DISSERTATION

MODELING AND ANALYSIS OF NANOSCALE SURFACE PATTERNS PRODUCED BY BROAD BEAM ION BOMBARDMENT

Submitted by

Kevin M. Loew

Department of Physics

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Fall 2020

Doctoral Committee:

Advisor: R. Mark Bradley

Martin Gelfand
Patrick Shipman
Mingzhong Wu

ABSTRACT

MODELING AND ANALYSIS OF NANOSCALE SURFACE PATTERNS PRODUCED BY BROAD BEAM ION BOMBARDMENT

When a solid surface is exposed to broad beam ion bombardment, nanoscale patterns may spontaneously form. This physical phenomenon is of interest to both the academic and nanofabrication communities. Ion bombardment has the potential to provide a cost-efficient method of producing nanoscale patterns over a large area. As such, it has gathered substantial interest and has been the focus of numerous studies, both experimental and theoretical. However, despite more than half a century of study, there are still many unknowns which limit the application of this method to fabrication.

In this dissertation, I present contributions to the field of ion bombarded surfaces (IBS). The first is the development of a Python module which facilitates the rapid production and analysis of simulations. This module provides a well-documented tool to allow collaborators to numerically integrate a user-defined partial differential equation, specifically with IBS in mind. Second is a study of dispersive effects on IBS. Dispersion can lead to the formation of raised and depressed triangular regions traversed by parallel-mode ripples, highly ordered parallel-mode ripples, protrusions and depressions that are elongated along the projected beam direction even when there is no transverse instability, and needle-like protrusions that are visually similar to structures observed in experimental studies. Finally, we applied deep learning techniques to estimate the parameters in the underlying equation of motion from an image of a surface exposed to broad beam ion bombardment at a particular fluence. Our trained neural network will allow experimentalists to quickly ascertain the parameters for a given sputtering experiment.

ACKNOWLEDGEMENTS

I want to thank the numerous individuals who supported me in the completion of my research, without whom I would not have completed this degree.

Firstly, I would like to express my utmost gratitude to my advisor and mentor Dr. R. M. Bradley. I cannot adequately express my appreciation for the insight and knowledge he provided throughout my time at Colorado State University. Beyond just academic and professional support, Dr. Bradley acted as a true mentor and offered guidance critical to navigating graduate school as a parent. He helped me strike the balance necessary to grow as both a scientist and as a father.

I would also like to thank the friends and colleagues who took the time, and trips for coffee, to talk through problems and get additional perspective.

Most importantly, I would like to thank my family. Your support and patience has meant more than words can state.

DEDICATION

This dissertation is dedicated to my wife, Kate, who supported me even when it seemed impossible, and to my children, Grayson and Reylyn, whose excitement and curiosity continually inspired me to face the unknown.

Ad astra per aspera.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

Introduction

Nanoscale patterns spontaneously emerge on solid surfaces exposed to broad beam ion bombardment. This physical phenomenon could potentially provide a technique to rapidly fabricate large scale nanostructures. This dissertation describes the research contributions that I have made to this topic during my time at Colorado State University.

In Chapter 2, an overview of key developments within the field of ion bombarded surfaces (IBS) is provided. First, notable experimental findings that influenced the direction and evolution of the field are discussed. Unfortunately, contamination of the surfaces occurs in many experiments carried out before 2005. Therefore, it makes more sense to focus on the bombardment of elemental surfaces instead of taking a purely historical approach. The emphasis is on experimental studies in which great care was taken to preserve the purity of the surface throughout the bombardment process. The theoretical models used to describe the bombardment of these elemental surfaces are then presented. The bombardment of multi-element surfaces is then briefly addressed to make a connection with the contamination mentioned above. Finally, applications of the emergent patterns are presented.

The work presented in this thesis is theoretical. Numerical simulations of a surface governed by a particular equation of motion (EoM) are carried out and analyzed. The mathematical methods used to produce these simulations are discussed in Chapter 3. First, a general overview of numerical integration is provided. Then, the particular integration scheme utilized for the work presented in this dissertation is described. The limitations and potential pitfalls that can emerge when applying this method of integration are the last topics considered in Chapter 3.

In Chapter 4, a high-level discussion of a software tool that I developed to produce simulations of a user-defined partial differential equation (PDE) and that provides a suite of analysis tools to assist in understanding the simulations is presented. The motivation for and benefits

of having this software tool available, not only for myself but also for future members of the research group, are then addressed. Next, a dialogue on the input which the user needs to provide to the tool to produce simulations and the potential problems that can emerge as well as possible solutions are covered. Additionally, some insight into some of the techniques used by the framework to provide accurate results quickly is included. Last is a brief discussion of some of the built-in functions which assist in the analysis of the simulations.

Triangular structures have been observed in many experiments and cannot be reproduced via the traditional theoretical model [1–8, 10, 11, 33]. There are also experimental results which indicate that dispersion has a more substantial impact on the pattern formation than previously thought [12]. These findings both motivated the study presented in Chapter 5. We included the lowest order dispersive terms, along with the terms already present in the traditionally accepted model. This expanded model produces four key features: the generation of ripple patterns with significantly higher order than what is otherwise produced, the formation of transient triangular protrusions with ripples superimposed on them which closely resemble experimentally observed structures, ripples with wavevectors that are perpendicular to the beam projection in the $x-y$ plane, and the production of so-called "nano-needles". Both the triangular and needle structures structures are experimentally observed [1–8, 10, 11, 13–15, 33] but not produced by the traditionally accepted equation of motion (EoM).

The patterns that emerge on a surface depend on many factors such as the target material, the ion species, the ion energy, and the orientation of the ion beam relative to the surface [16–22]. These factors all influence the parameters in the equation of motion and alter the observed pattern. Current work within the field of machine learning (ML) focuses on the study of algorithms that utilize statistical inference to identify and associate meaning to patterns that exist within data [135]. Ion sputtering produces nanoscale patterns; as such, ML algorithms that can interpret these patterns are of direct interest. A brief introduction to ML concepts and methodology used is presented in Chapter 6. This includes a discussion of using well-understood data to "train" a ML algorithm through gradient descent. Next, a discussion of

what artificial neural networks are, what their structure is, and why they have become such an active area of study is included. Lastly, some of the challenges that exist when trying to implement machine learning algorithms and train a model are considered.

In Chapter 7, machine learning is used to estimate the parameters in the EoM discussed in Chapter 2. A deep artificial neural network was trained to estimate the parameters in the generally accepted equation of motion for surfaces exposed to broad beam ion bombardment. This deep learning model uses a single image of the surface to estimate all five parameters in the equation of motion with root-mean-square errors that are under 3% of the parameter ranges used for training. This provides a tool that will allow experimentalists to ascertain the parameters for a given sputtering experiment quickly. It could also provide an independent check on other methods of estimating parameters such as atomistic simulations combined with the crater function formalism.

Chapter 2

Background

2.1 Experimental Findings

Before presenting original work, it is necessary to consider the current landscape of the field and how it arrived at the current state of affairs. An overview of the key experimental results and developments within the field is provided in this section.

The physical phenomenon of spontaneous nanopattern formation on solid surfaces exposed to broad beam ion bombardment has been a point of interest for 60 years. In 1960 Cunningham et al. [24] showed that bombarding a metal surface with Ar^+ ions could produce rough, disordered topographies of ripple patterns. Two years later, a study by Navez et al. [25] identified vital features of the pattern formation. A beam of ionized air was used to bombard a glass surface. Intermediate beam angles relative to the surface normal caused ripples with wavevectors $\vec{k}$ parallel to the projection of the beam onto the surface to emerge. This particular type of pattern is called "parallel mode ripples". This behavior indicates a crucial relationship between the orientation of the ion beam and the emergent patterns on the solid surface.

Further studies have shown that beam angles below a critical threshold cause the surface to smooth [26, 27, 29–31]. Beyond that critical angle, parallel mode ripples form. Then, as the beam angle is increased even further, the ripple orientation rotates, causing the wave vector $\vec{k}$ to now be perpendicular to the beam projection onto the surface [26, 27, 32]. This type of ripple pattern is named "perpendicular mode ripples".

Figure 2.1 shows experimental results of Si surfaces bombarded with 500 eV Ar^+ at a variety of angle of incidences, θ. No clear patterns form until the angle of incidence surpasses 55°. At this point parallel mode ripples are observed on the surface. Eventually the ripples "rotate" and perpendicular mode ripples are observed for θ values 80° and 85°.

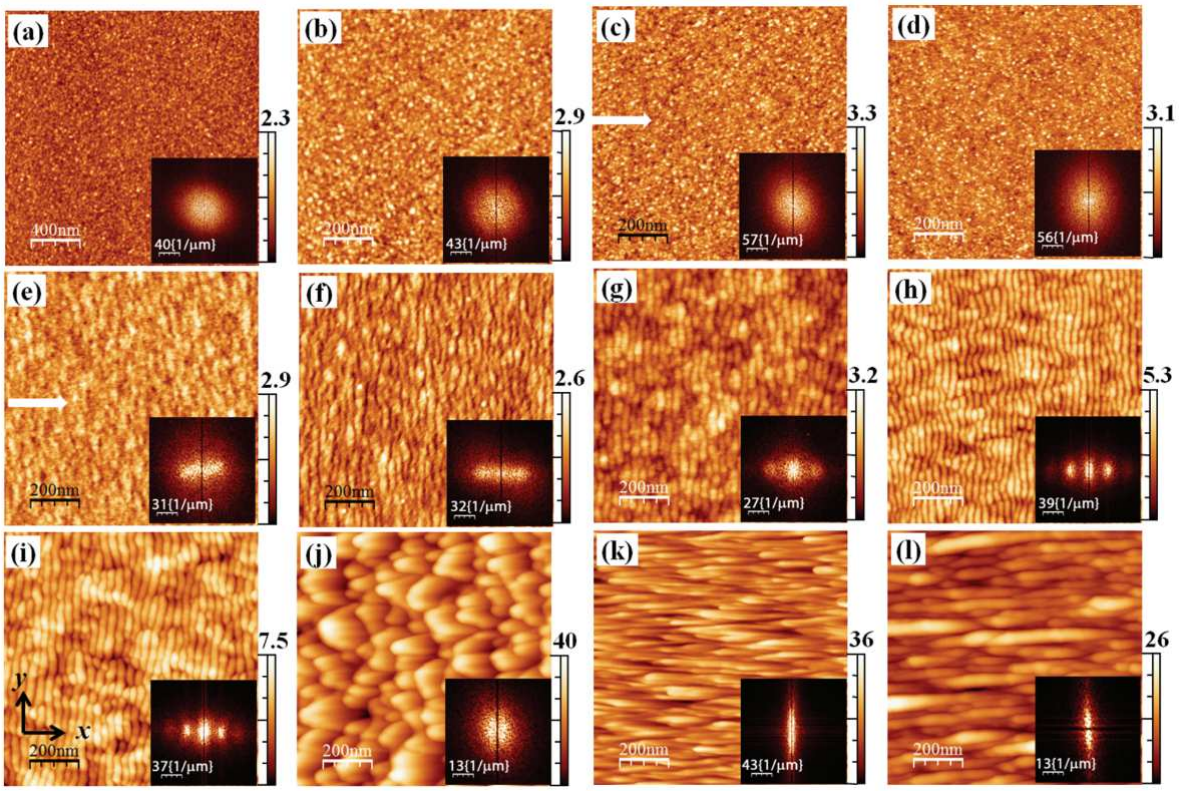

Figure 2.1: AFM images of 500 eV Ar^+ sputtered Si surfaces for ion incidence angles (a) 0°, (b) 25°, (c) 35°, (d) 45°, (e) 55°, (f) 57°, (g) 60°, (h) 65°, (i) 70°, (j) 75°, (k) 80°, and (l) 85°. Inset shows the corresponding spectral densities. White arrows indicate the ion beam direction for oblique incidence ion irradiation. This image originally appeared in Ref. [33].

Unfortunately, at this point, we must diverge from a purely chronological recount. Virtually every early IBS experiment had unintended deposition of impurities onto the surface. In 2005, Ozyadin et al. [34] established that these impurities have a significant influence on the pattern formation. In this study, the growth of Mo "islands" on a Si surface during 1 keV bombardment of Ar^+ ions at normal incidence was studied. When the atomic flux of Mo incident on the surface was large, disordered arrays of nanoscale mounds spontaneously formed on the surface. Interestingly, Mo collected on the tops of these mounds. By contrast, when the atomic flux was reduced to zero, no patterns emerged—indicating quite clearly that Mo had to be present on the surface for nanodots to form. Therefore, the composition of the surface is critical to the emergence of these patterns. Numerous studies since Ozyadin et al.'s discovery have probed the influence that impurities have on the surface morphology [35–38].

The contaminants on the surface could come from a variety of sources, such as a highly divergent ion beam. As the ion beam propagates through space, it diverges. If it diverges enough, some of the ions can impact the walls of the chamber containing the target. These chambers are typically constructed from stainless steel. Therefore, the ions which strike the chamber walls sputter stainless steel from them. Some of the sputtered material would then be deposited on the target surface. Before the influence of impurity co-deposition was understood, the changes in the surface topography were attributed to the beam divergence [39].

The impact of impurities on the emergent patterns raises concerns about the experimental setup used and requires careful consideration during experimental design. For example, if the structure used to hold the sample can be sputtered and produce a significant deposition flux, the surface dynamics can be changed, even if the holder is composed of the same material as the sample [40]. Another component of an experiment that can lead to impurity co-deposition is the grid used to accelerate the ions. The grid can be struck by ions and introduce contaminants as was the case in the experiments conducted by Ozaydin et al. [34].

2.2 Bombardment of Elemental Targets

Due to the significant influence that the impurities have on the surface dynamics, it is challenging to extract understanding of the physical mechanisms from experiments in which impurities are deposited during sputtering. That is not to say it is impossible; progress has been made in understanding ion bombardment with impurity co-deposition [35–38, 41]. However, the impurity species, rate of co-deposition, and direction the impurities move along before they strike the surface need to be known to understand the dynamics of these surfaces.

The discussion in this section focuses on experimental studies of elemental targets, specifically experiments of Si surfaces bombarded by noble gas ions in which precautions have been taken to ensure the purity of the surface material. By reducing the number of atomic species present during the bombardment, the number of physical parameters in the EoM is reduced. Additionally, the EoM which describes an elemental target is simpler than an EoM which accounts for impurities and co-deposition.

Noble gas IBS of Si targets is a common experimental setup and has yielded insight into spontaneous pattern formation. One finding is the existence of a critical angle of θ_c. When the angle of incidence between the beam and the target normal is below θ_c, no patterns emerge, and instead, the surface flattens. A study by Madi et al. found a critical angle $\theta_c \approx 48°$ for 250 keV Ar^+ ion bombardment of Si [26, 27]. For $\theta > \theta_c$, ripples emerge on the surface. Additional studies of IBS on Si have found similar values of θ_c [29–31].

Another key observation is the relationship between the orientation of the ripples and the beam angle. Between θ_c and a second critical angle $\theta_{c,2}$, parallel mode ripples emerge on the surface. Lastly, when the beam angle is above $\theta_{c,2}$, perpendicular mode ripples emerge on the surface. Experimental work by Madi et al. has determined that $\theta_{c,2}$ is in the range $[75°, 85°]$ [26, 27, 32]. Both of the critical angles depend on the ion energy, ion species, and the target material.

In addition to the possible patterns which can emerge, it is necessary also to consider how the surface changes with time. A representative IBS experiment with a θ value between θ_c and $\theta_{c,2}$ will pass through three distinct regimes [42]. These are the linear regime, the coarsening

regime, and the saturation regime. Sharp transitions between these regimes do not exist. Furthermore, the duration of each regime depends on numerous factors such as ion flux, ion energy, the angle of incidence, and the temperature of the sample.

In the linear regime, the surface can be modeled well by a linear theory, which will be discussed at length in Sec. 2.3.2. A distinctive feature of this regime is that the surface roughness increases exponentially with time. Exponential change is experimentally observed and measured using grazing incidence small angle X-ray scattering (GISAXS) [43–46]. The patterns that emerge in this regime are ripples with roughly sinusoidal form. The ripple wavelength does not change appreciably, and the surface exhibits up-down symmetry. The growth rate of the surface width depends on the target material, ion species, ion energy, and the angle of incidence between the target and incident ion beam. The linear regime lasts longer for θ values slightly above θ_c than in experiments with θ values well above the critical angle [26,47]. Due to the many contributing factors, the duration can vary greatly. However, for 700 eV Ar^+ bombardment of Si, the duration of the linear regime is a fluence on the order of 10^{17} ions/cm^2 [1].

The next regime is the coarsening regime. In this regime, the dynamics deviate substantially from the linear model, and the nonlinearities play an essential role. The change of the ripple amplitude slows down and instead adheres to a power-law scaling [26, 27, 48, 49]. The ripple wavelength also scales according to a power-law and increases with respect to time in this regime. Another notable feature is that the ripple profile changes as well. These changes are sensitive to the angle of incidence in an experimental setup. For angles less than $\theta \approx 75°$ and greater than θ_c, the ripples develop into "humps" [48–51]. These experimentally observed humps closely resemble parabolic arcs connected by sharp cusps. At this point, the up-down symmetry observed in the linear regime is absent. These humps are not observed in experiments in which a larger angle of incidence is used. Instead, the ripples coarsen into terraces with two selected slopes [2, 50–63]. Proposed explanations for terracing include shadowing [50, 51], the formation of under compressive shocks [22, 64], and ion reflection/redeposition [53].

The final regime, and the least studied, is that of saturation. As the name indicates, in this regime, the ripple amplitude and characteristic lateral length scale stop increasing. This phenomenon is known as "interrupted coarsening" [42, 65–67]. Unfortunately, there are significantly fewer experimental studies that look at this regime. It can take a very long time for a bombarded surface to reach this final regime.

2.3 Theoretical Considerations

The spontaneous pattern formation that comes from IBS has motivated many theoretical efforts to understand this physical phenomenon. To facilitate the discussion of these approaches, I will categorize them into two groups: atomistic and continuum.

Molecular dynamics (MD) is a type of atomistic simulation method which is useful for analyzing the physical movements of atoms and molecules. The atoms and molecules are allowed to interact for some duration of time, during which the trajectories of the atoms are calculated and the positions updated. The trajectories are typically determined by numerically solving Newton's equations for all of the interacting atoms and molecules. The potential which governs the forces that affect a particle are calculated using quantum mechanics [68]. In the the particular case of ion bombardment, the solid surface consists of an assembly of atoms and molecules. When an ion strikes the solid, the particles interact.

MD simulations provide highly accurate models of ion bombarded surfaces but have a significant computation cost. Each simulation utilizes an assembly which consists of a large number of particles. The complexity of these systems generally means that the system cannot be solved analytically, instead the potentials between all of the particles in the assembly and the incident ion need to be calculated with numerical methods. Quite simply, due to the computational requirements for the implementation of this method, MD simulations are only useful for low energy ions and simple monoatomic or diatomic target materials at early times and short length scales.

Another atomistic method is commonly employed to overcome some of these computational limitations. Rather than calculate the interactions between all of the particles in the assembly and with the incoming ion as is done in MD simulations, this method statistically samples a discrete number of "moves" to simulate the evolution of the surface. Due to the stochastic nature of the sampling, this method is commonly referred to as the Monte Carlo (MC) method. Statistically sampling the possible "moves" provides a very significant decrease in computational resources required and, as a consequence, this method can be used to analyze longer times and larger length scales than is possible with MD methods. MC methods have been used quite successfully to model solid surfaces exposed to ion bombardment [69].

Continuum theories further increase the length and time scales that can be considered. The height of the surface as a function of position and of time is described by continuum models. The continuous height field $h(x, y, t)$ describes the height of the surface above a point (x, y) at a time t. An equation of motion which describes the time evolution of the surface h can be constructed. Continuum models offer multiple advantages over atomistic models. Unfortunately, the PDE is typically not exactly solvable. If this is the case, numerical integration can be conducted to approximate the time evolution of the surface.

While continuum models have significant advantages, limitations still exist. For example, when a numerical integration is performed, error is introduced from the approximation of that integral. The surface itself consists of discrete particles like the MD and MC methods consider; this information is obscured by approximating the surface as continuous. Another source of error comes from the truncation of the EoM. As is common in physics, we tend to truncate the EoM to a particular order rather than utilize the full nonlinear EoM; this introduces additional sources of error. Lastly, any faults or limitations of the underlying model directly affect the EoM.

2.3.1 The Sigmund Model

Continuum theories were used for all of the work presented in this dissertation. Therefore, the underlying model from which the EoM was derived needs to be addressed. In 1973 Peter

Sigmund proposed a model for the impact of an ion on a solid surface, and the material which is sputtered from the surface as a result [70].

When a sufficiently energetic ion strikes a surface it penetrates the surface and moves into the solid. As the ion penetrates the solid, it collides with atoms. The atoms in the solid recoil and collide with other atoms in a so-called collision cascade. Collisions continue as the ion continues to propagate through the solid until the ion has dissipated all of its energy and comes to rest. The Sigmund model predicts that energy from the incident ion is on average distributed to the surrounding atoms according to ellipsoidal contours.

Figure 2.2 illustrates an ion impact on a solid surface. The surface height of the solid is described by $h(x, y, t)$. The ion moves along the $-\hat{e}$ direction and strikes the surface at a location $\vec{r}'$ as measured from the origin O. The ion penetrates into the solid, depositing energy as it moves through it. The point of maximal energy deposition is at the location $\vec{r}_0 = \vec{r}' - a\hat{e}$. The energy is not equally distributed along the parallel and perpendicular directions. This asymmetry produces ellipsoidal contours of equal energy deposition instead of spherical ones. The orientation of the ellipsoids depends on the beam direction $\hat{e}$, but not on $h(x, y, t)$. The average energy per unit volume deposited at a point $\vec{r}$ from many ion impacts at $\vec{r}'$ is given by

$$E(\vec{r}, \vec{r}', t) = E_0 e^{-d_\parallel^2/2\alpha^2 - d_\perp^2/2\beta^2}, \tag{2.1}$$

where $d_\parallel$ and $d_\perp$ and the parallel and perpendicular components of $\vec{d} = \vec{r} - \vec{r}_0$, respectively. α and β are parameters that describe how quickly the energy deposited attenuates along the parallel and perpendicular directions, respectively. The Sigmund model also makes the assumption that the erosion rate at a particular point on the surface is directly proportional to power per unit volume deposited at that point.

Unfortunately, the Sigmund model of ion sputtering has limitations. If the energy of the incident ions is sufficiently small, sputtering is negligible, and the Sigmund model fails to explain pattern formation. The Sigmund model does not apply to solids that are not amorphous since the crystal lattice can significantly affect the contours of equal energy deposition and therefore

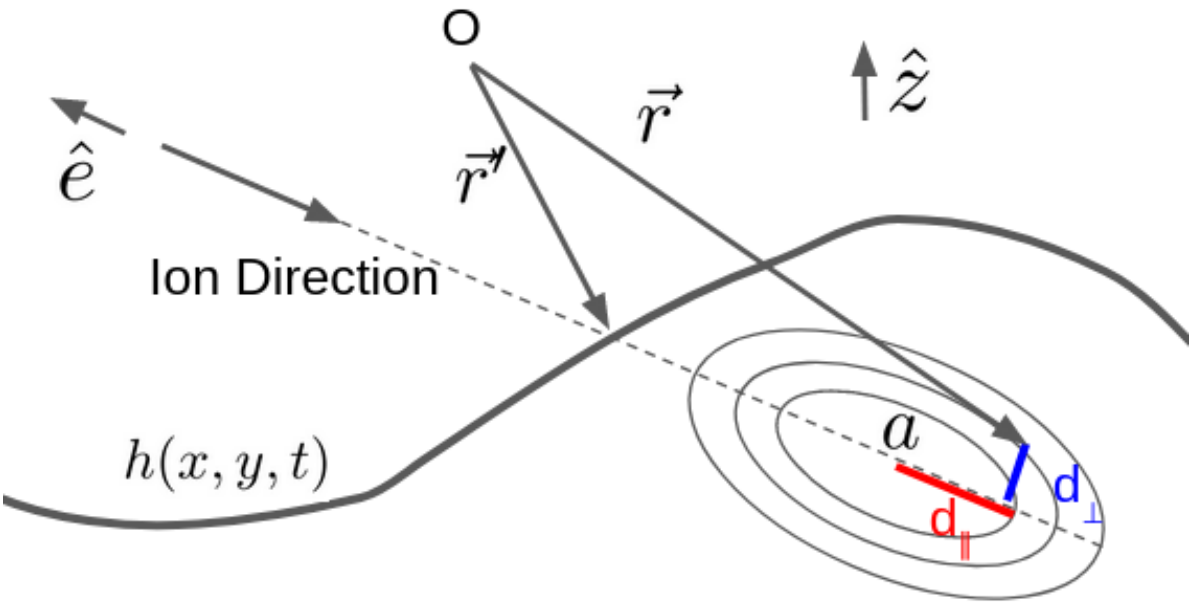

Figure 2.2: A surface with height $h(x, y, t)$ is struck by an ion. The ion moves along the $-\hat{e}$ direction and strikes the surface. The location of the impact is described by a vector $\vec{r}'$ as measured from the origin O. The point of maximal energy deposition occurs at the location $\vec{r}_0 = \vec{r}' - a\hat{e}$. The ellipses represent contours of equal energy deposition. The vector $\vec{r}$ points to the location at which the power per unit volume is to be calculated. The vector $\vec{d}$ points from $\vec{r}_0$ to $\vec{r}$ and can be broken up into components parallel to the beam direction and perpendicular to it, $d_\parallel$ and $d_\perp$, respectively.

change the sputter yield [72]. The Sigmund model also takes the power distribution to be independent of the surface shape; in both the case of high energy ions and for beams with a high angle of incidence, this is a poor approximation [73].

Despite these limitations, the Sigmund model is widely used within the field [16, 18, 70]. Amorphous materials exposed to moderate energy ion bombardment at intermediate angles of incident are common in experimental studies [1]. Additional information about the Sigmund model can be found in Ref. [20].

2.3.2 Bradley-Harper Theory

The Sigmund model provided insight into sputtering, but it cannot fully explain the nanopattern formation observed in experiments. In 1988, Bradley and Harper developed a theory, the so-called Bradley-Harper (BH) theory, which provided an explanation for the formation of ripple patterns on sputtered surfaces. This theory focuses on two physical features, the curvature dependence of the sputter yield and surface diffusion [16].

Figure 2.3 illustrates the curvature dependence of the sputter yield. The amount of material sputtered at a trough, O, or a peak, O', depends on the amount of energy deposited at that point. In both cases we consider three ion impacts: one directly below the point of interest and two off-center impacts with the same sideways displacement. For both O and O', the points of maximal energy deposition for the on-center impacts have the same distance to the surface point directly above the impact. However, this is not true for the off-center impacts. In both cases the ion strikes the surface and penetrates it having a point of maximal energy deposition a distance a beneath the surface. This point of maximal energy deposition is labelled A for the surface with positive curvature and A' for the surface with negative curvature. From the image, we can see that the distance $|OA| < |O'A'|$. Eq. (2.1) states that the energy deposited at a point depends on the distance of that point to the point of maximal energy deposition. As a consequence, more energy is deposited at the trough than the peak, and so more sputtering

occurs at the trough than at the peak. This indicates that the sputter yield depends on the surface curvature and that the surface is unstable.

Surface diffusion is also included in the BH theory. If only the curvature dependence of the sputter yield were included in the theory, extremely small wavelengths would have very large growth rates. To eliminate this unphysical behavior, Bradley and Harper included the effects of surface self-diffusion. The thermally activated mobility of particles on the target surface causes the surface to smooth over time. This physical feature prevents the short-wavelength ripples from having unbounded growth rates.

Using these physical features and assuming that the surface height is a slowly varying function of x and y leads to the EoM

$$h_t = -v_0 - v_0' h_x + \kappa_1 h_{xx} + \kappa_2 h_{yy} - B\nabla^2\nabla^2 h \tag{2.2}$$

for a nominally flat initial surface where the direction of the incident ions has polar angle $\theta \geq 0$ and zero azimuthal angle. The subscripts on h denote partial differentials, i.e., $h_t = \frac{\partial h}{\partial t}$, and $\nabla^2 \equiv \partial_x^2 + \partial_y^2$. The first term, $-v_0$, is the erosion rate of a completely flat surface. $-v_0' h_x$ is the erosion rate that results from a sloped surface. $\kappa_1 h_{xx}$ and $\kappa_2 h_{yy}$ are terms which account for the curvature dependence of the sputter yield. The final term $-B\nabla^2\nabla^2 h$ describes the smoothing effect of surface diffusion. The parameters v_0, v_0', κ_1, and κ_2 are all constants which depend on the target material and the ion species, energy and angle of incidence [16–22]. B does not depend on the nature of the ion beam if the surface diffusion is thermally activated. Although not originally considered, it is now known that ion-induced mass redistribution [81, 83, 84] and ion implantation [85, 86] also contribute to the second-order terms and ion-induced viscous flow near the surface contributes to the fourth-order term [87].

Equation (2.2) can produce nanopatterns. If both κ_1 and κ_2 are positive, there is no surface instability, and the surface smooths. This flattening occurs in the case of normal incidence and for angles of incidence below a threshold value. However, this flattening cannot occur in the BH theory. If $\kappa_1 < 0$ and $\kappa_1 < \kappa_2$, ripples with wave vector $\vec{k}$ parallel to the x axis emerge; these are

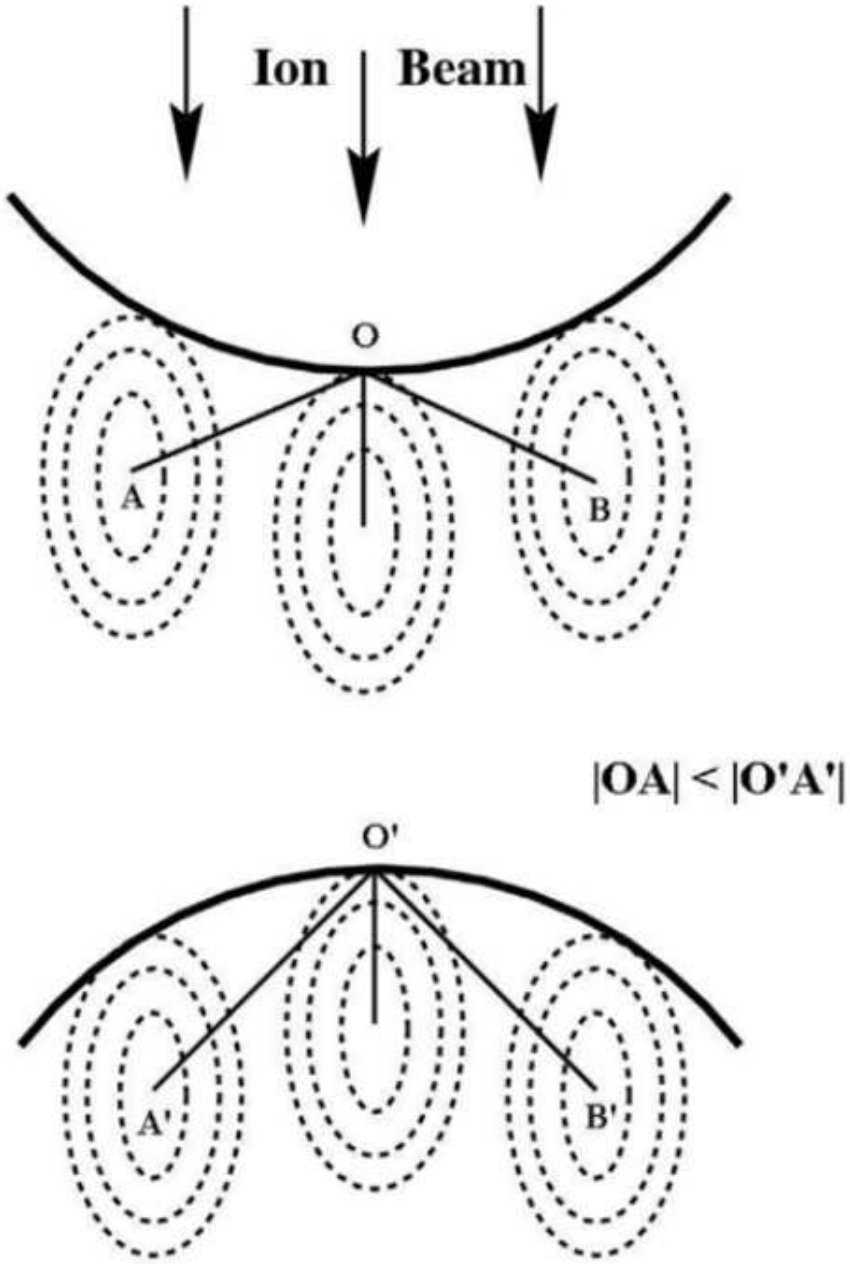

Figure 2.3: Graphical representation of the ellipsoidal contours of equal energy deposition for normal incidence ion bombardment of concave and convex surfaces. Reproduced from Ref. [1].

called parallel-mode ripples. If $\kappa_2 < \kappa_1$ and $\kappa_2 < 0$, on the other hand, the selected wave vector is along the y direction and so-called perpendicular-mode ripples form [16, 74].

In addition to pattern formation, the BH theory can explain ripple rotation. When the beam orientation passes through a particular angle, the ripple wave vector rotates by 90°. This occurs when $\kappa_2 < \kappa_1$ and $\kappa_2 < 0$. This experimentally observed phenomenon agrees with the prediction of the BH theory. The BH theory marked a breakthrough in continuum models and is widely used in the field even to this day.

2.3.3 Kuramoto-Sivashinsky Equation

The BH theory is successful in the prediction of parallel and perpendicular mode ripples; however, this theory is also imperfect. The variation in the sputter yield causes troughs to become deeper and peaks to become higher, and the ripple amplitude to grow exponentially in time. To account for the saturation in the amplitude of the ripples that is observed experimentally, the leading order nonlinear terms must be added to the linear BH equation of motion [17]. After transforming to an appropriately chosen moving frame of reference, this gives the anisotropic Kuramoto-Sivashinsky (KS) equation

$$u_t = \kappa_1 u_{xx} + \kappa_2 u_{yy} - B\nabla^2\nabla^2 u + \lambda_1 u_x^2 + \lambda_2 u_y^2, \tag{2.3}$$

where $u(x, y, t)$ is the height of the surface above the point (x, y) in the $x - y$ plane at time t. The constants κ_1, κ_2, λ_1 and λ_2 depend on the angle of incidence of the ion beam θ and were originally computed using the Sigmund theory of sputtering [16, 18, 70]. B is a constant which corresponds to the surface diffusion.

