

MASTEROPPGAVE I INFORMATIKK – OPTIMERING

VÅREN 2012

Modelling migration patterns of fish using depth and temperature preferences

Erik Natvig



uiblogo/UiBmerke_grayscaleV8.pdf

Universitetet i Bergen

Abstract

Time series of depth and temperature derived from electronic tagging of fish have been used to construct a stochastic model that aims at capturing main characteristics of the observations. Mixed Ornstein-Uhlenbeck process models are used to model attraction towards different concentration points in the depth/temperature plane, and a methodology to determine model parameters is presented. Simulations of the model displays very similar dynamics to the original data. Further, an optimization problem for finding a path expressing the geographical location of the tagged fish is formulated. An interpolation procedure using thin-plate splines for interpolating an atlas over temperature in the ocean is introduced. As general-purpose optimization solvers fail to find optimal solutions to the problem, a special-purpose algorithm, based on an ensemble search, is developed. The algorithm solves the problem to optimality, both for test instances and for real data, but demonstrates that there may be many radically different paths through the ocean that match the temperature and depth time series. The algorithm has a potential of making good estimates on the geolocation of fish provided external information is used to guide the algorithm and to select the most likely solutions.

Preface

This Master of Science project came to be because I wanted to apply my favourite mathematical subject, optimization, to solving problems within another interest of mine, protection of the environment. Luckily, Dag Haugland at the Department of Informatics knew how, and Sam Subbey from the Institute of Marine Research was involved to suggest a very interesting project.

The work has given important insight into the world of uncertainty. Accepting the presence of uncertainty, and forever giving up all hope of eliminating it, is crucial in order to understand that a mathematical model for real phenomena can and should never attempt to give a complete description of reality.

I would like to thank Sam for making this thesis possible by providing many ideas and intuition. Also, thanks for asking nasty questions forcing me to rethink the work I had done, and either reconsider it (because I realized it was wrong) or continue working on it (having been convinced that I was on the right track). Thanks also for co-writing the paper accepted at Norsk informatikkonferanse and for great support in the final stages of the writing.

Big thanks also go to Dag, whose contribution in discussions, reading my thesis, pointing out points for improvement, and for questioning the reasoning of Sam and myself when needed, has been invaluable. Also, without Dag's encouragement, I would never have thought of submitting a paper to Norsk informatikkonferanse, which resulted in a publication.

Other people at the Institute of Marine Research deserve thanks. Kathrine Michalsen has been responsible for the fish tagging experiments, and explained to me how they were done. Cecilie Hansen and Vidar Lien helped me get what I needed

out of the atlas over depth and temperature.

Thanks to Dag Tjøstheim at the Department of Mathematics, University of Bergen, for a discussion on time series and stochastic processes, and for taking an interest in my thesis work.

I want to thank my friend André Lynum for discussions on my work, providing new ideas on solving tough problems, and for reading parts of my thesis, giving valid comments particularly on making the text more accessible. Thanks also go to my friend Jon Nyfløtt for proof-reading parts of the thesis.

Thanks to my parents Kristin and Richard Natvig for raising me as an academic, completely without pressure on topics such as “what to do with myself” and for providing a home for me while visiting Bergen.

The biggest thanks go to my dear Heidimarie, who has both taken a keen interest in my work and also supported me through the most demanding periods by being there for me with lots of love, reminders to take breaks or go to sleep, and with a large capacity for housework. Without her support, the writing process would have been much more arduous. Also thanks to Heidimarie for loads of proof-reading and for laughing at some of my long and twisted sentences and arcane mathematical terminology.

Erik Natvig, Oslo, April 26, 2012

Contents

1	Introduction	11
1.1	Why model fish movement?	11
1.2	Electronic tagging	12
1.3	Analysis challenges	12
1.4	Geolocalization	13
1.5	Structure of the thesis	13
1.6	Notation, conventions and acronyms	13
2	Theory and definitions	17
2.1	Stochastic processes	17
2.2	Markov processes	19
2.3	Stochastic differential equations	21
2.4	The Ornstein-Uhlenbeck process	22
2.5	Markov chains	28
2.6	Time series analysis	31
2.7	Interpolation	38
2.8	Constrained optimization	39
2.9	Biological background information	40
2.10	Geography	41
3	Previous work	43
3.1	Stochastic modelling of wildlife movements	43
3.2	Mixed Ornstein-Uhlenbeck process models	44
3.3	Analyzing migratory behaviour of cod	48

I	Stochastic modelling of fish migration	51
4	Basic data analysis and modelling approach	53
4.1	Data overview	53
4.2	Basic one-dimensional model	56
4.3	Extending the model and inference methods to 2-D	62
4.4	Chapter summary	67
5	Model validation and improvements	69
5.1	Simulation and validation of the 1-D model	70
5.2	Creating a mean model path	80
5.3	Model improvements	83
5.4	Parameter estimation methods	91
5.5	Final model simulation results	96
5.6	Summary and conclusion	101
II	Geolocalization of fish by optimization	103
6	Geolocalization: definitions, formulations and atlases	105
6.1	Simplifications and basic definitions	105
6.2	The simplified geolocalization problem	107
6.3	The ROMS atlas	108
6.4	Test instances for testing optimization methods	115
6.5	Towards the original geolocalization problem	116
6.6	Maximum travelling distance for the fish	119
6.7	Solution methods for the optimization problem	120
6.8	Summary	126
7	Interpolation and algorithm for geolocalization	127
7.1	Temperature atlas interpolation	127
7.2	Test instances	133
7.3	Solving the problem using standard solvers	135

7.4	Analysis on one time step	136
7.5	Suggested algorithm for geolocalization	138
7.6	Discussion on algorithm	142
7.7	Testing the algorithm	145
7.8	Results of geolocalization on real data	152
7.9	Discussion	155
 III Suggestions for future work		159
 8 Future work		161
8.1	More on the stochastic state model	161
8.2	Extending the model to several fish	162
8.3	Other choices for parameter estimation	166
8.4	Movement to the far side of a concentration point	167
8.5	Improvements to geolocalization algorithm	168
8.6	Incorporating prior knowledge	169
8.7	Correcting infeasible solution	170
8.8	Particle swarm optimization for geolocalization	170
8.9	Considering uncertainties	171
8.10	Validating the geolocalization algorithm	172
8.11	Geolocalization by migration model	172
8.12	Geolocalization by other factors	173
 A Article published at NIK		175
 B Algorithms		189
B.1	Estimating a transition matrix	189
B.2	Simulating a Markov chain	190
B.3	Local maxima of a matrix	190
B.4	Detailed description of geolocalization algorithm	190

C	Implementation issues	195
C.1	2-D kernel density estimation	195
C.2	Adaptive-degree polynomial filter	196
C.3	Implementation of spline interpolation	196
C.4	Running time of geolocalization algorithm	197
C.5	Atlas matrices	197

Chapter 1

Introduction

This thesis is about modelling and analyzing movements of fish. The data used is from electronic tagging of an Atlantic cod, belonging to a stock of Norwegian Coastal cod in the Lofoten-Barents Sea ecosystem.

1.1 Why model fish movement?

Fisheries are important both as food sources and for employment. Modelling fish movement is needed to understand complex interactions in the ecosystem and for making fact-based decisions to ensure that fisheries are sustainable. Understanding the food chain is crucial in order to avoid depleting species that are at the bottom of the chain, which will affect all the organisms in the chain. This requires information on where and when different fish species overlap in space and time.

Knowledge on migration patterns makes can support decisions on e.g. restrictions on fishing that ensure sustainable population levels of each fish stock. The International Council for the Exploration of the Sea (ICES) has since 2004 recommended no fishing of Norwegian Coastal cod [35], because of low stock levels [14]. Knowledge on when and where the fish travels to spawn can be used recommend measures that prevent disturbances in this important part of the life-cycle of the fish.

Different stocks of Atlantic cod can have distinct migration patterns, and are

indistinguishable to the eye. There are also significant differences in the population levels of different stocks of cod, and the stock of Northeast Arctic cod is in good condition. To ensure that it is this stock that is harvested, instead of Norwegian coastal cod, one must know where and when the two different stocks overlap. Knowledge about migration patterns will help protect vulnerable stocks by providing insight on suitable areas for zones with restrictions on fishing.

1.2 Electronic tagging

Electronic tagging of fish is a method to provide long term, high resolution observation data on individual fish behaviour. To obtain such high-resolution data, for fish that moves over large areas and does not stay at the surface, currently the only available method is tags that store the data (data storage tags). The method relies on the fish being caught e.g. through the commercial fisheries for the tag to be returned to the researcher. Such tags do not record position directly, and therefore depth and temperature are the two environmental factors that are used as basis for modelling in this thesis.

1.3 Analysis challenges

The huge amount of data captured by a data storage tag represents challenges for standard time series analysis methodologies. There is for instance much variation in the distribution of the vertical position of the fish, between month and between seasons. Much of the data displays periodic behaviour, but with varying amplitude and frequency.

Some of these issues have been addressed in e.g. [51] and [50]. A major task that remains, however, is to model individual fish behaviour and use this to gain insight on the movement patterns of entire fish stocks. A start for this is the work in part I in this thesis, where a stochastic model is constructed to mimic the movements of a single fish in the depth and temperature plane.

1.4 Geolocalization

It is desirable to obtain information on geographical position of the fish as well as depth and temperature to establish migration patterns of fish [58]. Some previously presented approaches for geolocalization utilize light measurements recorded by data storage tags. A challenge for using such approaches on fish in Arctic waters is that for parts of the year, there is very little light. Another approach involves using tidal variations [37]. However, the tidal difference in sea depth in the Barents sea is relatively small [58, 50], making this approach less relevant.

One previous approach for geolocalization in the ecosystem considered is presented in [58]. Part II in this thesis expands the toolbox of researchers by providing an alternative approach. Information from a data storage tag is combined with an atlas of temperature and depth, and optimization is used to find paths through the ocean where it is possible to observe the characteristics in the time series. This approach is tried on both synthetic examples and on true fish tag data.

1.5 Structure of the thesis

The thesis has the following structure:

- Introductory chapters on theory and previous work
- Part I on stochastic modelling
- Part II on geolocalization
- Part III on suggestions for future work
- Appendixes

1.6 Notation, conventions and acronyms

Notation

$\mathbf{x} = (x_1, x_2, \dots, x_n)^T$: column vector of length n

$\mathbf{D} = (D_1, D_2, \dots, D_N)$: length N time series of depth observations

$\mathbf{T} = (T_1, T_2, \dots, T_N)$: length N time series of temperature observations

$\mathbf{0}$: a vector of length N of zeros

$\mathbf{1}$: a vector of suitable size of only ones

$\mathbb{P}(X = x)$: probability that stochastic variable X attains value x

X_t : discrete-time stochastic process

$X(t)$: continuous-time stochastic process

Conventions

- In Part I, indices for vectors, time series and stochastic processes run from 1, to indicate that observation 1 is the first observation.
- In Part II, the indices run from 0 because the time or day of release is used as starting point for optimization, where the first iteration corresponds to day 1.
- For vectors, the notation $\mathbf{a} \leq \mathbf{b}$ means that the inequality is valid element-wise
- Entire time series are usually considered as row vectors, while other vectors are usually column vectors.

Acronyms

DST Data storage tag

SDE Stochastic differential equation

OU Ornstein-Uhlenbeck (process)

ACF Autocorrelation function

PACF Partial autocorrelation function

KDE Kernel density estimate

ADPF Adaptive-degree polynomial filter

ROMS Regional Ocean Modelling System

Chapter 2

Theory and definitions

This chapter introduces theory and definitions needed in the rest of the thesis.

2.1 Stochastic processes

Stochastic processes describe how one or several characteristics of a system change with time, when there is a randomness in the dynamics. This randomness implies that the dynamics or state of the system after passage of some time, with the same initial values, is not necessarily the same if the experiment or situation is repeated. A time series can be considered as a snapshot or realization of a stochastic process. Sometimes, the two terms are used interchangeably, but the distinction will be kept in this thesis.

2.1.1 Notation, time index sets and state spaces

A stochastic process can be denoted $\{X(t), t \in T\}$, where for each $t \in T$, $X(t)$ is a random variable or vector (see e.g. [43]) which represents the *state* of the process at time t . The set T is the index set of the process. For a discrete-time stochastic process, T is a countable set (either a finite number of elements, or infinite sets such as the natural numbers), and the notation $\{X_t, t = 0, 1, \dots\}$ can be used. If T is an interval of real numbers (and thus an infinite set), then the process is called continuous, and for $T = [0, \infty)$ it can be denoted $\{X(t), t \geq 0\}$. Continuous-time

processes can be considered at any real valued point in time within the interval of T . For easier notation, X_t (discrete-time) and $X(t)$ (continuous-time) will be used in this thesis to denote stochastic processes where the index set T can be understood from the context.

The state space I of a stochastic process X_t is the range of states that the process can have. The space can be limited to a finite number of values, to infinitely countably many (e.g. natural numbers) or to a continuous set of numbers (e.g. the real line).

2.1.2 Expectation, variance and autocovariance

The ideas of expectation and variance of random variables generalize to functions of time for stochastic processes. It is desirable to derive expressions for these functions, $\mathbb{E}[X(t)]$ and $\text{Var}(X(t))$, and also to analyze their behaviour as time tends to infinity.

If $X(t)$ is the solution of a stochastic differential equation (defined in a later section) expectation and variance can be derived directly from the expressions for the solutions of the equation.

Definition 2.1.1. (Autocovariance) The autocovariance function of a stochastic process X_t is defined as

$$\gamma(s, t) \equiv \mathbb{E}[(X_s - \mu_s)(X_t - \mu_t)] \quad (2.1)$$

for all time-points s and t , where μ_s and μ_t are the means at each time-point.

2.1.3 Stationarity

A property that some stochastic processes have are strict and weak stationarity.

Definition 2.1.2. (Strict stationarity) A stochastic process or time series $\{X_t, t = 0, 1, \dots\}$ is strictly stationary if (X_1, \dots, X_n) and $(X_{1+h}, \dots, X_{n+h})$ have the same joint distributions for all positive integers h and $n \geq 0$ [8].

Definition 2.1.3. (Weak stationarity) A stochastic process or time series X_t is weakly stationary if [8]

- the expected value of X_t is independent of t , and
- the autocovariance of X_t is independent of t for any time lag.

If a stochastic process is stationary, its long-term behaviour is called the stationary expectation and variance. For stationary time series, the autocovariance is given in terms of the lag h between time points instead of the time points themselves, i.e.

$$\gamma(h) = \mathbb{E}[(X_{t+h} - \mu)(X_t - \mu)]. \quad (2.2)$$

Definition 2.1.4. The autocorrelation function (ACF) of a stochastic process is defined as

$$\rho(h) = \frac{\gamma(h)}{\gamma(0)} \quad (2.3)$$

This can be thought of as a scaled version of the autocovariance function (2.2). It is restricted to the interval $[-1, 1]$ and a value of zero indicates no autocorrelation. The autocorrelation function of a stationary process is symmetric around the origin, so it is only necessary to consider its values when $h \geq 0$.

2.1.4 White noise

Definition 2.1.5. (White noise) A stochastic process where the values at different points in time are uncorrelated, independent and identically distributed (iid), with finite variance, is called a *white noise process* [27].

When the probability density function generating the values of a white noise process is a normal distribution, the process is called *Gaussian white noise*.

2.2 Markov processes

One special type of stochastic process is a *Markov process* (see [34]). A Markov process X_t is characterized by a memoryless property. A value X_{t+h} is dependent only on X_t and not on X_{t-h} for any positive value of h .

2.2.1 Discrete-time Markov processes

For a discrete-time Markov process, the memoryless property is given by:

$$\mathbb{P}(X_{k+1} = j_{k+1} | X_k = j_k, X_{k-1} = j_{k-1}, \dots, X_0 = j_0) = \mathbb{P}(X_{k+1} = j_{k+1} | X_k = j_k),$$

where j_t is the state of the chain at time t . The property says that the probability of an outcome j_{k+1} of X at time $k + 1$, conditional on the whole history of the process, is equal to the probability of outcome i at time $k + 1$ conditional *only* on the outcome at time k . Further, the Markov Property in [34] gives us

$$\mathbb{P}(X_{n+m} = i | X_n = j) = \mathbb{P}(X_m = i | X_0 = j).$$

Thus the probability of transition from a state j to a state i in time n to time $n + m$ depends only on the time shift n , not on the actual times. This implies that a time-shifted Markov process is also a Markov process.

2.2.2 Continuous-time Markov processes

We now consider continuous-time Markov processes. The memoryless property is now given by

$$\begin{aligned} \mathbb{P}(X(t_{k+1}) = j_{t_{k+1}} | X(t_k) = j_{t_k}, X(t_{k-1}) = j_{t_{k-1}}, \dots) \\ = \mathbb{P}(X(t_{k+1}) = j_{t_{k+1}} | X(t_k) = j_{t_k}) \end{aligned}$$

with $\{t_k, t_{k-1}, \dots\}$ being some previous points in time satisfying $t_{k+1} > t_k > t_{k-1} > \dots$. Thus only the observation closest in time is necessary for defining the probability for the value of the process at the current point in time.

An example of a continuous-time Markov process with a real-valued outcome is a Wiener process, or Brownian motion. The Wiener process is a central component of stochastic differential equations defined in Section 2.3. The following definition is taken from [18]:

Definition 2.2.1. A Wiener process is a process $W(t)$ depending continuously on time t on an interval $[0, T]$ and satisfying the three following conditions:

1. $W(0) = 0$ (with probability 1).
2. For $0 \leq s < t \leq T$, the increment $W(t) - W(s) \sim \mathcal{N}(0, t - s)$, that is, normally distributed with mean 0 and variance $t - s$.
3. Increments at non-overlapping time intervals are independent.

2.3 Stochastic differential equations

Some stochastic processes are derived through solving stochastic differential equations (SDEs). We base the description of these on [18]. An SDE is a differential equation where at least one of the independent variables represents a stochastic process. An SDE differs from a deterministic differential equation, where all variables are deterministic. Thus, the solution of an SDE is not a function, but a stochastic process, and is referred to as a diffusion process.

Wiener processes are central elements in SDEs (see Definition 2.2.1). Since it is nowhere differentiable (with probability 1), one cannot use the familiar d/dt operator, but instead one must define the relation between infinitesimal quantities. A scalar SDE in its most general form is:

$$dX(t) = f(X(t), t)dt + g(X(t), t)dW(t), \quad X(0) = X_0, 0 \leq t \leq T. \quad (2.4)$$

Here, f and g are functions of the process and time and X_0 is the initial condition (which may be a stochastic variable as well). The term $dW(t)$ is an infinitesimal increment of a Wiener process, which is a normally distributed variable with mean 0 and variance dt . Equation (2.4) is a compact and informal way of saying that $X(t)$ solves the stochastic integral equation

$$X(t) = X_0 + \int_0^t f(X(s))ds + \int_0^t g(X(s))dW(s), \quad 0 \leq t \leq T. \quad (2.5)$$

The first term in the SDE (2.4) is the drift term, while the second, random term is the diffusion term. If the function $g(X(t), t)$ for the diffusion term is taken as constantly equal to 0, the SDE reduces to a deterministic differential equation. Integrals over a

function with random variables, and integrals with respect to the random variable $dW(t)$, are defined using integration rules from *stochastic calculus*.

A stochastic differential equation may also define a vector-valued random process. If the process is n -variate, $f(X(t), t)$ and $g(X(t), t)$ in (2.4) are replaced by $n \times n$ matrices, where each entry in the matrices is a function of the process and t . The scalar Wiener process $W(t)$ is replaced by an n -variate Wiener process $\mathbf{W}(t)$, with independent entries.

2.4 The Ornstein-Uhlenbeck process

There are many different stochastic differential equations used for modelling time-dependent phenomena, particularly in physics [27] and in economics [23]. One SDE defines the Ornstein-Uhlenbeck (OU) process. This process is particularly suited for modelling attraction towards a mean value.

Definition 2.4.1. (Ornstein-Uhlenbeck process) An Ornstein-Uhlenbeck process is a stochastic process that solves the stochastic differential equation

$$dX(t) = b(\mu - X(t))dt + c dW(t). \quad (2.6)$$

The process was originally used to describe Brownian motion in physics, see [54] and [16]. The OU SDE (2.6) is obtained from the general form of stochastic differential equations (2.4) by setting $f(X(t), t) = b(\mu - X(t))$ and $g(X(t), t) = c$.

The parameter b is the drift term parameter, while c is the diffusion term parameter. The parameter c can be thought of as a short-term variance of the process, since it scales the normally distributed variable $dW(t)$. See Figure 2.1 for example trajectories of OU processes for different choices of parameters.

For the general properties of OU processes, refer to e.g. [27], [23] and [59]. The OU process is a Markov process, and it is strictly stationary. To show that the process is attracted to μ , we write the drift term as $-b(X(t) - \mu)dt$ (assuming b strictly positive for now). We observe that if the difference between the current point and the attraction point is positive, then the infinitesimal change in the random variable given by this term is negative, attracting the process towards the attraction point.

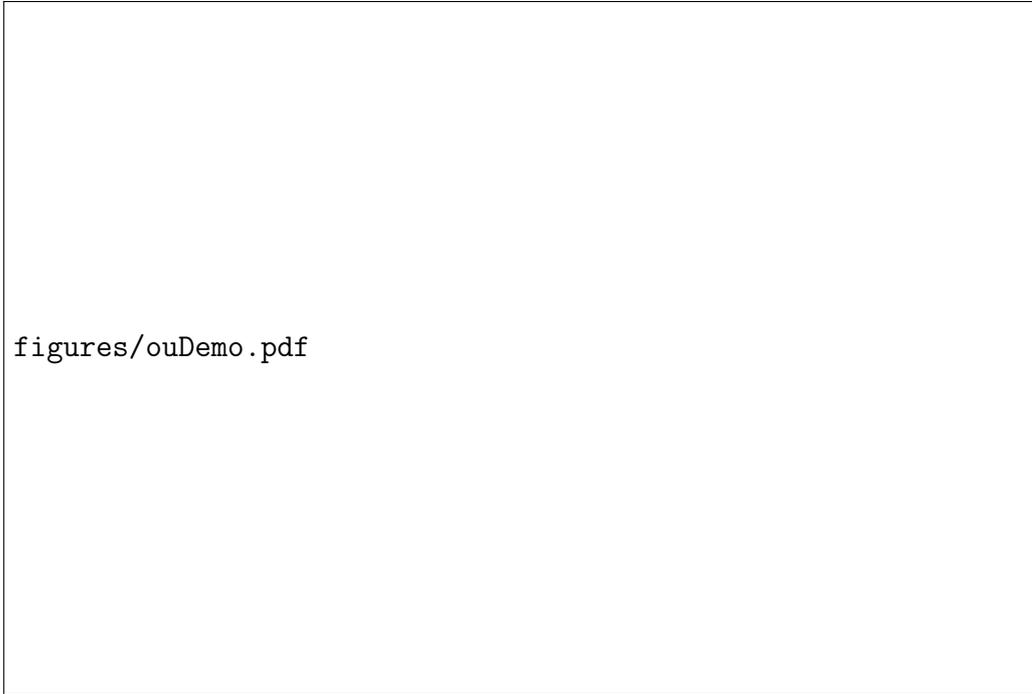


Figure 2.1: Example trajectories of OU processes

2.4.1 Generalization to n -variate OU processes

For the Ornstein-Uhlenbeck process, the generalization of the SDE to n -D is [6]

$$d\mathbf{X}(t) = B(\boldsymbol{\mu} - \mathbf{X}(t)) dt + \Sigma d\mathbf{W}(t) \quad (2.7)$$

where B and Σ are $n \times n$ matrices and $\mathbf{X}(t), \boldsymbol{\mu}, \mathbf{W}(t) \in \mathbb{R}^n$. The matrix B is the drift term parameter, Σ is the short-term covariance matrix and $\boldsymbol{\mu}$ is the attraction vector or mean of the process.

2.4.2 Analytic solutions

A solution (see for instance [42]) to the Ornstein-Uhlenbeck SDE, with initial condition $X_0 = x_0$ and $\mu = 0$, is

$$X(t) = x_0 e^{-bt} + \frac{ce^{-bt}}{\sqrt{2b}} W(e^{2bt} - 1) \quad (2.8)$$

where $W(e^{2bt} - 1)$ represents a Wiener process with variance¹ $e^{2bt} - 1$. To model attraction towards a non-zero μ , we replace $X(t)$ by $X(t) - \mu$ and x_0 by $x_0 - \mu$, and write

$$X(t) = \mu + (x_0 - \mu)e^{-bt} + \frac{ce^{-bt}}{\sqrt{2b}}W(e^{2bt} - 1). \quad (2.9)$$

Now we note that $X(0) = x_0$ since $e^{0b} = 1$ and $W(0) = 0$ by Definition 2.2.1. When t tends to infinity, the deterministic part of $X(t)$ (the two first terms) tends to μ .

2.4.3 Expectation and variance

From [16], with substitution for $\mu \neq 0$ (or by taking the expectation and variance of (2.9)), we get the following expressions for the Ornstein-Uhlenbeck process, conditional on an initial state x_0 :

$$\mathbb{E}[X(t)] = \mu + (x_0 - \mu)e^{-bt}, \quad (2.10a)$$

$$\text{Var}(X(t)) = \text{Var}\left(\frac{ce^{-bt}}{\sqrt{2b}}W(e^{2bt} - 1)\right) = \frac{c^2}{2b}(1 - e^{-2bt}). \quad (2.10b)$$

This can also be used for determining the distribution of the process at some time $s + t$, given a state x_s at a previous point s in time:

$$(X_{s+t}|X_s = x_s) \sim \mathcal{N}\left(\mu + (x_s - \mu)e^{-bt}, \frac{c^2}{2b}(1 - e^{-2bt})\right).$$

Since the OU process is Markovian and stationary, this statement is true for any value of s (changing s corresponds to time-shifting the process). For the long-term behaviour, with $b > 0$, we obtain

$$\lim_{t \rightarrow \infty} \mathbb{E}[X(t)] = \mu \text{ and} \quad (2.11a)$$

$$\lim_{t \rightarrow \infty} \text{Var}(X(t)) = \frac{c^2}{2b}. \quad (2.11b)$$

Thus, the long-term, or stationary, distribution of $X(t)$ is $\mathcal{N}(\mu, c^2/(2b))$.

For the properties of the n -variate OU SDE, we need the matrix exponential (see [29] and [19]). The exponential function e^x has Maclaurin expansion $\sum_{k=0}^{\infty} \frac{1}{k!}x^k$. This expansion can also be used for matrices:

¹In the same way that $W(t) = W(t) - W(0)$ has variance $t - 0 = t$ by Definition 2.2.1, $W(e^{2bt} - 1) = W(e^{2bt} - 1) - W(0) = W(e^{2bt} - 1) - W(e^{2b0} - 1)$ has variance $e^{2bt} - 1$.

Definition 2.4.2. (Matrix exponential) For any given square matrix A , the matrix exponential is defined as

$$e^A \equiv \exp(A) := \sum_{k=0}^{\infty} \frac{A^k}{k!}. \quad (2.12)$$

When the exponent tends towards minus infinity, the exponential tends to zero, just as the standard exponential function. Now we can define the properties of the n -variate OU process (see [6]):

$$\mathbb{E}[\mathbf{X}(t)] = \boldsymbol{\mu} + e^{-Bt}(\mathbf{x}_0 - \boldsymbol{\mu}) \rightarrow \boldsymbol{\mu} \text{ as } t \rightarrow \infty. \quad (2.13a)$$

$$\text{Var}(\mathbf{X}(t)) = \Lambda - e^{-Bt}\Lambda e^{-B^T t} \rightarrow \Lambda \text{ as } t \rightarrow \infty. \quad (2.13b)$$

Here, $\Lambda = \Sigma(B + B^T)^{-1}\Sigma^T$ is the long-term covariance matrix, which means that the long-term stationary distribution of $\mathbf{X}(t)$ is $\mathcal{N}(\boldsymbol{\mu}, \Lambda)$, i.e. n -variate normally distributed with mean vector $\boldsymbol{\mu}$ and covariance matrix Λ . If the matrices of parameters are replaced by scalar parameters, the expressions above reduce to the previously determined expressions for expectation and variance (2.10) for the univariate case.

