} &\text{Stage 1:} \quad \sum_{n=0}^{N} c_1 MLd_{max}^2 \cdot \frac{N-n}{J} = \frac{c_1 N^2}{2J} MLd_{max}^2, \\ &\text{Stage 2:} \quad \sum_{n=0}^{N} c_2 MLd_{max}^2 = c_2 NMLd_{max}^2. \end{aligned} \tag{6.49}$$

Equation (6.49) suggests that when N is large, Stage 1 is dominant. However, we can significantly reduce the computational time of Stage 1 by increasing J, as discussed in Section 6.4.5. In our numerical simulation, we chose $d_{max} = d + 5$. Then the total computational time of the proposed algorithm is approximately $(\frac{c_1 N}{2J} + c_2)NMLd^2$, which is quadratic in the dimension d.

Regarding the Longstaff-Schwartz method, if we assume that the standard normal equation or QR factorization is used for solving regression problems, then the computational time is $O\left(NM(\frac{1}{\chi!}d^\chi)^2\right) = O(NMd^{2\chi})$, which is worse-than-quadratic in d. Hence, the proposed neural network method is asymptotically more efficient than the Longstaff-Schwartz method in high dimensions.

6.6.3 Errors

The errors of the computed option prices and deltas come from the following sources:

- the sampling error, resulting from the finite (rather than infinite) number of samples;
- the $O(\sqrt{\Delta t})$ truncation error, resulting from the timestepping of BSDE (6.18);
- the parameterization error, i.e., the difference between the true underlying solution of the discretized BSDE and the optimal parameterized solution that minimizes the least squares residual of the BSDE; and
- the training error, i.e., the difference between the optimal parameterized solution and the trained parameterized solution.

One may argue that the $O(\sqrt{\Delta t})$ truncation error using our BSDE formulation is not as accurate as the truncation error of some other American option approaches, particularly the Longstaff-Schwartz method. However, we note that the dominant error of the Longstaff-Schwartz method is the parameterization error. Even when the other sources of errors are minimized, the total error of the Longstaff-Schwartz method is still large. This is because the function space that the Longstaff-Schwartz parameterization can represent is limited by the degree χ of the polynomials, and may not cover the true solution. Conversely, we will show in our numerical simulation (e.g., Tables 6.1-6.2) that our proposed neural network approach yields a much smaller total error (which includes parameterization error). This agrees with the knowledge in the literature that deep neural networks have a desirable capacity and expressiveness, i.e., deep neural networks can represent a very large function space that is likely to include the true underlying solution [80].

6.7 Numerical Results

In this section, we solve the American option problem (6.2)-(6.6) using our neural network described in Algorithm 6.1. We compute the price $u(\boldsymbol{x}^0, 0)$ and the delta $\nabla u(\boldsymbol{x}^0, 0)$ at $t = 0$ for given $\boldsymbol{x}^0 = (x_1^0, \cdots, x_d^0)$ where $x_1^0 = \cdots = x_d^0 = 0.9K, K$ or $1.1K$. We also compute the prices $u(\boldsymbol{x}, t)$ and the deltas $\nabla u(\boldsymbol{x}, t)$ for sample paths of $(\boldsymbol{x}, t)$ spread over the entire spacetime.

In our experiments, we set the strike price $K = 100$, the number of the timesteps $N = 100$, the number of timesteps between the network recurrence $J = 4$, the smoothing parameter in (6.31) $\kappa = \frac{2}{\Delta t}$, the coefficient in (6.46) $\theta = 0.5$. At each timestep, we train an ensemble of $C = 3$ neural networks, where each neural network has a depth of $L = 7$ and a uniform width of $d^{[l]} = d + 5$ across all the hidden layers. We let the number of samples per network be $M = 240000$ (or the total number of samples be $CM = 720000$), and let the batch size and the number of training steps be 400 and 600 respectively. Each numerical experiment is implemented on one Cedar[5] base-GPU node, which contains 4 NVIDIA P100-PCIE-12GB GPUs, 24 CPUs and 128GB memory.

We compare the numerical results computed by our proposed method with those computed by the finite difference method, the Longstaff-Schwartz method and the method proposed in [142]. For the Longstaff-Schwartz method, we choose degree-χ monomial basis (6.47) with $\chi = 4$. Finite difference solutions with very fine grids are used as exact solutions. We note that this is feasible only if $d \leq 3$.

We note that when finite difference solutions are available, we can evaluate the absolute and percent errors of computed prices and deltas. More specifically, denote the finite difference solutions as u_{exact}. Then the percent errors of the price and the delta at $t = 0$ are

$$\frac{|u(\boldsymbol{x}^0, 0) - u_{exact}(\boldsymbol{x}^0, 0)|}{|u_{exact}(\boldsymbol{x}^0, 0)|} \times 100\%, \quad \frac{\|\nabla u(\boldsymbol{x}^0, 0) - \nabla u_{exact}(\boldsymbol{x}^0, 0)\|_{L_2}}{\|\nabla u_{exact}(\boldsymbol{x}^0, 0)\|_{L_2}} \times 100\%; \tag{6.50}$$

and the percent errors of the spacetime price and the spacetime delta are

$$\begin{aligned} &\frac{\sum_{m,n} |u(\boldsymbol{X}_m^n, t_m^n) - u_{exact}(\boldsymbol{X}_m^n, t_m^n)|}{\sum_{m,n} |u_{exact}(\boldsymbol{X}_m^n, t_m^n)|} \times 100\%, \\ &\frac{\sum_{m,n} \|\nabla u(\boldsymbol{X}_m^n, t_m^n) - \nabla u_{exact}(\boldsymbol{X}_m^n, t_m^n)\|_{L_2}}{\sum_{m,n} \|\nabla u_{exact}(\boldsymbol{X}_m^n, t_m^n)\|_{L_2}} \times 100\%. \end{aligned} \tag{6.51}$$

[5]Cedar is a Compute Canada's cluster. See https://docs.computecanada.ca/wiki/Cedar and https://docs.computecanada.ca/wiki/Using_GPUs_with_Slurm for details.

In addition, we can evaluate the quality of the computed exercise boundaries. More specifically, each sample point $(\boldsymbol{X}_m^n, t^n)$ is classified as "exercised" or "continued" by either the proposed algorithm or other algorithms that we compare with. Meanwhile, the true "exercised" or "continued" class of each sample point can be determined by the finite difference method. Let "exercised" class be the positive class, and denote the numbers of true positive, true negative, false positive and false negative samples as TP, TN, FP, FN, respectively. Then the quality of the exercise boundaries can be evaluated by the f1-score:

$$\text{f1-score} \equiv \frac{2TP}{2TP + FP + FN}. \tag{6.52}$$

The best (or worst) case of the f1-score is 1 (or 0), respectively. We note that another common metric to evaluate the quality of classification problems is the accuracy. Since in all our experiments, the positive class is skewed (around 3-17%), the f1-score would be a better metric than the accuracy; see [122] for explanations.

6.7.1 Multi-dimensional geometric average options

Consider a d-dimensional "geometric average" American call option, where $\rho_{ij} = \rho$ for $i \neq j$, $\sigma_i = \sigma$ for all i's, and the payoff function is given by $f(\boldsymbol{x}) = \max\left[\left(\prod_{i=1}^d x_i\right)^{1/d} - K, 0\right]$. Although such options are rarely seen in practical applications, they have semi-analytical solutions for benchmarking the performance of our algorithm in high dimensions. More specifically, it is shown in [78, 142] that such a d-dimensional option can be reduced to a one-dimensional American call option in the variable

$$s' \equiv \left(\prod_{i=1}^d x_i\right)^{1/d}, \tag{6.53}$$

where the effective volatility is $\sigma' = \sqrt{\frac{1+(d-1)\rho}{d}}\sigma$ and the effective drift is $r - \delta + \frac{1}{2}(\sigma'^2 - \sigma^2)$. Hence, by solving the equivalent one-dimensional option using finite difference methods, one can compute the d-dimensional option prices and (sometimes) deltas[6] accurately.

In the following Examples 6.1-6.5, we consider the geometric average option in Section 4.3 of [142], where $\rho_{i,j} = 0.75$, $\sigma = 0.25$, $r = 0$, $\delta = 0.02$, $T = 2$.

[6] We note that solving the equivalent one-dimensional option is not sufficient for computing the d-dimensional delta except at the symmetric points $x_1 = \cdots = x_d$. Interested readers can verify this by straightforward algebra.

Example 6.1 (Comparison between our proposed method and the Longstaff-Schwartz method)**.** First we compare the computed prices at $t = 0$; see Table 6.1. Each sub-table includes the following:

- the exact prices computed by the Crank-Nicolson finite difference method with 1000 timesteps and 16385 space grid points,
- the prices and the corresponding percent errors computed by our proposed method,
- the prices and the corresponding percent errors computed by the Longstaff-Schwartz method.

(i) 7-dimensional geometric average call option

x_i^0	exact price $u(\boldsymbol{x}^0, 0)$	proposed method		Longstaff-Schwartz	
		computed price $u(\boldsymbol{x}^0, 0)$	percent error	computed price $u(\boldsymbol{x}^0, 0)$	percent error
90	5.9021	5.8822	0.34%	5.8440	0.98%
100	10.2591	10.2286	0.30%	10.1736	0.83%
110	15.9878	15.9738	0.09%	15.8991	0.55%

(ii) 13-dimensional geometric average call option

x_i^0	exact price $u(\boldsymbol{x}^0, 0)$	proposed method		Longstaff-Schwartz	
		computed price $u(\boldsymbol{x}^0, 0)$	percent error	computed price $u(\boldsymbol{x}^0, 0)$	percent error
90	5.7684	5.7719	0.06%	5.5962	3.0%
100	10.0984	10.1148	0.16%	9.9336	1.6%
110	15.8200	15.8259	0.04%	15.6070	1.4%

(iii) 20-dimensional geometric average call option

x_i^0	exact price $u(\boldsymbol{x}^0, 0)$	proposed method		Longstaff-Schwartz	
		computed price $u(\boldsymbol{x}^0, 0)$	percent error	computed price $u(\boldsymbol{x}^0, 0)$	percent error
90	5.7137	5.7105	0.06%	5.2023	9.0%
100	10.0326	10.0180	0.15%	9.5964	4.4%
110	15.7513	15.7425	0.06%	15.2622	3.1%

(iv) 100-dimensional geometric average call option

x_i^0	exact price $u(\boldsymbol{x}^0, 0)$	proposed method		Longstaff-Schwartz	
		computed price $u(\boldsymbol{x}^0, 0)$	percent error	computed price $u(\boldsymbol{x}^0, 0)$	percent error
90	5.6322	5.6154	0.30%	OOM	OOM
100	9.9345	9.9187	0.16%	OOM	OOM
110	15.6491	15.6219	0.17%	OOM	OOM

Table 6.1: Multi-dimensional geometric average call options: Computed prices at $t = 0$, i.e., $u(\boldsymbol{x}^0, 0)$. OOM means "out-of-memory".

For the proposed method, the computed prices are accurate up to 2 decimal places; the percent errors are bounded by 0.34%, and remain approximately the same as the dimension increases. As a comparison, for the Longstaff-Schwartz method, the percent errors deteriorate from 1% to 9% as the dimension increases from 7 to 20. If we keep increasing the dimension towards 100, the Longstaff-Schwartz method encounters an out-of-memory error, because, at $d = 100$, it requires storing $\binom{d+\chi}{d} CM = 3.3 \times 10^{12}$ floating point numbers, or around 23TB of memory.

The Longstaff-Schwartz algorithm combined with the approaches in [151, 33] can be used to compute the deltas at $t = 0$. Table 6.2 compares the deltas at $t = 0$ computed by our proposed approach with the ones computed by the Longstaff-Schwartz algorithm.

(i) 7-dimensional geometric average call option

x_i^0	exact delta $\nabla u(\boldsymbol{x}^0, 0)$	proposed method		Longstaff-Schwartz
		computed delta $\nabla u(\boldsymbol{x}^0, 0)$	percent error	percent error
90	(0.0523,$\cdots$,0.0523)	(0.0516,$\cdots$,0.0516)	1.2%	1.2%
100	(0.0722,$\cdots$,0.0722)	(0.0710,$\cdots$,0.0710)	1.7%	1.6%
110	(0.0912,$\cdots$,0.0912)	(0.0901,$\cdots$,0.0901)	1.2%	1.4%

(ii) 13-dimensional geometric average call option

x_i^0	exact delta $\nabla u(\boldsymbol{x}^0, 0)$	proposed method		Longstaff-Schwartz
		computed delta $\nabla u(\boldsymbol{x}^0, 0)$	percent error	percent error
90	(0.0279,$\cdots$,0.0279)	(0.0277,$\cdots$,0.0277)	0.76%	5.4%
100	(0.0387,$\cdots$,0.0387)	(0.0384,$\cdots$,0.0384)	0.83%	3.7%
110	(0.0492,$\cdots$,0.0492)	(0.0486,$\cdots$,0.0486)	1.1%	2.6%

(iii) 20-dimensional geometric average call option

x_i^0	exact delta $\nabla u(\boldsymbol{x}^0, 0)$	proposed method		Longstaff-Schwartz
		computed delta $\nabla u(\boldsymbol{x}^0, 0)$	percent error	percent error
90	(0.0180,$\cdots$,0.0180)	(0.0179,$\cdots$,0.0179)	0.70%	12.7%
100	(0.0251,$\cdots$,0.0251)	(0.0248,$\cdots$,0.0248)	1.2%	8.3%
110	(0.0320,$\cdots$,0.0320)	(0.0316,$\cdots$,0.0316)	1.2%	6.8%

(iv) 100-dimensional geometric average call option

x_i^0	exact delta $\nabla u(\boldsymbol{x}^0, 0)$	proposed method		Longstaff-Schwartz
		computed delta $\nabla u(\boldsymbol{x}^0, 0)$	percent error	percent error
90	(0.00359,$\cdots$,0.00359)	(0.00357,$\cdots$,0.00357)	0.58%	OOM
100	(0.00502,$\cdots$,0.00502)	(0.00495,$\cdots$,0.00495)	1.3%	OOM
110	(0.00639,$\cdots$,0.00639)	(0.00631,$\cdots$,0.00631)	1.3%	OOM

Table 6.2: Multi-dimensional geometric average call options: Computed deltas at $t = 0$, i.e., $\nabla u(\boldsymbol{x}^0, 0)$. Note that all the reported deltas in the table are length-d vectors where all the elements are the same. The column "Longstaff-Schwartz" is the Longstaff-Schwartz method combined with [151, 33]. OOM means "out-of-memory".

Geometric average call option

x_i^0	proposed method				Longstaff-Schwartz			
	$d=7$	$d=13$	$d=20$	$d=100$	$d=7$	$d=13$	$d=20$	$d=100$
90	0.96	0.95	0.96	0.95	0.72	0.56	0.42	OOM
100	0.95	0.95	0.97	0.97	0.75	0.61	0.47	OOM
110	0.98	0.96	0.96	0.97	0.78	0.65	0.51	OOM

Table 6.3: Multi-dimensional geometric average call options: The f1-score of the exercise boundary classification. OOM means "out-of-memory".

For the Longstaff-Schwartz algorithm, as the dimension increases from 7 to 20, the percent errors of the deltas worsen from 1.6% to 12.7%; as the dimension continues to increase towards 100, an out-of-memory error occurs. However, for our proposed method, the computed deltas are accurate up to 3 decimal places; the percent errors do not increase with the dimension and stay below 1.7%.

Furthermore, we compare the exercise boundaries computed by the proposed neural network approach with the ones computed by the Longstaff-Schwartz approach. Table 6.3 evaluates the f1-score of the exercise boundary classification, as defined in (6.52). For the proposed method, the f1-score remains around 0.95-0.98 as the dimension increases from 7 to 100. For the Longstaff-Schwartz algorithm, the f1-score drops from 0.78 to 0.42 as the dimension increases from 7 to 20. This illustrates a more precise exercise boundary determined by our proposed algorithm.