The wavelength of these ripples are determined by the linear terms in Eq. (2.3). The amplitude of these ripples grows exponentially at early times then saturates. Eventually the surface morphology exhibits spatial-temporal chaos. Another pattern that can form is "bowl" like formations. These occur when both $\kappa_1 < 0$ and $\kappa_2 < 0$ and $\kappa_1 \approx \kappa_2$. The depth of the bowls grows

with time at first, then ceases to continue to grow. Given enough time, the surface will exhibit spatial-temporal chaos.

2.4 Alternative Theories

2.4.1 Mass Redistribution

While the Sigmund model and BH theory with the lowest order nonlinear corrections have proven to be extremely useful, they do not fully describe the patterns produced by ion sputtering [17]. In BH theory, the curvature dependence of the sputter yield produces ripple patterns, but what if the arriving ions do not have enough energy to cause sputtering to occur? A reasonable thought would be that if no material is sputtered, no patterns will form. However, ripple structures are still observed in experiments that are in the low-energy regime [88, 89]. The BH theory also predicts the formation of ripples for all nonzero angles of incidence. However, in experimental studies, ripple formation is only observed for θ values exceeding a critical angle. As such, other competing models have been developed.

Mass redistribution, such as the Carter-Vishnaykov (CV) effect, is one such model. The CV model considers momentum transfer between the incoming ions and the atoms near the surface of the target, which causes the affected atoms to move. Depending on the angle of incidence between the incoming ions and the surface, parallel mode ripples can form, or the surface can smooth. A full derivation of this model can be found in Ref. [81].

The linearized equation of motion is

$$h_t = \Omega\mu J[\cos(2\theta)h_{xx} + \cos^2(\theta)h_{yy}] - B\nabla^2\nabla^2 h, \tag{2.4}$$

where Ω is the atomic volume, μ is a constant of proportionality, and J is the magnitude of the flux of incident ions onto the surface. The final term $-B\nabla^2\nabla^2 h$ describes the smoothing effect of surface diffusion, just as it did in the BH theory. The CV effect produces the terms of second order in the spatial derivatives.

A key feature of this model is that for $\theta < 45°$, the EoM causes the surface to flatten. This phenomenon is seen in experiments, although the critical angle is not exactly $45°$ [26, 27, 32]. Another notable feature of this model is that the mass is conserved. No sputtering occurs, which means that no material leaves the surface, unlike in the BH theory. There is also evidence that for some energy regimes and some beam angles, the CV effect is more critical to the dynamics than curvature dependent sputtering [27, 28].

In truth, both curvature dependent sputtering and mass redistribution need to be considered. In experiments, as θ approaches 90°, the ripple orientation changes and perpendicular mode ripples are observed. This phenomenon can never occur with mass redistribution alone. Thus, there must be regimes for which the predominant physical mechanism driving the dynamics is sputtering and not mass redistribution.

2.5 Uses of Patterns Produced by Ion Bombardment

The spontaneous emergence of patterns due to IBS is not only of academic interest but also has significant potential applications. Most applications use the nanopatterns as templates to facilitate the subsequent fabrication of other nanostructures. For example, rippled structures have been used as templates to grow nanostructured thin Ni [80], Co [95], and multilayer films [96]. This has allowed researchers to probe how the magnetic properties of a thin film are affected by the ripple wavelength [97]. Liedke *et al.* showed that above a wavelength threshold, a corrugated Permalloy film behaves like a flat film, but below it, they found an induced magnetic anisotropy [98]. Another use of rippled templates is the fabrication of nanowires. Rippled Si surfaces can also be used to grow Ag or Au nanowires [99]. The patterned surfaces can be used as templates to grow metal nanoparticles [100, 101].

Preferential sputtering can be used to generate arrays of nanoscale protrusions with the preferentially sputtered material collected in the valleys of the structures [100,101]. IBS provides a fast and cheap method to produce large-scale nanostructures with adjustable periodicity. The periodicity can be changed simply by changing the beam angle.

The last application we will discuss is the use of rippled Si surfaces to analyze the absorption of biomolecules. In 2012, Sommerfeld *et al.* carried out a study of the absorption of human plasma fibrinogen [102]. The study showed that the absorption takes place in mostly globular conformations on both amorphous, flat Si surfaces and rippled Si surfaces with sufficiently long wavelengths. If, however, a short wavelength rippled surface was used, the absorbed fibrils were aligned along or across the ripples.

There are many more applications and potential applications. However, my goal is not to give a detailed list of related studies, but to motivate the theoretical studies that we have done. To be able to use and control spontaneous nanopattern formation, it is necessary to understand the underlying physical mechanisms contributing to it.

Chapter 3

Methods for Numerical Integration

Continuum models of IBS are used to describe behavior observed in experimental studies. Most often, the EoMs have no analytical solution and need to be numerically integrated. In this chapter, an overview of numerical methods of integration is presented. Additionally, an in-depth look at the exponential time-differencing integration method is provided, and potential problems with implementing exponential time-differencing for IBS are considered.

3.1 Overview of Numerical Integration

While the ultimate goal is to numerically integrate PDEs, the methods included in this section are more easily understood in reference to an ordinary differential equation (ODE). The expansion of these methods to PDEs is discussed after the introduction and explanation of the methods.

Consider an ODE of the form

$$u_t = f(t,u), \tag{3.1}$$

where $u = u(t)$ is a real number value of some function at a time t. The subscript t on u indicates a partial derivative with respect to time. Unfortunately, many differential equations are not analytically solvable. As such, numerical methods of integration must be used to approximate the solution. The simplest integration method is Euler's method. This technique utilizes the knowledge that the derivative u_t describes the slope of u relative to t at a particular value of t. As such, for a small time step h, the next value of u at a time $t+h$ can be approximated using the equation

$$u_{n+1} = u_n + hf(t_n,u_n), \tag{3.2}$$

where n represents an integer number of steps, each separated by a small time interval h. u_n is the value of u at time $t_n = n \times h$. This method offers an easy to understand and reasonably

accurate integration method. However, it is an approximation. Starting from a value u_n at t_n, the next "step" can be found by "advancing" the value along first derivative $f(t_n, u_n)$ multiplied by a small step of h. This linear approximation of u_{n+1} will deviate from the actual value, especially for surfaces with large curvature and for larger values of h. The local truncation error associated with Euler's Method is of order h^2 [103].

Figure 3.1 shows the application of Euler's Method to the ODE $u_t = \sin t$ on the domain $t \in [0, 2\pi]$ with the initial condition $u(t = 0) = -1$. This ODE can be solved exactly and yields the equation $u = -\cos(t) + C$, where C is a constant determined by the initial condition. In this case, $C = 0$. The exact solution for u is displayed in the figure with a dashed black line. When the time step h is large, Euler's method does not perform well. The numerical integration for a time step of $h = 1.0$ is shown in red. This curve follows the general shape of the original function but deviates significantly from it. When the time step is reduced, for example to $h = 0.25$ (shown in blue), the numerical integration approximates the solution of the differential equation much better. The approximation is further improved by further reducing the time step to $h = 0.05$ (in green).

Euler's method works reasonably well. Nevertheless, to accurately describe rapidly changing solutions, very small time steps are required. This increases the number of calculations that must be conducted and the method can be computationally prohibitive. The desire to overcome this limitation has prompted the development of numerous extensions of Euler's method.

A common type of expansion is higher-order Runge-Kutta (RK) methods. RK methods are used to approximate solutions of ODEs. Euler's method is actually the simplest RK method. A common second order RK method (RK2) is the so-called improved Euler method [104]. This method uses the slope at the known point (t_n, u_n) and the slope at the point acquired using Euler's method. The average of these slopes is used to estimate u_{n+1}. The first slope is calculated by

$$k_1 = f(t_n, u_n). \tag{3.3}$$

k_1 is then used to solve for the second slope at a point $(t_n + h, u_n + hk_1)$. This is found by

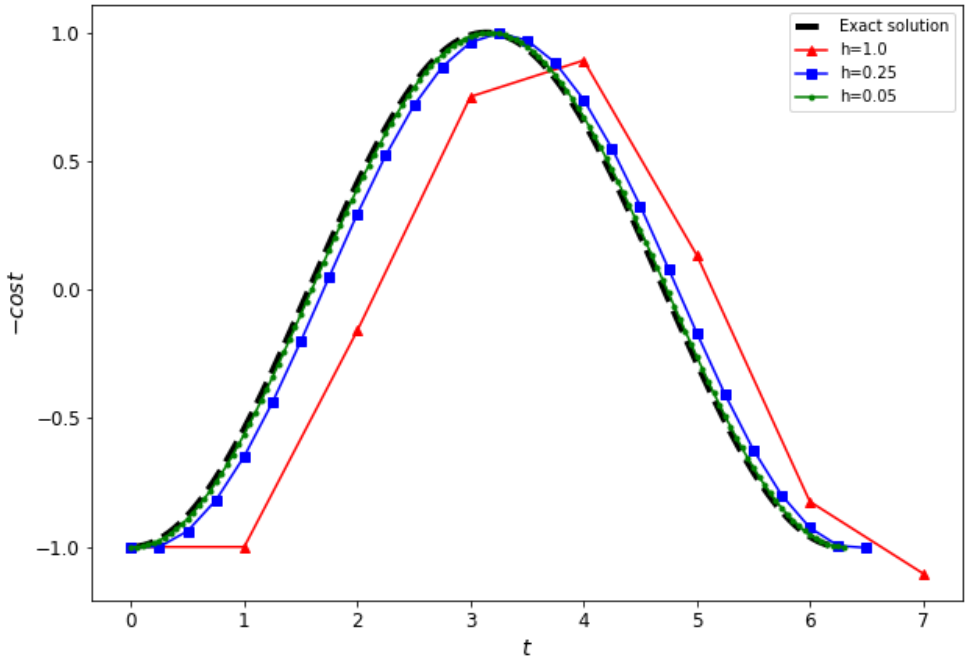

Figure 3.1: Euler's method approximation of $u_t = \sin t$ vs t (black) with a time step h of 1.0 (red), 0.25 (blue), and 0.05 (green). The initial condition is $u(0) = -1$. The domain size was 2π.

$$k_2 = f(t_n + h, u_n + hk_1). \tag{3.4}$$

The next value u_{n+1} is then calculated by

$$u_{n+1} = u_n + \frac{h}{2}(k_1 + k_2). \tag{3.5}$$

The local truncation error of the improved Euler method is of order h^3.

Figure 3.2 shows the estimation of a function using the improved Euler method. Consider the equation $\frac{du}{dt} = f(t,u) = 4t^3 + 3t^2 + 2t$. The blue square represents the starting point and a known value of $u(t)$, which is $(0,0)$. The improved Euler method is used to estimate the exact solution, which is shown as a dashed black line, at $t = 5$. First, $f(0,0)$ is solved to get k_1. The blue line has slope k_1 and is followed a distance $g = 5$ to get to the location $(5,0)$. Then k_2 is solved at this point. The green line starts from the original position and has slope k_2. We see that the Euler approximation underestimates the true value and the estimate using k_2 overestimates it. The magenta line has slope $\frac{k_1+k_2}{2}$. When the mean slope of the two points is used it produces a final estimate that is closer to the actual value of $u(t)$ than the estimate produced by Euler's method.

Using even more calculations of the slope can further improve results. The fourth-order Runge-Kutta (RK4) method is a typical example of this [105]. RK4 uses four distinct points to calculate slopes and estimate u_{n+1}. The four slopes used are:

$$k_1 = f(t_n, u_n), \tag{3.6}$$

$$k_2 = f(t_n + \frac{h}{2}, u_n + \frac{h}{2}k_1), \tag{3.7}$$

$$k_3 = f(t_n + \frac{h}{2}, u_n + \frac{h}{2}k_2), \tag{3.8}$$

and

$$k_4 = f(t_n + h, u_n + hk_3). \tag{3.9}$$

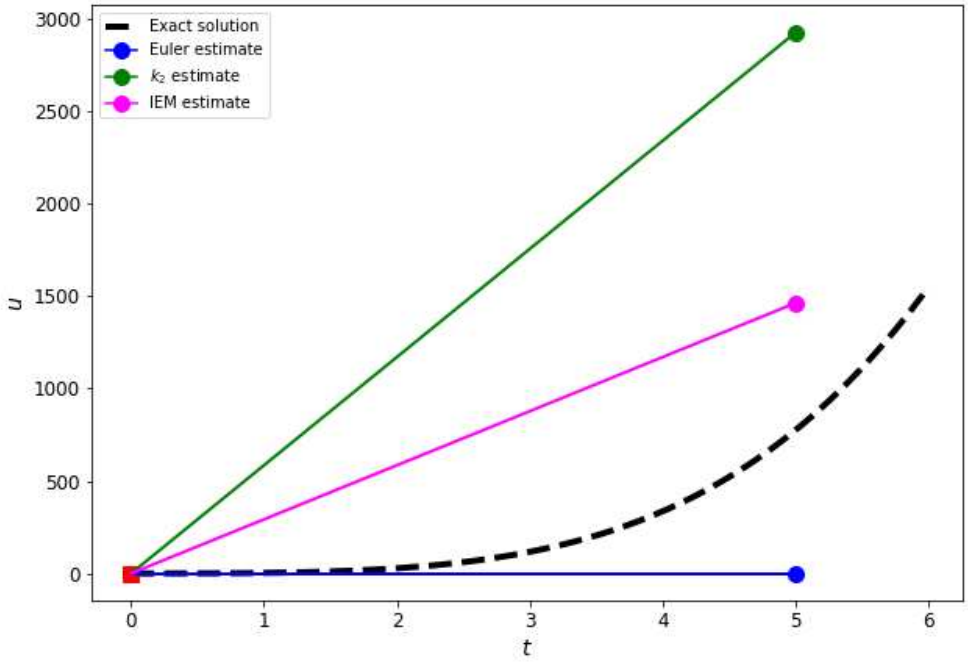

Figure 3.2: Improved Euler method approximation of $u_t = 4t^3 + 3t^2 + 2t$ (black) with a time step h of 5.0. The initial condition is $u(0) = 0$. The blue line has slope k_1, the green line has slope k_2, and the magenta line has slope $(k_1 + k_2)/2$.

k_1 is the slope at the original point (t_n, u_n), exactly as was found with Euler's method. This slope is then used to conduct a "half step" to the point $(t_n + \frac{h}{2}, u_n + \frac{h}{2}k_1)$. At this location k_2 is found. Then starting from the original point (t_n, u_n), another half step is taken along a line with slope k_2 to arrive at $(t_n + \frac{h}{2}, u_n + \frac{h}{2}k_2)$. The slope at this new location, k_3, is then found using Eq. (3.8). The fourth point is determined by taking a full step from the initial position (t_n, u_n) along a line with slope k_3, thus arriving at $(t_n + h, u_n + hk_3)$. A fourth slope k_4 is then found at this point. These four slopes are combined as a weighted average and the new location u_{n+1} is calculated using

$$u_{n+1} = u_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4). \tag{3.10}$$

The new slope $(k_1 + 2k_2 + 2k_3 + k_4)/6$ more accurately reflects the mean value of u_t over the interval of $t_n \rightarrow t_n + h$. This yields a more accurate estimate of the value of u_{n+1} than both the Euler and improved Euler methods. The local truncation error of the RK4 method is of order h^5 [106]. If a higher degree of accuracy is required, more points can be added, but at the cost of computational efficiency. More information about Euler's method, RK methods, and numerical methods in general can be found in [105, 107].

Figure 3.3 shows the application of two RK4 steps to the ODE $u_t = 4t^3 + 3t^2 + 2t$ with the initial condition $u(t = 0) = 0$ and a time step $h = 5$. The exact solution is shown in black and is given by $u = t^4 + t^3 + t^2$. The initial location is the point $(0, 0)$ and is shown by the first magenta triangle. At this point the slope k_1 is found. The slope is indicated by the blue line segment. A half step is taken along a line with slope k_1 to reach the yellow dot located at $(2.5, 0)$. The slope at this point, k_2, is shown by the yellow line segment. This slope is used to take a half step from $(0, 0)$ to the third location $(2.5, 215.63)$ shown by the green point. k_3 (shown by the green line segment) is then found. The fourth point (shown in purple) is found by stepping from the original point along a line with slope k_3. This step arrives at the point $(5, 431.25)$. The final slope k_4 is then calculated. These four slopes are then inserted into Eq. (3.10) to determine the next value of u. This step is illustrated by the magenta triangle at $t = 5$. The same procedure is then

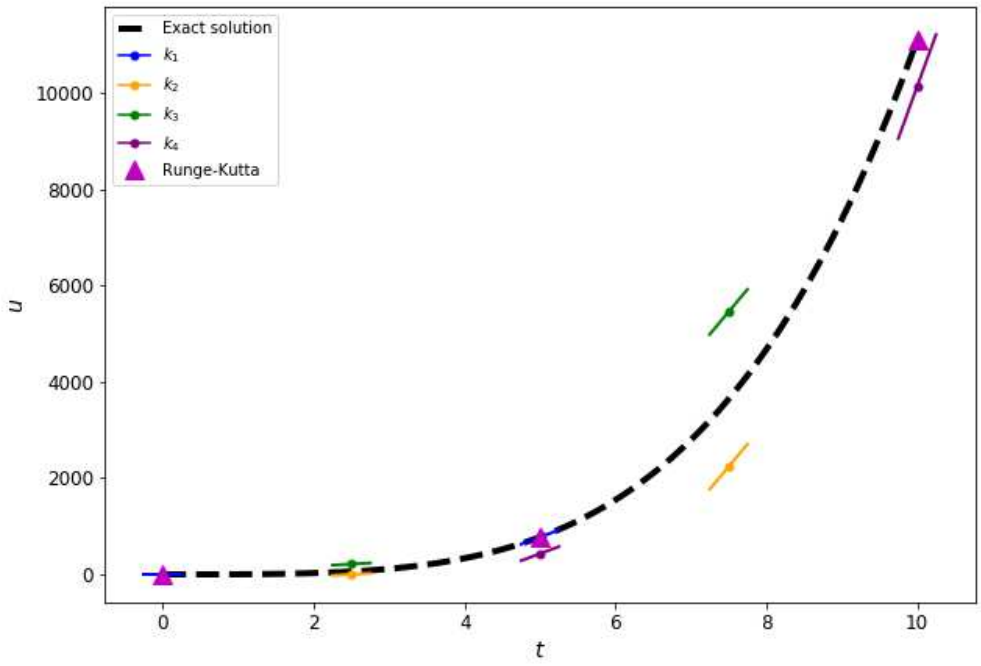

Figure 3.3: RK4 approximation of $u_t = 4t^3 + 3t^2 + 2t$ vs. t with a step of $h = 5$. The initial condition is $u(0) = 0$. The slope values k are shown by colored lines. The colored points indicate the location at which the k values are calculated.

conducted with the starting point at (5,775.0) to calculate new slope values and step to $t = 10$ (shown by the magenta triangle at the right of the image).

Consider again the ODE $u_t = \sin t$ with the initial condition, $u(0) = -1$. A comparison of Euler's method (blue), fourth-order Runge-Kutta (magenta), and the exact solution $u = \cos t$ (black) is shown in Fig. 3.4. The time step h is 0.25 for both integration methods. RK4 more closely approximates the curve u than Euler's method. RK methods allow for a higher level of precision and allow the curve u to be accurately modeled with fewer time steps than Euler's method.

RK methods can approximate the solution of ODEs. However, many equations are not ODEs. For example, Eq. (2.3), which is of direct interest for ion bombardment, is a PDE. Consider the equation

$$u_t = f(t, x, y, u, u_x, u_y), \tag{3.11}$$

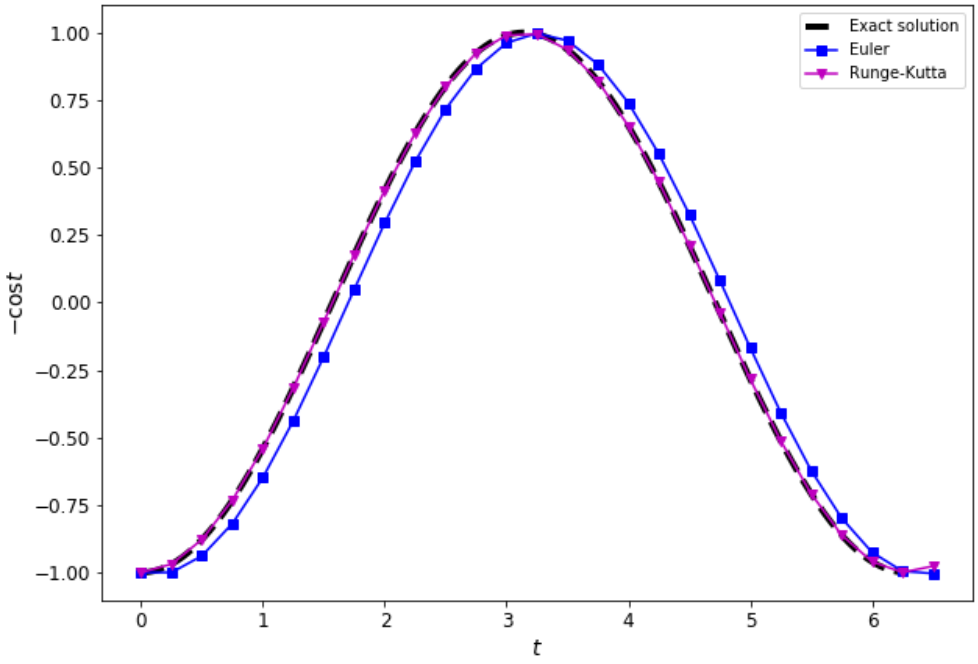

Figure 3.4: Numerical integration of $u_t = \sin t$ vs t with an inital condition of $u(0) = -1$. The exact solution, $u = -\cos(t)$, is shown in black, Euler's method is shown in blue, and the RK4 approximation is shown in magenta. The time step is $h = 0.25$. The domain size was 2π.

where $u = u(t, x, y)$ is a real value of some function at a particular time t and corresponding to a point on the $x - y$ plane. The time evolution of u depends not only on the time and value of u, but also the spatial location and derivatives. In this case, the previously discussed integration schemes cannot be directly applied. Instead the PDE must be transformed into a system of ODEs. The first step in this transformation is discretizing space. The $x - y$ plane is converted into a discrete set of points. Typically, this is done by setting distinct points separated by Δx in the x direction and Δy in the y direction [108]; however, adaptive meshes are also commonly used [109]. This converts u from a continuous function over the $x - y$ plane to a matrix in which each element corresponds to a particular point on the $x - y$ grid. Next, the spatial derivatives must be considered. These derivative terms can be approximated with a central difference finite difference scheme [110] which will be discussed in more detail in Chap. 4. At this point, an ODE can be written for each element of u corresponding to a particular location on the $x - y$ grid. This allows the function $u(t, x, y)$ to be described by a system of ODEs. Each ODE can be numerically integrated to approximate the time evolution of the function. Additional information about applying RK methods to PDEs can be found in Ref. [111].

3.2 Exponential Time Differencing

Both Euler's method and RK4 methods provide easily implemented means to approximate the solution to an ODE. However, they are not perfect. For example, if the solution has significant curvature, the linear approximation deviates from the exact solution quite rapidly. This divergence causes the numerical error to compound, which can lead to an even larger deviation.

When certain numerical methods like the Euler or RK4 methods are numerically unstable unless the step size h is extremely small, the differential equation is called stiff. A precise definition of "stiffness" is quite elusive, but the principle is that an equation tends to be stiff if it has at least one term that can lead to rapid variation in the solution [112].

Many of the PDEs used to model IBS are stiff. Therefore, using Euler's method or even the RK4 method is quite costly, requiring significant memory allotment within a computer and time to complete the required calculations. Instead, a different integration method was applied in most of our simulations. The method is the exponential time-differencing (ETD) scheme that was developed by Cox and Matthew [114] in 2002. This method works by breaking up the differential equation into a linear and nonlinear component. Once again, we start by considering an ODE, but the method can be applied to a PDE as well. An example of this can be found in section 4.4 of Cox and Matthew's work [114].

Consider an ODE with the form

$$u_t = Lu + N(u,t), \tag{3.12}$$

where u is the dependent variable, L is a constant, and $N(u,t)$ represents the nonlinear terms. Writing the equation in this form allows for the exact integration of the linear component, followed by an approximate integration of the nonlinear terms.

First, Eq. (3.12) is multiplied by an integration factor e^{-Lt}. At each step the time is advanced from t_n to $t_{n+1} = t_n + h$. This provides the exact formula

$$u(t_{n+1}) = e^{Lh}u(t_n) + e^{Lh}\int_0^h e^{-L\tau}N(u(t_n+\tau), t_n+\tau)d\tau. \tag{3.13}$$

For simplicity, the numerical approximation to $u(t_n)$ is denoted as u_n, the approximation of $u(t_{n+1})$ as u_{n+1} and $N(u_n, t_n)$ as N_n. Equation (3.13) is exact, but the integral of the nonlinear component must be approximated. As such, ETD methods primarily focus on approximating the integral of the nonlinear component. Exactly solving the linear portion of Eq. (3.13) directly greatly reduces the number of computational calculations that need to be executed. As such, exponential time differencing can be considerably more efficient than the Euler or RK4 method.

Assume $N(u,t)$ is constant on the interval from t_n to t_{n+1}, the integral simplifies and Eq. (3.13) reduces to

$$u_{n+1} = e^{Lh}u_n + N_n\left(\frac{e^{Lh}-1}{h}\right). \tag{3.14}$$

This formula is the so-called ETD1 scheme. The local truncation error is $\frac{h^2\dot{N}}{2}$ [114]. This error is of the same order as Euler's method, but since only the nonlinear components contribute to the error, the step size h can be larger than what is required by the Euler method. In fact, it is worth noting that for sufficiently small L values, Eq. (3.10) approaches Euler's Method [114].

ETD methods can be made more accurate through higher-order approximations. Instead of treating N_n as constant over the interval, consider a higher-order approximation

$$N \approx N_n + (N_n - N_{n-1})\left(\frac{\tau}{h}\right), \tag{3.15}$$

where N is an approximation of $N(u,t)$ on the interval $t_n \le t \le t_{n+1}$. Using this approximation yields the ETD2 integration scheme. In this case, Eq. (3.10) becomes

$$u_{n+1} = u_n e^{Lh} + N_n\left(\frac{(1+Lh)e^{Lh}-1-2Lh}{L^2h}\right) + N_{n-1}\left(\frac{-e^{Lh}+1+Lh}{L^2h}\right) \tag{3.16}$$

The ETD2 scheme has local truncation error of $\frac{5h^3\ddot{N}}{12}$ [114]. Likewise, the method can be generalized to any higher-order approximation. However, these higher-order approximations require additional points to evaluate $N(u,t)$ at. For example, Eq. (3.12) requires both N_n and N_{n-1}. Requiring additional points for evaluation makes these methods challenging to work with since often only the initial condition is known. This problem can be mitigated by incorporating RK time-stepping. RK methods also have smaller truncation error and larger stability regions than multistep methods [107].

The first exponential time-differencing with Runge-Kutta (ETDRK) stepping method that I want to address is the second-order form. This utilizes two points at which the nonlinear terms are evaluated. The first is at (u_n, t_n) and other is found at time $t_n + h$ and has the form

$$a_n = u_n e^{Lh} + N_n\left(\frac{e^{Lh}-1}{h}\right). \tag{3.17}$$

The nonlinear component on the interval $t_n \leq t \leq t_{n+1}$ is approximated by a combination of these two points. The equation to do this combination is

$$N \approx N_n + \left(\frac{\tau}{h}\right)(N(a_n, t_n + h) - N_n). \tag{3.18}$$

This approximation is used to advance $u_n \rightarrow u_{n+1}$. The equation to do this is

$$u_{n+1} = a_n + (N(a_n, t_n + h) - N_n)\left(\frac{e^{Lh} - 1 - Lh}{L^2 h}\right). \tag{3.19}$$

This method is called ETD2RK and has a local truncation error five times smaller than that of ETD2 [114].

The accuracy can be further improved by using more steps. The fourth-order ETD4RK scheme is given by:

$$a_n = u_n e^{Lh/2} + N_n\left(\frac{e^{Lh/2} - 1}{L}\right), \tag{3.20}$$

$$b_n = u_n e^{Lh/2} + N(a_n, t_n + \frac{h}{2})\left(\frac{e^{Lh/2} - 1}{L}\right), \tag{3.21}$$

$$c_n = a_n e^{Lh/2} + (2N(b_n, t_n + \frac{h}{2}) - N_n)\left(\frac{e^{Lh/2} - 1}{L}\right), \tag{3.22}$$

and

$$\begin{aligned} u_{n+1} = u_n e^{Lh} + \{N_n[-4 - Lh + e^{Lh}(4 - 3Lh + L^2h^2)] \\ +2(N(a_n, t_n + \frac{h}{2}) + N(b_n, t_n + \frac{h}{2}))[2 + Lh + e^{Lh}(-2 + Lh)] \\ +N(c_n, t_n + h)[-4 - 3Lh - L^2h^2 + e^{Lh}(4 - Lh)]\}/(L^3h^2). \end{aligned} \tag{3.23}$$

This scheme is significantly more accurate than the previously mentioned ETD schemes. The local truncation error of ETD4RK is of order h^5 [114]. ETD4RK is the most frequently used integration method in the work presented in this dissertation.

The application of ETD methods to PDEs is described in Ref. [114] but is more clearly explained in Ref. [116]. Often L is not a single constant scalar, but is instead a linear operator acting on u. In Eq. (2.3), for instance, the linear component consists of spatial derivatives,

$L = \kappa_1 \partial_x^2 + \kappa_2 \partial_y^2 - B\nabla^2\nabla^2$. To apply ETD methods, the spatial part of the PDE is discretized as previously discussed. This allows L to be written as a matrix [115]. The result is a system of ODEs that describe the value u at distinct locations on an $x-y$ grid. Then, just like with RK methods, the value of u at each point on the $x-y$ grid can be numerically integrated to update the entire surface.

3.3 Problems with Exponential Time-Differencing

ETD methods are used in every integration included in this dissertation. As such, it is important to address the limitations and potential pitfalls of these methods.

The most obvious limitation is that the differential equation of interest has to be separated into linear and nonlinear components. Not every equation can be. A possible solution to this is adding a "pseudo-linear" term to the EoM to act as L, then the "pseudo-linear" term is subtracted from the EoM in N. However, this eliminates the benefit of an exactly solvable linear portion. Preliminary tests have shown that this technique requires a comparable time step h to the traditional RK4 method. Simply put, ETD is not the optimal method for equations that do not have a linear component.

Another consideration can be directly observed in the traditionally accepted model of IBS, Eq. (2.3). The linear terms depend on the second-order and fourth-order spatial derivatives of u. For simplicity, consider the 1D special case in which all parameters have a magnitude of 1. L can be written as

$$L = -\partial_x^2 - \partial_x^4 \tag{3.24}$$

and the nonlinear operator is

$$N(u,t) = u_x^2. \tag{3.25}$$

L is a differential operator. To simplify the calculation, the Fourier transform of the surface $\tilde{u}(k_x, k_y)$ is used. The transformed version of Eq. (3.12) is

$$\tilde{u}_t = \tilde{L}\tilde{u} + \tilde{N}(\tilde{u}, t). \tag{3.26}$$

This transformation then allows the derivatives to be determined by simple multiplication [107]. The nth spatial derivative of u with respect to x can be determined from the equation

$$\widetilde{u^{(n)}}(k_x, k_y) = (ik_x)^n \tilde{u}(k_x, k_y), \tag{3.27}$$

where $\tilde{u}(k_x, k_y)$ is the Fourier Transform counterpart of $u(x, y)$. Likewise, if the derivative was with respect to y, k_y would be used instead. In the case of the 1D KS equation, L takes the form

$$\tilde{L} = -(ik_x)^2 - (ik_x)^4. \tag{3.28}$$

Likewise, $\tilde{N}(\tilde{u}, t)$ is the Fourier Transform of the real space nonlinear portion of the 1D KS equation. We apply our ETD scheme to approximate the solution of $\tilde{u}$. Once the integration has been conducted, $\tilde{u}$ is converted back to real space for analysis. In IBS, the linear operator is often proportional to spatial derivatives. This allows L to be written as a diagonal matrix in Fourier space and to be handled computationally as an array of scalars.

The final consideration that emerges is in the e^{Lh} term that is present in every ETD scheme. When eigenvalues of the matrix L are very small, the ETD methods suffer from a numerical instability. Kassam and Trefethen produced a solution to this instability through the use of contour integration [116].

Consider the expression

$$g(z) = \frac{e^z - 1}{z}, \tag{3.29}$$

where z is much smaller than 1 and is an eigenvalue of L. While a Taylor expansion of $g(z)$ will yield a finite result, a computer will not always report one due to floating-point precision. If z is below a minimum threshold, the computer will return 0 as the value of $g(z)$ instead of the actual value. This introduces substantial inaccuracies and can cause numerical overflow/underflow. There are clear solutions to the extreme cases of very large or very small eigenvalues. When all

the eigenvalues of L are large, no numerical instability will occur, and ETD, as stated by Cox and Matthew, can be used. If all of the eigenvalues are small, a Taylor series of e^z can be produced and truncated at the appropriate order to get the desired precision. However, real systems rarely fall into either of these categories. Most L matrices for IBS have some large eigenvalues and some very small eigenvalues. In IBS models, the eigenvalues correspond to growth rates of different Fourier modes, i.e., some wavelengths will grow faster than others.

The solution to this problem is achieved via contour integration. Kassam and Trefethen used the expression

$$f(L) = \frac{1}{2\pi i} \int_\Gamma f(t)(tI - L)^{-1} dt, \tag{3.30}$$

where Γ is a contour within the complex plane that encloses the eigenvalues of L and I is the identify matrix. It is important to note that L is a matrix, so if L is an operator, we have transformed to an appropriate basis to represent the operator as a matrix. In the case of ETD, $f(L) = e^{Lh}$. This allows the values of the matrices e^{Lh} to be calculated and applied to the schemes produced by Cox and Matthew. Additional information on contour integration in general can be found in [107].

Chapter 4

Python Tool for Numerical Integration

4.1 Introduction

In the previous chapter, the mathematical methods used for numerical integration were presented. However, the computational implementation of these methods was not. When I first arrived at CSU and joined the Bradley Theory Group, more experienced members, Dr. M. Harrison and Dr. D. Pearson, shared the codes that they used for numerical integration and directed me to papers about the methods in general. The code as it existed was for personal use and therefore lacked documentation; the only person who needed to understand it was the person using it, so why spend time documenting it? Also, parts of the code were developed by different people at different times. Some of the programs were in Matlab, others in Python 2, and still others in Python 3. There was also variation in one code to the next in terms of libraries used and implementation.

