Modeling the Spatio-Temporal Dynamics of Interacting Fish Species in the Northeast Continental Shelf

---

2018 WPI REU Program in Industrial Mathematics and Statistics

WPI REU 2018

Sara Amato, Assumption College

Lauren Moore, University of Kentucky

Katie Ragosta, Boston University

Shelby Stowe, Sterling College

Date: July 28, 2018

Approved:

Dr. Andrea Arnold, Advisor

Dr. Burt Tilley, Advisor

Abstract

Knowledge of the population dynamics of marine species is vital to understanding ocean sustainability. This project aims to develop and analyze spatio-temporal single-species and multi-species models for studying fish population dynamics in the Northeast Continental Shelf, specifically Atlantic cod and Atlantic herring. We formulate partial differential equation models and integrodifference models that take into account species interactions between Atlantic cod and Atlantic herring. We determine a method to compare our single-species and multi-species models, to provide information to the Northeast Fisheries Science Center on whether either species would benefit from being assessed with a multi-species model. All models consider species' behavior, including seasonal migrations. We employ statistical approaches such as nonlinear filtering to estimate model parameters and quantify uncertainty in model predictions, comparing the results to synthetic data.

Acknowledgements

We would like to acknowledge the support from the National Science Foundation under grant DMS-1757685 for the 2018 Worcester Polytechnic Institute Research Experience for Undergraduates program in Industrial Mathematics and Statistics, as well as the support from the Center for Industrial Mathematics and Statistics at WPI. We would also like to acknowledge our industrial sponsor, the National Oceanic and Atmospheric Administration, particular the Northeast Fisheries Science Center, as well as Dr. Sarah Gaichas for being our industrial liason. Finally, we would like to thank the Department of Mathematical Sciences at WPI, as well as our two advisors, Dr. Andrea Arnold and Dr. Burt Tilley.

Contents

1 Introduction **1**

1.1 Motivation 1

1.2 Data Provided 3

1.3 Fish Overview 5

1.3.1 Atlantic Herring 5

1.3.2 Atlantic Cod 6

2 Partial Differential Equation Models **8**

2.1 Background 8

2.1.1 Logistic Growth Model 9

2.1.2 Lotka-Volterra Predator-Prey Model 9

2.2 Derivation 11

2.2.1 Overview 11

2.2.2 Conservation Law and Flux Definition 12

2.3 Single-species Model 15

2.3.1 Velocity Function 16

2.3.2 Birth Rate Function 18

2.4 Multi-species Model 21

2.5 Numerical Methods 23

3 Integrodifference Models **28**

3.1 Background 28

3.2 Single-species Models 28

3.2.1 Spatial Shift Function 30

3.2.2 Single-species Model 31

3.3 Multi-species Model 33

4 Parameter Estimation **35**

4.1 Motivation 35

4.2 Inverse Problem 36

4.3 Ensemble Kalman Filter 36

4.4 Results 38

4.4.1 PDE: Single-species 39

4.4.2 PDE: Multi-species 42

4.4.3 Integrodifference: Single-species 44

4.4.4 Integrodifference: Multi-species 47

5 Model Comparison **51**

5.1 Comparison Method 51

5.2 PDE Comparison 52

5.3 Integrodifference Comparison 53

6 Discussion **56**

6.1 Summary 56

6.2 Interpretation of Results 57

6.3 Future Work 57

References **59**

Appendix **63**

A Matlab Code **63**

A.1 Partial Differential Equation Model Codes 63

A.2 Integrodifference Model Codes 81

A.3 Ensemble Kalman Filter Codes 99

A.4 Model Comparison Code 151

Chapter 1

Introduction

1.1 Motivation

In an ever-growing world, ensuring sustainability is of the utmost concern. One industry concerned with sustainability is the fishing industry. The Food and Agriculture Organization (FAO) of the United Nations monitors the state of world fisheries. According to the FAO, from 1990 to 2007, approximately one-quarter of fish stocks were overexploited, depleted, or in a state of recovering from depletion. This accounts for 17% of fish stocks in the Northwest and Northeast Atlantic, which is the primary area of concern in this project [1].

Historical examples demonstrate the importance of using accurate population models to ensure that fishing is kept at a sustainable level. For instance, Atlantic cod, now classified as depleted, were once an abundant species in the North Atlantic [2]. In 1852, the biomass of Atlantic cod was around $1,260,000$ metric tons [3]. High demand for cod combined with an underestimate of mortality led to overfishing, and the Atlantic cod population eventually plummeted. In 2005, the biomass of cod was estimated to be $50,000$ metric tons [3].

The Northeast Fisheries Science Center (NEFSC), located in Woods Hole, MA, is a subdivision of the National Oceanic and Atmospheric Administration (NOAA). The NEFSC manages

marine resources and provides options for balancing their harvesting and conservation. They set restrictions on fishing and determine the target levels for fish species in the Northeast Continental Shelf [4]. Directly observing fish abundance and behavior is difficult and expensive, therefore the NEFSC uses mathematical models to estimate the state of fish populations [5].

Mathematical models can examine either one species or multiple species. Single-species models are simpler in comparison; however, multi-species models account for interspecies interactions. NOAA's National Marine Fisheries Service has many types of stock assessment models grouped into the NOAA Fisheries Toolbox [6]. These models fall into two categories, core assessments and research models. Core assessments are based on peer reviewed methods, while research models have undergone testing within the organization but have not yet been subjected to peer review. Each stock assessment model evaluates a single species. The NEFSC has also developed multi-species models, but these are not currently used for stock assessments [7]. The NEFSC wants to ensure their models are giving reasonable results, but they do not want to put resources into multi-species stock assessments if the improvement in predictions is minimal.

The goal of this project is to compare single-species and multi-species models for stock assessments. We aim to determine whether or not a particular species would benefit from having stock assessments performed with a multi-species rather than a single-species model. To do this, we begin by developing single-species and multi-species models. We develop two types of models, partial differential equation (PDE) models and integrodifference models. We then use an ensemble Kalman filter to estimate parameters that are poorly known due to the difficulty of observing fish behavior. In this project, we model the Atlantic herring and Atlantic cod where the herring are prey to the cod.

1.2 Data Provided

The NEFSC provided us with survey data from 1968 to 2013. Their data is collected twice every year, once in the spring and once in the fall, from 640 different survey stations in the Northeast Continental Shelf. This region, shown in Figure 1.1, and consists of the Mid-Atlantic Bight (MAB), Georges Bank (GB), Scotian Shelf (SS), and the Gulf of Maine (GoM). In this project, we focus on Georges Bank.

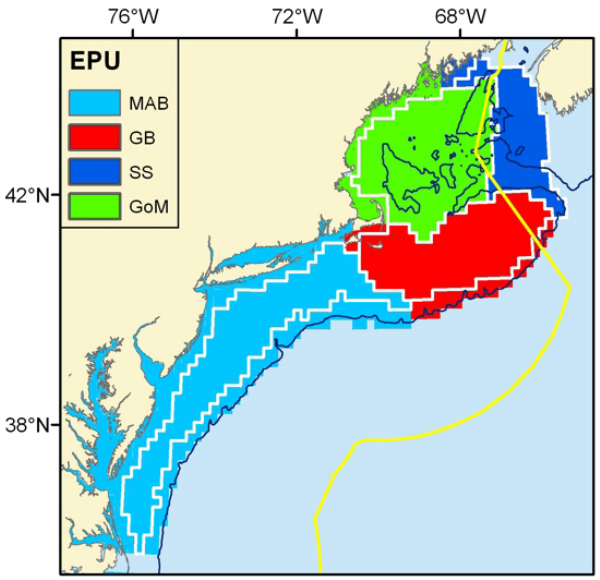


Figure 1.1: Map of the Northeast Continental Shelf. Shelf is divided into four regions: Mid-Atlantic Bight (MAB), Geroges Bank (GB), Scotian Shelf (SS), and Gulf of Maine (GoM).

Each region is subdivided into strata. Data is collected via a bottom trawl survey (BTS) at randomly selected strata, where each strata is an equally likely choice. In these bottom trawl surveys, a trawling boat, shown in Figure 1.2, goes to the preselected strata, drops a net, and drags it for 20 minutes along the ocean floor. When the time is up, the net is pulled up and the fish are sorted by species [8]. Data is then collected for each species. This includes the abundance, or total count, of each fish species, the total mass of the catch (in kilograms), the surface and bottom temperature of the water (in °Celsius), the strata number surveyed, the latitude and longitude of the survey location, and the region of the survey location (MAB, GB, SS, or GoM) [8]. The models in

Figure 1.2: Trawling boat used to conduct a BTS.

this project utilize the abundance as well as the year, season, and location of the survey.

The data within Georges Bank is shown in Figure 1.3, where each dot is a survey location. The surveys are conducted every spring between March and May, and every fall between September and November [8]. Since we do not know the exact month in which each survey was conducted, we treat spring data as occurring in April and fall data as occurring in October for the purposes of our models. Because the models in this paper are one-dimensional, the survey locations of the two-dimensional graph are projected onto the red line, and the value at each point on the line is the average of the abundance at each point mapped to it.

Figure 1.4 shows the abundance for Atlantic herring and Atlantic cod. Because of the noise

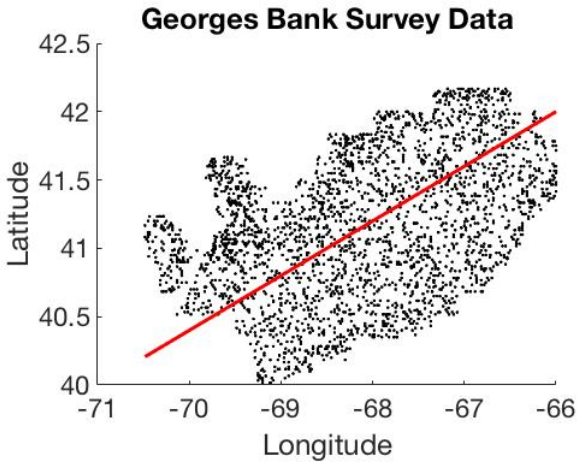
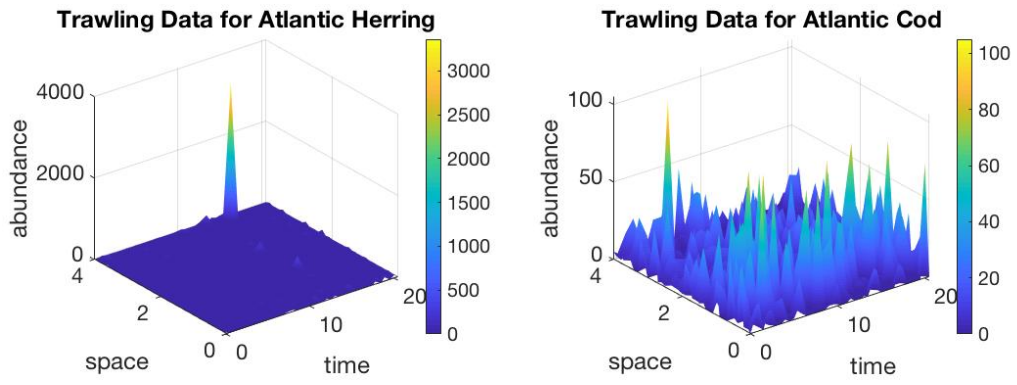


Figure 1.3: All survey data collected between 1980 through 2000 in Georges Bank in April and October



(a) (b)

Figure 1.4: The abundance of Atlantic herring (a) and Atlantic cod (b) from 1980 through 2000 in Georges Bank in April and October

.

level of the data, the filtering process we describe later for parameter estimation did not respond well to the data in its raw form. We instead use synthetic data to estimate parameters, and use of the real data is addressed further in the Future Work section of this paper.

1.3 Fish Overview

1.3.1 Atlantic Herring

Atlantic herring can grow up to 14 inches and 1.5 pounds [9]. NOAA fisheries reported that Atlantic herring are not overfished according to the 2015 stock assessment. NOAA estimates the Atlantic herring population is at about $517,930$ metric tons, which is above their target population of $157,000$ metric tons [10].

We plot the abundance of the Atlantic herring based on the trawling data that was provided by the NEFSC in Figure 1.5. The average abundance of the Atlantic herring in the Northeast Continental Shelf is $8,953$ fish. Herring travel in schools of up to hundreds or thousands of fish. It is very rare to see a single herring or even a small school. Also, herring are not frightened by

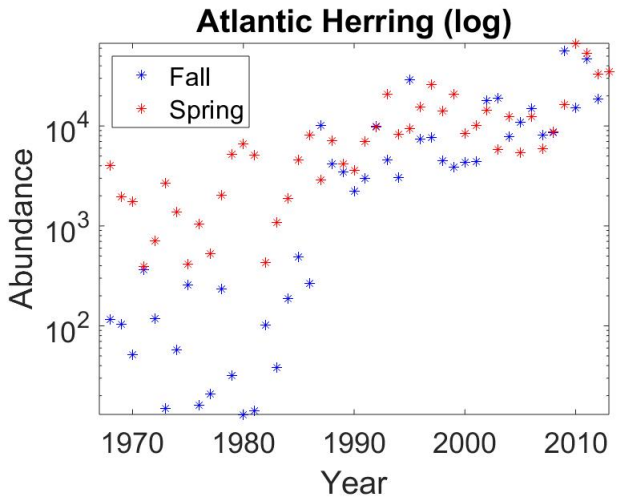


Figure 1.5: Atlantic herring abundance on a semi-log scale over all regions in the Northeast Continental Shelf.

boats, which could cause the data collected from the BTS to be skewed in comparison to other fish species [9].

Atlantic herring do not normally prey on other fish; however, they are often preyed upon by larger fish species [9]. Most notably, they are eaten by Atlantic cod, haddock, silver hake, striped bass, pollock, mackerel, salmon, tuna, dogfish, and mackerel sharks [9]. Atlantic herring migrate twice per year, north in the spring and south in the fall. The spring migration occurs between May and June and the fall migration occurs between November and December. Spawning occurs between July and October [11].

1.3.2 Atlantic Cod

Female Atlantic Cod can grow up to 57 inches and 54 pounds and males can grow up to 46 inches and 43 pounds. Some Atlantic cod can grow upwards of 100 pounds [9]. NOAA fisheries reported that Atlantic cod are overfished according to the 2017 stock assessment. Atlantic cod in the Northeast continental shelf live in Georges Bank and the Gulf of Maine [2].

We plot the abundance of the Atlantic cod based on the data that was provided by the NEFSC

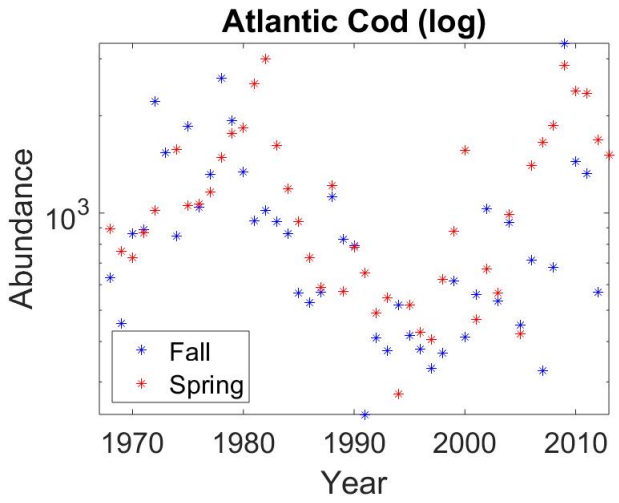


Figure 1.6: Atlantic cod abundance on a semi-log scale over all regions in the Northeast Continental Shelf.

in Figure 1.6. The average abundance of the Atlantic cod in the Northeast Continental Shelf is $1,093$ fish. Atlantic cod often stay close to the ocean floor, and large cod keep closer to the ground than small ones. Atlantic cod form compact schools during the day and will scatter at night [9].

Atlantic cod consume mollusks, Atlantic herring, crabs, lobsters, shrimps, brittle stars, sea urchins, sea cucumbers, and sea worms; there are times even a wild duck does not escape from a large cod. Young cod, from seven to eight inches, are also eaten by the larger cod. Large sharks and spiny dogfish sometimes prey on adult Atlantic cod [9]. Atlantic cod migrate twice per year, prior to and after spawning [12]. They migrate north to spawning grounds in March and return south in June [9]. Larger Atlantic cod travel very little outside of the spawning season. If they exhaust the food supply in one spot, they are driven to move to fresh foraging grounds. Additionally, if Atlantic cod are harassed too much by the spiny dogfish, they will move to another location [9].

Chapter 2

Partial Differential Equation Models

2.1 Background

For our partial differential equation (PDE) models, we consider the spatial area of Georges Bank along the Northeast Continental Shelf. We plot the survey collection locations that were used between 1980 and 2000 according to their specific latitude and longitude coordinates (Figure 1.1). This gives us a two-dimensional spatial region; for simplicity, we would like our PDE model to depend on only one spatial dimension. Therefore, we project the survey location points onto a line to create one dimensional data.

Our PDE model takes into account advection, diffusion, and reaction. The advection term represents a shift of the population as a whole and will take into account the migratory movements of the Atlantic cod and Atlantic herring, that were discussed in the Introduction. Diffusion models the spread of the population. Our reaction term will represent growth in population, as well as interaction in the multi-species model. The reaction term uses the logistic growth equation, as well as a variation of the Lotka-Volterra model.

2.1.1 Logistic Growth Model

For the logistic growth model, there are a variety of assumptions that are made for a given population in a defined region. The first assumption is the amount of resources in the region is a fixed amount. It is also assumed there will not be immigration or emigration from the defined region. All individuals in the population reproduce equally, and the rate of increase in the population is instantaneous.

The equation for the differential logistic growth model defines

$$\frac{dN}{dt} = rN\left(1 - \frac{N}{K}\right), \qquad N(0) = N_0 \tag{2.1}$$

where r is the growth rate of the population, measured in units of reciprocal time, and K is the carrying capacity, measured in units of number of individuals [13]. We are finding the growth of the population N at time t.

For insight in the behavior of the population governed by (2.1), we consider two cases $N_1(t)$ and $N_2(t)$ with the same values of r and K, but different initial conditions. We use the conditions $r = 0.1$, $K = 50$, $N_1(0) = 1$, and $N_2(0) = 60$ within the differential logistic growth model and we plot the behavior of this model (see Figure 2.1). If an initial population is chosen that is below the carrying capacity, the population will grow until it converges at the carrying capacity. If an initial population is chosen above the carrying capacity, the population will decrease to converge to the carrying capacity.

2.1.2 Lotka-Volterra Predator-Prey Model

For our multi-species model, we choose to incorporate predator-prey interactions between the Atlantic cod and Atlantic herring populations following the Lotka-Volterra model.

The growth rate of the cod is proportional to the rate of predation upon the herring, which is represented by the term ε in (2.3) and β in (2.2) [13]. Unlike the Lotka-Volterra model, we assume

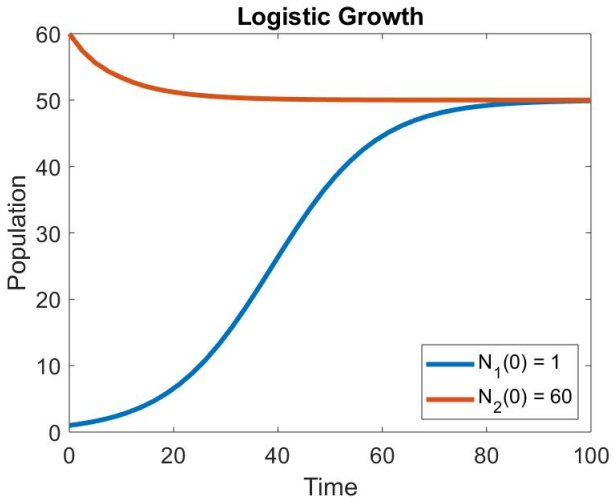


Figure 2.1: Differential logistic growth model where $r = 0.1$, $K = 50$, $N_1(0) = 1$, $N_2(0) = 60$, and time spans 100 years.

there are other prey for the cod to consume and can therefore grow in absence of the herring. We also assume the prey population will not grow to carrying capacity in the absence of the predator because of predation from other species.

The change in the prey population is determined by the following equation

$$\frac{dH}{dt} = \alpha H - \beta HC, \qquad H(0) = H_0 \tag{2.2}$$

where H is Atlantic herring and C is Atlantic cod. The parameters used within this model include α, the growth rate of Atlantic herring in the absence of the Atlantic cod and is measured in units of reciprocal time, and β, the mortality of the Atlantic herring due to predation by the Atlantic cod is measured in units of reciprocal of the number of predators times time. The equation for the predator population is determined by

$$\frac{dC}{dt} = \delta\beta CH - \gamma C, \qquad C(0) = C_0 \tag{2.3}$$

where, again, H is the population of herring and C is the population of cod. The parameter δ is

the growth rate of the Atlantic cod due to consumption of the Atlantic herring and is measured in units of the number of predators divided by the number of prey. The γ term is the death rate of the Atlantic cod in the absence of Atlantic herring and is measured in units of reciprocal time. Figure 2.2 shows the interaction of cod and herring using the conditions $\alpha = 1$, $\beta = 0.1$, $\delta = 0.75$, $\gamma = 1.5$, $H(0) = C(0) = 20$.

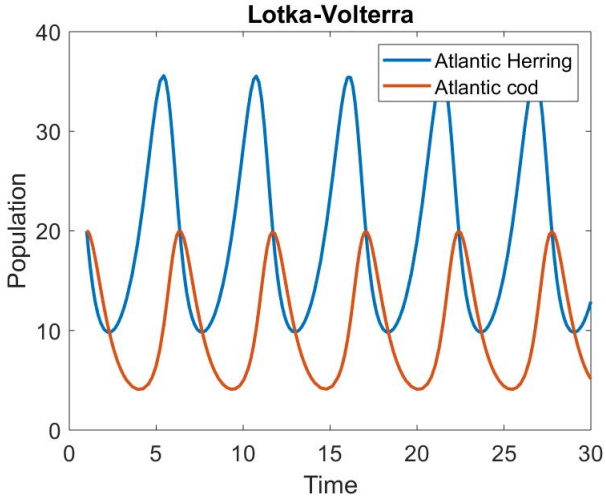


Figure 2.2: The values of the parameters used for this example are $\alpha = 1$, $\beta = 0.1$, $\delta = 0.75$, and $\gamma = 1.5$. The initial conditions are $H(0) = C(0) = 20$ and time spans 30 years. Cod populations increase when herring populations are high, and high cod populations decrease the herring population. This results in the oscillations pictured.

2.2 Derivation

2.2.1 Overview

To sensibly represent a physical system, any PDE model must satisfy the conservation law which states

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \vec{\phi} = R(\rho, \vec{x}, t) \tag{2.4}$$

where we choose definitions for the flux term, $\vec{\phi}$, and the reaction term, $R(\rho, \vec{x}, t)$.

2.2.2 Conservation Law and Flux Definition

Flux represents movement in and out of the system, and its definition is generally based on scientific observation. For our project, we define flux to be

$$\vec{\phi} = \vec{v}\rho - D\nabla\rho. \tag{2.5}$$

The source term, $R(\rho,\vec{x},t)$, represents creation and destruction within the system [14]. The source term in our model represents population growth and population death. The reaction term in our single species models is

$$R(\rho) = \alpha\rho\big(1-\frac{\rho}{K}\big) - \mu\rho \tag{2.6}$$

which is the logistic growth equation with a mortality term subtracted to account for population decline due to fishing.

In the multi-species case, we combine the Lotka-Volterra equations for predator-prey interactions with the logistic growth equation and again subtract fishing mortality terms for both the herring and the cod. Denoting Atlantic herring with the subscript h and Atlantic cod with the subscript c,

$$R_h(\rho_h,\rho_c,t) = \alpha_h(t)\rho_h\big(1-\frac{\rho_h}{K_h}\big) - \beta\rho_h\rho_c - \mu_h\rho_h \tag{2.7}$$

$$R_c(\rho_h,\rho_c,t) = \varepsilon\rho_h\rho_c + \alpha_c(t)\rho_c\big(1-\frac{\rho_c}{K_c}\big) - \mu_c\rho_c \tag{2.8}$$

where β is the predation rate of herring due to the cod and ε is the conversion efficiency rate of cod growth due to consuming herring.

We begin by deriving the conservation law to demonstrate that the PDE must satisfy it. Consider a region V and an amount of matter M inside of V. In our project, V is an arbitrary three-dimensional region of the ocean and M is the total number of fish inside of region V. By the

definition of density,

$$M = \int_V \rho(\vec{x},t)\,dV \tag{2.9}$$

where ρ is the mass density and has units $\frac{kg}{m^3}$, M is the amount of mass with units kg, and $\vec{x}$ is space.

Flux is defined as the rate at which mass flows through a given area. V is a volume. The source term is the rate per unit volume where mass is generated or lost, notated by $R(\rho,\vec{x},t)$. Finally, the rate of increase of mass in V is $\frac{dM}{dt}$. Mathematically, this can be written as

$$\frac{dM}{dt} = \frac{d}{dt}\int_V \rho(\vec{x},t)\,dV = +\int_V -\vec{\phi}\cdot\hat{n}\,dS + \int_V F(\rho,\vec{x},t)\,dV \tag{2.10}$$

where $\hat{n}$ is the outward normal vector to V. Since V is independent of time, equation (2.10) can be simplified to

$$\frac{dM}{dt} = \int_V \frac{\partial\rho}{\partial t}\,dV = -\int_V \vec{\phi}\cdot\hat{n}\,dS + \int_V F\,dV. \tag{2.11}$$

From here, the Divergence Theorem is applied. The theorem states that given a differentiable vector field $\vec{\phi}$ on a volume V:

$$\int_{dV} \vec{\phi}\cdot\hat{n}\,dS = \int_V \nabla\cdot\vec{\phi}\,dV. \tag{2.12}$$

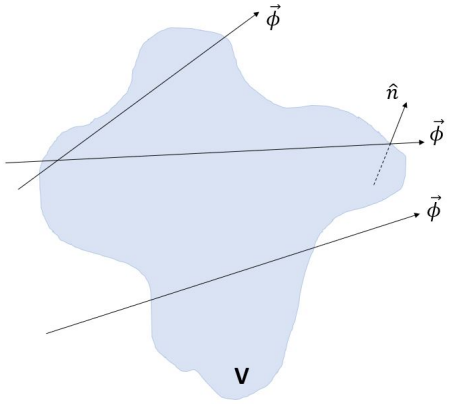


Figure 2.3: A control volume V through which a flux $\vec{\phi}$ of fish flows.

From (2.11), we find that

$$\int_V \frac{\partial \rho}{\partial t} + \nabla \cdot \vec{\phi} - F(\rho, \vec{x}, t)\, dV = 0. \tag{2.13}$$

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \vec{\phi} = F(\rho, \vec{x}, t). \tag{2.14}$$

Applying (2.13) to the ϕ term in equation (2.11) yields

$$\int_{\partial V} \vec{\phi} \cdot \hat{n}\, dS = \int_V \nabla \cdot \vec{\phi}\, dV. \tag{2.15}$$

Then, since all three terms are integrated with respect to V, we can condense

$$\int_V \frac{d\rho}{dt}\, dV = -\int_{\partial V} \vec{\phi} \cdot \hat{n}\, dS + \int_V R\, dV \tag{2.16}$$

to

$$\int_V \left(\frac{\partial \rho}{\partial t} + \nabla \cdot \vec{\phi} - R(\rho, \vec{x}, t) \right) dV = 0. \tag{2.17}$$

Since V is an arbitrary volume, then the integrand of (2.17) must be identically zero, so

$$\frac{\partial \rho}{\partial t} = -\nabla \cdot \vec{\phi} + R(\rho, \vec{x}, t), \tag{2.18}$$

where

$$\vec{\phi} = \vec{v}\rho - D\nabla\rho. \tag{2.19}$$

Since divergence is a linear operator,

$$\nabla \cdot \vec{\phi} = \nabla \cdot (\vec{v}\rho) - \nabla \cdot (D\nabla\rho). \tag{2.20}$$

With the addition of a reaction term $R(\rho,\vec{x},t)$, the conservation law gives the general form

$$\frac{\partial \rho}{\partial t} = -\nabla \cdot (\vec{v}\rho) + \nabla \cdot (D\nabla\rho) + R(\rho,\vec{x},t). \tag{2.21}$$

The PDE in this paper will only examine one spatial dimension, so (2.21) can be simplified for spatially uniform D to

$$\frac{\partial \rho}{\partial t} = -\frac{\partial}{\partial x}(v\rho) + D\frac{\partial^2 \rho}{\partial x^2} + R(\rho,t). \tag{2.22}$$

In the next section, we will further describe the terms in 2.22, including the velocity term and the reaction term.