Figure 6.5 visualizes the exercise boundaries computed by both algorithms. In order to visualize this, we start with $(\boldsymbol{x}^0, t^0) = (1.1K, 0)$ and use the SDE (6.14)-(6.15) to generate sample points on the entire spacetime, i.e., $\{(\boldsymbol{X}_m^n, t^n) \,|\, n = 0, ..., N; m = 1, ..., M\}$; we classify each sample point using either our proposed method, i.e., (6.42), or the Longstaff-Schwartz method; then we project these $(d+1)$-dimensional points onto the 2-dimensional points $\{(s'^n_m, t^n)\}$, where $s'^n_m = \left(\prod_{i=1}^d (X_i)_m^n\right)^{1/d}$ is the geometric average of the underlying asset prices $\boldsymbol{X}_m^n$. We use bold dark blue to mark the sample points that should be exercised but are misclassified as continued, and bold dark red to mark the ones that should be continued but are misclassified as exercised. The plots show that the proposed neural network approach (top left and bottom left) has fewer misclassified sample points than the Longstaff-Schwartz approach (top right and bottom right). In other words, the proposed neural network approach yields more precise exercise boundaries.

Example 6.2 (Confidence intervals by the proposed method)**.** We repeat the experiments of computing the prices and deltas at $t = 0$ (Tables 6.1-6.2) for 9 times. Tables 6.4-6.5

report the mean values of the computed prices and deltas, and the corresponding 95% T-statistic confidence intervals. The last columns of the tables show that, for both the prices and the deltas, the deviations from the mean values remain a constant of ±0.2% as

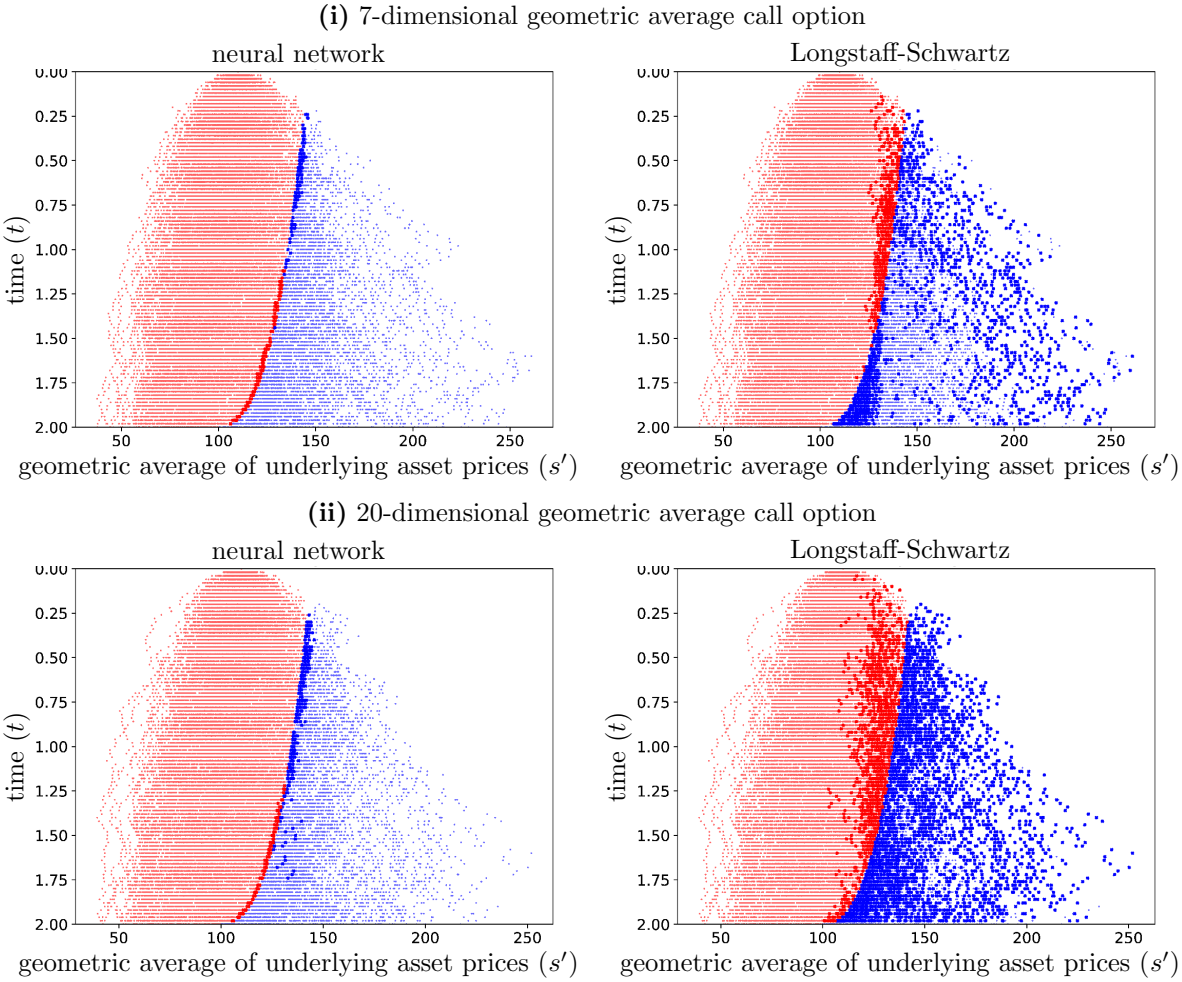


Figure 6.5: Multi-dimensional geometric average call options: Comparison of exercise boundaries between the proposed neural network approach (top left and bottom left) and the Longstaff-Schwartz approach (top right and bottom right). All blue points: sample points that should be exercised; all red points: sample points that should be continued; bold dark blue points: sample points that should be exercised but are misclassified as continued; bold dark red points: sample points that should be continued but are misclassified as exercised.

geometric average call option, $x_i^0 = 100$

d	exact price $u(\boldsymbol{x}^0, 0)$	mean of computed prices	percent error	95% CI
7	10.2591	10.2468	0.12%	$\pm$0.0161 ($\pm$0.16%)
13	10.0984	10.0822	0.16%	$\pm$0.0201 ($\pm$0.20%)
20	10.0326	10.0116	0.21%	$\pm$0.0173 ($\pm$0.17%)
100	9.9345	9.9163	0.18%	$\pm$0.0038 ($\pm$0.04%)

Table 6.4: Multi-dimensional geometric average call options: mean values and 95% T-statistic confidence intervals (CIs) of the computed prices at $t = 0$, i.e., $u(\boldsymbol{x}^0, 0)$, using the proposed neural network method.

geometric average call option, $x_i^0 = 100$

d	exact delta $\nabla u(\boldsymbol{x}^0, 0)$	mean of computed deltas	percent error	95% CI of $\frac{\partial u}{\partial x_1}(\boldsymbol{x}^0, 0)$
7	(0.0722,$\cdots$,0.0722)	(0.0717,$\cdots$,0.0717)	0.67%	$\pm 1.8 \times 10^{-4}$ ($\pm$0.25%)
13	(0.0387,$\cdots$,0.0387)	(0.0384,$\cdots$,0.0384)	0.70%	$\pm 7.3 \times 10^{-5}$ ($\pm$0.19%)
20	(0.0251,$\cdots$,0.0251)	(0.0249,$\cdots$,0.0249)	0.78%	$\pm 4.2 \times 10^{-5}$ ($\pm$0.17%)
100	(0.00502,$\cdots$,0.00502)	(0.00498,$\cdots$,0.00498)	0.76%	$\pm 8.9 \times 10^{-6}$ ($\pm$0.18%)

Table 6.5: Multi-dimensional geometric average call options: mean values of the computed deltas at $t = 0$, i.e., $\nabla u(\boldsymbol{x}^0, 0)$, using the proposed neural network method, and the corresponding 95% T-statistic confidence intervals (CIs) of the first elements of deltas, i.e., $\frac{\partial u}{\partial x_1}(\boldsymbol{x}^0, 0)$.

the dimension increases.

Example 6.3 (Evaluation of computed spacetime prices and deltas by the proposed method)**.** Our proposed algorithm yields not only the prices and deltas at $t = 0$, but also the prices and deltas on the entire spacetime, which are directly extracted from the output of the neural networks. We emphasize that the computation of spacetime prices and deltas using the Longstaff-Schwartz method is infeasible. The reason is that using the Longstaff-Schwartz method to compute prices and deltas on the entire spacetime would require repeating the algorithm at *every* sample point, noting that the Longstaff-Schwartz method at one sample point is already non-trivial. We also remark that although one may consider using the Longstaff-Schwartz regressed values as an estimate of the spacetime prices, Figure 1 in [26] shows that using such regressed values as the spacetime solution is inaccurate.

First we evaluate the absolute and percent errors of the spacetime price $u(\boldsymbol{x}, t)$ and the derivative $\frac{\partial u}{\partial s'}(s', t)$ computed by our proposed method, where s' is defined in (6.53). Here we evaluate the errors of the derivative $\frac{\partial u}{\partial s'}(s', t)$ instead of the delta $\nabla u(\boldsymbol{x}, t)$, because the

exact values of the former can be computed by finite difference method spacetime-wise, but not the latter. Table 6.6 shows that the absolute errors of the spacetime prices and derivatives are around 0.04-0.07 and 0.01 respectively, or in other words, the spacetime prices and derivatives are accurate up to 2 decimal places; the percent errors are less than 1.2% and 3.8%, respectively.

To visualize the spacetime solutions, we consider the 100-dimensional case, select three time slices $t = 0.5$, 1.0, 1.5, and project the 100-dimensional sample points of $u(\boldsymbol{x}, t)$ and $\nabla u(\boldsymbol{x}, t)$ to 1-dimensional points of $u(s', t)$ and $\frac{\partial u}{\partial s'}(s', t)$, as shown in Figure 6.6. The spacetime option prices and deltas computed by the proposed neural network approach (the blue/red dots) agree well with the exact solutions by finite difference methods (black lines). We note that small fluctuations exist for the computed spacetime deltas (right subfigures), especially near the strike price $K = 100$. This is expected, as the deltas of the

(i) 7-dimensional geometric average call option

x_i^0	spacetime price $u(\boldsymbol{x}, t)$		spacetime derivative $\frac{\partial u}{\partial s'}(s', t)$	
	absolute error	percent error	absolute error	percent error
90	0.0688	1.2%	0.0102	3.3%
100	0.0545	0.54%	0.0102	2.3%
110	0.0450	0.29%	0.0092	1.6%

(ii) 13-dimensional geometric average call option

x_i^0	spacetime price $u(\boldsymbol{x}, t)$		spacetime derivative $\frac{\partial u}{\partial s'}(s', t)$	
	absolute error	percent error	absolute error	percent error
90	0.0540	0.94%	0.0101	3.3%
100	0.0475	0.48%	0.0106	2.4%
110	0.0465	0.30%	0.0093	1.6%

(iii) 20-dimensional geometric average call option

x_i^0	spacetime price $u(\boldsymbol{x}, t)$		spacetime derivative $\frac{\partial u}{\partial s'}(s', t)$	
	absolute error	percent error	absolute error	percent error
90	0.0567	1.00%	0.0115	3.7%
100	0.0455	0.46%	0.0111	2.5%
110	0.0397	0.26%	0.0090	1.6%

(iv) 100-dimensional geometric average call option

x_i^0	spacetime price $u(\boldsymbol{x}, t)$		spacetime derivative $\frac{\partial u}{\partial s'}(s', t)$	
	absolute error	percent error	absolute error	percent error
90	0.0534	0.96%	0.0117	3.8%
100	0.0458	0.47%	0.0107	2.4%
110	0.0480	0.31%	0.0099	1.7%

Table 6.6: Multi-dimensional geometric average call options: Spacetime prices and deltas (in terms of absolute and percent errors) computed by our proposed method.

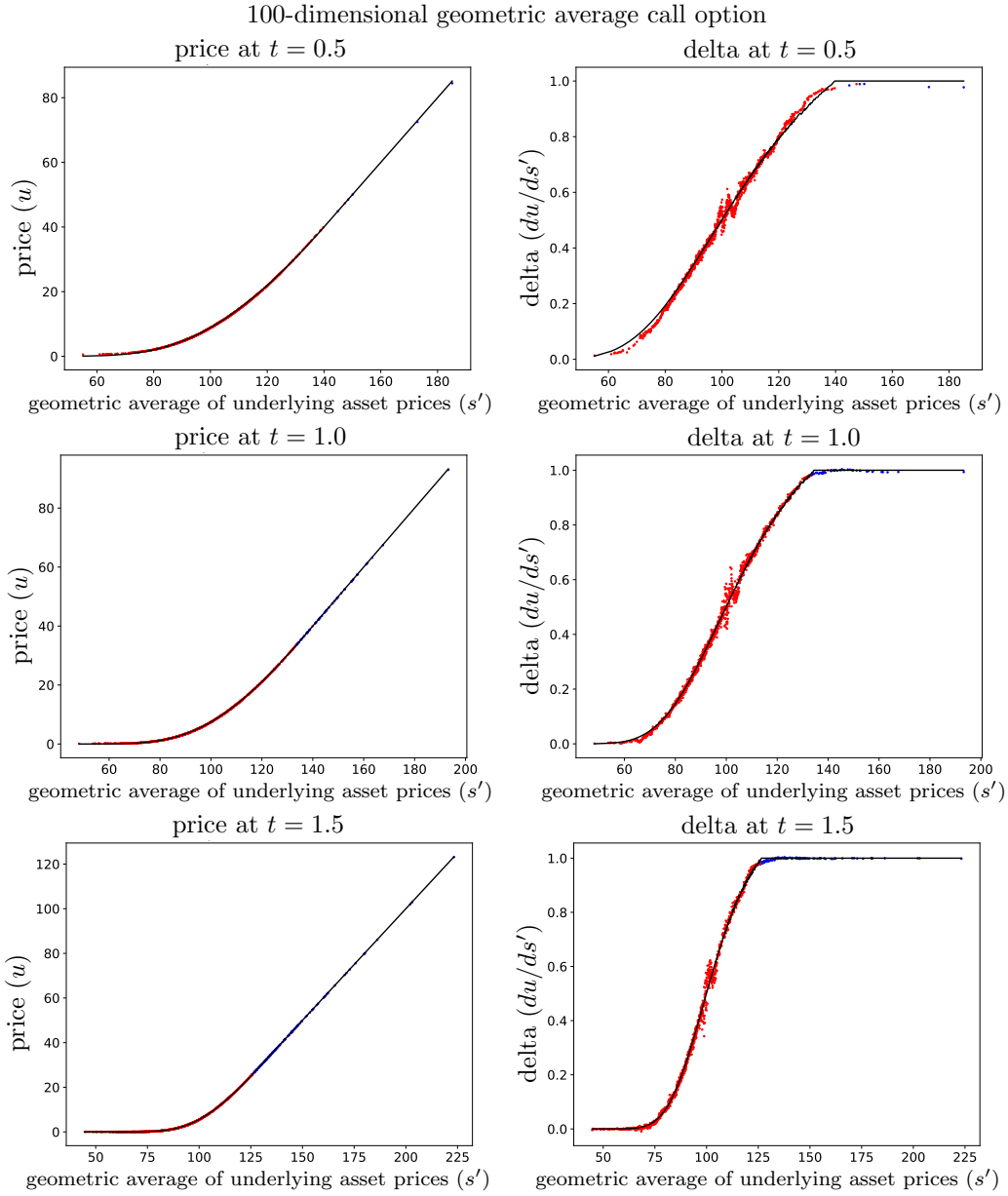


Figure 6.6: 100-dimensional geometric average call option: Prices (left subfigures) and deltas (right subfigures) computed by the proposed neural network approach at t =0.5, 1.0, 1.5. The blue/red dots are neural network output values of the exercised/continued sample points. The black lines are the exact solutions computed by finite difference methods.

payoff functions are discontinuous at the strike price. Smoothing the payoff, as described in Section 6.4.3, can mitigate this issue, although it does not eliminate the fluctuations.

Example 6.4 (Comparison between our proposed method and the method in [142])**.** First we compare the computed prices at $t = 0$; see Table 6.7. Up to 200 dimension is tested. In particular, by comparing the last two columns of the table, we observe that the percent errors computed by our method are bounded by 0.17%, while the ones computed by [142] are bounded by 0.22%.