None of this was unreasonable or unwarranted. Research is complicated, as is software development. Researchers will develop in whatever language they are most familiar with for the problem at hand. The goal of the project also drifts as new discoveries are made. Even programming languages themselves evolve along with analysis libraries. Furthermore, the emphasis is on the research, not documenting a program. These factors, along with many others, all contribute to a software engineering concept called "technical debt". This is quite simply the "cost" that is associated with inefficiencies in code. Much like financial debt, this compounds over time and creates difficulties with implementing the code, such as the interpretability for new researchers in the group or increased computation time.

To aid in my understanding of the methods and to help new members of the research group, I developed a generalized ETD framework in Python 3. This software tool allows the user to define an EoM by giving the linear and nonlinear components, L and $N(u,t)$. The user also defines

a domain size, spatial grid spacing to discretize the $x-y$ plane, final time to stop the simulation, time step, and initial condition (IC). The tool then takes this information and calculates the time integration of the surface. The tool can also produce images and binary files for direct analysis and contains a collection of functions to assist in the analysis process. This tool goes beyond being an example of numerical integration: it is a full framework because it allows the user to define the linear and nonlinear functions and then uses those to generate a simulation.

As mentioned, the tool itself is a Python 3 module. Python 3.8 is the current version at the time of writing this document. Python was selected because it is easily interpretable by users, has numerous libraries to assist in analysis, has a large community to help with support and development of new code, and has good speed and productivity. All of the code for the framework is saved in a file named BradleyTheoryGroup.py. It can be included as any other Python module would be and applied to analysis code. In addition to the base module, multiple examples and README documents are included in addition to internal help files for every function in the module.

While this work is not directly advancing the field of ion bombardment, it does provide a direct benefit to the Bradley Theory research group by increasing the speed with which simulations can be produced, virtually eliminating the startup time of new members, and providing a well documented base to expand upon for future analysis and development. Note that Dan Pearson helped optimize the code and expand the tool to include the numerical integration of the EoMs for ion bombardment of a binary material.

The necessary steps to execute a simulation using this framework are discussed in this chapter. Some of the behind-the-scenes work conducted by the module to facilitate the numerical integration are then addressed. Lastly, some of the functions built into the software tool to facilitate in the analysis of the simulations are mentioned.

4.2 User Inputs

To facilitate the production of simulations, an interactive Python notebook using the Jupyter platform has been created. The framework is loaded like any other Python library: 'import BradleyTheoryGroup' is simply added to the top of the Python program. An example notebook can be found in Appendix A. However, it is worth mentioning the steps that the user needs to take to use the program.

First, the user needs to define L. This is done in Fourier space to facilitate the computational representation of the spatial derivative operators. The linear component for the aKS equation is

$$L = \kappa_1 (ik_x)^2 + \kappa_2 (ik_y)^2 - B((ik_x)^4 + (ik_y)^4 + 2(ik_x)^2 (ik_y)^2), \tag{4.1}$$

where k_x is the wave number in the x direction and k_y is the wave number in the y direction. The parameters κ_1, κ_2, and B are the same coefficients that appear in Eq. (2.3). Fortunately, the L operator is typically diagonal in the Fourier basis for IBS, and so the extension of ETD to a system of ODEs is the same as outlined in Ref. [114].

In the next step, the user defines the nonlinear function N. This is done in real space to reduce the calculations needed in Fourier space. Spatial derivatives within this function can be calculated however the user desires, but the framework has built in finite differencing. The nonlinear component, written in real space, for the aKS equation is

$$N = \lambda_1 u_x^2 + \lambda_2 u_y^2, \tag{4.2}$$

which is directly from Eq. (2.3).

Both L and N are defined as Python functions to feed to the framework and then converted into a usable form to execute ETD. If the user is not familiar with Python coding, both L and N_{real} can be written as strings which the framework will automatically convert into a Python function and apply throughout. However, this requires more computation than writing the function in Python.

The user specifies all of the linear and nonlinear parameters. For the aKS equation, the linear parameters are κ_1, κ_2, and B. The nonlinear parameters are λ_1 and λ_2. The linear and nonlinear parameters are then stored in separate lists.

The user then specifies the number of grid points along the x and y directions and the range for each direction. If the user wishes to study a PDE in one space dimension rather than two, the number of grid points along the y direction is set to 1 and the framework will automatically handle the rest. The framework then generates an evenly spaced grid of points with the designated number of points in the mesh over the user defined ranges. Next, the user defines the starting value of t, the final value of t to stop the simulation at, and the small time step h. The final step before sending the input to the module is specifying an IC. This is typically low amplitude spatial white noise for a nominally flat surface, but could be a sinusoid or anything that the user specifies.

The user calls the "timeIntegrator" function with the user defined functions and conditions as arguments to be passed into the framework in the final step. Additional, optional arguments of the function can tell it to create and save files to a specific directory to make data management easier, to generate and save images of the surface and, if desired, the Fourier transform of the surface for the user to view, to generate numpy data files for further analysis and specify the rate at which this is done. The function also allows the user to chose between ETD schemes. ETD2RK and ETD4RK are the recommended integration schemes. Other schemes are in the process of being incorporated, but have yet to be documented and case tested. ETD2RK takes less time to advance the surface through a time step, but requires a smaller time step h. When the surface consists of many grid points, it can be more computationally efficient to do easier calculations more times than to use the more complicated ETD4RK.

This process allows the user to define a PDE and start a simulation. Once the simulation has begun, the module prints a progress bar to allow the user to monitor its progress. An example notebook which uses this framework to numerically integrate Eq. (2.3) is included in Appendix

A. This example shows the explicit Python commands needed to produce a simulation from a white noise initial condition.

4.3 Behind the Scenes

After the user has defined the EoM and domain, the module handles all of the complexities of implementing ETD methods for that EoM. The most significant considerations are converting to and from Fourier space repeatedly, calculating the contour integral to get the elements of the matrix e^{Lh}, approximating the spatial derivatives, and reducing compounding error in the imaginary components that arise from an approximate Fourier transform. In this section, I will present how the framework implements each of these.

The ETD method is conducted in Fourier space, but the nonlinear contribution is calculated in real space. The surface u has to be converted to and from Fourier space repeatedly. This is done using a fast Fourier Transform (FFT). Unfortunately, this can be computationally taxing when done so many times. Furthermore, there are numerous methods for calculating the FFT which are optimized for different cases. Originally scipy's fftpack was used, but D. Pearson recommended and helped to implement the speedy FFT library. This library has a Pythonic wrapper called pyFFTW, which allowed it to be directly implemented. The speedy FFT library takes the initial surface and tests different methods of computing the FFT to find the fastest for the surface. Then this method is used for every FFT and inverse FFT conducted. The module will also choose a 1D or 2D FFT scheme based on the input provided by the user.

The contour integral previously mentioned in Sec. 3.2 must be calculated. Unfortunately, this has to be done numerically. The model does this by approximating the circular contour as a hexadecagon by default. The number of sides of the polygon can be changed by the user, but sixteen is used by default because it can be calculated quickly and produces results with minimal error.

The spatial derivatives of u are approximated in real space by a central difference finite differencing scheme accurate to fourth order in the grid spacing. For example, the partial deriva-

tive of u with respect to x was approximated by

$$\frac{\partial u}{\partial x}(x,y) \approx \frac{u(x-2\Delta x,y)-8u(x-\Delta x,y)+8u(x+\Delta x,y)-u(x+2\Delta x,y)}{12\Delta x}, \tag{4.3}$$

where Δx is the spatial separation of grid points in the x direction [110, 113]. This produces errors of order $\mathcal{O}(\Delta x^4)$. The module also will compute spatial derivatives up to fourth order. The user can specify whether the derivative is in the x direction or the y direction and the module handles implementing the finite difference along either axis.

The surface u is converted from real space to Fourier space and back many times by approximations of the Fourier transform and inverse Fourier transform. This introduces error. The individual error of a transformation is very small, but can compound through many iterations. This presents itself as a growing imaginary component; when the surface u is transformed to Fourier space and then transformed back to real space, it now has a small imaginary component. Each time the error causes this to grow and eventually it produces numerical overflow and the simulation crashes.

The solution to this problem is simply retaining only the real portion of u after each ETD step of h. The surface u in real space is transformed to Fourier space and the ETD step for a time interval h is calculated. The updated surface in Fourier space is then transformed back to real space. The imaginary component is then dropped. If the simulation time is not over, the process is repeated, advancing the time by h. Otherwise, the simulation ends.

4.4 Analysis Suite

In addition to the ETD integrator, the module includes various functions to aid in the analysis of the simulations. Each function has a full help file and documentation to help the user. For example, if the user wants to generate mp4 movie files to watch the simulation evolve in time, the user can call the function "MakeMovie" and it will automatically generate it. Likewise, if images of the FFT of the surface is desired, the user can turn on a "DoFFTPlots" flag in the

integration function. There are built in functions to calculate surface roughness, peak-to-peak and trough-to-trough distances to get the wavelength distributions, make 1D cuts along the x or y direction for 2D surfaces, and to plot the derivatives.

The surface information is also stored as numpy arrays and can be loaded into a Python environment. Once the data is loaded, any Python library can be loaded and used to analyze the surface. This allows the user to use the various analysis tools that others have written and greatly expands the analysis tools available.

4.5 Examples of Simulations produced with this Tool

This module provides a fast and well documented tool to conduct simulations of PDEs, particularly those that occur in IBS. This is the tool that was used to produce every simulation in the following work and will hopefully prove useful for future studies by other group members.

Figure 4.1 presents three example simulations of Eq. (2.3). All three simulations had $B = 1$ and $\lambda_1 = \lambda_2 = -1$. The top left image shows the bowl patterns that emerge for the isotropic KS equation with $\kappa_1 = \kappa_2 = 1$. The top right image shows parallel mode ripples with $\kappa_1 = -1$ and $\kappa_2 = 1$. The bottom image shows perpendicular mode ripples. The parameters are $\kappa_1 = 1$ and $\kappa_2 = -1$.

Figure 4.2 shows a simulation of the 1D KS equation. The parameters are $\kappa_1 = -1$, $B = 1$, and $\lambda_1 = 1$. κ_2 and λ_2 correspond to spatial derivatives along the y direction and therefore are irrelevant in this special case. The module will automatically produce u vs x plots if the user input is 1D and contour plots otherwise.

2D simulations of the aKS equation with parameters $\kappa_1 = -1$, $\kappa_2 = 1$, $B = 1$, and $\lambda_1 = \lambda_2 = -1$ with a range of $[0, 400]$ for both x and y with a spatial separation of 0.5 and a time step of $h = 0.25$. The simulation started from $t = 0$ and ended at $t = 400$. Using ETD4RK, it took 116.70 seconds for the simulation to finish while producing both images of the surface and numpy files after every interval of $\Delta t = 5$. The same simulation was run for the 1D case, but now with a grid of 1×200 instead of 200×200. This simulation took only 9.02 seconds. The machine used to

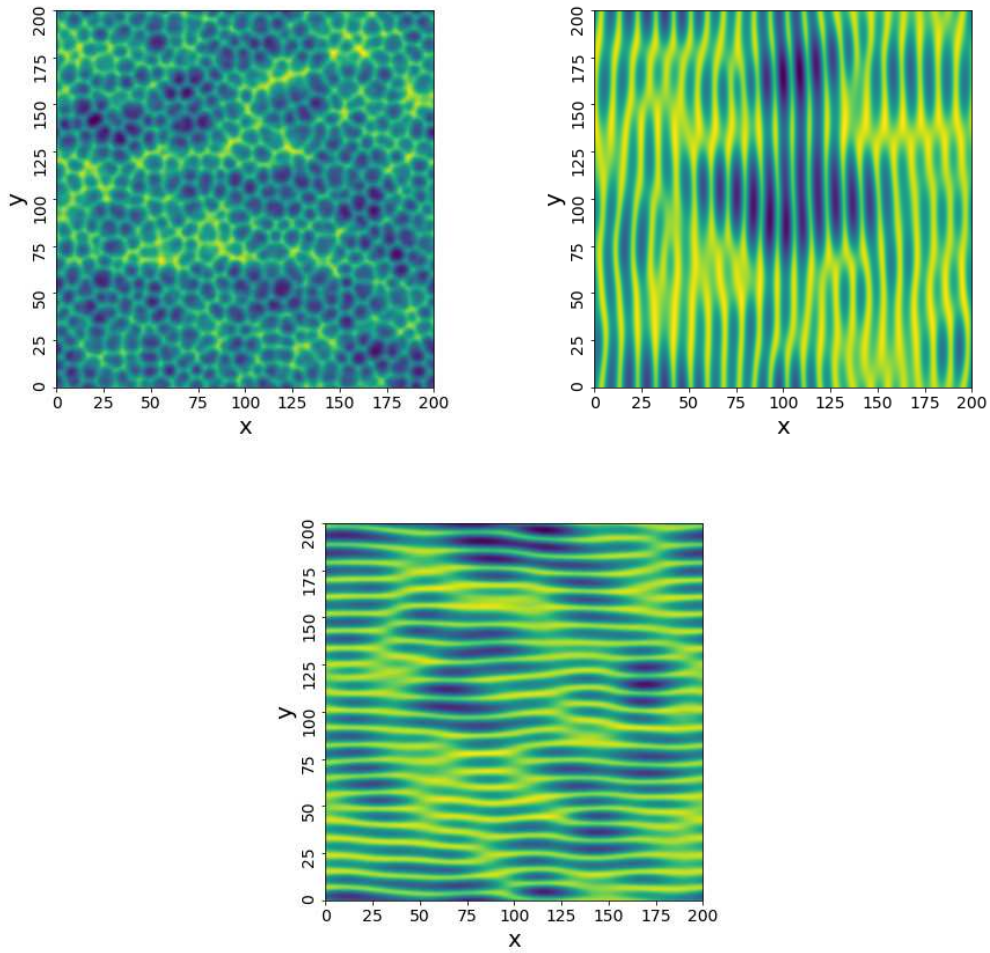

Figure 4.1: Contour plots of u at $t = 50$ for $\kappa_1 = \kappa_2 = -1$ (top left), $\kappa_1 = -1$ and $\kappa_2 = 1$ (top right), and $\kappa_1 = 1$ and $\kappa_2 = -1$ (bottom). $B = 1$ and $\lambda_1 = \lambda_2 = -1$ for all three. The domain size was 200x200.

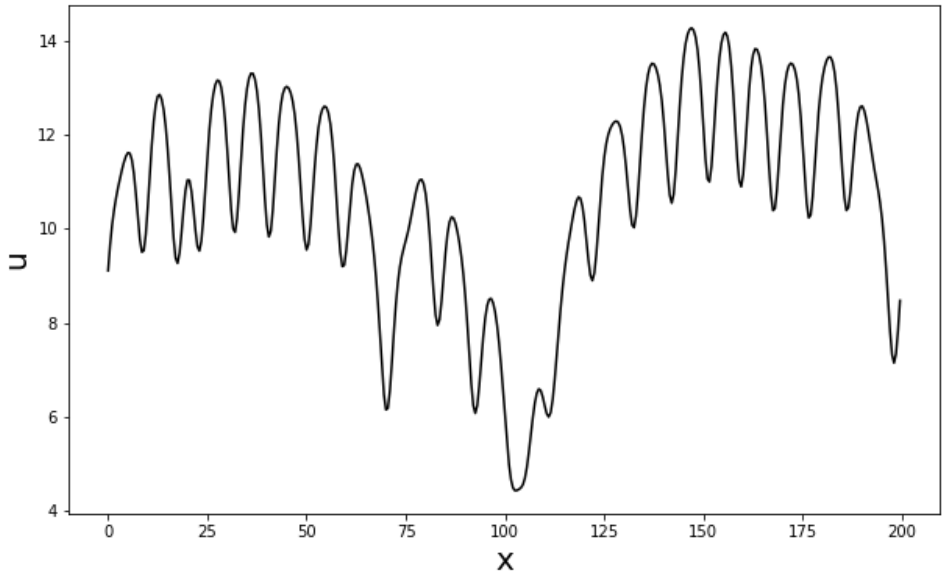

Figure 4.2: Plot of u vs x at $t = 50$ for $\kappa_1 = -1$, $B = 1$, and $\lambda_1 = 1$. The domain size was 200.

benchmark the performance was the Bradley Theory Group machine which has 78GB of RAM and a ASUS X299-A motherboard.

Chapter 5

The Effect of Dispersion on the Nanoscale Patterns Produced by Ion Sputtering

5.1 Introduction

In the original BH theory, the Sigmund model of ion sputtering was reduced to a partial differential equation for a nearly flat surface with slowly varying surface height [16]. The dependence of the sputter yield on the surface curvature (i.e., on second derivatives of the surface height u was included in the theory, but its dependence on third and higher order spatial derivatives of u was omitted. Makeev, Cuerno and Barabási extended the BH analysis to include terms proportional to third order spatial derivatives of u as well as other, higher order terms which we shall omit [18]. The resulting equation of motion is

$$u_t = C_{11}u_{xx} + C_{22}u_{yy} - B\nabla^2\nabla^2 u + \lambda_1 u_x^2 + \lambda_2 u_y^2 + C_{111}u_{xxx} + C_{122}u_{xyy}. \tag{5.1}$$

For a surface with slowly varying height, u_{xxx} and u_{xyy} are in general larger than $\nabla^2\nabla^2 u$, and so it is actually inconsistent to omit the third-order terms.

The simplest and most obvious effect of including terms proportional to u_{xxx} and u_{xyy} in the equation of motion (EOM) is that it makes the ripple propagation dispersive [18,183]. There is direct experimental evidence that establishes that the effect of dispersion can be substantial [12]. The dispersive terms in Eq. (5.1) can result from ion-induced plastic flow [117,118] or viscous relaxation of ion-induced stresses [31,119] as well as from sputtering [18,20].

In this chapter, we will explore the effects of dispersion on the patterns produced by oblique-incidence ion sputtering. We find that dispersion can lead to the formation of raised and depressed triangular regions that are traversed by parallel-mode ripples. These regions — which we will often refer to as “triangles" for the sake of brevity — strongly resemble structures that are

commonly observed in experiments. In a different range of the model's parameters, highly ordered ripples emerge. This suggests that dispersion could be employed to produce much more orderly ripples than those that are usually found in experiments. Finally, protrusions and depressions that are elongated in the longitudinal direction can emerge at sufficiently long times. These topographies resemble the so-called perpendicular-mode ripples that are frequently observed for high angles of ion incidence.

This chapter is organized as follows. We study the behavior of the surface for the special case in which its height does not depend on the transverse coordinate y in Sec. 5.2. The general case in which u depends on x, y and t is considered in Sec. 5.3. We put our results in context in Sec. 5.4 and discuss them further in Sec. 5.5. Our conclusions are presented in Sec. 5.6.

5.2 One Dimensional Results

Consider a 1D disturbance (i.e., one with $u_y = 0$ everywhere) and suppose that $C_{11} < 0$ so that there is an instability. We introduce the dimensionless quantities $\tilde{x} = \mathrm{sgn}(C_{111})(|C_{11}|/B)^{1/2}x$, $\tilde{t} = (C_{11}^2/B)t$ and $\tilde{u} = (\lambda_1/|C_{11}|)u$, where $\mathrm{sgn}(C_{111})$ denotes the sign of C_{111}. After dropping the tildes, we obtain the rescaled equation of motion

$$u_t = -u_{xx} - u_{xxxx} + u_x^2 + \alpha u_{xxx}, \tag{5.2}$$

where $\alpha \equiv (B|C_{11}|)^{-1/2}|C_{111}|$ is a nonnegative, dimensionless measure of the strength of the dispersion. In the new, rescaled coordinate system, the projection of the ion beam onto the $x-y$ plane points in the $-x$ direction if $C_{111} > 0$, while if $C_{111} < 0$, it points in the $+x$ direction.

For $\alpha = 0$, Eq. (5.2) reduces to the one-dimensional KS equation. In this limit, solutions to Eq. (5.2) with low-amplitude spatial white noise initial conditions exhibit spatio-temporal chaos.

We differentiate Eq. (5.2) with respect to x and set $\bar{x} = -x$, $\bar{t} = \alpha t$ and $v = (2/\alpha)u_x$. After dropping the bars, we have

$$v_t + \alpha^{-1}(v_{xx} + v_{xxxx}) + vv_x + v_{xxx} = 0. \tag{5.3}$$

Equation (5.3) is known as the Kawahara equation [120] and has previously been studied as a model of step-bunching dynamics on vicinal surfaces [121, 122]. It reduces to the Korteweg-Vries (KdV) equation

$$v_t + vv_x + v_{xxx} = 0, \tag{5.4}$$

a paradigmatic equation in the study of solitons, in the strongly dispersive limit $\alpha \to \infty$. When α is large and finite and the initial condition is low-amplitude spatial white noise, the solution to Eq. (5.3) tends to a highly ordered steady state that consists of a chain of equally-spaced solitons of the same amplitude [120].

We numerically integrated Eq. (5.2) using the ETD4RK method originally presented by Cox and Matthews [114] with periodic boundary conditions. The linear terms of the equation were computed exactly in Fourier space; on the other hand, the nonlinear terms were approximated in real space using finite differencing. A central difference scheme accurate to fourth order in the grid spacing along the x direction, Δx, was used for the derivatives. In particular, the partial derivative of u with respect to x was approximated by Eq. (4.3). The integration method is described in Chap. 3 and the computational tool used was the framework presented in Chap. 4.

Figure 5.1 shows the results of simulations for a selection of α values at time $t = 300$. In each case, the initial condition was low amplitude spatial white noise. (The amplitude of the white noise was chosen to be 0.01 for all simulations discussed in this chapter.) For $\alpha = 0$, Eq. (5.2) is the 1D KS equation and the solution exhibits spatio-temporal chaos. The dispersive term seems not to fundamentally alter the behavior for $\alpha = 0.1$. However, for $\alpha = 0.5$, the ripples are much closer to being periodic. For $\alpha = 1$, sections of the surface have well-ordered ripples superimposed on an overall upward slope. These sections are followed by precipitous downward drops. The order increases still further for larger values of α until, for $\alpha = 25$, the ripples are almost perfectly ordered but are modulated by a low amplitude, long wavelength height variation. For still larger values of α, the modulation is of smaller amplitude, and the highly ordered steady

state can be thought of as a chain of equally-spaced solitons of the same amplitude, as noted above [120].

To quantify the increase in order with increasing α, we generated a new set of simulations. Simulations were carried out for a range of values of α between 0.1 and 40 on a domain of width 10000, and for each value of α, ten independent simulations were generated with low amplitude spatial white noise initial conditions. We then took the absolute value of the Fourier transform of each of the surfaces corresponding to a single α value at time $t = 800$ and averaged. We next fit a Gaussian to the first peak of the averaged curve and calculated the full width at half max (FWHM) of the Gaussian. As usual, the smaller the FWHM, the more orderly the surface is. Figure 5.2 (a) indeed shows that as α increases, the FWHM of the fitted Gaussian decreases.

The location of the Gaussian's peak yields the wavelength of the ripples at time $t = 800$, a time when a steady state has very nearly been reached. As shown in Fig. 5.2 (b), the ripple wavelength Λ depends on α at $t = 800$. In contrast, the wavelength at early times takes on the linearly-selected value $2\sqrt{2}\pi$, which is independent of α. This means that the ripple wavelength must in general depend on time. This time dependence is illustrated in Fig. 5.3 for $\alpha = 2$, 10, 25, and 50. Importantly, the wavelength at long times exceeds the linearly selected value for all values of α we examined, which means that the dispersive term produces ripple coarsening. This coarsening is interrupted, i.e., it does not continue indefinitely. Our simulations suggest that the long-time value of Λ tends to the linearly-selected value $2\sqrt{2}\pi$ in the $\alpha \to \infty$ limit.

An interesting feature that is not apparent in Fig. 5.1 is the asymmetry of the ripple pattern for nonzero α. Figure 5.4 shows a portion of the surface for $\alpha = 1$ at time $t = 300$. The right and left sides of the ripple crest shown in the figure are clearly different. This is to be expected since Eq. (5.2) is not invariant under the transformation $x \to -x$ for $\alpha > 0$.

It is natural to ask whether the effects of dispersion could be observed in an experiment. For a typical material, $C_{11} = C_{11}(\theta)$ changes sign from positive to negative as θ is increased from zero through a critical angle θ_c. Ripples form for θ just above θ_c and the dimensionless measure of the strength of the dispersion $\alpha = (B|C_{11}|)^{-1/2}|C_{111}|$ is large in this regime. When C_{11} is small

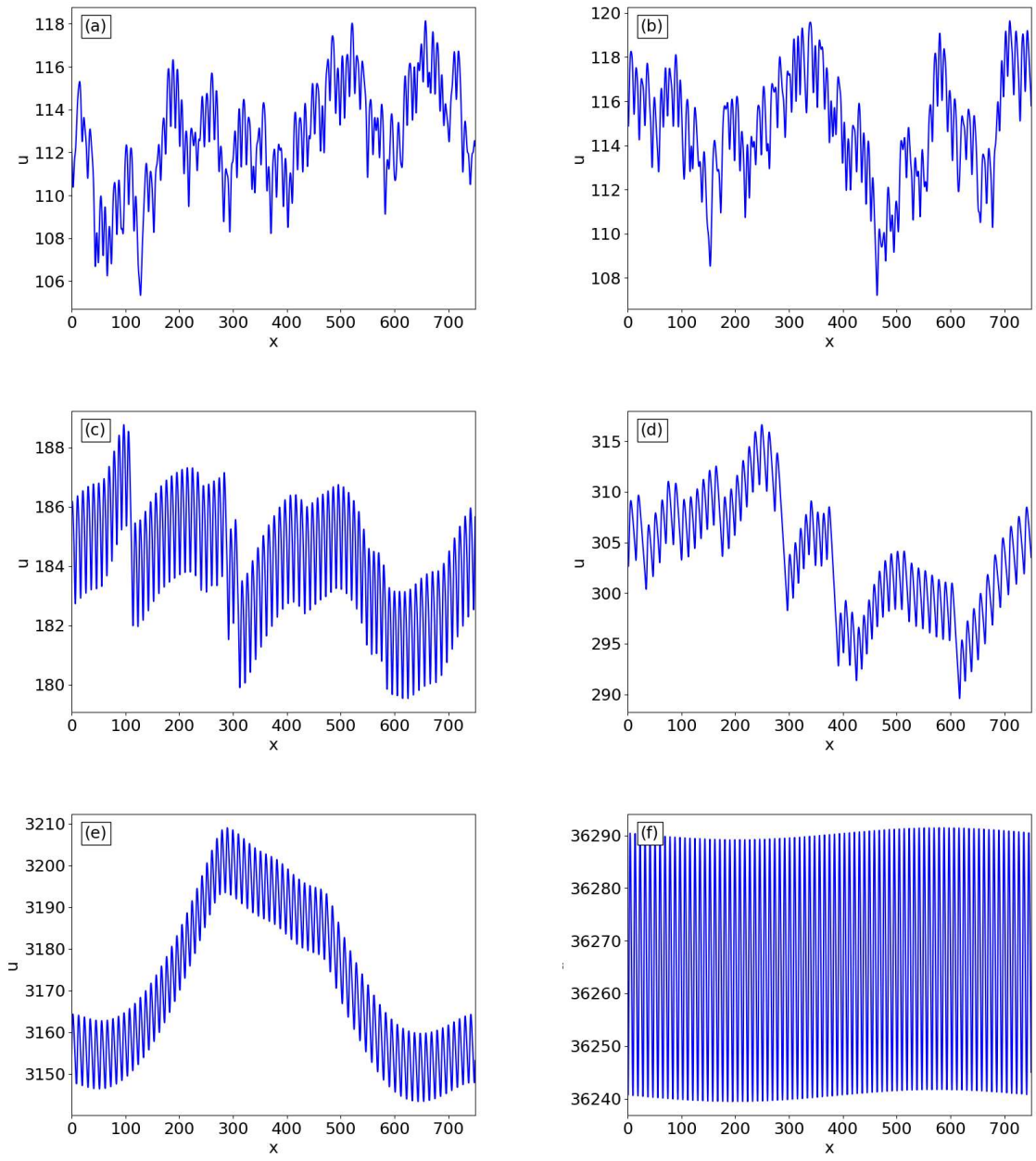

Figure 5.1: u versus x at time $t = 300$ for (a) $\alpha = 0$, (b) $\alpha = 0.1$, (c) $\alpha = 0.5$, (d) $\alpha = 1$, (e) $\alpha = 5$ and (f) $\alpha = 25$. The domain width was 750.

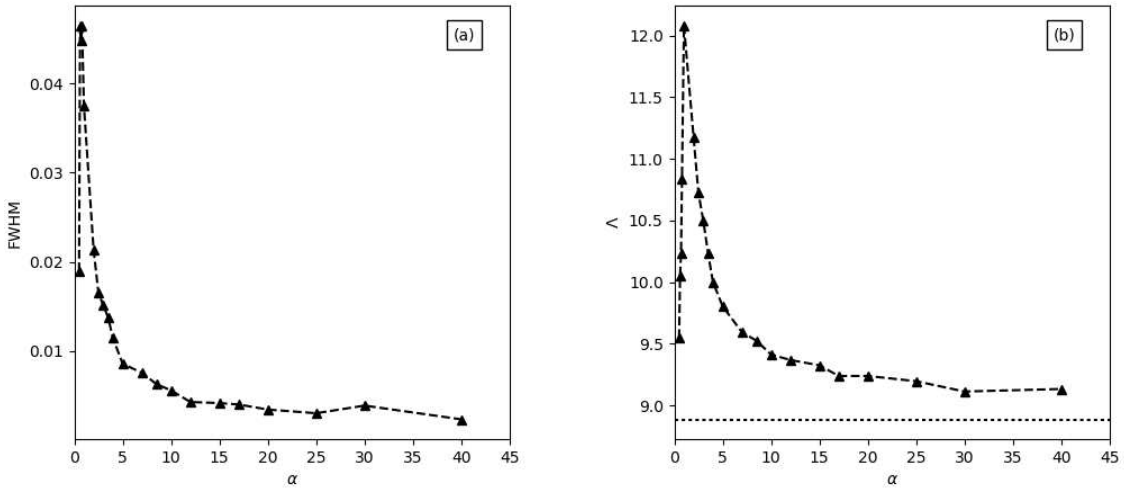

Figure 5.2: (a) The full width at half max (FWHM) and (b) wavelength Λ versus α at time $t = 800$. The dashed black lines are guides to the eye, the triangles are the actual points calculated, and the dotted line in (b) displays the linearly selected wavelength $2\sqrt{2}\pi$. The domain width was 10000.

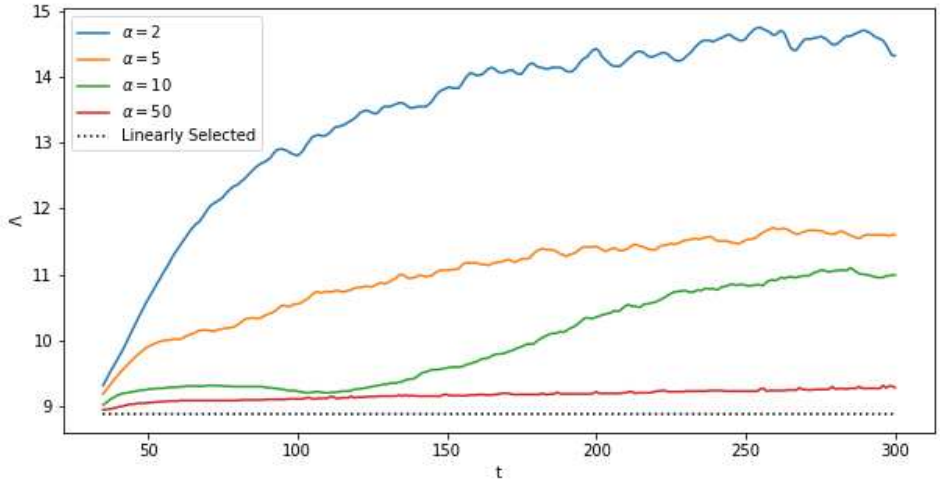

Figure 5.3: Wavelength versus time for α = 2, 5, 10, and 50. The dotted black line shows the linearly-selected wavelength. The domain width was 10000.

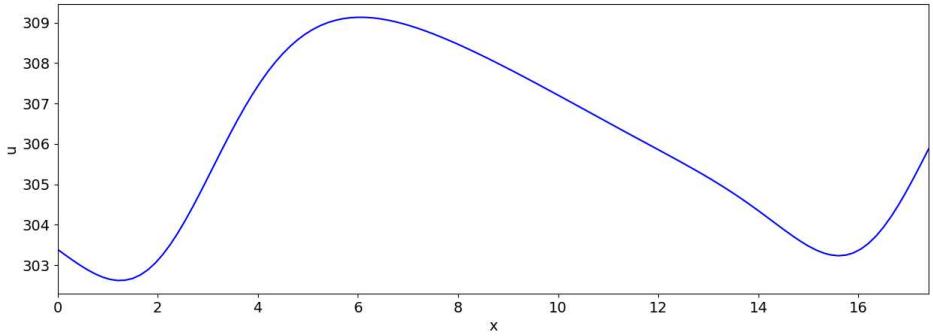

Figure 5.4: u versus x at time $t = 300$ for $\alpha = 1$.

and negative, however, the characteristic time B/C_{11}^2 that it takes for the ripple amplitude to become appreciable is large. This means that an unattainably high fluence might be needed for ripples to form and for the effects of dispersion to become noticeable. Fortunately, α need not be large for the effects of dispersion to be substantial. If α is of order 1, our simulations show that the behavior of the surface is strongly affected by dispersion when times are reached that are several hundred times the characteristic time B/C_{11}^2 [see Fig. 5.1 (d)]. Times this long are commonly reached in experiments.

5.3 Pattern Formation in Two Dimensions

As we have seen, the dispersive term has a significant effect on the surface produced by Eq. (5.1) for the special case in which u is independent of the transverse coordinate y. However, there are variations in height in both the longitudinal and transverse directions in experiments, and so we must consider the behavior predicted by the full equation of motion (5.1) when u depends on x, y, and t. This will henceforth be called the two-dimensional (2D) case.

We begin by defining $\tilde{x}$, $\tilde{t}$ and $\tilde{u}$ as in Sec. 5.2 and by setting $\tilde{y} = (|C_{11}|/B)^{1/2} y$. We once again assume that C_{11} is negative so that an instability is present in the longitudinal direction. After dropping the tildes, Eq. (5.1) becomes

$$u_t = -u_{xx} + r_1 u_{yy} - \nabla^2 \nabla^2 u + u_x^2 + r_2 u_y^2 + \alpha u_{xxx} + \beta u_{xyy}, \tag{5.5}$$

where $r_1 \equiv C_{22}/|C_{11}|$, $r_2 \equiv \lambda_2/\lambda_1$ and $\beta \equiv (B|C_{11}|)^{-1/2}C_{122}$ are dimensionless parameters and we remind the reader that $\alpha \equiv (B|C_{11}|)^{-1/2}C_{111}$.