2.4.4 Stability of the OU process

If we let the drift term parameter b in the univariate OU process SDE (2.6) be negative, the term e^{-bt} in the solution (2.9) tends to infinity as time increases, and not to μ . Thus, the process is *unstable* if $b < 0$. For $b = 0$, the process is unstable since the drift term in the SDE disappears and we are left with only the diffusion term. The solution of this new SDE is then just a Wiener process (Brownian motion) with increments having variance $c\sqrt{dt}$, which also can tend to infinity as time increases. Hence, for modelling attraction towards μ , only cases with $b > 0$, and thus $\lim_{t \rightarrow \infty} e^{-bt} = 0$, are of interest.

For the multivariate case of the OU process, note that $\lim_{t \rightarrow \infty} e^{-Bt} = 0$ where 0 is the zero matrix only if B has positive eigenvalues, and this is true for positive definite matrices [53]. Thus, the requirement for stability is that the drift term parameter matrix B is positive definite (see [42] and [6]).

2.4.5 Simulating the OU process

Sample values of a normal distribution are *realizations* of a normally distributed variable. For a stochastic process, the analogue is *simulating*, used to generate realizations of the sequence of stochastic variables the process defines. For an introduction to simulation of SDEs, see [18]. Simulation can be used to obtain knowledge of the approximate behaviour of a stochastic process. A process is simulated on an interval $[0, T]$ by dividing it into N small intervals of width Δt . The first point is set to X_0 , and then for each step Δt up until T we let the increment in the stochastic process be given by an approximate or exact updating formula derived from an expression for the stochastic process. For stochastic differential equations, a simple approximation is to use the infinitesimal quantity given by the right-hand side of the SDE.

Example 2.4.3. (First-order updating formula for the OU process) From [16], we get a first-order updating formula for the OU process. Modifying it here for the case when $\mu \neq 0$, using substitution, we get

$$X(t + \Delta t) = X(t) + b(\mu - X(t))\Delta t + c\sqrt{\Delta t}n \quad (2.14)$$

with $n \sim \mathcal{N}(0, 1)$. This follows directly from the OU SDE by replacing infinitesimal quantities with changes in value for small time steps (i.e. $X(t + \Delta t) - X(t)$ instead of $dX(t)$, and Δt instead of dt). This formula, however, will not be precise enough if Δt is too large [16]. Nevertheless, it was used in [31] for simulating fish migration, and the results were adequate.

Example 2.4.4. (Exact updating formula for the OU process) An exact updating formula from [16] for the OU process yields, with our notation and with substitution for $\mu \neq 0$,

$$X(t + \Delta t) = X(t)e^{-b\Delta t} + \mu(1 - e^{-b\Delta t}) + c\sqrt{\frac{1 - e^{-2b\Delta t}}{2b}}n. \quad (2.15)$$

with $n \sim \mathcal{N}(0, 1)$. This actually follows directly from (2.9) by using substitution for the case with $\mu \neq 0$, replacing the initial condition x_0 with $X(t)$, replacing the time t by the fixed time step Δt , and rewriting the increment of the Wiener process using its variance.

For a generalization of the exact updating formula for the OU process to 2-D, we note that for the case used in this thesis, the parameter matrices B and Σ will be 2×2 diagonal matrices. It is therefore sufficient to determine the update in each dimension separately as if they were univariate OU processes, using the (i, i) th component of the matrices to compute the update for the i th component of the process. See complete discussion in Chapter 4.

2.4.6 Estimating parameters of the OU process from data

If observation data is assumed to have been generated by an OU process, there exist several methods for estimating the parameters of the process. We present here the least squares and maximum likelihood methods.

We base the following on [55] and [48]. The least squares approach given here is only valid for time series equally spaced in time. Rewriting the updating formula (2.15) for the OU process in a discrete notation, we get

$$X_{t+1} = X_t e^{-b\Delta t} + \mu (1 - e^{-b\Delta t}) + c \sqrt{\frac{1 - e^{-2b\Delta t}}{2b}} n.$$

This is a linear relationship in the form

$$X_{t+1} = \alpha X_t + \beta + \epsilon \tag{2.16}$$

where

$$\alpha = e^{-b\Delta t}, \quad \beta = \mu (1 - e^{-b\Delta t}),$$

and ϵ is a normally distributed random term with

$$\mathbb{E}(\epsilon) = 0 \text{ and } \text{Var}(\epsilon) = \frac{c^2(1 - e^{-2b\Delta t})}{2b}.$$

We rewrite these equations and get

$$b = -\frac{\ln \alpha}{\Delta t}, \quad \mu = \frac{\beta}{1 - \alpha},$$

$$c^2 = \text{Var}(\epsilon) \frac{-2 \ln \alpha}{\Delta t(1 - \alpha^2)} = \text{Var}(\epsilon) \frac{2b}{1 - \alpha^2}.$$

So, if we assume that the linear relationship (2.16) is valid for a set of data and that the errors ϵ are uncorrelated in time, we can estimate the required parameters by making an ordinary least squares estimate for α, β using this data, and through this getting the variance of ϵ . The assumption that data can be described using this linear relationship is equivalent to assuming that it can be modelled by an autoregressive process of order 1 (see Section 2.6.2 below). The quantity Δt is just used for scaling the parameters. Setting $\Delta t = 1$ implies simulations with the same time step as data.

The method of maximum likelihood estimation uses the conditional probability density for an observation X_{t+1} given a previous observation X_t (see [55] for details). The log-likelihood function can be derived from this conditional density, and the values of μ, b and c that maximize this function can be found analytically if observations are evenly spaced in time. If observations are not evenly spaced, we would need to use iterative optimization methods to find the maximum, see [15].

2.5 Markov chains

When a Markov stochastic process has a countable state-space I , it is usually called a *Markov chain* (see e.g. [34]).

2.5.1 Discrete-time Markov chains

We first consider discrete-time Markov chains. Such a Markov chain can be described by a probability distribution λ for the initial state X_0 , and a set of probabilities for transitions between all pairs of states in a time step of one time unit:

$$\mathbb{P}(X_{k+1} = j | X_k = i) \text{ for all } (i, j) \text{ in the set } I \text{ of states of } X_t, \text{ and } k = 0, 1, \dots$$

If these probabilities are independent of the time k , the chain is time-homogeneous. For Markov chains with a finite state-space, these probabilities can be organized in a finite *transition matrix* P , where (when the states are named as the natural numbers) the elements of the matrix are given by

$$P_{ij} = \mathbb{P}(X_{k+1} = j | X_k = i).$$

See Section 2.5.3 for an example of a transition matrix. Transition matrices always have rows where the elements sum up to one, and matrices with this property are called *stochastic matrices*.

2.5.2 Important properties of Markov chains

We begin by listing some important properties and characterizations of Markov chains, taken mainly from [34]:

- The probabilities for transitions to one state from another over several time-steps are obtained by raising P to the power of the number of time steps, and use the resulting stochastic matrix in the same way.
- Given an initial distribution λ for X_0 , we obtain the unconditional distribution vector of X_n by taking λP^n .
- A state of a Markov chain is called *transient* if there is zero probability for the chain of returning to the state once it has been left once, and *recurrent* otherwise.
- A state of a Markov chain is called *absorbing* if it has zero probability of being left once it has been entered by the chain.
- A state of a Markov chain is called *aperiodic* when no states have a period. A period for a state is the smallest fixed number of steps greater than 1 that it takes to get back to the same state. For aperiodic states, on the other hand, the return probability over n steps is strictly positive, for sufficiently large n . A Markov chain is called aperiodic when all states are aperiodic.
- A Markov chain is *irreducible* if all states can be reached from each other.
- A Markov chain is called *ergodic* if it is both aperiodic and irreducible [40].
- The equilibrium distribution π of a Markov chain with a finite state-space is the distribution of the states as time tends to infinity, and corresponds to the

proportion of time spent in each state when the chain is allowed to evolve infinitely.

A distribution for the states of a Markov chain that satisfies $\pi P = \pi$ (i.e. π is a left eigenvector of P associated with the eigenvalue 1) is called an invariant distribution for the chain. This means that the unconditional distribution of X_n , when the initial distribution of the chain is π , will simply be π itself, since

$$\pi P^n = \pi$$

for any value of n . From Theorem 1.8.3 in [34] we get that if an ergodic chain X_t has an invariant distribution π , the equilibrium distribution is the same as the invariant distribution, i.e.

$$\mathbb{P}(X_n = j) \rightarrow \pi_j \text{ as } n \rightarrow \infty \text{ for all } j \in I.$$

A consequence of this is that the n -step transition matrix P^n converges to a matrix with μ in each row as n tends to infinity.

2.5.3 Simulating discrete-time Markov chains

To simulate a discrete-time Markov chain with a finite number of states we use a random number generator and map these random numbers into a new state. See [34] for a complete description of this procedure. We illustrate by an example. Given a transition matrix

$$P = \begin{pmatrix} 0.10 & 0.80 & 0.10 \\ 0.10 & 0.60 & 0.30 \\ 0.05 & 0.60 & 0.35 \end{pmatrix}$$

for a Markov chain with the state-space $I = \{1, 2, 3\}$. Now, if the process is in state 1, the probabilities for transition to states 1, 2 and 3 are 0.1, 0.8 and 0.1, respectively. Given a random number u from a uniform distribution on $(0,1)$, we need a setup that gives us a 10% chance of staying in state 1, an 80% chance of transition to state 2 and a 10% chance of transition to state 3. By using the cumulative sums of the probabilities except the last, we can find intervals for the random variable that will trigger a certain transition. The random variable should then be mapped to transitions in the following way:

- $u \in (0, 0.1) \Rightarrow$ transition to state 1
- $u \in (0.1, 0.9) \Rightarrow$ transition to state 2
- $u \in (0.9, 1) \Rightarrow$ transition to state 3

2.5.4 Estimation of transition matrices

A transition matrix can be estimated from a time series on a discrete set of states, under the assumption that the time series is generated by a Markov chain. For a finite-state discrete-time Markov chain S_t with the natural numbers as names for the states, [34] gives the following consistent estimator for the elements of the transition matrix P :

$$\hat{p}_{ij} = \left(\frac{\sum_{n=0}^{N-1} \mathbf{1}_{\{S_n=i, S_{n+1}=j\}}}{\sum_{n=0}^{N-1} \mathbf{1}_{\{S_n=i\}}} \right). \quad (2.17)$$

This is the proportion of all transitions from state i that go to state j , using $\mathbf{1}_{\{\cdot\}}$ to denote the indicator function which is equal to 1 if the subscripted expression in braces is true, and 0 if it is false. The estimator \hat{p}_{ij} will tend to p_{ij} with probability 1 as N tends to infinity, by the strong law of large numbers.

2.6 Time series analysis

We base most of the basic time series analysis theory on [46] and [8]. An observed time series can be written as $\{x_t, t = 0, 1, \dots\}$, whereas we will use X_t for denoting the stochastic variables in time series models. In standard statistics, observations are often assumed to be independent and identically distributed (i.i.d.). In time series analysis, however, this is usually not the case, as there will often be some closeness in value for observations that are made close to each other in time. Thus, the usual statistical properties of mean and variance are replaced by the mean function and the autocovariance functions, that depend on time. The definition of autocovariance is the same as for stochastic processes in Section 2.1.2.

Definition 2.6.1. The mean function for a time series X_t is defined as

$$\mu_t \equiv \mathbb{E}[X_t] = \int_{-\infty}^{\infty} x f_t(x) dx, \quad (2.18)$$

if the integral exists, and where $f_t(x)$ is a time-varying probability density function for the values in the time series.

It is a common approach to attempt to transform nonstationary time series into stationary ones, using methods such as *detrending* and *differencing* (see the next section). Thus, much of the theory for stationary time series can also be applied to nonstationary time series. For stationary time series, we write $\mathbb{E}(X_t) = \mu$ for the mean function and $\gamma(h)$ with $h = |s - t|$ for the autocovariance function, as these functions (in the stationary case) are independent of time. For the rest of this section, the theory we present on time series analysis concerns stationary time series. We therefore use the *lag* h instead of the general time points s and t . We now give the *sample* autocovariance and autocorrelation functions, that are estimators for their theoretical counterparts in 2.1.2.

Definition 2.6.2. The sample autocovariance function (ACF) is defined as [46]

$$\hat{\gamma}(h) = \frac{1}{n} \sum_{t=1}^{n-h} (x_{t+h} - \bar{x})(x_t - \bar{x}) \quad (2.19)$$

and the sample autocorrelation follows as

$$\hat{\rho}(h) = \frac{\hat{\gamma}(h)}{\hat{\gamma}(0)}. \quad (2.20)$$

The ACF of a white-noise process W_t of length n is, under some conditions [46], approximately normally distributed with mean 0 and standard deviation given by $1/\sqrt{n}$. From this, 95% confidence bounds for an ACF can be given. If a value of an ACF for a process x_t for some lag h is outside the interval \pm/\sqrt{n} , it can be considered as significantly different from that of a white-noise process. Then, if many of the lags of are significant, the process is autocorrelated.

2.6.1 Smoothing

Smoothing is a technique used to discover trends in a time series. For instance, it can be used to remove high-frequency statistical noise from a series with an underlying, slower variation or pattern (see e.g. [11]).

The terms filtering and smoothing are sometimes used interchangeably. An important distinction is that filters always keeps the time step in the original data, while smoothing may involve producing smoothed data points at intermediate points in time. In the following, all smoothing will be done using filters. An example of a filter is the moving average filter:

Definition 2.6.3. (Moving average filter) A $(2n + 1)$ point symmetric moving average of a time series x_t is the series

$$m_t = \frac{1}{n} (x_{t-n} + \cdots + x_{t-1} + x_t + x_{t+1} + \cdots + x_{t+n}).$$

The number $(2n + 1)$ is called the *window* of the filter, as it gives the amount of data points around the current that are included in the average. The term window is also used for the set of nearby points used in the filter. Only windows symmetric about the current point are considered here. Near the start and end of a time series, less points are available to compute the average, so the smoothing is less heavy in these areas.

In the moving average definition, the weights for each observation are the same. Other filters may be obtained by using different weights. Another example is an exponentially weighted moving average, where observations further away from the time-point considered are given less weight, according to an exponential function decreasing with the distance in time.

Savitzky-Golay filters

When smoothing a time series, it may be useful to retain local peaks in the series. A filter which has a tendency to do so, provided that a suitable window is chosen, is the Savitzky-Golay filter [45, 49]:

Definition 2.6.4. (Savitzky-Golay filter) A time series x_t filtered by an order p Savitzky-Golay filter with window $k = 2n + 1$ (with $k \geq p + 1$) is the series

$$m_t = \sum_{i=-n}^n \alpha_i x_{t+i} \tag{2.21}$$

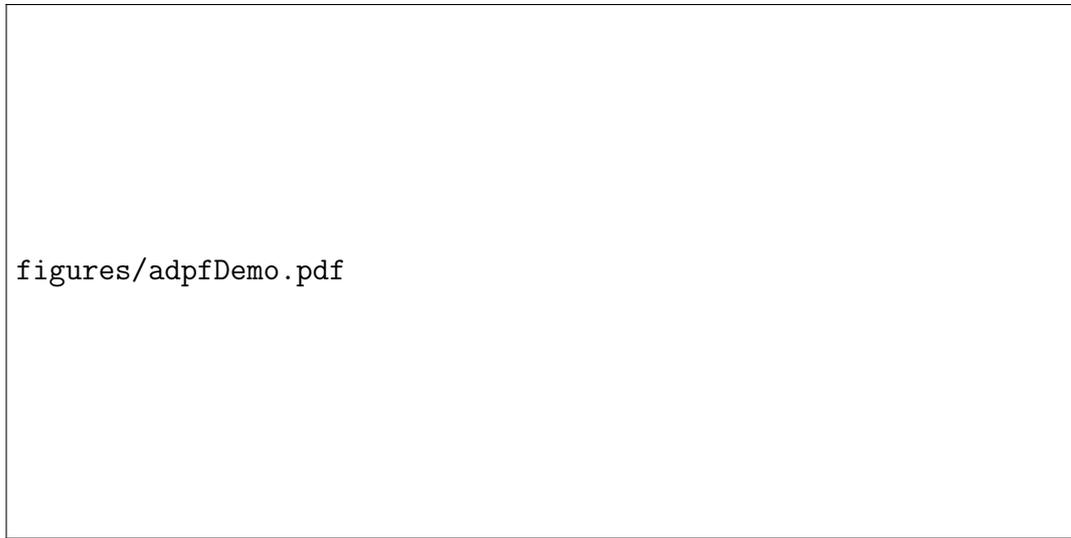


Figure 2.2: Example of ADPF smoothed time series

where the weights α_i are obtained by least-squares fitting a degree p polynomial to the data points x_{t-n}, \dots, x_{t+n} .

See for instance [39] for a discussion on properties of the filter. Also, when the data points are evenly spaced in time, it is not necessary to do a complete polynomial regression for each data point; the set of weights for neighbouring points is the same for every point, the weights depend only on the window and the degree of the filter.

Adaptive-degree polynomial filters

For the Savitzky-Golay filters to optimally smooth high-frequency noise, both the window and the degree of the filter must be chosen correctly. For maximum smoothing, low-order polynomials are desirable, whereas high-order polynomials are needed to maintain the local peaks [3]. Thus, the fixed-degree Savitzky-Golay filters may fail to smooth all parts of a time series optimally. We consider instead adaptive-degree polynomial filters (ADPF, see [3]). For these filters, only the window needs to be chosen in advance, while the degree is chosen automatically for each data point. A least-squares fit of a polynomial is done for the points in each data window, and statistical testing is used to determine the minimum degree needed for the polynomial

to provide optimal smoothing. See Figure 2.2 for an example of ADPF smoothing of a time series.

2.6.2 Autoregressive models

Autoregressive models are a common way to model discretely observed data with observations evenly spaced in time. When the current value of a time series X_t is explained as a function of p past values $X_{t-1}, X_{t-2}, \dots, X_{t-p}$ of the series, it is called an autoregressive model. It implies that p past values are considered necessary for predicting the current value. Formally, such a model is defined as follows:

Definition 2.6.5. An autoregressive model of order p (denoted $\text{AR}(p)$) is a model of the form

$$X_t = \phi_1 X_{t-1} + \phi_2 X_{t-2} + \dots + \phi_p X_{t-p} + w_t \quad (2.22)$$

with X_t stationary, $\phi_1, \phi_2, \dots, \phi_p$ constants (and $\phi_p \neq 0$) and w_t a noise process.

The noise process w_t in the definition is typically chosen to be a white noise process with some given variance. The mean of X_t above is zero, meaning that the process is attracted towards zero (it can be shown that the $\text{AR}(1)$ process is a discrete-time analogue of the Ornstein-Uhlenbeck process). If an autoregressive model with non-zero mean μ is desired, substitute X_t by $X_t - \mu$ for all t to obtain

$$X_t - \mu = \phi_1 (X_{t-1} - \mu) + \phi_2 (X_{t-2} - \mu) + \dots + \phi_p (X_{t-p} - \mu) + w_t$$

or, equivalently, for simpler notation,

$$X_t = \alpha + \phi_1 X_{t-1} + \phi_2 X_{t-2} + \dots + \phi_p X_{t-p} + w_t$$

with $\alpha = \mu(1 - \phi_1 - \dots - \phi_p)$.

In order to identify which model is appropriate for modelling a time series, properties of the autocorrelation function defined above are central. For instance, for an autoregressive process, the autocorrelation function shows a slow decay from 1 towards 0 for increasing lag. The partial autocorrelation function can be used to identify the order of an autoregressive model suitable for modelling a time series x_t .

Definition 2.6.6. The partial autocorrelation function (PACF) ϕ_{hh} of a stationary process x_t , $t \geq 0$ for lags $h = 1, 2, \dots$, is [46]

$$\phi_{11} = \text{corr}(x_1, x_0) = \rho(1)$$

and

$$\phi_{hh} = \text{corr}(x_h - x_h^{h-1}, x_0 - x_0^{h-1}), \quad h \geq 2 \quad (2.23)$$

where x_h^{h-1} denotes the regression of x_h on $\{x_{h-1}, x_{h-2}, \dots, x_1\}$.

The PACF is, in words, the correlation between x_t and x_{t-h} with the dependence on the terms between them in time removed. A PACF for an AR(p) process shows a sharp cutoff after the p 'th lag.

2.6.3 The Persist algorithm

When summarizing data using histograms, the data is divided into different bins for determining the height in a bar diagram. The Persist algorithm [30] provides a method for selecting such bins for equally spaced time series data, taking the time ordering of the values into consideration. This can also be used for discretizing the continuous values in a time series into a finite set of symbols. The algorithm works by computing a *persistence score* for candidate bins, and modifying these bins to maximize the sum of persistence scores over all bins. The persistence score is high for a bin if the resulting symbol is persisting, i.e. that it occurs many times in a row in the series. The implementation of the algorithm allows the user to select the number of desired bins, or let the algorithm determine the optimal bin count between 2 and 7 bins. Persisting states that account for less than 5 % of the whole time series are not given their own bin.

2.6.4 Kernel density estimation

Kernel density estimation (KDE) is a non-parametric method used for estimating the probability density function that has generated a random sample d_1, \dots, d_n (see for instance [28] and references). Note that kernel density estimation is applicable

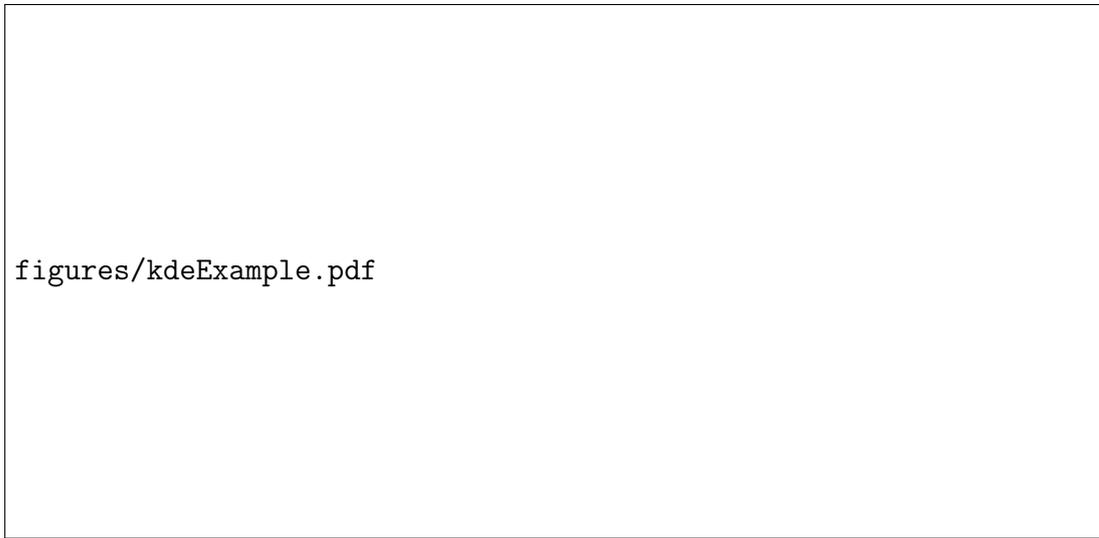


Figure 2.3: Examples of kernel density estimates for 1-D (with varying bandwidth) and for 2-D data (blue is low density, red is high).

to any random sample, not just for time series. A kernel is a symmetric function $K(\cdot)$ (typically the standard normal density function) that must integrate to one. By placing a weighted kernel function at each data point and summing them up, we get a new function that integrates to one. This estimate has peaks at regions where there is a high density of data points. Formally, the estimated function \hat{f}_h of the data d is

$$\hat{f}_h(d) = \frac{1}{nh} \sum_{i=1}^n K\left(\frac{d - d_i}{h}\right), \quad (2.24)$$

with d_i the data points and h a smoothing parameter called the bandwidth or window. See Figure 2.3 on the left for an example of how varying the bandwidth affects the KDE.

The kernel density estimate has a two-dimensional counterpart, with the estimate being a function

$$\hat{f}_h : \mathbb{R}^2 \rightarrow [0, 1],$$

which is a probability density surface $\hat{f}_h(x, y)$, with the height in the surface corresponding to high concentration of data points near by. The standard normal density function of the univariate KDE is typically replaced by the bivariate standard nor-

mal distribution. The bandwidth setting is generalized in such a way that separate bandwidths for each data dimension are allowed. The 2-dimensional KDE is usually represented using level curves or colours to display the function. See Figure 2.3 on the right for an example 2-D KDE of a bivariate time series (x_t, y_t) , where the time series values are plotted in the plane.

2.7 Interpolation

Given a set of data $(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)$, assumed to be evaluations of an unknown, continuous function $y = f(x)$, one might need to obtain function values $y_i = f(x_i)$ for some points x_i lying between the given points. Interpolation is a technique to achieve this. When interpolating, a function is found that passes through all the data points, called *nodes*, and this function is then evaluated at the desired point.

Example 2.7.1. (Nearest neighbour interpolation) Nearest neighbour interpolation is in practice fitting piecewise constant functions to the data points, in such a way that the closest value is given.

Example 2.7.2. (Linear interpolation) Connecting all neighbouring data points with lines, i.e. using a piecewise linear function to interpolate, provides linear interpolation.

2.7.1 Thin plate spline interpolation

Spline interpolation is useful for finding a smooth interpolating function [21]. A spline is made up of piecewise polynomials, and cubic splines are often used in practice. Natural cubic splines produce the smoothest spline possible. For functions of two variables $z = f(x, y)$, i.e. with a set of data

$$(x_0, y_0, z_0), (x_1, y_1, z_1), \dots, (x_n, y_n, z_n)$$

assumed to be evaluations of f , a parallel to spline interpolation is thin-plate spline interpolation [7]. The name refers to a physical analogue of bending a thin metal

plate to fit to the function value at the interpolation nodes. The spline is constructed by minimizing the “bending energy”, and gives very smooth interpolating functions.

2.8 Constrained optimization

A general constrained, continuous optimization problem on n real variables can be written as

$$\underset{\mathbf{x}}{\text{minimize}} f(\mathbf{x}) \text{ subject to } G(\mathbf{x}) \leq 0 \quad (2.25)$$

where $\mathbf{x} \in \mathbb{R}^n$ is the decision variable, $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is a smooth objective function and $G(\mathbf{x}) = (g_1(\mathbf{x}), \dots, g_m(\mathbf{x}))^T$ is a vector of m smooth functions for inequality constraints, with $g_k : \mathbb{R}^n \rightarrow \mathbb{R}$ for $k = 1, \dots, m$ (see [33]). Such an optimization problem may also have equality constraints, but such constraints can always be written as a pair of inequality constraints of opposite direction.

The constraints define together the *feasible set* Ω of the optimization problem, the space of values of \mathbf{x} that satisfy all constraints.

2.8.1 Global and local solutions and convexity

An optimization problem may have both local and global solutions. A point \mathbf{x}^* is a local solution if there is exists an open set containing \mathbf{x}^* where $f(\mathbf{x}^*) \leq f(\mathbf{x})$ [33]. If a point \mathbf{x}^* satisfies $f(\mathbf{x}^*) \leq f(\mathbf{x})$ for all $\mathbf{x} \in \Omega$, then it is a global solution. For optimization problems where the objective function is a convex function and the feasible set is a convex set, any local solution is also a global solution.