Next we compare the computed spacetime prices by the two approaches. Figure 6.7 compares the absolute errors of the spacetime prices. To plot the figure, we start with $(\boldsymbol{x}^0, t^0) = (K, 0)$ and use the SDE (6.14)-(6.15) to generate sample points on the entire spacetime, i.e., $\{(\boldsymbol{X}_m^n, t^n) \,|\, n = 0, ..., N; m = 1, ..., M\}$. We compute the error at each sample point, $e(\boldsymbol{X}_m^n, t^n) \equiv |u(\boldsymbol{X}_m^n, t^n) - u_{exact}(\boldsymbol{X}_m^n, t^n)|$. Then we project $\{e(\boldsymbol{X}_m^n, t^n)\}$ from $(d+1)$-dimensional to 2-dimensional space and get the sample points $\{e(s'^n_m, t^n)\}$, where s'^n_m is the geometric average of $\boldsymbol{X}_m^n$. From the discrete data points $\{e(s'^n_m, t^n)\}$, we use interpolation to obtain a continuous error function $e(s', t)$ and represent it by a heatmap (also known as filled contour plot), where the x and y axes are the time t and the geometric average s', and the color represents the magnitude of $e(s', t)$. The red, green and blue areas represent the areas where the samples have large, median and small errors, respectively. The white areas are the areas outside the convex hull of the sampled points, where no value of $e(s', t)$ can be interpolated from the sampled $\{e(s'^n_m, t^n)\}$. We remark that this plotting procedure is the same as [142]. Indeed, Figure 6.7(ii) is directly taken from [142]. In addition, we note that the colored areas of Figure 6.7 (i) and (ii) are not exactly the same. This is because the points on (or near) the boundary of the convex hull are only sampled with a small probability and would have a large variation under the two independent stochastic sampling processes that generate the two subplots.

geometric average call option, $x_i^0 = 100$

d	exact price $u(\boldsymbol{x}^0, 0)$	proposed method		method in [142]
		computed price $u(\boldsymbol{x}^0, 0)$	percent error	percent error
3	10.7185	10.7368	0.17%	0.05%
20	10.0326	10.0180	0.15%	0.03%
100	9.9345	9.9187	0.16%	0.11%
200	9.9222	9.9088	0.14%	0.22%

Table 6.7: Multi-dimensional geometric average call options: Computed prices at $t = 0$, i.e., $u(\boldsymbol{x}^0, 0)$. $x_i^0 = 100$. The percent errors reported in Table 1 of [142] are also included in the last column of this table.

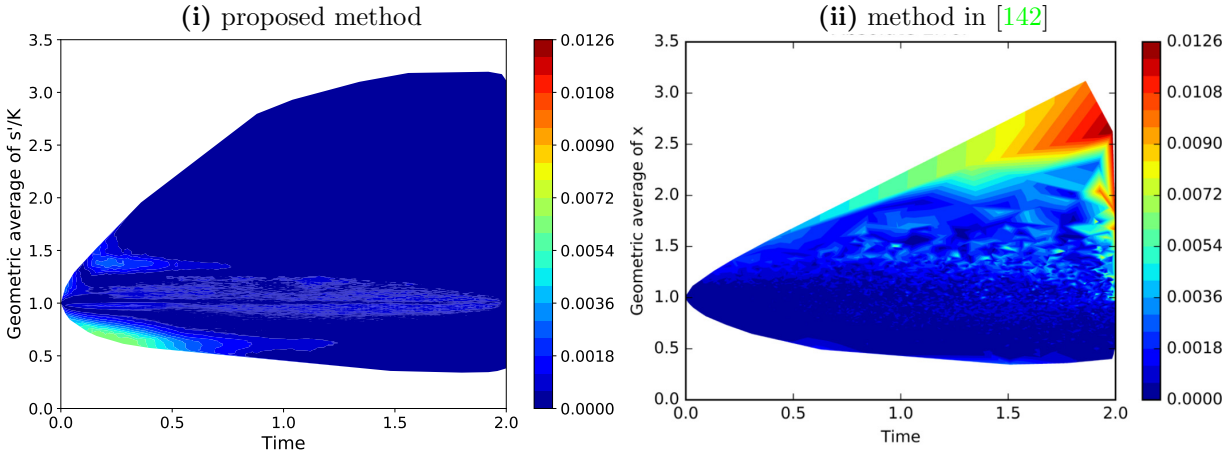


Figure 6.7: 20-dimensional geometric average call options: Heatmaps of the absolute errors of the computed spacetime prices. **(i)** absolute error computed by the proposed approach; **(ii)** absolute error computed by [142].

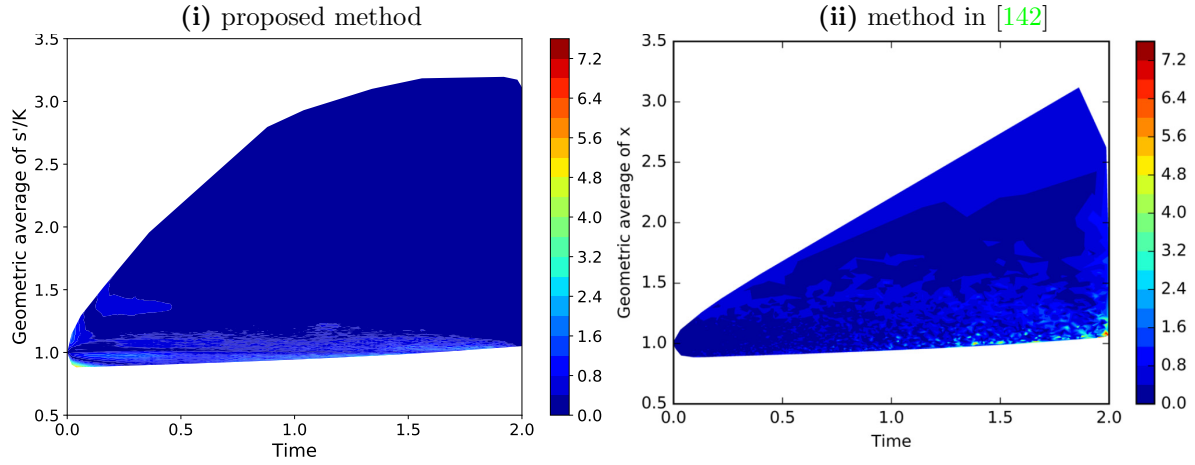


Figure 6.8: 20-dimensional geometric average call options: Heatmaps of the percent errors of the computed spacetime prices. **(i)** percent error computed by the proposed approach; **(ii)** percent error computed by [142].

Figure 6.7(i) shows that the absolute error computed by our proposed approach is close to zero almost on the entire spacetime domain. The error is slightly larger near $(t, s'/K) \approx (0.2, 0.7)$ and bounded by 0.0072. The reason why the error is slightly larger near $t = 0$ is that our proposed approach computes the price in a backward manner, and hence the error may accumulate near $t = 0$. As a comparison, Figure 6.7(ii) shows that the

error computed by [142] has a larger error in most of the spacetime domain. In particular, the error reaches 0.0126 near $(t, s'/K) \approx (2.0, 2.7)$, which is larger than the upper bound of our error, 0.0072.

Figure 6.8 compares the heatmaps of the corresponding percent errors. Following [142], the percent errors are only plotted for the areas where $|u_{exact}(s', t)| > 0.05$. Similar to Figure 6.7, Figure 6.8(i) shows that our proposed approach yields zero error almost everywhere, except that near $(t, s'/K) \approx (0.05, 0.9)$ the error reaches 5.6%; Figure 6.8(ii) shows that the approach in [142] results in a larger error, particularly near $(t, s'/K) \approx (2.0, 1.05)$, where the error reaches 7.2%.

We emphasize that [142] does not compute deltas, whereas our proposed method does yield the deltas. Table 6.8 reports the deltas at $t = 0$ computed by our proposed method. The percent errors are bounded by 1.3%, and remain approximately the same as the dimension increases. Our approach also computes spacetime deltas, which has been discussed in Example 6.3 and is thus skipped here.

Example 6.5 (Delta hedging)**.** We perform delta hedging simulations over the period $[0, T]$ with our proposed method. We evaluate the quality of the approach using the distribution

geometric average call option, $x_i^0 = 100$

d	exact delta $\nabla u(\boldsymbol{x}^0, 0)$	proposed method	
		computed delta $\nabla u(\boldsymbol{x}^0, 0)$	percent error
3	(0.1702,$\cdots$,0.1702)	(0.1683,$\cdots$,0.1683)	1.1%
20	(0.0251,$\cdots$,0.0251)	(0.0248,$\cdots$,0.0248)	1.2%
100	(0.00502,$\cdots$,0.00502)	(0.00495,$\cdots$,0.00495)	1.3%
200	(0.00251,$\cdots$,0.00251)	(0.00250,$\cdots$,0.00250)	0.53%

Table 6.8: Multi-dimensional geometric average call options: Computed deltas at $t = 0$, i.e., $\nabla u(\boldsymbol{x}^0, 0)$. $x_i^0 = 100$.

geometric average call option

x_i^0	$d = 7$		$d = 13$		$d = 20$		$d = 100$	
	mean	std	mean	std	mean	std	mean	std
90	-0.0023	0.1788	0.0017	0.1827	-0.0003	0.1877	-0.0021	0.1908
100	-0.0016	0.1159	0.0021	0.1170	-0.0007	0.1184	-0.0010	0.1184
110	-0.0001	0.0757	0.0013	0.0755	0.0005	0.0751	-0.0009	0.0763

Table 6.9: Multi-dimensional geometric average call options: computed means and standard deviations of the relative P&Ls, subject to 100 hedging intervals.

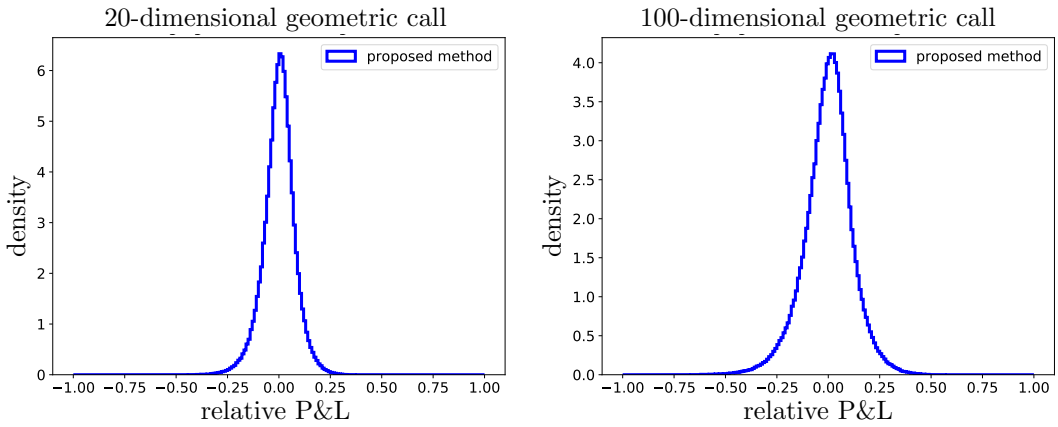


Figure 6.9: Multi-dimensional geometric call options: Distributions of the relative P&Ls computed by the proposed neural network approach, subject to 100 hedging intervals.

of the relative profit and loss [65, 90]:

$$\text{Relative P\&L} \equiv \frac{e^{-rT}\Pi_T}{u(\boldsymbol{x}^0, 0)}, \tag{6.54}$$

where Π_T is the balance of an initially-zero hedging portfolio at the expiry T. For perfect hedging, the relative P&L should be a Dirac delta function. Due to the discretization of time, the relative P&L would be close to a normal distribution, where the mean is zero and the standard deviation is a small value depending on Δt. We emphasize that the computation of the relative P&L must use both prices and deltas on the entire spacetime. Hence, none of the existing methods referenced in this chapter, except our proposed method, are designed to compute the relative P&L.

Table 6.9 shows the means and the standard deviations of the relative P&Ls for all the 720000 simulation paths, computed by our proposed method. The reported values are indeed close to zero. Figure 6.9 illustrates the distributions of the relative P&Ls. The resulting distributions are indeed approximately normal distributions with zero means. These results confirm the accuracy of the spacetime prices and the spacetime deltas computed by the proposed method.

6.7.2 Multi-dimensional max and basket options

Multi-dimensional max options and basket options are common in practical applications. In this section, we report simulation results for these types of options.

Example 6.6 (2-dimensional max call option)**.** Consider the 2-dimensional max call option from Table 3 of [34], where the payoff function is (6.4), and the parameters are $\rho = 0.3$, $\sigma = 0.2$, $r = 0.05$, $\delta = 0.1$, $T = 1$. The reason to consider this example is that the exact prices and deltas are available spacetime-wise. More specifically, we approximate the exact prices and deltas by the Crank-Nicolson finite difference method with 1000 timesteps and

2-dimensional max call option

x_i^0	exact price $u(\boldsymbol{x}^0, 0)$	proposed method		Longstaff-Schwartz	
		computed price $u(\boldsymbol{x}^0, 0)$	percent error	computed price $u(\boldsymbol{x}^0, 0)$	percent error
90	4.2122	4.1992	0.31%	4.1748	0.89%
100	9.6333	9.6080	0.26%	9.5646	0.71%
110	17.3487	17.3313	0.10%	17.2751	0.42%

Table 6.10: 2-dimensional max call option: Computed prices at $t = 0$, i.e., $u(\boldsymbol{x}^0, 0)$.

2-dimensional max call option

x_i^0	exact delta $\nabla u(\boldsymbol{x}^0, 0)$	proposed method		Longstaff-Schwartz
		computed delta $\nabla u(\boldsymbol{x}^0, 0)$	percent error	percent error
90	(0.2062, 0.2062)	(0.2025, 0.2019)	1.9%	5.2%
100	(0.3338, 0.3338)	(0.3300, 0.3324)	0.84%	4.4%
110	(0.4304, 0.4304)	(0.4252, 0.4277)	0.96%	3.3%

Table 6.11: 2-dimensional max call option: Computed deltas at $t = 0$, i.e., $\nabla u(\boldsymbol{x}^0, 0)$.

2-dimensional max call option

x_i^0	spacetime price $u(\boldsymbol{x}, t)$		spacetime delta $\nabla u(\boldsymbol{x}, t)$	
	absolute error	percent error	absolute error	percent error
90	0.0563	1.3%	0.0155	4.9%
100	0.0828	0.85%	0.0180	3.4%
110	0.0678	0.39%	0.0207	3.0%

Table 6.12: 2-dimensional max call option: Spacetime prices and deltas (in terms of absolute and percent errors) computed by our proposed method.

2-dimensional max call option

s_i^0	proposed method	Longstaff-Schwartz
90	0.93	0.74
100	0.95	0.76
110	0.94	0.79

Table 6.13: 2-dimensional max call option: The f1-score of the exercise boundary classification.

2-dimensional max call option

s_i^0	finite difference		proposed method	
	mean	std	mean	std
90	0.0025	0.1683	0.0022	0.1932
100	0.0014	0.0894	0.0016	0.0990
110	0.0011	0.0544	0.0016	0.0614

Table 6.14: 2-dimensional max call option: Means and standard deviations of the relative P&Ls by finite difference versus by the proposed method, subject to 100 hedging intervals.

2049×2049 space grid points. Hence, we can again benchmark the values computed by our approach with the exact ones.

Using our proposed method, the percent errors of the computed prices at $t = 0$ are less than 0.31% (Table 6.10); the percent errors of the computed deltas at $t = 0$ are less than 1.9% (Table 6.11). These errors are smaller than the corresponding ones computed by the Longstaff-Schwartz method. In addition, the percent errors of the computed spacetime prices and deltas are less than 1.3% and 4.9% (Table 6.12).

Here we also compare the exercise boundary computed by the proposed approaches with the one computed by the Longstaff-Schwartz method. Table 6.13 shows that the f1-scores computed by our proposed method are around 0.94, higher than the ones computed by the Longstaff-Schwartz algorithm (around 0.76). Figure 6.10 plots the exercise boundaries at the time slices $t = 0.75$ and 0.5. Similar to Figure 6.5, here the misclassified sample points are highlighted by dark cross markers, and we observe again that the proposed neural network approach has fewer misclassified points than the Longstaff-Schwartz method. Both Table 6.13 and Figure 6.10 illustrate a more accurate exercise boundary determined by our proposed method than by the Longstaff-Schwartz method.

In addition, we compute the relative P&Ls by the finite difference method[7] and compare them with the values computed by our approach. Table 6.14 and Figure 6.11 show the means, standard deviations and the distributions of the relative P&Ls computed by the proposed approach versus by finite difference methods. The results computed by the proposed approach are similar to the ones computed by finite difference methods. This again verifies the accuracy of the spacetime prices and deltas computed by our proposed algorithm.