5.3.1 Highly Ordered Ripples

A sensible starting point for the study of the 2D problem is a case that is roughly analogous to the 1D case: we assume that r_1 is positive so that there is no linear instability in the transverse direction. In fact, we will go further and consider the case $r_1 = 10$ in this subsection so that variations in the y direction are rather strongly suppressed.

Just as was done for the 1D special case, a set of simulations were conducted starting from a low-amplitude spatial white noise initial condition. The numerical integrations were once again performed using the ETD4RK method. Several different values of α were considered. The values of the remaining parameters were $r_2 = 1$, $\beta = 0$ and, as already mentioned, $r_1 = 10$.

Figure 5.5 shows that increasing α promotes the formation of highly ordered ripples, just as in the 1D case. In the absence of dispersion, there is no clear order in the x direction, but as the coefficient of the dispersive term u_{xxx} is increased, order in the x direction starts to emerge and then becomes very strong for large dispersion, e.g., for $\alpha = 50$.

Figure 5.6 shows that a one-dimensional cross section of the surface with $\alpha = 50$ is qualitatively similar to the result of the 1D simulation with $\alpha = 25$ that is displayed in Fig. 5.1 (f). Both simulations produce highly ordered ripples with a minor long-wavelength modulation in the x-direction.

Although the surface is highly ordered in the x direction in Fig. 5.5 (d), it is noticeably depressed for y in the vicinity of 200. To see how this kind of feature emerges and what its ultimate fate is, we simulated the time evolution of the surface for the same parameters as in Fig. 5.5 (d), i.e., $r_1 = 10$, $r_2 = 1$, $\alpha = 50$ and $\beta = 0$, but ran the simulation for a longer period of time. The results are shown in Fig. 5.7. At early times, the time evolution is well described by the linearized equation of motion and parallel-mode ripples form. As the surface continues to evolve, the nonlinear terms start to have a significant effect. By $t = 80$, multiple band-shaped regions

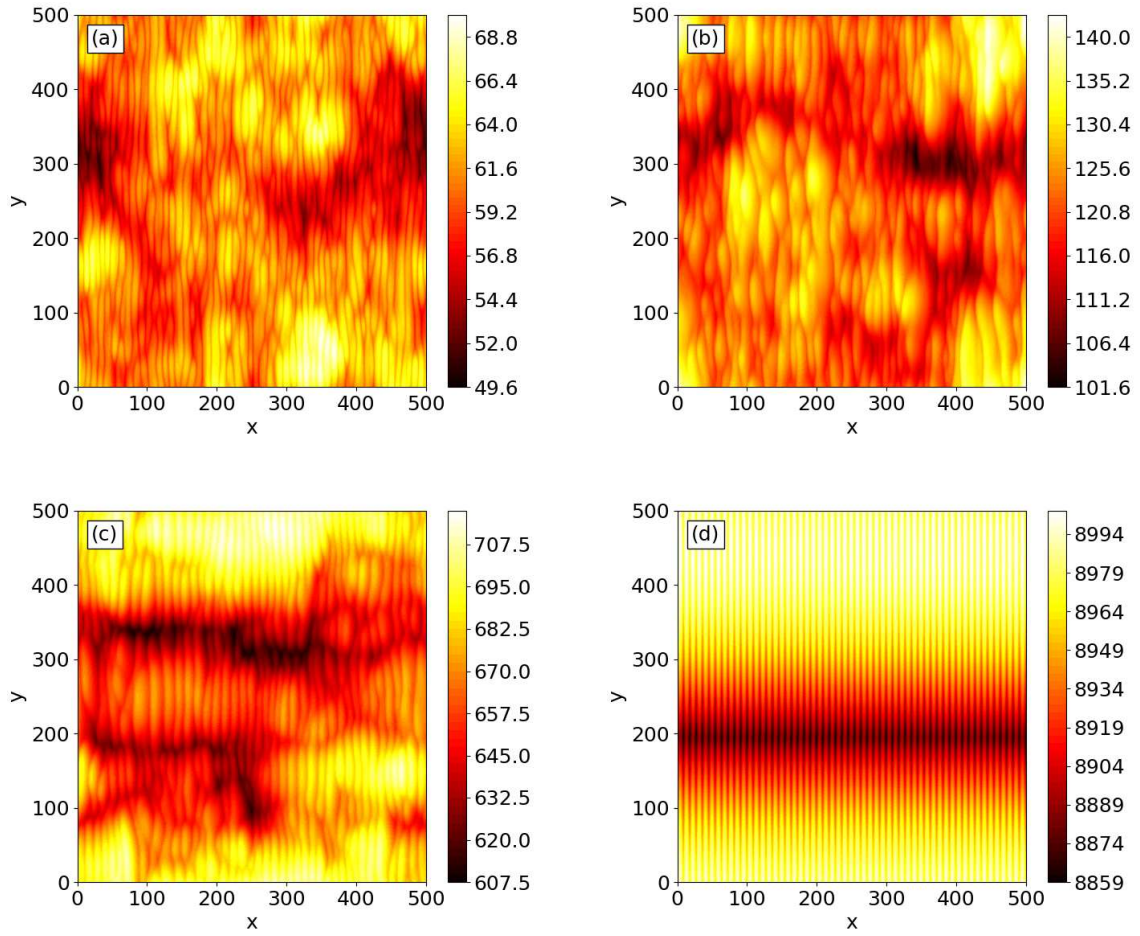

Figure 5.5: u vs x and y at time $t = 200$ for $r_1 = 10$, $r_2 = 1$, $\beta = 0$ and (a) $\alpha = 0$, (b) $\alpha = 1$, (c) $\alpha = 5$ and (d) $\alpha = 50$. The domain size was 500×500.

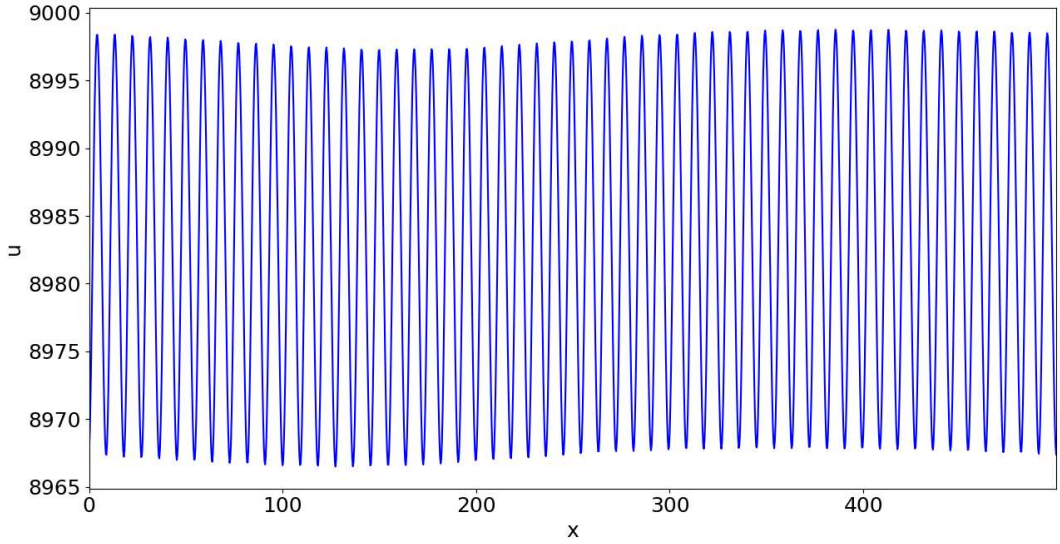

Figure 5.6: Cross section of the simulation shown in Fig. 5.5 (d) at $y = 400$.

have formed in which the ripples are highly ordered. These bands are roughly aligned with the x-direction. Each pair of adjacent bands is separated by a unique type of defect that we will call a “seam." A seam is depressed relative to its surroundings and the ripples on either side of it may or may not be in phase. In fact, as we trace along a seam, the relative phase of the flanking ripples may change. Comparable but much shorter defects can be found in images of surfaces bombarded with an obliquely incident ion beam [30, 57, 123, 124] — see the middle and right panels of Fig. 3 of Ref. [57], for example.

Figure 5.8 is a close-up of the time evolution of two seams. As time passes, the two seams near each other and then merge. After this has occurred, the ripples to either side of the one remaining seam are quite close to being in phase, as in Fig. 5.8 (c). The depression then becomes shallower and at long times the seam disappears for all intents and purposes, very much like the behavior of the seam shown in Fig. 5.7 (d)-(f). The ripples are highly ordered at that point.

Close examination of Fig. 5.5 shows that the wavelength is shorter for $\alpha = 50$ than for $\alpha = 5$. To investigate this point, we calculated the peak-to-peak and trough-to-trough separations for a regularly-spaced sequence of 1D cuts through the surface at time $t = 800$. Each of these cuts

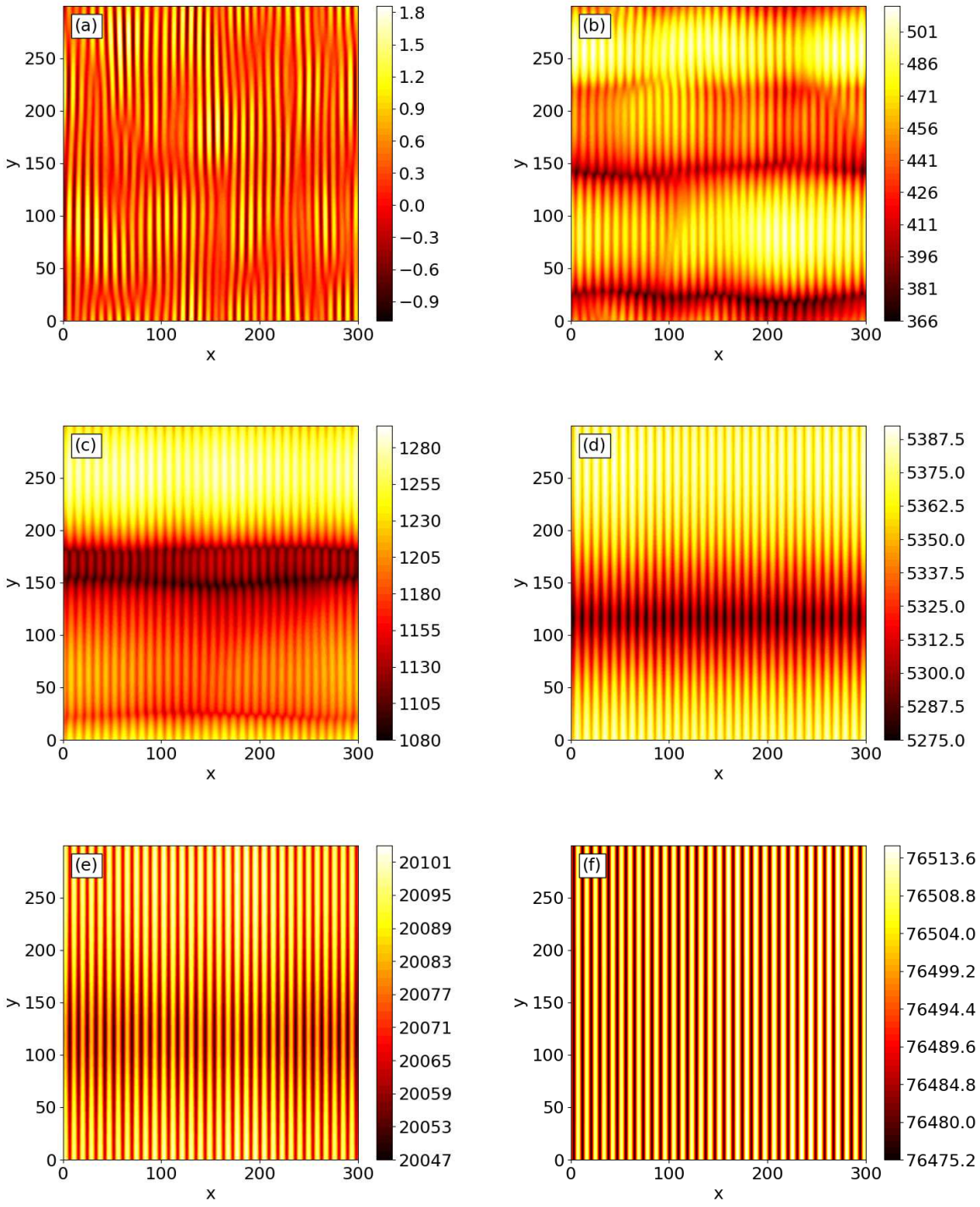

Figure 5.7: u vs x and y for $r_1 = 10$, $r_2 = 1$, $\alpha = 50$ and $\beta = 0$ for times (a) $t = 50$, (b) $t = 70$, (c) $t = 80$, (d) $t = 130$, (e) $t = 310$, and (f) $t = 1000$. The domain size was 300×300.

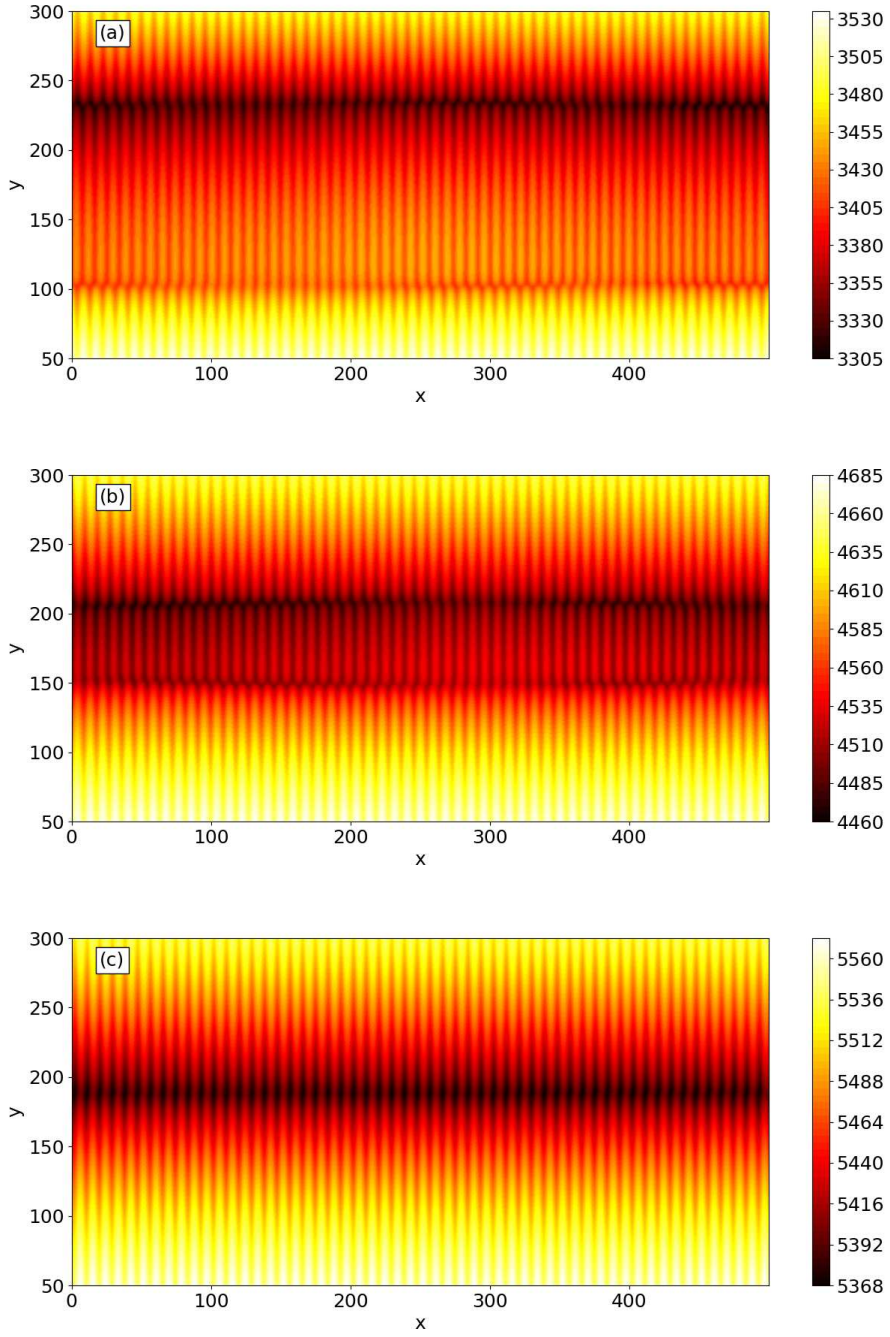

Figure 5.8: u vs x and y for $r_1 = 10$, $r_2 = 1$, $\alpha = 50$ and $\beta = 0$ for times (a) $t = 114$, (b) $t = 132$, and (c) $t = 146$. The simulation domain size was 500×500, but the displayed interval is $50 \leq y \leq 300$ to highlight the seams.

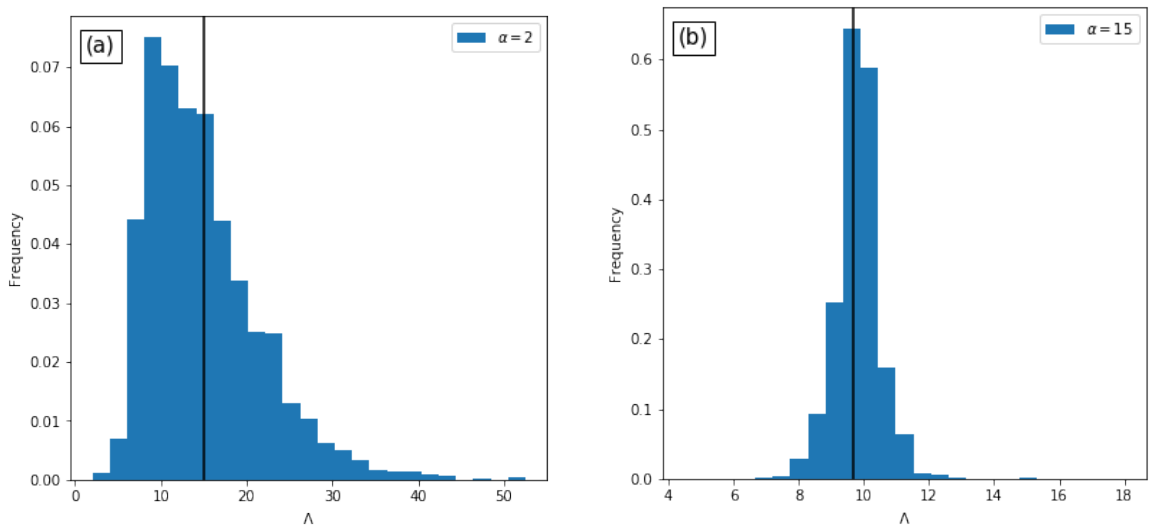

Figure 5.9: Normalized histograms of the peak-to-peak and trough-to-trough separations along the x direction for (a) $\alpha = 2$ and (b) $\alpha = 15$. The black lines display the mean of the distributions. The mean value is 15.1 for $\alpha = 2$ and 9.7 for $\alpha = 15$.

had a different, fixed value of y. Normalized histograms of the separations for $\alpha = 2$ and $\alpha = 15$ are displayed in Fig. 5.9. We observe that just as in the 1D case, the mean wavelength is shorter for the larger value of α. For both values of α, the mean wavelength at long times exceeds the linearly-selected wavelength $2\sqrt{2}\pi \cong 8.886$, and so coarsening occurs in the 2D case, as it does in 1D. The distribution of separations is narrower for $\alpha = 15$ than for $\alpha = 2$, which is in accord with our observation that the ripples become more ordered as α is increased.

The ordered ripples generated by strong dispersion is a fascinating phenomenon. However, thus far, we have restricted our attention to the case $\beta = 0$. It is unlikely that β will be exactly zero in an experiment. We therefore carried out simulations in which we again set $r_1 = 10$, $r_2 = 1$ and $\alpha = 50$ but chose β to be ± 10 (see Fig. 5.10). In both cases, there are regions of highly ordered ripples that are separated by seams, as for $\beta = 0$. For $\beta = 10$, though, the ordered regions are smaller and the seams are more abundant than for $\beta = 0$ or -10.

5.3.2 Triangular Structures

To this point, we have considered systems that either have no variation or that are strongly stabilizing in the transverse direction. (r_1 is relatively large and positive in the latter case.) In

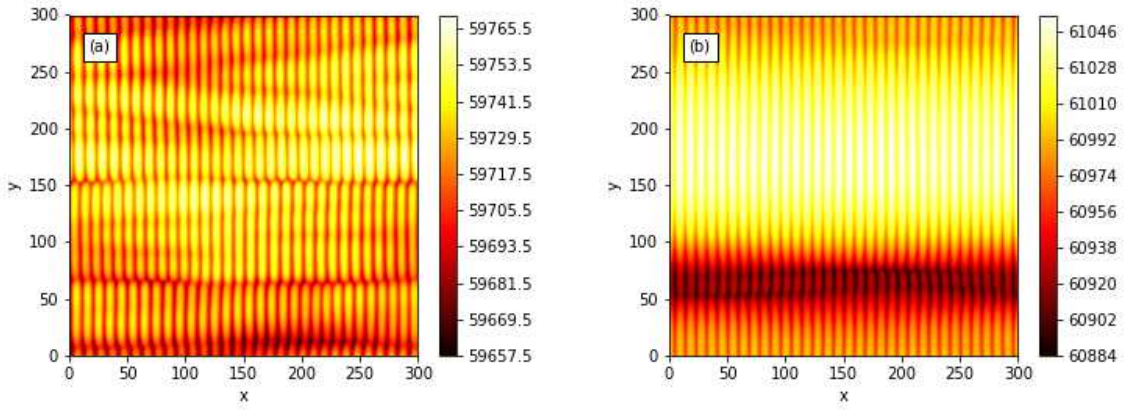

Figure 5.10: u vs x and y at $t = 200$ for (a) $\beta = 10$ and (b) $\beta = -10$. The remaining parameters were $r_1 = 10$, $r_2 = 1$ and $\alpha = 50$ and the domain size was 300×300.

both of these cases, the presence of the dispersive term αu_{xxx} greatly alters the behavior of the surface when α is sufficiently large: the behavior changes from spatio-temporal chaos in the absence of dispersion to highly ordered ripples in the highly dispersive case. However, if the angle of incidence is not too high, r_1 is typically of order 1 and positive. To analyze this situation, we generated a new set of simulations with the parameters $r_1 = 1$, $r_2 = 0$, $\beta = 0$, and with a range of α values. The choice $r_1 = 1$ yields stabilization in the transverse direction, but not to the same degree as previously considered.

The simulations presented in Fig. 5.11 once again show that the behavior changes fundamentally when α is sufficiently large in magnitude. The most interesting feature that is observed in these simulations are the triangular structures that emerge for intermediate values of α: see Fig. 5.11 (c) - (e). Comparable features have been observed in numerous experiments [1–8, 10, 11, 33]. Fig. 5.12 displays a side-by-side comparison of an experimental image of triangular features [1] and the triangular features seen in our simulations. The left image is of a Si surface bombarded with a 700 eV Ar^+ beam oriented at 65°. The simulated surface was produced with parameters $r_1 = 1$, $r_2 = 0$, $\alpha = -3$, and $\beta = 0$ on a 500×500 domain.

In both experimental results and in our simulations, triangular structures with parallel-mode ripples superimposed on them appear. These structures come in two varieties: those

that protrude out of the surface and those that are depressed below the surrounding region. The raised triangular structures are invariably oriented in one direction while the depressed triangles are oriented in the opposite direction. Furthermore, the two varieties do not form with a one-to-one ratio: there are a larger number of raised triangles than depressed ones in our simulations and in experiments.

For the range of α values in which we do observe triangular structures, they are transient. Figure 5.13 shows the surface before the emergence of triangles [panel (a)], while triangles are present [panels (b) and (c)], and after they have disappeared [panel (d)]. Figure 5.13 panels (e) - (h) show the corresponding power spectral densities (PSDs). The PSD is defined to be the squared modulus of the Fourier transform of the difference between the surface height and its spatial average. Before the triangular structures emerge, we see parallel-mode ripples and the corresponding PSD has two peaks centered on nonzero wave vectors on the k_x axis. When triangles are present, these peaks are broader and lower. The PSDs have a distinctive “butterfly" appearance in this regime. The surface at long times appears to exhibit spatio-temporal chaos and there are no discernible peaks in the Fourier spectrum aside from the one centered at $\mathbf{k} = 0$. There are raised and depressed regions on the surface that are elongated along the x-direction at these times.

To further explore the changing nature of the surface morphology, we computed the 1D PSDs for a regularly-spaced sequence of 1D cuts through the surface. Each of these cuts had a different, fixed value of y. We then averaged these PSDs together to give the average 1D PSD for the x direction. We computed the average 1D PSD for the y direction in a completely analogous fashion, except that in this case the 1D cuts had fixed values of x. The results obtained for the same simulation as is shown in Fig. 5.13 are displayed in Fig. 5.14 for times $t = 70$ and $t = 500$. These times were selected because triangles were present at the earlier time but not at the later. The average 1D PSDs for the x direction have a strong peak at $t = 70$ but not at $t = 500$. Thus, ripples were present only at the earlier of the two times. The average 1D PSDs for the y direction show that there was no periodicity in the y direction for either time.

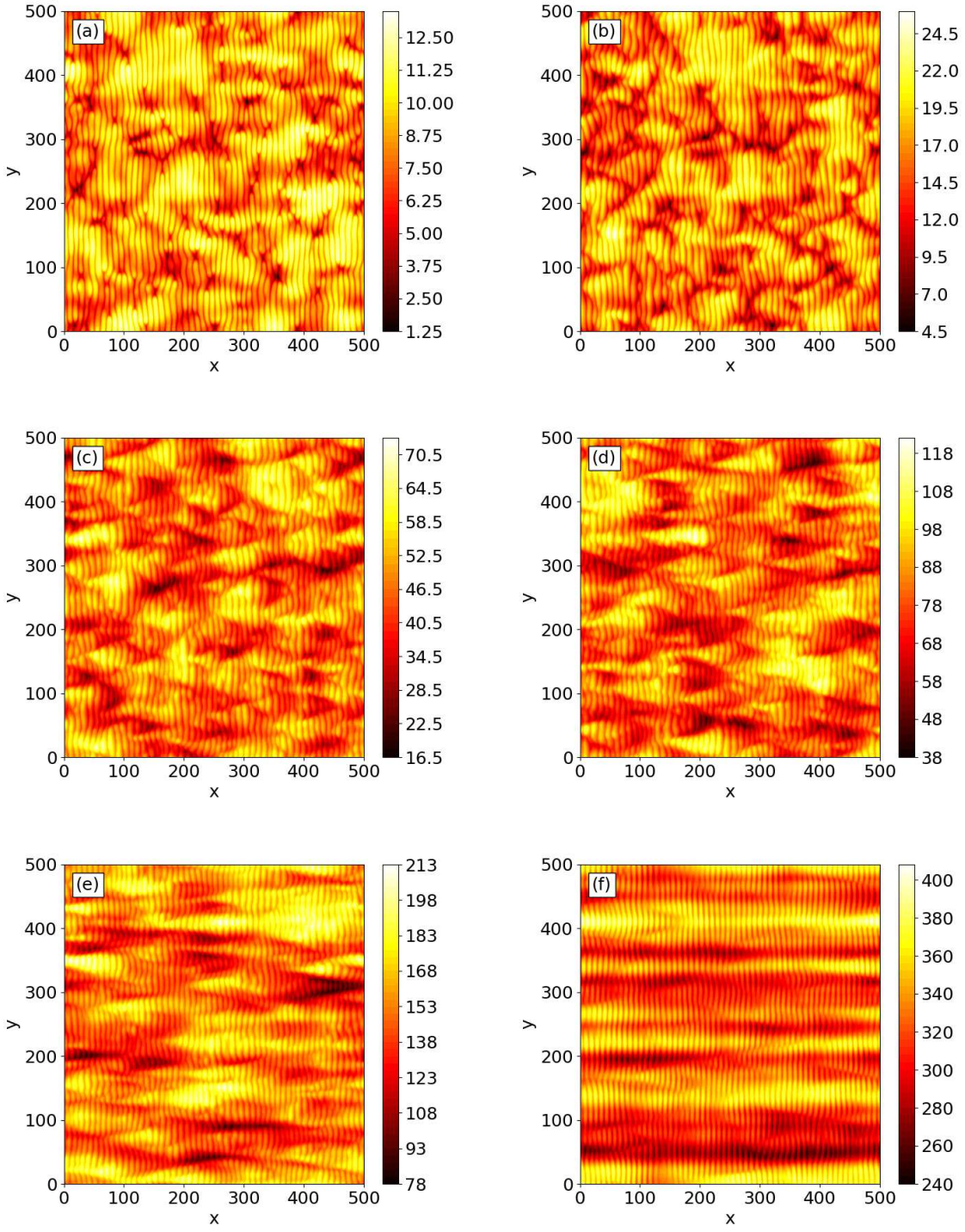

Figure 5.11: u vs x and y at $t = 60$ for (a) $\alpha = 0$, (b) $\alpha = 1$, (c) $\alpha = 3$, (d) $\alpha = 5$, (e) $\alpha = 10$, and (e) $\alpha = 50$. The remaining parameter values were $r_1 = 1$, $r_2 = 0$ and $\beta = 0$, and the domain size was 500×500.

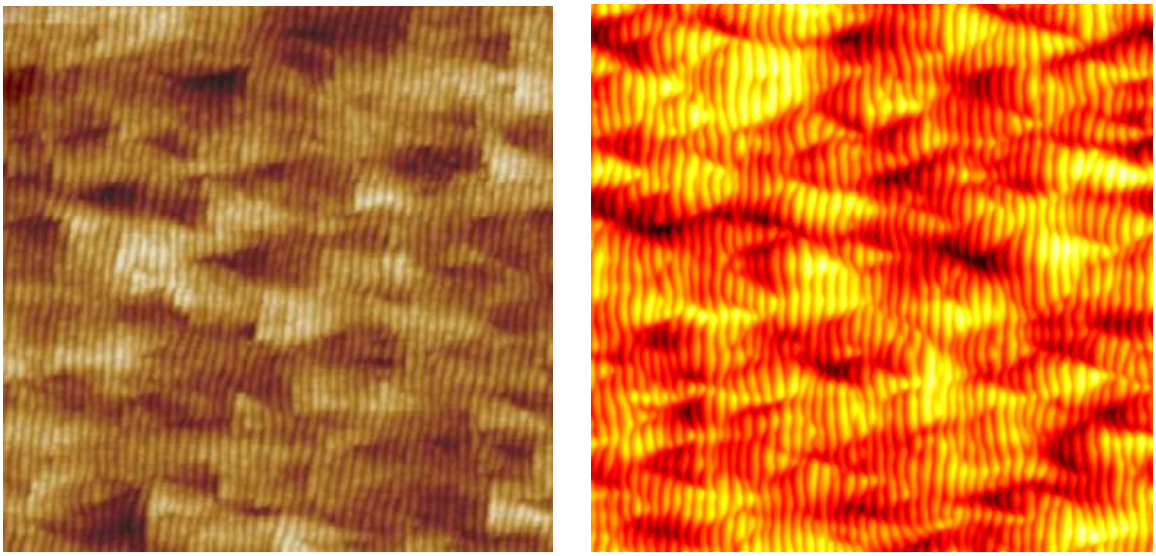

Figure 5.12: Si surface bombarded with a 700 eV Ar^+ beam oriented at 65° at a fluence of 5×10^{17} ions/cm^2 (left). This image was originally originally appeared in Ref. [1]. A simulated surface which was produced with parameters $r_1 = 1$, $r_2 = 0$, $\alpha = -3$, and $\beta = 0$ on a 500×500 domain at $t = 60$ (right).

Figure 5.15 shows the time dependence of the surface width, i.e., the root-mean-square deviation of the surface height from its mean value, for the same values of α as in Fig. 5.11. In the linear regime, the surface width grows exponentially with a growth rate that is independent of α. The surface width rapidly saturates as the nonlinear terms become important. The steady-state surface width is an increasing function of α. There is no discernible regime of power-law growth of the surface width at intermediate times.

To this point in this subsection, we have varied α and kept r_2 and β fixed at zero. It is unlikely that that these parameters are exactly equal to zero in experiments, and so we must see if the triangular structures persist in the presence of the terms $r_2 u_y^2$ and βu_{xyy}. We reduced the domain size and ran simulations with $r_1 = 1$ and $r_2 = 0$ once again, but now with two nonzero values of β and fixed $\alpha = 3$. The simulations — which are shown in Fig. 5.16 (a) and (b) — demonstrate that the additional dispersive term βu_{xyy} alters the detailed structure of the triangles, but they are still present. An additional simulation with $\alpha = 3$, $\beta = 0$, $r_1 = 1$ and $r_2 = 0.2$ shows that the triangles can also form when r_2 is nonzero [see Fig. 5.16 (c)].