2.8.2 Optimization algorithms

Numerical algorithms for solving continuous optimization problems typically work iteratively from a starting point \mathbf{x} to improve the objective function value. Many algorithms use information on the derivative of both the objective function and constraint functions (constructed by finite difference if not given) to promote fast convergence. For non-convex and non-smooth optimization problems, there is a risk that algorithms “get stuck” at local solutions rather than finding global solutions.

2.9 Biological background information

Knowledge on some aspects about cod biology is necessary for the discussions in this thesis. General information on this can be found in e.g. [50] and its references, and in [44] and [36]. The species considered in this thesis is Atlantic cod (*Gadus morhua*), and in the Lofoten-Barents sea ecosystem, specimens can either belong to the stock of North-East Arctic cod (“*skrei*”) or to one of the populations of Norwegian coastal cod. Cod is a predator, feeding on zooplankton, benthic organisms² and other fish. Cod is mainly a demersal fish, meaning that it lives mainly on or near the bottom of the sea.

2.9.1 North-East Arctic cod

North-East Arctic cod is the largest cod stock in the world. During some parts of the year, it is not only demersal, but also distributed in the open sea. It mainly lives on the warm side of the Polar Front in the Barents sea, while it spawns in the open sea in the Lofoten/Vesterålen area in Norway.

2.9.2 Norwegian coastal cod

The Norwegian coastal cod is a variant of Atlantic cod which lives in coastal areas and fjords. There are many distinct stocks along the coast of Norway, from Stad to Russia. 75 % of the cod lives north of the 67th latitude. Only older cod (> 2 years) moves to deeper water – in general the cod stays in shallow water. Its spawning grounds are fjords.

2.9.3 Vertical migration

Cod can migrate horizontally, for instance for the North-East Arctic cod to reach its spawning grounds. In addition, cod migrates vertically [17]. Vertical position may, for North-East arctic cod, vary with time of day and time of year, and it is affected by for instance light, current, bottom depth and prey abundance in space and time.

²organisms living on the seabed

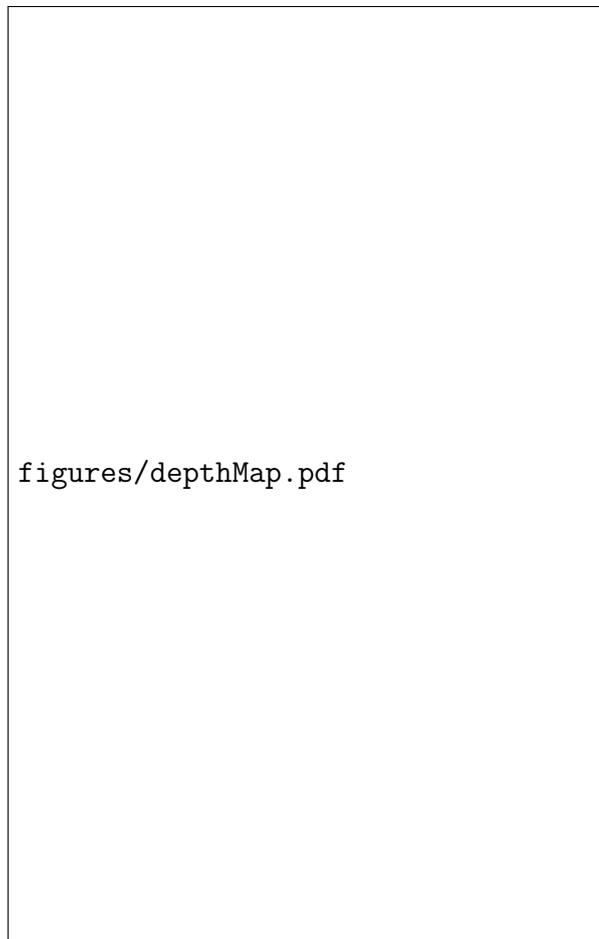


Figure 2.4: Depth map of Norwegian and Barents seas, and of the area around the Lofoten Archipelago. Source: The ROMS atlas, see Section 6.3.

2.10 Geography

To get an idea of the depths of the waters in which the fish analyzed in this thesis belongs, see Figure 2.4 for a colour map showing the depth of the seabed off the coast of Norway. The bottom figure shows a closeup near the spawning grounds of the North-East Arctic cod in the Lofoten/Vesterålen area, showing how narrow the band of shallow waters along the coast is.

Chapter 3

Previous work

This chapter gives an overview over some previous work done on stochastic modelling of data from tracking of the horizontal movement of individuals in various ways, and on analyzing the migratory behaviour of cod.

3.1 Stochastic modelling of wildlife movements

Various technologies can be used to track the behaviour of wildlife. Radio transmitters attached to animals can for instance be used to obtain time series of approximate positions in the terrain [38]. Acoustic tags, attached to fish, can be used to determine the position of fish, but only for smaller areas [58]. Marine animals spending much time at the surface can be tracked by satellite, but high-resolution tracking is currently not possible for fish that stays most of the time in deep waters. Instead, data storage tags (DSTs) are used. These can give time series on the ambient environment such as depth (through pressure measurement) and temperature [17]. Also, data storage tags do not transmit information, but rely on the fish being recaptured, typically through the commercial fisheries. See [32] for a recent overview on electronic tagging and tracking of marine animals.

A method for extracting knowledge from movement data is stochastic modelling. In [38], stochastic differential equations of different forms are used to model the

movement in the terrain of 216 elk. The general form of the equation used is

$$d\mathbf{r}(t) = \begin{pmatrix} \mu_x(\mathbf{r}(t), t) \\ \mu_y(\mathbf{r}(t), t) \end{pmatrix} dt + \mathbf{D}(\mathbf{r}(t), t) \begin{pmatrix} d\Psi_x(t) \\ d\Psi_y(t) \end{pmatrix} \quad (3.1)$$

with $\mathbf{r}(t) = (X(t), Y(t))$ being the position vector, $\boldsymbol{\mu}$ the drift parameter, \mathbf{D} the diffusion matrix and Ψ_x, Ψ_y random processes with expectation 0.

Different forms of the parameters in Equation (3.1) were suggested, and they were estimated using approximating difference equations and nonparametric regression techniques.

3.2 Mixed Ornstein-Uhlenbeck process models

Mixed Ornstein-Uhlenbeck process models are fundamental for the modelling of fish migration in this thesis. We now introduce them through their origins, and present some examples on how they have been used to model animal movement patterns.

3.2.1 OU process models in home range analysis

The first application of OU process models (to the knowledge of the author), in modelling wildlife movement behaviour, is by Dunn and Gipson [12]. In their work, a multivariate OU process model is used to analyze radio telemetry data for studying the *home range*¹ of mammals. The process is defined such that the positions of many animals are modelled simultaneously, using a $(2n)$ variate process to model the positions of n animals. One of the forms of the SDE (3.1) considered in [38] was actually the bivariate Ornstein-Uhlenbeck process.

3.2.2 Mixed diffusion models

Blackwell [6] maintains that the single diffusion models in [12] are unrealistic for modelling home range behaviour, and describes a generalization of the OU model in

¹In [10], home range is defined as “[...] the area, usually around a home site, over which the animal normally travels in search of food”.

which the individual animals make switches between different diffusion processes in a random manner. The approach is based on [4], where a bivariate Markov process

$$\{X(t), S(t)\} \quad (3.2)$$

is introduced with $X(t)$ a continuous process representing position and a Markov chain $S(t)$ on a discrete state-space. The value of the continuous variable $X(t)$ is governed by a Markov diffusion process, with parameters that depend on the discrete variable $S(t)$ and if desired on the position. $S(t)$ is governed by a continuous-time Markov chain, with transition probabilities that are fixed or that depend on the position given by $X(t)$. The result shown is that the overall process of these two variables is a Markov chain on the discrete space of states for $S(t)$ [4]. The result generalizes to higher-dimensional diffusion processes as well [6].

The model is in [6] extended to a bivariate diffusion process with a third variable being a Markov chain, in order to model the position of wood mice. The mouse model is then a bivariate OU diffusion process with a third variable determining the behavioural state π of the animal, that is

$$\{X(t), Y(t), S(t)\}. \quad (3.3)$$

$S(t)$ is defined as a Markov chain with a state-space Π consisting of states $\pi_1, \pi_2, \dots, \pi_{|\Pi|}$. The process $\mathbf{X}(t) = [X(t), Y(t)]^T$ is a bivariate Ornstein-Uhlenbeck process, i.e. it is governed by the stochastic differential equation

$$d\mathbf{X}(t) = B^{(k)} (\boldsymbol{\mu}^{(k)} - \mathbf{X}(t)) dt + \Sigma^{(k)} d\mathbf{W}(t). \quad (3.4)$$

Here the parameters $\boldsymbol{\mu}^{(k)}$, $B^{(k)}$ and $\Sigma^{(k)}$ depend on the value of $S(t)$ in the sense that when $S(t) = \pi_k$, the parameter set

$$\{\boldsymbol{\mu}^{(k)}, B^{(k)}, \Lambda^{(k)}\} \quad (3.5)$$

for state π_k of the chain $S(t)$ is used in the SDE. The diffusion term parameter $\Sigma^{(k)}$ is defined implicitly through the relation $\Lambda = \Sigma(B + B^T)^{-1}\Sigma^T$.

The positions of the animals thus follow diffusion processes with parameters given by the Markov chain. When the Markov chain makes a transition, the new

parameters are used instead of the old, with the start point of the new diffusion being the end point of the previous one.

For modelling the home range of wood mice, Blackwell in [6] defines three different states for the behaviour of a mouse: resting, feeding and travelling. A common centre $\boldsymbol{\mu}$ (the nest of the mouse) is used for all states, while the parameter matrices $\Lambda^{(k)}$ and $B^{(k)}$ are estimated separately for each state using Bayesian inference and Markov Chain Monte Carlo methods (see [5] for details).

3.2.3 Modelling mobile phone movements

Rosenblum in [42] used bivariate mixed Ornstein-Uhlenbeck models to model the movement of mobile phone users in e.g. a city. The choice of model is based on the need to model a tendency of users to move towards and cluster at certain points, are referenced to as concentration points. The approach is similar to that of Blackwell, but uses other methods to determine attraction points and for inference.

The terrain is partitioned into a grid, and a concentration point is defined at the centre of each grid cell with a sufficiently high occupation frequency. For each concentration point, denoted μ_i , a *neighbourhood* is defined as a circle around the point with radius half the distance to the closest other concentration point. See Figure 3.1 for a demonstration, where an area of 60 times 60 units is divided into a grid. Squares with high mobile phone occupation are shaded, and concentration points are placed at the centre of each of these. The neighbourhood associated with each concentration point is shown as a green circle. States π_k are defined for each ordered pair of neighbouring concentration points.

Movement data of mobile phones within and between these neighbourhoods is used to construct a generator matrix A for a continuous-time Markov chain

$$\{J(t), t \geq 0\},$$

having a state $\pi_k = \{i \rightarrow j\}$ for each pair of concentration points. Also, the data is used to estimate parameters

$$\{\mu_i, B_i, \Sigma_i\}$$

for the OU process in each state.



Figure 3.1: Mobile phone modelling: dividing an area into neighbourhoods

We now present the approach to parameter estimation for each state given in [42]. State indices are skipped for easier notation. The goal is that the matrix Λ , the covariance matrix of the stationary or long-term distribution, should match the matrix with the variances of the data within the neighbourhood of the destination concentration point on the diagonal. I.e.

$$\Lambda = \Sigma(B + B^T)^{-1}\Sigma^T = \begin{bmatrix} \sigma_D^2 & 0 \\ 0 & \sigma_T^2 \end{bmatrix}. \quad (3.6)$$

The average time h spent travelling from between the origin and destination concentration points while in a state is introduced, without a precise definition. This is used to set

$$B = \frac{\lambda_B}{h}I$$

and

$$\Sigma = \frac{\lambda_\Sigma}{\sqrt{h}} \begin{bmatrix} \sigma_D & 0 \\ 0 & \sigma_T \end{bmatrix}$$

with λ_B and λ_Σ parameters to be determined. Since both B and Σ are diagonal, and thus symmetric, we can write

$$\begin{aligned} \Lambda &= \Sigma(B + B^T)^{-1}\Sigma^T = \Sigma(2B)^{-1}\Sigma \\ &= \frac{\lambda_\Sigma^2}{2\lambda_B} \begin{bmatrix} \sigma_D & 0 \\ 0 & \sigma_T \end{bmatrix} I \begin{bmatrix} \sigma_D & 0 \\ 0 & \sigma_T \end{bmatrix} = \frac{\lambda_\Sigma^2}{2\lambda_B} \begin{bmatrix} \sigma_D^2 & 0 \\ 0 & \sigma_T^2 \end{bmatrix} \end{aligned}$$

which satisfies (3.6) if we set $\frac{\lambda_\Sigma^2}{2\lambda_B} = 1$. In [42], λ_Σ and λ_B are chosen arbitrarily, but it is stated that "In a more complex model, they could be based on the empirical variances, since these constants determine on average how close a mobile will approach its destination concentration point".

3.3 Analyzing migratory behaviour of cod

This sections presents a few examples of analyses made on the migratory behaviour of cod using data storage tags. For general information on migratory behaviour, see cited litterature in references highlighted here.

3.3.1 The 1996 cod tagging experiment

An experiment with tagging of cod using data storage tags in 1996 has been described by Godø and Michalsen in [17]. The practical tagging was described in detail, and some descriptive data analyses were made.

The data capture interval was 2 hours during the first 6 days, and 12 hours on the 7th day, with this pattern repeating until the storage capacity of 3900 recordings was exhausted. For the vertical movements, short-term movements of less than 10 meters was typical, while movements of up to 250 meters did occur sporadically. For some periods, the fish moved with a diel rythm (24 hour rythm). Analysis on the buoyancy, influenced by the fish swim bladder, was made. Seasonal differences in depth and temperature showed that the cod experience higher temperatures in

winter and spring than in summer and autumn. An acclimatization period of up to 14 days after release was discovered, with behaviour unusual compared to later recordings. Subsequently, the data from the first 14 days was excluded from the analysis.

The effects of the tagging on the swimming speed of the fish was determined to be neglectable. Tests by the manufacturer determined precision of temperature recordings of ± 0.2 degrees C and ± 1 atmospheres (approximately 10 meters) for depth.

3.3.2 DST analysis with the continuous wavelet transform

In [51], Subbey and Michalsen suggest using the continuous wavelet transform for extracting periodical features of DST data and displaying them in an intuitive way. This has revealed a clear diel and tidal rhythm in DST data derived from the same set of tagging experiments as the data used in this thesis. The paper also suggests that the wavelet transform can be used to analyze corresponding temperature and depth time series simultaneously, and for instance determining a cause-effect relation between the two series.

3.3.3 Tidal migration of cod

It is known that cod can use tidal currents for their horizontal migration. The extent of this is unknown, so the phenomenon is investigated in [50] using depth data from data storage tags. Since the geographical location (geolocation) of the fish is unknown, known tidal information for different positions in the ocean cannot be used directly. Instead, the depth time series is fitted to a tidal model to create a tide time series. The resulting two time series are analyzed together using the continuous wavelet transform, to identify times when the fish can be assumed to be using the tides for transport.

3.3.4 Hidden Markov Models for geolocalization

An example of the use of data from data storage tags for determining the geolocation of fish is the modelling using Hidden Markov models in the thesis by Pedersen [37]. The observed DST data is considered as the observed state of the fish, while the actual geolocation is the hidden state. Since DSTs determine depth using pressure, the depth reading at the seabed will change when the ocean surface falls or rises due to tides. If a tagged fish stays at the seabed for a long time (which happens if the fish is of a demersal species), this tidal rhythm can be retrieved. A random walk model for the fish movement is combined with a model on the amplitude and frequency of the tides at different positions in the sea to estimate a "most probable track" for the fish through the ocean.

The thesis also shows examples of the use of a temperature atlas as an aid in geolocalization (the tidal information remaining the main geolocating variable), with the simplification that only temperature information for the seabed is used.

3.3.5 Geolocalization by biased random walks

A method for geolocalization using only depth and temperature information is presented in [58]. A database of depth and temperature for the area in question is constructed. A huge number of synthetic fish are then moved through a virtual ocean using a biased random walk procedure, starting at the release position of the true tagged fish. The bias lies in that the fish is slightly attracted to the recapture position. Fish that observe temperature conditions that are not in agreement with the data from the DST data are terminated. Those that "survive" all the way are used to make a mean trajectory. Examples are shown with geolocalization of tagged North-East Arctic cod in the Barents sea.

Part I

Stochastic modelling of fish migration

Chapter 4

Basic data analysis and modelling approach

This chapter introduces the data used and the basic approach used for modelling. The data is derived from electronic tagging of an Atlantic cod (*Gadus Morhua*) in the Lofoten-Barents Sea ecosystem. The data has been provided by the Institute of Marine Research, Norway, and is referenced in [51] and [50].

4.1 Data overview

The data is given as a time series of depth and temperature sampled at 10 minute intervals, from the time of release until recapture. We denote the depth/temperature time series tuple as (\mathbf{D}, \mathbf{T}) , with $\mathbf{D} = (D_1, D_2, \dots, D_N)$ for a length N time series (and similarly for temperature). Sometimes, when developing methods applicable to both depth and temperature time series, $\mathbf{x} = (x_1, x_2, \dots, x_N)$ will be used instead.

The modelling work has been done using a time series from just a single tag, labeled “Tag 1664”, which is from a fish that was tagged and released in April 2004, and recaptured in December 2006. Data is available from the release day and until May 2006. This is a rare example of a long time series for one tagged fish. For general properties of the time series, see Table 4.1. See Figures 4.1 and 4.2 for plots of the data. In Table 4.2, some basic statistical characteristics are shown. Box plots for the

Table 4.1: Properties of tag 1664

Property	Value
Release date	02.04.2004
Release position	67.4°N, 12.6°E
Date/time first observation	03.04.2004 04:40
Recapture date	22.12.2006
Recapture position	68.4°N, 16.1°E
Date/time last observation	19.05.2006 20:40
Sex	Male
Length on release date	89 cm
Genetic ID	Norwegian Coastal Cod

distribution of the data for each month are shown for both time series in Figure 4.3.

The precision in the data is unknown, but as mentioned in the chapter on previous work, the precision in a previous tagging experiment was of $\pm 0.2^\circ$ C for temperature and ± 1 atmospheres (approximately 10 meters) for depth. For the lack of more recent information, this will be used here as well.

Table 4.2: Selected statistical characteristics of tag 1664

	Whole period		First year		First month	
	Temp	Depth	Temp	Depth	Temp	Depth
Mean	7.66	99.86	7.48	114.53	6.22	84.55
Standard deviation	1.15	63.9	0.9	64.25	0.48	29.05
Median	7.54	89.7	7.47	113.6	6.1	82.4
Min	4.63	-1.99	4.63	-1.99	4.63	0.93
Max	11.71	329.25	11.21	329.25	7.37	156.61
Max diff	2.14	140.58	2.09	140.58	0.6	90.63
Max diff in one day	3.38	289.79	3.38	289.79	1.59	112.98

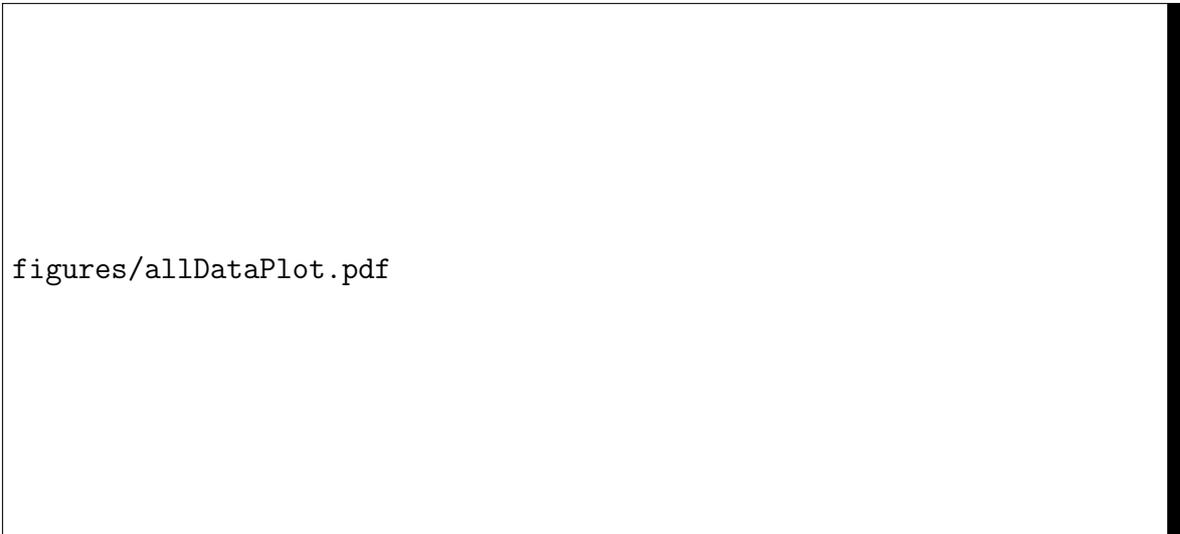


Figure 4.1: All the data from tag 1664

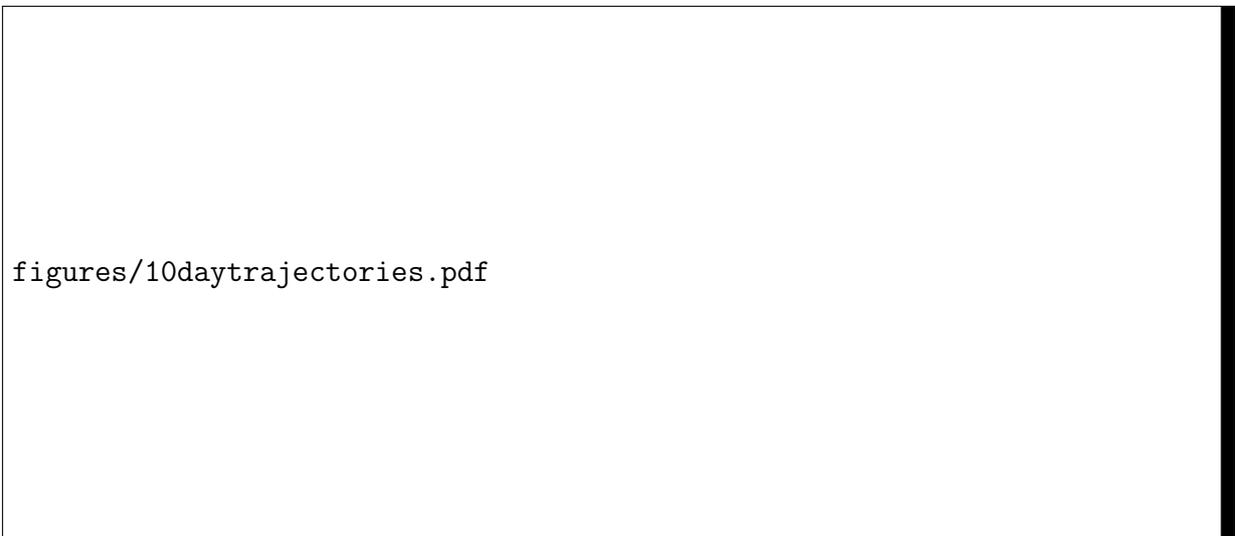


Figure 4.2: Two example 10-day trajectories shown as scatterplot and as time series

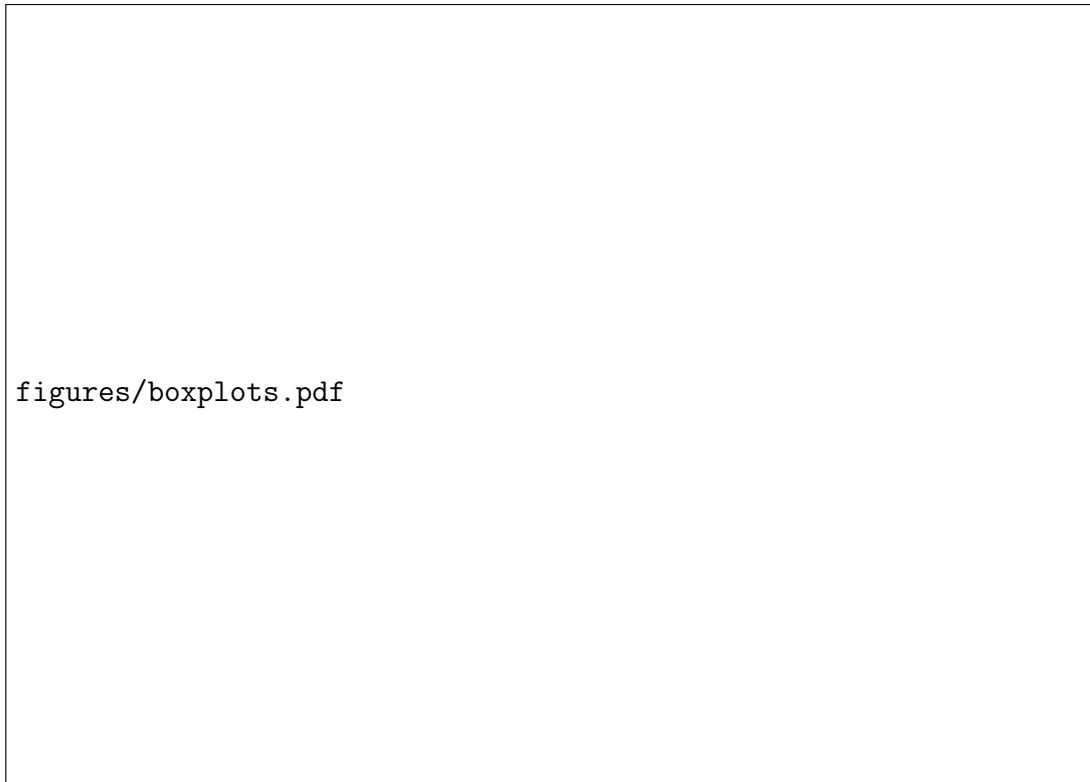


Figure 4.3: Box plots for each month of data

4.2 Basic one-dimensional model

The modelling approach in this thesis builds on the work presented previously in a published paper [31]. The most important ideas and findings are repeated here for the sake of completeness. For complete details, refer to Appendix A. The most important inspiration has been the mobile phone modelling in [42].