Example 6.7 (5-dimensional max call option)**.** We study the 5-dimensional max call option from Table 3.5 of [64], where $\rho_{i,j} = 0$, $\sigma = 0.2$, $r = 0.05$, $\delta = 0.1$, $T = 3$. We note

[7]We note that even though finite difference methods yield nearly exact spacetime prices and deltas, due to the finite number of hedging intervals, the resulting relative P&Ls are not a Dirac delta distribution.

that unlike the previous experiments, here the exact solutions are not available. Table 6.15 reports the option prices and deltas at $t = 0$ computed by the proposed method. The table also includes the Longstaff-Schwartz prices reported in [64]. The prices given by the proposed algorithm and the Longstaff-Schwartz method differ by 10^{-2}. We note that the Longstaff-Schwartz method is a low-biased method due to its sub-optimal computed exercise boundary, as explained in [117, 64]. The proposed algorithm gives slightly higher

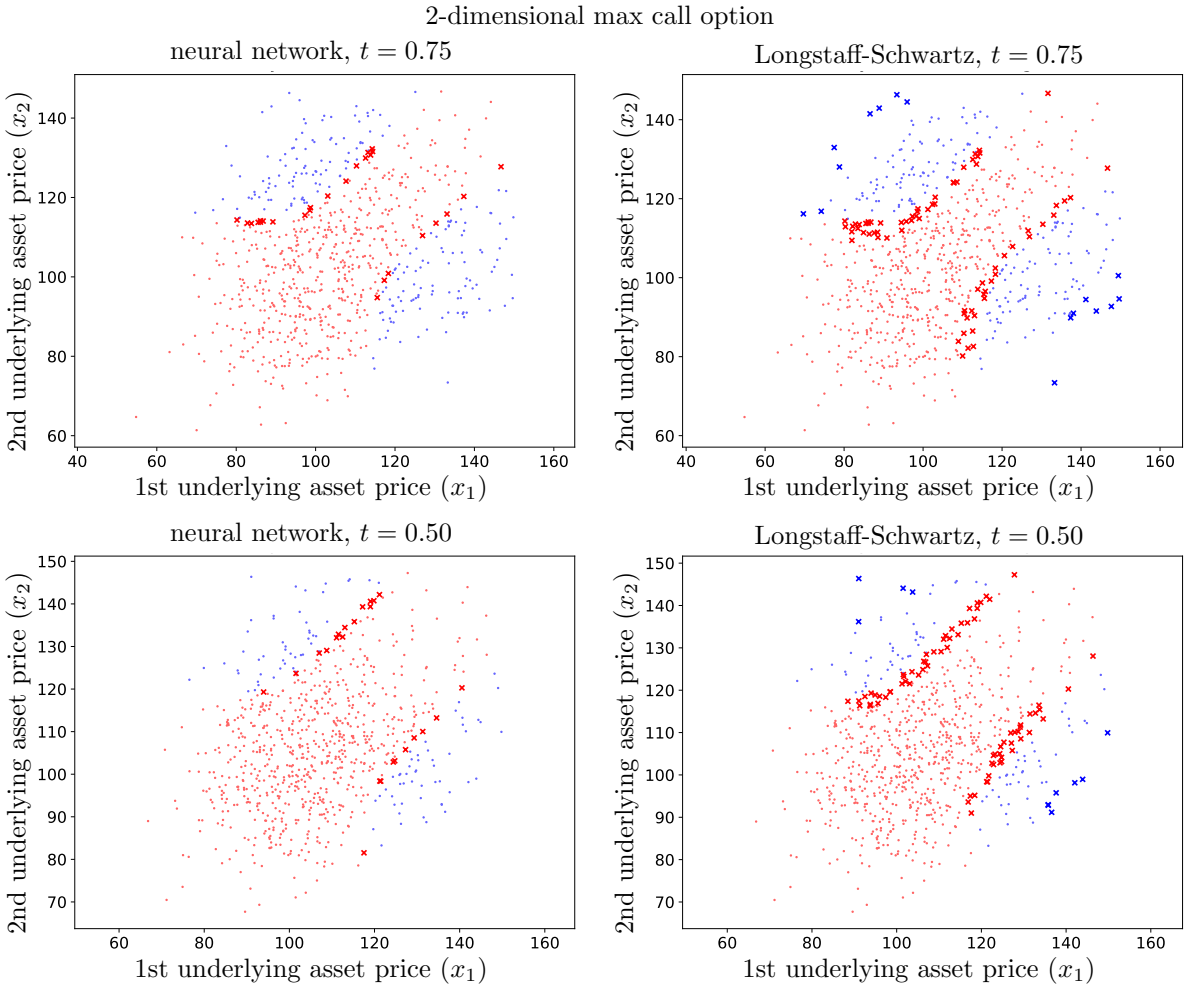


Figure 6.10: 2-dimensional max call option: Comparison of exercise boundaries between the proposed neural network approach (top left and bottom left) and the Longstaff-Schwartz approach (top right and bottom right). Only the time slices of t =0.75 and 0.5 are plotted. The meaning of blue and red markers are the same as in Figure 6.5.

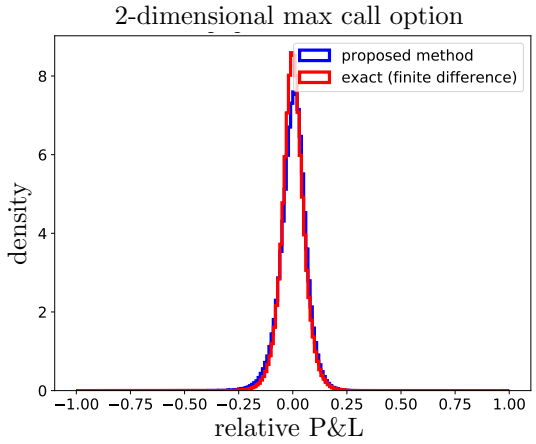


Figure 6.11: 2-dimensional max call option: Comparison of the distributions of the relative P&Ls computed by the proposed neural network approach (blue) versus by finite difference method (red), subject to 100 hedging intervals.

5-dimensional max call option

x_i^0	computed price $u(\boldsymbol{x}^0, 0)$		computed delta $\nabla u(\boldsymbol{x}^0, 0)$ by proposed method
	proposed method	Longstaff-Schwartz	
90	16.8896	16.76	(0.1728, 0.1732, 0.1747, 0.1754, 0.1738)
100	26.4876	26.28	(0.2017, 0.2004, 0.1998, 0.2071, 0.2041)
110	37.0996	36.89	(0.2157, 0.2198, 0.2190, 0.2149, 0.2202)

Table 6.15: 5-dimensional max call option: Computed prices and deltas at $t = 0$, i.e., $u(\boldsymbol{x}^0, 0)$ and $\nabla u(\boldsymbol{x}^0, 0)$. The column "Longstaff-Schwartz" is the Longstaff-Schwartz prices reported in [64].

prices.

Example 6.8 (50-dimensional basket call option)**.** As the final example, we consider the 50-dimensional basket call option from Section 3.2.2 of [75], where $\rho_{i,j} = 0.3026$, $\sigma = 0.2$, $r = 0.03$, $\delta = 0.07$, $T = 0.5$. We note that the exact solutions (including both prices and deltas) are unavailable. The price at $t = 0$, or more precisely, $u(110, 0)$, reported in [75], is 9.7. Our proposed algorithm yields 9.9774, which is close to the value in [75]. We note that, unlike [75], our proposed algorithm also yields the delta at $t = 0$, the spacetime price and the spacetime delta. For the interested readers, the delta $\nabla u(110, 0)$ computed by our method is a length-50 vector, where all the entries are approximately equal, and where the mean and the standard deviation of the entries are 0.0200 and 1.95×10^{-4}, respectively.

6.7.3 Computational cost

We compare the computational time of the proposed neural network algorithm with the Longstaff-Schwartz method; see Table 6.16. We note that the computation of the proposed algorithm is split across 4 GPUs. When $d \geq 50$, we note that, as the dimension doubles, the computational time is slightly more optimal than a four-fold increase. This agrees with the theoretical $O(Md^2)$ computational time analyzed in Section 6.4.5. When $d \leq 50$, we note that the computational time is close to constant. This is due to the fixed computational costs including the constructions of network graphs, and the communication among CPUs / GPUs, etc.

As a comparison, the computational time of the Longstaff-Schwartz method grows at a higher rate as the dimension increases. This is due to the $O(Md^8)$ computational time when the monomial basis is degree-4 ($\chi = 4$). When $d \geq 20$, the proposed algorithm outperforms the Longstaff-Schwartz method. In particular, when $d \geq 50$, the Longstaff-Schwartz method fails due to the out-of-memory error, while the proposed algorithm succeeds in the evaluation.

d	1	7	13	20	50	100	200
Proposed method	9515	10119	10571	10918	13273	24960	74449
Longstaff-Schwartz	22	310	2299	30057	OOM	OOM	OOM

Table 6.16: Comparison of the computational time (seconds) between the proposed neural network method and the Longstaff-Schwartz method. For the Longstaff-Schwartz method, the computation fails for d =50, 100, 200 due to the OOM (out-of-memory) error.

6.8 Conclusion

We propose a neural network framework for high-dimensional American option problems. Our algorithm minimizes the residual of the backward stochastic differential equation that couples both prices and deltas. The neural network is designed to learn the differences between the price functions of the adjacent timesteps. We improve the algorithm by various techniques, including feature selection, weight reuse, redefining training input u^{n+1}, etc. The proposed algorithm yields not only the prices and deltas at $t = 0$, but also the prices and deltas on the entire spacetime. The cost of the proposed algorithm grows quadratically with the dimension d, which mitigates the curse of dimensionality. In particular, our algorithm outperforms the Longstaff-Schwartz algorithm when $d \geq 20$.

Chapter 7

Conclusion

This thesis proposes multiple numerical methods for solving HJB equations with applications in mean field games, image registration and American options. First, Chapters 2-5 study HJB equations where the spatial dimension is less than three. Our approach to solving such HJB equations involves discretization of PDEs, applying nonlinear solvers for discretized systems, and speeding up nonlinear solvers.

An appropriate discretization is critical for a discrete solution to converge towards the continuous viscosity solution of an HJB equation. When an HJB equation contains cross derivatives, we propose a mixed discretization. Furthermore, we prove that the mixed discretization is consistent, stable, monotone, and fulfills the strong comparison principle, and hence converges to the viscosity solution. We demonstrate the optimal $O(h^2)$ convergence rate of the mixed scheme, which is a significant improvement over the pure semi-Lagrangian scheme.

Discretization gives rise to nonlinear discretized systems. In general, we use policy iteration to solve the systems. Depending on applications, the solver may require some adaptations. For instance, in mean field games, where a backward HJB equation is coupled with a forward KFP equation, we consider a spacetime nonlinear solver; in image registration, where the HJB differential operator is no longer linear under a fixed control and where Jacobians become singular due to our periodic boundary condition, we propose using the Levenberg-Marquardt algorithm.

In order to speed up solving the nonlinear discretized systems, we proposed various multigrid methods. The proposed multigrid methods belong to either global linearization methods or FAS methods. Under the spacetime formulation, we propose hybrid full and semi coarsening strategy. In the presence of convection, we propose subtracting artificial

viscosity from the direct discretization coarse grid operators, which yields a more precise coarse grid estimated error. In the presence of wide stencil discretization, we propose keeping wide stencil points as coarse grid points and using injection as the restriction. Multiple numerical simulations demonstrate that our multigrid methods achieve mesh-independent convergence rates, and faster convergence rates than existing approaches in the literature. In addition, numerical simulations suggest that, if an FAS scheme can be successfully devised, it converges faster than the corresponding global linearization method, since the FAS only involves one layer of iteration (as opposed to the global linearization's outer-inner iteration).

Furthermore, in Chapter 6, we study high-dimensional HJB equations, where conventional discretization approaches suffer from the curse of dimensionality. Our approach is to use a neural network formulation. We propose minimizing the residual norm of an equivalent BSDE rather than the residual norm of an HJB equation itself, which avoids the cost of Hessian tensors. We devise a novel architecture of neural network, which utilizes domain knowledge of American option problems. As a result, our approach simultaneously and accurately computes the solution and its gradient, i.e., the price and the delta of an American option, on the entire spacetime; and, our approach addresses the curse of dimensionality. Numerical simulations show that our proposed approach solves American option prices and deltas in as high as 200 dimension, whereas the state-of-the-art Longstaff-Schwartz method fails to solve the problems due to the out-of-memory error and the worse-than-quadratic cost when $d \geq 20$. These demonstrate the capability of neural network formulation for solving high-dimensional HJB equations.

This thesis has several unexplored and unresolved issues concerning numerical solution of HJB equations. We summarize them below as potential future works.

Regarding discretization of HJB equations, the mixed scheme proposed in Chapter 3 could be potentially extended to higher dimensional cases. Assuming that the dimension is d, the idea is to parameterize the control of the HJB equation (3.9) as $\mathbf{\Sigma}(\boldsymbol{x}) = Q(\boldsymbol{x})\Lambda(\boldsymbol{x})Q(\boldsymbol{x})^T$, where $Q(\boldsymbol{x}) \in SO(d)$ and $\Lambda(\boldsymbol{x})$ is a trace-1 non-negative diagonal matrix. Then the standard 7-point stencil discretization can be applied if $\mathbf{\Sigma}(\boldsymbol{x})$ is weakly diagonal dominant, and the semi-Lagrangian wide stencil discretization is applied otherwise. Such extension is still yet to be experimented with numerically.

Regarding multigrid methods for solving HJB equations, two aspects could be improved. One is that the artificial viscosity coarse grid operator proposed in Chapter 2 is efficient under the assumption that $(\frac{|c_1|h}{\sigma}, \frac{|c_2|h}{\sigma})$ are not much larger than 1. This may not hold if the convection is extremely large or if the diffusion is extremely small. In Example 2.2,

we have seen that applying W-cycle on the coarsest grids is a good remedy in practice. However, it is desirable to find a more elegant solution, where V-cycle can be applied on the coarsest grids and still yields an effective multigrid scheme. The other potential aspect of improvement is that Chapter 4 has only proposed a global linearization multigrid method for wide stencil discretization. Developing an FAS scheme for wide stencils is more challenging, but could potentially achieve faster convergence.

Regarding the application of image registration, there are two major limitations of the mass transport model. One is that the morphing effect may misinterpret the physics of the deformation in some applications, although the morphing effect is an intrinsic component of the mass transport model [123]. The other limitation is that the mass transport model does not recover rotation globally, although it recovers rotation locally. The reason is that a transformation field of the form $\phi^* = \nabla u$ must be curl-free, while a global rotation is not curl-free. Addressing these two limitations would require a fundamental modification of the mass transport model. The objective of this thesis is to develop numerical schemes for the mass transport model, rather than to do modelling. Hence, modifying the mass transport model is beyond the scope of this thesis. Nevertheless, a future work is to propose modification of the mass transport model, such as incorporating ideas from [86, 123, 41]. In addition, if a modified model is proposed, it would be interesting to develop efficient numerical schemes.

Regarding the neural network framework for solving high-dimensional HJB equations, we note that the computational cost of the proposed framework in Chapter 6 is quadratic (rather than linear) in the number of the timesteps N, even though a mitigation is proposed in Section 6.4.5. Consequentially, the framework can still be expensive when N is large. A potential future work is to re-design the architecture of the neural network in order to eliminate this drawback.

Last but not least, we would like to mention a booming application of HJB equations - reinforcement learning [148]. Reinforcement learning, together with supervised and unsupervised learning, are the three most fundamental categories of machine learning. Reinforcement learning is widely applied in robotic control, natural language processing, games, economics, etc. In particular, reinforcement learning is deemed as the key to the realization of Artificial Intelligence. It turns out that many reinforcement learning problems in continuous spacetime can be formulated as HJB equations [57, 149]. Solving HJB equations for reinforcement learning problems is a promising research field.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[2] Yves Achdou. Finite difference methods for mean field games. In *Hamilton-Jacobi equations: approximations, numerical analysis and applications*, volume 2074 of *Lecture Notes in Math.*, pages 1–47. Springer, Heidelberg, 2013.

[3] Yves Achdou, Fabio Camilli, and Italo Capuzzo-Dolcetta. Mean field games: numerical methods for the planning problem. *SIAM J. Control Optim.*, 50(1):77–109, 2012.

[4] Yves Achdou, Fabio Camilli, and Italo Capuzzo-Dolcetta. Mean field games: convergence of a finite difference method. *SIAM J. Numer. Anal.*, 51(5):2585–2612, 2013.

[5] Yves Achdou and Italo Capuzzo-Dolcetta. Mean field games: numerical methods. *SIAM J. Numer. Anal.*, 48(3):1136–1162, 2010.

[6] Yves Achdou and Victor Perez. Iterative strategies for solving linearized discrete mean field games systems. *Netw. Heterog. Media*, 7(2):197–217, 2012.