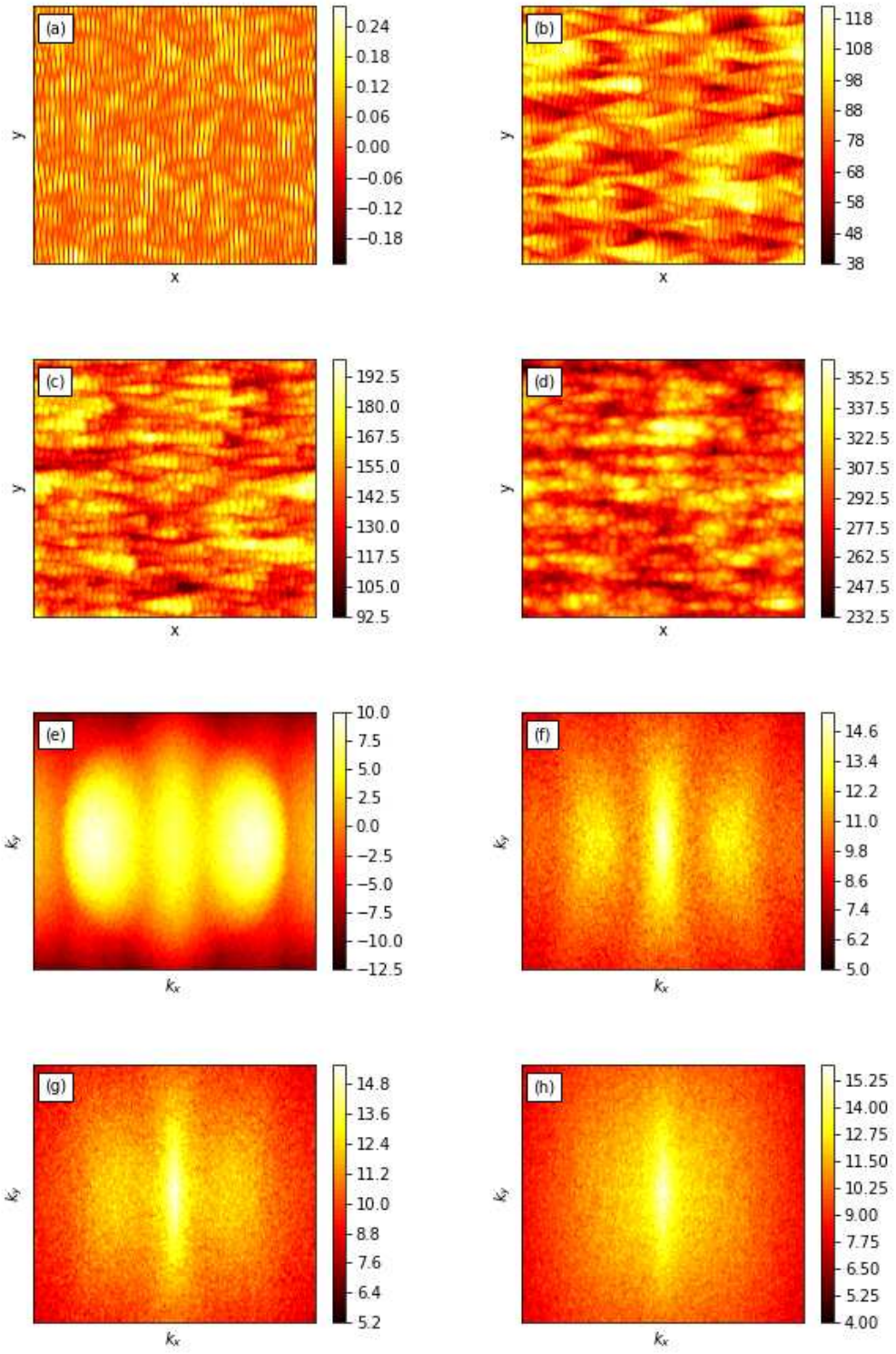

Figure 5.13: u vs x and y (top) and the corresponding PSDs (bottom) for $r_1 = 1$, $r_2 = 0$, $\alpha = 5$ and $\beta = 0$ for times $t = 32$ [(a) and (e)], $t = 60$ [(b) and (f)], $t = 70$ [(c) and (g)], and $t = 108$ [(d) and (h)]. A logarithmic color scale was used for the PSDs so that the satellite peaks could be seen despite the high central peak. The domain size was 500×500.

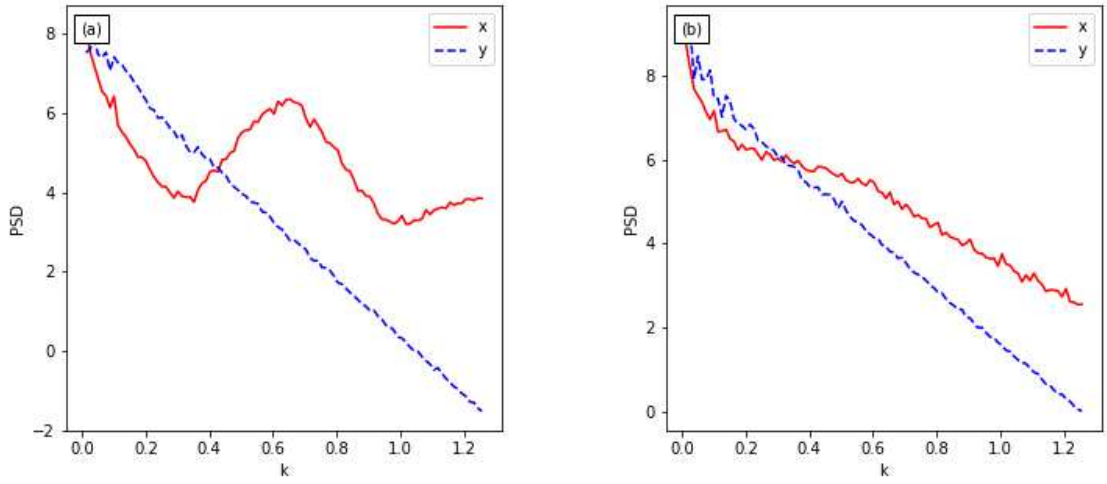

Figure 5.14: Semilog plots of the averaged PSDs for 1D cuts along the x and y directions are shown for $t = 70$ (a) and $t = 500$ (b). The parameter values were the same as in Fig. 5.13 and the domain size was 500×500.

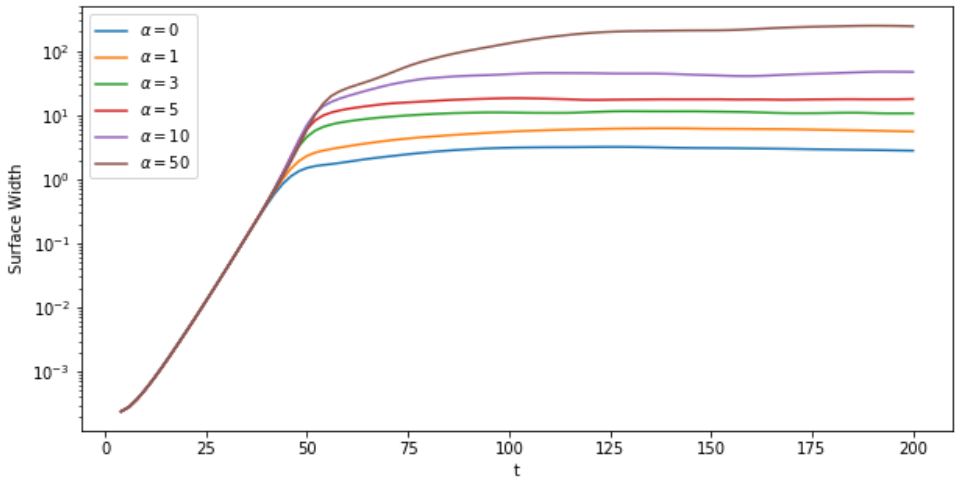

Figure 5.15: Surface width vs time for $\alpha = 0$, 1, 3, 5, 10, and 50. The remaining parameters were $r_1 = 1$, $r_2 = 0$, and $\beta = 0$. The domain size was 500×500.

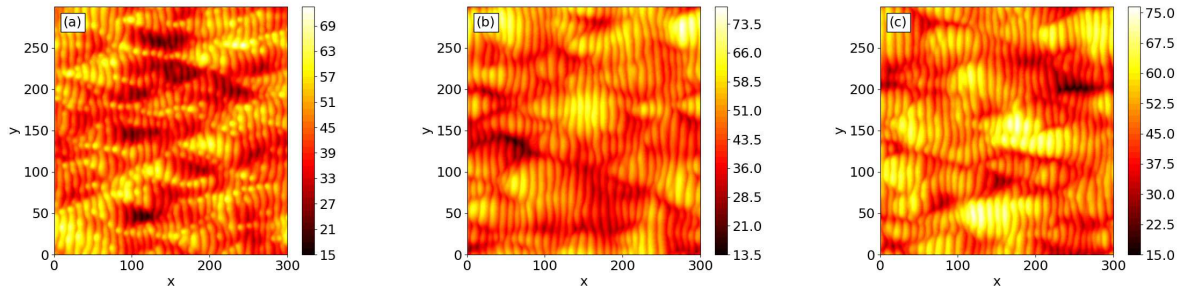

Figure 5.16: u vs x and y at $t = 60$ for (a) $\beta = 3$ and $r_2 = 0$, (b) $\beta = -3$ and $r_2 = 0$, and (c) $\beta = 0$ and $r_2 = 0.2$. The remaining parameter values were $r_1 = 1$ and $\alpha = 3$. The domain size was 300×300.

The triangular structures are not invariant under the transformation $x \rightarrow -x$. This symmetry is broken only if α or β or both are nonzero, i.e., dispersive effects are present. Thus, the experimental observation of the formation of triangular structures provides compelling evidence that dispersive effects are significant for many choices of ion beam and target material.

5.3.3 Perpendicular-Mode Ripples

The presence of the term αu_{xxx} in our equation of motion (5.5) makes the ripple propagation dispersive, i.e., ripples with different k_x values propagate with different velocities. This term therefore tends to elongate protrusions and depressions along the x-direction. This tendency is very much in evidence in Fig. 5.17, which shows the surface height at time $t = 200$ for $r_1 = 1$, $r_2 = 0$ and $\beta = 0$ and for a selection of α values. As α is increased, the protrusions and depressions become increasingly elongated until, for $\alpha = 50$, they span the entire domain. In addition, for $\alpha = 50$, highly ordered, low amplitude parallel-mode ripples are superimposed on the protrusions and depressions. These superimposed ripples are less orderly for smaller values of α.

Protrusions and depressions that are elongated along the x-direction are commonly observed in experiments in which the angle of incidence is relatively high [1]. Traditionally, these have been called perpendicular-mode ripples. Perpendicular-mode ripples occur if C_{22} is negative and smaller than C_{11}, according to the linear BH theory [16]. In this case, there is an instability in the y-direction and it is stronger than the instability if the x-direction, and $r_1 \equiv$

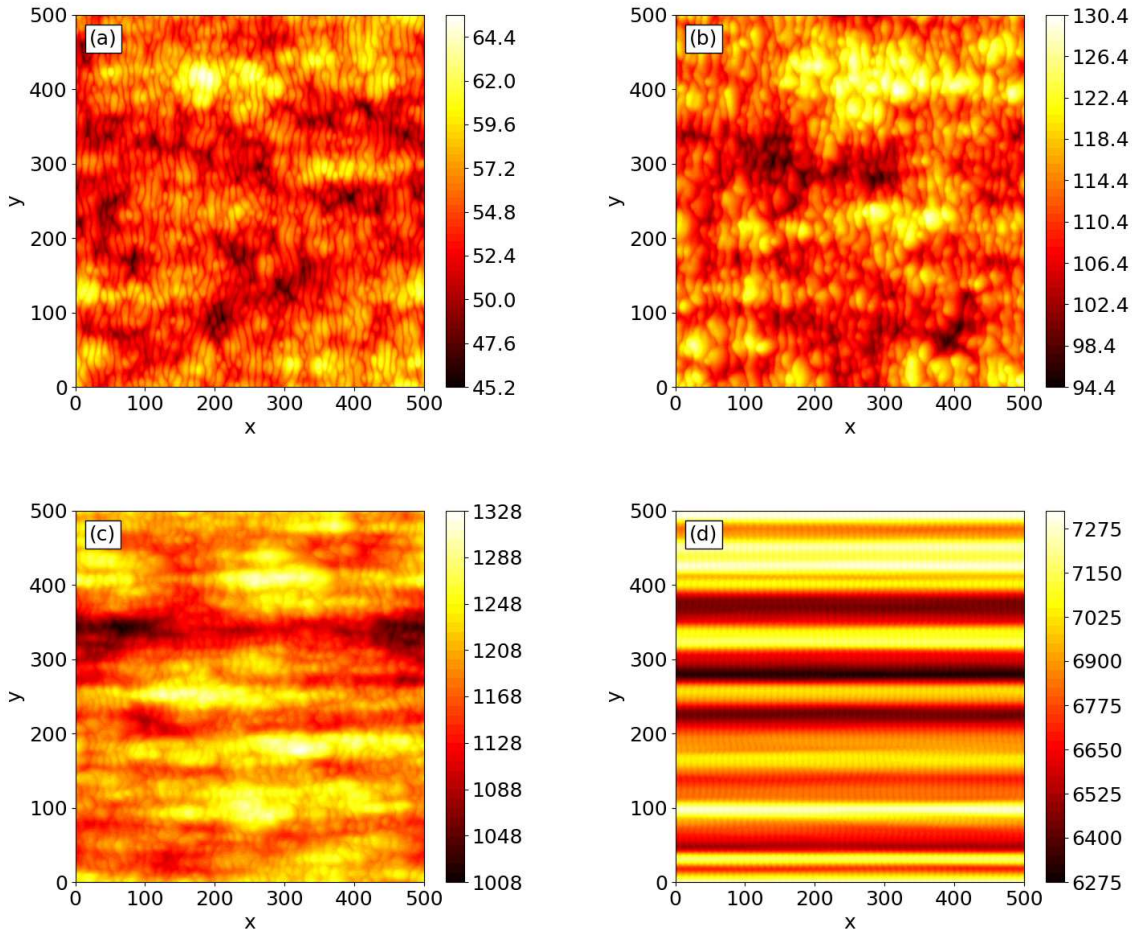

Figure 5.17: u vs x and y at $t = 200$ for (a) $\alpha = 0$, (b) $\alpha = 1$, (c) $\alpha = 10$, and (d) $\alpha = 50$. In each case, $r_1 = 1$, $r_2 = 0$ and $\beta = 0$. The domain size was 500×500.

$C_{22}/|C_{11}| < -1$. The situation is more complex when the equation of motion is taken to be the anisotropic KS equation (2.3): if $r_1 < -1$ and λ_2/λ_1 is positive and sufficiently small, parallel-mode ripples appear at first but are later supplanted by perpendicular-mode ripples [80].

In the simulations that yielded Fig. 5.17, $r_1 = 1$ and so C_{22} was positive. This means that elongated protrusions and depressions that resemble the experimental topographies can occur even if there is no instability in the y-direction. These topographies are instead the result of dispersion and the lowest-order nonlinearities.

In recent years, atomistic simulations combined with the crater function formalism have been used to estimate the values of the coefficients in the continuum equation of motion [19, 21, 28, 86]. These studies have shown that if the ion energy is on the order of 1 keV or smaller, the contribution mass redistribution makes to the curvature coefficients C_{11} and C_{22} is usually about an order of magnitude larger than sputtering's contribution. Mass redistribution makes a positive contribution to C_{22} [28, 84]. The experimental observation of so-called perpendicular-mode ripples at relatively high angles of incidence has therefore been a puzzle. Our work shows that this apparent quandary may be no quandary at all: adding dispersion and the lowest-order nonlinear terms to the EOM can yield topographies very much like those seen in experiments even if the positive contribution of mass redistribution to C_{22} is larger than negative contribution of curvature-dependent sputtering.

If the angular dependence of the sputter yield is expanded to third order in the surface slope, the cubic nonlinearity u_x^3 appears in the EOM [22]. Recent work has shown that this term can lead to the formation of pyramidal structures that are elongated along the projected beam direction [181]. These too might be considered to be perpendicular-mode ripples. As a consequence, the situation is complex: in a given experiment, so-called perpendicular-mode ripples might result from dispersion, the angular dependence of the sputter yield, from curvature-dependent sputtering, or from a combination of all three.

5.3.4 Nano-needles

Another interesting feature observed in experiments is the production of "nano-needles". In many examples of perpendicular-mode ripples observed in experimental studies, needle-like structures are present [1, 2, 15, 33, 50, 58]. These structures are not invariant under the transformation $x \to -x$ but Eq. (2.3) is. This led us to ask if dispersion could explain these experimentally observed features.

Figure 5.18 shows an AFM image from Ref. [15] (left) and a simulated surface produced by Eq. (5.1) (right). This AFM image is of a Si surface exposed to bombardment with 1 keV Ar^+ ions at a 75° angle of incidence. The fluence for this image is 1.6×10^{19} ions/cm^2. The simulated surface was produced with the parameters $C_{11} = 1$, $C_{22} = -1$, $B = 1$, $\lambda_1 = 1$, $\lambda_2 = 1/5$, $C_{111} = 50$, and $C_{122} = 0$. The domain size was 500×500 and the selected image was at $t = 200$. Both the experimental surface and the simulated surface show needle-like protrusions.

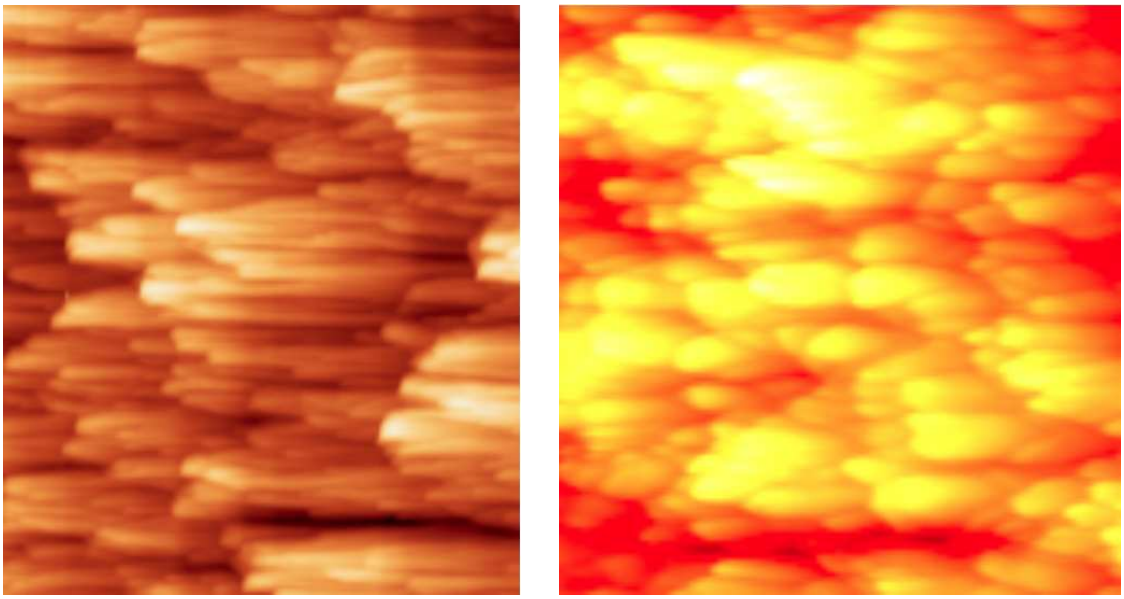

Figure 5.18: AFM scan of a Si surface exposed to Ar^+ ions bombardment at 75° (left). This image originally appeared in Ref. [15] as Fig. 1. A contour plot of a simulated surface with parameters $C_{11} = 1$, $C_{22} = -1$, $B = 1$, $\lambda_1 = 1$, $\lambda_2 = 1/5$, $C_{111} = 50$, and $C_{122} = 0$ for Eq. (5.1) at $t = 200$ (right). The domain size was 500×500.

This shows that the lowest order dispersive terms can produce not only triangular protrusions, but also perpendicular-mode ripples with broken $x \to -x$ symmetry. These bear a striking resemblance to experimentally observed structures that cannot be reproduced by Eq. (2.3).

5.4 Related Work

The equation of motion (5.1) studied in this chapter is a special case of the equation derived by Makeev, Cuerno and Barabási (MCB) [18] in which terms proportional to $u_x u_{xx}$ and $u_x u_{yy}$ are omitted and the effective surface diffusivity is taken to be isotropic. MCB only studied the low-amplitude, linearized behavior predicted by their equation.

A still more general equation of motion was derived by Muñoz-García, Cuerno and Castro (MCC) starting from a model in which it is assumed that there is an amorphized, mobile surface layer [183]. Unlike MCB, MCC carried out numerical integrations of their equation of motion in which the nonlinear terms were retained. MCC noted that when terms proportional to u_{xxx} and $u_x u_{xx}$ appear in the EOM, the ripples become asymmetric and their propagation becomes dispersive. They did not observe the formation of triangular structures or highly ordered ripples in 2D, however.

The special case of our rescaled equation of motion (5.5) in which $r_1 = r_2 = 0$ and $\beta = \alpha$ has already been studied in an entirely different physical context. In this case, much as we did in 1D, we differentiate Eq. (5.5) with respect to x and set $\bar{x} = -x$, $\bar{y} = y$, $\bar{t} = \alpha t$ and $v = (2/\alpha) u_x$. After dropping the bars, we obtain

$$v_t + \alpha^{-1}(v_{xx} + \nabla^2\nabla^2 v) + v v_x + \nabla^2 v_x = 0. \tag{5.6}$$

Equation (5.6) has been studied previously since it models the flow of a thin liquid film down a vertical plane [126]. It reduces to the Kawahara equation (5.3) if u is independent of the transverse coordinate y.

In the highly dispersive $\alpha \to \infty$ limit, Eq. (5.6) becomes the 2D version of the Zakharov-Kuznetsov (ZK) equation [127] for ion-acoustic waves propagating along the magnetic field in a strongly magnetized, two-component plasma,

$$v_t + v v_x + \nabla^2 v_x = 0. \tag{5.7}$$

The ZK equation (5.7) is a generalization of the KdV equation (5.4) to 2D. The soliton solutions to the KdV equation are also solutions of the ZK equation. However, these so-called 1D solitons are unstable against perturbations of sufficiently long transverse wavelength [128–132]. When disturbed, a 1D soliton breaks up into a chain of localized, bell-shaped solitons [133].

Simulations of Eq. (5.6) for relatively large α and with white noise initial conditions produce V-shaped structures composed of equally spaced, identical bell-shaped solitons that are superimposed on a background of low amplitude ripples [126]. This conclusion is also supported by analytical work [134]. We initially thought that the V-shaped structures could be the differentiated form of the triangular structures we observe in our simulations of Eq. (5.5). This turns out not to be the case, however: the V-shaped structures occur at much later times than the triangles, and, when integrated, they do not resemble the triangular structures. In addition, the V-shaped structures are found for significantly larger values of α than those for which triangular structures occur. Finally, the V-shaped arrangements of bell-shaped solitons seem not to be transient, unlike the triangular structures we find.

5.5 Discussion

In this chapter, we focused on the effect that the linear dispersive term u_{xxx} has on the patterns produced by oblique-incidence ion bombardment of a solid surface. We found that it can lead to the formation of triangular regions that are traversed by parallel-mode ripples, and these strongly resemble nanostructures observed in many experiments. We included the lowest order nonlinear terms (which are proportional to u_x^2 and u_y^2) in our equation of motion (5.5). As

time passes and the amplitude of the pattern grows, higher order nonlinear terms could begin to modify the behavior of the surface. Quite a number of such terms have been considered and they would likely all influence the surface dynamics at sufficiently long times [18, 22, 181, 183]. However, adding additional terms to the equation of motion would further complicate it, would introduce more dimensionless parameters into a model that already has four, and is accordingly beyond the scope of the present work.

Perhaps because we included only the lowest order nonlinear terms in our EOM, it does not reproduce all aspects of the experiments. At early times, the triangular structures produced in experiments have cross-sectional profiles that are similar to the ones we find in our simulations (see Fig. 5.19). However, at later times, the cross section of a triangular structure begins to take on a sawtooth form in some experiments. Such a sawtooth is asymmetric: one of its sides has a significantly larger slope than the other. Our model does not yield this behavior. In addition, the triangular structures produced by our EOM (5.5) are transient. This transience has not been observed experimentally. This may be because the experimental fluences are not high enough for the triangles to be replaced by perpendicular-mode ripples. However, it is also possible that adding additional nonlinear terms to the EOM would increase the longevity of the triangles and so yield improved agreement with experiment. The experimentally observed triangular structures can also become larger with passing time, and our model fails to reproduce this behavior.

Our equation of motion (5.5) produces coarsening, i.e., the ripple wavelength increases with time. This increase can be substantial — we find that the wavelength nearly doubles for $\alpha = 1$ in 1D, for example. The ripple wavelength also increases as time passes in experiments that produce triangular structures. Coarsening can also be caused by the so-called conserved Kuramoto-Sivashinsky nonlinearity $\sum_{i=1}^{2}\sum_{j=1}^{2}\lambda_{i,j}^{(2)}\partial_{x_i}^2(\partial_{x_j}u)^2$; here the $\lambda_{i,j}^{(2)}$s are constants [66, 183]. This term is of higher order in both u and the spatial derivatives than the linear dispersive terms u_{xxx} and u_{xyy}, though, and so it is expected to play an important role later in the time evolution than the linear dispersive terms. The cubic nonlinearity u_x^3 also causes coarsening [22, 181] and is of higher order than the linear dispersive terms. In experiments, it

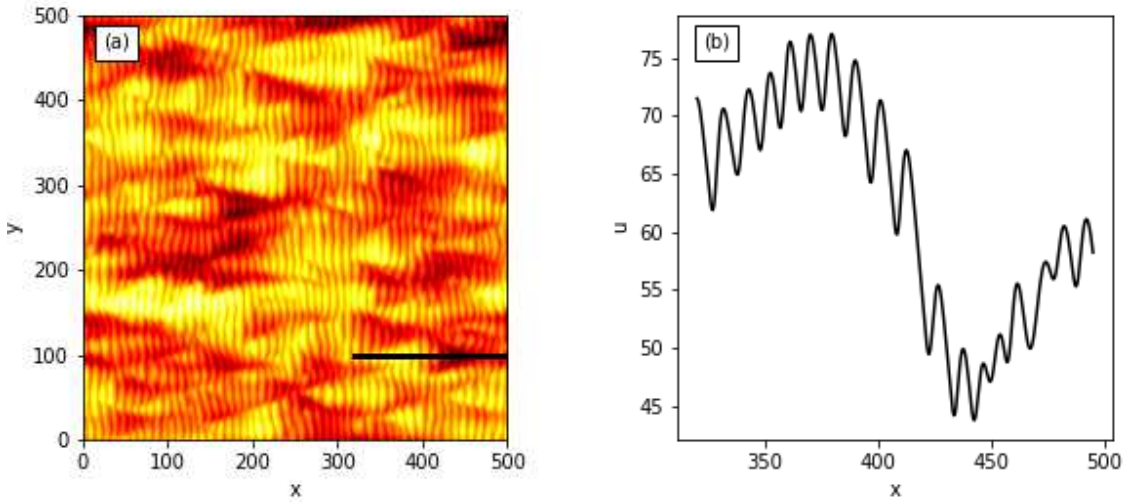

Figure 5.19: (a) u vs x and y for $r_1 = 1$, $r_2 = 0$, $\alpha = 3$, and $\beta = 0$ at $t = 76$. The domain size was 500×500. (b) The cross section of the surface taken along the black line shown in (a). The cross section is through one of the triangular structures.

is likely that linear dispersion, the conserved Kuramoto-Sivashinsky nonlinearity and the cubic nonlinearity all contribute to ripple coarsening, and that their relative importance changes with time. In all three cases, the ripple wavelength saturates at sufficiently long times, and so the coarsening is interrupted.

The linear dispersive term u_{xxx} is not invariant under the reflection $x \to -x$. Including this term in the EOM has the effect of making the ripples asymmetric, as we saw in Sec. 5.2. There are also nonlinear terms that are not reflectionally symmetric — for example, u_x^3 and $u_x u_{xx}$. If present in the EOM, these terms will also contribute to the asymmetry of the ripples [22, 183]. In addition, these terms alter the propagation velocity of the ripples as the ripple amplitude grows, i.e., they produce nonlinear dispersion [22, 183].

5.6 Conclusions

We showed that dispersion can have a crucial effect on the patterns produced by oblique-incidence ion sputtering, even though its effect has almost universally been ignored in the past. Our work yielded four key conclusions:

1. Dispersion can lead to the formation of triangular regions that are traversed by parallel-mode ripples. These structures come in two varieties: those that are raised above the surrounding surface and those that are depressed below it. The raised triangular structures are always oriented in one direction while the depressed triangles are oriented in the opposite direction. We also found that there are a larger number of raised triangles than depressed ones. Triangular structures with precisely these features can be found in many micrographs of ion-sputtered surfaces.

2. If dispersion and transverse smoothing are sufficiently strong, highly ordered ripples can form. Strong effective dispersion can be realized by choosing the angle of ion incidence so that it is just above the critical angle for the formation of parallel-mode ripples. Strong transverse smoothing, however, might be difficult to achieve in practice. Nonetheless, the possibility of exploiting dispersion to produce highly ordered ripples seems a topic worthy of experimental investigation.

3. Dispersion can lead to the formation of protrusions and depressions that are elongated along the projected beam direction even if there is no transverse instability. This may explain why topographies of this kind can form when ion-induced mass redistribution dominates curvature-dependent sputtering.

4. The ripple wavelength increases with time and then saturates as a result of dispersion. Prior research has shown that this behavior — which is known as interrupted coarsening — can also occur due to the presence of nonlinear terms in the equation of motion [22, 66, 181, 183].

Chapter 6

Overview of Machine Learning

6.1 Introduction

Before the next project can be discussed, it will be necessary to give a general overview of machine learning (ML) and deep learning (DL). ML is a subset of artificial intelligence (AI) that focuses on the study and application of algorithms and statistical models that enable a computer to learn to recognize patterns within a dataset and to report the associated meaning to the user. ML methods allow a computer to "learn" without explicit rules or human intervention [135]. ML has proven to be extremely useful when given sufficient data to "train" the model. These models are used in the financial and tech sectors as well as academic research [136–138].

Depending on the nature of the data and the objective of the model, ML algorithms can be categorized into supervised, unsupervised, and reinforcement models. Supervised learning is the ideal case in which the data is well understood and contains known "labels" or "targets" that the model is designed to predict [135]. These models are trained by giving the model data and then asking it to predict a label. The model is then tuned to maximize the performance. This tuning process will be discussed in section 6.2. One common application of supervised learning is image classification.

The second category of ML models is unsupervised models. Unsupervised ML algorithms are useful for data that do not have known labels. Typical applications of this type of ML algorithm include anomaly detection and clustering of user preferences [135]. Clustering can be useful for content-based recommendation systems; an algorithm can cluster user preferences in order to recommend items to other users in the same cluster.

The last category is reinforcement models. Models in this category are especially useful when dealing with data that changes as a result of the model's output, such as driving a car or playing a video game. These models take input data, such as an image on a screen, and come up

with a beneficial action. This action then causes the environment to change, thereby altering the data. The new state of the environment is once again given to the model. AlphaGo is the premier example of a successful reinforcement model. In 2015, DeepMind Technologies developed AlphaGo, the first computer program to beat a professional Go player [139]. AlphaGo's successor AlphaZero is currently considered the top "player" in Go and possibly chess [140]. In 2019, Deepmind published a paper on the newly developed AlphaStar. This reinforcement model achieved the Grandmaster level in StarCraft II, a real-time strategy game developed by Blizzard Entertainment. AlphaStar is ranked above 99.8% of officially ranked human players [141].

ML algorithms are being applied in a wide variety of fields and seeing astonishing success. Examples include the speech recognition in Amazon's Alexa or a Google Home device [142,143], driving assistance in Tesla cars [144], and the recommendation system utilized by Netflix [145]. The advent of robust models which can outperform humans, and which can be adapted for many tasks, is rapidly reshaping the world. The impact of AI development and the potential outcome is a point of great discussion ranging from Utopian idealism to the cataclysmic collapse of human civilization [146, 147]. The direction of the field of AI, as well as the ethics thereof, are deep and interesting topics; however, it is beyond the scope of the work presented in this dissertation.

6.2 Training and Evaluating a Supervised Learning Model

Supervised training was used for the work presented in Chapter 7. Supervised learning is particularly useful because it directly compares the output values from the ML algorithm to the "true" values associated with the data and alters the algorithm to align these two sets of values. This comparison can be made to identify distinct classes, such as species of dogs, or continuous values, such as the weight of a dog. These two types of problems are called classification and regression, respectively. Supervised learning algorithms include linear regression, logistic regression, decision trees, and support vector machines [135].

Supervised learning models use three main components: features, labels, and weights. Features, commonly denoted by the variable x, are measurable properties or characteristics of items in the dataset. The features given to the model usually do not include all of the features in a dataset. A human usually needs to reduce the number of features used in a model; this process is called dimensionality reduction [135]. This prevents the model from training on features that are not useful. The model uses the provided features to predict what the "true" values are. These "true" values are called labels and are denoted by the variable y. By directly comparing the predicted value for a given set of features to the "true" value, the weights, w, can be adjusted to better match the predicted and true values. This process of adjusting the weights is called training. The weights are the tunable parameters within a ML model. The user can also vary how the training is done by changing external hyperparameters. Common hyperparameters include the learning rate which adjusts how much the weights can change in a single step, the optimization method used to adjust the weights, and the size of the training set.

In order to apply a supervised learning model to a given problem, a few steps must be taken. First, the goal has to be clearly defined: what exactly is the objective of the model and which criteria will be used to gauge how it is performing? Next, data must be gathered for training and validation. This data has to be labeled so that the model can compare the predicted values to the true values.

Once the dataset has been gathered, the useful features within the dataset must be identified. Often datasets come with redundant or irrelevant information. Feeding all of these features into the model and allocating computation resources for them is wasteful. If the number of features is too large, the model is harder to train and faces the so-called "curse of dimensionality" [135]. If not enough features are included, the model will not have enough information to produce accurate predictions.

Next, an evaluation method must be selected. The evaluation is done either through a metric, such as accuracy or precision, for classification problems, or through a loss function, such as mean-squared error (MSE), for regression. In both cases, the objective is to maximize perfor-

mance by aligning the model's output as closely to the true values as possible. For classification, the objective could be maximizing the number of correctly predicted classes. Which metric or loss function is chosen depends on the problem at hand. For example, if the data has two classes and one is significantly more common than the other, accuracy is not a useful metric because the model will always guess the majority case and perform "well" according to this metric. A complete discussion of metrics and loss functions can be found in Ref. [148].

At this point, the dataset is segmented into at least two subsets, but three is generally preferred: training, validation, and test sets. The training set is the data that is given to the model and used for tuning the weights and "teaching" the model how to predict the correct result. For each round of training, the model predicts the target for the training data, and then compares these results to the true values. The weights in the model are typically adjusted through some form of gradient descent (GD) as will be discussed in the next section. The adjusted weights help fit the model to the data and allow accurate predictions.

After the model weights have been adjusted, the performance of the model must be tested. The validation set is used for this. The model is given the data in the validation set and asked to predict what the labels are. The predicted values are again compared to the true values, but the weights are not updated. This provides an unbiased check of the model's performance. This allows the user to make sure that the model is learning correctly and adjust hyperparameters like the learning rate. If more training is needed after validation, the training set is again given to the model, and the weights adjusted again. The new results are then validated. Each time the model processes the entire training dataset is called an epoch. After each epoch, the model is validated. If the model's performance is considered adequate, training is stopped, and the model is considered fully trained and is considered to be ready for use.

Often an optional third set of data is used as a final verification. The test set is a reserved segment of the dataset that the model is only shown once it is fully trained. This means that the test set is a truly unbiased source for evaluation of the final model. It is not used for training or

hyperparameter tuning. The model's performance on the test set is used for making the final decision as to if the model is ready for use.