2.3 Single-species Model

For our single-species models, we account for the seasonal migrations and spawning seasons of the Atlantic cod and the Atlantic herring. Recall, the Atlantic cod spawn annually April through May, and Atlantic herring spawn annually July through October [11]. Atlantic cod generally stay in the same area, called their home range, throughout the year, but they do migrate to spawn. Their spring migration occurs in March, and they migrate back to their home range in June [9].

For the general PDE single-species model, we construct an advection-diffusion-reaction equation

$$\frac{\partial \rho_i}{\partial t} = -v_i(t)\frac{\partial \rho_i}{\partial x} + D_i\frac{\partial^2 \rho_i}{\partial x^2} + R_i(\rho_i,t) \tag{2.23}$$

where the subscript i represents the species being modeled, $i = h$ for Atlantic herring and $i = c$ for Atlantic cod, and ρ_i is the population density of the species. The advection term is dependent on velocity, $v_i(t)$, which models the migration movement for the species. The diffusion term is dependent on the diffusion coefficient, D_i, which models the spatial spread of the species. The

reaction term, $R_i(\rho_i, t)$, is defined as

$$R_i(\rho_i, t) = \alpha_i(t)\rho_i\left(1 - \frac{\rho_i}{K_i}\right) - \mu_i\rho_i. \tag{2.24}$$

The reaction term models the population dynamics of the species. $\alpha_i(t)$ represents the population birth rate, K_i represents the carrying capacity, and μ_i is the death rate of the population.

We describe our modeling choices for $v_i(t)$ and $\alpha_i(t)$ in the following subsections. Since exact analytical solutions to these families of PDE's are not known, the choices for $v_i(t)$ and $\alpha_i(t)$ need to be amenable to computational simulation of these equations. Hence, we are going to require smooth, continuous functions to represent these different fish behaviors.

2.3.1 Velocity Function

The migratory movements of the Atlantic cod and the Atlantic herring are dependent upon time of year, therefore we create respective velocity terms dependent on time to model this spatial movement. Both species migrate to spawning grounds once every year. In order to represent these migration patterns, the velocity function is zero during most of the year when there is no migration, positive during spring migration, and negative during fall migration.

We construct a piecewise function using hyperbolic tangent functions (see Figure 2.27 and Figure 2.29). The general form of the *tanh* function is

$$f(t - t_0) = \frac{a}{2}\left[c + \tanh\left(b[t - t_0]\right)\right] \tag{2.25}$$

where a determines if the function is multiplied by one or negative one, in other words whether the step approximated by the function is a step up or a step down. The b term determines the steepness of the function. In this paper, $b = 200$ in all cases to model the change between moving and not moving. The c term determines the vertical shift of the function, $c = 1$ for the functions shaping

the positive pulse and $c = -1$ for the negative pulse. The horizontal shift, t_0, gives the location of the midpoint of the step.

For the Atlantic herring, we change the values of a, c, and t_0 depending on the time of year to match their migratory patterns to create

$$v_h(t) = \frac{a}{2}\left[c + \tanh\left(200[t - t_0]\right)\right] \tag{2.26}$$

where

$$\begin{cases} a = \ \ 1, c = \ \ 1, t_0 = \frac{2}{12} & 0 \le t \le \frac{5}{12} \\ a = -1, c = -1, t_0 = \frac{3}{12} & \frac{5}{12} < t \le \frac{9}{12} \\ a = -1, c = \ \ 1, t_0 = \frac{5}{12} & \frac{9}{12} < t \le \frac{11}{12} \\ a = \ \ 1, c = -1, t_0 = \frac{6}{12} & \frac{11}{12} < t \le 1 \end{cases}. \tag{2.27}$$

For the Atlantic cod, we change the values of a, c, and t_0 depending on the time of year to

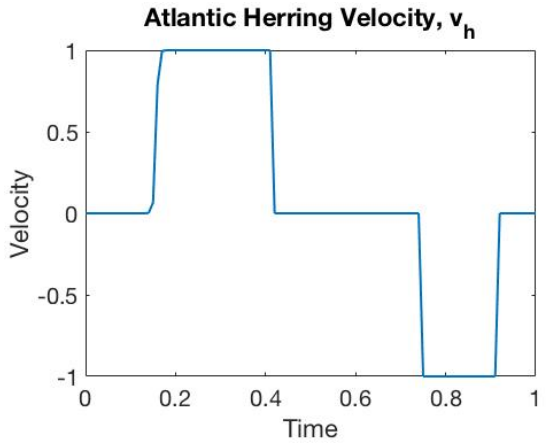


Figure 2.4: Atlantic herring velocity for the PDE model shows that from January to May and from June to November the Atlantic herring do not migrate because $v_h = 0$. From May to June the Atlantic herring migrate north because $v_h = 1$. From November to December the Atlantic herring migrate south because $v_h = -1$. The migration pattern will repeat for each additional year.

follow the species migratory pattern and construct their velocity term to be

$$v_c(t) = \frac{a}{2}\left[c + \tanh\left(200[t - t_0]\right)\right] \tag{2.28}$$

where

$$\begin{cases} a = \ \ 1, c = \ \ 1, t_0 = \frac{2}{12} & 0 \le t \le \frac{2.5}{12} \\ a = -1, c = -1, t_0 = \frac{3}{12} & \frac{2.5}{12} < t \le \frac{4}{12} \\ a = -1, c = \ \ 1, t_0 = \frac{5}{12} & \frac{4}{12} < t \le \frac{5.5}{12} \\ a = \ \ 1, c = -1, x_0 = \frac{6}{12} & \frac{5.5}{12} < t \le 1 \end{cases} \tag{2.29}$$

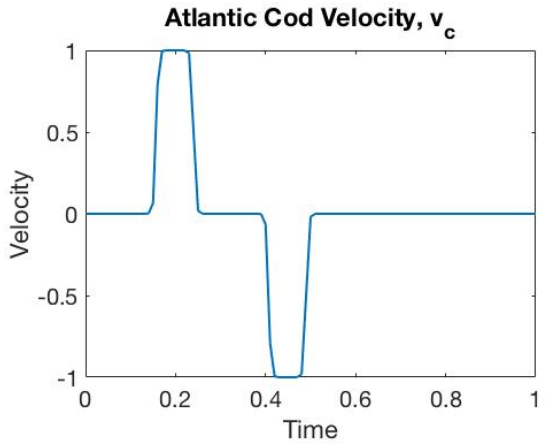


Figure 2.5: Atlantic cod velocity for the PDE model shows that from January to March, May to June, and July to December the Atlantic cod do not migrate because $v_h = 0$. From March to May the Atlantic cod migrate north because $v_h = 1$. From June to the July the Atlantic cod migrate south because $v_h = -1$. The migration pattern will repeat for each additional year.

2.3.2 Birth Rate Function

We use birth rate functions for Atlantic cod and Atlantic herring to model their spawning as dependent on time. Both Atlantic herring and Atlantic cod spawn annually: the herring from July to October and the cod from April to May. To represent this, the birth rate function $\alpha(t)$ should be zero during most of the year and positive only during the spawning season.

For the birth rate function, we use a Gaussian distribution such that

$$\alpha(t) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(t-\bar{\mu})^2}{2\sigma^2}} \tag{2.30}$$

where the midpoint of each spawning season is the mean, $\bar{\mu}$, and the standard deviation, σ, represents the duration of the spawning season. This allows the birth rate to be zero for most of the year and for the majority of the spawning season to occur at the midpoint of each spawning season. A larger σ represents a longer spawning season. The midpoint for the Atlantic herring spawning season is the beginning of September, therefore $\bar{\mu} = \frac{7}{12}$. The Atlantic herring have a spawning season that lasts four months, so σ will be larger for the Atlantic herring. The midpoint for the Atlantic cod spawning season is the beginning of May, therefore $\bar{\mu} = \frac{5}{12}$. The Atlantic cod have a spawning season that lasts two months, so σ will be smaller for the Atlantic cod. The Atlantic cod produce less offspring than the Atlantic herring, so the magnitude of the birth rate is smaller (see Figure 2.6).

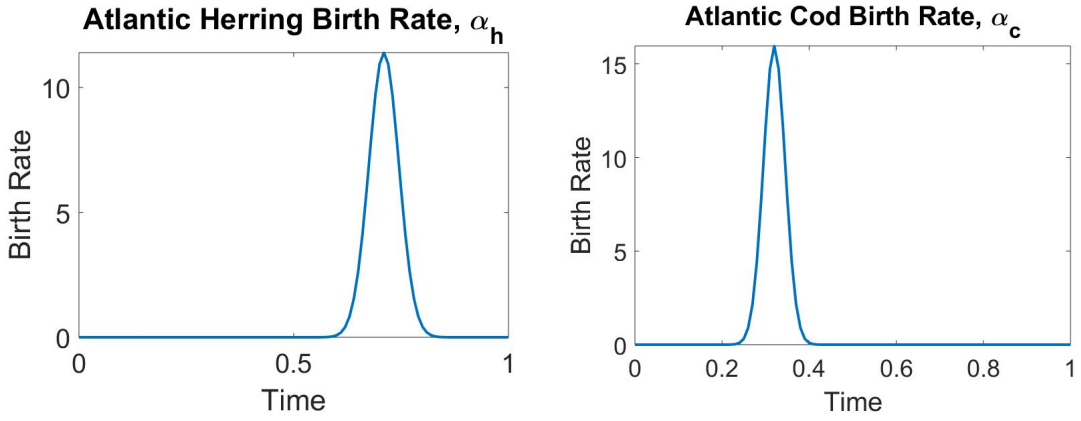


Figure 2.6: The Atlantic herring birth rate for the PDE models and integrodifference models where $\bar{\mu} = \frac{7}{12}$ and $\sigma = 0.035$. The Atlantic cod birth rate for the PDE models and integrodifference models where $\bar{\mu} = \frac{5}{12}$ and $\sigma = 0.025$.

The general single-species model for Atlantic herring and Atlantic cod is as follows:

$$\frac{\partial \rho_i}{\partial t} = -v_i(t)\frac{\partial \rho_i}{\partial x} + D_i\frac{\partial^2 \rho_i}{\partial x^2} + R_i(\rho_i, t) \tag{2.31}$$

$$R_i(\rho_i, t) = \alpha_i(t)\rho_i\left(1 - \frac{\rho_i}{K_i}\right) - \mu_i\rho_i. \tag{2.32}$$

The initial conditions are chosen to be:

$$\rho_h(x,0) = 5(x(3-x))^6\frac{1}{\sqrt{2\pi}}e^{\frac{-(x)^2}{2}} + 10 \tag{2.33}$$

$$\rho_c(x,0) = (x(3-x))^6\frac{1}{\sqrt{2\pi}}e^{\frac{-(x)^2}{2}} + 10. \tag{2.34}$$

Using the parameters in Table 2.1, we graph the solution to the model for Atlantic herring and the Atlantic cod. The parameters were chosen in order to show the interaction of the different functions in the model.

In Figure 2.7, the Atlantic herring begin to slowly decrease in population. The spawning seasons are represented by the increases and decreases of the population. The population increases during the spawning season and will decrease when no new fish are born. The migration patterns are also seen in Figure 2.7.

In Figure 2.8, you can see the Atlantic cod begin to slowly decrease in population. The spawning seasons are represented by the increases and decreases of the population. The population increases during the spawning season and will decrease when no new fish are born. Note the mag-

Meaning	*Parameter*	*Herring Value*	*Cod Value*
Carrying capacity	K_i	100	50
Mortality rate	μ_i	0.5	1.5
Diffusion constant	D_i	0.1	0.005

Table 2.1: Parameters for single-species PDE models to show the interaction of the different functions $v_i(t)$ and $\alpha_i(t)$.

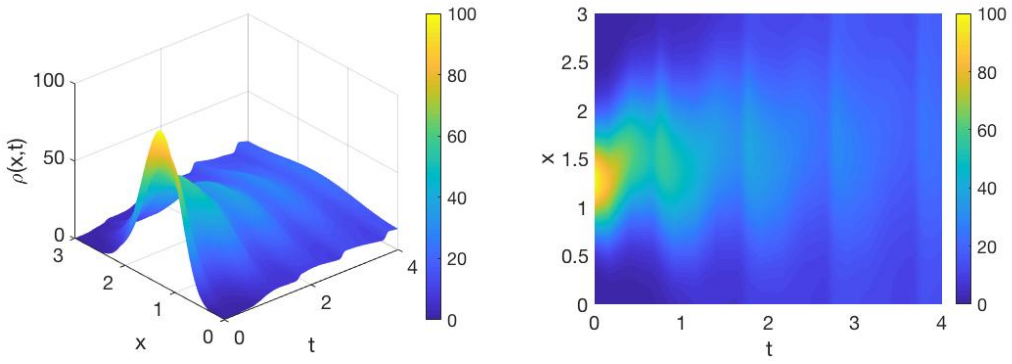


Figure 2.7: Results of the evolution of the single-species PDE model (2.31) using the initial conditions (2.33) for the parameter values are shown in Table 2.1.

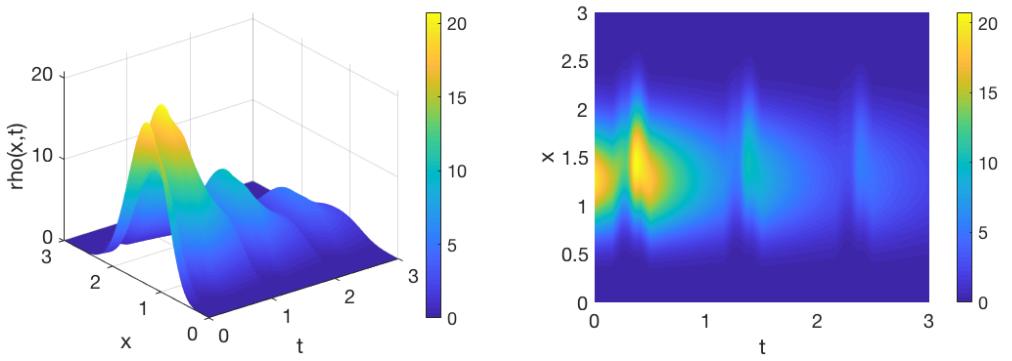


Figure 2.8: Results of the evolution of the single-species PDE model (2.31) using the initial conditions (2.34) for the parameter values are shown in Table 2.1.

nitude of the Atlantic cod is smaller than the Atlantic herring. This is because there are fewer Atlantic cod than there are Atlantic herring as defined in Table 2.1. In Figure 2.8, you can see the movement of the Atlantic cod, which is from the migration.

2.4 Multi-species Model

We examine the relationship between the Atlantic cod and the Atlantic herring. Recall the Atlantic cod is the predator in the relationship, and the Atlantic herring is the prey. The following coupled model displays the interaction between the pair using the same components of the single-species

models, but additionally uses a Lotka-Volterra type interaction. The coupled model is as follows:

$$\frac{\partial \rho_h}{\partial t} = -v_h(t)\frac{\partial \rho_h}{\partial x} + D_h\frac{\partial^2 \rho_h}{\partial x^2} + R_h(\rho_h, \rho_c, t) \tag{2.35}$$

$$\frac{\partial \rho_c}{\partial t} = -v_c(t)\frac{\partial \rho_c}{\partial x} + D_c\frac{\partial^2 \rho_c}{\partial x^2} + R_c(\rho_h, \rho_c, t) \tag{2.36}$$

where,

$$R_h(\rho_h, \rho_c, t) = \alpha_h(t)\rho_h\left(1 - \frac{\rho_h}{K_h}\right) - \beta\rho_h\rho_c - \mu_h\rho_h \tag{2.37}$$

$$R_c(\rho_h, \rho_c, t) = \varepsilon\rho_h\rho_c + \alpha_c(t)\rho_c\left(1 - \frac{\rho_c}{K_c}\right) - \mu_c\rho_c \tag{2.38}$$

are the growth rate functions. These growth rate functions show the predator-prey relationship between the Atlantic cod and the Atlantic herring. Observe, the logistic growth function. Note, $\beta\rho_h\rho_c$ is being subtracted from the logistic growth portion in the Atlantic herring portion of the multi-species model, which accounts for the mortality of Atlantic herring due to being consumed by the Atlantic cod. In the Atlantic cod part of the multi-species model, $\varepsilon\rho_h\rho_c$ is being added to the logistic growth part. This is the conversion efficiency, which accounts for the growth rate of the Atlantic cod from consuming the Atlantic herring. The velocity and birth rate functions for each species are unchanged from the single-species model.

Using the parameters in Table 2.1, we graph the solution to the multi-species model for Atlantic herring and Atlantic cod. Additionally, the initial conditions for the PDE multi-species model match the single-species initial conditions defined in (2.33) and (2.34).

In Figure 2.9 the Atlantic herring and the Atlantic cod populations start out at their initial conditions. We can see an initial growth in the cod population and an initial decrease in the herring population. We can also see that cod growth follows increases in the herring population.

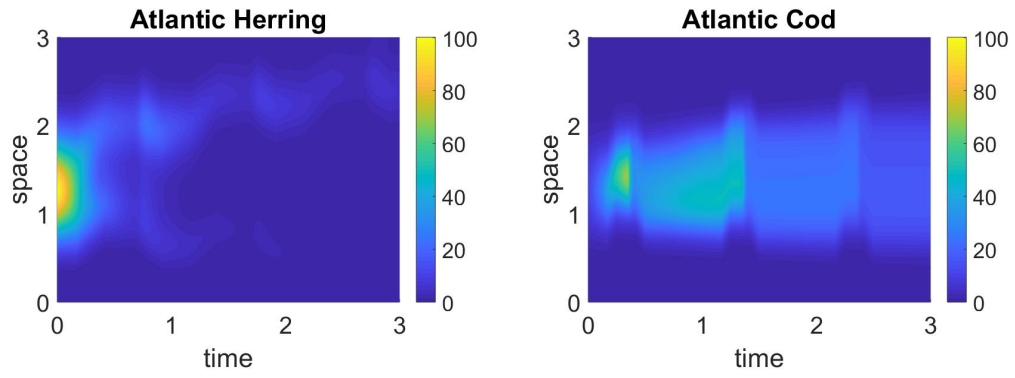


Figure 2.9: Results of the evolution of the multi-species PDE model (2.35) and (2.36) using the initial conditions (2.34) and (2.33) for Atlantic Herring and Atlantic Cod. Parameter values are shown in Table 2.1.

2.5 Numerical Methods

Since analytical solutions to the system (see (2.35) and (2.36)) are not evident, we are required to use computational techniques to find approximate solutions to these equations. For simplicity, we adopt the notation

$$\rho(x_j, t_n) \approx U_j^n. \tag{2.39}$$

Where U is the solution matrix approximating the population density function at points x_j and t_n [15]. Let $\Delta x = h$ and $\Delta t = k$ For $\frac{\partial \rho}{\partial t}$, we use a forward difference in time

$$\frac{\partial \rho}{\partial t} \approx \frac{U_j^{n+1} - U_j^n}{k}. \tag{2.40}$$

For the advection term, we use the upwinding method. We start by using a one-sided approximation to $\frac{\partial \rho}{\partial x}$,

$$\frac{\partial \rho}{\partial t} \approx -v\frac{U_j^n - U_{j-1}^n}{h}. \tag{2.41}$$

We combine this with a forward difference in time, as shown in (2.41), and apply the approximations to the advection equation,

$$\frac{\partial \rho}{\partial t} = -v\frac{\partial \rho}{\partial x} \tag{2.42}$$

to find the upwinding method

$$U_j^{n+1} = U_j^n - \frac{vk}{h}\left(U_j^n - U_{j-1}^n\right), \tag{2.43}$$

which is first order accurate in both space and time [15].

For the diffusion term, we use the Crank-Nicolson method, which approximates a solution to the diffusion equation

$$\frac{\partial \rho}{\partial t} = D\frac{\partial^2 \rho}{\partial x^2} \tag{2.44}$$

by approximating $\frac{\partial \rho}{\partial t}$ using a forward difference in time and $\frac{\partial^2 \rho}{\partial x^2}$ using the average of two central difference approximations at times t and $t+1$

$$\frac{\partial^2 \rho}{\partial x^2} \approx D\frac{U_{j-1}^{n+1} - 2U_j^{n+1} + U_{j+1}^{n+1} + U_{j-1}^n - 2U_j^n + U_{j+1}^n}{2h^2}. \tag{2.45}$$

For the reaction terms, we replace $\rho(x_j, t_n)$ with U_j^n . This is referred to as the forward Euler method, and it is the first order case of the Adams-Bashforth family of explicit approximations [15]. Using the upwinding, Crank-Nicolson, and forward Euler methods, we rewrite Equation (2.22) as

$$-\frac{U_j^{n+1} - U_j^n}{k} = \frac{(U_j^n - U_{j-1}^n)}{h} + \frac{(U_{j-1}^n - 2U_j^n + U_{j+1}^n)}{h^2} + R(U_j^n), \tag{2.46}$$

and we rearrange so the $n+1$ terms are on the left-hand side and the n terms are on the right-hand

side

$$-\frac{1}{h^2}U_{j-1}^{n+1}+(1+\frac{2}{h^2})U_j^{n+1}-\frac{1}{h^2}U_{j+1}^{n+1}=U_j^n+\frac{k}{h}(U_j^n-U_{j-1}^n)+\frac{k}{h^2}(U_{j-1}^n-2U_j^n+U_{j+1}^n)+kR(U_j^n). \tag{2.47}$$

The left-hand side of (2.47) can be written as

$$\left(\mathsf{I}-\frac{1}{h^2}\mathsf{B}\right)U^{n+1} \tag{2.48}$$

and the right-hand side becomes

$$\left(\mathsf{I}+\frac{k}{h}\mathsf{A}+D\frac{k}{h^2}\mathsf{B}\right)U^n+kR(U^n) \tag{2.49}$$

where

$$\mathsf{A}=\begin{bmatrix} -1 & 0 & 0 & 0 & \dots & 0 \\ 1 & -1 & 0 & 0 & \dots & 0 \\ 0 & 1 & -1 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & 1 & -1 \\ 0 & 0 & \dots & 0 & 0 & 1 \end{bmatrix} \tag{2.50}$$

$$\mathsf{B}=\begin{bmatrix} -2 & 1 & 0 & 0 & \dots & 0 \\ 1 & -2 & 1 & 0 & \dots & 0 \\ 0 & 1 & -2 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & -2 & 1 \\ 0 & 0 & \dots & 0 & 1 & -2 \end{bmatrix} \tag{2.51}$$

and

$$U^n = \begin{bmatrix} U_1^n \\ U_2^n \\ \vdots \\ U_{m-1}^n \\ U_m^n \end{bmatrix}. \tag{2.52}$$

We enforce the zero-slope boundary conditions through the use of image points, which results in the modified matrices for A and B respectively

$$\mathsf{A} = \begin{bmatrix} 0 & 0 & 0 & 0 & \dots & 0 \\ 1 & -1 & 0 & 0 & \dots & 0 \\ 0 & 1 & -1 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & 1 & -1 \\ 0 & 0 & \dots & 0 & 0 & 0 \end{bmatrix} \tag{2.53}$$

and

$$\mathsf{B} = \begin{bmatrix} -2 & 2 & 0 & 0 & \dots & 0 \\ 1 & -2 & 1 & 0 & \dots & 0 \\ 0 & 1 & -2 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & -2 & 1 \\ 0 & 0 & \dots & 0 & 2 & -2 \end{bmatrix}. \tag{2.54}$$

We can then write (2.47)

$$U^{n+1} = \left(\mathsf{I} - \frac{1}{h^2}\mathsf{B}\right)^{-1} \left[\left(\mathsf{I} + \frac{k}{h}\mathsf{A} + D\frac{k}{h^2}\mathsf{B}\right)U^n + kR(U^n)\right]. \tag{2.55}$$

To resolve stability issues during the first time step of the model, we use bootstrapping. Instead of

computing the first time step directly, we define a smaller time step

$$k_2 = \frac{k^2}{2} \tag{2.56}$$

and evaluate the model with a time step of k_2 until we reach $t = k$. We save the model output at time $t = k$ as the value of our first update, and then we return to updating with a time step of k. Smaller time steps allow the model to make more accurate approximations of the solution, but running the model with a very small time step for the total time period would require much more computational power than using a larger time step. To make the model more stable but maintain efficiency, we use this smaller time step, k_2, only at the beginning of our time period to better capture initial behavior without sacrificing computational time.

Chapter 3

Integrodifference Models

3.1 Background

Integrodifference models are commonly used for ecology applications such as invasive species models [16]. The integrodifference model gives a total population. Our model is comprised of two parts, a survival rate and a net growth rate. The net growth rate includes a kernel and a growth term [17]. There are a variety of possibilities for the growth term; for our project, we use the differential logistic growth equation to mirror the PDE models. Also, there are different options for the kernel; we use the Gaussian kernel. Additionally, we use the trapezoidal rule to numerically approximate the definite integrals in the integrodifference models

3.2 Single-species Models

The general form of our integrodifference model is

$$N_{i,\,t+1}(x) = s\,N_{i,\,t}(x - v_i(t)) + \int_a^b \Big[k(x-y)\; f(N_{i,\,t}(y))\Big]\,dy, \tag{3.1}$$

where

$$f(N_{i,\,t}(y)) = \alpha_i(t) N_{i,\,t}(y)\left(1 - \frac{N_{i,\,t}(y)}{K_i}\right) \tag{3.2}$$

and

$$s = e^{-\mu_i \Delta t}. \tag{3.3}$$

We let i represent either Atlantic herring, h, or Atlantic cod, c. In our model, we measure time t as months. The output of this model, $N_{i,\,t+1}(x)$, is the net population of the species at location x and time $t+1$. Each term in this model contributes to the net population. First, s is the probability of survival at location x and time t and $N_{i,\,t}(x - v_i(t))$ is the population at location x and time t. These two terms multiplied together, $sN_{i,\,t}(x - v_i(t))$, yield the total amount of the population that survived from time t and location x. The integral represents the net population growth at time t. The transition kernel, $k(x-y)dy$, represents the probability that two individuals are separated by a distance of $(x-y)$ units. The population growth function, $f(N_{i,\,t}(y))$, is the population growth at location y and time t. Therefore, adding the net population that survived at location x and time t to the population growth yields the population at time $t+1$ and location x [16].

In our models, $e^{-\mu_i \Delta t}$ is the probability of survival at location x and time t. Also, $k(x-y)dy$ is the Gaussian kernel of the form

$$k(x-y)dy = \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{(x-y-\bar{\mu})^2}{2\sigma^2}} dy, \tag{3.4}$$

where $\bar{\mu} = 0$ and $\sigma = 2$. For the growth term, we use $f(N_{i,\,t}(y))$, which is the differential logistic growth equation. In the logistic growth equation, $\alpha_i(t)$ represents the growth rate, and we use equation 2.30. The term $N_{i,\,t}(x - v_i(t))$ is the population at time t shifted by $v_i(t)$, where $v_i(t)$ is the velocity function, which is determined by the migration pattern of each species. We describe the velocity function in more detail in the following section.

3.2.1 Spatial Shift Function

The spatial shift $v_i(t)$ appears in the term $N_{i,\,t}(x - v_i(t))$. This is the population at time t and space $x - v_i(t)$. Shifting by $v_i(t)$ accounts for migration. The whole population of Atlantic herring or Atlantic cod in space x will shift either north or south depending on the season. Recall, from the Introduction, that Atlantic cod migrate north in March and south in June. The Atlantic herring migrate north from May to June and south from November to December. To represent this, the spatial shift, $v_i(t)$, should be zero during most of the year, positive during spring migration, and negative during fall migration. The spatial shift function for the integrodifference model is similar in shape to the velocity function used in the PDE model; however, here we use a Heaviside function. This was chosen over the hyperbolic tangent function because we do not need the graph to be smooth or continuous.

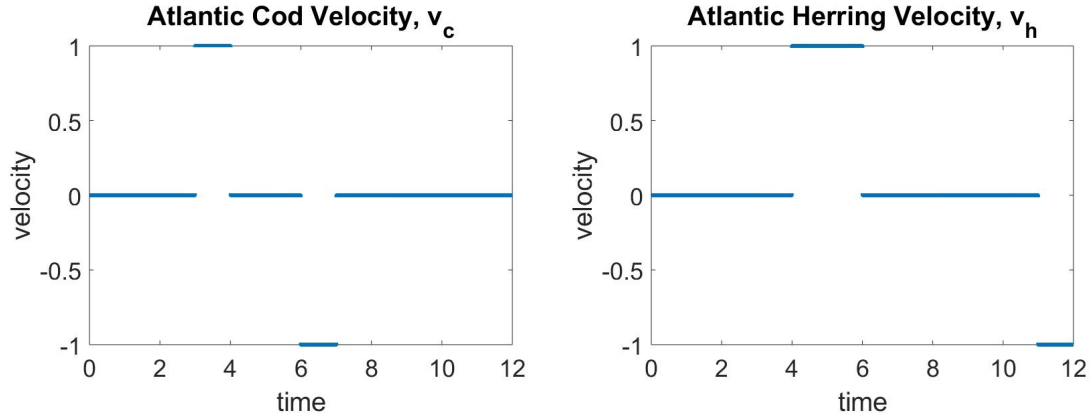


Figure 3.1: These are the spatial shift functions for the Atlantic cod and the Atlantic herring for the integrodifference model. When $v_i(t) = 1$, $N_{i,\,t}(x - v_i(t))$ represents the population at time t shifted north of x. Similarly when $v_i(t) = -1$, $N_{i,\,t}(x - v_i(t))$ represents the population at time t shifted south of x. When $v_i(t) = 0$, no shift occurs.