4.2.1 Temperature and depth preferences

The modelling approach is motivated by the fact that there is a tendency in data to clustering around specific points in the temperature/depth plane. This tendency is confirmed by referring to Figure 4.4, where the correlation between temperature and depth for each month of the first year of data is shown. For many of the months,

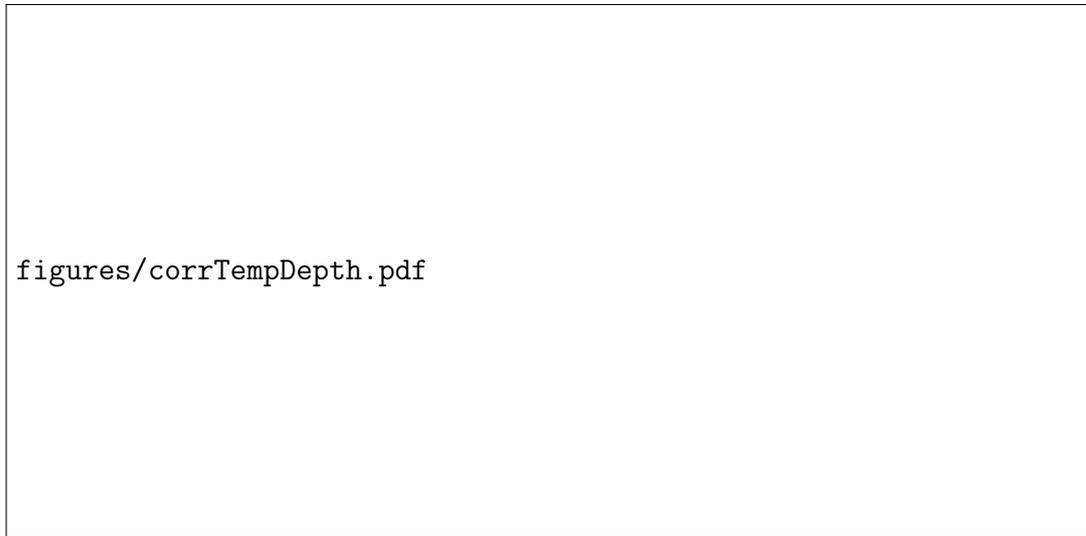


Figure 4.4: Correlation between depth and temperature

the correlation is markedly different from zero. Also, refer to the scatterplot in Figure 4.2 where such clustering is evident for an example time series. Attraction to specific temperature/depth combinations can have biological explanations, such as food availability, the need to avoid predators, physiological factors and the need for specific conditions for spawning. Refer to e.g. [44] for more on the ecosystem in the Barents sea and interaction of fish with the physical environment.

4.2.2 Modelling approach

We start by assuming that within short time intervals, the main characteristics of the fish's vertical movement can be modelled using a single diffusion (Ornstein-Uhlenbeck) process. However, over time, the data shows changes in its characteristics. It exhibits switches in what depth values the fish is localized over, and how much the position varies around such depths. A similar variation over time is visible in the temperature data, though not as clear-cut as for the depth data. We assume that this behaviour could be evidence of switches between different behavioural and physiological processes, such as, among others, feeding, swimming, passive tidal transport and spawning. Thus we can consider that the fish over time is in several states rep-

representing the varying behavioural/physiological processes, with different points and strengths of attraction, and thus we need to use a *mixed* diffusion process to model it. We use the following definitions from [31]:

Definition 4.2.1. A *concentration point* is a point in the space of the data being considered with a high density of data points around it. Concentration points are denoted μ_i . The space of the data will be partitioned into subsets which contain one concentration point each, and the *neighbourhood* of a concentration point will be defined as a region Ω_{μ_i} containing the point.

Definition 4.2.2. A (behavioural) *state* is uniquely defined by the combination of the concentration point the fish was attracted to last and the one the fish is currently attracted to. I.e. a state with movement from μ_i to μ_j is denoted by $\pi_{ij} = \{i \rightarrow j\}$.

The mixed diffusion model suggested involves letting the parameters $\{\mu, b, c\}$ of an Ornstein-Uhlenbeck process $X(t)$ be determined by the state π of a discrete-time Markov chain $\{S_t, t \geq 1\}$. This is similar to the description of a two-dimensional case with a continuous-time Markov chain in Sections 3.2 and 3.2.3. When the Markov chain makes a transition, the new parameters are used instead of the old, in practice starting a new diffusion process with the end-point of the previous diffusion as starting point.

The inference method for the model involves determining:

- the concentration points μ_1, \dots, μ_m from data
- which states of the chain S_t that are observed (and thus form the state-space Π of S_t)
- the probabilities for transitions between the states
- the parameters b and c for the Ornstein-Uhlenbeck process for each state.

4.2.3 Determining a time series of states and Markov chain

The idea in this section is to summarize a slice of a time series of either depth or temperature, $\{x_t, t = 1, \dots, N\}$, into a time series of observed states $\{\hat{s}_t, t =$

$1, \dots, N\}$ to use to estimate the transition probabilities for a Markov chain S_t . To begin with, the slices considered are whole months of the time series. This is an arbitrary choice.

Locating concentration points

We use Kernel Density Estimation (see Section 2.6.4) to locate concentration points μ_i , by detecting values in the space of the data around which there is high density of observations. The concentration points correspond to local maxima (peaks) in the density estimate graph. See Figure 2.3 on the left-hand side for an example where three concentration points are visible. However, the number of peaks in a KDE depends on the bandwidth setting. In Figure 2.3 on the right-hand side, we show how four peaks join into one when the bandwidth is increased.

To choose the bandwidth, the Persist algorithm (see Section (2.6.3) and [30]) is used to find suitable bins for the data (from now on these bins are referred to as Persist bins), and the bandwidth is adjusted so that no bin contains more than one peak. Further, concentration points with low function value in the KDE are ignored. See details on how this was done in [31]. A more systematic approach to adjusting the bandwidth and locating concentration points is suggested in the chapter on future work in Part III of the thesis.

Defining states for the Markov chain

After having chosen concentration points, we define a possible state $\pi_{ij} = \{i \rightarrow j\}$ of the Markov chain for each ordered pair (μ_i, μ_j) of concentration points. The use of each *pair* of points in defining the state is to allow the model to capture for instance different speeds of attraction from below and above in the water column. This gives us a candidate state-space $\tilde{\Pi}$ for S_t . Using the standard deviation σ_μ of the data within the Persist bin containing a concentration point μ , we can define a neighbourhood around μ as an interval $\Omega_\mu = (\mu - n\sigma_\mu, \mu + n\sigma_\mu)$ for some suitable value of n . We will use $n = 1$ to begin with.

State classification of time series

We can now divide the time series into states. Whenever the time series leaves one neighbourhood Ω_{μ_i} and after some time enters a different neighbourhood Ω_{μ_j} (with $j \neq i$), the time that Ω_{μ_i} was left is considered as the time of state-change, to the state $\{i \rightarrow j\}$. Using the algorithm¹ given in [31], we end up with a time series $\{\hat{s}_t, t = 1, \dots, N\}$ of states. Not all possible states in $\tilde{\Pi}$ are necessarily observed, so only those that are will be used to define the final state space Π for the Markov chain S_t . The states are now re-indexed such that we get $\Pi = \{\pi_1, \dots, \pi_{|\Pi|}\}$.

Initial Markov chain state

The first data point in the time series, and those that follow until the first state change is observed, must be assigned a special state $\pi_{\text{initial}} = \{i \rightarrow i\}$. This state represents movement both from and to the same concentration point μ_i , since there is no previously visited concentration point. This state is transient (cannot be returned to) by construction, since there will always be a previously visited concentration point once this state is left.

Estimating transition matrices

Using Equation (2.17) on the time series of states $\{\hat{s}_t, t = 1, \dots, N\}$, we can estimate the $|\Pi| \times |\Pi|$ transition matrix \hat{P} for the chain S_t . We set the initial distribution of S_t as the row vector with 1 in the position for π_{initial} and zero otherwise (i.e. $S_1 = \hat{s}_1$ with probability 1), and together with \hat{P} this gives us the complete description of the estimated Markov chain S_t .

4.2.4 Estimating the Ornstein-Uhlenbeck parameters

Now that the time series has been divided into states, we can consider data observed while the fish is in each state separately, to estimate parameters for the OU processes. It may be that a state is visited several times during the whole times series, and

¹The algorithm is left out here, since a similar algorithm for 2-dimensional time series is presented later in this thesis

care must be taken to combine data from each visit to estimate OU parameters that are representative of all visits to the same states. However, with the methods used for parameter estimation in [31], which are used preliminarily in the following, this is as we will see not an issue.

The drift parameter b is chosen by experiments. The value $b = 0.05$ has, for now, been used throughout all states because this has seemed to give simulation results with speeds of attraction towards the concentration points, similar to that observed in data. For the diffusion parameter c , an estimate of long-term variance is used. From the long-term variance (2.11) $\text{Var}(X(t)) = \frac{c^2}{2b}$ of an OU process $X(t)$, we get a relation between the two parameters, $c^2 = 2b\text{Var}(X(t))$. Thus, the long-term variance for each state from observed data must be estimated. This is done, inspired by [42], by taking the empirical variance $\hat{\sigma}_j^2$ of all data points within the defined neighbourhood of the destination concentration point μ_j . So for state $\pi_k = \{i \rightarrow j\}$, we let $c^{(k)} = \hat{\sigma}_j\sqrt{2b}$, which has the consequence that, for now, the parameters are the same for all states with the same destination concentration point.

The methods for parameter estimation in [31] were simple in order to get a working model with as quickly as possible, as a starting point to work with. In the next chapter, more careful analysis will be presented on how different methods for parameter estimation work, and what assumptions are made.

4.2.5 Example of inference results for 1-D model

We now show some examples of inference results using the methods developed so far. These results are the same as in [31], except that the data used for making the plots is not smoothed. We start with the depth data from Tag 1664 for January 2005.

In the left-hand figure in Table 4.3, we show a flipped kernel density estimate of the data, with the concentration points as lines, and their corresponding neighbourhoods as gray backgrounds. The bandwidth used for generating the KDE is indicated in the figure caption, and the bin boundaries determined by the Persist algorithm are shown as red dashed lines. The right-hand figure shows again the concentration points and neighbourhoods, and in addition shows a plot of the data as

Table 4.3: Inference results from January 2005, depth data

Inference results January 2005, depth	Conc. pt. μ_1 :	97.670	$\hat{\sigma}_1 = 12.47$
	Conc. pt. μ_2 :	172.558	$\hat{\sigma}_2 = 19.98$
$\hat{\mathbf{P}} \approx \begin{pmatrix} 0.9907 & 0.0093 & 0 \\ 0 & 0.9981 & 0.0019 \\ 0 & 0.0030 & 0.9970 \end{pmatrix}$	State π_1 :	$\{1 \rightarrow 1\}$	$c^{(1)} = 1.89$
	State π_2 :	$\{1 \rightarrow 2\}$	$c^{(2)} = 3.00$
	State π_3 :	$\{2 \rightarrow 1\}$	$c^{(3)} = 1.89$
figures/conc_january_2005_1664.pdf			

a function of time. The colour of the plotted line indicates which state the fish is considered to be in.

The concentration points, states, diffusion parameters and the transition matrix are all summarized at the top of the table, with $\hat{\sigma}_i$ indicating the standard deviation of the data within the Persist bin around concentration point μ_i .

4.3 Extending the model and inference methods to 2-D

We now extend the mixed Ornstein-Uhlenbeck model to two dimensions. Recall from Section 2.4.1 that the stochastic differential equation for the OU process for multiple dimensions is

$$d\mathbf{X}(t) = B(\boldsymbol{\mu} - \mathbf{X}(t))dt + \Sigma d\mathbf{W}(t) \quad (4.1)$$

where B and Σ are constant $n \times n$ matrices, $\boldsymbol{\mu}$ is a vector with n entries, $\mathbf{X}(t)$ is an n -variate function of time and $\mathbf{W}(t)$ is an n -variate Wiener process. To use this

process to model fish tag data (\mathbf{D}, \mathbf{T}) , we set $n = 2$. We write

$$\mathbf{X}(t) = \begin{pmatrix} D(t) \\ T(t) \end{pmatrix} \text{ and } \boldsymbol{\mu} = \begin{pmatrix} \mu_D \\ \mu_T \end{pmatrix} \quad (4.2)$$

to clarify that the first component of the process is depth and the second is temperature. Next we need to determine a set of states Π for the mixed process and a corresponding parameter set

$$\{\boldsymbol{\mu}^{(k)}, B^{(k)}, \Sigma^{(k)}\}$$

for each state $\pi_k \in \Pi$. In the following, we will always use depth as the first entry of the vectors and temperature as the second, and often use (x, y) to denote points in the depth/temperature plane. We avoid using (d, t) , since t can be confused with the variable name for time.

4.3.1 Determining 2-D concentration points and neighbourhoods

We use the (x, y) position of peaks in the 2-D kernel density estimate as concentration points $\boldsymbol{\mu} = (\mu_D, \mu_T)^T$. The bandwidth matrix for the 2-D KDE may also have to be adjusted in the same manner as for 1-D data, if several peaks appear within the same Persist bins for either data type.

Choice of neighbourhood

The neighbourhood Ω around a concentration point is used for both observing state changes and for parameter estimation. With a second dimension added, it is no longer an interval, but a region in \mathbb{R}^2 . The size and shape of the neighbourhoods can affect the model results. Some alternative sizes and shapes will be considered in the next chapter, but for now, to establish some preliminary inference results, we settle for a rectangular neighbourhood region defined by ± 1 degree for temperature and ± 15 meters for depth. This seems to be appropriate for some data slices, in order to capture approximately the areas where the kernel density estimate attains its highest values. See Table 4.4 for an example of how the rectangle surrounds a peak in the KDE.

Algorithm for locating concentration points

To wrap up this section, we present Algorithm 1 which is used for determining the set of concentration points. For now, the algorithm uses the unspecified function `getNeighbourhood()`, which takes a concentration point, the time series and the KDE as input and returns a neighbourhood. The parameter h is the KDE bandwidth, and p is a ratio to obtain the minimum threshold for accepting a peak of the KDE as a concentration point. See Appendix B for a description of the other named functions that are not self-explanatory.

Algorithm 1 function `findConcentrationPoints()`

Input: $\mathbf{X} \in \mathbb{R}^{2 \times N}$ (\mathbf{X} = matrix of bivariate time series $(\mathbf{D}, \mathbf{T})^T$)

Output: $\boldsymbol{\mu}, \boldsymbol{\Omega}$ (sets of concentration points and of corresponding neighbourhoods)

- 1: $\hat{\mathbf{f}} \leftarrow \text{KDE}(\mathbf{X}, h)$
 - 2: $[\mathbf{x}, \mathbf{y}, \mathbf{v}] \leftarrow \text{localMaxima}(\hat{\mathbf{f}})$ {Gets position and value of local maxima of KDE}
 - 3: **Discard** entries x_i, y_i of \mathbf{x}, \mathbf{y} where $v_i < p \times \max(\mathbf{v})$ and renumber indices.
 - 4: $k \leftarrow \text{numRows}(\mathbf{x})$
 - 5: $\boldsymbol{\mu}, \boldsymbol{\Omega} \leftarrow \emptyset$
 - 6: **for** $i \leftarrow 1, \dots, k$ **do**
 - 7: $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} \cup \{(x_i, y_i)^T\}$
 - 8: $\boldsymbol{\Omega} \leftarrow \boldsymbol{\Omega} \cup \text{getNeighbourhood}((x_i, y_i)^T, \mathbf{X}, \hat{\mathbf{f}})$
-

4.3.2 State sequence

Given the assumption that it is easy to determine whether an observation (x, y) is inside a neighbourhood Ω or not, the procedure for determining the sequence of states is very similar to the one-dimensional case, given in [31]. The 2-dimensional version is defined by the function `determineStates()` (see Algorithm 2).

The algorithm gives a time series \hat{s}_t of states, and the Markov chain transition matrix \hat{P} for S_t can be estimated exactly as in the 1-D case, again using (2.17). The initial state is set such that $S_1 = \hat{s}_1$.

Algorithm 2 function `determineStates()`

Input: $\mathbf{X} \in \mathbb{R}^{2 \times N}$, $\boldsymbol{\mu}$, Ω **Output:** Π , $\{\hat{s}_t, t = 1, \dots, N\}$

```

1:  $\boldsymbol{\phi}, \boldsymbol{\tau} \leftarrow \mathbf{0}_N$  {zero vector of length  $N$ }
2: for  $t \leftarrow 1, \dots, N$  do {letting  $(x_t, y_t)$  denote a  $(D, T)$  observation at time  $t$ }
3:   for all  $\Omega_i \in \Omega$  do
4:     if  $(x_t, y_t) \in \Omega_i$  then
5:        $\tau_t \leftarrow i$ 
6:   if  $\tau_N = 0$  then {take special care of the end of time series}
7:     find the last time-point  $t_{last}$  such that  $\tau_{t_{last}} \neq 0$ 
8:     for  $t \leftarrow (t_{last} + 1), (t_{last} + 2), \dots, N$  do
9:        $\tau_t \leftarrow \tau_{t_{last}}$ 
10:  for  $t \leftarrow N - 1, \dots, 1$  do
11:    if  $\tau_t = 0$  then {attraction to next point visited}
12:       $\tau_t \leftarrow \tau_{t+1}$ 
13:   $\phi_1 \leftarrow \tau_1$ 
14:  for  $t \leftarrow 2, \dots, N$  do
15:    if  $\tau_t \neq \tau_{t-1}$  then {attraction to new point, i.e. state change}
16:       $\phi_t \leftarrow \tau_{t-1}$ 
17:    else {same attraction as before}
18:       $\phi_t \leftarrow \phi_{t-1}$ 
19:   $\Pi \leftarrow \emptyset$ 
20:   $k \leftarrow 0$ 
21:  for  $t \leftarrow 1, \dots, N$  do
22:    if  $\{\phi_t \rightarrow \tau_t\} \notin \Pi$  then
23:       $k \leftarrow k + 1$ 
24:       $\pi_k \leftarrow \{\phi_t \rightarrow \tau_t\}$ 
25:       $\Pi \leftarrow \Pi \cup \pi_k$ 
26:       $\hat{s}_t \leftarrow \pi_i = \{\phi_t \rightarrow \tau_t\}$ 

```

4.3.3 2-D Ornstein-Uhlenbeck parameters

To determine the parameter matrices B and Σ for each state $\pi \in \Pi$, reference is made to [42] and [6].

Drift term parameter structure

For stability in the diffusion (so that the process does not tend to infinity), we already know from Section 2.4.4 that the drift term parameter matrix B should be positive definite. Many examples of different structures of B have been shown [6], giving different expected trajectories leading towards the attraction point $\boldsymbol{\mu}$. This allows for a wide range of movement behaviours, but it has been argued that it is neither necessary nor realistic to allow this [6]. Instead, it has been suggested to restrict B to the class of matrices that are multiples of the identity matrix, i.e.

$$B = bI = \begin{pmatrix} b & 0 \\ 0 & b \end{pmatrix}.$$

The argument is that these matrices give expected trajectories that are invariant under rotation, which avoids attaching special significance to the coordinate system used for the measurements [6].

We will, however, not restrict B as much as in [6]. For our data, there is a special significance attached to the coordinate system, since we are dealing with depth/temperature observations with different units, and not (x, y) -position in the plane. Restricting B to diagonal matrices with positive entries allows for different strengths of attraction in each dimension, but does not open for e.g. spiralling in towards $\boldsymbol{\mu}$. Since we have chosen to write observations from depth/temperature time series with depth first and temperature second, the entry in position $(1, 1)$ in B corresponds to the drift for depth, and $(2, 2)$ corresponds to the drift for temperature. Thus we write

$$B^{(k)} = \begin{pmatrix} b_D^{(k)} & 0 \\ 0 & b_T^{(k)} \end{pmatrix}$$

for the drift term parameter for state π_k .

Diffusion term parameter structure

Since the diffusion term parameter matrix Σ represents a short-term covariance, it must necessarily be symmetric. In [42], diagonal matrices are used, because it is assumed that there is no interaction or covariance between x and y coordinates. The parameter is thus determined using the variances of data in each dimension. In [6], examples are shown with non-zero covariance. As a simplification, which is acceptable if the model turns out to be good enough, we assume for now that Σ is diagonal (i.e. no covariance). Thus we write

$$\Sigma^{(k)} = \begin{pmatrix} c_D^{(k)} & 0 \\ 0 & c_T^{(k)} \end{pmatrix}$$

for the diffusion term parameter for state π_k .

Parameter estimation

With the diagonal structure of the parameter matrices, we observe that the two components of the 2-D OU SDE (4.1) for depth and temperature, respectively, are completely independent. Because of this, we can estimate parameters for each data dimension separately, and reuse the approach described in Section 4.2.4 to establish OU parameters for the preliminary 2-D inference results.

4.3.4 Preliminary 2-D model inference results

See Table 4.4 for an example of inference results for one month of data for the preliminary 2-D model.

4.4 Chapter summary

This chapter has introduced the Data Storage Tag data, and the basic one-dimensional modelling approach from [31] for modelling the data. An extension to 2-D has been developed. In the next chapter, it will be verified that the model captures important aspects of the fish migration behaviour, and the preliminary choices in the modelling work will be reconsidered to improve model fit.

Table 4.4: Inference results for 2-D model for April 2004 data

Inference results, April 2004	Conc. pt. μ_1 :	(66.3, 6.0)	
	Conc. pt. μ_2 :	(105.6, 6.5)	
$\hat{P} \approx \begin{pmatrix} 0.990 & 0.010 & 0 \\ 0 & 0.997 & 0.003 \\ 0 & 0.003 & 0.997 \end{pmatrix}$	State π_1 :	{1 \rightarrow 1}	$\Sigma^{(1)} = \begin{pmatrix} 0.65 & 0 \\ 0 & 0.00 \end{pmatrix}$
	State π_2 :	{1 \rightarrow 2}	$\Sigma^{(2)} = \begin{pmatrix} 4.59 & 0 \\ 0 & 0.30 \end{pmatrix}$
	State π_3 :	{2 \rightarrow 1}	$\Sigma^{(3)} = \begin{pmatrix} 4.99 & 0 \\ 0 & 0.22 \end{pmatrix}$
<p>figures/conc2d_april_2004_1664.pdf</p>			

Chapter 5

Model validation and improvements

The previous chapter has seen the construction of a stochastic model for depth and temperature data for fish migration, with resulting Markov states and Ornstein-Uhlenbeck process parameters for each state. Any computer model describing real-world phenomena must be verified and validated. For a survey on verification and validation of simulation models, see [22].

Verification of a model is to ensure that the computer program for modelling works as intended (debugging).

Validation of a model is to establish that a verified model is able to capture intended characteristics of the data, i.e. that simulations made using the model are similar to observed data, in some way. The point in making a model is not to fit data exactly, as the only “perfect” model of a system is the system itself [22]. Instead, the model must be good enough for capturing the features of interest. Our model has been designed to capture the attraction to concentration points and the variability within each state, not to replicate all data properties. Thus, we consider our model “good enough” if the average simulated fish spends approximately the same amount of time in each region of the temperature/depth as the actual fish has done, and with a similar variability in each of the states.

This chapter considers the model fit of the basic fish migration model from the

previous chapter, and examines the validity of some of the assumptions made previously. Some improvements to the model are suggested, and alternative methods for estimating the parameters B (or b) and Σ (or c) for each OU process are considered.

5.1 Simulation and validation of the 1-D model

We start with generating simulations of the model for just the depth component of the time series. The reason is that it is far easier to examine and improve on the simulations for one dimension, than considering both dimensions simultaneously. In a later section, the methods and improvements determined in this chapter are applied to the 1-D with temperature data and to the 2-D model. The main difference in considering the 1-D model instead of the 2-D model is that concentration points and corresponding neighbourhoods are differently placed. This is because the peaks in the KDE for the 2-D model corresponds to points with high concentration of nearby data points in both dimensions, not necessarily the same regions that have high concentrations when each dimension is viewed separately.

5.1.1 Simulation time step

According to Section 5 of [31],

“The time step used in the simulation should not be the same as in the data, because the random effects in the fish behaviour occur over a larger time-scale than just one time step of 10 minutes. For instance, one may observe a steady increase in depth over an hour before the depth decreases again, all while the fish is in the same state. Had we used a time step of only 10 minutes in the simulation, we would get large oscillations in the values for very short time intervals, since the random part of the model is very dominant once the process is close to a concentration point.”

A thorough explanation is offered here. In Figure 5.1 an example of depth data is shown for 100 data points starting at April 16 2004 at 00:00. For this data, based on the inference results for the entire month in Table 1 in [31], the fish is in a state which involves attraction towards $\mu = 105.8$. Apart from some high frequency

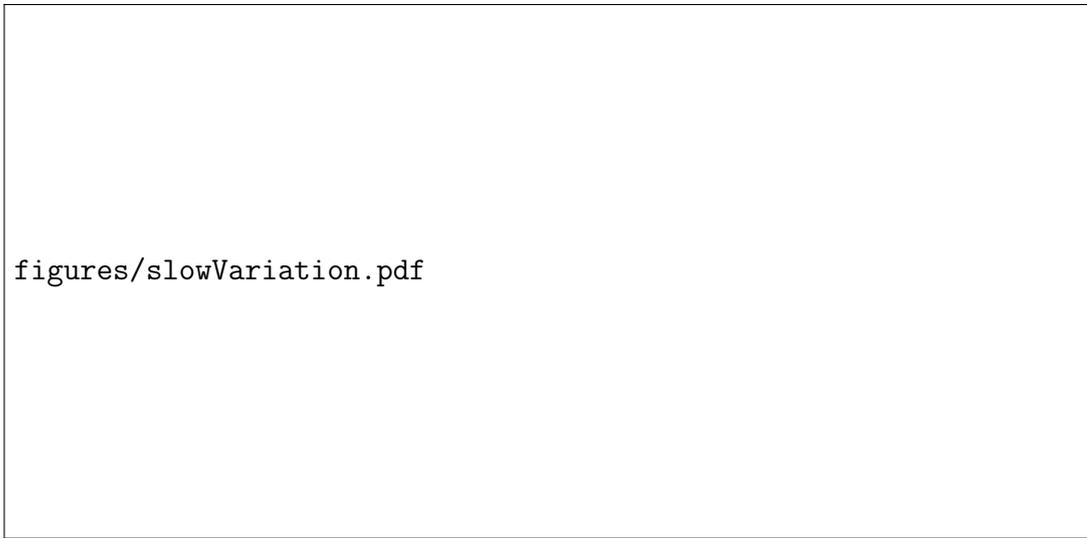


Figure 5.1: Depth data showing slow variation

noise, it seems as though the main variability in the depth happens on a longer time-scale than the very high sampling rate of 10 minutes. In Section 5.3.4, we will examine this noise more carefully. For now, it suffices to conclude that simulations will match data variability better if simulations are made using a longer time step than the data sampling interval. Thus, we use a time step of 12 points, or two hours, in the simulations. This is an arbitrary choice without any heuristic for determining it, but has proved to give satisfactory results.

5.1.2 Simulation details

Most of the discussion and methods in this section are taken directly from [31]. We make simulations $\mathbf{s}^{(i)}$ of the Markov chain and $\mathbf{x}^{(i)}$ of the Mixed Ornstein-Uhlenbeck process, for $i = 1, \dots, numSims$, with $numSims$ a selected number of simulations.

The Markov chain S_t is considered first. For a two hour time step, we note that we must use the 12-step transition probabilities obtained from \widehat{P}^{12} . The number of points \widetilde{N} in the simulation will be 1/12 of those in the original time series, so we set $\widetilde{N} = \lfloor N/12 \rfloor$ to ensure that it is an integer. The chain is simulated as described in Section 2.5.3 on a time grid $t = 1, 2, \dots, \widetilde{N}$ with $\pi_{initial} = \{i \rightarrow i\}$ for the first visited concentration point μ_i as initial state. This gives a state time series s_t . Next, the

OU process is simulated using the first order updating formula (2.14), using $x_1 = \mu_i$ and the parameter set $\{\mu^{(k)}, b^{(k)}, c^{(k)}\}$ corresponding to the current state π_k of the Markov chain. That is,

$$x_{t+1} = x_t + b^{(k)} (\mu^{(k)} - x_t) \Delta t + c^{(k)} n \sqrt{\Delta t},$$

with k such that $s_{t+1} = \pi_k$ and $\Delta t = 12$. The mean time spent in each state of a Markov chain over the whole simulation period is slightly affected by the choice of initial state. However, as the chain evolves, the effect of the initial state diminishes. To obtain simulation results that are less affected by the initial state, it is common to let simulations run for a little longer than needed, and discard the start of the obtained series (see for instance [40]). This is known as “burn-in”. The burn-in here involves discarding the first 200 time steps, and letting the process evolve an extra 200 steps.