6.2.1 Gradient Descent

Gradient descent (GD) is the primary method used to adjust model's weights [135, 148, 149]. Just like the gradient can be used to find maxima and minima of a function, it can be be used to determine the parameters in an algorithm that results in the maxima or minima of the evaluation metric. In the case of regression, the objective is to minimize the loss function. This maximizes performance and provides the best possible model. The equation for GD is

$$w_{n+1} = w_n - \gamma \nabla_w f(w_n), \tag{6.1}$$

where w_n is a vector of the weights at the nth step. γ is the learning rate; a simple analog is the the step size h used in Euler's method. If γ is too large, the adjusted weights will overshoot the minimum value. If γ is too small, the model will take a very long time to converge. ∇_w is the gradient with respect to w. $f(w)$ is the loss function evaluated for the weights w.

With Eq. (6.1), the weights can be adjusted, and the model's performance maximized. However, this can take a very long time, especially for complex models. Variations of GD for faster optimization have been developed. Some popular ones are Momentum, RMSProp, AdaGrad, and Adam optimization. A more complete discussion can be found in Refs. [135, 149].

6.2.2 Adam Optimization

In Chapter 7 we will use the Adam optimization method to train our ML model. "Adam", which stands for adaptive moment estimation, was introduced in 2014 by D. Kingma and J. Ba [151]. Two key ideas that were introduced in other optimization methods, AdaGrad and RMSProp, are combined in the Adam optimization method. Adam is an adaptive learning rate method like AdaGrad [152]. This means that each individual parameter has a different learning rate. Adam also dynamically adjusts the learning rates like RMSProp [153]. The learning rate

of a particular parameter is adapted based on running averages of the first two moments of the gradient.

The motivation for using Adam is most clearly stated in the abstract of Ref. [151]: "The method is straightforward to implement, is computationally efficient, has little memory requirements, is invariant to diagonal rescaling of the gradients, and is well suited for problems that are large in terms of data and/or parameters. The method is also appropriate for non-stationary objectives and problems with very noisy and/or sparse gradients. The hyper-parameters have intuitive interpretations and typically require little tuning" [151]. Simply put, Adam is a computationally efficient method to optimize the parameters of a ML model.

Using Adam to update the weights requires some initialization. First, the number of data points to be analyzed before updating the weights needs to be selected. This can be the entire training set, where the weights are updated after an epoch, or a smaller subset called a batch. This is necessary to build up a distribution of the gradients from which the first and second moments can be calculated. Then the initial weight vector w_0 is initialized, usually with random values. Two vectors, m_0 and v_0, are then initialized to track the mean and variance of the gradient, which are the first and second moments, respectively. All of the elements in both m_0 and v_0 are initialized as 0's.

The process of updating the weights from a step n to $n+1$ is done according to the same process for every step. The model is given data for training purposes, either a batch or the whole training set, and the loss function $f(w_n)$ is calculated for each data point. The gradient of the loss function with respect to the weights is then calculated by

$$g_{n+1} = \nabla_w f(w_n). \tag{6.2}$$

where g_{n+1} is defined as the gradient used to update the weights at a step $n+1$.

The algorithm then updates an exponential moving average of the gradient. This is calculated by

$$m_{n+1} = \beta_1 m_n + (1-\beta_1) g_{n+1}, \tag{6.3}$$

where β_1 is a hyper-parameter which sets the exponential decay rate of the moving average. By using the mean to update the weights, it can gain "momentum" by amplifying the effect of gradients which are aligned in the same direction. This helps move the weights in the desired direction to minimize the loss function.

The variance is updated through a similar process. The equation for this is

$$v_{n+1} = \beta_2 v_n + (1-\beta_2) g_{n+1}^2, \tag{6.4}$$

where β_2 is another hyper-parameter which controls the exponential decay of the moving average of the second moment. Using the variance to update the weights helps push the trajectory of optimization towards smooth paths. This reduces oscillations and random walks that can occur during the optimization process.

These moving averages are estimates of the first and second moments of the gradient. However, these values are biased towards 0, especially at small n values. This occurs because both m_0 and v_0 are initialized as vectors of 0's. To counteract this bias, a correction needs to be done to both m_{n+1} and v_{n+1}. The bias-corrected moment estimates are calculated by

$$\hat{m}_{n+1} = \frac{m_{n+1}}{1-\beta_1^{n+1}} \tag{6.5}$$

and

$$\hat{v}_{n+1} = \frac{v_{n+1}}{1-\beta_2^{n+1}}. \tag{6.6}$$

The weights w_n are then updated using the bias-corrected first and second moment estimates. This is done according to the equation

$$w_{n+1} = w_n - \alpha \frac{\hat{m}_{n+1}}{\sqrt{\hat{v}_{n+1}} + \epsilon}, \tag{6.7}$$

where α is the "step" size and ϵ is a very small nonzero value included to prevent division by zero. This process is repeated until the model is trained.

By using the mean and variance, Adam is able to minimize the loss function without getting trapped in local minima. This process has shown favorable results when compared to Stochastic Gradient Descent, AdaGrad, and RMSProp; both in terms of minimizing the loss function and reducing the training time. Adam handles sparse gradients efficiently and has shown the ability to work very well on large datasets and high-dimensional parameter spaces. Adam is commonly used for natural language processing and computer vision problems. It is included in most deep learning frameworks and tends to perform well with default parameter values. These default parameter values were determined in Ref. [151] and given as: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\alpha = 0.001$, and $\epsilon = 10^{-8}$.

6.2.3 Overfitting and Underfitting

Adam provides an effective method to optimize the weights of a model given features of the training data, but that is not the primary objective. The objective is to train a model that can reliably predict the labels starting from arbitrary data, not just from the training set. This raises the question of how to determine that the model training is sufficient. This is actually a question about overfitting and underfitting.

An underfit model is one that does not understand how to associate the input features with the target. This is identifiable by poor performance on the test, validation, and even the training sets. The most common sources of underfitting are lack of training or lack of information. If the weights in a model have not yet been tuned, it makes sense that the model does not yet know how to identify the target. This can be remedied by more training. However, that does not always work. The more insidious case is when the data itself is insufficient. If the features given to the model do not contain enough information to identify the target, it will never be able to do so.

On the other hand, a model can also be "overfit". This is a common concern with ML algorithms. A model is considered "overfit" when it starts to identify and utilize patterns that exist within the training set but do not generalize to the validation or test sets. Overfitting leads to

excellent performance on the training data, but poor performance on the validation and/or test sets. This can occur when the minima of the loss function on the training set do not agree with the minima from the other sets.

Figure 6.1 is a graphical demonstration of this phenomena. A dataset was generated using the function $y = \cos(x)$, where x is the feature and y is the target. The objective is to create a model which can take a x value and predict the associated y value. Fifty x values were randomly selected in the range [0,1]. Then the corresponding y values were calculated for each point. Noise was added to the dataset by adding a random value between -0.1 and 0.1 to each y value. This dataset was then segmented into a training and a validation set using an 80/20 split.

Next, a polynomial model was generated to describe the data. The degree of the polynomial varied from 1 to 45. Each polynomial equation had the parameters tuned to minimize the MSE between the training data and the predicted value from the model. The linear fit clearly does not represent either the training or validation data and is underfit. A higher degree polynomial is needed to describe the data accurately. The 4th-degree polynomial, however, quite accurately models both datasets and closely aligns with the true function used to generate the data. However, if we continue adding more degrees of freedom and more tuneable parameters, the model starts to overfit. The 35th-degree polynomial very closely matches the training data but does not match the validation set or the true function. The additional parameters in the 35th degree polynomial lead to a smaller MSE on the training data than what was observed for the 4th-degree polynomial. Fig. 6.2 shows how the MSE changes as the degree of the polynomial function increases. Initially, the model is underfit and has large MSE values for both the training and validation set. As more parameters are added and tuned, the performance on both sets increases. However, at a certain point, the MSE values begin to diverge. Adding more complexity and more parameters allows the MSE of the training set to be continually reduced but increases the MSE on the validation set. There is overfitting in this regime.

Overfitting is very common in ML applications. Regulation techniques, such as L1 and L2 regularization, can help to reduce overfitting [135]. Regularization terms are added to the loss

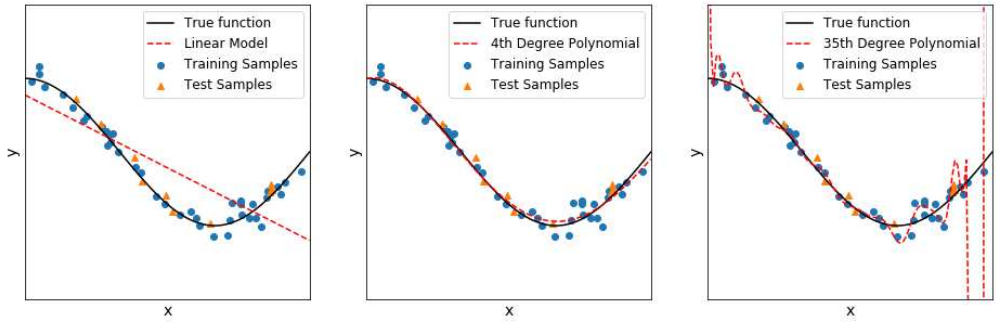

Figure 6.1: Plots of various models (red) compared to the training data (blue), test data (orange), and the true function used to generate the points (black). The comparisons are made with a linear model (left), a fourth-degree polynomial (middle), and with a thirty fifth-degree polynomial (right).

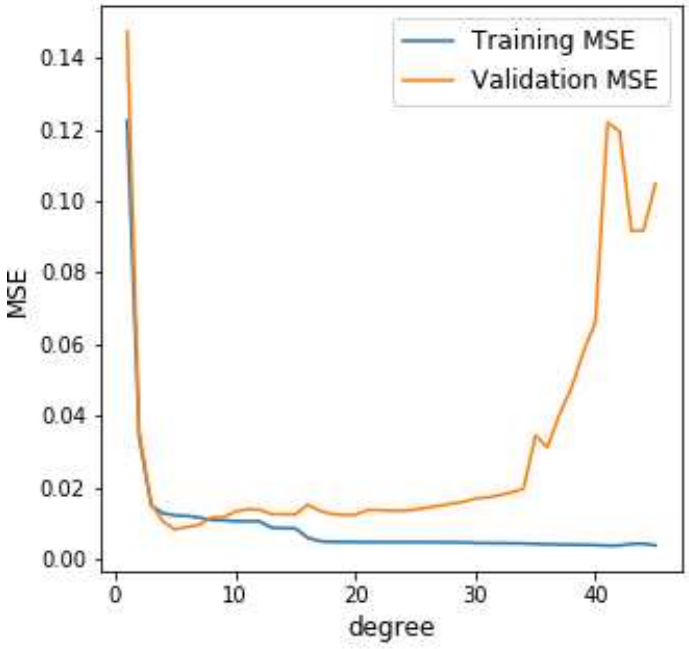

Figure 6.2: Plot of the MSE for both the training set and test set as a function of the degree of the polynomial model.

function and create a penalty for additional complexity. In the above example, each higher degree of x for the polynomial would be penalized more harshly than the previous. Effectively, this forces the model to strike a balance between performance and simplicity. This helps to produce a model which is more capable of generalizing to new data beyond what was used for training. This leads to improved performance on the validation set. More information about regularization can be found in Ref. [135].

6.3 Perceptrons

Now that the tuning of parameters to train a model has been discussed, it is necessary to discuss neural networks. An artificial neural network (ANN) is a type of machine learning algorithm loosely based on the neural networks in biological brains. These networks are currently at the forefront of ML techniques and their uses range from computer vision problems to natural language processing, reinforcement learning, and recommendation systems.

The simplest ANN that can be constructed is called a perceptron. A perceptron is a neural network with only one layer [154]. To further simplify the discussion, the layer will consist of a single neuron. Figure 6.3 is a diagram of the architecture for a perceptron with four inputs. The x_i's are the input features and the w_i's are the corresponding weights. These are then combined in a weighted sum $\sum_{i=1}^{n} w_i x_i$, where $n = 4$ in this case. The weighted sum is then passed to an activation function σ, which calculates the output and returns that to the user. The combination of the weighted sum and activation function is a neuron.

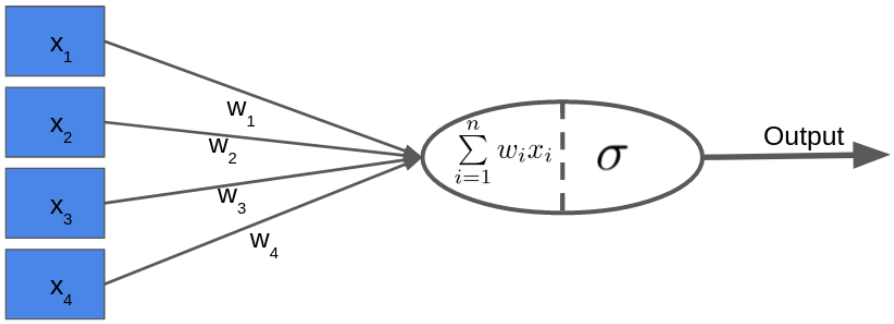

Figure 6.3: A graphical representation of a perceptron with four inputs.

Perceptrons are typically used as binary classifiers to segment data into two categories. Standard activation functions for a perceptron include a sigmoid, hyperbolic tangent, linear unit, rectified linear unit, exponential linear unit, and a step function. Either the "neuron" activates and outputs a 1, or it does not and the output is 0. This is inspired by a biological neuron, where it either fires or does not. In the years since the development of the perceptron, the neurons in ANNs have been modified in numerous ways that deviate substantially from biological neurons.

A perceptron is trained with some version of gradient descent, such as Adam. The weights are randomly assigned at the start; then, data is given to the perceptron for training. During training, the w_i values are tuned to minimize the loss function. The training is repeated until the perceptron is able to perform the desired task.

6.4 Deep Learning

Deep learning is a specialized subset of machine learning. DL focuses on the training of ANNs [135, 149]. An ANN is created by connecting many neurons together. ANNs are currently at the forefront of AI research and have been shown to be very useful in industrial applications. The "deep" in deep learning refers to the depth of the ANN, or rather the number of layers. By adding more neurons to the network, the number of tunable parameters or weights is increased. This can be done by adding more neurons to existing layers, thereby making it "wider", or by adding new layers of neurons, making it "deeper". Research has shown that adding additional layers of neurons generally leads to better performance than adding more neurons to existing layers.

Figure 6.4 shows a visual representation of a relatively small ANN. This network has five input neurons that connect to a layer of ten neurons; that layer connects to a layer of seven neurons, which in turn connects to the output layer of five neurons. The first layer is called the input layer. It takes the raw input and prepares it for analysis by the rest of the network. The final layer is called the output layer. The activation function chosen for this final layer depends

on the desired output: for example, if the final output of each neuron is to be a continuous value, a linear activation function ($y = x$) could be used. All of the layers between the input and output layers are called "hidden" layers. These do the bulk of the calculations and conduct the abstraction mentioned earlier.

The network shown in Fig. 6.4 is a relatively small network, but still has many parameters that must be tuned. It has 27 neurons and 155 edges connecting them. That means GD or some variant of it must be used to optimize 155 different parameters. The number of parameters increases as the number of layers and neurons increases. For example, an ANN that takes a 224×224 pixel grayscale image as input would have 50,176 individual elements for the input layer, and many networks used to analyze images are dozens if not hundreds of layers deep. This makes constructing and training these ANNs very difficult.

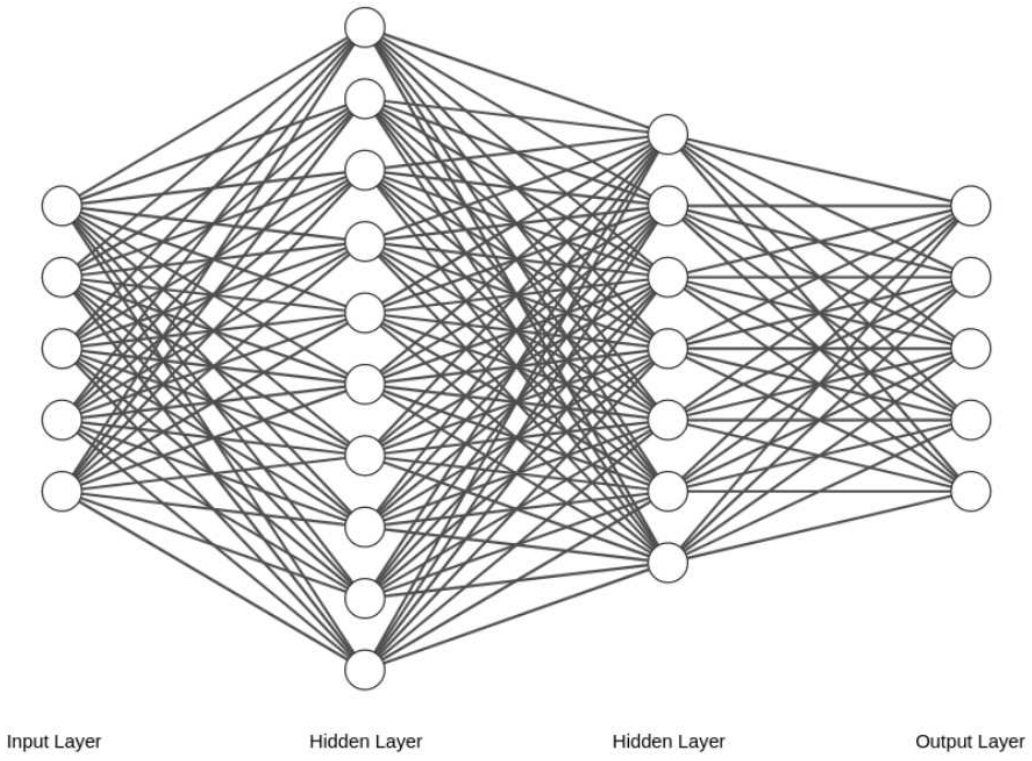

Figure 6.4: A graphical representation of a four-layer ANN. The input layer consists of 5 neurons, the first hidden layer has ten neurons, the second hidden layer has seven neurons, and the output layer has five neurons.

Each level of the network "abstracts" the input to identify features relevant to the network's specified goal. In the case of image processing, a grayscale image would be converted into a matrix where each element corresponds to the intensity of a pixel at that location. The layers in the network would abstract and "pass forward" information about the edges, bright spots, shapes, etc. More layers allow for more abstraction and an increased ability to associate patterns with the data to the desired output. However, more layers mean more tunable parameters. Thus the network is more prone to overfitting.

Deep learning models are particularly useful because they can use multiple layers to extract higher-level features from the raw data. Often a dataset will contain more information than is needed to solve the problem. When classical ML algorithms like linear regression are used, a user needs to identify the features that matter for the final prediction. This feature engineering is very time consuming and requires further analysis to determine what features to use. Therefore, feeding an ANN the raw data is generally preferred. Deep learning models also outperform classical models for large enough training sets.

It is also important to note that DL models are not always better than more traditional algorithms. Significantly larger datasets are required to train DL models. These models are much more complicated to build and to train. However, numerous frameworks and software tools have been developed to facilitate DL, such as Google's TensorFlow and Facebook's PyTorch.

Studies from academia, finance, and the tech industry are rapidly driving DL forward. This discussion is a very brief introduction, but Ian Goodfellow's *Deep Learning* [149] goes into great detail about ANNs and their implementation and is highly recommended for readers interested in understanding and utilizing DL methods.

6.4.1 Convolutional Neural Networks

Many variations of deep learning networks have been designed for different tasks. Convolutional neural networks (CNNs) are a particular subset of ANNs which make use of a linear operation called a convolution [149]. CNNs are highly effective for evenly spaced time-series

data and computer vision problems. We used a CNN to analyze images produced from simulations in Chapter 7.

Convolutions are incredibly efficient and useful operations for signal processing. In fact, S. W. Smith described it as "the single most important technique in Digital Signal Processing" in Ref. [155]. At the most basic level a convolution is a mathematical operation that creates a new function which describes the "overlap" of two input functions. The benefits and applications of convolutions are most easily described with examples. Ian Goodfellow discussed using a laser to measure the location of a spaceship [149]. I will discuss a conceptually similar example that is more relevant to the current state of the world. Namely, the hospitalization rate of COVID-19 patients.

Consider a hospital currently treating and monitoring patients infected with the COVID-19 virus. It needs supplies for the patients and the doctors, and therefore needs to accurately understand the hospitalization rate at a particular point in time, t. The hospital can keep a continuous record $I(t)$, which describes the number of hospitalized patients at a given time. However, the data will be noisy. There will be daily fluctuations, a lag in the record being updated if things get busy, and even variation based day of the week [156]. As such, the particular value of $I(t)$ will not be the best way to measure the hospitalization rate. Instead, measurements over a time interval can be used, such as a seven day average. For our example, we want to give more significance to new measurements; the virus can spread very quickly and we want to be sensitive to that. We define a weighting function $K(\tau)$, where τ is the "age" of the measurement and the value of K decreases as τ increases. The convolution of these two functions is then defined as

$$S(t) = (K * I)(t) = \int_0^t I(\tau)K(t-\tau)d\tau, \tag{6.8}$$

where $S(t)$ is the output of the convolution, in our case a weighted average at time t, and the convolution operation is denoted by $*$. The function $S(t)$ be more useful to describe the hospitalization as a function of time because it reduces the noise. Generally, I is called the input and K is called the kernel.

In most real world scenarios, and even in our COVID example, the input will not be a continuous function. Usually, data is discretized and reported at set intervals, such as a daily report of infections. A discrete convolution can be done by approximating the integral in Eq. (6.8) as a summation. This takes the form

$$S(t) = (I * K)(t) = \sum_{\tau_n=0}^{t} I(\tau_n) K(t - \tau_n), \tag{6.9}$$

where both I and K are defined at particular t values and τ_n is set to be discrete recorded values between 0 and t.

Convolutions are not restricted to a single axis. In ML and signal processing, the input data is usually stored in multidimensional arrays. These multidimensional arrays are referred to as tensors. In the COVID-19 example, the input I is a vector which counts the number of hospitalized patients at set time intervals. A grayscale image is a two-dimensional array with dimensions $M \times N$, called a matrix, in which each element of I is the pixel value of the grayscale image at a location (i, j). Because the input is a two-dimensional array, it is necessary to pick a two-dimensional kernel as well. The equation for a discrete convolution of a two-dimensional image with a two-dimensional kernel is

$$S(i, j) = (I * K)(i, j) = \sum_{m=0}^{M} \sum_{n=0}^{N} I(m, n) K(i - m, j - n). \tag{6.10}$$

By adding more dimensions, higher order tensors can be constructed and used. For example, an RGB image would be stored as a rank 3 tensor. Each channel of the image is converted into a matrix where the elements of these matrices correspond to the pixel value of that channel and that location. These matrices are then stacked and used for calculations.

A wide variety of operations can be implemented by changing the kernel. Convolutions are often used in image processing to blur images, reduce noise, and detect edges [157]. Different kernels are used for different purposes. In most ML applications, the kernel is a tensor in which the elements are tunable parameters. During training, these parameters are modified to assist

in "feature extraction". This could be identifying edges of objects in an image, determining color saturation, calculating weighted averages, or something else entirely depending on the problem and the dataset. Adding convolutions with tunable kernels to a deep neural network allows the network to learn to extract features which are useful in minimizing the loss function.

While CNNs are so-called because of convolutions, most ML libraries do not actually calculate convolutions and instead implement a different function. In a convolution, the kernel is flipped. This is seen in the $K(t-\tau)$ term in Eq. (6.8). While this is a very minor computation, flipping the kernel repeatedly adds a computational cost to training a deep learning model. Instead, the implemented function is

$$\hat{S}(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n). \tag{6.11}$$

This is called the cross-correlation function. It is the same as a convolution, except the kernel does not need to be flipped. The cross-correlation is what is calculated by the computer, but is called and discussed as convolution by convention within deep learning. Fig. 6.5 provides a visual representation of a 2D convolution of a 2x2 kernel applied on a 3x4 matrix of input data. Additional information on CNNs can be found in Ian Goodfellow's book [149].

6.5 Challenges in Effective Machine Learning Implementation

While ML algorithms have proven extremely useful, they are not trivial to implement. In this section, the key challenge to effective ML implementation is discussed.

Since ML algorithms learn to identify patterns within the data, the data itself is the first requirement for an effective ML solution. The most common problem that prevents an ML solution from working is lack of data. There must be enough data collected for the training process and that data needs to contain enough information to solve the problem. Without enough data, the training will not optimize the parameters, and the model will not perform well. Likewise, if the data does not have features that relate to the desired output, the model will never identify

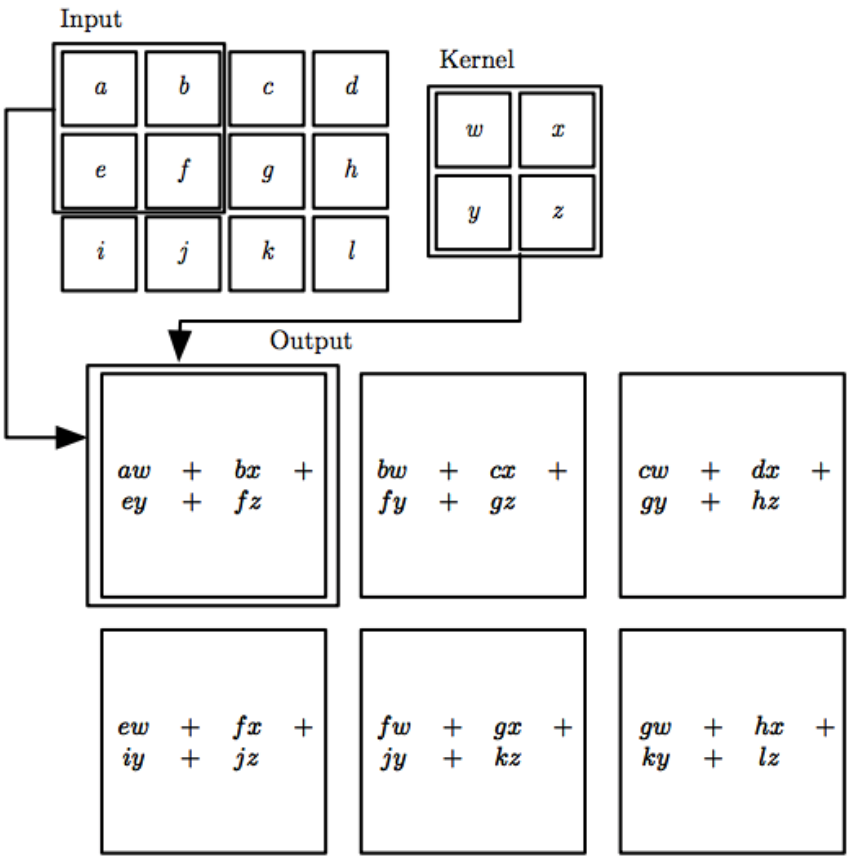

Figure 6.5: A visual representation of a 2D convolution of 3x4 matrix with a 2x2 kernel. This image originally appeared in Ref. [149].

a pattern that connects them. Additionally, the model needs good quality data. If the data has many outliers or is very noisy, optimizing the model becomes substantially more difficult and the final performance of the model will be compromised.

Another concern is that the data may not be representative. The data used for training should provide a representative sample of whatever situation is being analyzed. If the training set is biased, the model will be as well. The model can only learn patterns that exist within the training data. Effective ML implementation requires careful consideration of what data and assumptions are given to a model for training purposes.

During the training of the model, it is necessary to consider underfitting and overfitting. Since this was discussed previously, further discussion will be omitted.

Lastly, the interpretability of ML algorithms is a concern. Unfortunately, many ML algorithms are not easily understood. ANNs, in particular, are complex combinations of linear algebra, tensor operations, statistics, and topology. The computational implementation of an ANN and the calculations mentioned is also far from trivial. This raises concerns about how to explain what the network is doing and convince people that it is reasonable. Understanding the predictions of neural networks is an area of active research and the subject of numerous studies [158].

Chapter 7

Parameter Estimation via Deep Learning Methods

7.1 Introduction

Multiple studies in which the parameters in Eq. (2.3) are estimated have been conducted using a variety of techniques. Grazing-incidence small angle X-ray scattering (GISAXS) has been employed to estimate the coefficients κ_1 and κ_2 for semiconductor surfaces bombarded with noble gas ions [27, 159, 160]. More recently, GISAXS with a coherent X-ray beam rather than an incoherent one has been used to estimate the coefficients of the leading-order linear and nonlinear terms [161]. Estimates of the parameters have also been obtained for normal-incidence bombardment of silicon with iron co-deposition starting from measurements of physical attributes of the pattern like the characteristic lateral length scale and the pattern's amplitude at long times [42, 65].

The coefficients can also be determined by atomistic simulations combined with the so-called crater function formalism (CFF) [19, 21]. Norris *et al.* input the results of molecular dynamics simulations into the CFF to estimate the coefficients κ_1 and κ_2 for irradiation of a silicon target with 100 and 250 eV Ar ions [28]. More recently, Hofsäss and Bobes carried out Monte Carlo simulations using SDTrimSP and input the results into the CFF [162]. This yielded estimates for all of the parameters in Eq. (2.3) for a variety of target materials and ion species.

We propose that deep learning could be used effectively to recognize the parameters associated with the patterns produced by ion sputtering and to predict what the parameters are for a particular ion-target combination given a single AFM scan of the surface. The first steps in this direction have recently been made by Reiser [165]. However, his work was restricted to normal incidence ion bombardment, and the surface of an elemental material simply remains flat when it is bombarded with a normally incident noble gas ion beam. Reiser's method also

requires two images of the surface that are taken at times separated by a short time interval. This is impractical if the surface scans must be made *ex-situ*, as is invariably the case.

In this study, we develop an ANN that is able to estimate the five parameters in the EoM (2.3) from a single atomic force microscope (AFM) scan of a solid surface that has been bombarded with an obliquely incident ion beam. We trained the network for a range of parameters using images generated by numerical integration of the aKS equation (2.3). The parameter ranges selected are appropriate for sputtering of a silicon surface with a 1 keV argon ion beam, and the ANN was trained for fluences of 5×10^{17} ions/cm^2 and 1×10^{17} ions/cm^2. For the first fluence, our ANN was able to estimate all five parameters in the EoM with root-mean-square errors less than 3% of the parameter ranges used for training. For the second fluence, it estimated the parameters with root-mean-square errors values under 2% of the parameter training ranges. This demonstrates that our network can reliably predict the parameters and can be trained for various ion fluences.

A key benefit of the tool we have developed is that it could be used to provide a validation check on parameter estimates determined by other means, e.g., grazing-incidence small angle x-ray scattering (GISAXS) or atomistic simulations combined with the CFF. Just like estimates based on GISAXS, our tool takes experimental results and analyzes them to estimate the parameters in the EoM. A notable difference is that our method utilizes a single AFM image, and AFMs are widely available. GISAXS, on the other hand, requires the experiment to be conducted at a facility with a synchrotron x-ray beamline.

This chapter is organized as follows. A brief discussion of the ML algorithm used is included in Sec. 7.2. In Sec. 7.3, we discuss the particular architecture used for our analysis as well as the production of the dataset used for training and evaluation. A simple test of estimating a single parameter in the EoM is conducted in Sec. 7.4. In Sec. 7.5, we generalize the ANN to estimate all five parameters and present the estimates. Next, we consider related work and the previous studies of the KS equation using machine learning techniques in Sec. 7.6. Potential extensions

and considerations of the utility of the network are addressed in Sec. 7.7. Our conclusions and final thoughts are given in Sec. 7.8.

7.2 Machine learning and convolutional neural networks

We leverage the ability of ML algorithms to recognize and interpret patterns to analyze the pattern formation observed on ion-sputtered solid surfaces. Supervised ML algorithms learn to associate patterns in data with known "targets". The network developed in our study uses supervised learning and the targets are the values of κ_1, κ_2, B, λ_1, and λ_2. Since our targets κ_1, κ_2, B, λ_1, and λ_2 are continuous variables, regression methods were applied to our problem.

Numerous machine learning algorithms that can analyze data and extract meaning exist [135, 148, 149]. In our work, the algorithm we employed was an artificial neural network. We chose to utilize a particular kind of ANN — a CNN — because of their high levels of performance on computer vision problems. CNNs are adept at identifying spatial relationships and have had significant success in practical applications [149, 150] as discussed in Chap. 6

A key benefit of CNNs is that they can accept high dimensional data like an image as input, whereas many ML algorithms require the user to reduce the number of dimensions and determine useful features manually. CNNs take the raw images as input and determine which features of the image are useful in estimating the parameters during the training process. Other algorithms, such as a support vector machine, can do well in image processing but typically require the user to do some preprocessing to determine which features of the image are of interest and then translate that information into a form that the algorithm can use.

7.3 The convolutional neural network and dataset generation

Rather than building and training a network from scratch, we chose to use a Dense Convolutional Network (DenseNet) with 121 layers [167]. This architecture is well understood and has been shown to perform remarkably well for computer vision problems, making it an excellent candidate for the analysis of AFM images. An additional benefit of using this structure

for the network is that one can obtain one that is pre-trained. The DenseNet architecture was trained on the ImageNet dataset, which consists of more than 14 million images and 20,000 classes [168]. This means that the network had already learned to extract information from images, a benefit that we repurposed for our problem using transfer learning. Transfer learning takes a network that has already been trained for a similar task and repurposes it for a new problem [169]. This drastically reduces the training time needed and allows a smaller dataset to be used for the training process. We adapted the network by replacing the output layer with a new layer designed for our problem. The layers of the base model were frozen and the output layer and associated connections were trained to understand what to report for the new problem. Once the model had a good “understanding" of what the output should be, the base layers were unfrozen and the entire model was fine-tuned to optimize the results.

We used PyTorch for our analysis. PyTorch is a software tool that facilitates in the construction, training, and implementation of deep neural networks. This tool is openly available, is commonly used in deep learning research, and has a dedicated team for the support and advancement of the tool itself [170]. We also made use of the fastai library, which provides additional functionality and assists in the training process.

Our dataset was produced via numerical simulations. Equation (2.3) was numerically integrated using the fourth-order Runge-Kutta exponential time-differencing method described by Cox and Matthews [114]. Periodic boundary conditions were applied. The linear terms were calculated exactly in Fourier space and the nonlinear terms were approximated in real space by a central difference finite differencing scheme accurate to fourth order in the grid spacing. For example, the partial derivative of u with respect to x was approximated by Eq. (4.3). Each simulation was started from a low-amplitude white noise initial condition and the parameters κ_1, κ_2, λ_1, λ_2 and B were randomly selected from parameter ranges that will be specified later. The Python framework discussed in Chap. 4 was used to produce the simulations.