3.2.2 Single-species Model

The initial populations are chosen to be:

$$N_{h,0}(x) = 4(x(3-x))^6 \frac{1}{\sqrt{2\pi}} e^{\frac{-(x)^2}{2}} + 10 \tag{3.5}$$

$$N_{c,0}(x) = 2.5(x(3-x))^6 \frac{1}{\sqrt{2\pi}} e^{\frac{-(x)^2}{2}} + 10. \tag{3.6}$$

Figure 3.2 and Figure 3.3 show the model output for Atlantic herring and Atlantic cod using the parameters in Table 3.1. As an example, we selected parameters that clearly show the interaction of the different terms in the model.

Parameter	*Meaning*	*Herring Value*	*Cod Value*
μ_i	Mortality rate	0.5	0.2
α_i	Growth rate scale	5	2
v_i	Velocity magnitude	1	1
K_i	Carrying capacity	100	50
Δt	Time step	$\frac{1}{12}$	$\frac{1}{12}$

Table 3.1: Parameters for single-species Integrodifference models to show the interaction of the different functions $v_i(t)$ and $\alpha_i(t)$..

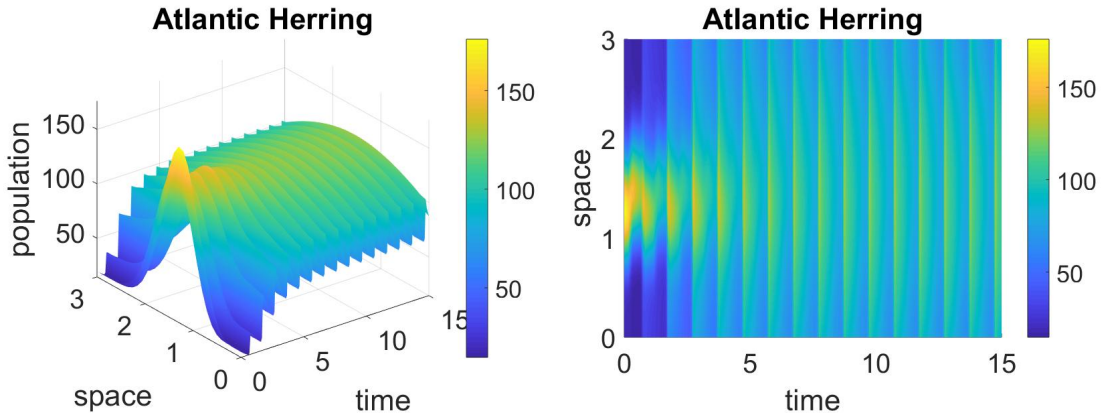


Figure 3.2: This figure shows the populations at each time and space step using (3.1). The initial conditions are (3.5) and the parameters are stated in Table 3.1.

Figure 3.2, shows the growth of the Atlantic herring population over 15 years. Additionally, in Figure 3.2 you can see increases and decreases in the Atlantic herring population. This is due to the Atlantic herring's population growing during spawning season and then decreasing when there is no spawning. The shift caused by the velocity function is seen in the changes in population along the x-axis.

Figure 3.3 shows the growth of the Atlantic cod over the same 15 years. Because the initial population is larger than the carrying capacity, the population declines. Over time, the population begins to recover. The magnitude of the Atlantic cod in Figure 3.3 is smaller than the magnitude of the Atlantic herring in Figure 3.2. This is because there are more Atlantic herring than Atlantic cod in the ocean. Additionally, in Figure 3.3 you can see increases and decreases in the Atlantic cod population. This is due to the Atlantic cod's population increasing during spawning season and decreasing during the rest of the year. The shift caused by the velocity function is seen in the changes in population along the x-axis.

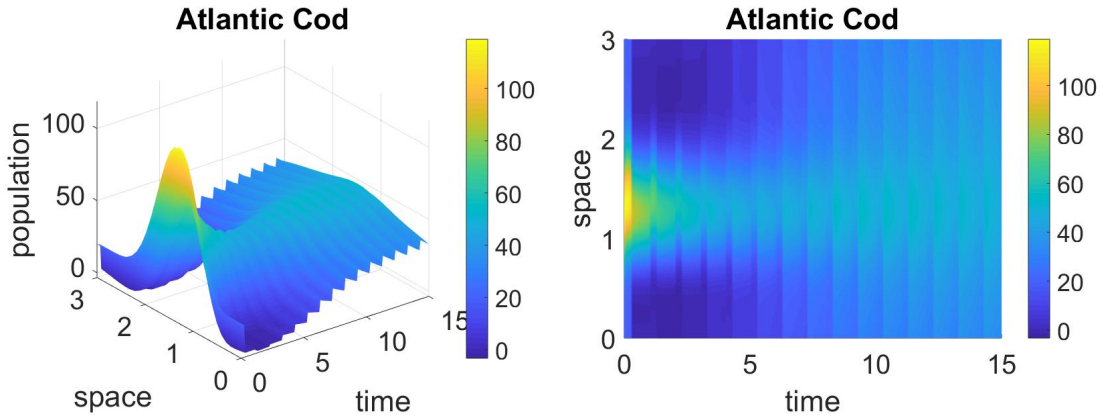


Figure 3.3: This figure shows the populations at each time and space step using (3.1). The initial conditions are (3.6) and the parameters are stated in Table 3.1.

3.3 Multi-species Model

Now, we examine the relationship between the Atlantic cod and the Atlantic herring. Recall that the Atlantic cod is preys on the Atlantic herring. The following coupled model displays the interaction between the pair using the same components as the single-species models, but it also uses a Lotka-Volterra type interaction. The coupled model is as follows:

$$N_{h,\,t+1}(x) = e^{-\mu_h \Delta t}\, N_{h,\,t}(x - v_h(t)) + \int \Big[k(x-y)\; f(N_{h,\,t}(y))\Big] dy \tag{3.7}$$

$$N_{c,\,t+1}(x) = e^{-\mu_c \Delta t}\, N_{c,\,t}(x - v_c(t)) + \int \Big[k(x-y)\; f(N_{c,\,t}(y))\Big] dy, \tag{3.8}$$

where the growth rate functions

$$f(N_{h\,t}(y)) = \alpha_h N_{h,\,t}(y))\Big(1 - \frac{N_{h,\,t}(y)}{K_h}\Big) - \beta N_{h,\,t}(y) N_{c,t}(y) \tag{3.9}$$

$$f(N_{c,\,t}(y)) = \alpha_c N_{c,\,t}(y)\Big(1 - \frac{N_{c,\,t}(y)}{K_c}\Big) + \varepsilon N_{h,\,t}(y) N_{c,\,t}(y) \tag{3.10}$$

show the predator-prey relationship between Atlantic cod and Atlantic herring. Observe that in (3.9), $\beta N_{h,\,t}(y) N_{c,\,t}(y)$ is being subtracted from the logistic growth term. This accounts for the mortality rate of Atlantic herring due to being consumed by the Atlantic cod. In (3.10), $\varepsilon N_{h,\,t}(y) N_{i,\,t}(y)$ is being added to the logistic growth term. This accounts for the growth rate of the Atlantic cod from consuming the Atlantic herring.

The initial conditions for the multi-species model are:

$$N_{h,0}(x) = (x(3-x))^6 \frac{1}{\sqrt{2\pi}} e^{\frac{-(x)^2}{2}} + 10 \tag{3.11}$$

$$N_{c,0}(x) = 0.25(x(3-x))^6 \frac{1}{\sqrt{2\pi}} e^{\frac{-(x)^2}{2}} + 10. \tag{3.12}$$

Using the parameters in Table 3.2, we graph the solution to the multi-species model for Atlantic herring and Atlantic cod. The length of time is increased from 15 years to 50 years to show the predator-prey interaction over a longer period of time.

Parameter	*Meaning*	*Herring Value*	*Cod Value*
μ_i	Mortality rate	1.5	0.5
α_i	Growth rate scale	0.5	0.5
v_i	Velocity magnitude	1	1
K_i	Carrying capacity	150	25
Δt	Time step	$\frac{1}{12}$	$\frac{1}{12}$

Table 3.2: Parameters for multi-species integrodifference models using (3.7) and (3.8) to show the interaction of the different functions $v_i(t)$ and $\alpha_i(t)$..

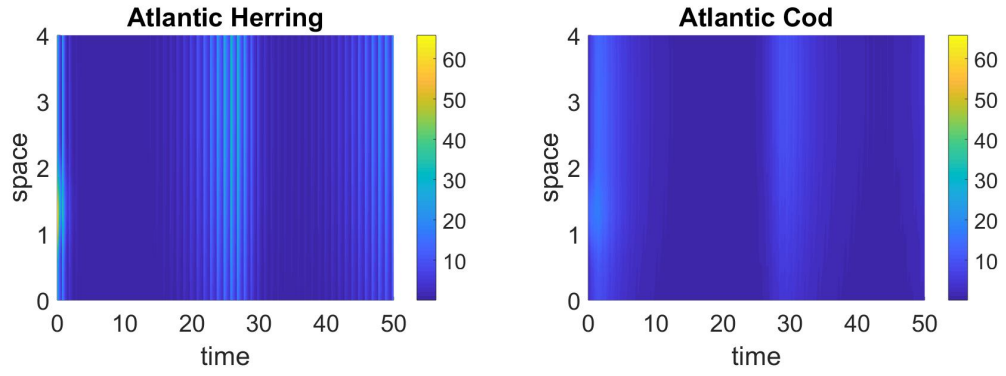


Figure 3.4: Results of the multi-species model (3.7) and (3.8) using the initial conditions (3.11) and (3.12) for Atlantic Herring, and Atlantic Cod respectively. Parameter values are shown in Table 3.2.

The Atlantic cod population spikes due to the abundance of the Atlantic herring at the beginning of Figure 3.4. This causes a decrease in the Atlantic herring population, and the Atlantic cod population eventually decreases because there are fewer Atlantic herring. This allows for the Atlantic herring population to recover. Soon after, the Atlantic cod population increases because there are more Atlantic herring to consume. This relationship continues over time.

Chapter 4

Parameter Estimation

4.1 Motivation

All of our models rely on parameters based on the behavior of the Atlantic cod and the Atlantic herring. However, many of these behaviors have not been quantified from direct observation, therefore their real values are unknown. In order to get reasonable estimates for these unknown parameters, we use an ensemble Kalman filter. Within our models, there are five parameters we chose estimate. For the single-species PDE Atlantic herring model and Atlantic cod model, we estimate the carrying capacities and the diffusion coefficients. For the multi-species PDE model, we estimate the carrying capacities, diffusion coefficients, rate of predation on herring by cod, and conversion efficiency. For the single-species integrodifference models, we estimate the carrying capacities and the death rates. For the multi-species integrodifference, we estimate the carrying capacities, death rates, predation rate, and conversion efficiency.

4.2 Inverse Problem

Determining unknown parameters that fits a given model and system is considered an inverse problem. Solving inverse problems requires a known model function and discrete observations of the system being modeled, but the parameters and initial states may be unknown or poorly known. There are many ways to approach solving inverse problems [18]. We solve for our unknown parameters using the Bayesian perspective. This allows for states and parameters to be treated as random variables with probability distributions. We understand our solution, also called our joint posterior density, to be

$$\pi(x, \theta \mid y) \propto \pi(y \mid x, \theta)\pi(x, \theta) \tag{4.1}$$

which says the probability of our states and parameters given our observations, is proportional to the probability of our observations given our states and parameters multiplied by the probability of our states and our parameters [19].

4.3 Ensemble Kalman Filter

We use a parameter estimation algorithm known as the ensemble Kalman filter to estimate the parameters in our models. In particular, the augmented ensemble Kalman filter allows for the unknown model states and parameters to be estimated simultaneously. It utilizes our model's outputs as well as observation data to determine a best estimate for parameter values. An ensemble Kalman filter approaches the inverse problem from the Bayesian perspective, meaning we will treat states and parameters as random variables with probability distributions. Each probability distribution is represented by random samples known as ensemble members. At each time step within the filter, the random samples are updated and we calculate their means and standard deviations. The final mean is our resulting parameter estimate [19].

The ensemble Kalman filter algorithm is comprised of two steps: a prediction step and an

analysis step. For the prediction step, we begin by assuming the current density, represented in terms of N ensemble members, is

$$S_{j|j} = (x^1_{j|j}, \theta^1_j), (x^2_{j|j}, \theta^2_j), \ldots, (x^N_{j|j}, \theta^N_j) \tag{4.2}$$

where x represents states and θ represents parameters. Each state prediction ensemble is updated at every time step following the equation

$$x^n_{j+1|j} = F(x^n_{j|j}, \theta^n_j) + v^n_j, \qquad n = 1, 2, \ldots, N \tag{4.3}$$

where F is a known model function and v^n_j is added noise, accounting for error in the model prediction. Next in the prediction step, we create an augmented vector, z, of states and parameters such that

$$z^n_{j+1|j} = \begin{bmatrix} x^n_{j+1|j} \\ \theta^n_j \end{bmatrix}, \qquad n = 1, 2, \ldots, N. \tag{4.4}$$

Then, we compute the ensemble statistics which include the mean,

$$\bar{z}_{j+1|j} = \frac{1}{N} \sum_{n=1}^{N} z^n_{j+1|j} \tag{4.5}$$

and the prior covariance matrix,

$$\Gamma_{j+1|j} = \frac{1}{N-1} \sum_{n=1}^{N} (z^n_{j+1|j} - \bar{z}_{j+1|j})(z^n_{j+1|j} - \bar{z}_{j+1|j})^T. \tag{4.6}$$

Next, we begin the analysis step. For the analysis step we utilize observation data, y_j, to create an observation ensemble

$$y^n_{j+1} = y_{j+1} + w^n_{j+1}, \qquad n = 1, 2, \ldots, N \tag{4.7}$$

where observation error, w^n_{j+1}, is added to each ensemble member. Once we have our observation ensemble, we compute the observation prediction ensemble

$$\hat{y}^n_{j+1} = g(x^n_{j+1|j}, \theta^n_j), \qquad n = 1,2,\ldots,N \tag{4.8}$$

where g is the nonlinear observation model. We also need to compute the Kalman gain for every time step in the filter

$$K_{j+1} = \sigma^{z\hat{y}}_{j+1}(\sigma^{\hat{y}\hat{y}}_{j+1} + \sigma^{y}_{j+1})^{-1} \tag{4.9}$$

where $\sigma^{z\hat{y}}_{j+1}$ is the cross covariance of the state and observation predictions, $\sigma^{\hat{y}\hat{y}}_{j+1}$ is the forecast error covariance of the observation prediction ensemble, and σ^{y}_{j+1} is the observation noise covariance.

Once these have been computed, we can calculate our posterior ensemble

$$z^n_{j+1|j+1} = z^n_{j+1|j} + K_{j+1}(y^n_{j+1} - \hat{y}^n_{j+1}), \qquad n = 1,2,\ldots,N \tag{4.10}$$

and posterior ensemble statistics, the means and covariances, for every state and parameter included in our ensemble members. In the final time step, the calculated means of the parameters are the parameter estimates.

4.4 Results

We are doing validation of our parameter estimation using synthetic data generated from our models, allowing us to begin with data that has more predictable behavior. To generate synthetic data, we add noise that follows a Gaussian distribution to our model's output. The Gaussian distribution is predetermined with a chosen variance and mean. In order to mimic the real NEFSC data provided, our synthetic data has values correlating to data being collected twice per year, in April and

October, as in the NEFSC's spring and fall data surveys. After validating the filter with synthetic data, we hope in the future to utilize the real data to estimate chosen parameters.

4.4.1 PDE: Single-species

For our partial differential equation single-species models, we estimate the carrying capacities of Atlantic cod and Atlantic herring, K_c and K_h, as well as their diffusion coefficients, D_c and D_h, respectively. The first step in the parameter estimation process is to run our model with selected parameter values to determine the true solution of the model. To find our true solutions, we run our models using the parameter values found in Table 4.1. First, we estimate our carrying capacity and diffusion coefficient for the single-species Atlantic cod PDE.

Meaning	*Parameter*	*Herring Value*	*Cod Value*
Mortality rate	μ_i	0.5	0.01
Growth rate scale	α_i	3	1
Velocity magnitude	v_i	1	1
Carrying capacity	K_i	100	20
Diffusion coefficient	D_i	0.1	0.1

Table 4.1: Parameter values used in computing single-species PDE model solutions, where estimated parameters are highlighted in red.

Parameter	True Value	Estimated Value	Relative Error
K_c	20	19.0627	0.0469
D_c	0.1	0.0314	0.686

Table 4.2: Relative error for Atlantic cod single-species PDE estimated parameters.

When we run our ensemble Kalman filter, we take the means of all of our ensemble members and plot these over the time span of 15 years. As seen in Figure 4.1, the Atlantic cod ensemble mean follows the same pattern as the true solution, which confirms that our filter is doing what it is supposed to. Turning to Table 4.2 and Figure 4.1 we see that our filter estimates the true

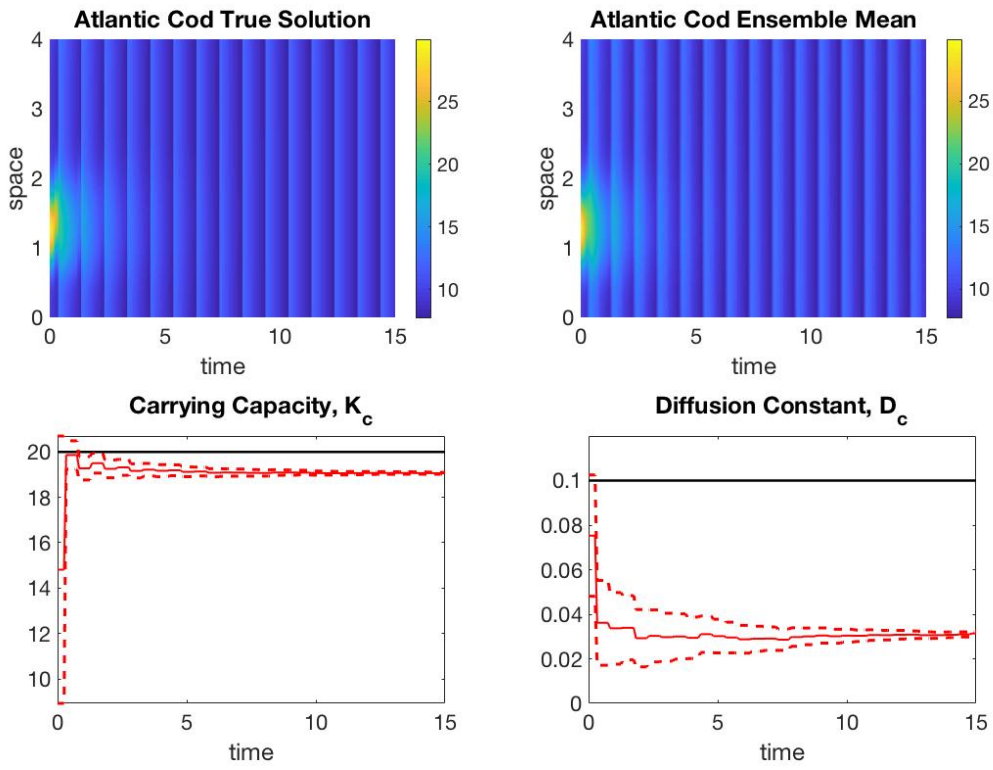


Figure 4.1: The true solution and the estimated mean of the filter ensembles of the single-species model, using (2.31) and the initial conditions (2.34) for Atlantic cod. This figure also shows the ensemble Kalman filter's estimate for K_h and D_h. The black line represents the true solution of the parameter. The solid red line represents the estimated value of the parameter over time. The dotted red lines are two standard deviations of error about the estimated parameter. The true and estimated parameter values are shown in Table 4.2.

value of the carrying capacity, 20 as 19.0627. This gives a relative error of 0.0469. Similarly, for the diffusion coefficient, the filter estimates the real diffusion coefficient of 0.1 to be 0.0314 with a relative error of 0.686. The estimated parameters for carrying capacity and the diffusive coefficient are an underestimate of the true parameters.

Next, we estimate our carrying capacity and diffusion coefficient for the single-species Atlantic herring PDE. In Figure 4.2, the Atlantic herring ensemble mean follows the same pattern as the true solution, which confirms that our filter is doing what it is supposed to. Turning to carrying capacity in Table 4.3 and Figure 4.2 we see that our true value is 100 with a filter estimate of 87.0043. This

gives a relative error of 0.13. Similarly, for the diffusion coefficient, the filter estimates D_h the true value of 0.1 to be 0.0768, which yields a relative error of 0.232. The estimated parameters for carrying capacity and the diffusion coefficient are an underestimate of the true parameters.

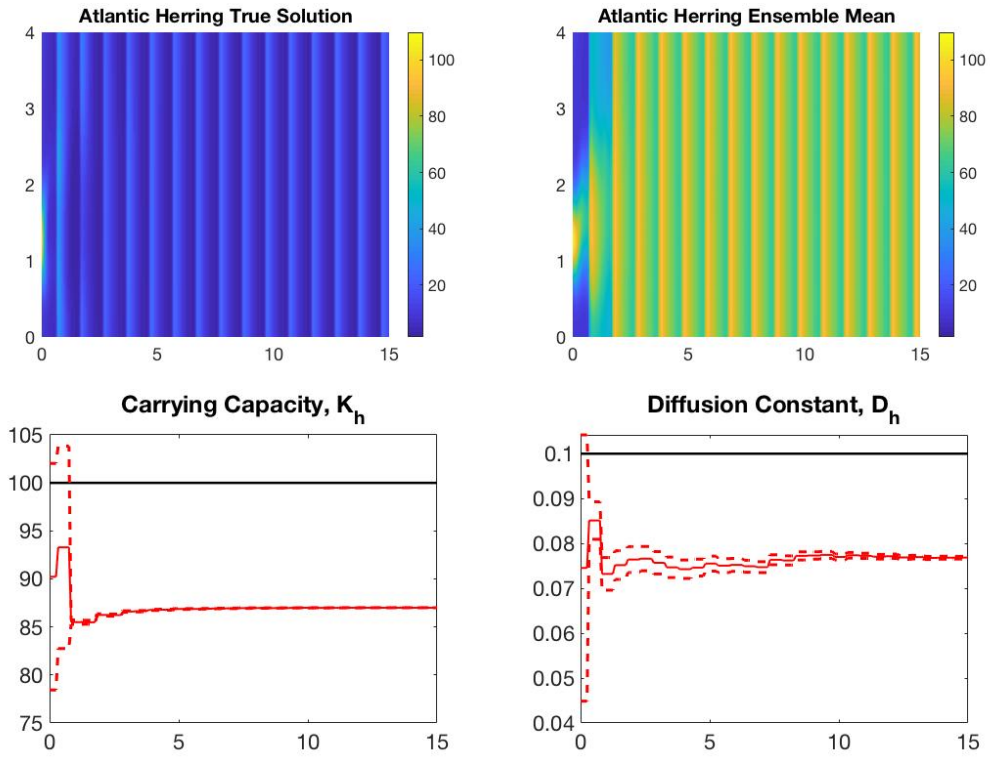


Figure 4.2: The true solution and the estimated mean of the filter ensembles of the single-species model using (2.31) and the initial conditions (2.33) for Atlantic herring. This figure also shows the ensemble Kalman filter's estimate for K_c and D_c. The black line represents the true solution of the parameter. The solid red line represents the estimated value of the parameter over time. The dotted red lines are two standard deviations of error about the estimated parameter. The true and estimated parameter values are shown in Table 4.3.

Parameter	True Value	Estimated Value	Relative Error
K_h	100	87.0043	0.13
D_h	0.1	0.0768	0.232

Table 4.3: Relative error for Atlantic herring single-species PDE estimated parameters.

4.4.2 PDE: Multi-species

For the multi-species PDE model, we estimate carrying capacity and the diffusion coefficient for each species, as well as β. We also attempt to estimate ε.

Meaning	*Parameter*	*Herring Value*	*Cod Value*
Mortality rate	μ_i	0.5	0.5
Growth rate scale	α_i	3	1
Velocity magnitude	v_i	1	2
Carrying capacity	K_i	100	20
Diffusion coefficient	D_i	0.1	0.1
Mortality from cod	β	0.1	N/A
Conversion efficiency	ε	N/A	0.1

Table 4.4: Parameter values used in computing multi-species PDE model solutions, where the parameters we estimate are highlighted in red.

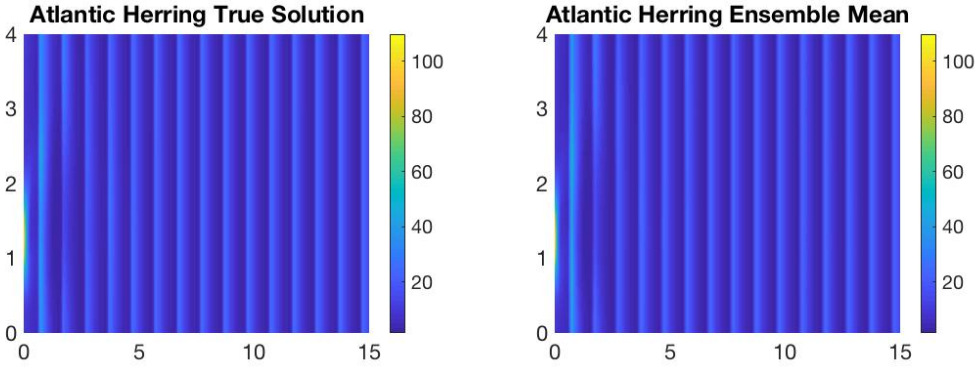


Figure 4.3: The true solution and the estimated mean of the filter ensembles of the single-species model use (2.35) and (2.36) using the initial conditions (2.34) and (2.33) for Atlantic herring and Atlantic cod.

In Figures 4.3 and 4.4, we can see that the Atlantic herring and Atlantic cod ensemble means follow the same patterns as the true solution to both of these. This verifies that our filter is working correctly. Moving on to the parameters in 4.5, we can observe what the ensemble Kalman filter estimates for the parameters. Starting at the carrying capacity for Atlantic herring, K_h, we can see, by the black line, that the true value of K_h is 100. The filter was able to estimate K_h as 57.4966.

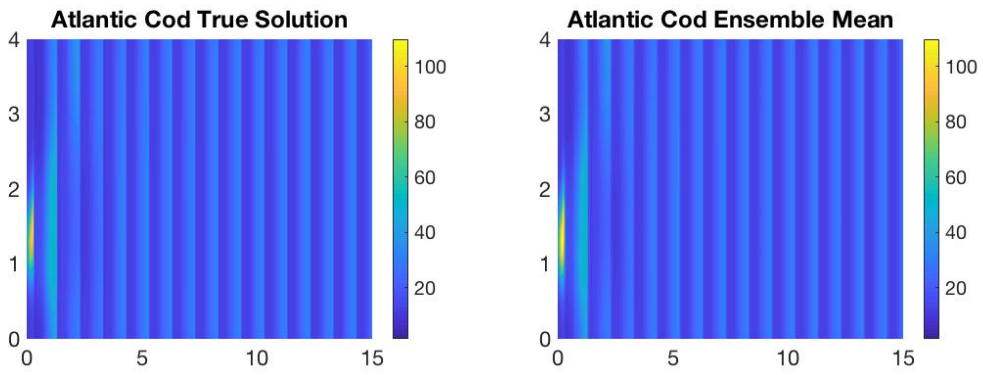


Figure 4.4: The true solution and the estimated mean of the filter ensembles use the (2.35) and (2.36) using the initial conditions (2.34) and (2.33) for Atlantic herring and Atlantic cod.

Parameter	True Value	Estimated Value	Relative Error
K_h	100	57.4966	0.425
D_h	0.1	0.0437	0.563
K_c	20	20.1249	0.0062
D_c	0.1	0.074	0.26
β	0.1	0.0901	0.099

Table 4.5: Relative error for Atlantic herring and Atlantic cod PDE multi-species model.