5.1.3 Simulation comparison

The resulting simulations can be compared to original time series by plotting them in the same graph, when taking care to align the paths in time (since the number of points in the simulations is roughly $1/\Delta t$ of those in the data). A more systematic approach is to compare the distribution of data to those of simulations using kernel density estimates. Due to randomness in both the Markov chain and in the OU process, the simulations are very likely to be different every time. To get an idea of the “average behaviour” of the model, we take the average of the density estimates of many simulations, and compare that to the density estimates of original data. To ensure that the density estimates are comparable, we make them both using the bandwidth originally selected for locating the concentration points.

Example of a simulation with good model fit

In Figure 5.2 we show on the left an example of a simulation using the states, transition matrix and OU parameters inferred from the January 2005 depth data, as shown in Table 4.3 in Chapter 4. The blue line indicates which concentration

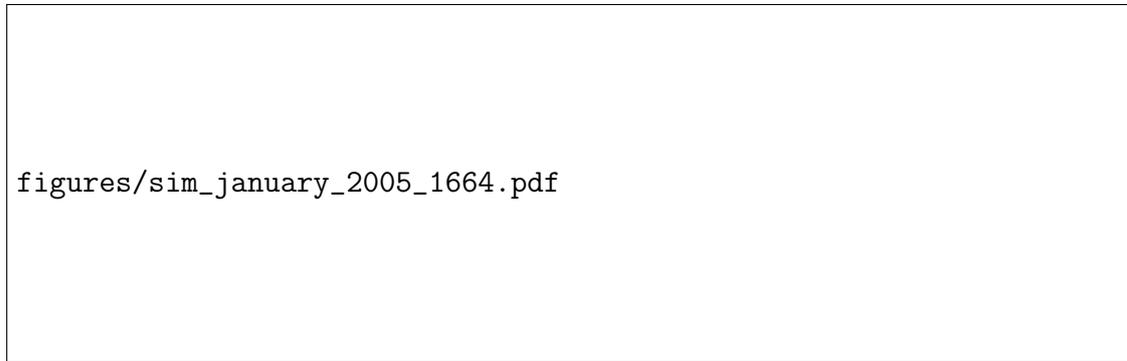


Figure 5.2: Example simulation for January 2005 and KDE comparison plot

point the process is currently attracted towards. The units on the x -axis correspond to the original numbering of the observation data points.

Some important features of the depth data for January 2005 are captured in the example simulation. The process spends time around the same concentration points, and with more variability around the concentration point μ_2 at 172.6 meters than μ_1 at 97.7 meters. A comparison of kernel density estimates for the data and for this simulation (not shown) confirms this tendency. To get an idea of the “mean behaviour” of the model, we show in Figure 5.2 on the right a comparison between the data KDE (green) and the average of KDEs for 50 simulations (red).

Observe, by the higher peak in the KDE, that the proportion of time spent very close to μ_2 is overestimated by the model. This is natural, since there is a lot of area under the green curve to the right of this point, at 200–250 meters. This corresponds to data points that are not within the neighbourhood of any concentration point, and thus this behaviour has not been captured by the model. In [31], Figure 3, simulation results for April 2004 depth data is shown. It is apparent that there is a model tendency to generate simulations with a tighter clustering around the concentration points than in data. This is also confirmed by looking at the single example simulation, where the variability within each state appears smaller in simulations than in original data. This indicates that improvements in the method for estimating c , or for selecting the size of the neighbourhoods, should be considered. But overall, model fit for these two months is quite good, when judged on the basis of the distribution of the data.

Table 5.1: Inference results from May 2004, depth data

Inference results from May 2004, depth data	Conc. pt. μ_1 :	64.2	$\hat{\sigma}_1 = 6.24$
	Conc. pt. μ_2 :	107.7	$\hat{\sigma}_2 = 4.23$
$\hat{P} \approx \begin{pmatrix} 0.9997 & 03 & 0 \\ 0 & 0.9880 & 0.0120 \\ 0 & 0.0789 & 0.9211 \end{pmatrix}$	State π_1 :	$\{1 \rightarrow 1\}$	$c^{(1)} = 0.99$
	State π_2 :	$\{1 \rightarrow 2\}$	$c^{(2)} = 0.70$
	State π_3 :	$\{2 \rightarrow 1\}$	$c^{(3)} = 0.99$
figures/conc_may_2004_1664.pdf			
figures/sim_may_2004_1664.pdf			

Examples of simulations with poor model fit

We now show example inferences and simulations on two slices of data that give poor model fit. The inference and simulation results for depth data for May 2004 are shown in Table 5.1. Observe that the ratio of time spent around each concentration point is switched between data and simulations. This is due to the fact that most of the time spent around the concentration point at 64 meters is during the initial transient state (shown in red in the top right figure). Since this state is transient and likely to be left early, too little time is spent around this point in the simulations.

For the depth data from October 2004, shown on the right in the first figure in Table 5.2, we see that the data has very different characteristics from those of data

Table 5.2: Inference results from October 2004, depth data

Inference results, October 2004, depth data	Conc. pt. μ_1 :	103.1	$\hat{\sigma}_1 = 17.81$
	Conc. pt. μ_2 :	204.4	$\hat{\sigma}_2 = 24.86$
$\hat{P} \approx \begin{pmatrix} 0.9541 & 0.0459 & 0 \\ 0.0385 & 0.9615 & 0 \\ 0 & 0.0018 & 0.9982 \end{pmatrix}$	State π_1 :	{1 \rightarrow 2}	$c^{(1)} = 5.68$
	State π_2 :	{2 \rightarrow 1}	$c^{(2)} = 3.88$
	State π_3 :	{2 \rightarrow 2}	$c^{(3)} = 5.68$
figures/conc_october_2004_1664.pdf			
figures/sim_october_2004_1664.pdf			

considered so far. Transitions occur much more often, and seemingly periodically. The simulation results in the bottom part of the table show that the process spends too much time between the two concentration points. This suggests that the attraction towards the concentration points is too weak, such that the simulated process takes too much time to get to the destination point, compared to the behaviour in data. This calls for adjustment of the drift term parameter b , which so far has been set fixed to 0.05 for all states.

5.1.4 The problem with long-lasting initial states

The problem with the May 2004 data arises because the one visit to the initial state may account for a large proportion of the time spent around the destination concentration point during the time series. Since the state is transient by design, it is likely that the state will be left forever in the simulation of the Markov chain earlier than in the data. This may even happen during burn-in. Thus, the simulated fish spends more time around the concentration points associated with other states.

We examine this problem analytically by considering the invariant distributions of the Markov chains for different months. For the January 2005 data, the proportion of time spent in each state is 0.0240, 0.6048 and 0.3712, respectively, for the states π_1, π_2 , and π_3 . Since states π_1 and π_3 have concentration point μ_1 as their destination concentration points, $0.0240 + 0.3712 = 0.3952$ is the proportion of time that should be spent being attracted towards μ_1 , and the rest of the time towards μ_2 . From the Markov chain transition matrix

$$\hat{P} \approx \begin{pmatrix} 0.9907 & 0.0093 & 0 \\ 0 & 0.9981 & 0.0019 \\ 0 & 0.0030 & 0.9970 \end{pmatrix}$$

for this data from from Table 4.3 in Section 4.2.5 we get an invariant distribution of the Markov chain for January by solving $\lambda_{jan} \hat{P} = \lambda_{jan}$ for λ_{jan} , with the added requirement that the components of the row vector λ_{jan} sum up to 1 (see Section 2.5.2). This gives us

$$\lambda_{jan} = \begin{pmatrix} \{1 \rightarrow 1\} & \{1 \rightarrow 2\} & \{2 \rightarrow 1\} \\ 0, & 0.6196, & 0.3804 \end{pmatrix}$$

with the states indicated above for reference. It has been confirmed that this invariant distribution matches the equilibrium distribution of the chain, by raising \hat{P} to high powers (P^{100000} has been tried) and seeing that the rows of the resulting matrix tend to λ_{jan} . Since π_1 is transient and thus is not revisited, it has zero probability in the equilibrium distribution. The chain S_t thus has, for large t , a probability of 0.3804 of being in a state with attraction to μ_1 . The diagonals of \hat{P} are approximately equal to 1 with one decimal point precision. Hence, we consider accuracy to

one decimal point to be good enough, and within this precision, 0.3952 and 0.3804 are equivalent.

Conversely, consider the May 2004 data discussed in Section 5.1.3. The proportion of time spent in each state is 0.8060, 0.1651 and 0.0289, respectively, meaning $0.8060 + 0.0289 = 0.8349$ is the proportion of time that should be spent attracted to μ_1 . The equilibrium distribution of the transition matrix \hat{P} in Table 5.1 above is

$$\lambda_{may} = \begin{pmatrix} \{1 \rightarrow 1\} & \{1 \rightarrow 2\} & \{2 \rightarrow 1\} \\ 0, & 0.8509, & 0.1491 \end{pmatrix}$$

meaning that the proportion of time being attracted to μ_1 is only 0.1491. Thus, we have shown that the months with long-lasting transient states are very likely to give poor model fit.

It is worth noting that the plot of the January 2005 data shown in Table 4.3 depicts clustering mainly around one concentration point for the first part of the month, and mostly around another at the end of the month. The only reason that we do not observe a long-lasting transient state is the very short visit to state $\pi_2 = \{1 \rightarrow 2\}$ early in the series. A close examination of the data shows that it is in fact only *two* data points within the neighbourhood of the other concentration point μ_2 that causes this transition. This visit to π_2 is so short that it is more likely to be caused by large variability around π_1 at 97.7 meters. This reveals that it is pure “luck” in the modelling choices that lead to the good model fit for this month. For April 2004, shown in [31], the departure from the initial state is much more clearly a change in behaviour. It may be conjectured that the problem demonstrated for months with long-lasting initial states may also be present for other months, though not with as dramatic effects as for May 2004.

5.1.5 Addressing the problem with long-lasting transient states

It has been observed that some months end with a long visit to state, that is only visited once. This produces an absorbing state, and can cause problems similar to those with transient states. This is because the state may be reached earlier in simulations than in data, which creates a bias towards this state. A possible solution

both to this and to the problem with long-lasting transient states is choosing other ways to divide up the data, instead of using whole months. If a longer time period is chosen, it is more likely that the time spent in the initial state accounts for a smaller proportion of the data. As long as all the other states are in the same communicating class, the average time spent in each state over many simulations is more likely to match the time spent in each state in the data. Also, care must be taken such that no periods end with long-lasting absorbing states. Choosing a different division of the data may lead to other concentration points being chosen, because the kernel density estimates are made using larger amounts of data which may have more varying behaviour.

In [52] the frequencies in the fish tag data for the two whole years for this tag is combined with a tidal model. The paper suggests that the fish is in 4 different biological “modes” over the year, resulting in a division of the year into four periods: April-July, August-November, December-January and February-March. The result of determining concentration points from the data from each of these four periods is shown in Figure 5.3. As we can see, for the first three periods, all initial states are short-lasting. For the last period, February-March, there seems to be an abrupt change in behaviour at around 2500 data points when the initial state is left. To solve this, the first part of February can be considered as part of the previous period. We will use this division from now on, and model fit for these periods will be considered later, after some further improvements have been made to the model.

5.1.6 Preliminary conclusions

In [31], it was shown that the model estimated from slices of data (i.e. months) with a long-lasting initial state in the data gave poor model fit, while those with a short-lasting initial state generate model simulations that match rather well. A preliminary conclusion from the discussion here is that a new division of data may help solve this problem.

The fact that the model works rather well for some months without long-lasting initial states, is perhaps a little surprising, considering the many ad hoc choices and the simple methods for determining OU parameters. For other months, such as

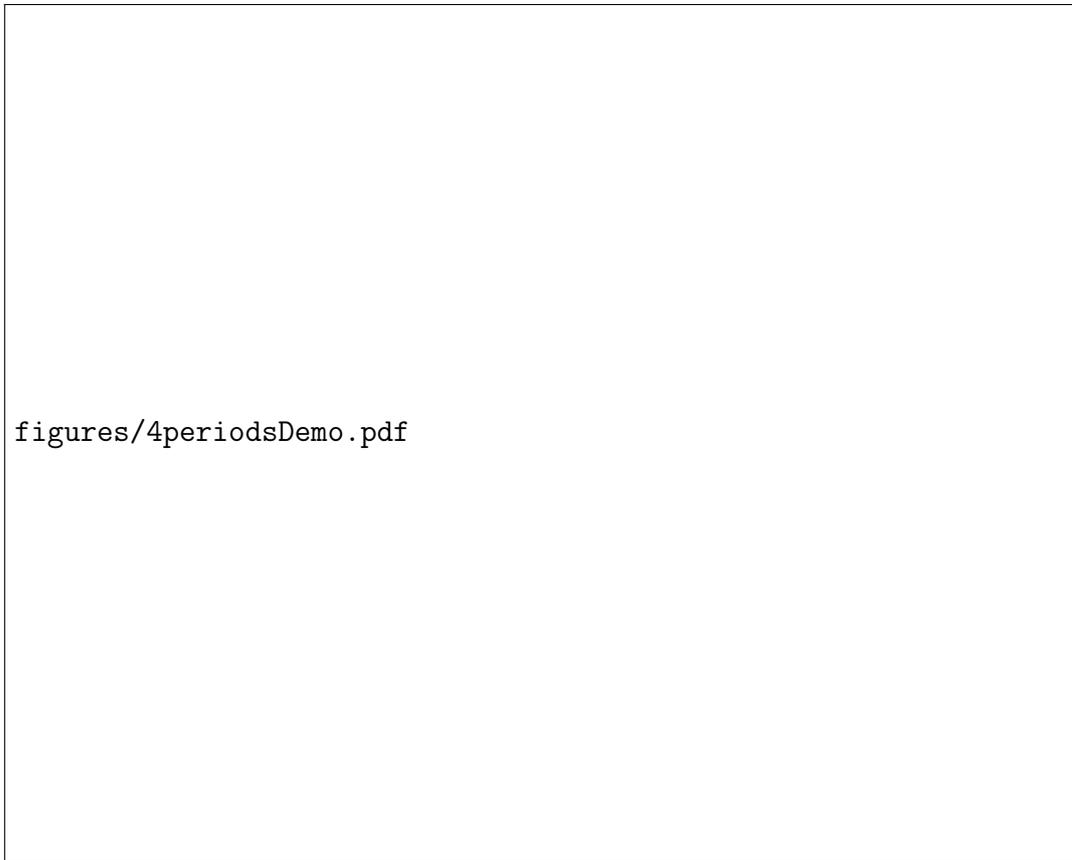


Figure 5.3: Inference on data for four periods

October 2004 whose results were shown in Section 5.2, more systematic approaches to parameter estimation are needed. For months with long-lasting initial states, a solution can be to divide the data up in a different way, such that no period starts with a long-lasting state.

It should be noted, however, that how well the model captures the time aspect in the data has not been examined (for instance, the difference between consecutive data points). Other ways of examining the validity of the model may reveal further points for improvement of the model itself.

Simulations have been made also using the exact OU updating formula (2.15) instead of the first-order formula used so far. It seems that there are no important differences between the results for the two methods, concerning the conclusions drawn here. From now on, only the exact formula will be used.

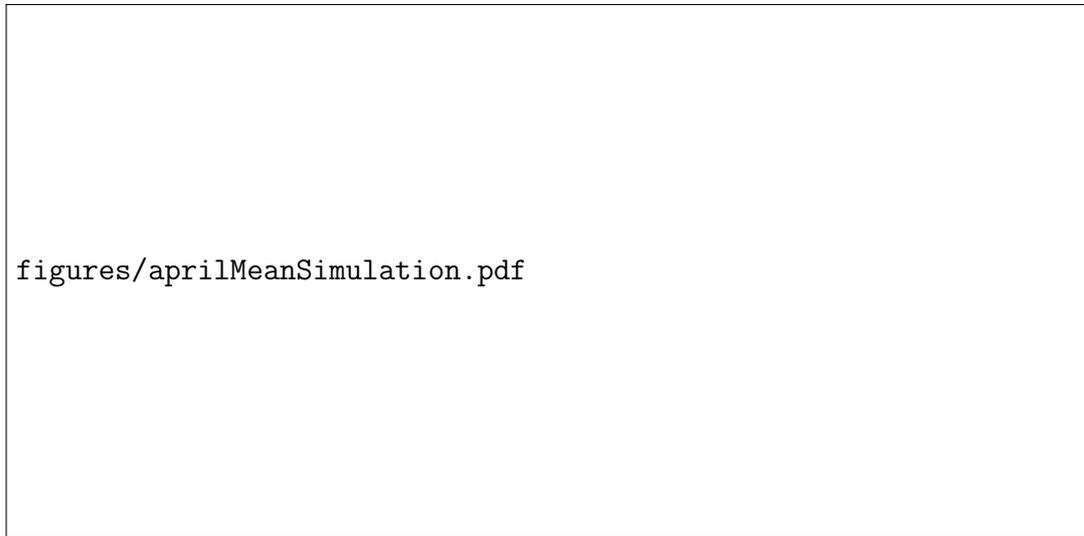


Figure 5.4: Example of mean of 50 simulations

5.2 Creating a mean model path

If the fish migration model is going to be used for certain applications, we may need to generate a single simulated path $\bar{\mathbf{x}}$ that closely mimics the original data, also when the time aspect is considered. A natural way to do this is to take the average of many simulations $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(numSims)}\}$, to get a sort of “mean path” for a simulated fish. A mean of 50 simulations for the April 2005 data is shown in Figure 5.4, and is obtained in this way:

$$\bar{\mathbf{x}} = \sum_{i=1}^{numSims} \mathbf{x}^{(i)} / numSims$$

It can be seen that this averaging essentially cancels out most of the variation in the model, and the mean process spends time around the average of the two concentration points. This is because the Markov chain controlling the behavioural state has an almost equal chance of being in both the recurrent states π_1 and π_2 , since the equilibrium/invariant distribution is $\lambda_{april} = (0.5252, 0.4748, 0)$. Hence a simple average can be inappropriate.

5.2.1 Weighted means

A weighted mean could be an alternative approach, where simulations that are more similar to original data are given higher weight.

Consider a realization $\mathbf{s}^{(i)}$ of the Markov chain S_t , belonging to a simulated path $\mathbf{x}^{(i)}$. The Markov chain has been estimated using the time series s_t of states, represented as the vector \mathbf{s} (downsampled if necessary to make it the same length as the simulation). If the two are similar, the Euclidian distance $\|\mathbf{s}^{(i)} - \mathbf{s}\|_2$ is small. If they are very different, this quantity is larger. We would like to find a weighting $\mathbf{w}^{(i)}$ for each realization $\mathbf{x}^{(i)}$ such that large differences give low weight and small differences give high weight. We do this by using the reciprocal of the norm of differences, and normalize the weights such that they sum up to 1. That is, the computed mean path is

$$\bar{\mathbf{x}} = \sum_{i=1}^{numSims} \mathbf{w}^{(i)} \mathbf{x}^{(i)}$$

with

$$\mathbf{w}^{(i)} = \left(\frac{\|\mathbf{s}^{(i)} - \mathbf{s}\|_2^{-1}}{\sum_{j=1}^{numSims} \|\mathbf{s}^{(j)} - \mathbf{s}\|_2^{-1}} \right).$$

A way to make the weighting even heavier towards promoting paths with small differences from data is for instance to square the norm before it is used in the above formula, This will penalize poor matches and favourize close matches even more.

An example of a mean model paths, made with very heavy weighting of 50 simulations for the April 2004 data, is shown in Figure 5.5. The resulting path is still very different from original data, with only a slight tendency of being closer to the main tendency of the data. The same type of weighting has been tried with a much larger number of simulations, with data from other months and with even more heavy weighting. There seems to be simply too much variation in the order of which states appear in simulations of the Markov chains, compared to the relatively rare transitions in data such as April 2004, for this idea to work.

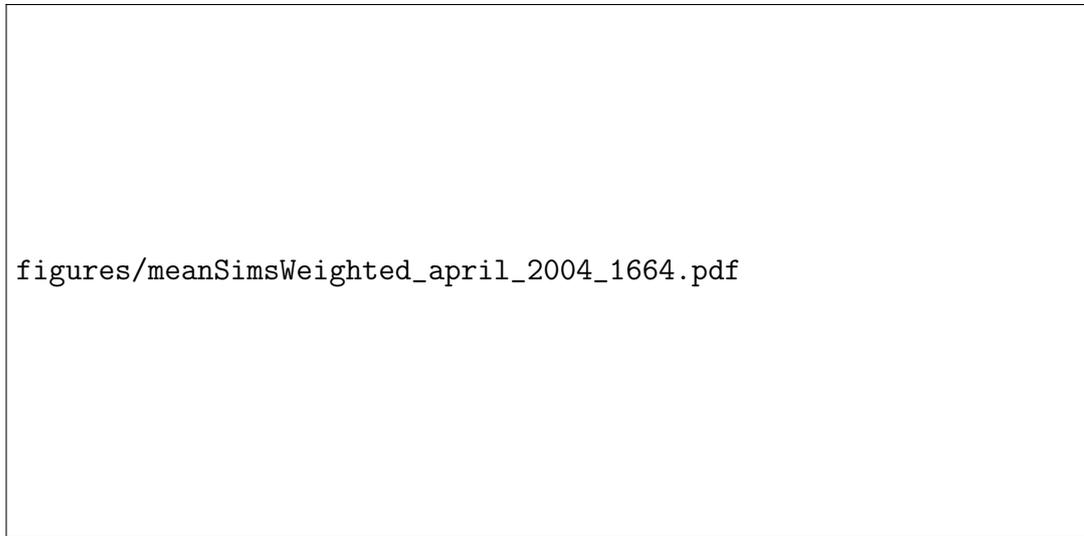


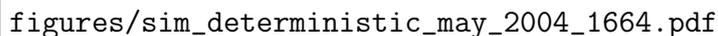
Figure 5.5: Example of weighted mean of 50 simulations

5.2.2 Deterministic state model

Given the poor performance of the mean model path, we consider now an alternative. If it is desired to make simulations that match data closely also in the time-sense, the sequence of states inferred from data can be used directly for the simulation of the mixed OU model. Also, any problems that may remain with long-lasting transient or absorbing states are solved by this. We will refer to the model, with this modification, as the deterministic state model, and use it for the results in the rest of this thesis. To avoid the need for using the entire sequence of states as model parameter, one can instead record the time indices when transition happens to each state, to get a compact description of the model.

The deterministic state model is suitable only if the model is used for modelling the migration of just one fish, since different fish are likely to have both different sets of behavioural states and different times at which each state is visited. Suggestions for how the model can be extended to modelling the migration of several fish will be discussed in Part III of the thesis. Because of this, we will include the consideration of both random and deterministic state models in the rest of this thesis, when appropriate.

The main advantage of using a deterministic state model, is that any problems



figures/sim_deterministic_may_2004_1664.pdf

Figure 5.6: Simulation results for the depth data for May 2004 using a deterministic sequence of states, instead of states generated by a Markov chain.

experienced with long-lasting initial states are solved, since the initial state can simply be used for the first part of the data. No burn-in is necessary since there is no Markov chain. Burn-in for the OU process alone while in the initial state should be unnecessary as long as the process starts at the initial concentration point. See results of simulating the May 2004 depth data using a deterministic state sequence in Figure 5.6, and notice how the model fit is much better than before.

5.3 Model improvements

We now consider some ideas for improvements of the model. This is motivated by studying the inference results, i.e. how data is used to determine states and parameters, by visual inspection of simulated paths and by comparison of kernel density estimates. Also, some of the ad hoc choices made in the model construction are examined more closely.

5.3.1 Removal of short-lasting state visits

For some parts of data, there are examples of very short visits to certain states. For instance, in the first plot in Figure 5.3 showing inference results for the depth data from April-July 2004, we can observe a very short departure from state $\{2 \rightarrow 1\}$ with a visit to state $\{1 \rightarrow 2\}$. This visit consists of only 3 observations starting

close to the end of the period at data index 12113, and is triggered by a short excursion of only 1 observation into the neighbourhood of the concentration point μ_2 at 64 meters. It is reasonable to believe that the excursion to this depth is not the same kind of behaviour as in the first, longer-lasting visit to state $\{1 \rightarrow 2\}$ in early April which involves attraction to the same depth. So the question is whether this excursion really can be attributed to a state change, or instead is just evidence of large variability or noise within the behaviour while the fish is in state $\{2 \rightarrow 1\}$. This situation has been observed at many places in the data.

When using a deterministic state model, the state sequence is reproduced accurately, up to the downsampling of the state sequence used for making longer time steps than 10 minutes. But for the stochastic state model, it has been observed that the short visits can make simulations biased towards the states visited shortly. This is because in some parts of the data, the number of transitions observed is very low, such that just a few extra transitions affects the transition matrix dramatically. This leads to simulations that overestimate the time spent in the state that is visited shortly. An estimated Markov chain does not have any “knowledge” of how long visits last, since it is a memoryless process. The only thing that matters is the number of transitions in data used for estimating the transition matrix.

Since we use a deterministic state model, the bias will not be a problem, but for parameter estimation, very short visits to a state are not usable for determining parameters for the OU process, and also they may affect the values of the parameters considerably. We propose to let the data of any visit to a state shorter than some lower threshold (we use 60 minutes, i.e. 6 time steps) belong to the previous state. Parameter estimation methods must, then, be designed so that this choice does not give biased parameters in these previous states.

5.3.2 Neighbourhood size and shape for 2-D model

In Chapter 4, we settled for an ad hoc choice of neighbourhood shape and size for the 2-D model. Here are some suggestions for more systematic approaches for determining the size and shape of neighbourhoods:

- One alternative is rectangular regions constructed using the standard deviation

of data within bins given by the Persist algorithm for each data dimension. This might be inaccurate since this algorithm is only 1-D, so it does not take into account that states should be persisting in both dimensions. See for instance Table 5.5 in Section 5.5.1, the top left figure, for an example KDE of 2-D data with Persist bin boundaries shown. Note how cuts are placed in the middle of the KDE peaks. An advantage of a rectangular region is that it is very easy to determine if an observation is inside or outside the region.

- Another alternative is a region around the concentration point where the KDE surface $z(x, y)$ is above a certain value, small enough to exclude other concentration points and not to overlap with other neighbourhoods. This is more difficult to handle than the rectangular region, since it both requires a heuristic to determine the threshold value and since it is more challenging to determine whether an observation is inside it or not.
- A third alternative is rectangular regions with size determined using the 2-D KDE. For instance, the widest part in each dimension, of the part of the KDE surface that is above some lower threshold value, can be used. This keeps the simplicity of the first alternative, but exploits the extra information given by the KDE in alternative 2.

If the Persist bin boundaries for each time series cross through a neighbourhood, this is an indication that the neighbourhood size (for any choice of shape and size above) should be reduced by excluding areas that do not belong to the same persisting state.

5.3.3 Analysis of data for visits to states

In order to give insight on how the data in each state should be modelled (for instance, the best way to estimate OU parameters), we now present an analysis of the data belonging to each visit to the state $\{2 \rightarrow 3\}$ in the April-July depth data, as an example. See Figure 5.7, where data for all visits to the states is shown at the top, the first long visit in the middle plot and the final twelve visits to the state are shown at the bottom. The time indexing does not follow that of original data,



Figure 5.7: All visits to state $\{2 \rightarrow 3\}$ for April-July depth data.

but rather the observation number within all visits to the state. A gray dashed line indicates that the following data is from a new visit to the state.

Some observations

From studying the data for state $\{2 \rightarrow 3\}$ for April-July depth data visually, we make the following observations.