[7] Yves Achdou and Olivier Pironneau. *Computational methods for option pricing*, volume 30 of *Frontiers in Applied Mathematics*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2005.

[8] Marianne Akian, Pierre Séquier, and Agnès Sulem. A finite horizon multidimensional portfolio selection problem with singular transactions. In *Decision and Control, 1995., Proceedings of the 34th IEEE Conference on*, volume 3, pages 2193–2198. IEEE, 1995.

[9] Marianne Akian, Agnès Sulem, and Michael I. Taksar. Dynamic optimization of long-term growth rate for a portfolio with transaction costs and logarithmic utility. *Math. Finance*, 11(2):153–188, 2001.

[10] Roman Andreev. Preconditioning the augmented Lagrangian method for instationary mean field games with diffusion. *SIAM J. Sci. Comput.*, 39(6):A2763–A2783, 2017.

[11] Parsiad Azimzadeh. Impulse control in finance: numerical methods and viscosity solutions. *arXiv preprint arXiv:1712.01647*, 2017.

[12] Parsiad Azimzadeh and Peter A. Forsyth. Weakly Chained Matrices, Policy Iteration, and Impulse Control. *SIAM J. Numer. Anal.*, 54(3):1341–1364, 2016.

[13] Randolph E. Bank, Justin W. L. Wan, and Zhenpeng Qu. Kernel preserving multigrid methods for convection-diffusion equations. *SIAM J. Matrix Anal. Appl.*, 27(4):1150–1171, 2006.

[14] Guy Barles and Panagiotis E. Souganidis. Convergence of approximation schemes for fully nonlinear second order equations. *Asymptotic Anal.*, 4(3):271–283, 1991.

[15] Suleyman Basak and Georgy Chabakauri. Dynamic mean-variance asset allocation. *Review of Financial Studies*, 23(8):2970–3016, 2010.

[16] Christian Beck, Weinan E, and Arnulf Jentzen. Machine learning approximation algorithms for high-dimensional fully nonlinear partial differential equations and second-order backward stochastic differential equations. *Journal of Nonlinear Science*, pages 1–57, 2017.

[17] Sebastian Becker, Patrick Cheridito, and Arnulf Jentzen. Deep optimal stopping. *Journal of Machine Learning Research*, 20(74):1–25, 2019.

[18] Richard E. Bellman and Stuart E. Dreyfus. *Applied dynamic programming*. Princeton University Press, Princeton, N.J., 1962.

[19] Jean-David Benamou, Yann Brenier, and Kevin Guittet. The Monge-Kantorovitch mass transfer and its computational fluid mechanics formulation. *Internat. J. Numer. Methods Fluids*, 40(1-2):21–30, 2002. ICFD Conference on Numerical Methods for Fluid Dynamics (Oxford, 2001).

[20] Jean-David Benamou, Francis Collino, and Jean-Marie Mirebeau. Monotone and consistent discretization of the Monge-Ampère operator. *Math. Comp.*, 85(302):2743–2775, 2016.

[21] Jean-David Benamou, Brittany D. Froese, and Adam M. Oberman. Two numerical methods for the elliptic Monge-Ampère equation. *M2AN Math. Model. Numer. Anal.*, 44(4):737–758, 2010.

[22] Jürgen Bey and Gabriel Wittum. Downwind numbering: robust multigrid for convection-diffusion problems. *Appl. Numer. Math.*, 23(1):177–192, 1997. Multilevel methods (Oberwolfach, 1995).

[23] Martina Bloss and Ronald H. W. Hoppe. Numerical computation of the value function of optimally controlled stochastic switching processes by multi-grid techniques. *Numer. Funct. Anal. Optim.*, 10(3-4):275–304, 1989.

[24] Klaus Böhmer. On finite element methods for fully nonlinear elliptic equations of second order. *SIAM J. Numer. Anal.*, 46(3):1212–1249, 2008.

[25] Olivier Bokanowski, Stefania Maroso, and Hasnaa Zidani. Some convergence results for Howard's algorithm. *SIAM J. Numer. Anal.*, 47(4):3001–3026, 2009.

[26] Bruno Bouchard and Xavier Warin. Monte-Carlo valuation of American options: facts and new algorithms to improve existing methods. In *Numerical methods in finance*, volume 12 of *Springer Proc. Math.*, pages 215–255. Springer, Heidelberg, 2012.

[27] A. Brandt and I. Yavneh. On multigrid solution of high-Reynolds incompressible entering flows. *J. Comput. Phys.*, 101(1):151–164, 1992.

[28] Achi Brandt. Multi-level adaptive solutions to boundary-value problems. *Math. Comp.*, 31(138):333–390, 1977.

[29] Susanne C. Brenner, Thirupathi Gudi, Michael Neilan, and Li-yeng Sung. C^0 penalty methods for the fully nonlinear Monge-Ampère equation. *Math. Comp.*, 80(276):1979–1995, 2011.

[30] Alberto Bressan. Noncooperative differential games. *Milan J. Math.*, 79(2):357–427, 2011.

[31] Luis Briceño-Arias, Dante Kalise, Ziad Kobeissi, Mathieu Laurière, Álvaro Mateos González, and Francisco José Silva. On the implementation of a primal-dual algorithm for second order time-dependent mean field games with local couplings. *arXiv preprint arXiv:1802.07902*, 2018.

[32] Luis M Briceño-Arias, Dante Kalise, and Francisco J. Silva. Proximal methods for stationary mean field games with local couplings. *SIAM J. Control Optim.*, 56(2):801–836, 2018.

[33] Mark Broadie and Paul Glasserman. Estimating security price derivatives using simulation. *Management science*, 42(2):269–285, 1996.

[34] Mark Broadie and Paul Glasserman. Pricing American-style securities using simulation. *J. Econom. Dynam. Control*, 21(8-9):1323–1352, 1997. Computational financial modelling.

[35] Mark Broadie, Paul Glasserman, et al. A stochastic mesh method for pricing high-dimensional American options. *Journal of Computational Finance*, 7:35–72, 2004.

[36] P. A. Browne, Chris J. Budd, C. Piccolo, and M. Cullen. Fast three dimensional r-adaptive mesh redistribution. *J. Comput. Phys.*, 275:174–196, 2014.

[37] Chris J. Budd and J. F. Williams. Moving mesh generation using the parabolic Monge-Ampère equation. *SIAM J. Sci. Comput.*, 31(5):3438–3465, 2009.

[38] Hans-Joachim Bungartz and Michael Griebel. Sparse grids. *Acta Numer.*, 13:147–269, 2004.

[39] Luis A. Caffarelli. Boundary regularity of maps with convex potentials. II. *Ann. of Math. (2)*, 144(3):453–496, 1996.

[40] Elisabetta Carlini and Francisco J. Silva. A semi-Lagrangian scheme for a degenerate second order mean field game system. *Discrete Contin. Dyn. Syst.*, 35(9):4269–4292, 2015.

[41] Ken Y. K. Chan and Justin W. L. Wan. Reconstruction of missing cells by a killing energy minimizing nonrigid image registration. In *Engineering in Medicine and Biology Society (EMBC), 2013 35th Annual International Conference of the IEEE*, pages 3000–3003. IEEE, 2013.

[42] Patrick Chan and Ronnie Sircar. Bertrand and Cournot mean field games. *Applied Mathematics & Optimization*, 71(3):533–569, 2015.

[43] Rick Chartrand, Brendt Wohlberg, Kevin R. Vixie, and Erik M. Bollt. A gradient descent solution to the Monge-Kantorovich problem. *Appl. Math. Sci. (Ruse)*, 3(21-24):1071–1080, 2009.

[44] Yangang Chen and Justin W. L. Wan. Multigrid methods for convergent mixed finite difference scheme for Monge-Ampère equation. *Computing and Visualization in Science*, pages 1–15, 2017.

[45] Yangang Chen and Justin W. L. Wan. Numerical method for image registration model based on optimal mass transport. *Inverse Probl. Imaging*, 12(2):401–432, 2018.

[46] Yangang Chen and Justin W. L. Wan. Artificial viscosity joint spacetime multigrid method for Hamilton-Jacobi-Bellman and Kolmogorov-Fokker-Planck system arising from mean field games. *(submitted, under review)*, 2019.

[47] Yangang Chen and Justin W. L. Wan. Deep neural network framework based on backward stochastic differential equations for pricing and hedging American options in high dimensions. *(submitted, under review)*, 2019.

[48] Yangang Chen, Justin W. L. Wan, and Jessey Lin. Monotone mixed finite difference scheme for Monge-Ampère equation. *J. Sci. Comput.*, 76(3):1839–1867, 2018.

[49] Philippe G. Ciarlet. Discrete maximum principle for finite-difference operators. *Aequationes Math.*, 4:338–352, 1970.

[50] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

[51] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of machine learning research*, 12(Aug):2493–2537, 2011.

[52] Michael G. Crandall, Hitoshi Ishii, and Pierre-Louis Lions. User's guide to viscosity solutions of second order partial differential equations. *Bull. Amer. Math. Soc. (N.S.)*, 27(1):1–67, 1992.

[53] Michael G. Crandall and Pierre-Louis Lions. Viscosity solutions of Hamilton-Jacobi equations. *Trans. Amer. Math. Soc.*, 277(1):1–42, 1983.

[54] Edward J. Dean and Roland Glowinski. Numerical methods for fully nonlinear elliptic equations of the Monge-Ampère type. *Comput. Methods Appl. Mech. Engrg.*, 195(13-16):1344–1386, 2006.

[55] Kristian Debrabant and Espen R. Jakobsen. Semi-Lagrangian schemes for linear and fully non-linear diffusion equations. *Math. Comp.*, 82(283):1433–1462, 2013.

[56] Boualem Djehiche, Alain Tcheukam, and Hamidou Tembine. Mean-field-type games in engineering. *arXiv preprint arXiv:1605.03281*, 2016.

[57] Kenji Doya. Reinforcement learning in continuous time and space. *Neural computation*, 12(1):219–245, 2000.

[58] Daniel J. Duffy. *Finite difference methods in financial engineering*. Wiley Finance Series. John Wiley & Sons, Ltd., Chichester, 2006. A partial differential equation approach, With 1 CD-ROM (Windows, Macintosh and UNIX).

[59] Weinan E, Jiequn Han, and Arnulf Jentzen. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Commun. Math. Stat.*, 5(4):349–380, 2017.

[60] Nicole El Karoui, Shige Peng, and Marie Claire Quenez. Backward stochastic differential equations in finance. *Math. Finance*, 7(1):1–71, 1997.

[61] Xiaobing Feng, Roland Glowinski, and Michael Neilan. Recent developments in numerical methods for fully nonlinear second order partial differential equations. *SIAM Rev.*, 55(2):205–267, 2013.

[62] Xiaobing Feng and Max Jensen. Convergent semi-Lagrangian methods for the Monge-Ampère equation on unstructured grids. *SIAM J. Numer. Anal.*, 55(2):691–712, 2017.

[63] Xiaobing Feng and Michael Neilan. Vanishing moment method and moment solutions for fully nonlinear second order partial differential equations. *J. Sci. Comput.*, 38(1):74–98, 2009.

[64] Neil Powell Firth. *High dimensional American options*. PhD thesis, University of Oxford, 2005.

[65] Peter Forsyth. An introduction to computational finance without agonizing pain. 2017.

[66] Peter A. Forsyth and George Labahn. Numerical methods for controlled Hamilton-Jacobi-Bellman PDEs in finance. *Journal of Computational Finance*, 11(2):1, 2007.

[67] Peter A. Forsyth and Kenneth R. Vetzal. Quadratic convergence for valuing American options using a penalty method. *SIAM J. Sci. Comput.*, 23(6):2095–2122, 2002.

[68] Leopoldo P. Franca, Sérgio L. Frey, and Thomas J. R. Hughes. Stabilized finite element methods. I. Application to the advective-diffusive model. *Comput. Methods Appl. Mech. Engrg.*, 95(2):253–276, 1992.

[69] Stephanie Friedhoff and Scott MacLachlan. A generalized predictive analysis tool for multigrid methods. *Numerical Linear Algebra with Applications*, 22(4):618–647, 2015.

[70] Avner Friedman. *Differential games*. Courier Corporation, 2013.

[71] Brittany D. Froese. A numerical method for the elliptic Monge-Ampère equation with transport boundary conditions. *SIAM J. Sci. Comput.*, 34(3):A1432–A1459, 2012.

[72] Brittany D. Froese and Adam M. Oberman. Convergent finite difference solvers for viscosity solutions of the elliptic Monge-Ampère equation in dimensions two and higher. *SIAM J. Numer. Anal.*, 49(4):1692–1714, 2011.

[73] Brittany D. Froese and Adam M. Oberman. Fast finite difference solvers for singular solutions of the elliptic Monge-Ampère equation. *J. Comput. Phys.*, 230(3):818–834, 2011.

[74] Brittany D. Froese and Adam M. Oberman. Convergent filtered schemes for the Monge-Ampère partial differential equation. *SIAM J. Numer. Anal.*, 51(1):423–444, 2013.

[75] Masaaki Fujii, Akihiko Takahashi, and Masayuki Takahashi. Asymptotic expansion as prior knowledge in deep learning method for high dimensional bsdes. *arXiv preprint arXiv:1710.07030*, 2017.

[76] Martin J. Gander. 50 years of time parallel time integration. In *Multiple Shooting and Time Domain Decomposition Methods*, pages 69–113. Springer, 2015.

[77] Martin J. Gander and Martin Neumüller. Analysis of a new space-time parallel multigrid algorithm for parabolic problems. *SIAM J. Sci. Comput.*, 38(4):A2173–A2208, 2016.

[78] Paul Glasserman. *Monte Carlo methods in financial engineering*, volume 53 of *Applications of Mathematics (New York)*. Springer-Verlag, New York, 2004. Stochastic Modelling and Applied Probability.

[79] Sergei Konstantinovich Godunov. A difference method for numerical calculation of discontinuous solutions of the equations of hydrodynamics. *Mat. Sb. (N.S.)*, 47 (89):271–306, 1959.

[80] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, 2016.

[81] Arthur Ardeshir Goshtasby. *2-D and 3-D image registration: for medical, remote sensing, and industrial applications*. John Wiley & Sons, 2005.

[82] P. Jameson Graber and Alain Bensoussan. Existence and uniqueness of solutions for Bertrand and Cournot mean field games. *Appl. Math. Optim.*, 77(1):47–71, 2018.

[83] Olivier Guéant, Jean-Michel Lasry, and Pierre-Louis Lions. Mean field games and applications. In *Paris-Princeton lectures on mathematical finance 2010*, pages 205–266. Springer, 2011.

[84] Cristian E Gutiérrez. *The Monge-Ampère Equation*, volume 42. Springer Science & Business Media, 2012.

[85] Steven Haker and Allen Tannenbaum. Optimal mass transport and image registration. In *Variational and Level Set Methods in Computer Vision, 2001. Proceedings. IEEE Workshop on*, pages 29–36. IEEE, 2001.

[86] Steven Haker, Lei Zhu, Allen Tannenbaum, and Sigurd Angenent. Optimal mass transport for registration and warping. *International Journal of computer vision*, 60(3):225–240, 2004.

[87] Dong Han and Justin W. L. Wan. Multigrid methods for second order Hamilton-Jacobi-Bellman and Hamilton-Jacob-Bellman-Isaacs equations. *SIAM J. Sci. Comput.*, 35(5):S323–S344, 2013.

[88] Jiequn Han, Arnulf Jentzen, and Weinan E. Solving high-dimensional partial differential equations using deep learning. *Proc. Natl. Acad. Sci. USA*, 115(34):8505–8510, 2018.

[89] Martin B. Haugh and Leonid Kogan. Pricing American options: a duality approach. *Oper. Res.*, 52(2):258–270, 2004.

[90] Changhong He, J Shannon Kennedy, Thomas F. Coleman, Peter A. Forsyth, Yuying Li, and Kenneth R. Vetzal. Calibration and hedging under jump diffusion. *Review of Derivatives Research*, 9(1):1–35, 2006.

[91] Steve Heston and Guofu Zhou. On the rate of convergence of discrete-time contingent claims. *Math. Finance*, 10(1):53–75, 2000.

[92] Derek LG Hill, Philipp G Batchelor, Mark Holden, and David J Hawkes. Medical image registration. *Physics in medicine and biology*, 46(3):R1, 2001.

[93] Ronald H. W. Hoppe. Multigrid methods for Hamilton-Jacobi-Bellman equations. *Numer. Math.*, 49(2-3):239–254, 1986.