This gives us a relative error of 0.425, which is seen in 4.5. For the diffusion coefficient for Atlantic herring, we can see in 4.5 that the true value is 0.1 and that our filter estimates it to be 0.0437. This yields a relative error of 0.563, which is seen in 4.5. For the carrying capacity of Atlantic cod, K_c, we can see that the true value of K_c is represented by the black line in 4.5 and is 20. Our filter estimates K_c to be 20.1249, which gives a relative error of 0.0062. The diffusion coefficient for Atlantic cod, D_c, has a true value of 0.1 and the filter estimates it to be 0.074, which gives a relative error of 0.26. Next, we measure β, which has a true value of 0.1. The filter estimates it to be 0.0901, which gives a relative error of 0.099. The filter was unable to obtain a useful estimate of the parameter ε.

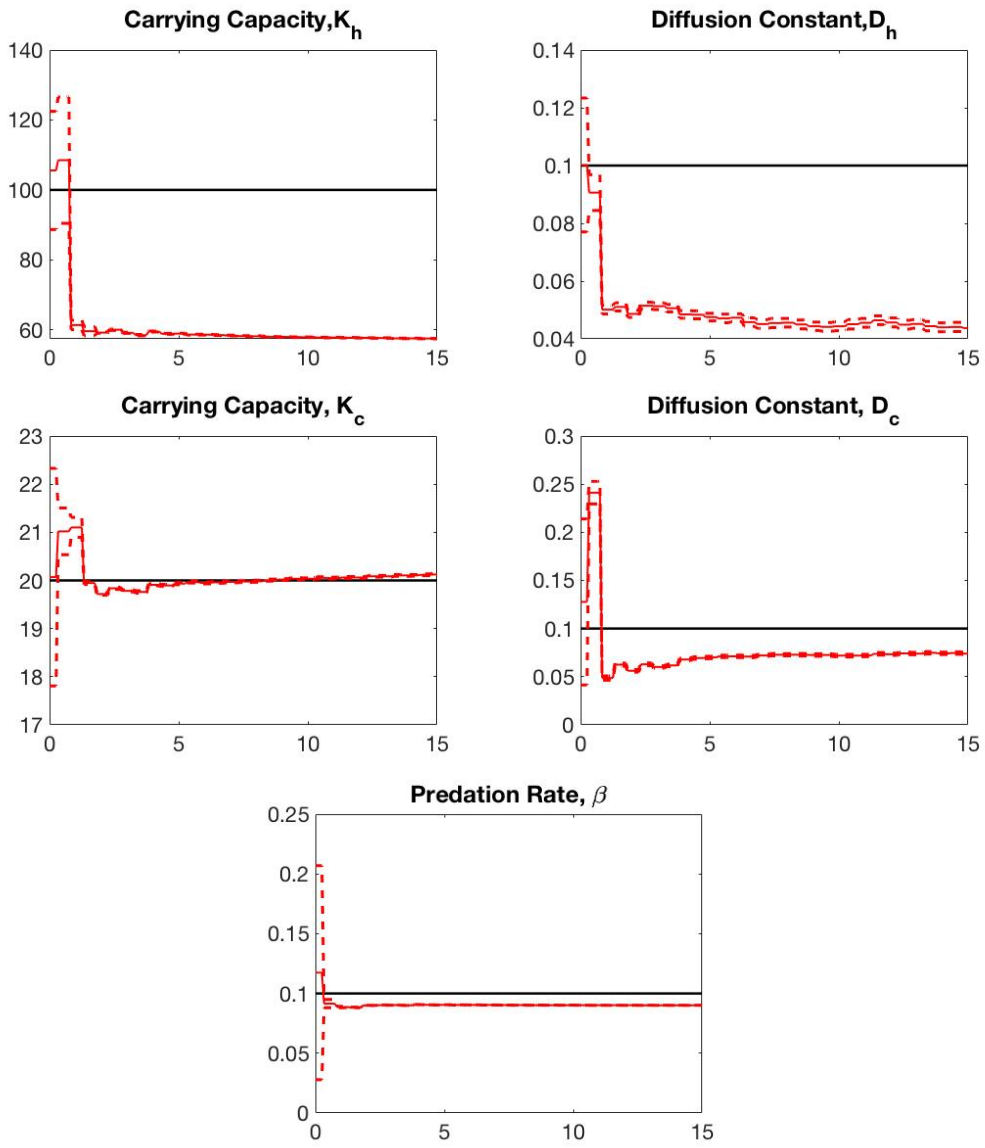


Figure 4.5: This figure shows the estimated parameters for K_h, D_h, K_c, D_c, and β. The black line represents the true solution of the parameter. The solid red line represents the estimated value of the parameter over time. The dotted red lines are two standard deviations of error about the estimated parameter. The true and estimated parameter values are shown in Table 4.5.

4.4.3 Integrodifference: Single-species

Now, we will observe the integrodifference single-species ensemble Kalman filter results. We will be estimating carrying capacity and mortality fro each species. First, we will look at the single-

species Atlantic cod. In Figure 4.6, the Atlantic cod ensemble mean follows the same pattern as the true solution, which confirms that our filter is doing what it is supposed to. The graphs for the estimated parameters K_c and μ_c in Figure 4.6 are as follows. The true parameters are 50 for the carrying capacity and 0.2 for the mortality rate. In Figure 4.6, we can see that the filter finds the parameters K_c and μ_c to be 41.0108 and 0.1758 respectively. In Table 4.7, the relative error of the estimated parameters are given. The relative error for $K_c = 0.1798$ and the relative error for $\mu_c = 0.121$. The estimated parameters K_c and μ_c are an underestimate of the true parameters.

Now, we will look at the Kalman filter results for the Atlantic herring integrodifference model. In Figure 4.7, the Atlantic herring ensemble mean follows the same pattern as the true solution, which confirms that our filter is doing what it is supposed to. The graphs for the estimated parameters K_h and μ_h in Figure 4.7 are as follows. The true parameters are 150 for the carrying capacity and 1.5 for the mortality rate. The true estimates are modeled by the black line in Figure 4.7. In Figure 4.7, we can see that the filter finds the parameters K_h and μ_h to be 94.141 and 0.3724 respectively. In Table 4.8, the relative error of the estimated parameters are given. The relative error for $K_h = 0.3724$ and the relative error for $\mu_h = 0.6879$. The estimated parameters K_h and μ_h are an underestimate of the true parameters.

Meaning	*Parameter*	*Herring Value*	*Cod Value*
Mortality rate	μ_i	1.5	0.2
Growth rate scale	α_i	0.5	10
Velocity magnitude	v_i	1	1
Carrying capacity	K_i	150	20

Table 4.6: Parameter values used in computing single-species integrodifference model solutions, where parameters we estimate are highlighted in red.

Parameter	True Value	Estimated Value	Relative Error
K_c	50	41.0108	0.1798
μ_c	0.2	0.1758	0.121

Table 4.7: Relative error for single-species integrodifference Atlantic cod.

Parameter	True Value	Estimated Value	Relative Error
K_h	150	94.141	0.3724
μ_h	1.5	0.4681	0.6879

Table 4.8: Relative error for single-species integrodifference Atlantic herring.

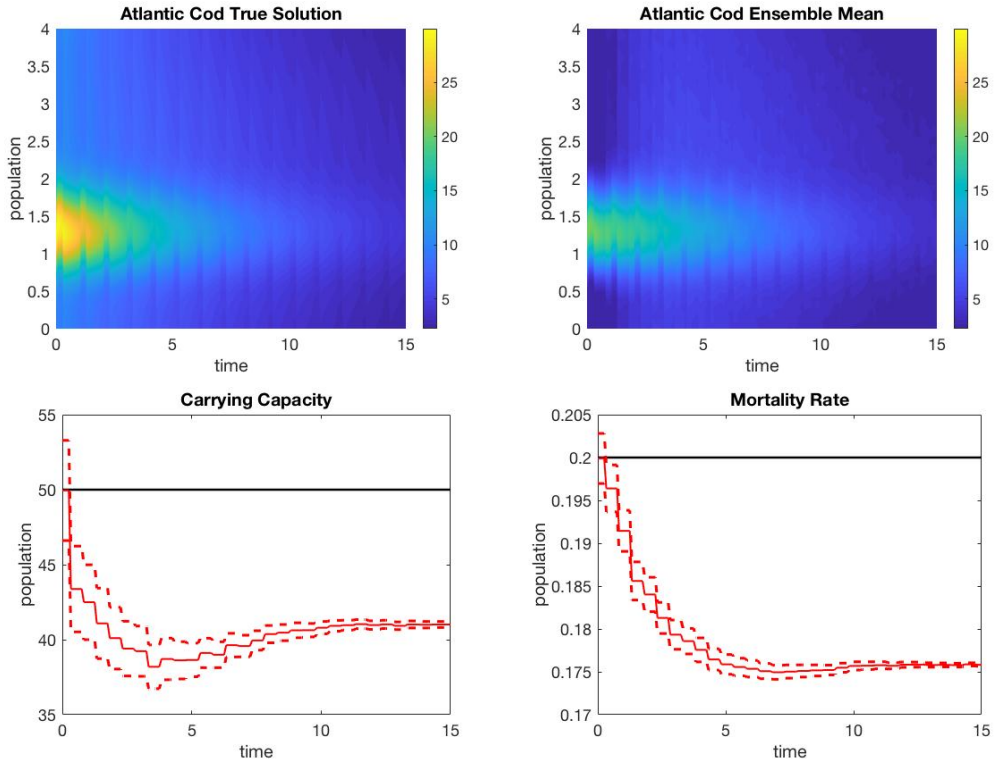


Figure 4.6: The true solution and the estimated mean of the filter ensembles of the single-species model use (3.1) and the initial conditions (3.6) for Atlantic cod. The black line represents the true solution of the parameter. The solid red line represents the estimated value of the parameter over time. The dotted red lines are two standard deviations of error about the estimated parameter. The true and estimated parameter values are shown in Table 4.7.

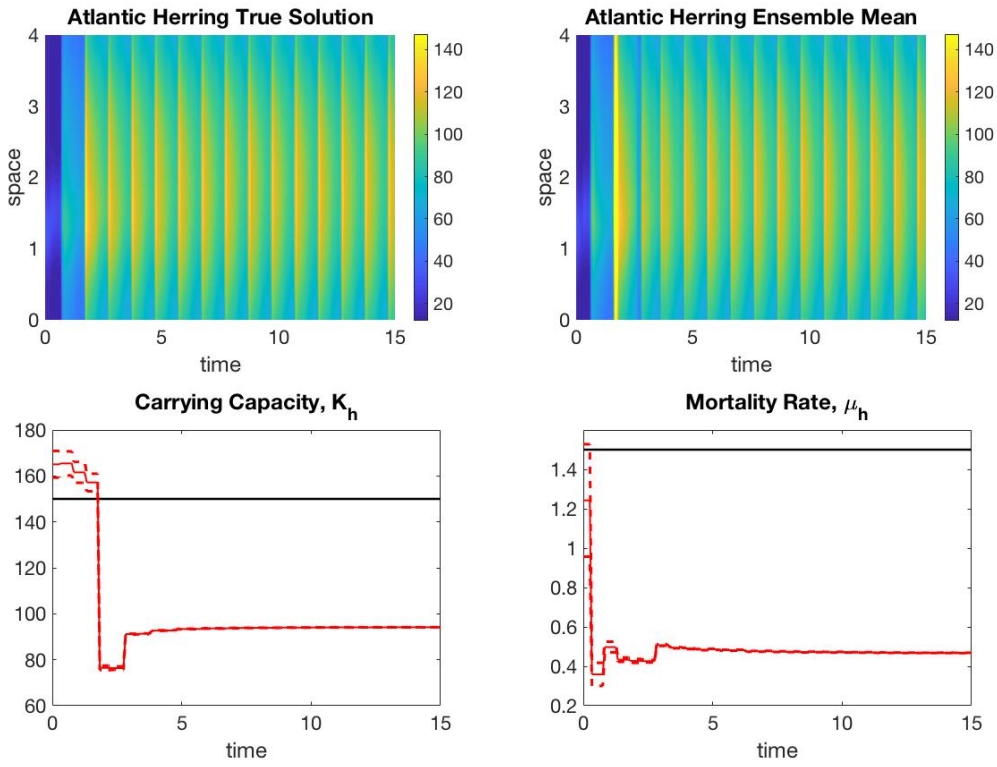


Figure 4.7: The true solution and the estimated mean of the filter ensembles of the single-species model use (3.1) using the initial conditions (3.5) and (3.6) for Atlantic herring and Atlantic cod. The black line represents the true solution of the parameter. The solid red line represents the estimated value of the parameter over time. The dotted red lines are two standard deviations of error about the estimated parameter. The true and estimated parameter values are shown in Table 4.8.

4.4.4 Integrodifference: Multi-species

Now, that we have seen the ensemble Kalman filter estimates for the single-species integrodifference models, we can look at the multi-species parameter estimations. We will be estimating carrying capacity and mortality rate for each species, as well as β and ε. In Figures 4.8 and 4.9, the Atlantic herring and Atlantic cod ensemble means follow the same patterns as the true solutions, which confirms that our filter is doing what it is supposed to. The graphs for the estimated parameters K_h, K_c, μ_h, μ_c, β, and ε in Figure 4.10 are as follows. The true parameters are 150 for K_h, 1.5 for μ_h, 25 for K_c, 0.5 for μ_c, and 0.2 for β. The true estimates are modeled by the black

line in Figure 4.7.The ensemble Kalman filter estimated the parameters K_h, μ_h, K_c, μ_c, and β to be 124.9427, 1.3687, 16.1985, 0.9395, and 0.0229 respectively. In Table 4.10, the relative error of the estimated parameters are given. The relative error for $K_h = 0.167$, $K_c = 0.352$, $\mu_h = 0.0875$, $\mu_c = 0.879$, and $\beta = 0.145$. Unfortunately, our ensemble Kalman was unable to estimate the parameter ε. The estimated parameters K_h, K_c and μ_h are an underestimate of the true parameters. The estimated parameters μ_c and β is an overestimate of the true parameter.

Meaning	*Parameter*	*Herring Value*	*Cod Value*
Mortality rate	μ_i	1.5	0.5
Growth rate scale	α_i	0.5	0.5
Velocity magnitude	v_i	1	1
Carrying capacity	K_i	150	25
Mortality from cod	β	0.02	N/A
Conversion efficiency	ε	N/A	0.01

Table 4.9: Parameter values used in computing multi-species integrodifference model solutions, where parameters we estimate are highlighted in red.

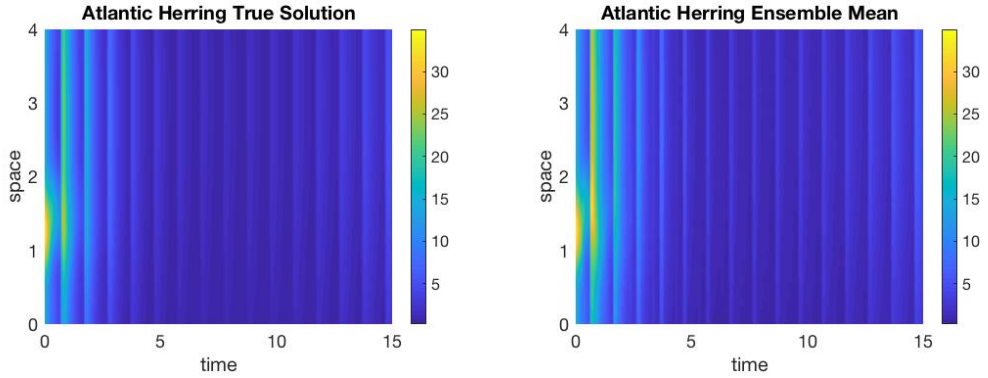


Figure 4.8: The true solution and the estimated mean of the filter ensembles of the multi-species model use (3.7) and (3.8) with the initial conditions (3.11) and (3.12) for Atlantic herring and Atlantic cod. Parameter values are shown in Table 4.9.

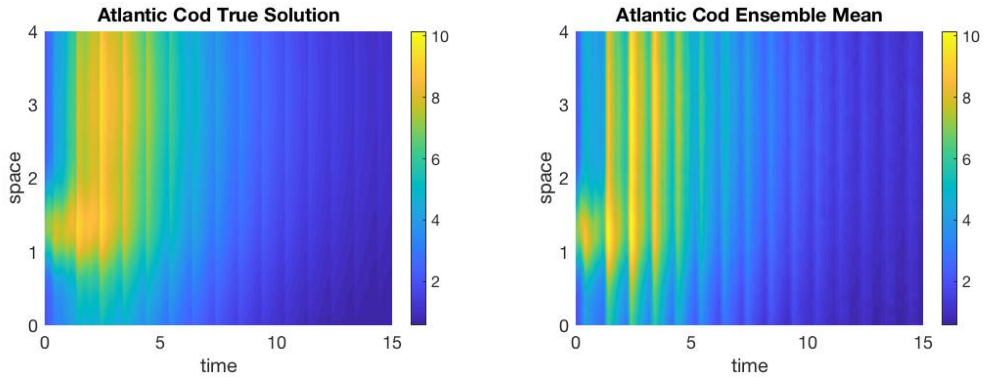


Figure 4.9: The true solution and the estimated mean of the filter ensembles of the multi-species model use (3.7) and (3.8) with the initial conditions (3.11) and (3.12) for Atlantic herring and Atlantic cod. Parameter values are shown in Table 4.9.

Parameter	True Value	Estimated Value	Relative Error
K_h	150	124.9427	0.167
μ_h	1.5	1.3687	0.0875
K_c	25	16.1985	0.352
μ_c	0.5	0.9395	0.879
β	0.02	0.0229	0.145

Table 4.10: Relative error for Atlantic herring and Atlantic cod ID multi-species model.

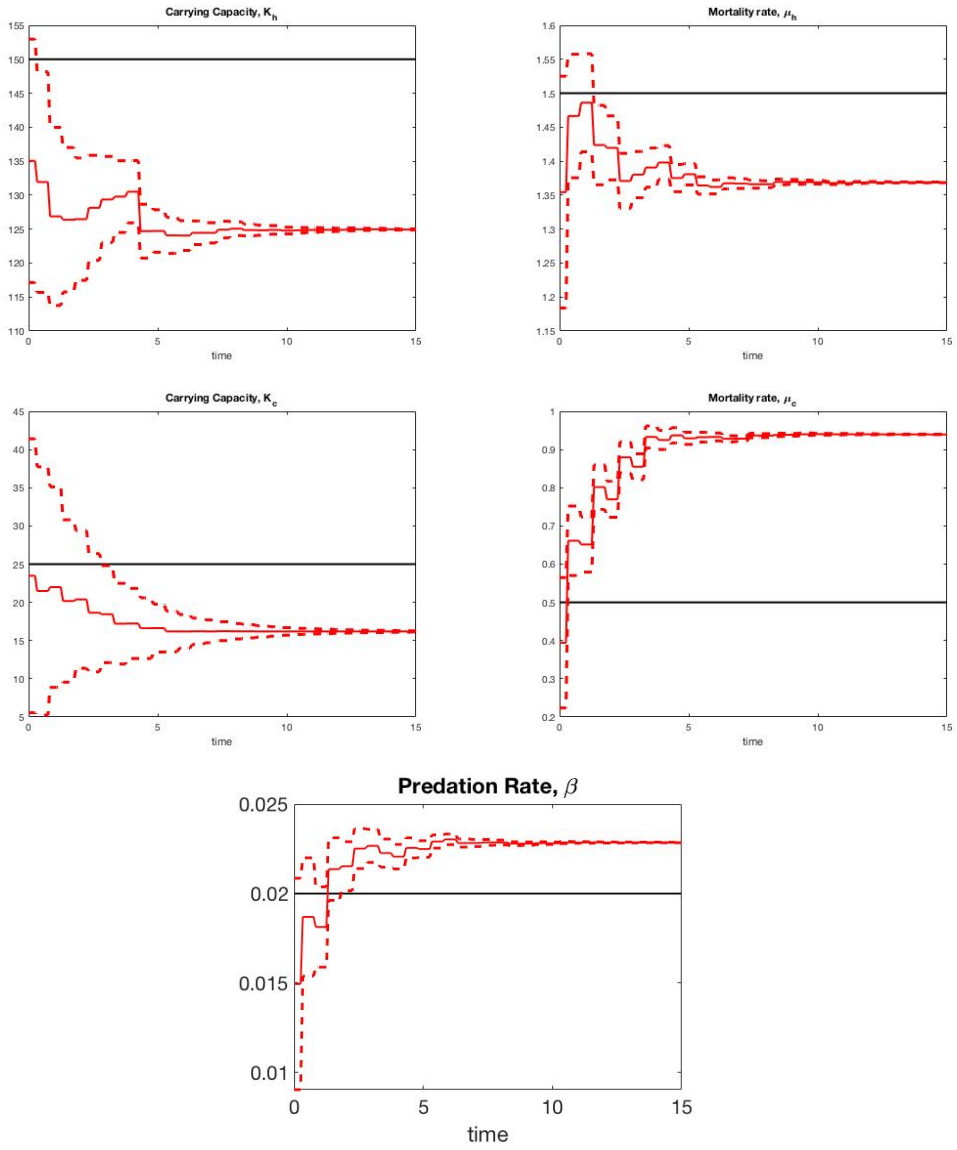


Figure 4.10: This figure shows the estimated parameters for K_h, μ_h, K_c, μ_c, and β. The black line represents the true solution of the parameter. The solid red line represents the estimated value of the parameter over time. The dotted red lines are two standard deviations of error about the estimated parameter. As time progresses, the standard deviations become smaller because our filter is more confident in finding the best estimated value for the provided data. The true and estimated parameter values are shown in Table 4.10.

Chapter 5

Model Comparison

5.1 Comparison Method

After both the single-species and multi-species forms of the integrodifference models and the PDE were all set, we determined a comparison technique. Our comparison technique utilizes synthetic data, as well as single-species and multi-species models. To compare the single-species models with the multi-species models, we first gather the synthetic data that was created and used in the PDE or integrodifference multi-species model. Then, we insert this data into appropriate single-species for either Atlantic herring or Atlantic cod, depending on which one is being compared. Then the single-species Kalman filter is run with the multi-species data. The posterior is taken, which is the last mean of the parameter estimates for the parameters that are being estimated (carrying capacity, K, and the diffusion coefficient, D for PDE and carrying capacity, K, and mortality, μ for integrodifference).

Now, we run the appropriate single-species model, either Atlantic herring or Atlantic cod and compare the output of this model to the true solution of the multi-species model. To compare we take the output of the regular single-species model and subtract the multi-species true solution and take the absolute value of them. This will give the absolute error. The true solution of the

multi-species model is generated from plugging in all of the true parameters into the multi-species.

5.2 PDE Comparison

In Figure 5.1, the PDE model comparison for Atlantic herring, it is evident that the population of Atlantic herring in the single-species model is greater than in the multi-species model. In the single-species model, the Atlantic herring grow during their spawning season, which is seen in the sharp increases and decreases of the graph. The multi-species true solution has less Atlantic herring, presumably because of the Atlantic cod eating them. Therefore, the population of Atlantic herring does not grow as much in the multi-species true solution. The absolute error shows this discrepancy.

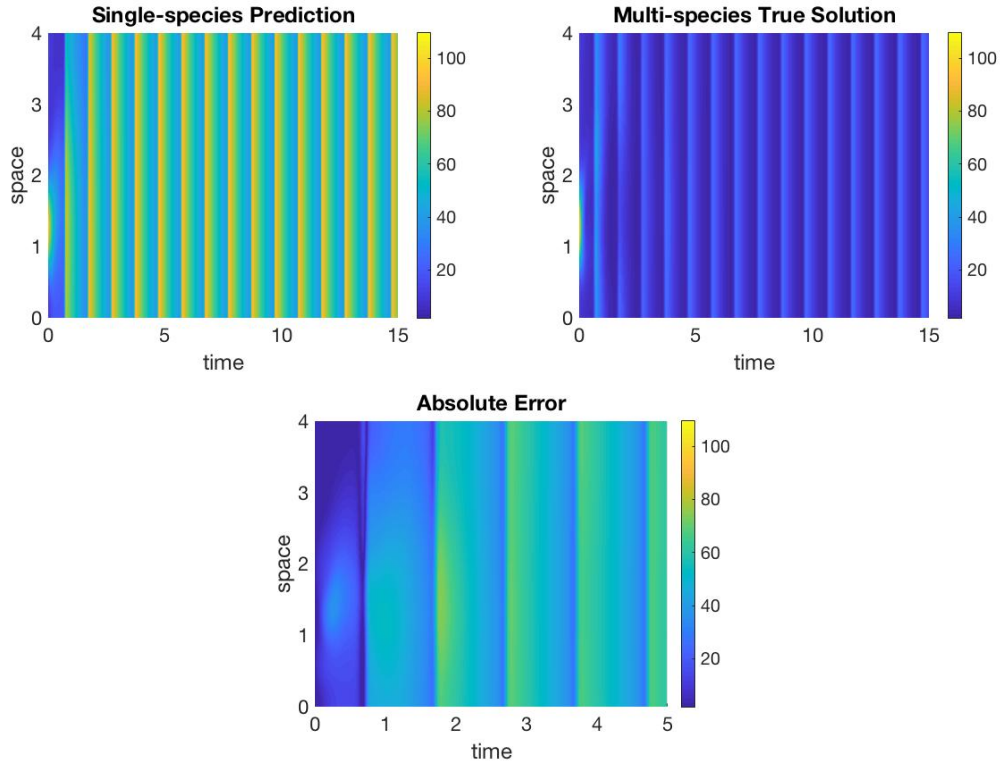


Figure 5.1: This is the comparison for the single-species and multi-species Atlantic herring PDE model using the equation (2.31) and initial condition (2.33).

In Figure 5.2, the PDE model comparison for Atlantic cod, the single-species model seems to grow less than the multi-species true solution does. In the single-species solution, it is evident through the sharp increase and decreases that the Atlantic cod population is growing during the spawning seasons. However, in the multi-species true solution, it appears that the population is growing more, presumably due to consumption of the Atlantic herring. In the absolute error, we see the discrepancy in the population growth.

5.3 Integrodifference Comparison

In Figure 5.3, the integrodifference Atlantic herring comparison, the single-species model allows for the population of herring to grow and recover, whereas the multi-species does not. In the

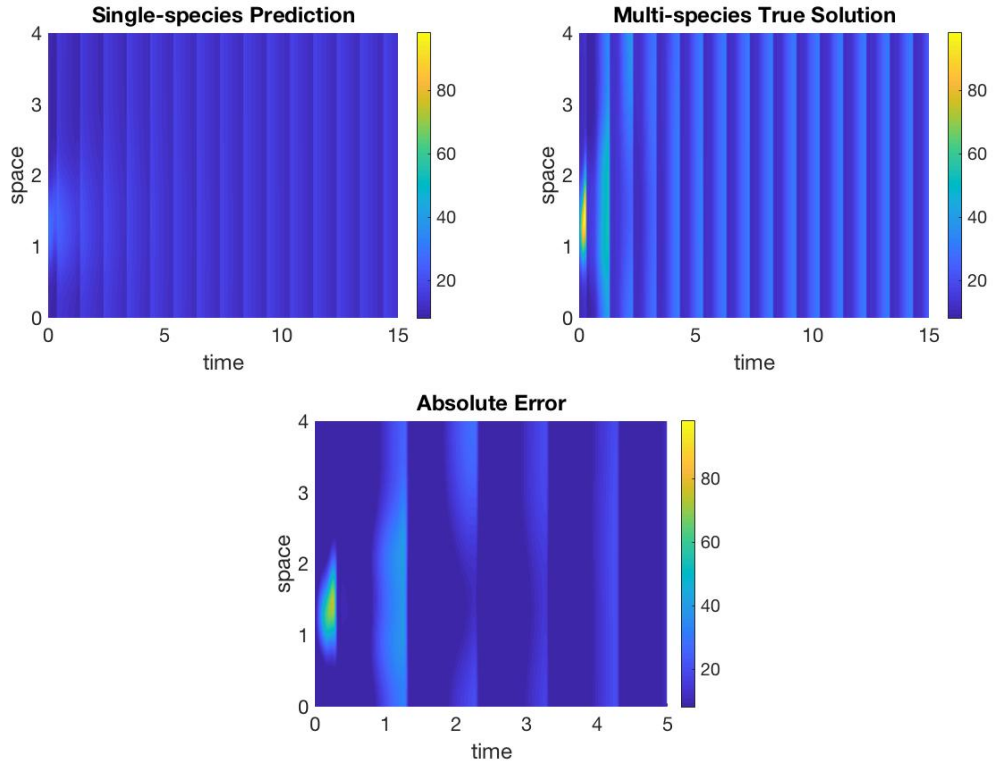


Figure 5.2: This is the comparison for the single-species and multi-species Atlantic cod PDE model using the equation (2.31) and initial condition (2.34).