- For this state, there is a significant amount of data outside the neighbourhood of the destination concentration point of the state.
- Even outside the bin provided by the Persist algorithms for this concentration point, there is a lot of data. See all data above 150 meters, where the lack of a concentration point for the deepest part of the data has the consequence that

this data falls within this state. Also, at the 4624'th observation of this state, the state change is triggered on the far side of the other concentration point at 64.8 meters. That is because no points on the way are actually *within* the neighbourhood of this point.

- Studying the data this close, it is possible to see that the main variation in depth happens on a slower time-scale than the sampling interval of 10 minutes. This supports the use of the large time step of two hours in simulations. However, there is a certain amount of high-frequency variation. We should consider smoothing this high-frequency noise.

A key question is how much of the variability for the data observed in a state should be used to model the behaviour in a state, and how much corresponds to data features that we have chosen not to model. We will return to this question when considering the parameter estimation techniques in Section 5.4.

Autocorrelation in the time series

The data can be considered as a realization of an OU process if it satisfies the memoryless property of Markov processes, i.e. that the values depend only on the previous time point. This assumption is examined in Figure 5.8, where the autocorrelation (ACF) and partial autocorrelation (PACF) functions for a sample subset of depth data are shown. The data is clearly correlated more than one time step, and from the PACF we see that there is still dependence over more than one lag when removing the dependence on the observations in between. The correlation does vary for different parts of data, and a tendency is that it is never just the first lag that is significant. Thus, the Markov assumption is not valid for this data. Still, as long as parameter estimation methods do not assume that the data is memoryless, the main idea of modelling *attraction* and *variability* using the OU process can still be considered valid.

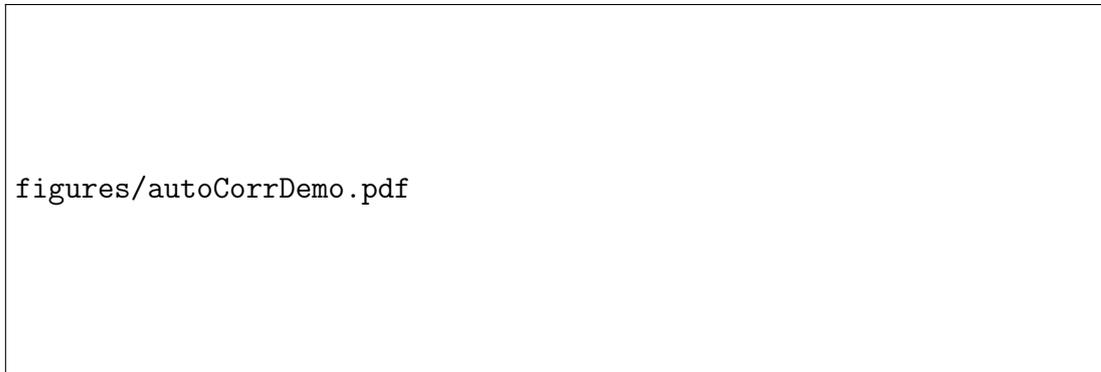


Figure 5.8: ACF and PACF for 1000 depth data points in April 2004, with confidence bounds.

5.3.4 Smoothing noise in the data

Smoothing the time series can be used to remove the high-frequency noise observed, as long as the removed noise is not correlated in time. We will refer to what remains after smoothing as *residuals*. Specifically, we write

$$(x_{res})_t = x_t - (x_{smooth})_t$$

with $\{x_t, t = 1, \dots, N\}$ being a slice of the time series.

Smoothing using moving averages and Savitzky-Golay filters (see Section 2.6.1) has been tested for parts of the data. Experiments with these filters, including variation of the degree and window of the Savitzky-Golay filter, have not been successful in obtaining uncorrelated residuals. Adaptive-degree polynomial filters (ADPF), however, do not require the setting of a fixed degree. Experimentation with the window setting has revealed that it is often possible to find a setting that gives residuals after smoothing that have little or no autocorrelation. This is determined by comparing the autocorrelation function of the residuals with its confidence bounds.

The optimal filter window varies between different parts of data. An idea is to use one smoothing window for the data for each behavioural state. This is only for the sake of model simplicity – there does not seem to be any clear connection between the optimal smoothing window and the state, and the optimal window may vary

between different parts of data within a state, particularly for long-lasting visits. Using variable window setting complicates the inference procedure.

We determine then a filter window setting for each state. Since autocorrelation is not a single number, but a function of the lag, it is difficult to make a robust heuristic for choosing the window. Instead, we introduce an ad hoc method for choosing the window. The method will require user input to the computer program. The autocorrelation function of the residuals after smoothing of data from several visits to a state is examined for several window settings. Only visits that are longer than some shortest length are considered. The aim is to find a minimum acceptable threshold for the autocorrelation for all visits. For instance, a smoothing window which is equal to the average of the two optimal window settings for the longest and the shortest visit to the state can be used. A helping heuristic can be to sum up the absolute value of the autocorrelation function $\rho_k(h)$ of the residuals (for a fixed number of lags, e.g. 20), for each choice k of window, that is

$$\eta(k) = \sum_{i=1}^{20} |\rho_k(i)|.$$

The ACF for the residuals corresponding to the window setting k giving the lowest values of $\eta(k)$ can then be inspected to see if the autocorrelation is sufficiently small, and this can be repeated for all visits to a state. Note that near the start and end of each visit, data belonging to other states may fall inside the filter window. We allow this to avoid having to use a smaller window near the start and end of each visit.

For 2-D time series, a compromise for filter window must be made between both components, such that the values in the two smoothed series correspond to the same time. The error made in this can be considered small in comparison with the total range of the data.

See Figure 5.9 for an analysis of autocorrelation and normality of residuals after smoothing for some example data. The departure from the line in the quantile-quantile (Q-Q) plot indicates non-normality. Even though the smoothing is not perfect, since some of the residuals are correlated, the smoothing is a good start for removing noise and allow the model to focus on the slower varying dynamics of the fish migration. A further point is that any temporal information that is lost when



Figure 5.9: Sample autocorrelation and Q-Q plot of residuals after smoothing, for one short and one long visit to a state of the April-July depth data. Smoothing window used: 27.

smoothing is correlated only for lags that are shorter than the assumed time frame of the slower variations.

The advantage of this smoothing is that the smoothed data for each state can be modelled separately from the noise, using the OU process. Depending on the application of the model, the noise may or may not be added to simulations. Even though the residuals are not normally distributed, their standard deviation is a useful measure of the magnitude of the noise, and can be added as an extra parameter $\sigma_{noise}^{(k)}$ to each state π_k . This can be used to generate a white-noise process to add to the OU process simulation. The smoothing will be used for all the results following.

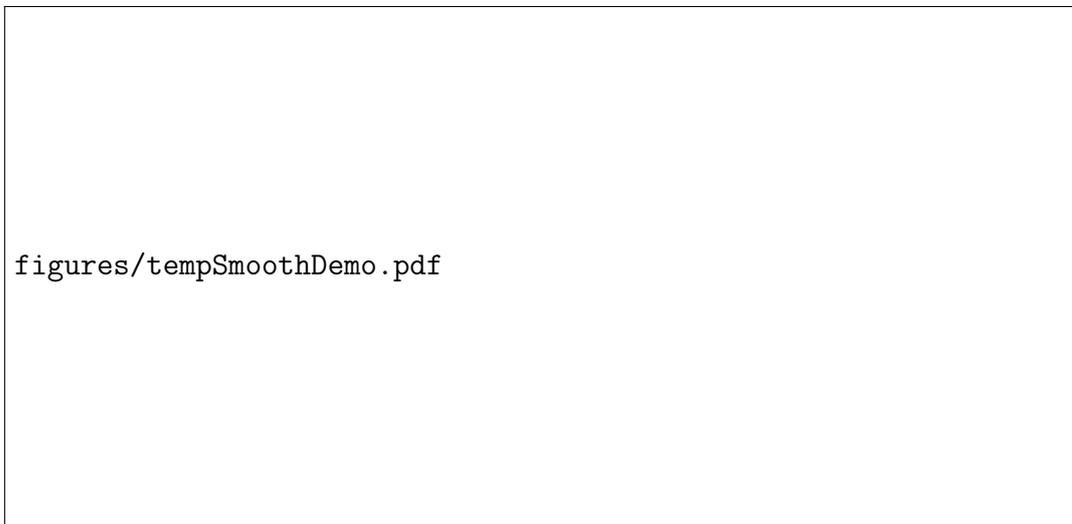


Figure 5.10: Smoothing that fits very closely to precision noise

Smoothing of temperature data

When all temperature observations for the whole time series are ordered according to magnitude, the smallest increment between two observations is 0.031 degrees. This is much smaller than the precision of the data. However, this smallest increment is visible in the data – it can be seen that the series only takes on values on a seemingly discrete set. This can be thought of as noise in the data arising from the precision it is stored with. It may happen that ADPF smoothing with small filter window can fit exactly to this noise (“overfitting”), even if this filter window is selected using the heuristic above. See Figure 5.10 for an example of this. This is undesirable, so very small filter windows must be avoided.

5.4 Parameter estimation methods

With the modifications made to the model so far in this chapter, we are now in a position to discuss different methods for parameter estimation for the OU process parameters for each state. The parameter estimation methods should:

- provide the best parameters for reproducing attraction towards concentration points and variability while in each state

- be able to combine information from each visit to a state when determining the parameters (each visit to the same state can be viewed as a separate diffusion process, but data from each visit should contribute to the estimation of the parameters in an amount corresponding to the number of data points in the visit)
- be possible to use for the data at hand without violating any assumptions made.

It suffices to discuss methods for 1-D modelling, since we have previously chosen to keep all parameter matrices diagonal. Recall that while we use the OU process to model data, we are not actually assuming that the fish moves according to an OU process – just that important characteristics of the data can be modelled by mixed OU process models. So a discussion on parameter estimation methods is not necessarily about improving the model’s ability to capture every detail of the vertical migration data. It is rather about ensuring that those choices made do not introduce any systematic bias into the parameters chosen, and thus bias in the simulations generated.

5.4.1 Least squares and maximum likelihood estimation

The least squares and maximum likelihood methods follow the procedure described in Section 2.4.6. Note that the estimation of μ is superfluous since it is set instead as the concentration point, so the intercept β for the regression method need not be considered. The set of points to include in the regression or likelihood estimation for state π_k is:

$$\{(x_n, x_{n+1}) : \hat{s}_n = \hat{s}_{n+1} = \pi_k\}$$

i.e. both current and next point must belong to the same visit to state π_k . Otherwise, we would include tuples like (x_n, x_{n+h}) , when different states has been visited at the intermediate time-points $n+1, \dots, n+h-1$. The linear relationship, and equal time step assumptions, is not valid for such tuples.

To obtain unbiased estimates for the coefficients when using least squares for solving a linear regression problem, the errors or residuals after regression must be

uncorrelated. We have already shown in Section 5.3.3 that the memoryless (Markov) assumption is not valid in the data for each state, so there is certainly correlation in the residuals after regression. Also, the linear dependence of just one previous value is not sufficient to describe the data – more previous terms should be included. Since both the linear regression and maximum likelihood methods rely on estimating the parameter just by using dependence on one previous point, we reject them both for using them on our data.

5.4.2 Method of mobile phone modelling

The method of parameter estimation from [42] for mobile phone movement modelling described in Section 3.2.3 needs some more precise definitions. The following quantities are recorded:

- \hat{h}_{ij} = the average number of time points spent travelling from the neighbourhood of μ_i to that of μ_j while in the state $\pi_k = \{i \rightarrow j\}$ (note that this would be an average of those values observed for each visit to this state).
- $\hat{\sigma}_j^2$ = the variance of the points in the neighbourhood of μ_j (note that this choice involves using data from *all* states with the same destination concentration point).

Next, the parameters of the OU process for the state $\pi_k = \{i \rightarrow j\}$ are set (we omit the k indices) as

$$c^2 = \lambda_c \hat{\sigma}_j^2 / \hat{h}_{ij} \text{ and } b = \lambda_b / \hat{h}_{ij},$$

where λ_c and λ_b are constants to be determined. We now assume that the empirical variance $\hat{\sigma}_j^2$ of the points that are within the neighbourhood of μ_j can be considered as the long-term variance of the process. Using the expression (2.11b) for the long-term variance, we get

$$\lim_{t \rightarrow \infty} \text{Var}(X_t) = \frac{c^2}{2b} = \frac{\lambda_c \hat{\sigma}_j^2 / \hat{h}_{ij}}{2\lambda_b / \hat{h}_{ij}} = \frac{\lambda_c}{2\lambda_b} \hat{\sigma}_j^2$$

so we see that we need to set $\lambda_c/(2\lambda_b) = 1$ in order for the parameters to reflect the observed variance. We set $\lambda \equiv \lambda_c = 2\lambda_b$ and get

$$c^2 = 2\lambda\hat{\sigma}_j^2/\hat{h}_{ij} \text{ and } b = \lambda/\hat{h}_{ij}.$$

One important difference from the mobile phone case is that so far, we are only using data from a single fish to estimate the parameters, whereas the mobile phone data is from many different phone users. Also, we have observations that are evenly spaced in time, so we use time units instead of actual times.

Setting the λ parameter

In essence, the parameter λ for state π_k decides the relationship between b and c , in such a way that any setting will lead to the same long-term variance, but not to the same “speed of approach” towards the concentration point. In order to choose λ , we note that in the literature [16], the quantity $1/b$ is called the relaxation time. We interpret this as “time before approximate equilibrium is reached” and choose to estimate it as the time from a state change is observed until the destination neighbourhood is reached, which is exactly \hat{h}_{ij} as defined above. Thus, setting $\lambda = 1$ we get $1/b = \hat{h}_{ij}$. This choice gives the same drift term parameter for both dimensions in the 2-D model. So even though we in Section 4.3.3 allowed any diagonal matrix with positive entries for this parameter, we do not use a parameter estimation method that exploits this possibility.

Adjustments

As shown in Chapter 4, the variability in the data was underestimated when using the initial for setting the diffusion parameter c . While in each state, we see in Figure 5.7 that the time series has so much movement outside the neighbourhood, that we underestimate the variability of the fish movement if we continue to only use the variance of the data *within* the neighbourhood for estimating the c parameter for the diffusion term. This can be observed also in the KDE comparison plot in the Figure 5.2 and in Figure 3 of [31], where KDE peaks for simulations are narrower

and taller than in the KDE for the data. We conclude that a modification to this approach is necessary.

First, we allow distinct diffusion term parameters for all states. In the original approach in Section 4.2.4, all states with the same destination concentration point used the variance of all points within the neighbourhood – now we restrict the data used for each state to only the data *belonging* to the state.

Second, for computing the variance, we include all data within the same *Persist bin* as the destination concentration point. This is a larger region than the neighbourhood. This is reasonable, since movement inside the Persist bin is confirmed to be consistent with the time structure of the data, not just with its distribution. To maintain the idea of reaching some sort of equilibrium when close to the concentration point, the first points in the visit *before* entering the destination neighbourhood are not included when computing the variance.

A different alternative is to use *all* data while in the state as basis for computing this variance. However, a consequence of this is that any data such as the deepest data in the April-July depth data in Figure 5.7, which is not associated with any concentration point, is interpreted as variability within states with destination as the closest concentration point. This is undesirable as it leads to overestimation of the variability around the concentration point. Thus, this alternative is rejected.

A consequence of these adjustments is that the role of the neighbourhood diminishes. The main role of the neighbourhood now is *capturing* state changes, and determining when “equilibrium” has been reached. The choice of size and shape of neighbourhoods will therefore not be given more attention. We settle for $n = 1$ for the size of the 1-D neighbourhood (ref. Section 4.2.3), and discuss the 2-D neighbourhood further in Section 5.5.1.

5.4.3 Discussion and conclusions on parameter estimation methods

If data is assumed to be a realization of an OU process, parameter estimation of the drift term b should incorporate information on the strength of attraction at all times – not just before the process has come close to the attraction point, as it has been

done in the methods presented here. However, the idea in this thesis is to model attraction and variability to and around concentration points, not all characteristics of data. Thus, we can consider a parameter estimation method good enough if the mean time taken to reach the destination concentration point, after state change, is comparable to that observed in data. Many simulated trajectories have been studied closely, and it seems that the methods here and the choice of $\lambda = 1$ gives reasonable speeds of attraction for most visits to each state.

5.5 Final model simulation results

A final inference and simulation result for the depth data for April-July 2004 is shown in Table 5.3, and for temperature data in Table 5.4. A deterministic model for the states has been used, but the transition matrices are shown for reference. Smoothing is applied before parameter estimation. Experimentation with the smoothing window for visits to all states have revealed that a window of 27 time steps (4.5 hours) is a good compromise, for both time series, and this has been used throughout. It is the smoothed data that is shown for comparing data to simulations. The parameter estimation is done using the methods of Section 5.4.2, with the Persist bin modification for the long-term variance and $\lambda = 1$ for the drift parameter. The $\hat{\sigma}_i$'s indicate the standard deviation of the data within the Persist bin around each concentration point.

The results for the depth data show a very good model fit, except for the failure to capture the deepest data. The temperature data has poor fit for the variability around μ_1 at 6.4 degrees. This can be explained by the existence of many Persist bin boundaries around the data for state $\pi_3 = \{2 \rightarrow 1\}$ that reduces the estimated variability around this concentration point.

5.5.1 Simulations using final 2-D model

Inference and simulation results for the 2-D model for April-July 2004 data are shown in Table 5.5. A limitation in making this has been the resolution of the kernel density estimate. For such a long time series (17252×2), the memory requirement

Table 5.3: Inference results from April-July 2004, depth data

Results for April-July 2004, depth	Conc. pt. μ_1 :	16.9	$\hat{\sigma}_1 = 10.43$
	Conc. pt. μ_2 :	64.8	$\hat{\sigma}_2 = 6.92$
	Conc. pt. μ_3 :	107.0	$\hat{\sigma}_3 = 12.35$
$\hat{P} \approx \begin{pmatrix} 0.985 & 0.013 & 0 & 0.003 & 0 & 0 \\ 0.002 & 0.998 & 0 & 0 & 0 & 0 \\ 0 & 0.167 & 0.833 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.996 & 0 & 0.004 \\ 0.025 & 0 & 0 & 0 & 0.975 & 0 \\ 0 & 0 & 0 & 0.003 & 0 & 0.997 \end{pmatrix}$	State π_1 :	$\{1 \rightarrow 2\}$	$c^{(1)} = 3.19$
	State π_2 :	$\{2 \rightarrow 1\}$	$c^{(2)} = 4.23$
	State π_3 :	$\{2 \rightarrow 2\}$	$c^{(3)} = 0.00$
	State π_4 :	$\{2 \rightarrow 3\}$	$c^{(4)} = 6.34$
	State π_5 :	$\{3 \rightarrow 1\}$	$c^{(5)} = 2.66$
	State π_6 :	$\{3 \rightarrow 2\}$	$c^{(6)} = 2.62$
figures/conc_april-july_2004_1664.pdf			
figures/sim_april-july_2004_1664.pdf			

Table 5.4: Inference results from April-July 2004, temperature data

Results for April-July 2004, temperature		Conc. pt. μ_1 :	6.4	$\hat{\sigma}_1 = 0.12$
		Conc. pt. μ_2 :	7.2	$\hat{\sigma}_2 = 0.14$
		Conc. pt. μ_3 :	8.5	$\hat{\sigma}_3 = 0.70$
$\hat{P} \approx$	$\begin{pmatrix} 0.990 & 0 & 0.010 & 0 & 0 \\ 0 & 0.999 & 0.001 & 0 & 0 \\ 0 & 0.001 & 0.999 & 0 & 0 \\ 0 & 0 & 0 & 0.998 & 0.002 \\ 0 & 0 & 0 & 0.009 & 0.991 \end{pmatrix}$	State π_1 :	{1 \rightarrow 1}	$c^{(1)} = 0.05$
		State π_2 :	{1 \rightarrow 2}	$c^{(2)} = 0.05$
		State π_3 :	{2 \rightarrow 1}	$c^{(3)} = 0.02$
		State π_4 :	{2 \rightarrow 3}	$c^{(4)} = 0.27$
		State π_5 :	{3 \rightarrow 2}	$c^{(5)} = 0.04$
<p>figures/conc_april-july_2004_Temp_1664.pdf</p>				
<p>figures/sim_april-july_2004_Temp_1664.pdf</p>				

and processing time for the algorithm for 2-D kernel density estimation is so large that it could only be evaluated on a set of points in a 30-by-30 grid. Thus, the accuracy is rather low, and peaks may have been placed a little off their actual position.

The 2-D neighbourhood is chosen based on the ad hoc setting from before of ± 15 meters and ± 1 degrees. However, a modification has been made, manually, utilizing the Persist bins. Any bin boundaries that cut across a neighbourhood, but not in the middle of the peak, are used to reduce the size of the neighbourhood. This can be seen in the plot on the top left in Table 5.5. The inference has otherwise been done as for the 1-D model, with an ADPF smoothing window of 27 time steps. Since the drift term parameters are equal for each dimension, they are given as a single parameter $b^{(k)}$. Data and sample simulations for both dimensions are shown at the bottom of the table, along with an average of the KDEs of 50 simulations. In Figure 5.11 on the left is a surface plot of the KDE of *smoothed* data, and on the right is a plot of the difference of the KDE of smoothed data and of the average of 50 simulations. This shows that the difference between data and model in KDE plots is in the order of 10^{-4} , compared to the scale of the KDE which is in the order of 10^{-3} , and verifies the close match. (Though it is unclear whether the match would be this good if the KDEs were made using a higher resolution.)

Observing the example simulations, the model fit for the 2-D model for the temperature part seems better than for temperature alone in Table 5.4. This is seen e.g. by the larger difference in height of the two highest peaks in the 2-D KDE. This is probably because of slightly differently placed peaks in the 2-D KDE, and because the two Persist bin boundaries in the middle of the peak were ignored. The latter explanation suggests that there is a possibility that this improved model fit may be more because of “luck” than because of a very good model. Thus, more robust methods for selecting neighbourhood size and shape should be used in future work.

Table 5.5: Inference results from April-July 2004, 2-D data

Inference and simulation results for 2D model for April-July 2004 data		
Conc. pt. μ_1 : (65.0, 6.4) Conc. pt. μ_2 : (15.0, 8.6) Conc. pt. μ_3 : (110.0, 6.9)	$\hat{P} \approx \begin{pmatrix} 0.999 & 0 & 0.001 & 0 & 0 & 0 \\ 0 & 0.996 & 0 & 0 & 0.004 & 0 \\ 0.007 & 0 & 0.993 & 0 & 0 & 0 \\ 0.020 & 0 & 0 & 0.980 & 0 & 0 \\ 0 & 0.003 & 0 & 0 & 0.997 & 0 \\ 0 & 0.009 & 0 & 0 & 0 & 0.991 \end{pmatrix}$	
State π_1 : {1 \rightarrow 2}	$\Sigma^{(1)} = \begin{pmatrix} 2.32 & 0 \\ 0 & 0.19 \end{pmatrix}$	$b^{(1)} = 0.0382$
State π_2 : {1 \rightarrow 3}	$\Sigma^{(2)} = \begin{pmatrix} 4.75 & 0 \\ 0 & 0.15 \end{pmatrix}$	$b^{(2)} = 0.0944$
State π_3 : {2 \rightarrow 1}	$\Sigma^{(3)} = \begin{pmatrix} 1.38 & 0 \\ 0 & 0.01 \end{pmatrix}$	$b^{(3)} = 0.0122$
State π_4 : {2 \rightarrow 3}	$\Sigma^{(4)} = \begin{pmatrix} 0.88 & 0 \\ 0 & 0.01 \end{pmatrix}$	$b^{(4)} = 0.0286$
State π_5 : {3 \rightarrow 1}	$\Sigma^{(5)} = \begin{pmatrix} 2.78 & 0 \\ 0 & 0.13 \end{pmatrix}$	$b^{(5)} = 0.1062$
State π_6 : {3 \rightarrow 3}	$\Sigma^{(6)} = \begin{pmatrix} 1.02 & 0 \\ 0 & 0.02 \end{pmatrix}$	$b^{(6)} = 0.0094$
<p>figures/conc2d_april-july_2004_1664.pdf</p>		
<p>figures/sim2d_april-july_2004_1664.pdf</p>		

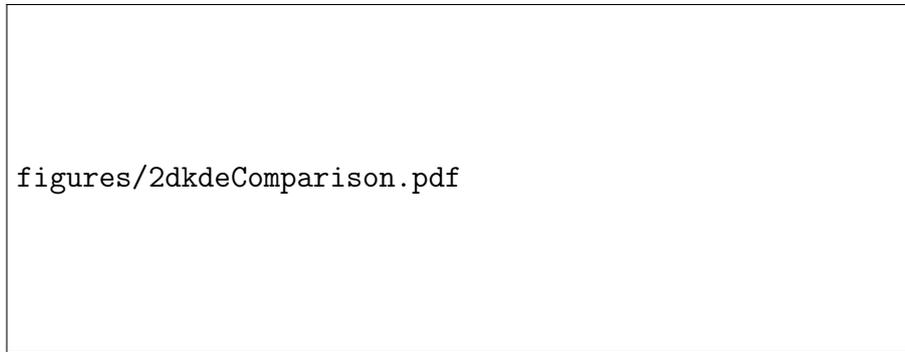


Figure 5.11: Comparison of 2-D KDEs for April-July 2004 data and simulations

5.6 Summary and conclusion

We sum up this chapter by repeating the most important modifications made to the model.

- Short lasting visits to states have been removed.
- The data has been divided in other ways than whole months, to avoid long-lasting transient states.
- The deterministic state model has been introduced as an alternative to the stochastic state model. This model uses the sequence of states inferred from data directly as the sequence of behavioural states in simulations, rather than using a Markov chain for controlling the state.
- Alternatives for the size and shape of the 2-D neighbourhood have been discussed.
- The data in each state is smoothed to remove high-frequency noise.
- Parameter estimation methods are more systematic and capture better the variability in data.

This part of the thesis has shown the successful creation and validation of the model for fish migration patterns, and has suggested many improvements to the model. More can be done to make validation of the model more systematic, and

to create more robust heuristics for making some of the choices that were made manually in the modelling work. There is much potential in expanding the model to e.g. several fish, and for using it for making inferences on cod behaviour on stock levels. See Part III of the thesis for suggestions for future work.

Part II

Geolocalization of fish by optimization

Chapter 6

Geolocalization: definitions, formulations and atlases

Given a time series of temperature and depth observations from a Data Storage Tag, an estimate of the actual geographical location (geolocation) of the fish, expressed in latitude/longitude coordinates, is of interest to researchers. We call the process of making such estimates geolocalization. Two points along the trajectory are known approximately, namely the release and recapture positions. We will use an atlas over temperature and depth to describe the ambient environment of the fish, and try to match DST temperature/depth observations with positions in the atlas. In this chapter we will formulate an optimization task for solving this problem, and define test problems for the optimization methodologies. Development of algorithms, and numerical results, will be presented in the next chapter.

6.1 Simplifications and basic definitions

To begin with, we regard the problem as purely mathematical and assume that atlases are given as continuous functions, and disregard the actual implementation which characterizes the atlases. Further, we ignore the fact that the earth is a sphere, that tidal variations change the depth of the sea and that the tide and ocean currents together may affect maximal swimming speed. Also, we disregard all considerations

of units. We proceed by making some basic definitions.

Definition 6.1.1. (Domain of a depth atlas, or surface of the ocean) We use Ω to denote the domain of a depth atlas. It can be viewed as the surface of the ocean, if the Earth was flat, and is thus a bounded subset of the plane \mathbb{R}^2 . We require that Ω contains its boundary (i.e. that it is closed). Areas of the flat “Earth surface” that are not covered by ocean are not allowed to be contained in Ω . We also assume that Ω is contiguous, since this is the case for the oceans of the world.