[94] Graham Horton and Stefan Vandewalle. A space-time multigrid method for parabolic partial differential equations. *SIAM J. Sci. Comput.*, 16(4):848–864, 1995.

[95] Ronald A. Howard. *Dynamic programming and Markov processes*. The Technology Press of M.I.T., Cambridge, Mass.; John Wiley & Sons, Inc., New York-London, 1960.

[96] Thomas J. R. Hughes, Leopoldo P. Franca, and Gregory M. Hulbert. A new finite element formulation for computational fluid dynamics: Viii. the Galerkin/least-squares method for advective-diffusive equations. *Computer Methods in Applied Mechanics and Engineering*, 73(2):173–189, 1989.

[97] John C. Hull. *Options futures and other derivatives*. Pearson/Prentice Hall, 2003.

[98] Michal Irani and Shmuel Peleg. Improving resolution by image registration. *CVGIP: Graphical models and image processing*, 53(3):231–239, 1991.

[99] Antony Jameson. Solution of the Euler equations for two-dimensional transonic flow by a multigrid method. *Appl. Math. Comput.*, 13(3-4):327–355, 1983.

[100] J. S. Kennedy, Peter A. Forsyth, and Kenneth R. Vetzal. Dynamic hedging under jump diffusion with transaction costs. *Oper. Res.*, 57(3):541–559, 2009.

[101] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[102] Martin Knott and Cyril S Smith. On the optimal mapping of distributions. *Journal of Optimization Theory and Applications*, 43(1):39–49, 1984.

[103] Michael Kohler. A review on regression-based Monte Carlo methods for pricing American options. In *Recent developments in applied probability and statistics*, pages 37–58. Springer, 2010.

[104] Michael Kohler, Adam Krzyżak, and Nebojsa Todorovic. Pricing of high-dimensional American options by neural networks. *Math. Finance*, 20(3):383–410, 2010.

[105] Nikolai Vladimirovich Krylov. The control of the solution of a stochastic integral equation. *Teor. Verojatnost. i Primenen.*, 17:111–128, 1972.

[106] Harold J. Kushner and Paul G. Dupuis. *Numerical methods for stochastic control problems in continuous time*, volume 24 of *Applications of Mathematics (New York)*. Springer-Verlag, New York, 1992.

[107] Aime Lachapelle, Julien Salomon, and Gabriel Turinici. Computation of mean field equilibria in economics. *Math. Models Methods Appl. Sci.*, 20(4):567–588, 2010.

[108] Omar Lakkis and Tristan Pryer. A finite element method for nonlinear elliptic problems. *SIAM J. Sci. Comput.*, 35(4):A2025–A2045, 2013.

[109] Jean-Michel Lasry and Pierre-Louis Lions. Jeux à champ moyen. I–le cas stationnaire. *Comptes Rendus Mathématique*, 343(9):619–625, 2006.

[110] Jean-Michel Lasry and Pierre-Louis Lions. Jeux à champ moyen. II–horizon fini et contrôle optimal. *Comptes Rendus Mathématique*, 343(10):679–684, 2006.

[111] Jean-Michel Lasry and Pierre-Louis Lions. Mean field games. *Jpn. J. Math.*, 2(1):229–260, 2007.

[112] Peter Lax and Burton Wendroff. Systems of conservation laws. *Comm. Pure Appl. Math.*, 13:217–237, 1960.

[113] Coenraad Cornelis Willem Leentvaar. Pricing multi-asset options with sparse grids. 2008.

[114] Kenneth Levenberg. A method for the solution of certain non-linear problems in least squares. *Quart. Appl. Math.*, 2:164–168, 1944.

[115] Jessey Lin. Wide stencil for the Monge-Ampère equation. Technical report, University of Waterloo master essay, supervised by Justin WL Wan, available on https://uwaterloo.ca/computational-mathematics/sites/ca.computational-mathematics/files/uploads/files/cmmain1.pdf, 2014.

[116] Pierre-Louis Lions. Hamilton-Jacobi-Bellman equations and the optimal control of stochastic systems. In *Proceedings of the International Congress of Mathematicians, Vol. 1, 2 (Warsaw, 1983)*, pages 1403–1417. PWN, Warsaw, 1984.

[117] Francis A. Longstaff and Eduardo S. Schwartz. Valuing American options by simulation: a simple least-squares approach. *The review of financial studies*, 14(1):113–147, 2001.

[118] K Ma and Peter A. Forsyth. An unconditionally monotone numerical scheme for the two factor uncertain volatility model. *Preprint*, 2014.

[119] JB Antoine Maintz and Max A Viergever. A survey of medical image registration. *Medical image analysis*, 2(1):1–36, 1998.

[120] Donald W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *J. Soc. Indust. Appl. Math.*, 11:431–441, 1963.

[121] Jan Modersitzki. *FAIR: flexible algorithms for image registration*, volume 6 of *Fundamentals of Algorithms*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2009.

[122] Kevin P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[123] Oleg Museyko, Michael Stiglmayr, Kathrin Klamroth, and Günter Leugering. On the application of the Monge-Kantorovich problem to image registration. *SIAM J. Imaging Sci.*, 2(4):1068–1097, 2009.

[124] Artem Napov and Yvan Notay. An algebraic multigrid method with guaranteed convergence rate. *SIAM journal on scientific computing*, 34(2):A1079–A1109, 2012.

[125] Yvan Notay. An aggregation-based algebraic multigrid method. *Electron. Trans. Numer. Anal.*, 37:123–146, 2010.

[126] Adam M. Oberman. Wide stencil finite difference schemes for the elliptic Monge-Ampère equation and functions of the eigenvalues of the Hessian. *Discrete Contin. Dyn. Syst. Ser. B*, 10(1):221–238, 2008.

[127] Vladimir I. Oliker and Laird D. Prussner. On the numerical solution of the equation $(\partial^2 z/\partial x^2)(\partial^2 z/\partial y^2) - ((\partial^2 z/\partial x \partial y))^2 = f$ and its discretizations. I. *Numer. Math.*, 54(3):271–293, 1988.

[128] Stanley Osher and Chi-Wang Shu. High-order essentially nonoscillatory schemes for Hamilton-Jacobi equations. *SIAM J. Numer. Anal.*, 28(4):907–922, 1991.

[129] Maziar Raissi. Forward-backward stochastic neural networks: Deep learning of high-dimensional partial differential equations. *arXiv preprint arXiv:1804.07010*, 2018.

[130] Christoph Reisinger and Julen Rotaetxe Arto. Boundary treatment and multigrid preconditioning for semi-Lagrangian schemes applied to Hamilton-Jacobi-Bellman equations. *arXiv preprint arXiv:1605.04821*, 2016.

[131] Christoph Reisinger and Jan Hendrik Witte. On the use of policy iteration as an easy way of pricing American options. *SIAM J. Financial Math.*, 3(1):459–478, 2012.

[132] Christoph Reisinger and Gabriel Wittum. Efficient hierarchical approximation of high-dimensional option pricing problems. *SIAM J. Sci. Comput.*, 29(1):440–458, 2007.

[133] Sander Rhebergen, Bernardo Cockburn, and Jaap J. W. van der Vegt. A space-time discontinuous Galerkin method for the incompressible Navier-Stokes equations. *J. Comput. Phys.*, 233:339–358, 2013.

[134] John W. Ruge and Klaus Stüben. Algebraic multigrid. In *Multigrid methods*, volume 3 of *Frontiers Appl. Math.*, pages 73–130. SIAM, Philadelphia, PA, 1987.

[135] Yousef Saad. *Iterative methods for sparse linear systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, second edition, 2003.

[136] Alexander A. Samarskii. *The theory of difference schemes*, volume 240 of *Monographs and Textbooks in Pure and Applied Mathematics*. Marcel Dekker, Inc., New York, 2001.

[137] Louis-Philippe Saumier, Martial Agueh, and Boualem Khouider. An efficient numerical algorithm for the L^2 optimal transport problem with periodic densities. *IMA J. Appl. Math.*, 80(1):135–157, 2015.

[138] Benjamin Seibold. Performance of algebraic multigrid methods for non-symmetric matrices arising in particle methods. *Numer. Linear Algebra Appl.*, 17(2-3):433–451, 2010.

[139] Rüdiger U. Seydel. *Tools for computational finance*. Universitext. Springer-Verlag, Berlin, third edition, 2006.

[140] P. N. Shivakumar, Joseph J. Williams, Qiang Ye, and Corneliu A. Marinov. On two-sided bounds related to weakly diagonally dominant M-matrices with application to digital circuit dynamics. *SIAM J. Matrix Anal. Appl.*, 17(2):298–312, 1996.

[141] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[142] Justin Sirignano and Konstantinos Spiliopoulos. DGM: A deep learning algorithm for solving partial differential equations. *J. Comput. Phys.*, 375:1339–1364, 2018.

[143] Iain Smears. Hamilton-Jacobi-Bellman equations analysis and numerical analysis. Technical report, research report available on www.math.dur.ac.uk/Ug/projects/highlights/PR4/\Smears_HJB_report.pdf.

[144] J Sola and Joaquin Sevilla. Importance of input data normalization for the application of neural networks to complex industrial problems. *IEEE Transactions on nuclear science*, 44(3):1464–1468, 1997.

[145] Lars Stentoft. Convergence of the least squares Monte Carlo approach to American option valuation. *Management Science*, 50(9):1193–1203, 2004.

[146] Klaus Stüben. An introduction to algebraic multigrid. *Multigrid*, pages 413–532, 2001.

[147] Klaus Stüben. A review of algebraic multigrid. *J. Comput. Appl. Math.*, 128(1-2):281–309, 2001. Numerical analysis 2000, Vol. VII, Partial differential equations.

[148] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: an introduction*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, second edition, 2018.

[149] Evangelos A. Theodorou, Jonas Buchli, and Stefan Schaal. A generalized path integral control approach to reinforcement learning. *J. Mach. Learn. Res.*, 11:3137–3181, 2010.

[150] Philippe Thevenaz, Urs E Ruttimann, and Michael Unser. A pyramid approach to subpixel registration based on intensity. *IEEE transactions on image processing*, 7(1):27–41, 1998.

[151] Howard Thom. Longstaff Schwartz pricing of Bermudan options and their Greeks, 2009.

[152] Eleuterio F. Toro. *Riemann solvers and numerical methods for fluid dynamics: a practical introduction*. Springer Science & Business Media, 2013.

[153] Ulrich Trottenberg, Cornelius W Oosterlee, and Anton Schüller. *Multigrid*. Academic Press, Inc., San Diego, CA, 2001. With contributions by A. Brandt, P. Oswald and K. Stüben.

[154] John N. Tsitsiklis and Benjamin Van Roy. Optimal stopping of Markov processes: Hilbert space theory, approximation algorithms, and an application to pricing high-dimensional financial derivatives. *IEEE Trans. Automat. Control*, 44(10):1840–1851, 1999.

[155] J Wang and Peter A. Forsyth. Maximal use of central differencing for Hamilton-Jacobi-Bellman PDEs in finance. *SIAM J. Numer. Anal.*, 46(3):1580–1601, 2008.

[156] Roman Wienands and Cornelis W. Oosterlee. On three-grid Fourier analysis for multigrid. *SIAM J. Sci. Comput.*, 23(2):651–671, 2001. Copper Mountain Conference (2000).

[157] Jinchao Xu and Ludmil T. Zikatanov. Algebraic multigrid methods. *arXiv preprint arXiv:1611.01917*, 2016.

Appendices

A.1 Pseudo-code for Multigrid Cycles

Denote a linear system arising from discretization of an elliptic PDE as

$$A_h u_h = b_h. \tag{A.1}$$

Then multigrid cycles for the linear system is described in Algorithm A.1.

Denote a nonlinear system arising from discretization of an elliptic PDE as

$$\mathcal{N}_h(u_h) = 0. \tag{A.2}$$

Then multigrid cycles for the nonlinear system is described in Algorithm A.2.

A.2 Timestepping for HJB/KFP Systems Arising from Mean Field Games

For the HJB/KFP system (2.11)-(2.12), timestepping needs to be implemented as a forward/backward fixed point iteration, where each iteration consists of two steps: One step is to start with an initial guess of m_h on the entire spacetime $\Omega \times [0, T]$, fix m_h and solve the HJB equation (2.11) for u_h and $\boldsymbol{c}_h^*$ by backward timestepping; the other step is to fix $\boldsymbol{c}_h^*$ and solve the KFP equation (2.12) for m_h by forward timestepping. This iteration is described in Algorithm A.3.

Line 6 of Algorithm A.3 requires solving discretized HJB equations on each timestep. One may consider using policy iteration; see Algorithm 1.1. We leave the implementation details to interested readers.

Algorithm A.1 Multigrid method for linear discretized system $A_h u_h = b_h$

1: Start with an initial guess $u_h^{(0)}$.
2: **for** $k = 1, 2, ...$ until convergence **do**
3: $u_h^{(k+1)} = \text{MGCYC}\ (A_h,\ u_h^{(k)},\ b_h,\ \gamma,\ \nu_1,\ \nu_2)$. See below for the subroutine "MGCYC".
4: **end for**

subroutine $u_h^{(k+1)} = \text{MGCYC}\ (A_h,\ u_h^{(k)},\ b_h,\ \gamma,\ \nu_1,\ \nu_2)$
5: Perform ν_1 **pre-smoothings** (e.g. Gauss-Seidel iterations) on $A_h u_h = b_h$, which updates the solution $u_h^{(k)} \to \bar{u}_h^{(k)}$.
6: Compute the residual: $r_h = b_h - A_h \bar{u}_h^{(k)}$.
7: Define a **coarsening strategy** $h \to 2h$.
8: **Restrict** the residual: $r_{2h} = R_h r_h$, where R_h is a restriction operator.
9: Construct A_{2h} by either the **direct discretization** of a PDE with mesh size $2h$, or the **Petrov-Galerkin coarse grid operator** $A_{2h} = R_h A_h P_h$.
10: **if** on the coarsest grid **then**
11: Solve $A_{2h} e_{2h} = r_{2h}$ for the coarse grid estimated error e_{2h} using Gaussian elimination.
12: **else**
13: Solve $A_{2h} e_{2h} = r_{2h}$ for e_{2h} approximately using γ-time recursions of
14: $e_{2h} = \text{MGCYC}\ (A_{2h},\ 0,\ r_{2h},\ \gamma,\ \nu_1,\ \nu_2)$.
15: **end if**
16: **Interpolate** the estimated error: $e_h = P_h e_{2h}$, where P_h is an interpolation operator.
17: Correct the fine grid solution: $\tilde{u}_h^{(k)} = \bar{u}_h^{(k)} + e_h$.
18: Perform ν_2 **post-smoothings** on $A_h u_h = b_h$, which updates the solution $\tilde{u}_h^{(k)} \to u_h^{(k+1)}$.

Algorithm A.2 FAS multigrid method for nonlinear discretized system $\mathcal{N}_h(u_h) = 0$

Start with an initial guess $u_h^{(0)}$.
for $k = 1, 2, ...$ until convergence **do**
  $u_h^{(k+1)}$ = FASCYC ($u_h^{(k)}$, 0, γ, ν_1, ν_2). See below for the subroutine "FASCYC".
end for

subroutine $u_h^{(k+1)}$ = FASCYC ($u_h^{(k)}$, b_h, γ, ν_1, ν_2)
Construct the nonlinear system $\mathcal{N}_h(u_h) = b_h$, where $\mathcal{N}_h$ is the direct discretization of a PDE with mesh size h.
Perform ν_1 **pre-smoothings** (e.g. nonlinear Gauss-Seidel iterations) on $\mathcal{N}_h(u_h) = b_h$, which updates the solution $u_h^{(k)} \to \bar{u}_h^{(k)}$.
Compute the residual: $r_h = b_h - \mathcal{N}_h(\bar{u}_h^{(k)})$.
Define a **coarsening strategy** $h \to 2h$.
Inject the solution: $\bar{u}_h^{(k)} \to \bar{u}_{2h}^{(k)}$.
Restrict the residual: $r_{2h} = R_h r_h$, where R_h is a restriction operator.
Construct the coarse grid nonlinear operator $\mathcal{N}_{2h}$ using the **direct discretization** of a PDE with mesh size $2h$.
Construct the coarse grid right hand side: $b_{2h} = \mathcal{N}_{2h}(\bar{u}_{2h}^{(k)}) + R_h r_h$.
if on the coarsest grid **then**
  Solve $\mathcal{N}_{2h}(\hat{u}_{2h}^{(k)}) = b_{2h}$ for $\hat{u}_{2h}^{(k)}$ using nonlinear iterative solvers (e.g. nonlinear Gauss-Seidel iteration, policy iteration).
else
  Solve $\mathcal{N}_{2h}(\hat{u}_{2h}^{(k)}) = b_{2h}$ for $\hat{u}_{2h}^{(k)}$ approximately using γ-time recursions of
    $\hat{u}_{2h}^{(k)}$ = FASCYC ($\bar{u}_{2h}^{(k)}$, b_{2h}, γ, ν_1, ν_2).
end if
Compute the coarse grid estimated error: $e_{2h} = \hat{u}_{2h}^{(k)} - \bar{u}_{2h}^{(k)}$.
Interpolate the estimated error: $e_h = P_h e_{2h}$, where P_h is an interpolation operator.
Correct the fine grid solution: $\tilde{u}_h^{(k)} = \bar{u}_h^{(k)} + e_h$.
Perform ν_2 **post-smoothings** on $\mathcal{N}_h(u_h) = b_h$, which updates the solution $\tilde{u}_h^{(k)} \to u_h^{(k+1)}$.