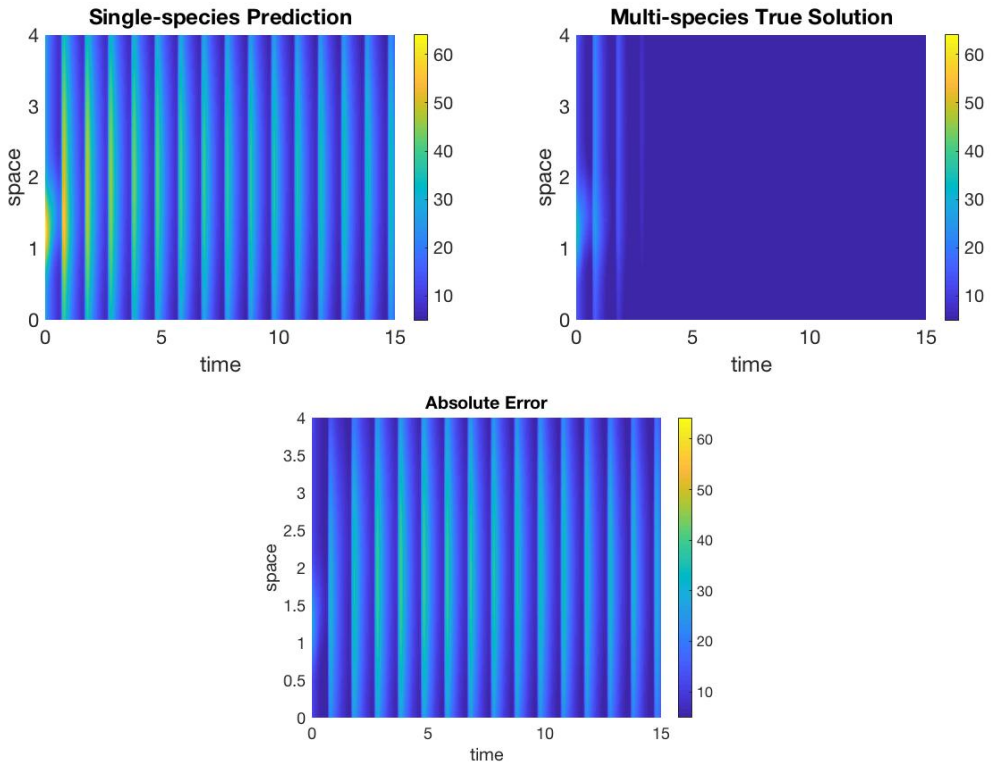


Figure 5.3: This is the comparison for the single-species and multi-species Atlantic herring integrodifference model using the equation (3.1) and initial condition (3.5).

single-species prediction, the sharp increases and decreases represent the population growth during spawning seasons. The population decreases when the fish are not spawning, but it can recover when they start to spawn again. This is not seen in the multi-species true solution. When there is the interaction between the Atlantic cod and the Atlantic herring, the Atlantic herring are being eaten by the Atlantic cod and dying out. The population is not able to recover because they are being continuously eaten. In the absolute error, there is less error initially because the initial conditions are the same. The first couple years also have less error because the population of cod in the multi-species model is able to recover up until around year 3. Then the herring die out. The single-species model seems to be overestimating the population of Atlantic herring.

In Figure 5.4, the integrodifference Atlantic cod comparison, observe that in the single-species prediction the Atlantic cod stay more between the first and second spaces. In the multi-species

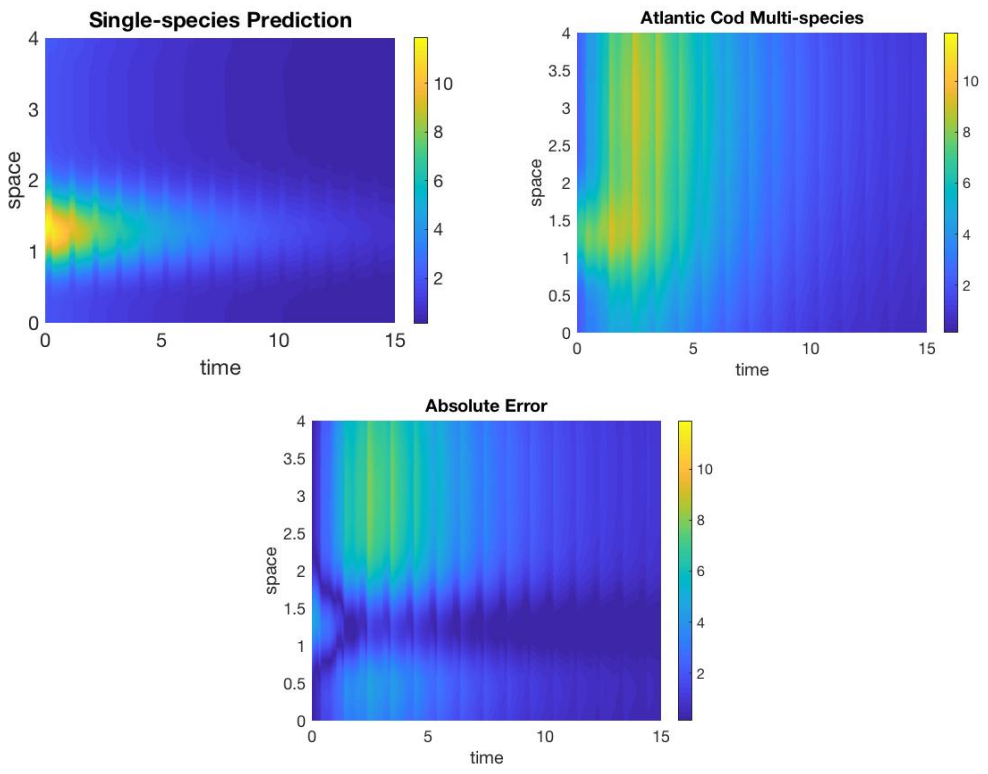


Figure 5.4: This is the comparison for the single-species and multi-species Atlantic cod integrod-ifference model using the equation (3.1) and initial condition (3.6).

true solution, the Atlantic cod are spreading to the third and fourth spaces, as well as the spaces between zero and one as eat the Atlantic herring. This is easily seen in the absolute error. The most discrepancy between the single-species prediction and the multi-species true solution occurs along the spaces three and four. In spaces between one and two, the single-species model estimates the population with less discrepancy with the multi-species true solution. However, in those outer spaces, the single-species model does not account for the Atlantic cod following the Atlantic herring.

Chapter 6

Discussion

6.1 Summary

The goal of this paper is to compare single-species and multi-species models in the context of NOAA Fisheries stock assessments. We create single and multi-species PDE and integrodifference models for the Atlantic cod and Atlantic herring populations in the Georges Bank region. Many of the parameters used in population modeling are poorly known for the Atlantic cod and Atlantic herring due to difficulty of measurement. To account for this, we use an ensemble Kalman filter to estimate carrying capacity, predation rate, and conversion efficiency for both model types as well as the diffusion coefficients in the PDE models and mortality rates in the integrodifference models.

To compare the single-species and multi-species models, we generate synthetic data by adding noise to the output of each type of multi-species model and use this as our observation data in the ensemble Kalman filter for the single-species models of the same type. We then run the single-species model with the parameters estimated in the filter and take the absolute error of the single-species model prediction and the true solution used to generate the synthetic data.

6.2 Interpretation of Results

Between the two types of models, the PDE estimates true parameter values for carrying capacity and predation rate better than the integrodifference, though both types of model estimate the predation rate with a relative error of less than 0.2. For the Atlantic cod, error in carrying capacity is almost entirely from underestimation rather than overestimation. This pattern is less clear for the Atlantic herring.

For the Atlantic cod, the multi-species PDE model estimates all shared parameters more closely than the single-species. For the integrodifference, the single-species estimates all shared parameters better than the multi-species. For the herring, the opposite is true. The single-species PDE model estimates parameters more closely than the multi-species for the Atlantic herring, but the multi-species integrodifference estimates more closely than the single-species.

We use a variety of ranges for the prior distributions of the parameters in the ensemble Kalman filters, and the estimated parameters result in very similar model predictions for all of the models. In the comparisons, single-species models overestimated the Atlantic herring and underestimated the Atlantic cod for both the PDE and the integrodifference models.

6.3 Future Work

There are still many interesting questions relating to this project that can be addressed in future studies. Here we discuss these questions as well as suggested directions for further exploration.

We were provided with almost 50 years of survey data by the NEFSC including measurements such as fish abundance, water temperature, season, and year. We initially planned to use this data in our ensemble Kalman filter after analyzing the synthetic data, but we found that the data was too jagged and our filter did not respond well. In order to utilize the data, we would need to find some way to make it smoother. One option suggested by a NEFSC employee would be to analyze the data on a log scale. Additionally, the NEFSC collects data twice per year, and we

mimicked this in our synthetic data. It is possible that more frequent observations could improve our filter estimates for some parameters. This could be tested by generating synthetic data more frequently than twice per year, running the ensemble Kalman filter, and comparing the relative errors of parameter estimates using both more frequent and less frequent data.

There are a variety of parameters in the models we developed, and not all of them may be able to be well-estimated using the filter given the sparsity of the data. We could perform sensitivity analysis [20] to see which subset of parameter values in each model type most affects the model output and focus on estimating those, and we could set the least sensitive parameters to constant values.

There are many options for modeling population dynamics such as growth and predation in both the PDE and integrodifference as well as many options for kernels in the integrodifference models. One option we considered but did not attempt was using a Ricker [17] or Beverton-Holt [13] growth function instead of the logistic. A Laplace kernel could also be used instead of a Gaussian kernel in the integrodifference model [21]. A variety of functions could be used instead of the versions used in this paper to represent velocity and birth rate. In the integrodifference model, some of the dynamics we placed outside the integral could be edited and included in the integrand.

Additionally, the PDE model does not account well for schooling, which is exhibited strongly by the Atlantic herring. Incorporating a feature to model this behavior could be significant to predicting the Atlantic herring's population, and it could counter the tendency of the PDE model to predict quick decreases in population due to diffusion which may not be prevalent in the ecosystem.

All of the models in this paper examine either one or two species of fish. There are, of course, many more species of fish in Georges Bank whose behavior may be relevant to Atlantic cod and Atlantic herring populations. For example, the spiny dogfish and silver hake compete with the Atlantic cod to eat the Atlantic herring, but they are also eaten by the Atlantic cod.

One could expand on this project by developing a model for three or more of these species,

generating data with the model including the most species, and repeating the comparison in this paper for each model using fewer species. For example, a three-species model could be used to generate data which could be compared against the species-pair model and single-species models developed in this paper.

Atlantic herring and Atlantic cod both have spawning ranges. While the models in this paper account for a spawning season with migration before and after, they all assume that spawning occurs at the same rate at every location. This may not be an accurate representation of fish spawning. One could easily add a specification that spawning occur only within a certain spatial range.

Additionally, the models in this paper examine only one dimension. Georges Bank is a three-dimensional region, and fish will at times move towards or away from the shore or to deeper or shallower water to find a more desirable temperature instead of moving directly along the coast. The models could be expanded to account for three dimensions, but the process may be very complex.

The model comparisons used give an absolute error, but this paper does not include a maximum allowable error. If a maximum error were chosen, they could be used to provide a definitive assessment of whether the models developed here show a significant improvement from single-species to multi-species or whether single-species are sufficient.

References

[1] J. Beddington, D. Agnew, and C. Clark, "Current Problems in the Management of Marine Fisheries," *Science*, vol. 316, no. 5832, pp. 1713 – 1716, 2007.

[2] "Atlantic Cod," https://www.fisheries.noaa.gov/species/atlantic-cod, 2018.

[3] A. Myers, N. Hutchings, and N. Barrowman, "Hypotheses for the decline of cod in the North Atlantic," *Marine Ecology Progress Series*, vol. 138, no. 293-308, 1996.

[4] R. O'Boyle, C. Francis, N. Hall, and N. Klaer, "SARC 54 PANEL SUMMARY REPORT," June 2012.

[5] J. Link, R. Gamble, and M. Fogarty, "An Overview of the NEFSC's Ecosystem Modeling Enterprise for the Northeast US Shelf Large Marine Ecosystem: Towards Ecosystem-based Fisheries Management," October 2011.

[6] "NEFSC Toolbox," http://nft.nefsc.noaa.gov/Comparing.html, May 2016.

[7] E. Olsen, G. Fay, S. Gaichas, R. Gamble, S. Lucey, and J. Line, "Ecosystem Model Skill Assessment. Yes We Can!" *Plos One*, vol. 11, no. 1, 2016.

[8] P. Politics, J. Galbraith, P. Kostovick, and R. Brown, "Northeast Fisheries Science Center Bottom Trawl Survey Protocols for the NOAA Ship Henry B. Bigelow," 2014, Northeast Fisheries Science Center Reference Document.

[9] H. Bigelow and W. Schroeder, *Fishes of the Gulf of Maine*. US Government Printing Office, 1953, vol. 53.

[10] "Atlantic Herring," https://www.fisheries.noaa.gov/species/atlantic-herring, 2018.

[11] K. Kanwit, "Atlantic Herring Tagging: Insights into Migration and Movement," 2006, Northeast Fisheries Science Center Reference Document.

[12] M. Morin, "Movement of Atlantic Cod in And Among the Western Gulf of Maine Rolling Closures as Determined Through Mark and Recapture," Master's thesis, University of New Hampshire, 2000.

[13] L. Edelstein-Keshet, *Mathematical Models in Biology*, 1st ed. SIAM: Society for Industrial and Applied Mathematics, 2005.

[14] R. Haberman, *Elementary Applied Partial Differential Equations*, 2nd ed. Pentice-Hall, INC, 1987.

[15] R. Leveque, *Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems*. Society for Industrial and Applied Mathematics, 2007, vol. 98.

[16] N. Marculis and R. Lui, "Modeling the Biological Invasion of Carcinus maenas," *Journal of Biological Dynamics*, vol. 10, no. 1, pp. 140–163, 2015.

[17] M. Kot and W. Schaffer, "Discrete-Time Growth-Dispersal Models," *Mathematical Biosciences*, vol. 80, no. 109-136, 1986.

[18] A. Arnold and A. Lloyd, "An approach to periodic, time-varying parameter estimation using nonlinear filtering," *To appear in Inverse Problems*, 2018.

[19] A. Arnold, D. Calvetti, and E. Somersalo, "Parameter estimation for stiff deterministic dynamical systems via ensemble Kalman filter," *Inverse Problems*, vol. 30, no. 10, 2014.

[20] D. M. Hamby, "A review of techniques for parameter sensitivity analysis of environmental models," *Environmental Monitoring and Assessment*, vol. 32, no. 2, pp. 135–154, 1994.

[21] M. Neubert and M. Lewis, "Dispersal and Pattern Formation in a Discrete-Time Predator-Prey Model," *Theoretical Population Biology*, vol. 48, pp. 7–43, 1995.

Appendix A

Matlab Code

A.1 Partial Differential Equation Model Codes

```
%Single species PDE model main file for Atlantic Cod

clear;clc;close all;
%% Define parameters
cod_parameters=load('Cod_parameters.txt'); % load in .txt file
K_c = cod_parameters(1); %Carrying capacity
mu_c = cod_parameters(2); %Fishing mortality
D_c = cod_parameters(3); %Diffusion constant
a_c = cod_parameters(4); %Advection constant
mag_c = cod_parameters(5); % scaling factor for velocity

%% Time discretization
time_space = load('time_space_parameters.txt'); % load .txt file
k = time_space(1); % time step
t_end = 3; % end time
```

```
time  = 0:k:t_end; % time vector
ntime = length(time); % length of time vector

k2 = .5*k^2; % bootstrapping time step
t_bar = 0:k2:k; % bootstapping time vector
nt_bar = length(t_bar); % length of bootstapping time vector

%% Spatial discretization
h = time_space(3); % space step
x_end = time_space(4); % end space
space = 0:h:x_end; % space vector
nspace = length(space); % length of space vector
M = nspace;

%% Define common constants
r = k/(2*h*h);
d_con = D_c*r; %Constant arising from the Crank-Nicolson

%% Call the birth rate function alpha
[alpha_c,t_period] = AlphaC_SS(k,t_end,a_c); %Returns a vector alpha_c containing
%the value of the growth rate at each time

%% Define relevant matrices

%Coefficient matrix due to Crank-Nicolson for V^n+1
CN_left = -d_con*diag(ones(M-1, 1), 1) + 2*d_con*diag(ones(M,1)) ...
    - d_con*diag(ones(M-1, 1), -1);
CN_left(1,2) = 2*CN_left(1,2); %Impose zero-slope boundary condition
CN_left(end,end-1) = 2*CN_left(end,end-1);
%Identity matrix to represent the time approx for V^n+1
Forward_time_left = diag(ones(M,1));
```

```
47
48 %Total matrix with the coefficients of the solution at time n+1
49 M_lhs = Forward_time_left + CN_left;
50
51 %Coefficient matrix due to CN for V^n
52 CN_right = d_con*diag(ones(M-1, 1), 1) - 2*d_con*diag(ones(M,1)) ...
53     + d_con*diag(ones(M-1, 1), -1);
54 %Impose zero-slope boundary condition
55 CN_right(1,2) = 2*CN_right(1,2);
56 CN_right(end,end-1) = 2*CN_right(end,end-1);
57
58 %Identity matrix representing the V^n term due to the time approx
59
60 Forward_time_right = diag(ones(M,1));
61
62 M_rhs_base = Forward_time_right + CN_right;
63
64
65 %% Bootstrapping matrices
66 r2 = k2/(2*h*h);
67 d_con2 = D_c*r2; %Constant arising from the Crank-Nicolson
68
69 %Coefficient matrix due to Crank-Nicolson for V^n+1
70
71 CN_left2 = d_con2*CN_left/d_con;
72 %Identity matrix to represent the time approx for V^n+1
73 Forward_time_left2 = diag(ones(M,1));
74
75 %Total matrix with the coefficients of the solution at time n+1
76 M_lhs2 = Forward_time_left2 + CN_left2;
77
```

```
%Coefficient matrix due to CN for V^n

CN_right2 = d_con2*CN_right/d_con;

%Identity matrix representing the V^n term due to the time approx

Forward_time_right2 = diag(ones(M,1));

M_rhs2_base = Forward_time_right2 + CN_right2;


%% Initial and boundary conditions

IC = (space.*(3-space)).^6.*normpdf(space,0,1)+10;
%g0_row = zeros(1, ntime); %Have not yet found g0 and g1
%g1_row = zeros(1, ntime);

%% Define the solution matrix

V_save = zeros(M, ntime);  %Creates a matrix to store values for V in
V_save(:,1) = IC'; %Changes the first column to the initial conditions


%% Define the Bootstrapping solution matrix

V_bar = zeros(M,nt_bar); %Matrix to store the bootstrapping solutions in
V_bar(:,1) = IC'; %Sets same initial condition for bootstrapping matrix as
%in the other matrix


%% Update by time step
```

```
for n=1:ntime-1

    %run('velocity_c.m')
    v_c = velocity_c(n,k,t_end);
    a_con = v_c*k/h; %Constant arising from upwinding
    UW_right = a_con*diag(ones(M-1, 1), -1) - a_con*diag(ones(M,1));

    %Impose zero-slope boundary conditions
    UW_right(1,:) = 0;
    UW_right(end,:) = 0;
    %Define the reaction term
    React_right = (k*alpha_c(n) - k*mu_c)*diag(ones(M,1));

    a_con2 = v_c*k2/h; %Constant from upwinding with boostrap matrices
    UW_right2 = a_con2*diag(ones(M-1, 1), -1) - a_con2*diag(ones(M,1));

    %Impose zero-slope boundary conditions
    UW_right2(1,:) = 0;
    UW_right2(end,:) = 0;
    React_right2 = (k2*alpha_c(n) - k2*mu_c)*diag(ones(M,1));

    %Assemble total matrices

    M_rhs = M_rhs_base + UW_right + React_right;
    M_rhs2 = M_rhs2_base + UW_right2 + React_right2;

    %Implement bootstrapping when overall time loop is on first iteration
    if n == 1
        for m=1:nt_bar-1

            V_bar(:,m+1) = M_rhs2*V_bar(:,m) - ...
```

```
140                 (k2*alpha_c(n)/K_c)*V_bar(:,m).*V_bar(:,m);
141
142             V_bar(:,m+1) = M_lhs2\V_bar(:,m+1);
143         end
144
145             V_save(:,2) = M_rhs*V_bar(:,end) - ...
146                 (k*alpha_c(n)/K_c)*V_bar(:,end).*V_bar(:,end);
147
148             V_save(:,2) = M_lhs\V_save(:,2);
149
150     else
151         %When you're done with bootstrapping, update normally
152
153         V_save(:,n+1) = M_rhs*V_save(:,n) - ...
154             (k*alpha_c(n)/K_c)*V_save(:,n).*V_save(:,n);
155
156         V_save(:,n+1) = M_lhs\V_save(:,n+1);
157
158     end
159
160 end
161
162
163 %% Plot
164 figure(1);
165 surf(time,space,V_save)
166 %view(2);
167 xlabel('time'); ylabel('space'); zlabel('\rho(x,t)');
168 title('Atlantic Cod');
169 set(gca,'FontSize',20);
170 shading interp
```

```
171 colorbar
```

```
1  % Multi-species PDE model for Atlantic cod and Atlantic herring
2
3  clear;clc;close all;
4
5  %% Define parameters for cod
6  parameters=load('Multispecies_parameters.txt'); % load values from .txt file
7  K_c = parameters(1); %Carrying capacity
8  mu_c = parameters(2); %Fishing mortality
9  D_c = parameters(3); %Diffusion constant
10 a_c = parameters(4); %Alpha scaling factor
11 mag_c = parameters(5); % Velocity scaling term
12
13 %% Define parameters for herring
14 K_h = parameters(6); %Carrying capacity
15 mu_h = parameters(7); %Fishing mortality
16 D_h = parameters(8); %Diffusion constant
17 a_h = parameters(9); %Alpha scaling factor
18 mag_h = parameters(10); %Velocity magnitude
19
20 %% Define predator-prey parameters
21 beta = parameters(11); %Predation rate
22 epsilon = parameters(12); %Conversion efficiency
23
24 %% Time discretization
25 time_space = load('time_space_parameters.txt');
26 k = time_space(1); % time step
27 t_end = time_space(2); % end time
28 time  = 0:k:t_end; % time vector
```

```
ntime = length(time); % length of time vector

k2 = .5*k^2; % bootstrapping time step
t_bar = 0:k2:k; % bootstrapping time vector
nt_bar = length(t_bar); % length of bootstrapping time vector

%% Spatial discretization
h = time_space(3); % space step
x_end = time_space(4); % end space
space = 0:h:x_end; % space vector
nspace = length(space); % length of space vector
M = nspace;

%% Define common constants
r = k/(2*h*h);
d_con_c = D_c*r; %Constant arising from the Crank-Nicolson for cod
d_con_h = D_h*r; %Constant arising from the Crank-Nicolson for herring

%% Call birth rate functions
[alpha_c,¬] = AlphaC_SS(k,t_end,a_c); %Birth rate vector for cod
[alpha_h,t_period] = AlphaH_AA(k,t_end,a_h); %Birth rate vector for herring

%% Define relevant matrices

%Coefficient matrix due to Crank-Nicolson for V^n+1
CN_quadrant = diag(ones(M-1, 1), 1) - 2*diag(ones(M,1)) + diag(ones(M-1, 1), -1);

%Impose zero-slope boundary conditions
CN_quadrant(1,2) = 2*CN_quadrant(1,2);
CN_quadrant(end,end-1) = 2*CN_quadrant(end,end-1);

```

```
CN_right = [d_con_h*CN_quadrant,zeros(M);zeros(M),d_con_c*CN_quadrant];
CN_left = -1*CN_right;

%Identity matrix to represent the time approx for V^n+1
Forward_time_left = diag(ones(2*M,1));

%Total matrix with the coefficients of the solution at time n+1
M_lhs = Forward_time_left + CN_left;

%Identity matrix representing the V^n term due to the time approx

Forward_time_right = diag(ones(2*M,1));

M_rhs_base = Forward_time_right + CN_right;

%% Bootstrapping matrices
r2 = k2/(2*h*h); %Redefine constants with the bootstrapping time step
d_con2_c = D_c*r2;
d_con2_h = D_h*r2;

%Coefficient matrix due to Crank-Nicolson for V^n+1

CN_right2 = [d_con2_h*CN_quadrant,zeros(M);zeros(M),d_con2_c*CN_quadrant];
CN_left2 = -1*CN_right2;
%Identity matrix to represent the time approx for V^n+1
Forward_time_left2 = diag(ones(2*M,1));

%Total atrix with the coefficients of the solution at time n+1
M_lhs2 = Forward_time_left2 + CN_left2;

%Identity matrix representing the V^n term due to the time approx
```

```
91
92  Forward_time_right2 = diag(ones(2*M,1));
93
94  M_rhs2_base = Forward_time_right2 + CN_right2;
95
96  %% Alpha/carrying capcity and beta/epsilon matrices
97
98  %Matrices to represent interspecies term coefficients
99
100 BetaEpsilon = k*[-1*beta*eye(M),zeros(M);zeros(M),epsilon*eye(M)];
101 BetaEpsilon2 = k2*[-1*beta*eye(M),zeros(M);zeros(M),epsilon*eye(M)];
102
103 %% Initial and boundary conditions
104
105 IC1 = 5*(space.*(3-space)).^6.*normpdf(space,0,1)+10;
106 IC2 = (space.*(3-space)).^6.*normpdf(space,0,1)+10;
107
108 %% Define the solution matrix
109
110 V_save = zeros(2*M, ntime);  %Creates a matrix to store values for V in
111 V_save(1:M,1) = IC1'; %Changes the first column to the initial conditions
112 V_save(M+1:end,1) = IC2';
113
114 %% Define the Bootstrapping solution matrix
115
116 V_bar = zeros(2*M,nt_bar); %Creates a matrix to store bootstrapping values
117 V_bar(1:M,1) = IC1'; %Changes the first column to initial conditions
118 V_bar(M+1:end,1) = IC2';
119
120 %% Update by time step
121 for n=1:ntime-1
```

```
122
123     v_c = velocity_c(n,k,t_end);
124     v_h = velocity_h(n,k,t_end);
125
126     a_con_c = v_c*k/h; %Constant arising from upwinding
127     a_con_h = v_h*k/h;
128
129     %Define the matrix representing upwinding for a single species
130     UW_quadrant = diag(ones(M-1, 1), -1) - diag(ones(M,1));
131     %Impose zero-slope boundary conditions
132     UW_quadrant(1,:) = 0;
133     UW_quadrant(end,:) = 0;
134
135     %Combine the UW_quadrant matrices for each species into one matrix
136     UW_right = [a_con_h*UW_quadrant,zeros(M);zeros(M),a_con_c*UW_quadrant];
137
138     React_right = [(k*alpha_h(n) - k*mu_h)*diag(ones(M,1)),...
139         zeros(M);zeros(M),(k*alpha_c(n) - k*mu_c)*diag(ones(M,1))];
140
141     %Repeat the setup with the bootstrapping time step
142     a_con2_c = v_c*k2/h;
143     a_con2_h = v_h*k2/h;
144     UW_right2 = [a_con2_h*UW_quadrant,zeros(M);...
145         zeros(M),a_con2_c*UW_quadrant];
146
147     React_right2 = [(k2*alpha_h(n) - k2*mu_h)*diag(ones(M,1)),zeros(M);...
148         zeros(M),(k2*alpha_c(n) - k2*mu_c)*diag(ones(M,1))];
149
150     M_rhs = M_rhs_base + UW_right + React_right;
151     M_rhs2 = M_rhs2_base + UW_right2 + React_right2;
152
```

```
153     V_save_flip = [V_save(M+1:end,n);V_save(1:M,n)];
154     AlphaK = k*[(alpha_h(n)/K_h)*eye(M),zeros(M);zeros(M),...
155         (alpha_c(n)/K_c)*eye(M)];
156     AlphaK2 = k2*[(alpha_h(n)/K_h)*eye(M),zeros(M);zeros(M),...
157         (alpha_c(n)/K_c)*eye(M)];
158
159     %Implement bootstrapping when time loop is on first iteration
160     if n == 1
161         for m=1:nt_bar-1
162             V_bar_flip = [V_bar(M+1:end,m);V_bar(1:M,m)];
163             V_bar(:,m+1) = M_rhs2*V_bar(:,m) - AlphaK2*V_bar(:,m).*V_bar(:,m)...
164                 + BetaEpsilon2*V_bar(:,m).*V_bar_flip;
165
166             V_bar(:,m+1) = M_lhs2\V_bar(:,m+1);
167         end
168
169             V_save(:,2) = M_rhs*V_bar(:,end) - AlphaK*V_bar(:,end).*V_bar(:,end)...
170                 + BetaEpsilon*V_bar(:,end).*V_bar_flip;
171
172             V_save(:,2) = M_lhs\V_save(:,2);
173
174     else
175
176         %Continue using regular time step after bootstrapping
177
178         V_save(:,n+1) = M_rhs*V_save(:,n) - AlphaK*V_save(:,n).*V_save(:,n)...
179             + BetaEpsilon*V_save(:,n).*V_save_flip;
180
181         V_save(:,n+1) = M_lhs\V_save(:,n+1);
182
183     end
```

```
184
185 end
186
187 %% Split into Cod and Herring solutions
188
189 V_herring = V_save(1:M,:);
190 V_cod = V_save(M+1:end,:);
191
192 %% Plots
193 % Atlantic Herring
194 figure(1);
195 surf(time,space,V_herring)
196 %view(2);
197 xlabel('time'); ylabel('space'); zlabel('\rho_h(x,t)');
198 title('Atlantic Herring');
199 set(gca,'FontSize',20);
200 shading interp
201 colorbar
202 cbar_lims = caxis;
203
204 % Plot the Cod
205 figure(2);
206 surf(time,space,V_cod)
207 %view(2);
208 xlabel('time'); ylabel('space'); zlabel('\rho_c(x,t)');
209 title('Atlantic Cod');
210 set(gca,'FontSize',20);
211 shading interp
212 colorbar
213 caxis(cbar_lims);
```

```
% Velocity function for the Atlantic Cod
% Break up into 12 month time period, assumes final time is 1
% Assumes the model will begin in January

function v_c = velocity_c(n,k,t_end)

mag_c = 2; % velocity scaling term
b = 200;

%% Time discretization
time  = 0:k:t_end;
ntime = length(time);
t_period = round(1/k);

m = mod(n,t_period);

%% Region 1, January through the middle of spring movement
if 0 ≤ m && m ≤ (2.5*ntime)/(12*t_end)
    v_c = .5*(1+tanh(b*(time(m+1)-2/12)));
end

%% Region 2, Middle of spring movement through middle of spawning
if (2.5*ntime)/(12*t_end) < m && m ≤ (4*ntime)/(12*t_end)
    v_c = -.5*(-1+tanh(b*(time(m+1)-3/12)));
end

%% Region 3, Middle of spawning through middle of summer movement
if (4*ntime)/(12*t_end) < m && m ≤ (5.5*ntime)/(12*t_end)
    v_c = -.5*(1+tanh(b*(time(m+1)-5/12)));
end