Definition 6.1.2. (Depth atlas) A depth atlas is a function

$$\mathcal{D} : \Omega \rightarrow \mathbb{R}_+$$

that assigns a depth value to each coordinate (x, y) in Ω , i.e. we have that depth $\equiv \mathcal{D}(x, y)$. Since all points in Ω correspond to areas of the Earth surface covered by ocean, we have that $\mathcal{D}(x, y) \geq 0$ for all $(x, y) \in \Omega$.

Definition 6.1.3. (The ocean) We define the ocean as a set

$$\Gamma = \{(x, y, d) \in \mathbb{R}^3 : (x, y) \in \Omega, 0 \leq d \leq \mathcal{D}(x, y)\}.$$

We see that Ω is the projection of Γ on \mathbb{R}^2 .

Definition 6.1.4. (Temperature atlas) A temperature atlas is a function

$$\mathcal{T} : \{\Gamma \times \mathbb{R}_+\} \rightarrow \mathbb{R},$$

i.e. a mapping from a combination of coordinates (x, y, d) in the ocean Γ , and a positive real number t which represents time, to temperature. That is, temperature $\equiv \mathcal{T}(x, y, d, t)$.

Definition 6.1.5. (Geolocation problem) Given a length $N+1$ time series (\mathbf{D}, \mathbf{T}) of temperature/depth observations on time-points $t = t_0, t_1, \dots, t_N$, spatial coordinates $(x_{init}, y_{init}), (x_{final}, y_{final})$ for the start and the end points of the time series, a depth atlas \mathcal{D} and a temperature atlas \mathcal{T} and some “speed limit” on maximum change in horizontal position per time unit. The geolocation problem is to find an estimate of a discrete-in-time path

$$\mathbf{P} = \{(x_t, y_t, d_t) \in \Gamma, t = t_0, t_1, \dots, t_N\}$$

(constrained by the speed limit) which is likely to have generated the time series, in the sense that a fish travelling this path will experience temperatures and depths as close as possible to the observations. Path estimates \mathbf{P} of solutions to the geolocalization problem are, through the restriction of the points on the path to the ocean Γ , restricted by the depth of the ocean and the surface area Ω .

6.2 The simplified geolocalization problem

Developing solution techniques for the geolocalization problem requires further simplification of the problem. Let a time series (\mathbf{D}, \mathbf{T}) be given. For testing purposes, we define the simplified geolocalization problem as in Definition 6.1.5 but with the following extra assumptions:

1. The time series is uniformly spaced in time.
2. The depth and temperature atlases are constant over time. Together with the previous assumption, this means that apart from the ordering of the observations in the time series, we can ignore the time aspect.
3. The speed limit is a given constant and is invariant of direction. This speed limit is realistic, in the sense that it is small compared to the total size of the ocean Γ . This is absolutely necessary, otherwise the next point on solution path can simply jump to any position in Γ having the observed temperature.
4. The upper constraint on depth can be ignored, because the sea is always deeper than any depth observations.

6.2.1 Mathematical formulation of the simplified problem

We suggest that the simplified geolocalization can be solved using optimization, and formulate it as an optimization problem. We let the \mathbf{x} , \mathbf{y} and \mathbf{d} coordinates of the path \mathbf{P} be vectors of length $N + 1$, and optimize with these as variables. We use non-negative numbers for indexing time.

The simplified geolocalization problem with time series (\mathbf{D}, \mathbf{T}) of length $N+1$ and initial/final coordinates $P_0 = (x_{init}, y_{init}, d_0)$, $P_N = (x_{final}, y_{final}, d_N)$ is formulated as

$$\underset{\mathbf{x}, \mathbf{y}, \mathbf{d}}{\text{minimize}} f(\mathbf{x}, \mathbf{y}, \mathbf{d}) = \left\| (\mathbf{D}, \mathbf{T}) - (\mathbf{d}, \mathcal{T}(\mathbf{x}, \mathbf{y}, \mathbf{d})) \right\|_2 \quad (6.1)$$

subject to

$$(x_0, y_0, d_0) = P_0, \quad (6.2)$$

$$(x_N, y_N, d_N) = P_N \quad (6.3)$$

$$\|(x_t, y_t) - (x_{t-1}, y_{t-1})\|_2 \leq v_{max}, t = 1, \dots, N \quad (6.4)$$

$$(x_t, y_t) \in \Omega, t = 0, \dots, N \quad (6.5)$$

$$\mathbf{d} \in \mathbb{R}_+^{N+1} \quad (6.6)$$

Here, (\mathbf{D}, \mathbf{T}) should be understood as the row vector obtained by augmenting the row vector \mathbf{D} with \mathbf{T} . Also, by $\mathcal{T}(\mathbf{x}, \mathbf{y}, \mathbf{d})$ we mean the row vector obtained by taking $\mathbf{T}_t = \mathcal{T}(x_t, y_t, d_t)$ for $t = 0, 1, \dots, N$.

The optimization suggested here involves minimizing the difference between observed values of depth and temperature and those given by evaluating the atlas at our variable $(\mathbf{x}, \mathbf{y}, \mathbf{d})$, the estimated discrete path. This distance would be zero if it was possible to exactly match the DST observations in a path through the atlas.

6.3 The ROMS atlas

The way in which a real-world atlas is implemented is necessary for the discussion on solution methods. We now present the atlas which will be used for geolocalization in this thesis. The model used for generating the atlas is a three-dimensional hydrodynamic model known as ROMS (Regional Ocean Modelling System, see [25, 57] and their references). It contains variables such as temperature, depth and salinity. For the application in this thesis, files containing daily averages for the time interval in question, on a grid with a spatial resolution of 4 km, have been provided by the Insitute of Marine Research. Those files where made specifically for use in the

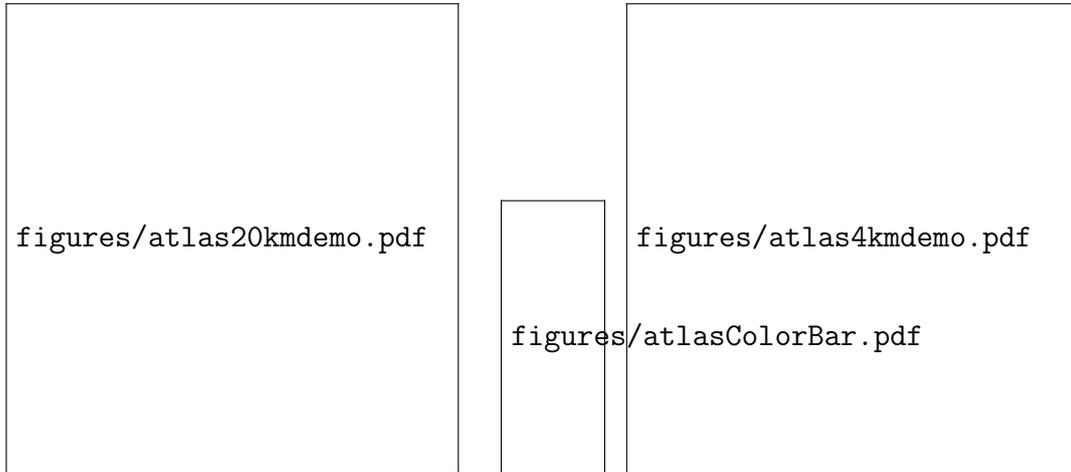


Figure 6.1: 20 km and 4 km atlases for different times

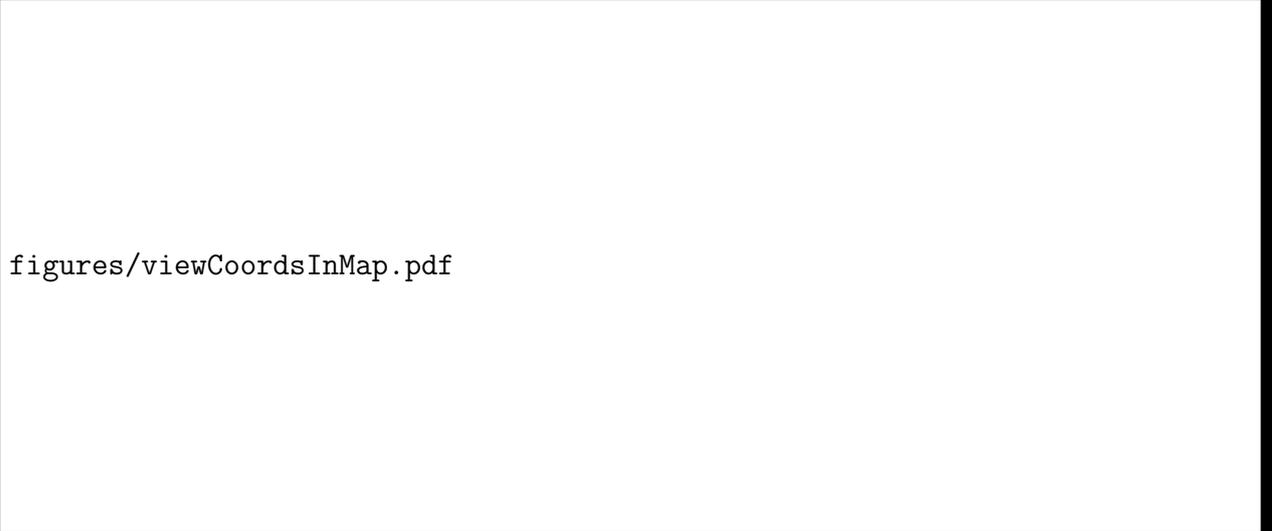
analysis in [57]. For testing purposes, a single file containing a six-day average on a grid with a spatial resolution of 20 km was provided as well. The validity of the ROMS model has been considered in both cited references, and is not considered further here.

The data in the atlas is given on a finite set of coordinates and organized in matrices. For concreteness, we present the relevant contents of the six-day average atlas on a grid in the plane of 20 kilometers. See Figure 6.1 for a plot of surface temperature in a 20 km gridded atlas and a 4 kilometer gridded atlas.

For illustration purposes, we study a subset of size 10-by-8 of the 20 km gridded atlas with a six-day average for the area around the Lofoten Archipelago in Northern Norway. This area corresponds to the intersection of rows 195 to 204 and columns 256 to 264 in the grid matrix. The submatrices associated with this area are shown in Appendix C for reference. In the following, i will denote row direction in the matrix, and j the column direction.

6.3.1 The grid

The grid in the 20 km atlas has 507×329 squares. A matrix each of longitude coordinates M_λ and latitude coordinates M_ϕ for the geographical position at the



figures/viewCoordsInMap.pdf

Figure 6.2: The coordinates in the matrix and matrix row/column indices in map

centre of each square is given. Squares that are adjacent in the matrix are also adjacent geographically. In Figure 6.2, a mapping from the matrices of coordinates to the map and the reverse for the atlas subset is shown.

Errors in distances in grid

In the geolocalization application used in this thesis, the space of coordinates used will be in grid units, and not in actual geographical coordinates. This introduces some errors in the results. Since the Earth is (approximately) spherical, the distance along the surface between all adjacent coordinates in a rectangular grid covering large areas can not be exactly 20 kilometers. The existence of this error has been confirmed by computing distances between points using the Haversine formula [47]. Figure 6.3 illustrates the error involved in assuming that the grid has a spacing of exactly 20 km.

The top plot on the left shows the assumed and actual distances travelled, by moving in straight lines along the i axis, along the j axis and diagonally. The lower plot shows the relative error in distance calculation in each direction, and the maximum of these three. We can assume that the error in travelling along any other direction is bounded by the maximum error in travelling along the coordinates axes

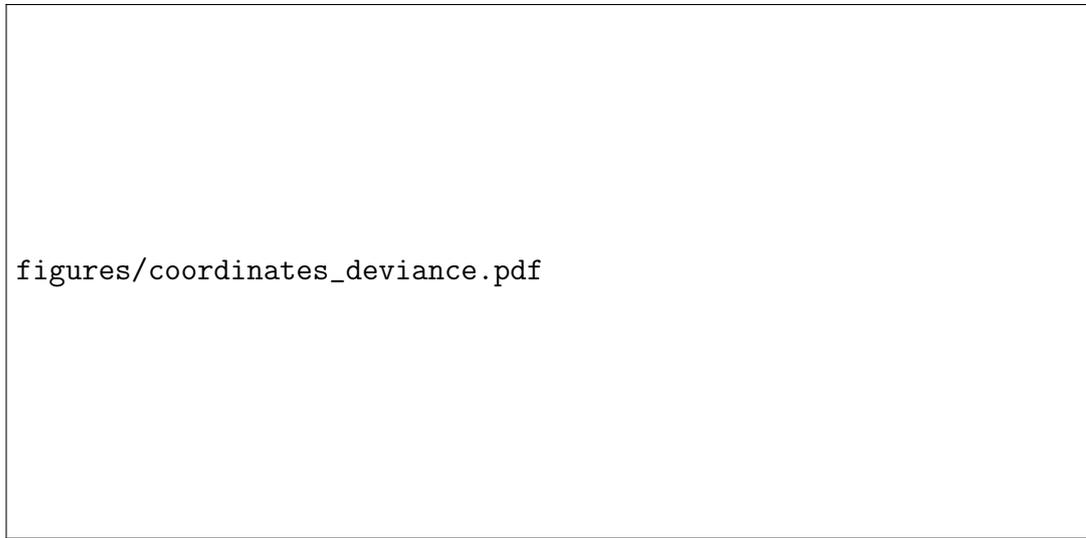


Figure 6.3: Error made in distance calculation in the 20 km gridded atlas when travelling in i direction, j direction and on the diagonal from a point north of Spitsbergen.

and on the diagonal. The conclusion is that in travelling 65 atlas units, the maximum error exceeds 5 percent. Travelling 135 units, the error may exceed 10 percent. For the movement of a fish in one day, say e.g. 10 units, the maximum error is less than 5 %. To compensate, the maximum travelling distance in the optimization can be increased by 5 %. The right-hand figure shows how the straight lines in the matrix unit coordinates translate to geographical coordinates.

It should also be pointed out that the Haversine formula assumes a spherical Earth, while the true Earth is an ellipsoid. This assumption adds a small extra error of up to 0.55 % when crossing the equator, but generally below 0.3 % [26]. This error is negligible for our purposes, and will be ignored. A further point is that whenever geographical coordinates are plotted on a map, there will be errors in the observed distances that depend on the projection used to create the map. So if a line is bent on a map, it might not appear as bent along the true Earth surface.

The only two identified consequences for our application of making these distance errors, is that computing the length in kilometers of a solution path is very inexact and that any speed limits in the optimization are applied less exactly. A solution to the last problem is to increase the speed limit by the error. This may lead to cases

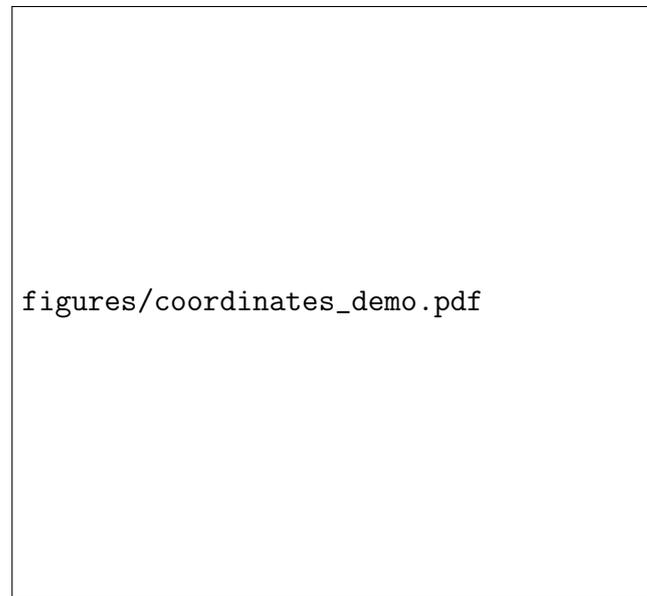


Figure 6.4: Map showing directions travelled when calculating distance errors

where the fish is estimated as travelling faster than its maximum speed limit. These cases can be revealed after optimization by converting the coordinates from grid units to geographical coordinates, and using the Haversine formula. Then, one can attempt to adjust solutions so that they obey the speed limit without increasing the objective function value considerably, or otherwise the whole solution can be rejected. If an optimization procedure is able to locate several optimal paths, this analysis can be used to pick out those that are most realistic.

6.3.2 The depth atlas

The depth atlas consists of a 507×329 matrix $M_{\mathcal{D}}$ giving (averaged) depth values in meters for each cell in the 20 kilometer grid. Depth values are restricted to $d \geq 50$.

The hydrodynamic model does not have defined values at all grid cells, typically when more than half of the area within the grid cell lies on land [24]. To determine which points are included in the atlas, a mask matrix M is given, with 1 for points in the sea and 0 in points that are defined as land. This mask can be “overlaid” any other matrices generated by the model so that only values where the temperature

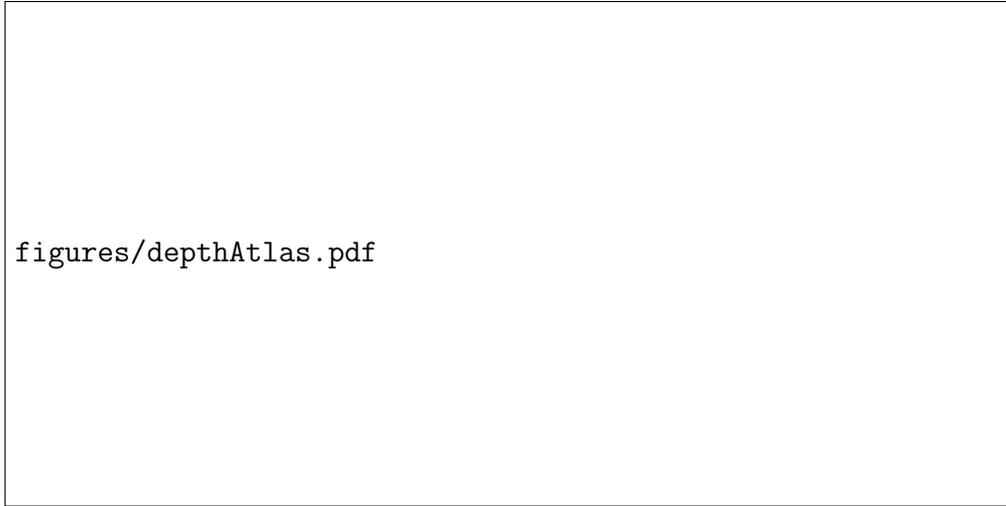


Figure 6.5: Depth values and example surface temperature around Lofoten

atlas is defined are shown. We let this mask define the ocean set Ω . Overlaying the mask gives the depth values for the area around Lofoten shown in Figure 6.5 on the left.

6.3.3 The temperature atlas

The temperature atlas is given as a $507 \times 329 \times 30$ matrix $M_{\mathcal{T}}$. For each square in the grid of the atlas, the temperature atlas provides 30 temperature values, numbered from 1 to 30. These temperature values correspond to each of 30 layers of the depth column. The first entry in the temperature vector is for layer 1 and corresponds to the deepest point, and the last is for layer 30 and is the point closest to the surface. See Figure 6.6 for an illustration.

In order to map each layer to depth values, there is an associated vector s of length 30 that when multiplied with the depth of the ocean at that point will give a vector of (negative) depth values that the temperature applies to. This vector is the same for all grid cells and contains the numbers $\{-\frac{59}{60}, -\frac{57}{60}, \dots, -\frac{3}{60}, -\frac{1}{60}\}$. Since we so far have considered depth to be a positive number, we will from here on take s with positive values instead of negative.

The surface temperatures (layer 30) for the atlas subset considered are shown in



Figure 6.6: Illustration of use of layers for three adjacent points for matrix positions $i = 203$ and $j = 259, 260, 261$.

Figure 6.5 on the right.

6.3.4 Time aspect

The temperature in the sea varies continuously over time, so the atlas provides temperatures for different times. The finest time-resolution in the files provided for this work is daily averages. Hence for the remainder of this chapter, we will consider a whole day as the smallest possible time unit for geolocalization. The amounts of data for each point in time (≈ 400 MB per daily average) suggests that any solution method for the geolocalization problem must allow for non-storage of all data in memory at a time.

6.3.5 Continuous extension of the atlas

Since the temperature in the atlas is given on a finite point set, interpolation of temperature values in three dimensions is necessary for the assumption of a continuous atlas. Methods for interpolating the atlas are presented in the next chapter.

6.4 Test instances for testing optimization methods

We have a mathematical formulation of the simplified geolocalization problem, and a general idea on how a real-world atlas works. Before discussing methods for solving the problem, we suggest a method for generating test instances of the problem, in order to be able to determine how well solution methods are applicable to the problem.

The idea is to generate a synthetic fish trajectory. Start with real-world atlases \mathcal{D} and \mathcal{T} , the latter evaluated at one point in time. Define a test path that is well within the ocean Γ given by the atlas, satisfying a speed limit. Generate depth/temperature time series by evaluating the (interpolated) atlas at the points on the path, and try to recover the path using optimization. The path can either stay well within the boundaries of the ocean, so constraint (6.5) can be relaxed, or the problem can be complicated by letting the path be close to the boundary.

An arbitrary test path within the ocean can be constructed by using a simulation procedure similar to a random walk. We make a path of length $N + 1$ where the time unit is one day. A starting point (x_{init}, y_{init}) is given, and an upper limit v_{max} (measured in grid units) for horizontal movement must be specified. The mask M is provided for checking the ocean constraint. Realistic movements in the depth column can be mimicked by using true DST depth observations, denoted \mathbf{D} . See Algorithm 3 for a procedure used for generating a test path of length $N + 1$.

The normal probability distribution for movements must use a variance that makes movements in one direction that exceed the speed limit unlikely. Note also that any horizontal movements that exceed the speed limit are ruled out by the

if-statement. Some further options for test paths are:

- only allow movement at the exact positions that temperature information exists without using interpolation
- adopting smaller time steps than one-day steps, and downsampling the temperature/depth time series as described later.

Algorithm 3 function $[\mathbf{x}, \mathbf{y}, \mathbf{T}] = \text{makeTestPath}(\mathbf{D}, M, v_{max}, N, x_{init}, y_{init})$

$x_0 \leftarrow x_{init}$

$y_0 \leftarrow y_{init}$

for $i \leftarrow 1, \dots, N$ **do**

$\text{step_ok} \leftarrow \text{false}$

while not step_ok **do**

$n_x, n_y \sim \mathcal{N}(0, \sigma^2)$

$(x_i, y_i) \leftarrow (x_{i-1} + n_x, y_{i-1} + n_y)$

if $n_x^2 + n_y^2 \leq v_{max}^2$ **and** $M_{\text{round}(x_i), \text{round}(y_i)} = 1$ **and** $d_i \leq \mathcal{D}(x_i, y_i)$ **then**

$T_i \leftarrow \mathcal{T}(x_i, y_i, d_i)$

$\text{step_ok} \leftarrow \text{true}$

6.5 Towards the original geolocation problem

Provided that an optimization method has been found that successfully solves the simplified problem to (approximately) recover a test path for several test instances, we can work on extending the mathematical problem description towards the original geolocation problem. We can view this as introducing “difficulties” to the problem. Correspondingly, increasingly “difficult” examples should be constructed and tested. Then the optimization methods should be adjusted, if necessary, to recover the test paths. It might be fruitful, if possible, to introduce each extra difficulty separately, introduce adjustments to optimization methods and then later examine how the difficulties *added together* affect the effectiveness of the solution methods.

The following subsections suggest how more complexity can be introduced and corresponding changes in the mathematical formulation and in synthetically constructed examples.

6.5.1 Favours temperature matches

With synthetic examples constructed from the real-world atlas, we do know that there exists at least one match of the temperatures on the test path, even though it may be difficult to find using optimization. In reality, the atlas is made using a model, with errors, and the DST data contains measurement errors due to averaging and the imprecision of individual measurements. With real data, it is less likely that there exist exact matches of temperature/depth combinations for the whole time series, within the swimming range of the fish.

With the formulation of the objective in (6.1), the depth variable \mathbf{d} can be set freely by optimization algorithms. Thus, it may be that the depth time series can be matched exactly in the optimization by simply setting $\mathbf{d} = \mathbf{D}$, while the temperatures will be only “close to” those observed in the time series. This is equivalent to favouring solutions where the differences in depth between time series and candidate path have zero contribution to the objective function, over those paths with only approximate matches in depth, but better matches in temperature. To adjust this, we can add lower weight to differences in depth in the objective function by introducing a parameter, a factor $\alpha \in [0, 1]$, and redefining the objective function (6.1) like this:

$$f(\mathbf{x}, \mathbf{y}, \mathbf{d}; \alpha) = \left\| (\alpha \mathbf{D}, \mathbf{T}) - (\alpha \mathbf{d}, \mathcal{T}(\mathbf{x}, \mathbf{y}, \mathbf{d}, \boldsymbol{\tau})) \right\|_2. \quad (6.7)$$

Subsequently, α can be decreased (from an initial value of 1) to see the effect of such weighting.

An alternative is to use the depth time series directly in the optimization, instead of having depth as a variable. When developing methods, this is an advantage, because it will be easier to analyze the problem for each step in 2-D rather than in 3-D. We can view this as setting the α parameter above to 0, so the objective

simplifies to

$$\tilde{f}(\mathbf{x}, \mathbf{y}; \mathbf{D}) = \left\| \mathbf{T} - \mathcal{T}(\mathbf{x}, \mathbf{y}, \mathbf{D}, \boldsymbol{\tau}) \right\|_2. \quad (6.8)$$

When solving the problem for real DST time series, or for e.g. a constructed example with noise added, the depth variable should ideally be free, so that there is some degree of flexibility in searching for the target temperature. An idea is to restrict the depth to an interval around the observed (daily averaged) depth.

6.5.2 Adding the depth constraint

Add the depth constraint to the mathematical description, by replacing constraint (6.5) by

$$(x_t, y_t, d_t) \in \Gamma, t = 0, \dots, N. \quad (6.9)$$

It should be noted that for averaged time series, it might be necessary to retain the depth of the deepest point visited during the day, and match the constraint with this value instead of the average. This is particularly relevant if the fish is assumed to swim close to the seabed, where this constraint can be used to resolve a tie between two different paths that are otherwise equally good solutions.

6.5.3 Introduce a time-varying temperature atlas

The addition of the possibility for an atlas to be time-varying requires the temperature atlas to be evaluated at the time that the point in the path corresponds to. We introduce the vector $\boldsymbol{\tau} = (0, 1, \dots, T)^T$, such that it gives the time-index (day index) for each observation in the time series. We then replace objective function (6.1) by

$$f(\mathbf{x}, \mathbf{y}, \mathbf{d}) = \left\| (\mathbf{D}, \mathbf{T}) - (\mathbf{d}, \mathcal{T}(\mathbf{x}, \mathbf{y}, \mathbf{d}, \boldsymbol{\tau})) \right\|_2 \quad (6.10)$$

A time varying atlas can be simulated, for creating and experimenting with constructed examples, by making time-varying random or systematic perturbations to the values in a time-constant atlas. Then we can test how this affects the efficiency of solution methods.

6.5.4 Downsampling strategies

Given a real DST time series, with a 10 minute sampling interval, it is not obvious how to match it against a time varying atlas with values given for each day. We could use linear interpolation of the atlas in time, to get temperature values for each point at 10 minute intervals. This would, however, not reflect the daily variability in the temperature, since the values given are daily averages. Instead we consider the following alternatives for *downsampling* the DST data:

1. using every 144th observation (since there are 144 observations per day)
2. do a moving average smoothing of the time series and pick every 144th smoothed value
3. compute averages of all 144 observations from each whole day

It is reasonable to use alternative 3, since it most closely resembles daily average nature of the temperature atlas.