When the images were given to the network for training purposes, the input image is normalized so that the pixel values for each channel had the same mean and standard deviation as

the corresponding channel of the images in the ImageNet dataset. There was also a 5% probability that a given image was rotated azimuthally before it was input into the CNN. If an image was selected to be rotated, the rotation angle was randomly chosen from the range $[-5°, 5°]$. This rotation was done because normally the AFM image would be produced *ex situ* and the sample could be slightly misaligned when it is positioned in the AFM sample holder. In addition, there could be a small error in the reported direction of the ion beam.

The produced images were segmented into a training set, a validation set, and a test set. The training set was segmented into batches of 64 images. A single batch was given to the model, and the weights between neurons were updated to increase performance. Another batch was then given to the model and the weights were updated again. This process was repeated until the model had seen all of the images in the training set. The model was then asked to predict the parameter values for all the images in the validation set. By checking the performance on the validation set, we could monitor how much the model had learned and determine how useful the network was for our problem. A complete pass through every image in the training set is called an epoch. After each epoch, the model was validated by comparing the predicted and true values for the validation set. This process of segmenting the training set into batches, showing images to the model, updating weights, and validating was reiterated for 60 epochs to train the model and evaluate its performance during the training process. The model's performance was gauged by determining the MSE between the predicted values and the actual values of the parameters. The network weights were adjusted during training in accordance with the Adam optimization algorithm [151]. Rather than using a fixed learning rate to update the network, we utilized a "1 cycle policy" [171]. This method uses a variable learning rate to help prevent overfitting.

Once training was completed, the final performance of the model was evaluated. The test set had been withheld from the model until this point, which made it suitable for the final test. The model predicted the targets for every image within the test set. These predictions were then compared to the true values and the root-mean-squared error (RMSE) was calculated. The

RMSE was selected for the final evaluation because of its familiarity to physicists, whereas the MSE was used during training because it stores the same information but is simpler computationally.

7.4 Estimation of a single parameter

We began by setting all of the parameters in Eq. (2.3) to fixed values and only allowing λ_1 to vary. We then trained our ANN to predict the value of λ_1. The coefficients in Eq. (2.3) were chosen to be $\kappa_1 = \kappa_2 = -1$, $B = 1$, and $\lambda_2 = 1$, while λ_1 could vary between 0.1 and 5.1. The anisotropy present for $\lambda_1 \neq 1$ gives a tendency for the pattern to be elongated along either the x or y direction. Three examples of simulated surfaces are shown in Fig. 7.1. As λ_1 increases, so does the elongation of protrusions and depressions along the x direction. Large values of λ_1 cause significant elongation in the x direction whereas smaller values do not.

Each of the images used for training, validation and testing was taken from a simulation in which λ_1 was randomly selected from the range $0.1 \leq \lambda_1 \leq 5.1$. All of the images were taken at time $t = 10$; this time was chosen to match previous work [172]. The network was trained on 4,000 images and the validation set consisted of 1,000 images. After training was completed, the final network was evaluated on a test data set consisting of 500 images.

Training the output layer resulted in a RMSE of 0.193, which is less than 4% of the range of λ_1 values. After unfreezing the hidden layers and fine-tuning the network, the RMSE on the validation set was 0.101, i.e. 2.12% of the range. Lastly, the CNN was given the test set and asked to predict the value of λ_1 for each image. Figure 7.2 shows that the values predicted by the network agree well with the true values. The final RMSE on the test set was 0.107, which is 2.14% of the range. These results demonstrate that using a single image, the CNN can very effectively estimate the parameter λ_1 in Eq. (2.3) for fixed values of κ_1, κ_2, λ_2 and B.

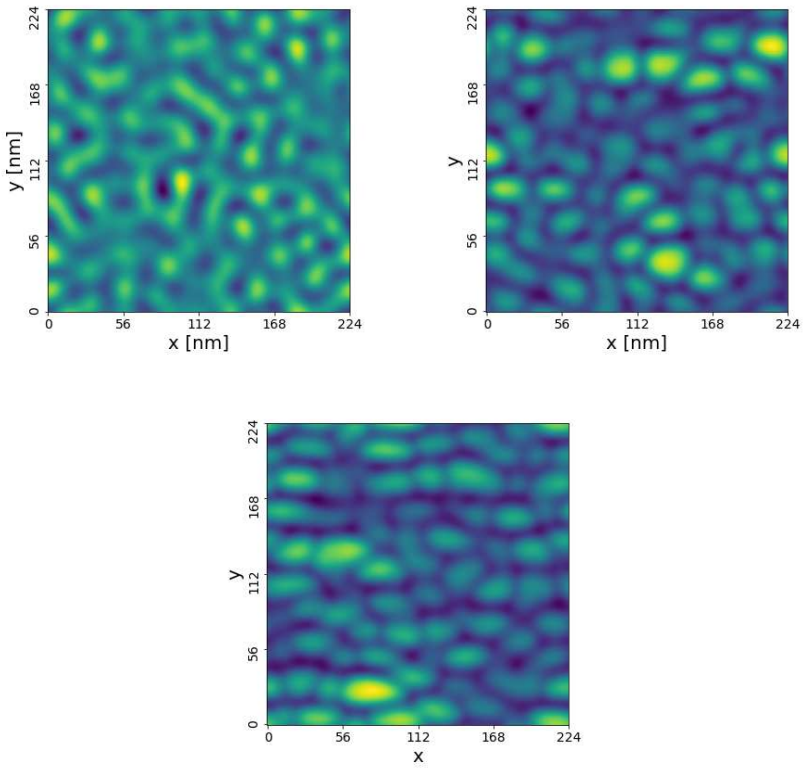

Figure 7.1: u vs x and y at $t = 10$ for $\lambda_1 = 0.10$ (left), $\lambda_1 = 2.55$ (middle), and $\lambda_1 = 4.79$ (right). The domain size is 224×224.

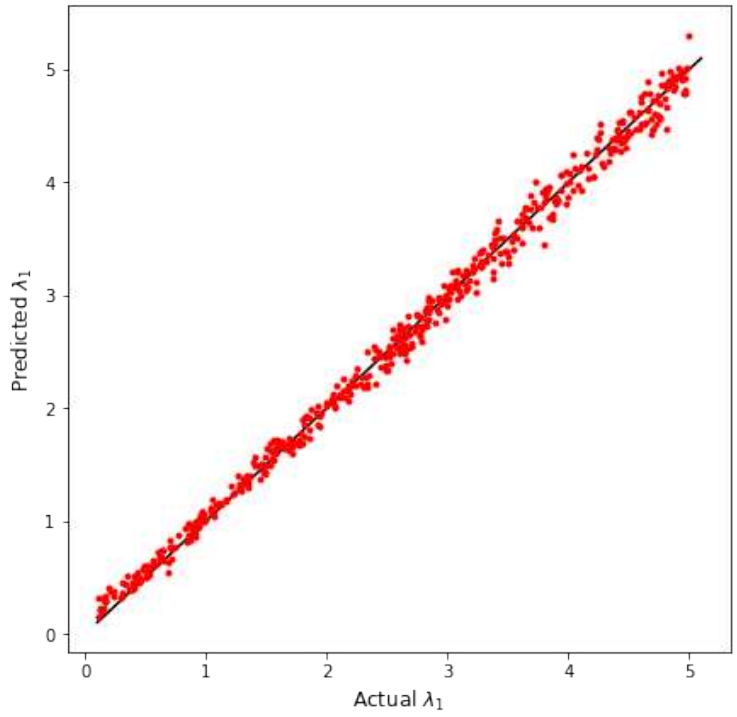

Figure 7.2: Predicted values of λ_1 vs the actual values. The red points show the actual points and the solid black line shows what perfect agreement would be. The RMSE was 0.107.

7.5 Prediction of all of the parameters

As we have seen, the model does quite well at estimating a single parameter. However, in an experiment, all of the parameters in Eq. (2.3) have unknown values. As such, it is necessary to train the model to estimate not just one parameter, but five. To achieve this, a training set of 32,000 images, a validation set of 8,000 images, and a test set of 5,000 images were generated.

For each simulation, the values of κ_1 and κ_2 were randomly selected from the range [-0.12 nm^2/s, 0.22 nm^2/s], B from the interval [0.010 nm^4/s, 3.96 nm^4/s], and λ_1 and λ_2 were each randomly selected from the interval [-1.47 nm/s, 0.72 nm/s]. These ranges were chosen to be approximately twice as wide as the parameter ranges determined in Ref. [162] for a 1 keV Ar beam incident on Si with an angle of incidence θ between 55° and 80°. (This is the range in which there is a linear instability [162].) For each simulation, we required $\kappa_1 < 0$ and/or $\kappa_2 < 0$. This ensured that a surface instability always existed.

The images were taken at a time t that corresponds to a fluence of 5×10^{17} ions/cm^2. Most sputtering experiments are run until a fluence in excess of this value is reached. The domain size of the simulations was 500 nm × 500 nm and each surface image was saved as a 224×224 pixel image. The CNN used was the same as in Sec. 7.4 with the exception of the output layer; there were five neurons in the output layer instead of a single one, which allowed the network to return values for the five parameters κ_1, κ_2, λ_1, λ_2 and B.

Figure 7.3 shows three of the images used for training. Depending on the values of the parameters, the surface could be similar to the surfaces shown in Fig. 7.1, or could display parallel or perpendicular-mode ripples. When only a single parameter is varied, the effect on the simulated surface is fairly easy to recognize, but with five free parameters, each one's impact on the surface dynamics and the observed patterns is significantly harder to identify.

Figures 7.4, 7.5, and 7.6 show the predicted parameter values vs. the actual values for the 5000 images in the test set after preliminary training and fine-tuning. Since 5000 data points on a single plot would be difficult to interpret, we opted to visualize the data via binning. Figure 7.4 shows the predicted vs actual values for κ_1 and κ_2. The results for B are shown in Fig. 7.5. The results of the nonlinear terms are shown in Fig. 7.6. The parameter space was divided into hexagonal bins that were colored according to the number of points within each bin. For each of the five parameters, we observe that the data aligns well with the cyan 1-to-1 agreement line [173]. The best agreement between the values predicted by the CNN and the actual values is seen in the coefficients of the second order linear terms: κ_1 has a RMSE of 0.00037 nm^2/s while κ_2 has a RMSE of 0.00035 nm^2/s; these errors are 0.090% and 0.097% of the range used for training, respectively. The fourth order linear term's coefficient B has a RMSE of 0.11 nm^4/s, which is 3.0% of the range. The coefficients of the nonlinear terms λ_1 and λ_2 have RMSEs of 0.045 nm/s and 0.039 nm/s, respectively. These RMSE values are 2.1% and 1.8% of the parameter ranges. We conclude that the performance of the CNN was excellent: it learned to predict the five parameters in the EoM (2.3) to within 3% of the ranges used for training.

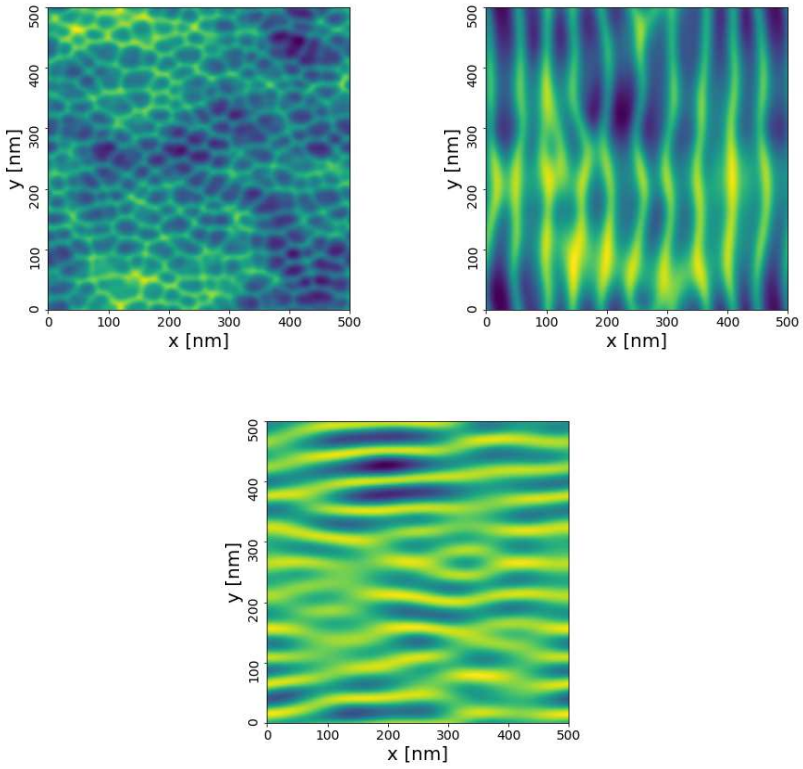

Figure 7.3: u vs x and y at a fluence of 5×10^{17} ions/cm^2 for $\kappa_1 = -0.098$ nm^2/s, $\kappa_2 = -0.094$ nm^2/s, $B = 1.169$ nm^4/s, $\lambda_1 = -1.075$ nm/s, and $\lambda_2 = -0.629$ nm/s (top left); $\kappa_1 = -0.107$ nm^2/s, $\kappa_2 = 0.128$ nm^2/s, $B = 2.893$ nm^4/s, $\lambda_1 = -0.152$ nm/s, and $\lambda_2 = 0.175$ nm/s (top right); and $\kappa_1 = 0.078$ nm^2/s, $\kappa_2 = -0.010$ nm^2/s, $B = 2.860$ nm^4/s, $\lambda_1 = 0.187$ nm/s, and $\lambda_2 = -0.035$ nm/s (bottom). The domain size is 500 nm ×500 nm.

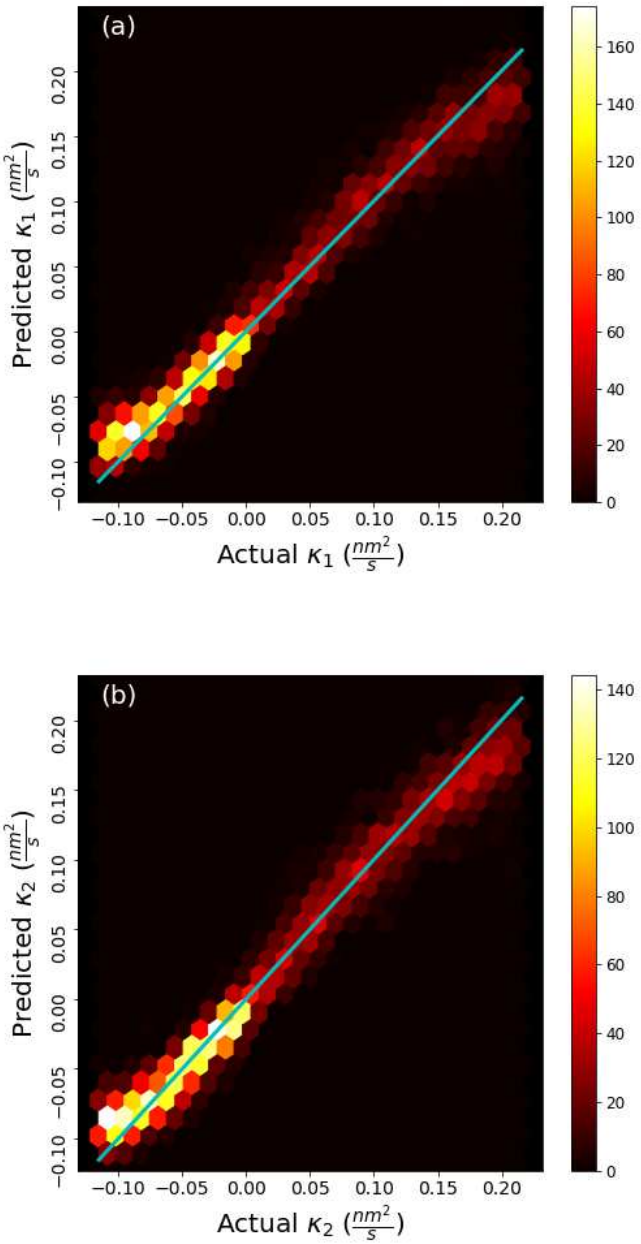

Figure 7.4: Predicted values vs. the true values of the parameters κ_1 (a) and κ_2 (b) for a fluence of 5×10^{17} ions/cm^2. The color of the hexagonal sections indicates the number of data points within that bin. The solid cyan line represents perfect agreement between the predicted and actual values. The respective RMSEs are 0.00037 nm^2/s (a) and 0.00035 nm^2/s (b).

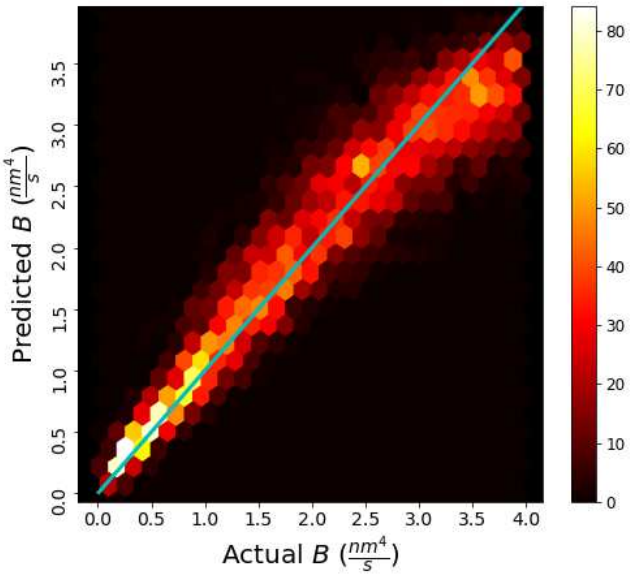

Figure 7.5: Predicted values vs. the true values of B for a fluence of 5×10^{17} ions/cm^2. The color of the hexagonal sections indicates the number of data points within that bin. The solid cyan line represents perfect agreement between the predicted and actual values. The RMSE is 0.11 nm^4/s.

The next logical question is whether the accuracy of the CNN's predictions could be further improved. This could potentially be achieved by more training on the dataset, by adding additional images to the dataset, or by increasing the number of layers within the network. Let us first consider whether more training would improve the model's performance. The sum of the fractional RMSEs of the five parameters is plotted versus the number of epochs in Fig. 7.7. The blue line shows the aggregate fractional RMSE and the dashed black line shows the point when the model was "unfrozen" and fine-tuned. The figure shows that at the start of training, the aggregate RMSE decreases quite dramatically, but later it levels off. We stopped the training at 60 epochs because we had entered a regime of diminishing returns. More training did not greatly improve the aggregate RMSE, and eventually it caused the aggregate RMSE for the validation set to increase (not shown).

Next we consider increasing the size of the dataset used for training the CNN. More data would certainly help to reduce the RMSE on the validation set: as with any analysis, more data is always desirable. However, the network was able to estimate the parameters to within 3%

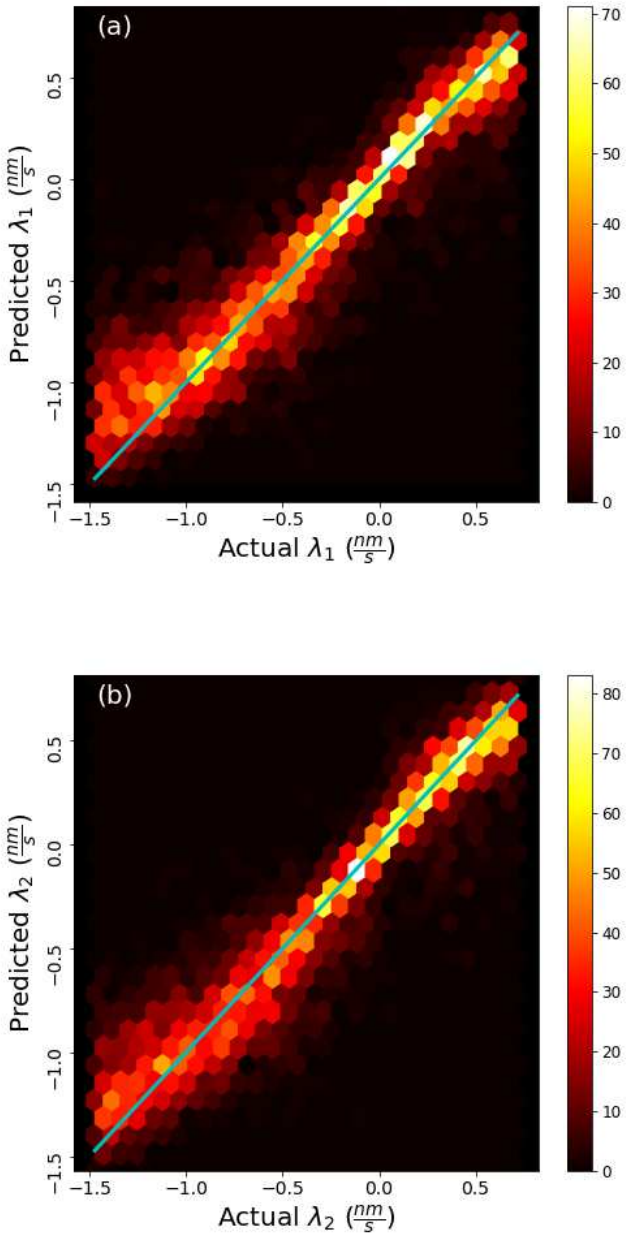

Figure 7.6: Predicted values vs. the true values of the parameters λ_1 (a), λ_2 (b) for a fluence of 5×10^{17} ions/cm^2. The color of the hexagonal sections indicates the number of data points within that bin. The solid cyan line represents perfect agreement between the predicted and actual values. The respective RMSEs are 0.045 nm/s (a) and 0.039 nm/s (b).

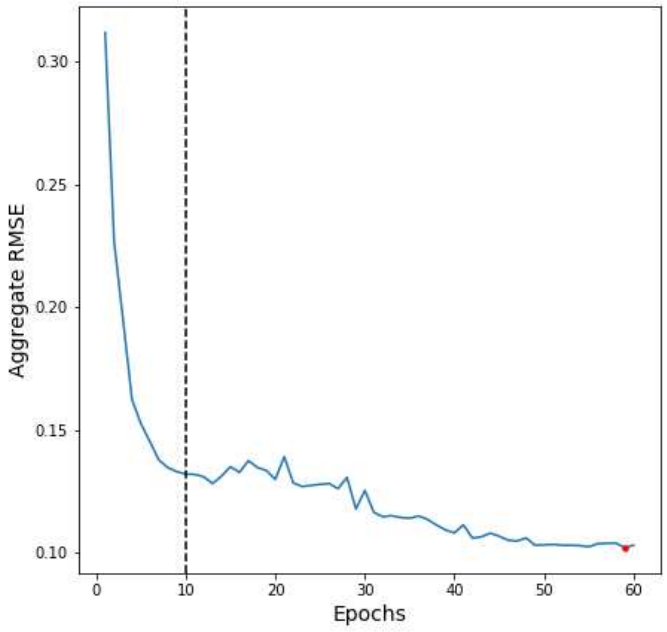

Figure 7.7: Aggregate fractional RMSE vs. epochs for the validation data. The dashed black line indicates when the model was unfrozen. The red dot indicates the minimum.

of the parameter ranges used for training. We consider this to be excellent performance, but if greater degree of accuracy were required, the dataset could be augmented by carrying out more simulations.

Lastly, we consider whether adding more layers of neurons to the network (i.e., increasing its “depth") would be beneficial. Recall the discussion of underfitting and overfitting. (A comprehensive general discussion of this topic can be found in Ref. [135].) In general, an ANN does not fully understand the patterns in the training data in the early stages of training and is "underfit." Continued training allows the network to learn more about the patterns and the RMSE on both the training and validation sets is reduced. However, eventually the model starts to recognize peculiarities of the training set that are not present in the validation or test sets. Once the model starts to make these connections, it is considered to be “overfit". Overfitting is particularly problematic when it causes a decrease of the RMSE on the training set, but increases the RMSE on the validation set.

Additional layers do give the network an increased ability to identify patterns and features within the data. This ability is what has made deep learning so successful for a wide variety of problems. However, deeper networks are not always better. At a certain point, adding more layers makes the model more likely to overfit and perform worse on test data [174]. When we compared results from DenseNet 121 (which has 121 layers of neurons) to results from DenseNet 201 (which has 201), we found that there was more overfitting in the case of the larger network. We therefore opted to use DenseNet 121.

7.6 Related Work

Considering the significant progress achieved with ML and deep learning in particular, it is unsurprising that these algorithms were eventually applied to the study of partial differential equations (PDEs). Numerous studies have been conducted in which these techniques were used both to carry out numerical integration of PDEs as well to infer the values of the parameters in a PDE used to model some aspect of the real world.

Multiple studies of the Kuramoto-Sivashinsky (KS) equation in particular have been conducted using machine learning algorithms and methods [165,172,175–179]. Most of these studies focused on the 1D KS equation, whereas, in our study, we considered the more general case in which the field u depends on both the transverse and longitudinal coordinates x and y. Raissi and Karniadakis used Gaussian processes to estimate the parameters in the one-dimensional KS equation, but required two snapshots of the field u separated by a small time interval Δt and the performance of their model decreases as Δt becomes larger [178]. In another study, Adams *et al.* implemented a support vector machine to analyze the two-dimensional aKS equation [172]. However, their study only considered variation of the single parameter λ_1. In addition, rather than looking at a range of values, the parameter was chosen to belong to one of five classes. That reduced the problem from a regression to a classification problem.

The study most closely related to ours was conducted by Reiser [165]. In his study, Reiser extended the work of Raissi and Karniadakis to the isotropic two-dimensional KS equation and

specifically discussed applying his work to ion sputtering. His results demonstrated that ML techniques can predict parameters to a significant degree of accuracy. This was the first paper to use ML techniques to estimate the parameters in the EoM for an ion-bombarded surface and it provided a proof of concept. However, there would be serious difficulties in the application of Reiser's model to real experimental results. Firstly, Reiser's work was limited to normal incidence; this simplifies the problem because it makes the EoM isotropic and reduces the number of parameters. However, the linear terms in the EoM are stabilizing at normal incidence. This means that the surface simply flattens as it is irradiated and no nanostructures emerge. The parameter estimates obtained from real experimental surface images would be very inaccurate as a consequence. In any event, the case of normal incidence bombardment of an elemental target with a noble gas ion beam is of little interest. Reiser's method also requires two consecutive snapshots of the surface separated by a short time interval Δt; this would be particularly problematic for the implementation of his method. If a sample were removed from the vacuum chamber to take the first AFM scan, it would be exposed to atmosphere, allowing deposition and/or oxidation to occur. That would change the material present on the surface and its topography and, therefore, the dynamics when the sample was returned to the chamber and irradiation was resumed. Additionally, there would be no guarantee that the second AFM scan would be taken at exactly the same location on the sample as the first. This would introduce an additional source of error.

7.7 Discussion

In this paper, we demonstrated that a CNN can be trained to predict the parameter values in the EoM (2.3) from a single AFM scan of an ion-sputtered surface. Our network therefore provides a concrete tool to analyze the nanostructures produced by ion sputtering.

The parameter ranges we chose for training our CNN are for 1 keV Ar^+ bombardment of silicon. Silicon is the most frequently studied target material, argon is the most widely used ion species, and 1 keV is a typical ion energy used in studies of ion-induced pattern formation.

However, using a different ion species or target material or changing the ion energy to another energy in the range of say 500 eV - 1500 eV would not radically change the dynamics of the bombarded surface — ripple patterns would still be observed and the wavelengths would still be of the same order of magnitude. As a consequence, the parameters in the EoM (2.3) would not change radically either. Thus, although the parameter ranges used for the training of our CNN were selected for a particular choice of target material, ion species and ion energy, it could be used to estimate the parameter values for many choices of target material, ion species and ion energy.

We trained our network for an ion fluence of 5×10^{17} ions/cm^2. Most experiments are continued until at least this fluence is reached, and it is ordinarily large enough for discernible patterns to emerge. If the parameters are to be inferred for a particular choice of ion beam and target material, an experiment with that ion beam and target material and our selected fluence could be carried out, an AFM scan could be done, and the image could be input into our CNN. If it were for some reason necessary that the parameters be estimated using a preexisting AFM scan with a different ion fluence, our CNN could be retrained for that fluence. To show that this can be done, we retrained our CNN for one fifth of the original fluence, i.e., 1×10^{17} ions/cm^2. We again generated a training set of 32,000 images, a validation set of 8,000 images, and a test set of 5,000 images. The results are shown in Fig. 7.8, Fig. 7.9, and Fig. 7.10. Once again, we observed that the predicted parameters cluster about with the 1-to-1 agreement line. The respective RMSEs of the parameters κ_1, κ_2, B, λ_1, and λ_2 were 0.00026 nm^2/s, 0.00025 nm^2/s, 0.058 nm^4/s, 0.010 nm/s, and 0.011 nm/s respectively. To put these values into perspective, the RMSEs for κ_1 and κ_2 estimations were less than 0.08% of the range used for training, the RMSE of the B predictions was under 1.5%, and the RMSE's of λ_1 and λ_2 estimates were both under 0.5%. This demonstrates that the network could be retrained for other fluences and to reliably estimate the parameters. In fact, the network performed better for the lower fluence, but since we wanted to train our CNN for typical experimental conditions, we initially selected a fluence of 5×10^{17} ions/cm^2.

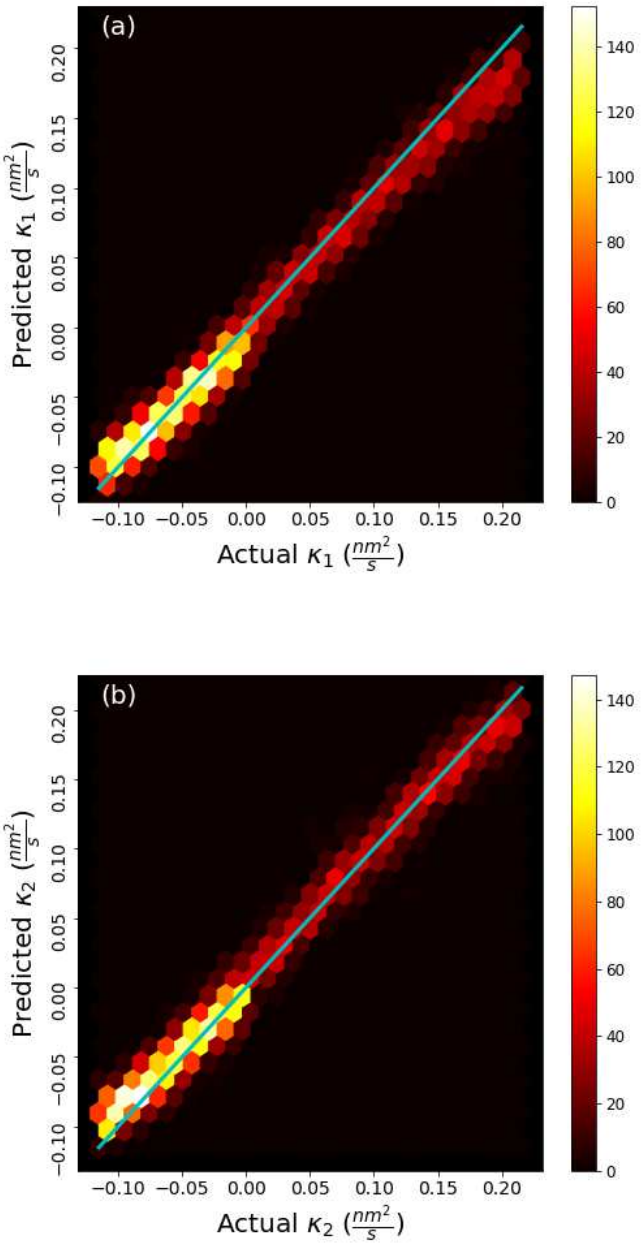

Figure 7.8: Predicted values vs. the true values of the parameters κ_1 (a) and κ_2 (b) for a fluence of 5×10^{17} ions/cm^2. The color of the hexagonal sections indicates the number of data points within that bin. The solid cyan line represents perfect agreement between the predicted and actual values. The respective RMSEs are 0.00037 nm^2/s (a) and 0.00035 nm^2/s (b).

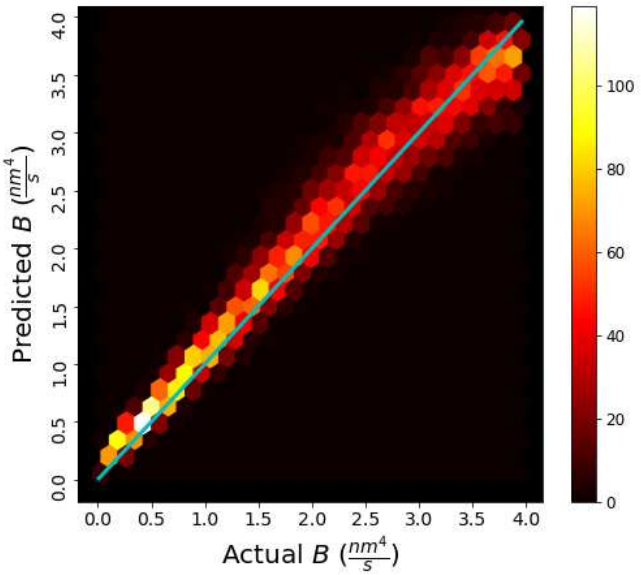

Figure 7.9: Predicted values vs. the true values of B for a fluence of 5×10^{17} ions/cm^2. The color of the hexagonal sections indicates the number of data points within that bin. The solid cyan line represents perfect agreement between the predicted and actual values. The RMSE is 0.11 nm^4/s.

For high ion fluences, additional terms that are not included in the aKS equation (2.3) may begin to play a non-negligible role in the dynamics [22, 66, 180–183]. The CKS nonlinearity $\partial_x^2 u_x^2$, for example, is likely responsible for the ripple coarsening that is often observed in experiments [66, 182, 183]. As a consequence, if only the parameters in Eq. (2.3) are to be estimated, it would be preferable to use an AFM scan in the low fluence regime in which it is safe to neglect the effect of the additional terms. If, on the other hand, the coefficients of the additional terms are to be estimated, an AFM scan in the high fluence regime would be needed. Simulations would then have to be carried out with the additional terms in the EoM. These would be used to train a CNN that had a number of neurons in the output layer equal to the number of parameters in the modified EoM. This CNN could then be used to estimate all of the parameters in the modified EoM using the AFM scan taken in the high fluence regime.