```

```
%% Region 4, Middle of summer movement through end of year
if (5.5*ntime)/(12*t_end) < m && m ≤ (12*ntime)/(12*t_end)
    v_c = .5*(-1+tanh(b*(time(m+1)-6/12)));
end

% Scaling the velocity
v_c = mag_c*v_c;
```

```
% Velocity function for the Atlantic Cod
% Break up into 12 month time period, assumes final time is 1
% Assumes the model will begin in January

function v_h = velocity_h(n,k,t_end)

mag_h = 1; % velocity scaling term
b = 200;

%% Time discretization
time  = 0:k:t_end;
ntime = length(time);
t_period = round(1/k);

m = mod(n,t_period);

%% Region 1, January through the middle of spring movement
if 0 ≤ m && m ≤ (5*ntime)/(12*t_end)
    v_h = .5*(1+tanh(b*(time(m+1)-2/12)));
end

%% Region 2, Middle of spring movement through middle of spawning
```

```
if (5*ntime)/(12*t_end) < m && m ≤ (9*ntime)/(12*t_end)
    v_h = -.5*(-1+tanh(b*(time(m+1)-3/12)));
end

%% Region 3, Middle of spawning through middle of summer movement
if (9*ntime)/(12*t_end) < m && m ≤ (11*ntime)/(12*t_end)
    v_h = -.5*(1+tanh(b*(time(m+1)-5/12)));
end

%% Region 4, Middle of summer movement through end of year
if (11*ntime)/(12*t_end) < m && m ≤ (12*ntime)/(12*t_end)
    v_h = .5*(-1+tanh(b*(time(m+1)-6/12)));
end

v_h = mag_h*v_h;
```

```
% Creating the alpha function for Cod in the PDE model
% Gives a vector of alpha over time

function [alpha_c,t_period] = AlphaC_SS(k,t_end,a_c)

% Define time
time  = 0:k:t_end;
ntime = length(time);
t_period = 1/k;

% Define the mean and standard deviation of the distribution
sigma = 0.015;
mean_c = time(floor((4.5*t_period)/12));

```

```
15 t_0 = floor((2*ntime)/(12*t_end)); % first value
16 t_1 = floor((6*ntime)/(12*t_end)); % last value
17
18 alpha_c_year = zeros(1,t_period); %Initilalize a year-long vetor of zeros
19
20 for n=t_0:t_1
21 %Birth rate is a Gaussian during the spawining season
22     alpha_c_year(1,n) = (1/(sqrt(2*pi*sigma^2)))*exp(-((time(n)-mean_c)^2)/(2*sigma^2));
23 end
24
25 alpha_c = alpha_c_year;
26 num_full_reps = floor(ntime/t_period); %Number of full years in the time period
27
28 for j=1:num_full_reps-1
29     alpha_c = [alpha_c,alpha_c_year]; %Repeat year-long vector
30 end
31
32
33 %Add on remaining entries if the time is not a whole number of years
34 remaining_cols = rem(ntime,t_period);
35 alpha_h_add = alpha_c_year(:,1:remaining_cols);
36 alpha_c = [alpha_c,alpha_h_add];
37
38 % Multiply by a scaling term, a_c
39 alpha_c = a_c*alpha_c;
40
41 % Plot it
42 figure(999);
43 plot(time,alpha_c,'-b','LineWidth',3);
44 set(gca,'FontSize',20);
45 title('Herring Birth Rate, \alpha_h');
```

```
xlabel('Time');
ylabel('Birth Rate');
```

```
% Creating the alpha function for Herring in the PDE model
% Gives a vector of alpha over time

function [alpha_h,t_period] = AlphaH_AA(k,t_end,a_h)

%Define time
time  = 0:k:t_end;
ntime = length(time);
t_period = 1/k;

% Define the mean and standard deviation of the distribution
sigma = .045;
mean_h = time(floor((7.8*t_period)/12));

t_0 = floor((6*ntime)/(12*t_end)); % first value
t_1 = floor((11*ntime)/(12*t_end)); % last value

alpha_h_year = zeros(1,t_period); %Initialize a year-long vector of zeros

for n=t_0:t_1
%Birth rate is a Gaussian during the spawining season
    alpha_h_year(1,n) = (1/(sqrt(2*pi*sigma^2)))*exp(-((time(n)-mean_h)^2)/(2*sigma^2));
end

alpha_h = alpha_h_year;
num_full_reps = floor(ntime/t_period); %Number of full years in time vector
```

```
for j=1:num_full_reps-1
    alpha_h = [alpha_h,alpha_h_year]; %Repeat year-long vector for full years
end


%Add on additional columns if total time is not a whole number of years
remaining_cols = rem(ntime,t_period);
alpha_h_add = alpha_h_year(:,1:remaining_cols);
alpha_h = [alpha_h,alpha_h_add];

% Multiply by a scaling factor, a_h
alpha_h = a_h*alpha_h;

% Plot it
figure(998);
plot(time,alpha_h,'-b','LineWidth',3);
set(gca,'FontSize',20);
title('Herring Birth Rate, \alpha_h');
xlabel('Time');
ylabel('Birth Rate');
```

A.2 Integrodifference Model Codes

```
%% Main Integrodifference Model Log
clear; clc; close all;
%% Define Parameters
mu = .2; % Mortality rate

space = 0:.1:3; %Space vector, space step = .1 and final space = 3
len_space = length(space);
```

```
nnodes = length(space);

growth_rate_c = 2; %Growth rate scaling factor
carrying_capacity = 50;

time = 0:1/12:15; %time vector
len_time = length(time);
tnodes = length(time);
dt = 1/12; %Time step, 1/12 for the integrodifference models in this project

run('AlphaC.m') %Returns vector of values of the birth rate at each time

%% Construct final matrix
pop = zeros(length(time),length(space)); %initialize the population matrix
%Set first row equal to initial conditions
pop(1,:) = 2.5*(space(1:end).*(3-space(1:end))).^6.*normpdf(space(1:end),0,1)+10;

%% Construct coefficient (term outside of the integral)
coefficient = zeros(length(time),length(space)); %initialize

%% Construct loop
for i=1:length(time)-1
    %Update integral
   pop(i+1,:) = PopStepLog_gaussian(pop(i,:),space,alpha_c(i),...
       carrying_capacity,len_space);
   %Shift population from previous time step by the value of the shift
   %function
   coefficient(i+1,:) = shiftC(pop(i,:),i+1,dt,tnodes);
   %Multiply by the survival rate
   coefficient(i+1,:) = (exp(-mu*dt)).*coefficient(i+1,:);
   %Add previous population and new population
```

```
    pop(i+1,:) = coefficient(i+1,:) + pop(i+1,:);
end

%% Construct graph
figure(1);
surf(time, space, pop'); hold on;
shading interp
xlabel('time'); ylabel('space'); zlabel('population');
set(gca,'FontSize',20);
%legend('N_0(x)', 'N_1(x)')
title('Atlantic Cod')
colorbar
axis tight
```

```
%% Main Integrodifference Model Log
clear; clc; close all;
%% Define Parameters

beta = .02; %Predation rate
epsilon = .01; %Conversion efficiency
mu_h = 1.5; %Herring mortality
mu_c = .5; %Cod mortality
space = 0:.1:4; %Space vector from 0 to 4 with step = 0.1
len_space = length(space);
nnodes = length(space);

growth_rate_h = .5; %Herring growth rate scale
growth_rate_c = .5; %Cod growth rate scale
carrying_capacity_h = 150; %Herring carrying capacity
carrying_capacity_c = 25; %Cod carrying capacity
```

```
17
18
19 time = 0:1/12:50; %Time vector, 0 to 50 years
20 len_time = length(time);
21 tnodes = length(time);
22 dt = 1/12; %Time step = 1/12
23
24 run('AlphaH.m') %Generate birth rate vector for herring
25 run('AlphaC.m') %Generate birth rate vector for cod
26
27 %% Construct final matrix
28 pop = zeros(length(time),2*length(space)); %Initialize solution matrix
29 M = length(space);
30
31 %% Construct Coefficient Matrix
32 %Initialize term outside of integral
33 coefficient = zeros(length(time),2*length(space));
34
35 %% Construct Initial Conditions
36 IC_h = (space(1:end).*(3-space(1:end))).^6.*normpdf(space(1:end),0,1)+15;
37 pop(1,1:M) = IC_h;
38 IC_c = .25*(space.*(3-space)).^6.*normpdf(space,0,1)+1;
39 pop(1, M+1:end) = IC_c;
40 %% Construct Coefficient
41 %Initialize coefficient to the shift of the initial population
42 coefficient(1,:) = shift_multi(pop(1,:),1,dt,tnodes, M, mu_c, mu_h);
43
44 %% Construct loop
45 for j=1:length(time)-1
46    %Update the integral term
47    pop(j+1,:) = PopStepLog(pop(j,:),space,alpha_h(j),alpha_c(j),...
```

```
        carrying_capacity_h, carrying_capacity_c, len_space, beta, epsilon);
    %Update the term outside integral
    coefficient(j+1,:) = shift_multi(pop(j,:),j+1,dt,tnodes, M, mu_c, mu_h);
    %Add surviving population and new population
    pop(j+1,:) = coefficient(j+1,:) + pop(j+1,:);
end

%Separate solution matrix into herring and cod
pop_h = pop(:, 1:M);

figure(1);
surf(time, space, pop_h');
shading interp
xlabel('time'); ylabel('space'); zlabel('Herring');
set(gca,'FontSize',20);
title('Atlantic Herring')
multi_ID = caxis;
colorbar
axis tight

pop_c = pop(:, M+1:end);

figure(2);
surf(time, space, pop_c');
shading interp
xlabel('time'); ylabel('space'); zlabel('Cod');
set(gca,'FontSize',20);
title('Atlantic Cod')
caxis(multi_ID);
colorbar
axis tight
```

```
% Velocity function for the Atlantic Cod
% Break up into 12 month time period, assumes final time is 1
% Assumes the model will begin in January
function v_c = integrov_c(n,dt,tnodes)
    %% Define Variables
    mag_c = 1; %Magnitude of velocity
    time  = 0:dt:tnodes; %Time vector
    t_period = 1/dt; %Number of time steps in one year

    m = n - floor(n/t_period)*t_period; %Define the current month, m
    %% Output 0 for staying in their homerange from (January and February)
    if 0 ≤ m && m < 2
        v_c = 0;
    end
    %% Output positive for pre-spawning movement (March), moving north
    if 2 ≤ m && m < 3
        v_c = mag_c;
    end
    %% Output 0 for spawning (April-May)
    if 3 ≤ m && m < 5
        v_c = 0;
    end
    %% Output negative for post-spawning movement (June), moving south
    if 5 ≤ m && m < 6
        v_c = -1*mag_c;
    end
    %% Output 0 for staying in their homerange from (July-January)
    if 6 ≤ m && m ≤ 12
        v_c = 0;
    end
end
```

```
% Velocity function for the Atlantic Herring
% This assumes that 1 in the model represents 1 year
% The year begins in January
%Function will take in the time step (dx), ending space (x_index), and the month(n)
%It will send out a number for the velocity
%% Define variables

function v_h = integrov_h(n, dt, tnodes)

mag_h = 1; %Velocity magnitude
time  = 0:dt:tnodes; %Time vector
t_period = 1/dt; %Number of time steps in one year

m = n - floor(n/t_period)*t_period; %Define the current month, m

%% Output a positive number for times that fall in the spring migration
% Spring migration occurs between the beginning of May and the end of June
if (4)≤m && m<(6)
   v_h =  mag_h;
end

%% Output zero for spawning and winter feeding
% Stationary period 1 occurs between the beginning of July and the end of October
if (6)≤m && m<(10)
    v_h = 0;
end
% Stationary period 2 occurs between the beginning of January and the end of
%April
if 0≤m && m<(4)
   v_h = 0;
end
```

```
32
33 %% Output a negative number for times in the fall migration
34 % Fall migration occurs between the beginning of November and the end of December
35 if (10)≤m && m≤12
36     v_h = -1*mag_h;
37 end
```

```
1  %% shift function for Cod
2  % given population row at time t and space x
3  % given elements to call velocity function
4  % returns a shifted population row
5  function Px_t = shiftC(Px_t,n,dt,tnodes)
6      v = integrov_c(n,dt,tnodes); %Calls the shift function to get a value v
7
8      num_shift = abs(v); %Absolute value of v
9
10     %Shift north if v is positive
11     if v > 0
12         Px_t = [Px_t(1)*ones(1,num_shift), Px_t(1:end-num_shift)];
13     end
14
15     %Shift south if v is negative
16     if v < 0
17         Px_t = [Px_t(num_shift+1:end),Px_t(end)*ones(1,num_shift)];
18     end
19
20
21 end
```

```
%% shift function for Herring
% given population row at time t and space x
% given elements to call velocity function
% returns a shifted population row
function Px_t = shift_multi(Px_t,n,dt,tnodes, M, mu_c, mu_h)
    %% Define row vector

    %Split solution vector into herring and cod
    H_row = Px_t(1, 1:M);
    C_row = Px_t(1, M+1:end);

    %Call shift functions for herring and cod to get constant shift values
    v_h = integrov_h(n,dt,tnodes);
    v_c = integrov_c(n,dt,tnodes);

    %Absolute value of shift function values
    num_shift_h = abs(v_h);
    num_shift_c = abs(v_c);

    if v_h > 0
        %Shift herring north and multiply by survival rate when v_h is positive
        H_row = exp(-mu_h*dt)*[H_row(1)*ones(1,num_shift_h), H_row(1:end-num_shift_h)];
    end

    if v_h < 0
        %Shift herring south and multiply by survival rate when v_h is negative
        H_row = exp(-mu_h*dt)*[H_row(num_shift_h+1:end),H_row(end)*ones(1,num_shift_h)];
    end

    if v_h == 0
        %Only multiply by survival rate when v_h = 0
```

```
        H_row = exp(-mu_h*dt)*H_row;
    end

    if v_c > 0
        %Shift cod north and multiply by survival rate when v_c is positive
        C_row = exp(-mu_c*dt)*[C_row(1)*ones(1,num_shift_c), C_row(1:end-num_shift_c)];
    end

    if v_c < 0
        %Shift cod south and multiply by survival rate when v_c is negative
        C_row = exp(-mu_c*dt)*[C_row(num_shift_c+1:end),C_row(end)*ones(1,num_shift_c)];
    end

    if v_c == 0
        %Only multiply by survival rate when v_c = 0
        C_row = exp(-mu_c*dt)*C_row;
    end


    %Recombine herring and cod pieces
    Px_t = [H_row, C_row];
end
```

```
% Creating the alpha script for Cod in the integrodifference model


sigma = .04; %Standard deviation of gaussian
a_c = growth_rate_c; %Growth rate scale
mean_c = time(6); %Define mean of Gaussian

```

```
%Define the Gaussian function
Gaussian = @(t,sigma,mean_h) (1/(sqrt(2*pi*sigma^2)))*exp(-((t-mean_c).^2)/(2*sigma^2));

alpha_c_year = zeros(1,12); %Initialize a year of zeros

%Update values during spawning season to have a Gaussian distribution
index = 4;
for t0=(3/12):(4/12)
t1 = t0 + (1/12);
int_val = integral(@(t) Gaussian(t,sigma,mean_c),t0,t1);
alpha_c_year(index) = int_val;
index = index+1;
end

alpha_c = alpha_c_year;
num_full_reps = floor(len_time/12); %Determine number of full years in total time

for j=1:num_full_reps-1
    alpha_c = [alpha_c,alpha_c_year]; %Repeat one-year vector
end


%Add remaining entries if the total time is not a whole number of years
remaining_cols = rem(len_time,12);
alpha_c_add = alpha_c_year(:,1:remaining_cols);
alpha_c = [alpha_c,alpha_c_add];

alpha_c = a_c*alpha_c; %Scale by birth rate scale

%
```

```
% Creating the alpha script for Cod in the PDE model

%clear; clc; close all;

ntime = length(time); %time is inherited from the PDE main code

%Define mean and standard deviation of Gaussian and pull in the birth rate
%scale from the PDE main code
sigma = .04;
a_h = growth_rate_h;
mean_h = time(9);

t_0 = 6; % first value
t_1 = 11; % last value

alpha_h_year = zeros(1,12); %Initialize a one-year vector of zeros

for n=t_0:t_1
    %Birth rate is a Gaussian during the spawining season
    alpha_h_year(1,n) = (1/(sqrt(2*pi*sigma^2)))*exp(-((time(n)-mean_h)^2)/(2*sigma^2));
end

alpha_h = alpha_h_year;
num_full_reps = floor(ntime/12); %Determine how many full years are in the total time

for j=1:num_full_reps-1
    alpha_h = [alpha_h,alpha_h_year]; %Repeat the one-year alpha
end

%Add remaining entries if the end time is not a whole number of years
remaining_cols = rem(ntime,12);
```

```
alpha_h_add = alpha_h_year(:,1:remaining_cols);
alpha_h = [alpha_h,alpha_h_add];

alpha_h = a_h*alpha_h; %Scale by birth rate scale
```

```
%% Create Kernel
% Receives length of space and space vector
% Returns kernel in matrix form

function A = kernel_gaussian(len_space,space)
    A = zeros(len_space); % create empty matrix to house the transition probabilities
    mu_kern = 0; %Defines the mean of the kernel
    sigma_kern = 2; %Defines the standard deviation of the kernel
    for i = 1:len_space %Loop over rows
        for j = 1:len_space %Loop over columns in row and define Gaussian
            % insert the probability of an individual moving from j to i,
            % used the gaussian quadrature and set sigma = 1
            A(i,j) = (1/sqrt(2*pi*sigma_kern^2))*exp(-(space(i)+...
                mu_kern-space(j)).^2/(2*sigma_kern^2));
        end
    end

end
```

```
%% Create Kernel
% Receives dimensions
% Returns Matrix

function A = kernel_gaussian_multi(len_space,space)
```

```
    A = zeros(len_space); % create empty matrix to house the transition probabilities
    mu_kern_h = 1; %Define the mean of the kernel for herring
    sigma_kern_h = 2.5; %Define the standard deviation of the kernel for cod
    for i = 1:len_space % Loop over rows
        for j = 1:len_space % Loop over columns in row to define Gaussian value
            % insert the probability of an individual moving from j to i,
            % used the gaussian quadrature and set sigma = 1
            A(i,j) = (1/sqrt(2*pi*sigma_kern_h^2))*exp(-(space(i)+...
                mu_kern_h-space(j)).^2/(2*sigma_kern_h^2));
        end
    end
    B = zeros(len_space); % create empty matrix to house the transition probabilities
    mu_kern_c = 1; %Define the mean of the kernel for cod
    sigma_kern_c = 2.5; %Define the standard deviation of the kernel for cod
    for i = 1:len_space % Loop over rows
        for j = 1:len_space %Loop over columns in row and define Gaussian value
            % insert the probability of an individual moving from j to i,
            % used the gaussian quadrature and set sigma = 1
            B(i,j) = (1/sqrt(2*pi*sigma_kern_c^2))*exp(-(space(i)+...
                mu_kern_c-space(j)).^2/(2*sigma_kern_c^2));
        end
    end
    A = [A;B];

end
```

```
%% Logistic function
% receives growth rate, carrying capacity (cap), population row vector (N_t)
% returns row vector transformed by logistic model

```

```
function N_log = logistic_new(alpha_c_val,carrying_capacity, N_t)
    len_vec = length(N_t); %Number of spaces in population vector
    N_log = zeros(1,len_vec); %Initialize output variable to 0
    %Calculate logistic growth
    for j = 1:len_vec
       N_log(j) = alpha_c_val*N_t(j)*(1 - N_t(j)/carrying_capacity);
    end
end
```

```
%% Logistic function with interaction
% receives rate, capacity (cap), vector (N_t)
% returns vector transformed by logistic model

function N_log = logistic_multi(alpha_h_val, alpha_c_val,carrying_capacity_h,...
    carrying_capacity_c, N_t, beta, epsilon)
    len_vec = length(N_t);
    N_log = zeros(1,len_vec);
    M = len_vec/2;

    for j = 1:len_vec
        if j ≤ M %For the herring rows, define logistic minus predation term
       N_log(j) = alpha_h_val*N_t(j)*(1 - N_t(j)/carrying_capacity_h) - ...
           (beta .* N_t(j) .* N_t(M+j));

        end
        if j > M %For the cod rows, define logistic plus predation term
            N_log(j) = alpha_c_val*N_t(j)*(1 - N_t(j)/carrying_capacity_c) + ...
                (epsilon .* N_t(j-M) .* N_t(j));
        end

```

```
    end
end
```

```
%% PopStepLog
% Receives a row in the population matrix, the space vector, value of alpha
% for herring, carrying capcity for the logistic growth function, and the
% length of space
% Returns a new row vector for the population matrix

function [N_new] = PopStepLog_gaussian(N_t,space,alpha_c_val,carrying_capacity,len_space)
    %% Call Kernel
    %Calls the kernel_gaussian function to define the kernel
    kern = kernel_gaussian(len_space,space);

    %% Call Example
    %Calls the logistic_new function to calculate logistic growth
    log_func = logistic_new(alpha_c_val,carrying_capacity,N_t);

    %% Call weights
    %Defines weights for the quadrature method
    w = weights(len_space,space(end)-space(end-1));

    %% For Loop to assemble solution update
    N_new = zeros(size(N_t)); %Initialize to 0
    for i = 1:len_space
        B = kern(:,i).*log_func'; %Multiply logistic growth output by kernel
        N_new(i) = quad_reu(B,w,len_space); %Approximate integral
    end
end
```

```
%% PopStepLog
% Receives a vector of zeros
% Returns a new vector

function [N_new] = PopStepLog(N_t,space,alpha_h_val, alpha_c_val,...
    carrying_capacity_h, carrying_capacity_c,len_space, beta, epsilon)
    %% Call Kernel
    %Call kernel_gaussian_multi to define kernel for herring and cod
    kern = kernel_gaussian_multi(len_space,space);

    %% Call Example
    %Call logistic_multi to calculate logistic growth inside integral
    log_func = logistic_multi(alpha_h_val, alpha_c_val,carrying_capacity_h,...
        carrying_capacity_c,N_t, beta, epsilon);
    %% Call weights
    %Return a vector with the weights for the trapezoid rule
    w = weights(len_space,space(end)-space(end-1));

    %% For Loop to assemble vec
    N_new = zeros(size(N_t)); %Initialize solution
    for i = 1:len_space
        B = kern(:,i).*log_func'; %Multiply logistic growth output by kernel
        %Approximate integral
        [N_new(i), N_new(len_space+i)] = quad_reu_multi(B,w,len_space);
    end
end
```

```
%% Gaussian function
% Receives a vector of the integrand, a vector of weights from the
% quadrature, andthe length of space
```

```
% Returns an approximation for the integrand
function I = quad_reu(f,wt,len_space)
    %% Approximation
    I = 0;
    %Sum the area under each section of the curve for the quadrature method
    for i=1:len_space
        I = I + f(i)*wt(i);
    end
end
```

```
%% Gaussian function
% Receives a vector of weights and a vector of the integrand approximated
% along grid points
% Returns an approximation for the integrand
function [I_1, I_2] = quad_reu_multi(B,w,len_space)
    %% Approximation
    I_1 = 0;
    I_2 = 0;

    %Approximate integral for herring using trapezoid rule
    for i=1:len_space
        I_1 = I_1 + B(i)*w(i);
    end

    % Approximate integral for cod using trapezoid rule
    for i=1:len_space
        I_2 = I_2 + B(len_space+i)*w(len_space+i);
    end

end
```

```
%% Weight
% receive dim, dx
% return a vector of weights for the trapezoid rule
function wt = weights(d,dx)
    wt = ones([d,1]).*dx; %column vector
    wt(1) = 0.5*dx;
    wt(d) = 0.5*dx;
end
```

A.3 Ensemble Kalman Filter Codes

```
% EnKF code to use with the PDE Herring model to estimate states and
% parameters

clear;clc;close all;

%% True Values of Parameters
true_K_h = 100; %carrying capacity
true_D_h = 0.1; %diffusion coefficient

%% Initial Conditions
load('ic_states.mat'); %Load in initial conditions
num_states = length(IC); %Tells how many states we will be working with
num_theta = 2; %Number of parameters being estimated

%% Load Data
load('obsData_multi.mat'); %Load in synthetic data

%Plot true solution
figure(1);
```

```
surf(true_time, true_space, true_sol_h)
shading interp
title('True solution')
set(gca,'fontsize',18);
colorbar
caxis('manual');
cl = caxis;

y_data = h_data;
%% Define space vector
X = 4; %The length of space
dx = 0.1; %the space step increment
space = 0:dx:X; %space vector

%% Define Time
T = true_time(end); %The number of years
dt = 1/12; %time increment will be one month
time = 0:dt:T; %time vector
len_time = length(time); %length of time vector

%% Define Ensemble members
N = 250; % Number of Ensemble members

%% Define noise
c_std = 0.2; %noise added to the model
d_std = 0.01; %noiselev; %noise added to the data
D = d_std^2;

%% Define Structure to store values
%states
Structure.x = NaN(num_states, N, len_time);
```

```
%parameters
Structure.theta = NaN(num_theta, N, len_time);
%mean of ensemble members for states
Structure.xmean = NaN(num_states, len_time);
%mean of ensemble members for parameters
Structure.thetamean = NaN(num_theta, len_time);
%standard deviation of ensemble members for states
Structure.xstd = NaN(num_states, len_time);
%standard deviation of ensemble members for parameters
Structure.thetastd = NaN(num_theta, len_time);

%% Put initial condition in structure
%Set initial value in structure equal to initial conditions
Structure.x(:, :, 1) = IC*ones(1,N);

%Define a new variable equal to the structure value
x_pred = Structure.x(:,:,1);

x_mean = mean(x_pred,2); %Take mean of ensemble members
Structure.xmean(:, 1) = x_mean; %Save


%For lin KF
diff = x_pred - x_mean;
x_cov = (diff*diff')/(N-1);
Structure.xstd(:,1) = sqrt(diag(x_cov));

%% Add noise to initial parameters in structure
K_max = 100;
K_min = 80;