Algorithm A.3 Forward/backward timestepping fixed point iteration for solving (2.11)-(2.12)

1: Use the initial condition in (2.4) to construct m_h^0.
2: Start with an initial guess $(m_h^n)^{(0)}$ for $n = 1, \cdots, n_t$.
3: Use the terminal condition in (2.1) to construct $u_h^{n_t}$.
4: **for** $k = 1, 2, ...$ until convergence **do**
5: **for** $n = n_t - 1, \cdots, 1, 0$ **do**
6: Solve the discretized HJB equations

$$A_{HJB}^n((\boldsymbol{c}_h^n)^{(k)})(u_h^n)^{(k)} = \frac{1}{\Delta t} \cdot (u_h^{n+1})^{(k)} + L((\boldsymbol{c}_h^n)^{(k)}) + \Phi((m_h^n)^{(k-1)}),$$
$$\text{subject to} \quad (\boldsymbol{c}_h^n)^{(k)} \equiv \underset{\boldsymbol{c}_h^n \in \mathbb{R}^{2n_x n_y}}{\arg\max} \left\{ A_{HJB}^n(\boldsymbol{c}_h^n)(u_h^n)^{(k)} - L(\boldsymbol{c}_h^n) \right\},$$

 for the unknowns $(u_h^n)^{(k)}$ and $(\boldsymbol{c}_h^n)^{(k)}$.
7: **end for**
8: **for** $n = 1, 2, \cdots, n_t$ **do**
9: Solve the discretized KFP equations
 $A_{KFP}^n((\boldsymbol{c}_h^n)^{(k)})(m_h^n)^{(k)} = \frac{1}{\Delta t} \cdot (m_h^{n-1})^{(k)}$ for the unknown $(m_h^n)^{(k)}$.
10: **end for**
11: **end for**
12: Convergent solution: $u_h^n = (u_h^n)^{(k)}$, $m_h^n = (m_h^n)^{(k)}$, $(\boldsymbol{c}^*)_h^n = (\boldsymbol{c}_h^n)^{(k)}$, for $n = 0, \cdots, n_t$.

A.3 Two-grid Analysis for the KFP Equation (2.36)

Following [156], the Fourier symbols of the differential operator of (2.36), the interpolation operators and the restriction operators are

$$\begin{aligned}\tilde{L}_h(\boldsymbol{\kappa}) = \; & \left(1 + \tfrac{\sigma\Delta t}{h^2}\left(4 + \tfrac{c_1 h}{\sigma} + \tfrac{c_2 h}{\sigma}\right)\right) - \tfrac{\sigma\Delta t}{h^2}\left(1 + \tfrac{c_1 h}{\sigma}\right) e^{-i\kappa_1} - \tfrac{\sigma\Delta t}{h^2}\left(1 + \tfrac{c_2 h}{\sigma}\right) e^{-i\kappa_2} \\ & - \tfrac{\sigma\Delta t}{h^2} e^{i\kappa_1} - \tfrac{\sigma\Delta t}{h^2} e^{i\kappa_2} - e^{-i\kappa_0}.\end{aligned}$$

$$\tilde{I}_{2h}^h(\boldsymbol{\kappa}) = \begin{cases} \frac{1}{8}\left(1 + \cos\kappa_1\right)\left(1 + \cos\kappa_2\right)\left(1 + \cos\kappa_0\right), & \text{full coarsening;} \\ \frac{1}{4}\left(1 + \cos\kappa_1\right)\left(1 + \cos\kappa_2\right), & \text{semi coarsening.} \end{cases}$$

$$\tilde{I}_h^{2h}(\boldsymbol{\kappa}) = \begin{cases} \frac{1}{8}\left(1+e^{-i\kappa_0}\right)\Big(1+\frac{e^{i\kappa_1}}{1+\exp(\sigma_1^{-1}c_1h)}+\frac{e^{-i\kappa_1}}{1+\exp(-\sigma_1^{-1}c_1h)}+\frac{e^{i\kappa_2}}{1+\exp(\sigma_2^{-1}c_2h)} \\ +\frac{e^{-i\kappa_2}}{1+\exp(-\sigma_2^{-1}c_2h)}+\frac{e^{i(\kappa_1+\kappa_2)}}{1+\exp(\sigma_1^{-1}c_1h+\sigma_2^{-1}c_2h)}+\frac{e^{-i(\kappa_1+\kappa_2)}}{1+\exp(-\sigma_1^{-1}c_1h-\sigma_2^{-1}c_2h)}\Big), \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{full coarsening;} \\ \frac{1}{4}\Big(1+\frac{e^{i\kappa_1}}{1+\exp(\sigma_1^{-1}c_1h)}+\frac{e^{-i\kappa_1}}{1+\exp(-\sigma_1^{-1}c_1h)}+\frac{e^{i\kappa_2}}{1+\exp(\sigma_2^{-1}c_2h)} \\ +\frac{e^{-i\kappa_2}}{1+\exp(-\sigma_2^{-1}c_2h)}+\frac{e^{i(\kappa_1+\kappa_2)}}{1+\exp(\sigma_1^{-1}c_1h+\sigma_2^{-1}c_2h)}+\frac{e^{-i(\kappa_1+\kappa_2)}}{1+\exp(-\sigma_1^{-1}c_1h-\sigma_2^{-1}c_2h)}\Big), \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{semi coarsening.} \end{cases}$$

Now we are ready to construct the Fourier symbol of the two-grid operator. Given a low frequency mode $\boldsymbol{\kappa}^{000} \equiv \boldsymbol{\kappa} \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right)^3$, we define an 8-dimensional space: $\text{span}\{\, \varphi_{h,\Delta t}(\boldsymbol{\kappa}^{\boldsymbol{\alpha}}; \cdot) : \boldsymbol{\alpha} = (\alpha_1, \alpha_2, \alpha_0),\ \alpha_1, \alpha_2, \alpha_0 \in \{0, 1\}\,\}$, where $\boldsymbol{\kappa}^{\boldsymbol{\alpha}} \equiv \boldsymbol{\kappa}^{000} - (\alpha_1 \text{sign}(\kappa_1), \alpha_2 \text{sign}(\kappa_2), \alpha_0 \text{sign}(\kappa_0)) \cdot \pi$. This 8-dimensional space is called $2h$-harmonics. The significance of the $2h$-harmonics is that it is invariant under the two-grid operator M_h^{2h}. In the $2h$-harmonics, the two-grid operator is given by an 8×8 matrix

$$M_h^{2h}(\boldsymbol{\kappa}) \equiv (S_h(\boldsymbol{\kappa}))^{\nu_2} \left[I_h - I_{2h}^h(\boldsymbol{\kappa})(L_{2h}(2\boldsymbol{\kappa}))^{-1} I_h^{2h}(\boldsymbol{\kappa}) L_h(\boldsymbol{\kappa})\right] (S_h(\boldsymbol{\kappa}))^{\nu_1},$$

where $I_h \in \mathbb{R}^{8\times 8}$ is an identity matrix, $L_h(\boldsymbol{\kappa}) \in \mathbb{C}^{8\times 8}$ is a diagonal matrix consisting of $\{\tilde{L}_h(\boldsymbol{\kappa}^{\boldsymbol{\alpha}})\}$, $S_h(\boldsymbol{\kappa}) \in \mathbb{C}^{8\times 8}$ is a diagonal matrix consisting of $\{\tilde{S}_h(\boldsymbol{\kappa}^{\boldsymbol{\alpha}})\}$. For full-coarsening, the 8-dimensional $2h$-harmonics is mapped to a single Fourier mode $\varphi_{2h,2\Delta t}(2\boldsymbol{\kappa}^{000}; \cdot)$. Hence, $I_{2h}^h(\boldsymbol{\kappa}) \in \mathbb{C}^{8\times 1}$ is a column matrix consisting of $\{\tilde{I}_{2h}^h(\boldsymbol{\kappa}^{\boldsymbol{\alpha}})\}$, $I_h^{2h}(\boldsymbol{\kappa}) \in \mathbb{C}^{1\times 8}$ is a row matrix consisting of $\{\tilde{I}_h^{2h}(\boldsymbol{\kappa}^{\boldsymbol{\alpha}})\}$, and $L_{2h}(2\boldsymbol{\kappa}) \equiv \tilde{L}_{2h}(2\boldsymbol{\kappa}^{000})$ is a 1×1 matrix. For semi-coarsening, the 8-dimensional $2h$-harmonics is mapped to two Fourier modes $\{\varphi_{2h,\Delta t}((2\kappa_1, 2\kappa_2, \kappa_0); \cdot)$, $\varphi_{2h,\Delta t}((2\kappa_1, 2\kappa_2, \kappa_0 - \text{sign}(\kappa_0)\pi); \cdot)\}$. Hence, $I_{2h}^h(\boldsymbol{\kappa})$, $I_h^{2h}(\boldsymbol{\kappa})$ and $L_{2h}(2\boldsymbol{\kappa})$ are changed accordingly into 8×2, 2×8 and 2×2 matrices. We refer readers to [156] for further technical details.

A.4 Semi-Lagrangian Wide Stencil Discretization

Section 3.4.2 has discussed basic ideas of semi-Lagrangian wide stencil discretization. In this appendix, we continue the discussion on discretization near the boundary of the computational domain, and provide an explicit demonstration of (at most) 17 stencil points.

If we apply the semi-Lagrangian wide stencil discretization at a grid point $\boldsymbol{x}_{i,j}$ that is close to the boundary, some of its associated stencil points may fall outside the computational domain $\overline{\Omega}$. In such case, our solution is to shrink the corresponding stencil length(s) such that the stencil point(s) are relocated onto the boundary $\partial\Omega$. Without loss

of generality, we analyze one scenario, as illustrated in Figure A.1. Let us assume that $\boldsymbol{x}_{i,j} + \sqrt{h}(\boldsymbol{e}_z)_{i,j}$, as represented by the hollow star, falls outside $\overline{\Omega}$. We truncate the corresponding stencil length from $\sqrt{h}$ to η_1 along the $\boldsymbol{e}_z$ axis, or equivalently, we relocate the stencil point to $\boldsymbol{x}_{i,j} + \eta_1(\boldsymbol{e}_z)_{i,j} \in \partial\Omega$, as represented by the black star. In this case, the finite difference approximation for $u_{zz}(\boldsymbol{x}_{i,j})$ is replaced by

$$D_z^+ D_z^- u_{i,j} \equiv \frac{\frac{g(\boldsymbol{x}_{i,j}+\eta_1(\boldsymbol{e}_z)_{i,j}) - u_{i,j}}{\eta_1} - \frac{u_{i,j} - \mathcal{I}_h u|_{\boldsymbol{x}_{i,j}-\sqrt{h}(\boldsymbol{e}_z)_{i,j}}}{\sqrt{h}}}{\frac{\eta_1+\sqrt{h}}{2}}, \tag{A.3}$$

where we have used the Dirichlet boundary condition (3.2), i.e. $u(\boldsymbol{x}_{i,j} + \eta_1(\boldsymbol{e}_z)_{i,j}) = g(\boldsymbol{x}_{i,j} + \eta_1(\boldsymbol{e}_z)_{i,j})$. We note that such procedure can be used whenever $\boldsymbol{x}_{i,j}$ is close to the boundary and a truncation of stencil is needed.

Section 3.4.2 has mentioned that semi-Lagrangian wide stencil discretization ends up at most 17 stencil points. To show it explicitly, we label all the 17 stencil points in Figure A.2. For instance, $\mathcal{I}_h u|_{\boldsymbol{x}_{i,j}+\sqrt{h}(\boldsymbol{e}_z)_{i,j}}$ can be written as the linear combination of the unknowns at the four neighboring points u_{r_1,s_1}, u_{r_1+1,s_1}, u_{r_1,s_1+1} and u_{r_1+1,s_1+1}, i.e.

$$\mathcal{I}_h u|_{\boldsymbol{x}_{i,j}+\sqrt{h}(\boldsymbol{e}_z)_{i,j}} = \sum_{k=0}^{1}\sum_{k'=0}^{1} p_{r_1+k,s_1+k'} \cdot u_{r_1+k,s_1+k'}, \tag{A.4}$$

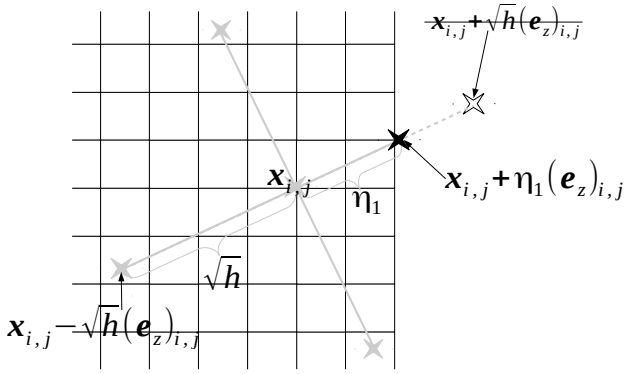


Figure A.1: Semi-Lagrangian wide stencil discretization at a grid point $\boldsymbol{x}_{i,j}$ near the boundary.

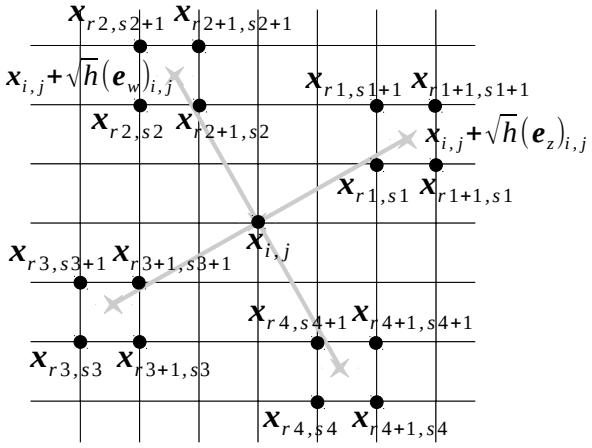


Figure A.2: 17-point stencils resulting from semi-Lagrangian wide stencil discretization.

where p denotes the interpolation weights. Then (3.24) can be expanded as

$$\begin{aligned}
\mathcal{L}_{i,j}(c_{i,j},\theta_{i,j};u_h) &= \frac{2}{h}u_{i,j} - \frac{c_{i,j}}{h}\,\mathcal{I}_h u\big|_{\boldsymbol{x}_{i,j}+\sqrt{h}(\boldsymbol{e}_z)_{i,j}} - \frac{c_{i,j}}{h}\,\mathcal{I}_h u\big|_{\boldsymbol{x}_{i,j}-\sqrt{h}(\boldsymbol{e}_z)_{i,j}} \\
&\quad -\frac{1-c_{i,j}}{h}\,\mathcal{I}_h u\big|_{\boldsymbol{x}_{i,j}+\sqrt{h}(\boldsymbol{e}_w)_{i,j}} - \frac{1-c_{i,j}}{h}\,\mathcal{I}_h u\big|_{\boldsymbol{x}_{i,j}-\sqrt{h}(\boldsymbol{e}_w)_{i,j}} + 2\sqrt{c_{i,j}(1-c_{i,j})f_{i,j}} \\
&= \frac{2}{h}u_{i,j} - \sum_{k=0}^{1}\sum_{k'=0}^{1}\frac{c_{i,j}p_{r_1+k,s_1+k'}}{h}u_{r_1+k,s_1+k'} - \sum_{k=0}^{1}\sum_{k'=0}^{1}\frac{c_{i,j}p_{r_3+k,s_3+k'}}{h}u_{r_3+k,s_3+k'} \\
&\quad -\sum_{k=0}^{1}\sum_{k'=0}^{1}\frac{(1-c_{i,j})p_{r_2+k,s_2+k'}}{h}u_{r_2+k,s_2+k'} - \sum_{k=0}^{1}\sum_{k'=0}^{1}\frac{(1-c_{i,j})p_{r_4+k,s_4+k'}}{h}u_{r_4+k,s_4+k'} \\
&\quad +2\sqrt{c_{i,j}(1-c_{i,j})f_{i,j}}.
\end{aligned} \tag{A.5}$$

As a result, (3.24) has 17 unknowns. We note that if $\boldsymbol{x}_{i,j}$ is near the boundary, then some of the unknowns can be determined by the Dirichlet boundary condition. As a result, there will be less than 17 unknowns.