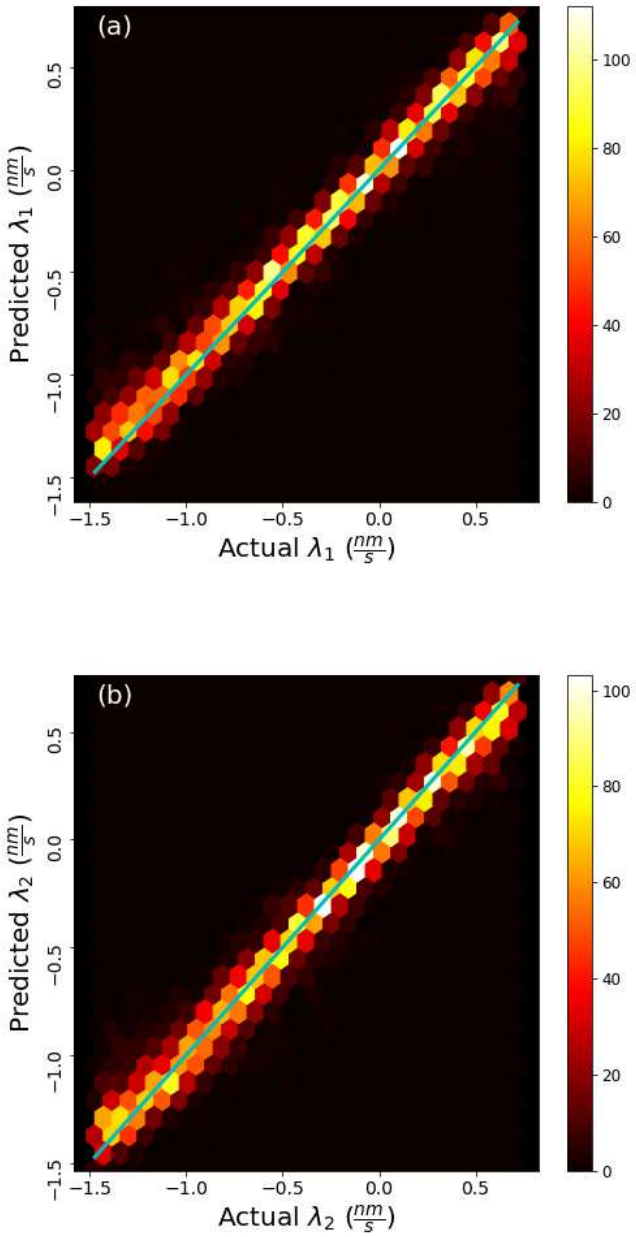

Figure 7.10: Predicted values vs. the true values of the parameters λ_1 (a) and λ_2 (b) for a fluence of 5×10^{17} ions/cm^2. The color of the hexagonal sections indicates the number of data points within that bin. The solid cyan line represents perfect agreement between the predicted and actual values. The respective RMSEs are 0.045 nm/s (a) and 0.039 nm/s (b).

7.8 Conclusions

The nanostructures produced by bombardment of the surface of an elemental material with an obliquely incident noble gas ion beam are usually modelled by the anisotropic Kuramoto-Sivashinsky equation. This equation has five parameters, each of which depend on the target material and the ion species, energy, and angle of incidence. In this paper, we developed a convolutional neural network that uses a single image of the surface to estimate all five parameters in the equation of motion with root-mean-square errors that are under 3% of the parameter ranges used for training. The network was trained and tested using thousands of images produced by numerically integrating the equation of motion, but it was developed to enable experimentalists to quickly ascertain the parameters for a given ion-sputtering experiment from a single AFM scan of the solid surface. In future work, our tool will be used to provide a check on parameter estimates determined by other means, e.g., GISAXS or atomistic simulations combined with the crater function formalism.

Chapter 8

Conclusion

The work discussed in this dissertation has expanded the generally accepted theory of IBS to include dispersion, produced a tool to estimate parameters in the EoM from a single AFM image, and provided a tool to facilitate the simulation and analysis of surfaces exposed to broad beam ion bombardment.

The developed Python module optimizes ETDRK methods for the rapid production of simulations from a user-defined PDE and IC. This tool provides a well documented and tested code base for collaborators and future graduate students to use. The module also contains numerous functions to facilitate the analysis process and hopefully accelerate future studies.

Our simulations show that dispersion can have a crucial effect on the patterns produced by oblique-incidence ion sputtering. It can lead to the formation of raised and depressed triangular regions traversed by parallel-mode ripples, and these bear a strong resemblance to nanostructures that are commonly observed in experiments. In addition, if dispersion and transverse smoothing are sufficiently strong, highly ordered ripples form. Dispersion can cause the formation of protrusions and depressions that are elongated along the projected beam direction even when there is no transverse instability. This may explain why topographies of this kind form for high angles of ion incidence in cases in which ion-induced mass redistribution is believed to dominate curvature-dependent sputtering. Dispersion can also provide an explanation for the production of perpendicular mode ripples even when there is no transverse instability.

The final study applied ML methods to estimate the parameters in the EoM from a single image. A CNN was trained to identify patterns associated with the underlying parameters in the EoM and report these parameters to the user. This provides a tool that will allow experimentalists to get parameter estimates for a particular experimental study by analyzing a single AFM image. Our neural network can be expanded and repurposed to deal with additional terms in the EoM, other fluences, and other materials and ion species.

Bibliography

[1] J. Muñoz-García, L. Vázquez, M. Castro, R. Gago, A. Redondo-Cubero, A. Moreno-Barrado and R. Cuerno, Mater. Sci. Eng. R-Rep. **86**, 1 (2014).

[2] G. Carter, M. J. Nobles, F. Paton, J. S. Williams, J. L. Whitton, Rad. Eff. **33**, 65-73 (1977).

[3] A. Keller, S. Facsko, and W. Möller, New J. of Phys. **10**, 063004 (2008).

[4] A. Keller, S. Rossbach, S. Facsko, and W. Möller. Nanotechnology **19**, 135303 (2008).

[5] A. Metya, D. Ghose, S. A. Mollick, and A. Majumdar, J. Appl. Phys. **111**, 074306 (2012).

[6] D. Chowdhury and D. Ghose, AIP Conf. Proc. **1536**, 353 (2013).

[7] M. Teichmann, J. Lorbeer, B. Ziberi, F. Frost, and B. Rauschenbach, New J. of Phys. **15**, 103029 (2013).

[8] M. Teichmann, J. Lorbeer, F. Frost, and B. Rauschenbach, Nanoscale Res. Lett. **9**, 439 (2014).

[9] D. Chowdhury and D. Ghose, Adv. Sci. Lett. **22**, 105-110 (2016).

[10] R. Gago, M. Jaafar and F. J. Palomares, J. Phys.: Condens. Matter. **30**, 264003 (2018).

[11] A. Lopez-Cazalilla, D. Chowdhury, A. Ilinov, S. Mondal, P. Barman, S. R. Bhattacharyya, D. Ghose, F. Djurabekova, K. Nordlund, and S. Norris, J. Appl. Phys. **123**, 235304 (2018).

[12] D. Kramczynski, B. Reuscher and H. Gnaser, Phys. Rev. B **89**, 205422 (2014).

[13] T. Basu, J. R. Mohanty, and T. Som, Appl. Surf. Sci. **258**, 9944 (2012).

[14] P. Mishra and D. Ghose, J. Appl. Phys. **105**, 014304 (2009).

[15] J. C. Perkinson, J. M. Swenson, A. Demasi, C. Wagenbach, K. F. Ludwig, S. A. Norris, and M. J. Aziz, J. Phys.: Condens. Matter. **30**, 294004 (2018).

[16] R. M. Bradley and J. M. E. Harper, J. Vac. Sci. Technol. A **6**, 2390 (1988).

[17] R. Cuerno and A.-L. Barabási, Phys. Rev. Lett. **74**, 4746 (1995).

[18] M. A. Makeev, R. Cuerno and, A.-L. Barabási, Nucl. Inst. Meth. Phys. Res. B **197**, 185 (2002).

[19] S. A. Norris, M. P. Brenner, and M. J. Aziz, J. Phys. Condens. Matter **21**, 224017 (2009).

[20] R. M. Bradley, Phys. Rev. B **84**, 075413 (2011).

[21] M. P. Harrison and R. M. Bradley, Phys. Rev. B **89**, 245401 (2014).

[22] D. A. Pearson and R. M. Bradley, J. Phys.: Cond. Matt. **27**, 015010 (2015).

[23] A. Gérón, *Hands-on Machine Learning with Scikit-Learn and TensorFlow Concepts, Tools, and Techniques to Build Intelligent Systems* (O'Reilly, Sebastopol, CA, 2019).

[24] R. L. Cunningham, P. Haymnn, C. Lecomte, W. J. Moore, and J.J. Trillat, J. Appl. Phys. **31**, 839 (1960).

[25] M. Navez, D. Chaperot, and C. Sella, Comptes Rendus Hebdomadaires Des Seances De L Academie Des Sciences **254**, 240 (1962).

[26] C. S. Madi, B. Davidovitch, H. B. George, S. A. Norris, M. P. Brenner, and M. J. Aziz, Phys. Rev. Lett. **101**, 246102 (2008).

[27] C. S. Madi, Benny Davidovitch, H. Bola George, Scott A. Norris, Michael P. Brenner, and Michael J. Aziz, Phys. Rev. Lett. **107**, 7 (2011).

[28] S. A. Norris, J. Samela, L. Bukonte, M. Backman, F. Djurabekova, K. Nordlund, C. S. Madi, M. P. Brenner, and M. J. Aziz, Nature Commun. **2**, 276 (2011)

[29] K. Zhang, H. Hofsäss, F. Rotter, M. Uhrmacher, C. Ronning, and J. Krauser, Surf. and Coatings Tech. **203**, 2395 (2009).

[30] A. Keller and S. Facsko, Materials **3**, 4811 (2010).

[31] M. Castro, R. Gago, L. Vázquez, J. Muñoz García, and R. Cuerno, Phys. Rev. B **86**, 12 (2012).

[32] C. S. Madi, H. Bola George, and Michael J. Aziz, J. Phys. Condens. Matter **21**, 224010 (2009).

[33] D. Chowdhury and D. Ghose, Adv. Sci. Lett. **22**, 105 (2016).

[34] G. Ozaydin, A. S. Özcan, Y. Wang, K. F. Ludwig, H. Zhou, R. L. Headrick, and D. P. Siddons, Appl. Phys. Lett. **87**, 163104 (2005).

[35] K. Zhang, M. Brötzmann, and H. Hofsäss, New J. of Phys. **13**, 013033 (2011).

[36] S. Macko, F. Frost, M. Engler, D. Hirsch, T. Höche, J. Grenzer, and T. Michely, New J. of Phys. **13**, 073017 (2011).

[37] J. A. Sánchez-arciá, L. Vázquez, R. Gago, A. Redondo-Cubero, J. M. Albella, and Z. Czigány, Nanotechnology **19**, 355306 (2008).

[38] A. Redondo-Cubero, R. Gago, F. J. Palomares, A. Mücklich, M. Vinnichenko, and L. Vázquez, Phys. Rev. B **86**, 8 (2012).

[39] B.Ziberi, F. Frost, M. Tartz, H. Neumann, and B. Rauschenbach, Thin Solid Films **459**, 106 (2004).

[40] C. S. Madi and M. J. Aziz, Appl. Surf. Sci. **258**, 4112 (2012).

[41] J. Zhou, S. Facsko, M. Lu, and W. Möller, J. Appl. Phys. **109**, 104315 (2011).

[42] J. Muñoz-García, R. Gago, L. Vázquez, J. A. Sánchez-García, and R. Cuerno, Phys. Rev. Lett. **104**, 026101 (2010).

[43] F. Ludwig, C. R. Eddy, O. Malis, and R. L. Headrick, Appl. Phys. Lett. **81**, 2770 (2002).

[44] D. Carbone, A. Biermanns, B. Ziberi, F. Frost, O. Plantevin, U. Pietsch, and T. H. Metzger, J. Phys. Condens. Matter **21**, 224007 (2009).

[45] C. S. Madi, E. Anzenberg, K. F. Ludwig Jr, and Michael J. Aziz, Phys. Rev. Lett. **106**, 066101 (2011).

[46] E. Anzenberg, Charbel S. Madi, Michael J. Aziz, and Karl F. Ludwig Jr, Phys. Rev. B **84**, 12 (2011).

[47] T. Basu and T. Som, Appl. Surf. Sci. **310**, 8 (2014).

[48] A. Keller, S. Rossbach, S. Facsko, and W. Möller, Nanotechnology **19**, 135303 (2008).

[49] A. Keller, R. Cuerno, S. Facsko, and Wolfhard Möller, Phys. Rev. B **79**, 3 (2009).

[50] T. Basu, J. R. Mohanty, and T. Som, Appl. Surf. Sci. **258**, 9944 (2012).

[51] T. Basu, D. P. Datta, and T. Som, Nanoscale Res. Lett. **8**, 289 (2013).

[52] M. Teichmann, Jan Lorbeer, Frank Frost, and Bernd Rauschenbach, Nanoscale Res. Lett. **9**, 439 (2014).

[53] W. Hauffe, Physica Status Solidi. A **35**, K93 (1976).

[54] F. Flamm, F. Frost, and D. Hirsch, Appl. Surf. Sci. **179**, 96 (2001).

[55] P. Karmakar and D. Ghose, Surf. Sci. 554, L101 (2004).

[56] D. P. Datta and T. K. Chini, Phys. Rev. B **69**, 6 (2004).

[57] D. P. Adams, T. M. Mayer, M. J. Vasile, and K. Archuleta, Appl. Surf. Sci. **252**, 2432 (2006).

[58] P. Mishra and D. Ghose, J. Appl. Phys. **105**, 014304 (2009).

[59] Q. Wei, Jie Lian, Lynn A. Boatner, L. M. Wang, and Rodney C. Ewing, Phys. Rev. B **80**, 085413 (2009).

[60] Tapas Kumar Chini, Debi Prasad Datta, and Satya Ranjan Bhattacharyya, J. Phys. Condens. Matt. **21**, 224004 (2009).

[61] J. Völlner, B. Ziberi, F. Frost, and B. Rauschenbach, J. Appl. Phys. **109**, 043501 (2011).

[62] A. Metya, D. Ghose, S. A. Mollick, and A. Majumdar, J. Appl. Phys. **111**, 074306 (2012).

[63] Martin Engler, Sven Macko, Frank Frost, and Thomas Michely, Phys. Rev. B **89**, 245412 (2014).

[64] D. A. Pearson, Matthew P. Harrison, and R. M. Bradley, Phys. Rev. E **96**, 032804 (2017).

[65] J. Muñoz-García, R. Gago, R. Cuerno, J. A. Sánchez-García, A. Redondo-Cubero, M. Castro, and L. Vázquez, J. Phys. Condens. Matter **24**, 375302 (2012).

[66] J. Muñoz-García, M. Castro, and R. Cuerno, Phys. Rev. Lett. **96**, 086101 (2006).

[67] J. Muñoz-García, R. Cuerno, and M. Castro, Phys. Rev. E. Stat. Nonlin. Soft Matter Phys. **74**, 050103 (2006).

[68] S. K. Theiss, M. J. Caturla, M. D. Johnson, J. Zhu, T. Lenosky, B. Sadigh, and T. Diaz De La Rubia, Thin Solid Films **365**, 219 (2000).

[69] A. Mutzke, R. Schneider, W. Eckstein, and R. Dohmen, Sdtrimsp: Version 5.00. IPP, Report,(12/8), 2011.

[70] P. Sigmund, J. Mater. Sci. **8**, 1545 (1973).

[71] R. M. Bradley and H. Hofsäss, J. Appl. Phys. **116**, 234304 (2014)

[72] Y. Rosandi and H. M. Urbassek, Phys. Rev. B **85**, 155430 (2012).

[73] G. Hobler, R. M. Bradley, and H. M. Urbassek, Phys. Rev. B **93**, 205443 (2016).

[74] S. Habenicht, W. Bolse, K. P. Lieb, K. Riemann, and U. Geyer, Phys. Rev. B **60**, R2200 (1999).

[75] M. A. Makeev and A. Barabási, Appl. Phys. Lett. **71**, 2800 (1997).

[76] J. Muñoz-García, R. Cuerno, and M. Castro, Phys. Rev. B **78**, 205408 (2008).

[77] B. Kahng, H. Jeong, and A.-L. . Barabási, Appl. Phys. Lett. **78**, 805 (2001).

[78] M. Castro, R. Cuerno, L. Vázquez, and R. Gago, Phys. Rev. Lett. **94**, 016102 (2005).

[79] G. Ozaydin, A. S. Özcan, Y. Wang, K. F. Ludwig, H. Zhou, and R. L. Headrick, Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms **264**, 47 (2007).

[80] A. Keller, M. Nicoli, S. Facsko, and R. Cuerno, Phys. Rev. E Stat. Nonlin. Soft Matter Phys. **84**, 015202 (2011).

[81] G. Carter, V. Vishnyakov, and M. J. Nobes, Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms, **115**, 440 (1996).

[82] V. B. Shenoy, W. L. Chan, and E. Chason, Phys. Rev. Lett. **98**, 256101 (2007).

[83] M. Moseler, P. Gumbsch, C. Casiraghi, A. C. Ferrari, and J. Robertson, Science **309**, 1545 (2005).

[84] B. Davidovitch, M. J. Aziz, and M. P. Brenner, Phys. Rev. B **76**, 205420 (2007).

[85] R. M. Bradley and H. Hofsäss, J. Appl. Phys. **120**, 074302 (2016).

[86] H. Hofsäss, K. Zhang, and O. Bobes, J. Appl. Phys. **120**, 135308 (2016).

[87] C. C. Umbach, R. L. Headrick, and K.-C. Chang, Phys. Rev. Lett. **87**, 246104 (2001).

[88] N. Kalyanasundaram, M. Ghazisaeidi, J. B. Freund, and H. T. Johnson, Appl. Phys. Lett. **92**, 131909 (2008).

[89] H. Hofsäss, O. Bobes, and K. Zhang, *Appl. of Accel. in Res. and Ind.: Twenty-Second Int. Conf.*, 1525 (2013).

[90] P. D. Shipman and R. M. Bradley, Phys. Rev. B **84**, 085420 (2011).

[91] R. M. Bradley and P. D. Shipman, Phys. Rev. Lett. **105**, 145501 (2010).

[92] R. M. Bradley and P. D. Shipman, Appl. Surf. Sci. **258**, 4161 (2012).

[93] S. Facsko, T. Dekorsy, C. Koerdt, C. Trappe, H. Kurz, A. Vogt, and H. L. Hartnagel, Science **285**, 1551 (1999).

[94] F. Frost, A. Schindler, and F. Bigl, Phys. Rev. Lett. **85**, 4116 (2000).

[95] D. Kumar, A. Gupta, Appl. Phys. Lett. **98** 123111 (2011).

[96] M. Körner, K. Lenz, M. O. Liedke, T. Strache, A. Mücklich, A. Keller, S. Facsko, and J. Fassbender, Phys. Rev. B **80**, 214401 (2009).

[97] J. Fassbender, T. Strache, M. O. Liedke, D. Markó, S. Wontz, K. Lenz, A. Keller, S. Facsko, I. Mönch, and J. McCord, N. J. of Phys. **11**, (2009).

[98] M. Liedke, M. Körner, K. Lenz, M. Fritzsche, M. Ranjan, A. Keller, E. FIX, S. Zvyagin, S. Facsko, K. Potzger, et al. Phys. Rev. B **87**, 024424 (2013).

[99] M. Ranjan, S. Facsko, Nanotechnology **23**, 485307 (2012).

[100] T.W. Oates, A. Keller, S. Facsko, A. FIX, Plasmonics **2**, 47 (2007).

[101] M. Ranjan, S. Facsko, M. Fritzsche, S. Mukherjee, Microelectron. Eng. **102**, 44 (2013).

[102] J. Sommerfeld, J. Richter, R. Niepelt, S. Kosan, T.F. Keller, K.D. Jandt, C. Ronning, Biointerphases **7**, 1 (2012).

[103] A calculation of the local truncation error of the Euler method can be found at http://www.math.unl.edu/ gledder1/Math447/EulerError.

[104] Another common RK2 scheme is the midpoint method.

[105] D. F. Griffiths and D. J. Higham, *Numerical Methods for Ordinary Differential Equations* (Springer, London, England, 2010).

[106] A calculation of the local truncation error of the RK4 method can be found at https://math.okstate.edu/people/binegar/4233/4233-l18.pdf.

[107] P. J. Davis and P. Rabinowitz, *Methods of Numerical Integration* (Dover, Mineola, 2007).

[108] A discussion of numerically integrating PDEs can be found at http://www.math.umd.edu/ dlevy/classes/amsc661/hyperbolic-fd.pdf.

[109] D. Terzopoulos and M. Vasilescu, Proceedings. 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition.

[110] A. Constantinides, "Finite Difference Methods" in *Applied Numerical Methods with Personal Computers*, (McGraw-Hill, New York, 1988), pg. 299.

[111] P. V. D. Houwen, Applied Numerical Mathematics **20**, 261 (1996).

[112] D. J. Higham and L. N. Trefethen, BIT **33**, 285 (1993).

[113] Higher order finite difference schemes can be determined via the tool found at http://web.media.mit.edu/ crtaylor/calculator.html.

[114] S. M. Cox and P.C. Matthews, J. Comput. Phys. **176**, 430 (2002).

[115] A disscusion of converting linear operators to matrices can be found at http://www.physics.miami.edu/ nearing/mathmethods/operators.pdf.

[116] A.K. Kassam and L. N. Trefethen, SIAM J. Sci. Comp. **26**, 1214 (2005).

[117] S. A. Norris, Phys. Rev. B **85**, 155325 (2012).

[118] S. A. Norris, Phys. Rev. B **86**, 235405 (2012).

[119] A. Moreno-Barrado, M. Castro, R. Gago, L. Vázquez, J. Muñoz-García, A. Redondo-Cubero, B. Galiana, C. Ballesteros, and R. Cuerno, Phys. Rev. B 91, 155303 (2015).

[120] T. Kawahara, Phys. Rev. Lett. **51**, 381 (1983).

[121] M. Sato and M. Uwaha, Europhys. Lett. **32**, 639 (1995).

[122] C. Misbah and O. Pierre-Louis, Phys. Rev. E **53**, R4318 (1996).

[123] D. P. Adams, M. J. Vasile, T. M. Mayer, and V. C. Hodges, J. Vac. Sci. Technol. B **21**, 2334 (2003).

[124] J. Grenzer, A. Biermanns, A. Mücklich, S. A. Grigorian, and U. Pietsch, Phys. Status Solidi A **206**, 1731 (2009).

[125] M. P. Harrison, D. A. Pearson, and R. M. Bradley, Phys. Rev. E **96**, 032804 (2017).

[126] K. Indireshkumar and A. L. Frenkel, Phys. Rev. E **55**, 1174 (1997).

[127] V. E. Zakharov and E. A. Kuznetsov, Zh. Eksp. Teor. Fiz. **66**, 594 (1974) [Sov. Phys. JETP **39**, 285 (1974)].

[128] E.W. Laedke and K.H. Spatschek, J. Plasma Phys. **28**, 469 (1982).

[129] E. Infeld, J. Plasma Phys. **33**, 171 (1985).

[130] E. Infeld and P. Frycz, J. Plasma Phys. **37**, 97 (1987).

[131] E. Infeld and P. Frycz, J. Plasma Phys. **41**, 441 (1989).

[132] M.A. Allen, G. Rowlands, J. Plasma Phys. **50**, 431 (1993).

[133] P. Frycz and E. Infeld, Phys. Rev. Lett. **63**, 384 (1989).

[134] S. Saprykin, E. A. Demekhin and S. Kalliadasis, Phys. Rev. Lett. **63**, 224101 (2005).

[135] A. Gérón, *Hands-on Machine Learning with Scikit-Learn and TensorFlow Concepts, Tools, and Techniques to Build Intelligent Systems* (O'Reilly, Sebastopol, CA, 2019).

[136] A. I. Maqueda, A. Loquercio, G. Gallego, N. Garcia, and D. Scaramuzza, *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 5419 (2018).

[137] Y. Wen, K. Zhang, Z. Li, and Y. Qiao, *Comp. Vis. – ECCV 2016 Lec. Notes Comp. Sci.* **499**, 499 (2016).

[138] T. Young, D. Hazarika, S. Poria, and E. Cambria, *IEEE Comp. Int. Mag.* **13**, 55 (2018).

[139] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. V. D. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, Nature **529**, 484 (2016).

[140] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, Science **362**, 1140 (2018).

[141] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. Mckinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver, Nature **575**, 350 (2019).

[142] M. Cotescu, T. Drugman, G. Huybrechts, J. Lorenzo-Trueba, and A. Moinet, IEEE Signal Processing Letters **27**, 186 (2020).

[143] B. Li, T. N. Sainath, A. Narayanan, J. Caroselli, M. Bacchiani, A. Misra, I. Shafran, H. Sak, G. Pundak, K. Chin, K. C. Sim, R. J. Weiss, K. W. Wilson, E. Variani, C. Kim, O. Siohan, M. Weintraub, E. Mcdermott, R. Rose, and M. Shannon, Interspeech 2017 (2017).

[144] V. Talpaert, I. Sobh, B. Kiran, P. Mannion, S. Yogamani, A. El-Sallab, and P. Perez, Proceedings of the 14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (2019).

[145] H. Steck, M. Dimakopoulou, N. Riabov, and T. Jebara, Proceedings of the 13th International Conference on Web Search and Data Mining (2020).

[146] A. Jobin, M. Ienca, and E. Vayena, Nature Machine Intelligence **1**, 389 (2019).

[147] K. Siau and W. Wang, Journal of Database Management **31**, 74 (2020).

[148] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning: with Applications in R* (Springer, New York, 2017).

[149] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (The MIT Press, Cambridge, MA, 2017).

[150] A. Dhillon and G. Verma, Prog. Art. Int. **9**, 85 (2020).

[151] D. P. Kingma, J. Ba, *ICLR Proc. of the 3rd Int. Conf. on Learn. Rep.* **112** (2014).

[152] J. Duchi, E. Hazan, Y. Singer, Jour. Mach. Learn Res. **12**, 2121 - 2159 (2011).

[153] Interestingly, RMSProp was not introduced in a paper. Instead G. Hinton, N. Srivastava, and K. Swersky presented the optimization method in a Coursera lecture. The lecture notes can be found at https://www.cs.toronto.edu/ tijmen/csc321/slides/lecture_slides_lec6.pdf.

[154] F. Rosenblatt, Psychological Review **65**, 386 (1958).

[155] S. W. Smith, *The Scientist and Engineer's Guide to Digital Signal Processing* (California Technical Pub., San Diego, CA, 1997).

[156] W. T. Cecil, Am. J. Manag. Care **26**, 7 (2020).

[157] Examples of convolutions of grayscale images and the associated kernels can be found at https://aishack.in/tutorials/image-convolution-examples/.

[158] F. Fan, J. Xiong, G. Wang, ArXiv.2001.02522 (2020).

[159] E. Anzenberg, J. C. Perkinson, C. S. Madi, M. J. Aziz, and K. F. Ludwig, Phys. Rev. B **86**, 245412 (2012).

[160] S. A. Norris, J. C. Perkinson, M. Mokhtarzadeh, E. Anzenberg, M. J. Aziz, and K. F. Ludwig, Sci. Rep. **7**, 2016 (2017).

[161] M. Mokhtarzadeh, J. G. Ulbrandt, P. Myint, S. Narayanan, R. L. Headrick, and K. F. Ludwig, Phys. Rev. B **99**, 165429 (2019).

[162] H. Hofsäss and O. Bobes, Appl. Phys. Rev. **6**, 021307 (2019).

[163] L. Deng, APSIPA Trans. Sig. Inf. Proc. **3**, 1 (2014).

[164] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis, *Comp. Int. Neurosci. 2018*, 1 (2018).

[165] D. Reiser, Phys. Rev. E **100**, 033312 (2019).

[166] Tutorials on implementing these algorithms can be found at https://scikit-learn.org/stable/tutorial/index.html, https://pytorch.org/tutorials/, and https://www.tensorflow.org/tutorials.

[167] G. Huang, Z. Liu, L. V. D. Maaten, and K. Q. Weinberger, *2017 IEEE Conf. Comp. Vis. Patt. Recog. (CVPR)* 4700, (2017).

[168] The ImageNet dataset can be found at http://www.image-net.org/.

[169] H.-C. Shin, H. R. Roth, M. Gao, L. Lu, Z. Xu, I. Nogues, J. Yao, D. Mollura, and R. M. Summers, *IEEE Trans. Med. Im.* **35**, 1285 (2016).

[170] We used the PyTorch framework which is documented at https://pytorch.org/. The fast.ai library was also used to facilitate calculations. This library can be found at https://www.fast.ai/.

[171] L. N. Smith, ArXiv.1803.09820.

[172] H. Adams, T. Emerson, M. Kirby, R. Neville, C. Peterson, P. Shipman, S. Chepushtanova, E. Hanson, F. Motta, and L. Ziegelmeier, Jour. Mach. Learn Res. **18**, 1 - 35 (2017).

[173] One feature that is worth mentioning is the "hot spots" present for the negative values of κ_1 and κ_2. These appear because the majority of our data lies in these regions.

[174] Y. Bengio in Lecture Notes in Computer Science, vol 7700, 2012. edited by M. Grègoire, G. Orr, and M. Klaus-Robert, p. 437.

[175] M. Wang, H.-X. Li, X. Chen, and Y. Chen, *IEEE Trans. Sys., Man, and Cybernetics: Sys.* **46**, 1664 (2016).

[176] J. Pathak, Z. Lu, B. R. Hunt, M. Girvan, and E. Ott, Chaos: An Interdisciplinary J. Nonlin. Sci. **27**, 121102 (2017).

[177] J. Pathak, A. Wikner, R. Fussell, S. Chandra, B. R. Hunt, M. Girvan, and E. Ott, Chaos: Interdisc. J. Nonlin. Sci. **28**, 041101 (2018).

[178] M. Raissi and G. E. Karniadakis, J. Comp. Phys. **357**, 125 (2018).

[179] J. Pathak, B. Hunt, M. Girvan, Z. Lu, and E. Ott, Phys. Rev. Lett. **120**, 024102 (2018).

[180] K. M. Loew and R. M. Bradley, Phys. Rev. E **100**, 012801 (2019).

[181] M. P. Harrison, D. A. Pearson, and R. M. Bradley, Phys. Rev. E **96**, 032804 (2017).

[182] M. Castro, R. Cuerno, L. Vazquez, and R. Gago, Phys. Rev. Lett. **94**, 016102 (2005).

[183] J. Muñoz-García, R. Cuerno and M. Castro, Phys. Rev. B **78**, 205408 (2008).

Appendix A

Framework Example

Below is an example of the Python framework being applied to numerically integrate Eq. (2.3). This example is directly imported from Jupyter notebooks and is the file that a new user can modify to use the framework. The notebook begins below the horizontal line below this paragraph. Comments guiding user are in plain text while the code is displayed using the Python language listing package. Author: Kevin M. Loew

Contact: kevin.m.loew@gmail.com

Last Modified: April 19, 2019

Description: This notebook displays an example of the 2D simulation of a surface which evolves according to the KS equation. The notebook is designed to act as a template for the production of surface simulations. The steps are as follows:

1. Define Linear Part
2. Define Nonlinear Function
3. Set parameters for the equation of motion
4. Set domain size and spatial resolution
5. Set time range and step size
6. Generate Initial Condition
7. Execute integration

Equation of Motion:

$$u_t = \kappa_1 u_{xx} + \kappa_2 u_{yy} - B\nabla^2\nabla^2 u + \lambda_1 u_x^2 + \lambda_2 u_y^2$$

Write as:

$$u_t = Lu + N(u,t)$$

Import Libraries

```
import numpy as np
import matplotlib.pyplot as plt
import BradleyTheoryGroup as btg
```

1. Define Linear Part:

We do this in Fourier Space so we can handle the derivative as an operator.

NOTE: For users that are not comfortable writing Python functions, the linear part can be written as a string. An example is included, but commented out with the # character.

```
#Lhat='u_xx+u_yy-DDu'

def Lhat(kx,ky,Kappa1,Kappa2,B):
    Kx,Ky=np.meshgrid(kx,ky)
    return Kappa1*(1j*Kx)**2+Kappa2*(1j*Ky)**2
           -B*((1j*Kx)**4+(1j*Ky)**4+2*((1j*Kx)**2)*((1j*Ky)**2))
```

2. Nonlinear Function

For the nonlinear part we generate a function in real space (the conversion to and from Fourier-Space is handled by the btg module).

NOTE: Once again the nonlinear function can be written as a string. This is currently commented out with #.

```
#Nhat='u_x^2+u_y^2'

def Nhat(u,kx,ky,dx,dy,Lambda1,Lambda2):
```

```
    out=Lambda1*btg.diff(u,dx,axis='x',deg=1)**2
        +Lambda2*btg.diff(u,dy,axis='y',deg=1)**2

    return out
```

3. Define Parameters of the EoM.

Kappa1=−1

Kappa2=1

B=1

Lambda1=1

Lambda2=1

LinParams=[Kappa1,Kappa2,B]

NonLinParams=[Lambda1,Lambda2]

4. Define the domain size and spatial resolution.

Nx=number of points in the x direction, Ny=number of points in the y direction, xRange=(x_min, x_max), $yRange = (y_min, y_max)$.

1D NOTE: Set Ny=1 to do 1D simulations.

Nx=400

Ny=400

xRange=(0,200)

yRange=(0,200)

5. Define the start and stop time and the time step tRange=(t_min, t_max) $and h = timestep$.

1D NOTE: You can define u0 as either a 1 dimensional numpy array or as a 2 dimensional array of shape (1,Nx) and the btg module will handle it.

```
u0=0.001*np.random.random((Ny,Nx))
```

7. Execute Loop

The BradleyTheoryGroup timeIntegrator is a default loop that the user defined EoM, parameters, domain, and time range. It will use exponential time differencing to time evolve the initial surface according to the equation of motion.

Additional arguments are:

imRate: How often should an image/file be produced. defaults to 1

saveLocation: directory for the program to produce and save files into. defaults to 'temp'

Method: defines which ETD method to use. defaults to ETD4RK

doPlots: determines if plots should be produced. defaults to True

doFiles: determines if the surface should be saved as a .npy file. defaults to True

doFFTPlots: determines if the simulation should also produce FFT plots. defaults to False

1D NOTE: When doing 1D simulations it will produce plots instead of contours automatically.

```
btg.timeIntegrator(u0,*tRange,h,*xRange,*yRange,Nx,Ny,
                   LinParams,NonLinParams,Lhat,Nhat,
                   imRate=1,saveLocation='Test',Method='ETD4RK',
                   doPlots=False,doFiles=True)
```

Integration Method: ETD4RK

Contour Integration

Complete

Generating Initial Plots...

Initializing Integration...

[**********]99.99%

Simulation Complete

```
btg.MakeMovie(saveLocation='Test',fps=15)
```

Making movie...

Movie made: Test/u/u_movie.mp4