```

```
D_max = 0.2;
D_min = 0.05;

%Define prior for carrying capacity
Structure.theta(1,:,1) = K_min + (K_max-K_min).*rand(1, N);
%Define prior for diffusion coefficient
Structure.theta(2,:,1) = D_min + (D_max-D_min).*rand(1, N);

%Save parameter values from random draws
theta_pred = Structure.theta(:,:, 1);

theta_mean = mean(theta_pred, 2); %Take mean of theta ensemble members
Structure.thetamean(:, 1) = theta_mean; %Save

%For linear Kalman filter
diff = theta_pred - theta_mean;
theta_cov = (diff*diff')/(N-1);
Structure.thetastd(:,1) = sqrt(diag(theta_cov));

%% Covariance inflation
%Define a value by which to artificially inflate the covariance
Δ = 0.25;

%% Define g for model observation prediction step
%Define the observation matrix
G = [eye(num_states) zeros(num_states,num_theta)];

%% For loop to constuct
m = 1;
for t = 1:len_time-1 %do this every time step (month)
    %% Prediction Step
```

```
113     mon = mod(t, 12); %% Determines what month to solve
114
115     for n=1:N
116         %Update model by one month
117         x_model = Herring_PDE_one_month(theta_pred(1,n), ...
118             theta_pred(2,n), dt/8, dx, X, x_pred(:, n), mon);
119
120         % Adding noise to the model output
121         x_pred(:,n) = x_model + c_std * randn(num_states, 1);
122     end
123
124     z = [x_pred; theta_pred]; %Augmented state and paramter vectors
125     z_mean(:,1) = mean(z, 2); %Predicition ensemble mean
126
127     diff = z - z_mean;
128     z_cov = (diff*diff')/(N-1);
129     z_cov = (1 + ⍙) * z_cov;
130
131     %% Analysis Step
132     % Analysis Step for April and October months to mimic data frequency
133     if mon ==  4 || mon == 10
134
135         y_hat = G * z; % Calculate model prediction observation
136
137         % Calculate observation ensemble
138         noisy_data = y_data(:, m) + d_std * randn(num_states, N);
139
140         % Compute Kalman gain
141         Kalman_gain = z_cov*G'/ (G*z_cov*G'+D);
142
143         % Compute combined posterior ensemble
```

```
144         z = z + Kalman_gain * (noisy_data - y_hat);
145
146         z_mean(:,1) = mean(z,2); % Saving the mean of the posterior ensemble
147         m = m+1; % Increase where the data is pulled from by one
148
149     end
150
151     %Set new initial conditions for states based on output
152     x_pred = z(1:num_states, :);
153     %Set new initial conditions for parameters based on output
154     theta_pred = z(num_states+1:end, :);
155
156     %Computing Covariances
157     z_final = z - z_mean;
158     covariances = z_final * z_final' / (N-1);
159     covariances2 = sqrt(diag(covariances));
160
161     % Saving the posterior ensemble for states
162     Structure.x(:,:,t+1) = z(1:num_states, :);
163     % Saving the mean of the posterior ensemble for states
164     Structure.xmean(:,t+1) = z_mean(1:num_states, :);
165     %Saving the standard deviations of the states
166     Structure.xstd(:, t+1) = covariances2(1:num_states, :);
167
168     %Saving the posterior ensemble for parameters
169     Structure.theta(:,:, t+1) = z(num_states+1:end, :);
170     %Saving the mean of the posterior ensemble for parameters
171     Structure.thetamean(:, t+1) = z_mean(num_states+1:end, :);
172     %Saving the standard deviations of the parameters
173     Structure.thetastd(:, t+1) = covariances2(num_states+1:end, :);
174
```

```
end

%% Plot parameters

K_pred = Structure.thetamean(1, :);
std_K = Structure.thetastd(1, :);

figure(3)
plot(time, true_K_h*ones(size(time)),'-k','LineWidth',2.5); hold on;
plot(time, K_pred, '-r','LineWidth',2);
plot(time, K_pred+2*std_K, '--r','LineWidth',2)
plot(time, K_pred-2*std_K, '--r','LineWidth',2)
hold off
title('Carrying Capacity')
set(gca,'fontsize',18);

D_pred = Structure.thetamean(2, :);
std_D = Structure.thetastd(2, :);

figure(4)
plot(time, true_D_h*ones(size(time)),'-k','LineWidth',2.5); hold on;
plot(time, D_pred,'-r','LineWidth',2);
hold on
plot(time, D_pred+2*std_D,'--r','LineWidth',2)
plot(time, D_pred-2*std_D,'--r','LineWidth',2)
hold off
title('Diffusion Constant')
set(gca,'fontsize',18);

%% Plot states

```

```
figure(5)
surf(time,space,Structure.xmean);
shading interp
title('EnKF Mean Solution');
set(gca,'fontsize',18);
colorbar
caxis(cl)

%% Save data

struct = Structure; %Save structure as variable name
true_sol_filter = true_sol_h;
save('Herring_PDE_from_filter', 'struct', 'y_data', 'true_sol_filter');
```

```
%Code for initial condition vector for Herring
%Will be fed into the main file for the Kalman Filter
% Creates ic_states.mat file

%% Define space vector
X = 4; %The length of space
dx = 0.1; %the space step increment
space = 0:dx:X; %space vector

%% Define initial condition
IC = 5*(space(1:end).*(3-space(1:end))).^6.*normpdf(space(1:end),0,1)+10;

%% Transpose initial condition
IC = IC';

save('ic_states','IC')
```

```
% Code to generate synthetic data for Herring PDE
% Creates obsData.mat file

clear; close all; clc;

%% Parameters

k = (1/12)/(2^3); %Define small time step that is 1/12 divided by a power of 2
t_end = 15; %15 years

h = 0.1; %Space step = 0.1
x_end = 4; %Space end = 4

%% Call the Function
%Call function to generate synthetic data and save it in .mat file
[y_data,t_data,noiselev,true_sol,true_time,true_space] = ...
    Herring_PDE_data_generate(k,t_end,h,x_end);
save('obsData_PDE_Herring','y_data','t_data','noiselev','true_sol',...
    'true_time','true_space')

%Plot
figure(1);
surf(t_data,true_space,y_data);
shading interp
colorbar

figure(2);
surf(true_time,true_space,true_sol);
shading interp
colorbar
```

```
%Function calling the Herring PDE model
% This generates synthetic data for Herring
function [data_obs,time_obs,noiselev,U_save,time,space]=...
    Herring_PDE_data_generate(k,t_end,h,x_end)

%% Load parameter values
herring_parameters=load('Herring_parameters.txt');
a_h = herring_parameters(4); %Growth rate
K_h = herring_parameters(1); %Carrying capacity
mu_h = herring_parameters(2); %Fishing mortality
D_h = herring_parameters(3); %Diffusion constant
mag_h = herring_parameters(5); %Advection constant

noiselev = 0.01; % noise level for synthetic data

%% Time discretization
time  = 0:k:t_end; %time vector
ntime = length(time);
tau = 1; %counter
k2 = .5*k^2; %Time step for bootstrapping
t_bar = 0:k2:k; %Time vector for bootstrapping
nt_bar = length(t_bar);
%% Spatial discretization

space = 0:h:x_end; %Space vector
nspace = length(space);
M = nspace;

%% Define common constants

%Vector containing growth rate values at each time
```

```
[alpha_h,t_period] = AlphaH_AA(k,t_end,a_h);

r = k/(2*h*h);
d_con = D_h*r; %Constant arising from Crank-Nicolson

%% Define the time_obs vector containing the times we make observations
time_obs = [];

pd = 1/k;
fac = pd/12;
for m = 1:ntime
    if ( round(mod(pd*time(m),pd)) == 4*fac || ...
            round(mod(pd*time(m),pd)) == 10*fac )
        time_obs = [time_obs, time(m)];
    end
end

num_obs = length(time_obs);

%% Define relevant matrices

%Coefficient matrix due to Crank-Nicolson for V^n+1
CN_left = -d_con*diag(ones(M-1, 1), 1) + 2*d_con*diag(ones(M,1)) - ...
    d_con*diag(ones(M-1, 1), -1);
%Impose zero-slope boundary conditions
CN_left(1,2) = 2*CN_left(1,2);
CN_left(end,end-1) = 2*CN_left(end,end-1);

%Identity matrix to represent the time approx for V^n+1
Forward_time_left = diag(ones(M,1));

```

```
%Total atrix with the coefficients of the solution at time n+1
M_lhs = Forward_time_left + CN_left;


%Coefficient matrix due to CN for V^n
CN_right = d_con*diag(ones(M-1, 1), 1) - 2*d_con*diag(ones(M,1)) + ...
    d_con*diag(ones(M-1, 1), -1);
%Impose zero-slope boundary conditions
CN_right(1,2) = 2*CN_right(1,2);
CN_right(end,end-1) = 2*CN_right(end,end-1);

%Identity matrix representing the V^n term due to the time approx

Forward_time_right = diag(ones(M,1));

M_rhs_base = Forward_time_right + CN_right;


%% Bootstrapping matrices
r2 = k2/(2*h*h);

%Constant arising from the Crank-Nicolson for bootstrapping time step
d_con2 = D_h*r2;

CN_left2 = d_con2*CN_left/d_con;

%Identity matrix to represent the time approx for V^n+1
Forward_time_left2 = diag(ones(M,1));

%Total atrix with the coefficients of the solution at time n+1
M_lhs2 = Forward_time_left2 + CN_left2;
```

```
94
95 CN_right2 = d_con2*CN_right/d_con;
96
97 %Identity matrix representing the V^n term due to the time approx
98
99 Forward_time_right2 = diag(ones(M,1));
100
101 M_rhs2_base = Forward_time_right2 + CN_right2;
102
103
104 %% Initial conditions
105
106 IC = 5*(space(1:end).*(3-space(1:end))).^6.*normpdf(space(1:end),0,1)+10 ;
107
108 %% Define the solution matrix
109
110 U_save = zeros(M, ntime);  %Creates a matrix to store values for V in
111 U_save(:,1) = IC'; %Changes the first column to the initial conditions
112
113 %Initializes the matrix for synthetic data
114 data_obs = zeros(M,num_obs);
115
116 %Initializes matrix for true values in only April and October
117 model_val = zeros(M,num_obs);
118
119 %% Define the Bootstrapping solution matrix
120
121 U_bar = zeros(M,nt_bar); %Bootstrapping solution matrix
122 U_bar(:,1) = IC'; %Set first column to initial conditions
123
124 %% Update by time step
```

```
for n=1:ntime-1
    m2 = mod(n,t_period);  %tells us the month
    v_h = velocity_h(n,k,t_end,mag_h); %get a velocity value
    a_con = v_h*k/h; %Constant arising from upwinding
    %Define matrix representing upwinding
    UW_right = a_con*diag(ones(M-1, 1), -1) - a_con*diag(ones(M,1));
    %Impose zero-slope boundary conditions
    UW_right(1,:) = 0;
    UW_right(end,:) = 0;
    %Define reaction term matrix
    React_right = (k*alpha_h(n) - k*mu_h)*diag(ones(M,1));


    %Repeat for bootstrapping time step
    a_con2 = v_h*k2/h;
    UW_right2 = a_con2*diag(ones(M-1, 1), -1) - a_con2*diag(ones(M,1));
    UW_right2(1,:) = 0;
    UW_right2(end,:) = 0;
    React_right2 = (k2*alpha_h(n) - k2*mu_h)*diag(ones(M,1));

    M_rhs = M_rhs_base + React_right + UW_right;
    M_rhs2 = M_rhs2_base + React_right2 + UW_right2;

    if n == 1 %Implement bootstrapping on first iteration of time loop
        for m=1:nt_bar-1

            U_bar(:,m+1) = M_rhs2*U_bar(:,m) - ...
                (k2*alpha_h(n)/K_h)*U_bar(:,m).*U_bar(:,m);

            U_bar(:,m+1) = M_lhs2\U_bar(:,m+1);
        end
```

```
156
157             U_save(:,2) = M_rhs*U_bar(:,end) - ...
158                 (k*alpha_h(n)/K_h)*U_save(:,end).*U_save(:,end);
159
160             U_save(:,2) = M_lhs\U_save(:,2);
161
162     else %Otherwise uodate as normal
163
164         U_save(:,n+1) = M_rhs*U_save(:,n) - ...
165             (k*alpha_h(n)/K_h)*U_save(:,n).*U_save(:,n);
166
167         U_save(:,n+1) = M_lhs\U_save(:,n+1);
168
169     end
170
171     % Save data
172     if m2==round(4*t_period/12) || m2==round(10*t_period/12)
173         model_val(:,tau) = U_save(:,n+1); %Save model value in April or October
174         sz = size(U_save(:,n+1));
175         %Add noise and save synthetic data in Apr or OCt
176         data_obs(:,tau) = U_save(:,n+1) + noiselev*randn(sz);
177         tau = tau+1; %Add one to the count
178     end
179
180 end
```

```
1 % This runs the model for one month at a time
2 % Used for estimating states and parameters(K,D)
3
4 function one_month_sol = Herring_PDE_one_month(K_h,D_h,k,h,x_end,IC,mon)
```

```
5
6  herring_parameters=load('Herring_parameters.txt');
7  a_h = herring_parameters(4); %Growth rate
8  mu_h = herring_parameters(2); %Fishing mortality
9  mag_h = herring_parameters(5); %Advection constant
10
11 %% Time discretization
12
13 %Month is defined using mod, but we want 12 instead of 0 for simplicity
14 if mon == 0
15     mon = 12;
16 end
17 time  = (mon-1)/12:k:(mon/12); %Fro beginning of month to end of month
18 ntime = length(time);
19 %Time for bootstrapping
20 k2 = .5*k^2;
21 t_bar = 0:k2:k;
22 nt_bar = length(t_bar);
23 %% Spatial discretization
24
25 space = 0:h:x_end; %Space vector
26 nspace = length(space);
27 M = nspace;
28
29 %% Define common constants
30 %Vector of growth rate values at each time
31 [alpha_h] = AlphaH_month_AA(k,a_h,mon,time);
32
33 r = k/(2*h*h);
34 d_con = D_h*r; %Constant arising from the Crank-Nicolson
35
```

```
%% Define relevant matrices

%Coefficient matrix due to Crank-Nicolson for V^n+1
CN_left = -d_con*diag(ones(M-1, 1), 1) + 2*d_con*diag(ones(M,1)) - ...
    d_con*diag(ones(M-1, 1), -1);
%Implement zero-slope boundary conditions
CN_left(1,2) = 2*CN_left(1,2);
CN_left(end,end-1) = 2*CN_left(end,end-1);

%Identity matrix to represent the time approx for V^n+1
Forward_time_left = diag(ones(M,1));

%Total matrix with the coefficients of the solution at time n+1
M_lhs = Forward_time_left + CN_left;

%Coefficient matrix due to CN for V^n
CN_right = d_con*diag(ones(M-1, 1), 1) - 2*d_con*diag(ones(M,1)) + ...
    d_con*diag(ones(M-1, 1), -1);
%Implement zero-slope boundary conditions
CN_right(1,2) = 2*CN_right(1,2);
CN_right(end,end-1) = 2*CN_right(end,end-1);

%Identity matrix representing the V^n term due to the time approx

Forward_time_right = diag(ones(M,1));

M_rhs_base = Forward_time_right + CN_right;


%% Bootstrapping matrices
r2 = k2/(2*h*h);
```

```
67
68 %Constant arising from the Crank-Nicolson for bootstrapping time step
69 d_con2 = D_h*r2;
70
71 CN_left2 = d_con2*CN_left/d_con;
72
73 %Identity matrix to represent the time approx for V^n+1
74 Forward_time_left2 = diag(ones(M,1));
75
76 %Total atrix with the coefficients of the solution at time n+1
77 M_lhs2 = Forward_time_left2 + CN_left2;
78
79 CN_right2 = d_con2*CN_right/d_con;
80
81 %Identity matrix representing the V^n term due to the time approx
82
83 Forward_time_right2 = diag(ones(M,1));
84
85 M_rhs2_base = Forward_time_right2 + CN_right2;
86
87
88 %% Define the solution matrix
89
90 U_save = zeros(M, ntime);  %Creates a matrix to store values for V in
91 U_save(:,1) = IC; %Changes the first column to the initial conditions
92
93 %initializes the solution for one month from initial time
94 one_month_sol = zeros(M,1);
95 %% Define the Bootstrapping solution matrix
96
97 U_bar = zeros(M,nt_bar); %Bootstrapping solution matrix
```

```
98  U_bar(:,1) = IC'; %Set first column to initial conditions
99
100 %% Update by time step
101 for n=1:ntime-1
102     %Call velocity_h_month to get a velocity value
103     v_h = velocity_h_month(mag_h,n,mon,time);
104
105     a_con = v_h*k/h; %Constant arising from upwinding
106     %Matrix representing upwinding
107     UW_right = a_con*diag(ones(M-1, 1), -1) - a_con*diag(ones(M,1));
108     %Impose zero-slope boundary condition
109     UW_right(1,:) = 0;
110     UW_right(end,:) = 0;
111     %Matrix representing reaction term
112     React_right = (k*alpha_h(n) - k*mu_h)*diag(ones(M,1));
113
114
115     %Repeat for bootstrapping time step
116     a_con2 = v_h*k2/h;
117     UW_right2 = a_con2*diag(ones(M-1, 1), -1) - a_con2*diag(ones(M,1));
118     UW_right2(1,:) = 0;
119     UW_right2(end,:) = 0;
120     React_right2 = (k2*alpha_h(n) - k2*mu_h)*diag(ones(M,1));
121
122     M_rhs = M_rhs_base + React_right + UW_right;
123     M_rhs2 = M_rhs2_base + React_right2 + UW_right2;
124
125     if n == 1 %implement bootstrapping at first iteration of time loop
126         for m=1:nt_bar-1
127
128             U_bar(:,m+1) = M_rhs2*U_bar(:,m) - ...
```

```
                (k2*alpha_h(n)/K_h)*U_bar(:,m).*U_bar(:,m);

            U_bar(:,m+1) = M_lhs2\U_bar(:,m+1);
        end

            U_save(:,2) = M_rhs*U_bar(:,end) - ...
                (k*alpha_h(n)/K_h)*U_save(:,end).*U_save(:,end);

            U_save(:,2) = M_lhs\U_save(:,2);

    else %otherwise implement update as normal

        U_save(:,n+1) = M_rhs*U_save(:,n) - ...
            (k*alpha_h(n)/K_h)*U_save(:,n).*U_save(:,n);

        U_save(:,n+1) = M_lhs\U_save(:,n+1);

    end

end
one_month_sol(:,1) = U_save(:,end); %Set output to last value
```

```
% Creating the alpha function for Cod in the PDE model
% Generates an alpha value for a single month depending on the month you want
% Used in the Herring_PDE_one_month code

%clear; clc; close all;

function [alpha_h] = AlphaH_month_AA(k,a_h,mon,time)

```

```
%Define time vectors for a full year
t_end_2 = 1;
time_2  = 0:k:t_end_2;
t_period_2 = 1/k;

ntime = length(time);

if mon≥6 && mon≤10 %During spawning season
    sigma = .035; %Define standard deviation of Gaussian

    mean_h = time_2(floor((8.7*t_period_2)/12)); %Mean of Gaussian

    alpha_h_month = zeros(1,ntime); %Initialize a one-year long vector

    for n=1:ntime
        %Birth rate is a Gaussian during the spawining season
        alpha_h_month(1,n) = (1/(sqrt(2*pi*sigma^2)))*...
            exp(-((time(n)-mean_h)^2)/(2*sigma^2));
    end

else
    %Define zeros if it isn't spawning season
    alpha_h_month = zeros(1,ntime);
end
%Scale by growth rate factor
    alpha_h = a_h*alpha_h_month;
```

```
% Velocity function for the Atlantic Herring
% Break up into 12 month time period, assumes final time is 1
% Assumes the model will begin in January
```

```
% Used in the Herring_PDE_one_month code

function v_h = velocity_h_month(mag_h,n,mon,time)

b = 200;


%% Region 1, January through the middle of spring movement
if mon≤5
    v_h = .5*(1+tanh(b*(time(n)-2/12)));
end

%% Region 2, Middle of spring movement through middle of spawning
if 5<mon  && mon≤9
    v_h = -.5*(-1+tanh(b*(time(n)-3/12)));
end

%% Region 3, Middle of spawning through middle of summer movement
if 9<mon && mon≤11
    v_h = -.5*(1+tanh(b*(time(n)-5/12)));
end

%% Region 4, Middle of summer movement through end of year
if 11<mon && mon≤12
    v_h = .5*(-1+tanh(b*(time(n)-6/12)));
end

v_h = mag_h*v_h; %Scale by magnitude
```

```
% EnKF code to use with the PDE multispecies model
```

```
2
3  tic
4
5  clear;clc;close all;
6
7  %% Define space vector
8  X = 4; %The length of space
9  dx = 0.1; %the space step increment
10 space = 0:dx:X; %space vector
11
12 %% Define Time
13 T = 15; %The number of years
14 dt = 1/12; %time increment will be one month
15 time = 0:dt:T; %time vector
16 len_time = length(time); %length of time vector
17
18 %% Define Ensemble members
19 N = 250; % Number of Ensemble members
20
21 %% True Values of Parameters
22 true_K_h = 100;
23 true_D_h = 0.1;
24
25 true_K_c = 20;
26 true_D_c = 0.1;
27
28 true_epsilon = 0.1;
29 true_beta = 0.1;
30
31 %% Initial Conditions
32 load('ic_states_multi.mat');
```

```
num_states = length(IC_h); %Tells how many states we will be working with
num_theta = 3; %Number of parameters being estimated per species

%% Load Data
%load('obsData.mat');
load('obsData_multi.mat');
%len_data = length(h_data);

data_obs = [h_data; c_data]; %%%%%%%Changed this %Augmented the two observation datas

figure(1);
surf(true_time,true_space,true_sol_h);
shading interp
colorbar

figure(2);
surf(true_time,true_space,true_sol_c);
shading interp
colorbar

%% Define noise
c_std = 0.01; %0.1; %noise added to the model
d_std = 0.01; %0.005; %noise added to the data
D = d_std^2;

%% Define Structure to save values
%States for herring
Structure.xh = NaN(num_states, N, len_time);
Structure.x_meanh = NaN(num_states, len_time);
Structure.xstdh = NaN(num_states, len_time);

```

```
64 %States for cod
65 Structure.xc = NaN(num_states, N, len_time);
66 Structure.x_meanc = NaN(num_states, len_time);
67 Structure.xstdc = NaN(num_states, len_time);
68
69 %Parameters for herring
70 Structure.thetah = NaN(num_theta, N, len_time);
71 Structure.thetameanh = NaN(num_theta, len_time);
72 Structure.thetastdh = NaN(num_theta, len_time);
73
74 %Parameters for cod
75 Structure.thetac = NaN(num_theta, N, len_time);
76 Structure.thetameanc = NaN(num_theta, len_time);
77 Structure.thetastdc = NaN(num_theta, len_time);
78
79 %% Put initial condition in structure
80 % Initial Herring conditions
81 Structure.xh(:, :, 1) = IC_h*ones(1,N);
82 x_predh = Structure.xh(:,:,1);
83
84 % Initial Cod conditions
85 Structure.xc(:, :, 1) = IC_c*ones(1,N);
86 x_predc = Structure.xc(:,:,1);
87
88 %% For linear Kalman Filter
89 % Atlantic Herring
90 %Initialize mean over ensemble states
91 x_meanh = mean(x_predh,2);
92 Structure.x_meanh(:,1) = x_meanh;
93
94 %For Kalman filter algorithm
```

```
95  diff_h = x_predh - x_meanh;
96  x_covh = (diff_h * diff_h') / (N-1);
97  Structure.xstdh(:,1) = sqrt(diag(x_covh));
98
99  % Atlantic Cod
100 x_meanc = mean(x_predc,2);
101 Structure.x_meanc(:,1) = x_meanc;
102
103 %For Kalman filter algorithm
104 diff_c = x_predc - x_meanc;
105 x_covc = (diff_c * diff_c') / (N-1);
106 Structure.xstdc(:,1) = sqrt(diag(x_covc));
107
108 %% Add noise to initial parameter values in structure
109
110 Kh_max = 120;
111 Kh_min = 105;
112
113 Dh_max = 0.2;
114 Dh_min = 0.15;
115
116 Kc_max = 27;
117 Kc_min = 22;
118
119 Dc_max = 0.22;
120 Dc_min = 0.14;
121
122 Beta_max = 0.27;
123 Beta_min = 0.18;
124
125 Ep_max = 0.2;
```

```
126 Ep_min = 0.05;
127
128 %Define prior distributions for the parameters
129 Structure.thetah(1,:,1) = Kh_min + (Kh_max-Kh_min).*rand(1, N);  %K_h
130 Structure.thetah(2,:,1) = Dh_min + (Dh_max - Dh_min).*rand(1, N); %D_h
131 Structure.thetac(1,:,1) = Kc_min + (Kc_max-Kc_min).*rand(1, N);  %K_c
132 Structure.thetac(2,:,1) = Dc_min + (Dc_max - Dc_min).*rand(1, N); %D_c
133
134 Structure.thetac(3,:,1) = Ep_min + (Ep_max - Ep_min).*rand(1, N); %Epsilon
135 Structure.thetah(3,:,1) = Beta_min + (Beta_max - Beta_min).*rand(1, N); %Beta
136
137 %Save parameter values
138 theta_predh = Structure.thetah(:,:, 1);
139 theta_predc = Structure.thetac(:,:, 1);
140
141 %Save parameter means
142 theta_meanh = mean(theta_predh,2);
143 theta_meanc = mean(theta_predc,2);
144
145 Structure.thetameanh(:, 1) = theta_meanh;
146 Structure.thetameanc(:, 1) = theta_meanc;
147
148 %% For Linear Kalman Filter (again)
149
150 diff_h = theta_predh - theta_meanh;
151 diff_c = theta_predc - theta_meanc;
152
153 theta_covh = (diff_h * diff_h')/(N-1);
154 theta_covc = (diff_c * diff_c')/(N-1);
155
156 Structure.thetastdh(:,1) = sqrt(diag(theta_covh));
```

```
157 Structure.thetastdc(:,1) = sqrt(diag(theta_covc));
158
159 %% Covariance Inflation
160 ∆ = 0.25; %Constant to artifically inflate covariance
161
162 %% Define g for model observation prediction step
163 %Define the observation matrix
164 G = [eye(2 * num_states) zeros(2*num_states,2*num_theta)];
165
166 %% For loop to constuct
167 m = 1; %initialize month to 1 (january)
168 for t = 1:len_time-1 %do this every time step (month)
169     %% Prediction Step
170     mon = mod(t, 12); %% Determines what month to solve
171
172     %Calculate model update for one month and add noise
173     for n=1:N
174         [x_modelh,x_modelc] = multi_PDE_one_month(theta_predc(1,n),...
175             theta_predc(2,n),theta_predh(1,n),theta_predh(2,n),theta_predh(3,n),...
176             theta_predc(3,n),dt/8,dx,X,x_predh(:,n),x_predc(:,n),mon);
177
178         % Adding noise to the model outputs
179         x_predh(:,n) = x_modelh + c_std * randn(num_states, 1);
180         x_predc(:,n) = x_modelc + c_std * randn(num_states, 1);
181     end
182
183     %Stack z with herring and cod states and parameters
184     z = [x_predh; x_predc; theta_predh; theta_predc];
185     z_mean = mean(z, 2);
186
187     diff = z - z_mean;
```

```
188      z_cov = (diff * diff')/(N-1);
189      z_cov = (1+ ∆) * z_cov;
190
191
192
193      %% Analysis Step
194
195      if mon ==  4 || mon == 10 %% Analysis Step for April and October months
196
197         % New y_hat augmented
198          y_hat = G *z;
199          noisy_data = data_obs(:, m) + d_std * randn(2* num_states, N);
200
201         % Kalman Gain
202          KG = z_cov * G' / (G*z_cov*G' + D);
203
204 %% Compute combined posterior ensemble
205 z = z + KG *(noisy_data - y_hat);
206 z_mean = mean(z, 2);
207          m = m+1;
208
209      end
210
211      x_predh = z(1:num_states,:); %update initial states based on output
212      x_predc = z(num_states + 1:2*num_states,:);
213
214      % Update initial parameters based on output
215      theta_predh = z(2*num_states+1: 2 * num_states + num_theta,:);
216      theta_predc = z(2*num_states+num_theta+1:end, :);
217
218      % Computing Covariances
```

```
219     z_final = z - z_mean;
220     covariances = z_final * z_final' / (N-1);
221     %take square root of diagonals for covariance matrix
222     covariances2 = sqrt(diag(covariances));
223
224
225     % Saving the posterior ensemble
226     Structure.xh(:,:,t+1) = z(1:num_states,:);
227     Structure.xc(:,:,t+1) = z(num_states+1:2*num_states, :);
228     % Saving the mean of the posterior ensemble
229     Structure.x_meanh(:,t+1) = z_mean(1:num_states,:);
230     Structure.x_meanc(:,t+1) = z_mean(num_states+1:2*num_states, :);
231     Structure.xstdh(:, t+1) = covariances2(1:num_states,:);
232     Structure.xstdc(:, t+1) = covariances2(num_states+1:2*num_states, :);
233
234     %Saving the posterior ensemble for parameters
235     Structure.thetah(:,:, t+1) = z(2*num_states+1:2*num_states+num_theta, :);
236     Structure.thetac(:,:, t+1) = z(2*num_states+num_theta+1:end, :);
237     %Saving the mean of the posterior ensemble for parameters
238     Structure.thetameanh(:, t+1) = z_mean(2*num_states+1:2*num_states+num_theta, :);
239     Structure.thetameanc(:, t+1) = z_mean(2*num_states+num_theta+1:end, :);
240     %Saving the standard deviations of the parameters
241     Structure.thetastdh(:, t+1) = covariances2(2*num_states+1:2*num_states+num_theta, :);
242     Structure.thetastdc(:, t+1) = covariances2(2*num_states+num_theta+1:end, :);
243
244 end
245
246 %% Plot Parameters
247
248 % Carring Capacity
249  K_predh = Structure.thetameanh(1,:);
```

```
250  K_predc = Structure.thetameanc(1,:);
251
252  std_Kh = Structure.thetastdh(1,:);
253  std_Kc = Structure.thetastdc(1,:);
254
255  % Carrying Capacity - Herring
256  figure(3)
257  plot(time,true_K_h*ones(size(time)),'-k','LineWidth',2.5); hold on;
258  plot(time,K_predh,'-r','LineWidth',2);
259  plot(time,K_predh+2*std_Kh,'--r','LineWidth',2);
260  plot(time,K_predh-2*std_Kh,'--r','LineWidth',2);
261  hold off
262  title('Herring Carrying Capacity')
263
264  % Carrying Capacity - Cod
265  figure(4)
266  plot(time,true_K_c*ones(size(time)),'-k','LineWidth',2.5); hold on;
267  plot(time,K_predc,'-r','LineWidth',2);
268  plot(time,K_predc+2*std_Kc,'--r','LineWidth',2);
269  plot(time,K_predc-2*std_Kc,'--r','LineWidth',2);
270  hold off
271  title('Cod Carrying Capacity')
272
273 % Diffusion Constant
274  D_predh = Structure.thetameanh(2,:);
275  D_predc = Structure.thetameanc(2,:);
276
277  std_Dh = Structure.thetastdh(2,:);
278  std_Dc = Structure.thetastdc(2,:);
279
280 % Diffusion Constant - Herring
```

```
figure(5)
plot(time,true_D_h*ones(size(time)),'-k','LineWidth',2.5); hold on;
plot(time,D_predh,'-r','LineWidth',2);
plot(time,D_predh+2*std_Dh,'--r','LineWidth',2);
plot(time,D_predh-2*std_Dh,'--r','LineWidth',2);
hold off
title('Herring Diffusion Constant')

% Diffusion Constant - Cod
figure(6)
plot(time,true_D_c*ones(size(time)),'-k','LineWidth',2.5); hold on;
plot(time,D_predc,'-r','LineWidth',2);
plot(time,D_predc+2*std_Dc,'--r','LineWidth',2);
plot(time,D_predc-2*std_Dc,'--r','LineWidth',2);
hold off
title('Cod Diffusion Constant')

% Beta Term
beta_predh = Structure.thetameanh(3,:);
std_betah = Structure.thetastdh(3,:);

% Beta Term - Herring
figure(7)
plot(time,true_beta*ones(size(time)),'-k','LineWidth',2.5); hold on;
plot(time,beta_predh,'-r','LineWidth',2);
plot(time,beta_predh+2*std_betah,'--r','LineWidth',2);
plot(time,beta_predh-2*std_betah,'--r','LineWidth',2);
hold off
title('Beta Interaction Term')

% Epsilon Term
```

```
Ep_predc = Structure.thetameanc(3,:);
std_Epc = Structure.thetastdc(3,:);

figure(8)
plot(time,true_epsilon*ones(size(time)),'-k','LineWidth',2.5); hold on;
plot(time,Ep_predc,'-r','LineWidth',2);
plot(time,Ep_predc+2*std_Epc,'--r','LineWidth',2);
plot(time,Ep_predc-2*std_Epc,'--r','LineWidth',2);
hold off
title('Epsilon Interaction Term')

%% Plot states
% Mean of Herring ensembles
figure(9)
surf(time,space,Structure.x_meanh);
shading interp
title('Herring EnKF Mean Solution');
colorbar

% Mean of Cod ensembles
figure(9)
surf(time,space,Structure.x_meanc);
shading interp
title('Cod EnKF Mean Solution');
colorbar


%% Saving the structure, true solution, and synthetic data in a .mat file
struct = Structure; %Save structure as variable name
save('MS_PDE_run3', 'struct', 'c_data', 'h_data', 'true_sol_c','true_sol_h');