A.5 Regional Optimization in Section 3.5

To explain the details of the regional optimization, consider again a grid point $\boldsymbol{x}_{i,j}$ and its admissible control set Γ. In Regions Γ^1, Γ^2, $\partial\Gamma^0$, $\partial\Gamma^{13}$ and $\partial\Gamma^{23}$ (see Figure 3.3), where the

standard 7-point stencil discretization (3.19) is applied, the discretizations of $D_x^+ D_x^- u_{i,j}$, $D_y^+ D_y^- u_{i,j}$ and $(D_x^+ D_y^\pm + D_x^- D_y^\mp) u_{i,j}$ do not depend on the control $(c_{i,j}, \theta_{i,j})$. This enables us to derive a closed-form formula for the optimal controls in these regions using the first derivative test, which can be evaluated by $O(1)$ operation and introduces no additional truncation error. More specifically:

Region Γ^1. The region is defined where Condition (3.15) is satisfied. Equation (3.19) gives the objective function in Γ^1:

$$\begin{aligned}\mathcal{L}_{i,j}(c_{i,j}, \theta_{i,j}) \quad &= -\sigma_{11}(c_{i,j}, \theta_{i,j}) D_x^+ D_x^- u_{i,j} - \sigma_{12}(c_{i,j}, \theta_{i,j})(D_x^+ D_y^+ + D_x^- D_y^-) u_{i,j} \\ &\quad -\sigma_{22}(c_{i,j}, \theta_{i,j}) D_y^+ D_y^- u_{i,j} + 2\sqrt{c_{i,j}(1 - c_{i,j}) f_{i,j}},\end{aligned} \tag{A.6}$$

where we only manifest the dependency of $\mathcal{L}_{i,j}$ on the control $(c_{i,j}, \theta_{i,j})$. By the first derivative test, the optimal control in Γ^1 is given by

$$\theta_{i,j}^* = \frac{1}{2} \arctan \frac{(D_x^+ D_y^+ + D_x^- D_y^-) u_{i,j}}{D_y^+ D_y^- u_{i,j} - D_x^+ D_x^- u_{i,j}}, \tag{A.7}$$

$$c_{i,j}^* = \frac{1}{2} \left(1 - \frac{\lambda_{i,j}}{\sqrt{4 f_{i,j} + \lambda_{i,j}^2}} \right), \tag{A.8}$$

where $\lambda_{i,j} \equiv [D_x^+ D_x^- u_{i,j} - D_y^+ D_y^- u_{i,j}] \cos 2\theta_{i,j}^* - (D_x^+ D_y^+ + D_x^- D_y^-) u_{i,j} \sin 2\theta_{i,j}^*$.

With a slight abuse of notations, here and for the rest of section, we use $(c_{i,j}^*, \theta_{i,j}^*)$ to denote the regional (rather than global) optimal control at $\boldsymbol{x}_{i,j}$. We note that $(c_{i,j}^*, \theta_{i,j}^*)$ given by (A.7)-(A.8) may not necessarily be inside Γ^1. If $(c_{i,j}^*, \theta_{i,j}^*) \in \Gamma^1$, then the maximum in Γ^1 must occur at $(c_{i,j}^*, \theta_{i,j}^*)$. Otherwise, the maximum must occur on the boundary of Γ^1, or more specifically, either $\partial\Gamma^0$ or $\partial\Gamma^{13}$, which will be investigated separately.

Region Γ^2. The region is defined where Condition (3.17) is satisfied. The analysis is similar to Region Γ^1.

Region $\partial\Gamma^0$. This is the line $\theta_{i,j} = 0$ which separates Regions Γ^1 and Γ^2. The analysis is similar to Region Γ^1.

Region $\partial\Gamma^{13}$. This is the boundary between Regions Γ^1 and Γ^3. We note that $\partial\Gamma^{13}$ contains two sub-regions: $(sign(c_{i,j} - 1/2), sign(\theta_{i,j})) = (1, -1)$ and $(sign(c_{i,j} - 1/2), sign(\theta_{i,j})) = (-1, 1)$. The objective function on $\partial\Gamma^{13}$ is (A.6). The first derivative test shows that for each of the two sections of $\partial\Gamma^{13}$, the maximum of the objective function

occurs at

$$\theta^*_{i,j} = \frac{sign(\theta_{i,j})}{2} \arctan\left(1 + \gamma^2_{i,j} - \gamma_{i,j}\sqrt{2 + \gamma^2_{i,j}}\right),$$
$$\text{where } \gamma_{i,j} \equiv \frac{sign(c_{i,j} - 1/2)}{2\sqrt{f_{i,j}}}\left(D^+_y D^-_y u_{i,j} - D^+_x D^-_x u_{i,j} - sign(\theta_{i,j})(D^+_x D^+_y + D^-_x D^-_y)u_{i,j}\right).$$

The corresponding $c^*_{i,j}$, derived from Condition (3.15), is

$$c^*_{i,j} = \frac{1}{2}\left(1 + \frac{sign(c_{i,j} - 1/2)}{\sqrt{2}\sin(2|\theta^*_{i,j}| + \frac{\pi}{4})}\right).$$

Region $\partial\Gamma^{23}$. This is the boundary between Regions Γ^2 and Γ^3. The analysis is similar to Region $\partial\Gamma^{13}$.

Region Γ^3. The region is defined where neither (3.15) nor (3.17) is satisfied. The semi-Lagrangian wide stencil discretization (3.24) is applied. Equation (3.24) gives the objective function in Γ^1:

$$\mathcal{L}_{i,j}(c_{i,j}, \theta_{i,j}) = -c_{i,j}\, D^+_z D^-_z u_{i,j} - (1 - c_{i,j})\, D^+_w D^-_w u_{i,j} + 2\sqrt{c_{i,j}(1 - c_{i,j}) f_{i,j}}. \tag{A.9}$$

The dependency of the discretization of $D^+_z D^-_z u_{i,j}$ and $D^+_w D^-_w u_{i,j}$ on the control $\theta_{i,j}$ prevents us from deriving a closed-form formula for $\theta^*_{i,j} \in \Gamma^3$. However, we note that the discretization of $D^+_z D^-_z u_{i,j}$ and $D^+_w D^-_w u_{i,j}$ is independent of the control $c_{i,j}$, which implies that a two dimensional bilinear search on the control $(c_{i,j}, \theta_{i,j}) \in \Gamma$ can be reduced to a one-dimensional linear search on the single control $\theta_{i,j} \in [-\frac{\pi}{4}, \frac{\pi}{4})$.

One can prove that the regional optimal control $(c^*_{i,j}, \theta^*_{i,j}) \in \Gamma^3$ must sit on the following parameterized curve

$$c_{i,j}(\theta_{i,j}) = \begin{cases} \mathcal{C}^\lambda(\theta_{i,j}), & \text{if } \mathcal{C}^\lambda(\theta_{i,j}) \le \mathcal{C}^-(\theta_{i,j}) \text{ or } \mathcal{C}^\lambda(\theta_{i,j}) \ge \mathcal{C}^+(\theta_{i,j}), \\ \mathcal{C}^-(\theta_{i,j}), & \text{if } \mathcal{C}^-(\theta_{i,j}) \le \mathcal{C}^\lambda(\theta_{i,j}) \le \frac{1}{2}, \\ \mathcal{C}^+(\theta_{i,j}), & \text{if } \frac{1}{2} \le \mathcal{C}^\lambda(\theta_{i,j}) \le \mathcal{C}^+(\theta_{i,j}). \end{cases} \tag{A.10}$$

where

$$\mathcal{C}^\pm(\theta_{i,j}) \equiv \frac{1}{2}\left(1 \pm \frac{1}{\sqrt{2}\sin(2|\theta_{i,j}| + \frac{\pi}{4})}\right),$$
$$\mathcal{C}^\lambda(\theta_{i,j}) \equiv \frac{1}{2}\left(1 - \frac{D^+_z D^-_z u_{i,j} - D^+_w D^-_w u_{i,j}}{\sqrt{4 f_{i,j} + (D^+_z D^-_z u_{i,j} - D^+_w D^-_w u_{i,j})^2}}\right), \quad \theta_{i,j} \in [-\frac{\pi}{4}, \frac{\pi}{4}).$$

We remark that $\mathcal{C}^{\pm}$ is given by Condition (3.15) and (3.17), while $\mathcal{C}^{\lambda}$ is given by the first derivative test of (A.9) with respect to $c_{i,j}$. If we plug in the parameterization (A.10), the objective function (A.9) becomes a function of the single control variable, i.e.

$$\mathcal{L}_{i,j}(\theta_{i,j}) \equiv \mathcal{L}_{i,j}(c_{i,j}(\theta_{i,j}), \theta_{i,j}), \quad \theta_{i,j} \in [-\frac{\pi}{4}, \frac{\pi}{4}).$$

This motivates us to discretize the set $[-\frac{\pi}{4}, \frac{\pi}{4})$ into an m-element set, and perform a linear search for the maximum of the parameterized objective function $\mathcal{L}_{i,j}(\theta_{i,j})$ over the single control variable $\theta_{i,j} \in [-\frac{\pi}{4}, \frac{\pi}{4})$. The computational cost is thus reduced to $O(m)$.

Once we obtain the six regional optimal controls and their corresponding objective function values, we search within them for the global optimal control on Γ. This step is cheap and straightforward.

A.6 Computational Cost of Image Registration

We study the computational complexity of Algorithm 5.1. Algorithm 5.1 can be split into two parts: (i) corrections of translation kernels (Lines 4–7), and (ii) the primary nonlinear solver (Lines 8–12). The experiments are implemented by MATLAB, where (ii) is implemented using MATLAB's *fsolve*. We record the CPU time of (i) and (ii) separately.

We first test the computational complexity for Example 5.3 with image sizes of 100×100, 200×200, 400×400 and 800×800. Table A.1 shows that the number of steps for convergence is roughly a constant 3 as the image size increases. In addition, (ii) takes more time than (i). The total CPU time for 800×800 image is around 20 minutes.

We also test the computational time for images of the same size 600×600 across different examples. Since the amounts of translation across different examples are different, which is difficult to compare fairly, we skip the comparison of the CPU time for (i). Table A.2 shows that for the examples where the non-rigid deformation components are larger (such as Example 5.7), the number of steps for convergence and the CPU time for the primary nonlinear solver are also larger.

A.7 Ensemble of Neural Networks

It is well-known that ensemble learning, which is a combination of the multiple machine learning models, usually outperforms individual models [80]. Inspired by this, we consider "ensemble of neural networks" for our neural network formulation in Chapter 6.

Example	Example 5.3			
Image size	100×100	200×200	400×400	800×800
Number of iterations for convergence	5	3	3	3
CPU time for (i) corrections of translation kernels in seconds	1.0	4.6	30	259
CPU time for (ii) the primary nonlinear solver in seconds	3.1	7.3	58	1083
Total CPU time in seconds	4.1	11.9	88	1342

Table A.1: Number of steps for convergence (residual tolerance 10^{-4}), and CPU time for Example 5.3 with different image sizes.

Example	Example 5.3	Example 5.4	Example 5.5	Example 5.6	Example 5.7
Image size	600×600				
Number of iterations for convergence	3	3	10	10	19
CPU time for (ii) the primary nonlinear solver in seconds	147	152	668	627	1613

Table A.2: Number of steps for convergence (residual tolerance 10^{-4}), and CPU time for nonlinear solver for Examples 5.3–5.7 with the same image size of 600×600.

To describe the details, at each timestep (e.g. the n-th timestep), we construct C networks $\{y^n(\boldsymbol{x};\, \Omega_c^n)\,|\,c = 1, \cdots, C\}$ instead of one network. All the C networks have the same architecture as defined in Sections 6.4.1-6.4.5. The difference is that their trainable parameters $\{\Omega_c^n\,|\,c = 1, \cdots, C\}$ are initialized by different set of numbers. Then the C networks are trained by different input data. To do this, we generate CM input samples (6.39) with $m = 1, \cdots, CM$, split them into C copies, and then use each copy of the input samples to train each of the C networks. Consequentially, the trained results of the C networks are independent of each other, i.e. $\{(\Omega_c^n)^*\,|\,c = 1, \cdots, C\}$ are distinct from each other. Then after training, we compute the averages across the ensemble:

$$y^n(\boldsymbol{x}) = \frac{1}{C}\sum_{c=1}^{C} y^n(\boldsymbol{x};\, (\Omega_c^n)^*), \quad \nabla y^n(\boldsymbol{x}) = \frac{1}{C}\sum_{c=1}^{C} \nabla y^n(\boldsymbol{x};\, (\Omega_c^n)^*), \tag{A.11}$$

for the prices and deltas, respectively. Eventually, we use the ensemble-average prices to determine the exercise boundary at the n-th timestep by (6.42) before proceeding to the $(n-1)$-th timestep.

Such ensemble technique yields more precise prices, deltas and thus more precise exercise boundaries. We note that the computation across different ensembles can be parallelized. In practice, we find that $C = 3$ is a good choice, in the sense that the accuracy is improved compared with $C = 1$ without dramatically increasing computational cost.

A.8 Improving Price and Delta at $t = 0$

Our neural network formulation proposed in Chapter 6 yields prices and deltas on the entire spacetime domain. In practical applications, the price and the delta at $t = 0$, $u(\boldsymbol{x}^0, 0)$ and $\nabla u(\boldsymbol{x}^0, 0)$, are of particular interest. We can extract their values from the trained neural network at $t = 0$. Here we discuss how to further improve the accuracy of their values.

Our approach is to use the expectation values of the Monte Carlo paths, subject to the exercise boundary computed by our neural network formulation. More specifically, given the m-th path, the trained neural networks determine its stopping time, denoted as τ_m. Then the price at $t = 0$ can be computed by the mean of the discounted payoffs:

$$u(\boldsymbol{x}^0, 0) = \frac{1}{CM} \sum_{m=1}^{CM} e^{-r\tau_m} f(\boldsymbol{X}_m(\tau_m)). \tag{A.12}$$

Regarding the delta at $t = 0$, we can use the method in [151], which is an adaptation of "pathwise derivative method" [33] to American options:

$$\frac{\partial u}{\partial x_i}(\boldsymbol{x}^0, 0) = \frac{1}{CM} \sum_{m=1}^{CM} \left(e^{-r\tau_m} \sum_{j=1}^{d} \frac{\partial f}{\partial x_j}(\boldsymbol{X}_m(\tau_m)) \frac{\partial (X_j)_m}{\partial (x_i)^0} \right). \tag{A.13}$$

When the underlying asset prices evolve under (6.2), we have $\frac{\partial (X_j)_m}{\partial (x_i)^0} = \frac{(X_j)_m}{(x_i)^0} \delta_{ij}$. We note that the pathwise derivative approach may not be applicable if $\frac{\partial (X_j)_m}{\partial (x_i)^0}$ is not evaluable (e.g. the underlying asset prices do not evolve under (6.2)) or if the payoff function is not differentiable. For such non-applicable cases, we can still obtain the deltas from our trained neural network at $t = 0$.

Using (A.12)-(A.13) to compute the price and the delta at $t = 0$ is also observed in other Monte Carlo style pricing approaches, including the Longstaff-Schwartz algorithm. However, we emphasize that our approach differs from the others. More specifically, (A.12)-(A.13) are not computable unless combined with an algorithm that can determine the exercise boundary on the entire spacetime. In this chapter, our neural network framework

is used to determine the exercise boundary before applying (A.12)-(A.13). In Section 6.7, we demonstrate that our neural network formulation yields a more accurate exercise boundary, and thus more accurate prices and deltas at $t = 0$, compared to the Longstaff-Schwartz algorithm.