```

```
343 toc
```

```
1  % Creating the alpha function for Cod in the PDE model
2  % Gives a vector of alpha over time
3  %clear; clc; close all;
4
5  function [alpha_c,t_period] = AlphaC_AA(k,t_end,a_c)
6
7  time  = 0:k:t_end; %time vector
8  ntime = length(time);
9  t_period = 1/k;
10
11 sigma = .025; %Define standard deviation of gaussian
12
13 mean_c = time(floor((4*t_period)/12)); %Mean of Gaussian
14
15 t_0 = floor((2*ntime)/(12*t_end)); % first value
16 t_1 = floor((6*ntime)/(12*t_end)); % last value
17
18 alpha_c_year = zeros(1,t_period); %Define a year-long vector of zeros
19
20 for n=t_0:t_1
21 %Birth rate is a Gaussian during the spawining season
22     alpha_c_year(1,n) = (1/(sqrt(2*pi*sigma^2)))*...
23         exp(-((time(n)-mean_c)^2)/(2*sigma^2));
24 end
25
26 alpha_c = alpha_c_year;
27 num_full_reps = floor(ntime/t_period); %Number of full years in time period
28
```

```
for j=1:num_full_reps-1
    alpha_c = [alpha_c,alpha_c_year]; %Repeat one-year vector
end

%Adds on additional values if the end time is not a whole number of years
remaining_cols = rem(ntime,t_period);
alpha_c_add = alpha_c_year(:,1:remaining_cols);
alpha_c = [alpha_c,alpha_c_add];

alpha_c = a_c*alpha_c; %Scale by growth rate scale
```

```
% Creating the alpha function for Cod in the PDE model
% Generates an alpha value for a single month depending on the month you want
% Used in the Herring_PDE_one_month code

%clear; clc; close all;

function [alpha_c] = AlphaC_month_AA(k,a_c,mon,time)
%Define time for a full year
t_end_2 = 1;
time_2  = 0:k:t_end_2;
t_period_2 = 1/k;

ntime = length(time);

if mon≥2 && mon≤6 %if its spawning season
    sigma = .025; %Standard deviation of Gaussian
    mean_h = time_2(floor((4*t_period_2)/12)); %Mean of Gaussian

    alpha_c_month = zeros(1,ntime); %Define one month-long vector
```

```
20
21      for n=1:ntime
22          %Birth rate is a Gaussian during the spawining season
23          alpha_c_month(1,n) = (1/(sqrt(2*pi*sigma^2)))...
24              *exp(-((time(n)-mean_h)^2)/(2*sigma^2));
25      end
26
27  else
28
29      alpha_c_month = zeros(1,ntime); %Zeros if it isn't spawning season
30  end
31      alpha_c = a_c*alpha_c_month; %Scale by growth rate scale
```

```
1   %Code for initial condition vector for Herring
2   %Will be fed into the main file for the Kalman Filter
3   %Made on 7/13/18
4
5   %% Define space vector
6   X = 4; %The length of space
7   dx = 0.1; %the space step increment
8   space = 0:dx:X; %space vector
9
10  %% Define initial condition
11  IC_h = 5*(space(1:end).*(3-space(1:end))).^6.*normpdf(space(1:end),0,1)+10;
12  IC_c = (space(1:end).*(3-space(1:end))).^6.*normpdf(space(1:end),0,1)+10;
13
14  %% Transpose initial condition
15  IC_h = IC_h';
16  IC_c = IC_c';
17
```

```
save('ic_states_multi','IC_h','IC_c')
```

```
function [data_obs_h,data_obs_c,time_obs,model_val_h,model_val_c,time,space]=...
    multi_PDE_data_generate_v2(k,t_end,h,x_end,IC_h,IC_c)
%clear;clc;close all;

tic

%% Define parameters for cod
parameters=load('Multispecies_parameters.txt');
K_c = parameters(1); %Carrying capacity
mu_c = parameters(2); %Fishing mortality
D_c = parameters(3); %Diffusion constant
a_c = parameters(4); %Alpha scaling factor
mag_c = parameters(5); %magnitude of velocity

%% Define parameters for herring
%herring_parameters=load('Herring_parameters.txt');
K_h = parameters(6); %Carrying capacity
mu_h = parameters(7); %Fishing mortality
D_h = parameters(8); %Diffusion constant
a_h = parameters(9); %Alpha scaling factor
mag_h = parameters(10); %Velocity magnitude

%% Define other parameters
beta = parameters(11); %predation rate
epsilon = parameters(12); %Conversion efficiency

noiselev = 0.01; % noise level for synthetic data

```

```
%% Time discretization
%time vector
time  = 0:k:t_end;
ntime = length(time);

%Time vector for bootstrapping
k2 = .5*k^2;
t_bar = 0:k2:k;
nt_bar = length(t_bar);

tau = 1; %Counter

%% Spatial discretization
space = 0:h:x_end;
nspace = length(space);
M = nspace;

%% Time_obs, the time values we take observations (Aprils and Octobers)
time_obs = [];

pd = 1/k;
fac = pd/12;
for m = 1:ntime
    if ( round(mod(pd*time(m),pd)) == 4*fac || ...
            round(mod(pd*time(m),pd)) == 10*fac )
        time_obs = [time_obs, time(m)];
    end
end

num_obs = length(time_obs);
%% Define common constants
```

```
r = k/(2*h*h);
d_con_c = D_c*r; %Constant arising from the Crank-Nicolson
d_con_h = D_h*r;

%% Call birth rate functions

%vector of birth rate values for cod
[alpha_c,~] = AlphaC_AA(k,t_end,a_c);
%vector of birth rate values for herring
[alpha_h,t_period] = AlphaH_AA(k,t_end,a_h);


%% Define relevant matrices

%Coefficient matrix due to Crank-Nicolson for V^n+1
CN_quadrant = diag(ones(M-1, 1), 1) - 2*diag(ones(M,1)) + ...
    diag(ones(M-1, 1), -1);
%Impose zero-slope boundary conditions
CN_quadrant(1,2) = 2*CN_quadrant(1,2);
CN_quadrant(end,end-1) = 2*CN_quadrant(end,end-1);

CN_right = [d_con_h*CN_quadrant,zeros(M);zeros(M),d_con_c*CN_quadrant];
CN_left = -1*CN_right;
%Identity matrix to represent the time approx for V^n+1
Forward_time_left = diag(ones(2*M,1));

%Total atrix with the coefficients of the solution at time n+1
M_lhs = Forward_time_left + CN_left;

%Identity matrix representing the V^n term due to the time approx

```

```
Forward_time_right = diag(ones(2*M,1));

M_rhs_base = Forward_time_right + CN_right;

%% Bootstrapping matrices
r2 = k2/(2*h*h);
d_con2_c = D_c*r2; %Constant from the Crank-Nicolson for bootstrapping time step
d_con2_h = D_h*r2;


%Coefficient matrix due to Crank-Nicolson for V^n+1

CN_right2 = [d_con2_h*CN_quadrant,zeros(M);zeros(M),d_con2_c*CN_quadrant];
CN_left2 = -1*CN_right2;
%Identity matrix to represent the time approx for V^n+1
Forward_time_left2 = diag(ones(2*M,1));

%Total atrix with the coefficients of the solution at time n+1
M_lhs2 = Forward_time_left2 + CN_left2;

%Identity matrix representing the V^n term due to the time approx

Forward_time_right2 = diag(ones(2*M,1));

M_rhs2_base = Forward_time_right2 + CN_right2;

%% Alpha/carrying capcity and beta/epsilon matrices
%Matrices representing predator-prey interaction terms
BetaEpsilon = k*[-1*beta*eye(M),zeros(M);zeros(M),epsilon*eye(M)];
BetaEpsilon2 = k2*[-1*beta*eye(M),zeros(M);zeros(M),epsilon*eye(M)];

```

```
122 %% Initial and boundary conditions
123
124 IC1 = IC_h; %herring
125 IC2 = IC_c; %cod
126
127 %% Define the solution matrix
128
129 V_save = zeros(2*M, ntime);  %Creates a matrix to store values for V in
130 V_save(1:M,1) = IC1'; %Changes the first column to the initial conditions
131 V_save(M+1:end,1) = IC2';
132
133 data_obs_h = zeros(M,num_obs); %Herring synthetic data matrix
134 data_obs_c = zeros(M,num_obs); %Cod synthetic data matrix
135
136 %% Define the Bootstrapping solution matrix
137
138 V_bar = zeros(2*M,nt_bar);
139 V_bar(1:M,1) = IC1';
140 V_bar(M+1:end,1) = IC2';
141
142 %% Update by time step
143 for n=1:ntime-1
144     m2 = mod(n,t_period);
145     %Call velocities for cod and herring
146     v_c = velocity_c(n,k,t_end,mag_c);
147     v_h = velocity_h(n,k,t_end,mag_h);
148
149     a_con_c = v_c*k/h; %Constant arising from upwinding
150     a_con_h = v_h*k/h;
151
152     %Upwinding matrix for single species
```

```
153     UW_quadrant = diag(ones(M-1, 1), -1) - diag(ones(M,1));
154     %Impose zero-slope boundary conditions
155     UW_quadrant(1,:) = 0;
156     UW_quadrant(end,:) = 0;
157     %Combine UW_quadrants for herring and cod
158     UW_right = [a_con_h*UW_quadrant,zeros(M);zeros(M),a_con_c*UW_quadrant];
159 %Reaction term matrix
160     React_right = [(k*alpha_h(n) - k*mu_h)*diag(ones(M,1)),zeros(M);...
161         zeros(M),(k2*alpha_c(n) - k2*mu_c)*diag(ones(M,1))];
162
163
164     %Repeat for the bootstrapping time step
165     a_con2_c = v_c*k2/h;
166     a_con2_h = v_h*k2/h;
167     UW_right2 = [a_con2_h*UW_quadrant,zeros(M);zeros(M),a_con2_c*UW_quadrant];
168
169     React_right2 = [(k2*alpha_h(n) - k2*mu_h)*diag(ones(M,1)),zeros(M);...
170         zeros(M),(k2*alpha_c(n) - k2*mu_c)*diag(ones(M,1))];
171
172     M_rhs = M_rhs_base + UW_right + React_right;
173     M_rhs2 = M_rhs2_base + UW_right2 + React_right2;
174
175     V_save_flip = [V_save(M+1:end,n);V_save(1:M,n)];
176     AlphaK = k*[(alpha_h(n)/K_h)*eye(M),zeros(M);zeros(M),(alpha_c(n)/K_c)*eye(M)];
177     AlphaK2 = k2*[(alpha_h(n)/K_h)*eye(M),zeros(M);zeros(M),(alpha_c(n)/K_c)*eye(M)];
178
179     if n == 1 %Implement bootstrapping on first iteration of time loop
180         for m=1:nt_bar-1
181
182             V_bar_flip = [V_bar(M+1:end,m);V_bar(1:M,m)];
183             V_bar(:,m+1) = M_rhs2*V_bar(:,m) - AlphaK2*V_bar(:,m).*V_bar(:,m) + ...
```

```
                BetaEpsilon2*V_bar(:,m).*V_bar_flip;

            V_bar(:,m+1) = M_lhs2\V_bar(:,m+1);
        end

            V_save(:,2) = M_rhs*V_bar(:,end) - AlphaK*V_bar(:,end).*V_bar(:,end) + ...
                BetaEpsilon*V_bar(:,end).*V_bar_flip;

            V_save(:,2) = M_lhs\V_save(:,2);

    else

      %Otherwise implement regular update

        V_save(:,n+1) = M_rhs*V_save(:,n) - AlphaK*V_save(:,n).*V_save(:,n) +...
            BetaEpsilon*V_save(:,n).*V_save_flip;

        V_save(:,n+1) = M_lhs\V_save(:,n+1);

    end

     if m2==round(4*t_period/12) || m2==round(10*t_period/12)
         %Add noise and save data only in April and October
        sz = size(V_save(1:M,n+1));
        data_obs_h(:,tau) = V_save(1:M,n+1) + noiselev*randn(sz);
        data_obs_c(:,tau) = V_save(M+1:end,n+1) + noiselev*randn(sz);
        tau = tau+1; %Add one to count
    end

end
```

```
%% Split into Cod and Herring solutions

V_herring = V_save(1:M,:);
V_cod = V_save(M+1:end,:);

%Save true solutions for output
model_val_h = V_herring;
model_val_c = V_cod;

%% Plots
% Atlantic Herring
figure(1);
surf(time,space,V_herring)
%view(2);
xlabel('time'); ylabel('space'); zlabel('\rho_h(x,t)');
title('Atlantic Herring');
set(gca,'FontSize',20);
shading interp
colorbar
cbar_lims = caxis;

% Plot the Cod
figure(2);
surf(time,space,V_cod)
%view(2);
xlabel('time'); ylabel('space'); zlabel('\rho_c(x,t)');
title('Atlantic Cod');
set(gca,'FontSize',20);
shading interp
colorbar
caxis(cbar_lims);
```

```
246
247 toc
```

```
1  function [one_month_sol_h,one_month_sol_c]=...
2      multi_PDE_one_month(K_c,D_c,K_h,D_h,beta,epsilon,k,h,x_end,IC_h,IC_c,mon)
3
4  %% Define parameters
5  parameters=load('Multispecies_parameters.txt');
6  mu_c = parameters(2); %Fishing mortality
7  a_c = parameters(4); %Alpha scaling factor
8  mag_c = parameters(5); %Magnitude of the velocity
9
10 %% Define parameters for herring
11 mu_h = parameters(7); %Fishing mortality
12 a_h = parameters(9); %Alpha scaling factor
13 mag_h = parameters(10); %Velocity magnitude
14
15
16 %% Time discretization
17 %Ensure month is 12 and not zero, it comes from mod
18 if mon == 0
19     mon = 12;
20 end
21 time  = (mon-1)/12:k:(mon/12); %from beginning to end of month
22 ntime = length(time);
23
24 %Bootstrapping time
25 k2 = .5*k^2;
26 t_bar = 0:k2:k;
27 nt_bar = length(t_bar);
```

```
28
29 %% Spatial discretization
30
31 space = 0:h:x_end;
32 nspace = length(space);
33 M = nspace;
34
35 %% Define common constants
36 r = k/(2*h*h);
37 d_con_c = D_c*r; %Constant arising from the Crank-Nicolson
38 d_con_h = D_h*r;
39 [alpha_c] = AlphaC_month_AA(k,a_c,mon,time); %cod growth rate vector
40 [alpha_h] = AlphaH_month_AA(k,a_h,mon,time); %herring growth rate vector
41
42 %% Define relevant matrices
43
44 %Coefficient matrix due to Crank-Nicolson for V^n+1
45 CN_quadrant = diag(ones(M-1, 1), 1) - 2*diag(ones(M,1)) + ...
46     diag(ones(M-1, 1), -1);
47 %Impose zero-slope boundary conditions
48 CN_quadrant(1,2) = 2*CN_quadrant(1,2);
49 CN_quadrant(end,end-1) = 2*CN_quadrant(end,end-1);
50
51 CN_right = [d_con_h*CN_quadrant,zeros(M);zeros(M),d_con_c*CN_quadrant];
52 CN_left = -1*CN_right;
53 %Identity matrix to represent the time approx for V^n+1
54 Forward_time_left = diag(ones(2*M,1));
55
56 %Total atrix with the coefficients of the solution at time n+1
57 M_lhs = Forward_time_left + CN_left;
58
```

```
%Identity matrix representing the V^n term due to the time approx

Forward_time_right = diag(ones(2*M,1));

M_rhs_base = Forward_time_right + CN_right;

%% Bootstrapping matrices
r2 = k2/(2*h*h);
d_con2_c = D_c*r2; %Constant from the Crank-Nicolson for bootstrapping time step
d_con2_h = D_h*r2;

%Coefficient matrix due to Crank-Nicolson for V^n+1

CN_right2 = [d_con2_h*CN_quadrant,zeros(M);zeros(M),d_con2_c*CN_quadrant];
CN_left2 = -1*CN_right2;
%Identity matrix to represent the time approx for V^n+1
Forward_time_left2 = diag(ones(2*M,1));

%Total atrix with the coefficients of the solution at time n+1
M_lhs2 = Forward_time_left2 + CN_left2;

%Identity matrix representing the V^n term due to the time approx

Forward_time_right2 = diag(ones(2*M,1));

M_rhs2_base = Forward_time_right2 + CN_right2;

%% Alpha/carrying capcity and beta/epsilon matrices
%Matrices for predator-prey terms
BetaEpsilon = k*[-1*beta*eye(M),zeros(M);zeros(M),epsilon*eye(M)];
BetaEpsilon2 = k2*[-1*beta*eye(M),zeros(M);zeros(M),epsilon*eye(M)];
```

```
90
91  %% Define the solution matrix
92
93  V_save = zeros(2*M, ntime);  %Creates a matrix to store values for V in
94  V_save(1:M,1) = IC_h'; %Changes the first column to the initial conditions
95  V_save(M+1:end,1) = IC_c';
96
97  one_month_sol_h = zeros(M,1);
98  one_month_sol_c = zeros(M,1);
99  %% Define the Bootstrapping solution matrix
100
101 V_bar = zeros(2*M,nt_bar);
102 V_bar(1:M,1) = IC_h';
103 V_bar(M+1:end,1) = IC_c';
104
105 %% Update by time step
106 for n=1:ntime-1
107     %Call velocity functions to get velocity values
108     v_c = velocity_c_month(mag_c,n,mon,time);
109     v_h = velocity_h_month(mag_h,n,mon,time);
110
111     a_con_c = v_c*k/h; %Constant arising from upwinding
112     a_con_h = v_h*k/h;
113
114     %Upwinding matrix for single species
115     UW_quadrant = diag(ones(M-1, 1), -1) - diag(ones(M,1));
116     %Impose zero-slope boundary conditions
117     UW_quadrant(1,:) = 0;
118     UW_quadrant(end,:) = 0;
119     %Combine UW_quadrants to get a matrix for both species
120     UW_right = [a_con_h*UW_quadrant,zeros(M);zeros(M),a_con_c*UW_quadrant];
```

```
121
122     %Reaction term matrix
123     React_right = [(k*alpha_h(n) - k*mu_h)*diag(ones(M,1)),zeros(M);...
124         zeros(M),(k2*alpha_c(n) - k2*mu_c)*diag(ones(M,1))];
125
126
127     %Repeat for bootstrapping
128     a_con2_c = v_c*k2/h;
129     a_con2_h = v_h*k2/h;
130     UW_right2 = [a_con2_h*UW_quadrant,zeros(M);zeros(M),a_con2_c*UW_quadrant];
131
132     React_right2 = [(k2*alpha_h(n) - k2*mu_h)*diag(ones(M,1)),zeros(M);...
133         zeros(M),(k2*alpha_c(n) - k2*mu_c)*diag(ones(M,1))];
134
135     M_rhs = M_rhs_base + UW_right + React_right;
136     M_rhs2 = M_rhs2_base + UW_right2 + React_right2;
137
138     V_save_flip = [V_save(M+1:end,n);V_save(1:M,n)];
139     AlphaK = k*[(alpha_h(n)/K_h)*eye(M),zeros(M);zeros(M),(alpha_c(n)/K_c)*eye(M)];
140     AlphaK2 = k2*[(alpha_h(n)/K_h)*eye(M),zeros(M);zeros(M),(alpha_c(n)/K_c)*eye(M)];
141
142     if n == 1 %Bootstrapping on first iteration of time loop
143         for m=1:nt_bar-1
144
145             V_bar_flip = [V_bar(M+1:end,m);V_bar(1:M,m)];
146             V_bar(:,m+1) = M_rhs2*V_bar(:,m) - AlphaK2*V_bar(:,m).*V_bar(:,m) + ...
147                 BetaEpsilon2*V_bar(:,m).*V_bar_flip;
148
149             V_bar(:,m+1) = M_lhs2\V_bar(:,m+1);
150         end
151
```

```
            V_save(:,2) = M_rhs*V_bar(:,end) - AlphaK*V_bar(:,end).*V_bar(:,end) + ...
                BetaEpsilon*V_bar(:,end).*V_bar_flip;

            V_save(:,2) = M_lhs\V_save(:,2);

    else

        %Otherwise update as normal

        V_save(:,n+1) = M_rhs*V_save(:,n) - AlphaK*V_save(:,n).*V_save(:,n) + ...
            BetaEpsilon*V_save(:,n).*V_save_flip;

        V_save(:,n+1) = M_lhs\V_save(:,n+1);

    end


end

%% Split into Cod and Herring solutions

V_herring = V_save(1:M,:);
V_cod = V_save(M+1:end,:);

one_month_sol_h = V_herring(:,end);
one_month_sol_c = V_cod(:,end);
```

```
% Code to generate synthetic data for Herring PDE
clear;clc;close all;
%% Parameters
```

```
%Space vector
h = 0.1;
x_end = 4;
space = 0:h:x_end;

%time vector
k = 1/(12*8); %1/12 divided by some power of 2
t_end = 15; %15 years
time = 0:k:t_end;

%% Call the Function

%Initial conditions
IC_h = 5*(space.*(3-space)).^6.*normpdf(space,0,1)+10;
IC_c = (space.*(3-space)).^6.*normpdf(space,0,1)+10;

%Call function to generate synthetic data
[h_data,c_data,time_obs,true_sol_h,true_sol_c,true_time,true_space] = ...
    multi_PDE_data_generate_v2(k,t_end,h,x_end,IC_h,IC_c);

%Save as .mat file
save('obsData_multi_2','h_data','c_data','true_sol_h','true_sol_c',...
    'true_time','true_space')

figure(3)
surf(time,space,true_sol_h);
shading interp

figure(4)
surf(time,space,true_sol_c);
shading interp
```

```
colorbar
```

```
% Velocity function for the Atlantic Herring
% Break up into 12 month time period, assumes final time is 1
% Assumes the model will begin in January
% Used in the Herring_PDE_one_month code

function v_c = velocity_c_month(mag_c,n,mon,time)

b = 200;

%% Region 1, January through the middle of spring movement
if mon≤3
    v_c = .5*(1+tanh(b*(time(n)-2/12)));
end

%% Region 2, Middle of spring movement through middle of spawning
if 3<mon  && mon≤4
    v_c = -.5*(-1+tanh(b*(time(n)-3/12)));
end

%% Region 3, Middle of spawning through middle of summer movement
if 4<mon && mon≤6
    v_c = -.5*(1+tanh(b*(time(n)-5/12)));
end

%% Region 4, Middle of summer movement through end of year
if 6<mon && mon≤12
    v_c = .5*(-1+tanh(b*(time(n)-6/12)));
end
```

```
29
30 v_c = mag_c*v_c; %Scale by velocity magnitude
```

A.4 Model Comparison Code

```
1  % Script to run Comparison 1
2
3  clc;close all;clear;
4
5  % Load time
6  T = 15; %The number of years
7  dt = 1/(12*8); %time increment will be one month
8  time = 0:dt:T; %time vector
9
10 % Load space
11 X = 4; %The length of space
12 dx = 0.1; %the space step increment
13 space = 0:dx:X; %space vector
14
15 % Load .mat files
16 % Cod file for comparison
17 load('Herring_PDE_for_comparison.mat');
18
19 % Observation data from multispecies
20 load('obsData_multi.mat');
21
22
23 % Plot the true solution from multispecies
24 figure(1)
25 surf(true_time,true_space,true_sol_h);
```

```
view(2)
title('Atlantic Herring True Solution');
set(gca,'FontSize',20);
shading interp
colorbar
h_PDE = caxis;

% Plot the Herring for Comparison
figure(2)
surf(time,space,U_save);
view(2)
title('Atlantic Herring');
set(gca,'FontSize',20);
shading interp
colorbar
caxis(h_PDE);

% Calculate absolute error
Herring_PDE_error = abs(U_save - true_sol_h);

% Plot the absolute error
figure(3)
%surf(time,space,Herring_PDE_error);
surf(time(1:480),space,Herring_PDE_error(:,1:480)); % Plots the first 5 years
view(2)
title('Atlantic Herring Absolute Error');
set(gca,'FontSize',20);
shading interp
colorbar
```