6.6 Maximum travelling distance for the fish

The swimming speed of fish has been studied e.g. in [2]. The speed depends on many biological, physiological and environmental factors, and varies between fish species. It is common [56] to express the maximum swimming speed as proportional to the body length L , e.g.

$$v_{max} = \lambda \times L$$

where the λ factor is species dependent and measured in body length (in centimetres) per second. In laboratory experiments, [56] reports λ values of 2.1 and 2.6 for sustained swimming speed for atlantic cod. A study with high-resolution tracking using acoustic tags [13] showed swimming speeds for atlantic cod giving $\lambda = 2.09$, but showed mean speeds that were less than half of this.

The maximum swimming speed can be used to establish an absolute maximum on the travelling range of the fish in one day. However, it is not realistic that a fish swims in a straight line at maximum speed for a very long period of time. In [41],

the maximum displacement in one day for the migrating cod studied was 59 km (one single day in three years) while mean displacement in one day varied from 6.6 to 24 km. From this we can conclude that the maximum swimming speed should be used only as an absolute upper limit on daily displacement, and that lower swimming speeds are more likely for most of the time. As a rough estimate, we will use $\lambda = 2.0$ in the geolocalization, and adjust if experience shows this to be necessary.

In Table 4.2 in Chapter 4, the maximum displacement in the vertical direction between 10 minute observations and over a 24 hour period are given for tag 1664. The maximum vertical displacement of 290 meters in one day is very small compared to the maximum swimming speed of the fish. This may not be the case for all fish, but to simplify we will ignore the vertical component of the swimming when considering speed limits.

6.7 Solution methods for the optimization problem

We describe in this section how the optimization could be solved using different approaches to optimization, and include some considerations on implementation issues.

6.7.1 Three solution approaches

Total optimization

The optimization problem described above, with constraints, can be implemented directly into a standard computer solver with a huge search space of $3(N + 1)$ variables. However, since the amounts of data in the atlas for each day is very large, not all temperature information can be loaded into memory at one time. Loading and unloading data is time-consuming, and not tractable unless it is done systematically so that as much as possible is done at each time step before unloading. Most standard optimization solvers treat all variables equivalently, and move through the search space in a theoretically efficient manner (for instance in a descent direction found

using supplied derivatives or finite differences, for line search methods). It will require severe modification of standard software for optimization to implement searches that do not use information from many points in time at once. However, with a time constant or artificially time-varying atlas, we can test this approach. This can give insights into how other methods may work.

Most numerical optimization solvers for non-linear problems require a starting point for all variables. A good start is a straight line from P_0 to P_N , with the depth variable set equal to the depth series. This may not be a feasible point, for instance if the ocean is not deep enough or the path crosses land. If that is the case, an option is to adjust the initial path so that it becomes feasible.

Local optimization of total function

A simple optimization method is 1-parameter line search or the coordinate search method [33]. This method searches along one variable for a minimum, and proceeds with the next variable. There are no guarantees of convergence and efficiency. An advantage, however, is that data for one dimension is all that is needed at a time, compatible with the way our temperature data is organized. An idea is to extend this method to solving for three variables at a time, x_t, y_t, d_t , such that only temperature data for time step t is needed. This is similar to *pattern search methods* [33], where a set of search directions are chosen at each iteration. The search space at each iteration is rather small because of the horizontal speed limit, making it possible to search it efficiently. This approach also requires a starting path.

Ensemble search

This approach involves solving a succession of local optimization problems for each time step, without considering the objective value for the whole time series. An advantage is that we do not need a starting path for the optimization. Let the position P_0 for the first time step be given. Then, given an estimated position on day n , and an observed temperature on day $n + 1$, what position within the swimming range of the fish gives the best temperature match? This introduces a *local* optimization problem in 3-D. This problem may have several good solutions, with only few or

none being close enough to the actual position. The best solutions define an ensemble that may be retained, and used as starting points for the next time step. This gives rise to an increasing number of candidate paths, and a strategy must be chosen to keep the paths that give the best total match (so far, in time) to the temperature series. It is also interesting to see if the point of recapture, P_N , is actually found using this method.

6.7.2 Implementation issues

We now discuss some aspects of implementing the optimization problem on a computer. It will be discussed mostly briefly, as it is desirable keep discussions in this thesis on a principal level using pseudocode.

Optimization problem algorithms

To solve the optimization problem in the previous sections, no matter whether the total, local or ensemble search variant is used, the following computer methods are suggested for solving the optimization problem:

- Standard solvers for constrained non-linear optimization. Derivatives are constructed using finite differences. Such methods are often more efficient if the objective and constraint functions are continuous and differentiable, and even though we interpolate the temperature atlas, this may not be true unless all the ocean constraints are interpolated as well, using smooth functions. A disadvantage of such a method is that it can easily get stuck in a local minimum.
- Derivative-free solvers for unconstrained optimization. The objective function may be implemented such that infeasible solutions are penalized.
- Derivative-free solvers supporting constraints, that use transformation of variables to implement the constraints.
- Integer programming, by searching only at the integer points or at a set of discrete points.

Software problem formulation

There are many ways to specify optimization problems in different programming languages or modelling systems. Here, only MATLAB will be considered, and we now present the standard way of specifying the problem as used by built-in functions such as `fmincon()` and `fminsearch()`. These are solvers for finding a local minimum of a function. The following is required (note that \mathbf{x} here should not be confused with the one horizontal component of paths in our optimization problem, it is simply a vector of unknowns):

- Objective function $f(\mathbf{x})$, taking a vector $\mathbf{x} \in \mathbb{R}^n$ (the n optimization variables) and returning a real number (the objective function value).
- Starting point \mathbf{x}_0 . Does not necessarily have to be feasible, but choice of starting point may affect whether a solution can be found, the rate of convergence and which local minimum is found if there is more than one.

Constraints can be given as:

- upper and/or lower bounds on each element of \mathbf{x} , i.e. two vectors \mathbf{l} and \mathbf{u} giving the constraint $\mathbf{l} \leq \mathbf{x} \leq \mathbf{u}$.
- constraint matrix A and right-hand side vector \mathbf{b} of inequality constraints on the form $A\mathbf{x} \leq \mathbf{b}$. Correspondingly for equality constraints $A\mathbf{x} = \mathbf{b}$
- a function $c : \mathbb{R}^n \rightarrow \mathbb{R}^m$ for m non-linear inequality constraints, taking in a current solution \mathbf{x} and returning a vector of m values. The current solution is regarded feasible if $c(\mathbf{x}) \leq 0$. Correspondingly a function $c_{eq}(\mathbf{x})$ for non-linear equality constraints, where the requirement for feasibility is $c_{eq}(\mathbf{x}) = 0$.

Variables and objective function

To satisfy the format given in the previous section, the path $\mathbf{x}, \mathbf{y}, \mathbf{d}$ of 3 coordinates at $(N + 1)$ time points must be written as a long vector of $3(N + 1)$ elements. We do this by augmenting them, giving us

$$\mathbf{p} = (\mathbf{x}^T, \mathbf{y}^T, \mathbf{d}^T)^T \in \mathbb{R}^{3(N+1)}$$

as optimization variable. Then, to convert from numbering of indices in the path back to separate coordinates, we subtract $(N + 1)$ for the y coordinates in the middle and $2(N + 1)$ for the d coordinates at the end of the vector. To consider the point in the path at time point t , we recover it using

$$(x_t, y_t, d_t) = (p_n, p_{n+(N+1)}, p_{n+2(N+1)}).$$

The objective function is implemented directly as given in (6.1), with the modification that the temperature atlas function takes a path instead of its individual components, i.e. $\mathcal{T}(\mathbf{p})$ for the time-constant case and $\mathcal{T}(\mathbf{p}, \boldsymbol{\tau})$ for the time-varying case.

Implementing constraints

The speed constraint can be implemented as a function $c_{speed} : \mathbb{R}^{3(N+1)} \rightarrow \mathbb{R}^N$, where the speed limit is subtracted from the horizontal Euclidian distance between the previous point on the path and the new. We use the indexing for the augmented path variable vector \mathbf{p} , and get

$$c_{speed}(\mathbf{p}) = \begin{pmatrix} \|(p_1, p_{N+2}) - (p_0, p_{N+1})\|_2 \\ \vdots \\ \|(p_{N+1}, p_{2N+2}) - (p_N, p_{2N+1})\|_2 \end{pmatrix} - \mathbf{1}v_{max}$$

where $\mathbf{1}$ is a vector of suitable size with only ones. If the speed at some point is too large, the produced vector will have a positive value at that point.

The constraint on depth can be skipped if the depth is set directly as the down-sampled DST depth time series instead of being a variable. Otherwise, the constraint can be the restriction of depth

1. to positive real numbers, i.e. $\mathbf{d} \in \mathbb{R}_+^{N+1}$. Then we use a bound constraint, $\mathbf{0} = \mathbf{1}_d \leq \mathbf{d}$. This constraint is relevant no matter which of the following alternatives is chosen.
2. by the depth of the ocean, $d_t \leq \mathcal{D}(x_t, y_t)$ for each time point t . This must be implemented as a non-linear constraint $c_{depth} : \mathbb{R}^{N+1} \rightarrow \mathbb{R}^{N+1}$ returning the amount the variable oversteps the bound at each point in time.

3. by some depth value that is larger than any of the DST observations, using a bound constraint \mathbf{u}_d .
4. restricted to some interval around the DST depth time series. We introduce a tolerance parameter β (which can be made time-varying if desired), and use bound constraints $\min\{\mathbf{D} - \beta, \mathbf{0}\} = \mathbf{l}_d \leq \mathbf{d} \leq \mathbf{u}_d = \mathbf{D} + \beta$. (The minimization for the lower bound is done element-wise.) The tolerance parameter can for instance be based on the variation within the data that has been downsampled to obtain a depth value for each day.

An additional option, which is non-trivial to implement, is to retain the maximum depth observed each day (before downsampling) and require this depth to be possible to attain within a reasonable radius around each point on the solution path. For this to work, a more detailed depth-atlas is required, since the depths given in the ROMS atlas are averages over the whole grid cell.

For the constraints concerning the region of the atlas, we can use upper and lower bounds for the indices of the square region that the atlas provides. For instance, if we use the 20 km gridded atlas, which has observations in a 507×329 matrix, we would take

$$\mathbf{1} = \mathbf{l}_x \leq \mathbf{x} \leq \mathbf{u}_x = 507 \times \mathbf{1}$$

and

$$\mathbf{1} = \mathbf{l}_y \leq \mathbf{y} \leq \mathbf{u}_y = 329 \times \mathbf{1}.$$

In addition, we need bounds for areas of land, where the atlas is not defined. We can use a constraint $c_{region} : \mathbb{R}^{2N+2} \rightarrow \mathbb{R}^{N+1}$. This function returns a vector with a positive number for each point (x_t, y_t) in the path that is outside the boundary, and zero for the other points. For invalid points, the function could simply return a number that is large in magnitude, or the infinity constant, since this will ensure that the constraint function is positive. On the other hand, many optimization methods take advantage of local changes in the constraint functions. This is used to determine the best way to alter variables of the current solution to obtain a feasible solution. To make this possible, we can let the constraint function increase gradually when very close to the boundary, to attain a value greater than 0 when violating the bound.

To sum up, the non-linear inequality constraint can be given as

$$c(\mathbf{p}) = \begin{pmatrix} c_{region}(\mathbf{p}) \\ c_{speed}(\mathbf{p}) \\ c_{depth}(\mathbf{p}) \end{pmatrix} \quad (6.11)$$

and the bound constraints as

$$\mathbf{l} = \begin{pmatrix} \mathbf{l}_x \\ \mathbf{l}_y \\ \mathbf{l}_d \end{pmatrix}, \text{ and } \mathbf{u} = \begin{pmatrix} \mathbf{u}_x \\ \mathbf{u}_y \\ \mathbf{u}_d \end{pmatrix}.$$

(Depending on the choices described above, c_{depth} may be zero or \mathbf{u}_d infinite.)

6.8 Summary

In this chapter, we have formulated the geolocation problem mathematically, and presented how the ROMS atlas is implemented. We have discussed the construction of synthetic examples, and suggested general methods for solving the optimization problem. In the next chapter, the interpolation procedure will be described in detail. Then, attempts will be made to solve the geolocation problem for given test problems and for real data.

Chapter 7

Interpolation and algorithm for geolocalization

This chapter demonstrates the practical interpolation which provides a continuous extension of the temperature atlas. Further, an algorithm for the geolocalization problem is described. The algorithm is run on test instances, giving insight to a wide range of issues concerning the algorithm. Finally, the algorithm is run on true DST data in order to find the geolocation of the cod.

7.1 Temperature atlas interpolation

Numerical methods for solving the optimization problem formulated in the previous chapter as a continuous problem will require the ability to estimate temperature values for any point in the ocean, not just those at the exact layers and points given in the grid. Also, a continuous extension may be needed to find good temperature/depth matches.

7.1.1 Why interpolation?

Integer programming or related approaches can work with only the integer points in the atlas. It would be much simpler to just search these integer points for temperature matches, with a high tolerance to compensate for lack of exact matches. This

has been attempted.

First, interpolation of the atlas was done to obtain a temperature time series associated test paths constructed as in Section 6.4. This is a good way to obtain a series that does not match exactly to points in the atlas. Then, a simple brute-force search method was used to attempt to locate the test path. The result of this was that completely different paths were found. These paths turned out to have much better temperature matches (i.e. much lower objective function value) than the path obtained by rounding off the true test path to the nearest integers in the atlas. Thus, it is very likely that we will need interpolation to avoid excluding true solutions, and to find a larger range of paths with optimal objective function values.

7.1.2 Linear interpolation for depth

The atlas gives temperatures for 30 layers for a grid cell in the ocean that is d_{max} meters deep. We wish to determine the temperature t_d at depth d , between 0 and d_{max} (if $d > d_0$, the ocean is not deep enough to have a temperature at this point). We multiply the s vector by d_{max} , to obtain vector \mathbf{d}_s . Now, there are three possibilities. The desired depth d might be smaller than the 30th element of \mathbf{d}_s , i.e. closer to the surface than the topmost layer. Or, d may be greater than the first element of \mathbf{d}_s , so it is deeper than the deepest layer. In these two cases, we use the temperature for the shallowest/deepest layer directly. A third possibility is that d falls between two elements of \mathbf{d}_s , i.e. between two layers a and b with depths d_a and d_b . Then, we do a linear interpolation between the two temperatures t_a and t_b .

Specifically, we use the formula for a straight line through two points and insert the depth d to obtain

$$t_d = \frac{t_b - t_a}{d_b - d_a}(d - d_a) + t_a$$

for the temperature t_d at d meters.

7.1.3 Thin plate spline interpolation for position

For determining the temperature at a point (x_0, y_0, d_0) which is not at one of the precise coordinates given by the atlas, we use thin plate spline interpolation (see

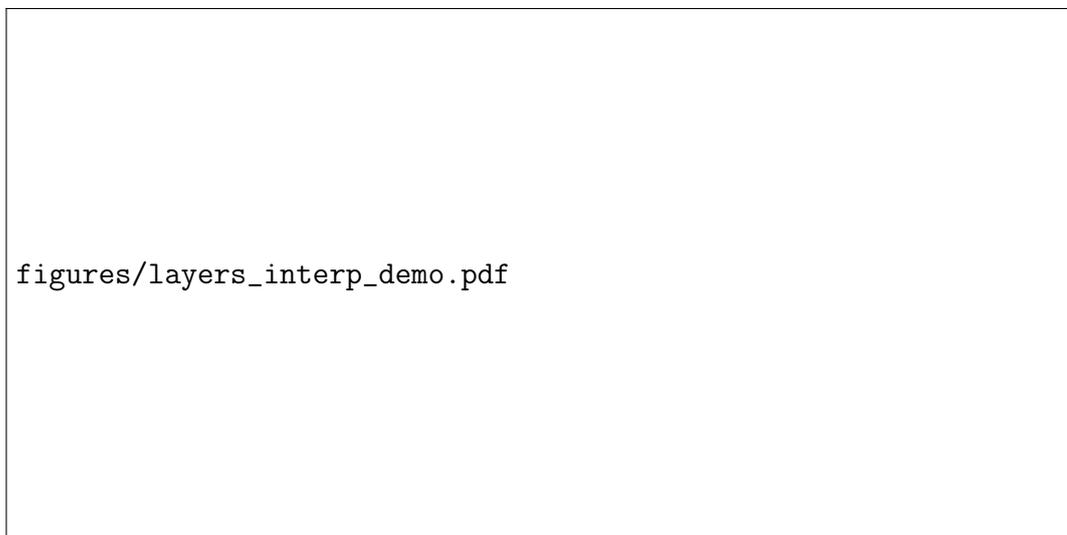


Figure 7.1: Illustration of interpolation of temperature for depth

Section 2.7.1). The computer software we use for this is designed for interpolating function values on points in the xy -plane. It takes in a set of interpolation nodes with corresponding temperature values, and returns a function $f(x, y)$ – a thin plate surface that passes through all the interpolation nodes at the given temperature.

We choose a set of points in the plane that exist as integer points in the atlas, around (x_0, y_0) , and use the linear interpolation described above to obtain the temperature at the right depth. Any points in this set that do not have a temperature defined are excluded from the set. The nodes and function values are given to the software, and the resulting spline is evaluated at (x_0, y_0) .

7.1.4 Atlas subset for testing interpolation

The Lofoten subset of the 20 km gridded temperature atlas on the left in Figure 6.2 will be used to demonstrate the interpolation. The elements of the atlas needed are the five matrices M (the mask), M_λ (longitude), M_ϕ (latitude), $M_{\mathcal{D}}$ (depth) and $M_{\mathcal{T}}$ (temperature). The row and column ranges are numbered $i = 1, \dots, 10$ and $j = 1, \dots, 8$, respectively.

The mask matrix is illustrated in Figure 7.2 on the left, with white cells indicating points in the ocean and black cells indicating land. We view the area provided by the

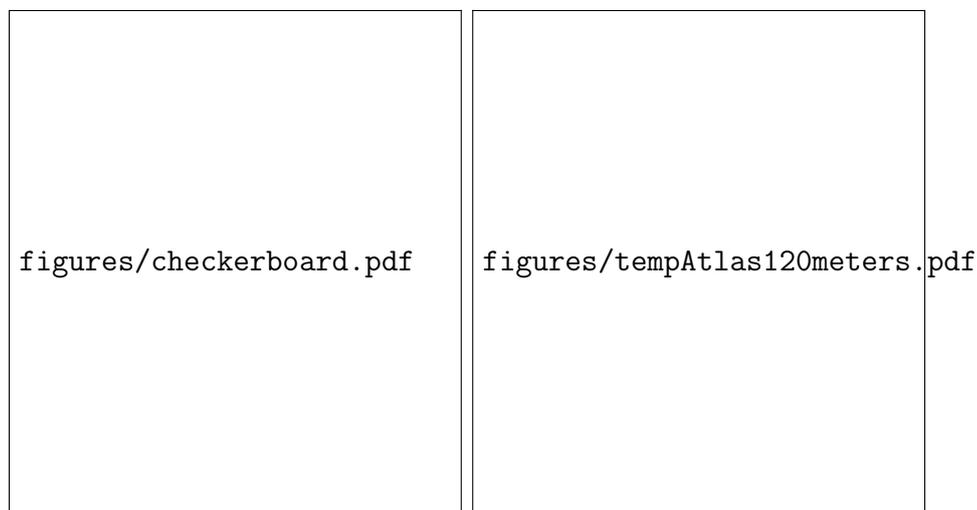


Figure 7.2: Mask and vertically interpolated temperatures at 120 meters.

matrix as a continuous set, and let our geolocation methods search freely through all points, without restriction. The red balls at the centre of each cell indicate what precise position in this set the geographical coordinates in M_λ and M_ϕ correspond to.

The temperature atlas is considered undefined at all points that fall within black cells in the mask. The surface temperature given by the atlas (using only layer 30), with the mask overlaid, was shown in Figure 6.5 in Section 6.3.3. An example where depth interpolation is used is the temperature at 120 meters in Figure 7.2 on the right. Note that grid cells where the atlas shows that the ocean depth is less than 120 meters are also black, and the temperature atlas is considered undefined at these points.

7.1.5 Testing interpolation schemes

We discuss how many neighbouring points should be included as nodes for making the spline. The centre point used for interpolation is the one we get by rounding both coordinates to the closest integer.

We consider eight different sets of points around the centre point. The centre point is in the middle of the innermost grid cell marked by a red rectangle in Figure

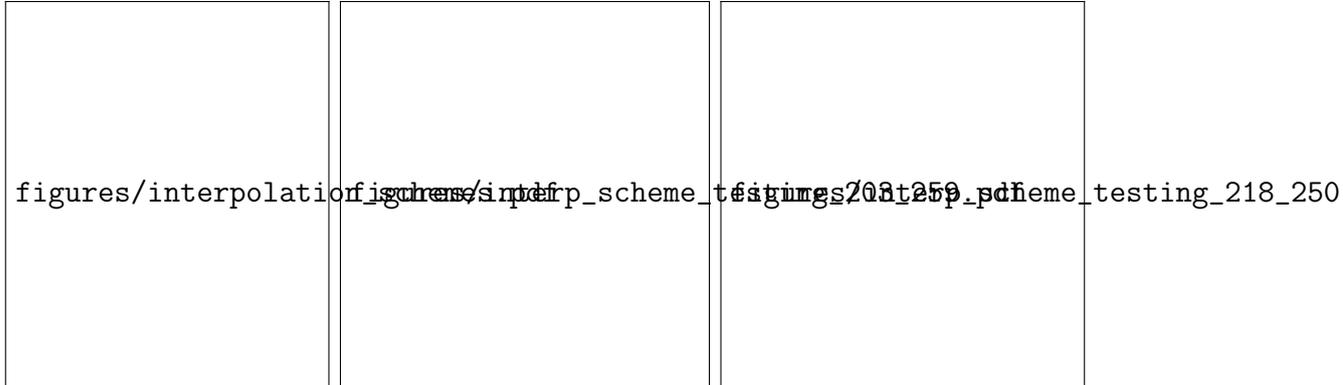


Figure 7.3: Different interpolation schemes and informal error analysis.

7.3, and the interpolation for this centre point is used for obtaining the temperature at any point within the cell

$$\{x, y : 4.5 \leq x < 5.5, 4.5 \leq y < 5.5\}.$$

The point sets are shown with different colours, where the smallest point set of 5 points is blue, and new point sets are obtained by adding more points to the previous set in a symmetric pattern. The eight configurations are 5, 9, 13, 25, 37, 41, 53, 61 and 81 points, respectively, and the orange points are included before the red, giving a “star” shape for the 41 point set.

We now analyze informally the differences that arise when using different sets of interpolation nodes. We hypothesize that the values given by evaluating the spline for the large square of $9 \times 9 = 81$ points is “correct”, in the sense that it is the highest precision we can obtain. We then examine how many points must be included to make the “error” as small as desired.

Starting with a point in the atlas, we see how large the difference is between the interpolated value from using n interpolation nodes and the value from using 81 nodes. We repeat this for many (150 random) points within the same grid cell, such that it is the same sets of node sets that is used. We plot all the “errors”, the worst (over all 150 points) for each node set and the mean error over the node sets. The results of such analysis of interpolation for the surface temperatures within two different grid cells are shown in Figure 7.3. The first cell is near land, such that there

may be undefined points in node set, and the second is in the open sea.

We conclude that using a “star shaped” set of 41 points is definitely sufficient to achieve the needed precision, for the 20 km gridded atlas. Including more nodes than 41 gives a “gain” of order 10^{-3} or less. This is an insignificant gain, compared to the extra processing time involved. A similar analysis has been done for the 4 km gridded atlas, where a very high precision is achieved with as little as 13 points. This may be because the higher resolution means less temperature variation between adjacent cells. On the other hand, the 4 km gridded atlas is much more detailed, including a better representation of eddies [24]. In case this leads to inaccuracies in the interpolation at some points, while not at others (e.g. the ones tested), we “play it safe” and include more points.

Another argument for including more than 13 points in the interpolation of the 4 km atlas is robustness. When moving between adjacent grid cells, the whole set of nodes used shifts one unit. With just 13 points, 5 of the points are replaced when shifting ($\approx 38\%$ of points). With 25 points and 37 points, 5 and 7 points ($\approx 20\%$ of points) are replaced, respectively. When replacing a large proportion of the points, we can risk getting artifacts in the form of discontinuities and non-smoothness in the resulting interpolation at the border between grid cells. The discontinuities may prevent temperature matches in the optimization, and the non-smoothness can cause problems for solvers for non-linear optimization that use derivatives. Thus, we never use less than 25 points in the interpolation when the atlas is used for geolocation with real DST data.

We show the results of using this interpolation scheme for the temperature in the Lofoten area in Figure 7.4. See Appendix C for a discussion on efficiency issues concerning this interpolation.

7.1.6 Consequence of missing values

As mentioned above, whenever one of the 41 points in the interpolation point set is one where the temperature atlas is undefined, the point is simply ignored such that the spline is constructed without a node and function value at that point. For some positions (for instance, at the end of a narrow fjord), this may result in there

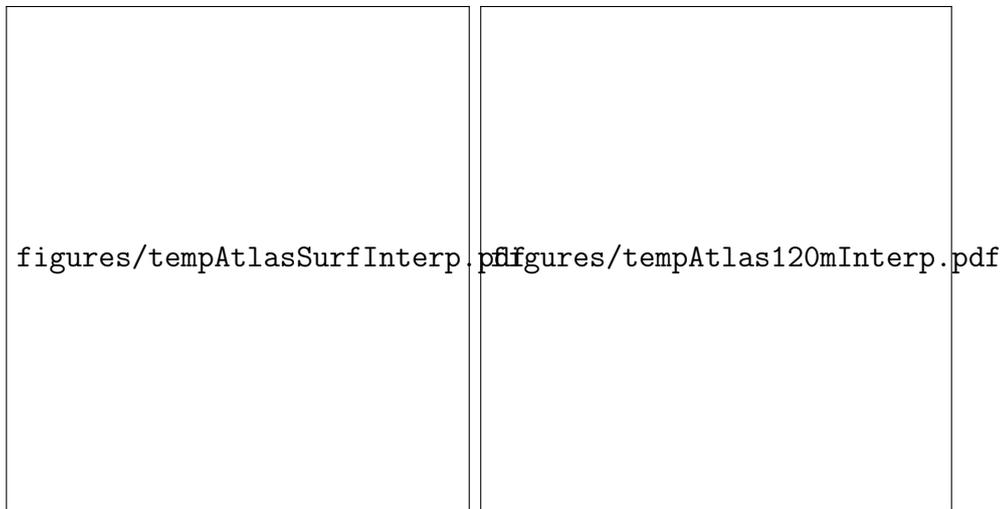


Figure 7.4: Horizontal interpolation of surface temperature and temperature at 120 meters for Lofoten area.

being too few data points to create a spline. This can be handled by letting the temperature atlas be undefined at all points within the grid cell, or by returning a constant value for cell. The first approach is used for the results in this thesis.

The approximate distance for the most remote interpolation node is 5×20 km. For some positions, a body of land may lie in between, making it unrealistic to use the far away point as node for a spline determining the temperature at the current point. We consider this problem to be negligible.

7.2 Test instances

Several test instances of the geolocalization problem have been created using the procedure described in Section 6.4, and have been used for testing and developing algorithms. Two of those that are used for demonstration purposes in this chapter are shown in Figure 7.5 as both temperature/depth time series and as geographical paths. A summary of their properties are given in Table 7.1. For each “synthetic fish”, a length L is given in centimetres, in order to establish a maximum swimming speed v_{max} measured in meters, and ultimately grid units, per day (ref Section 6.6).

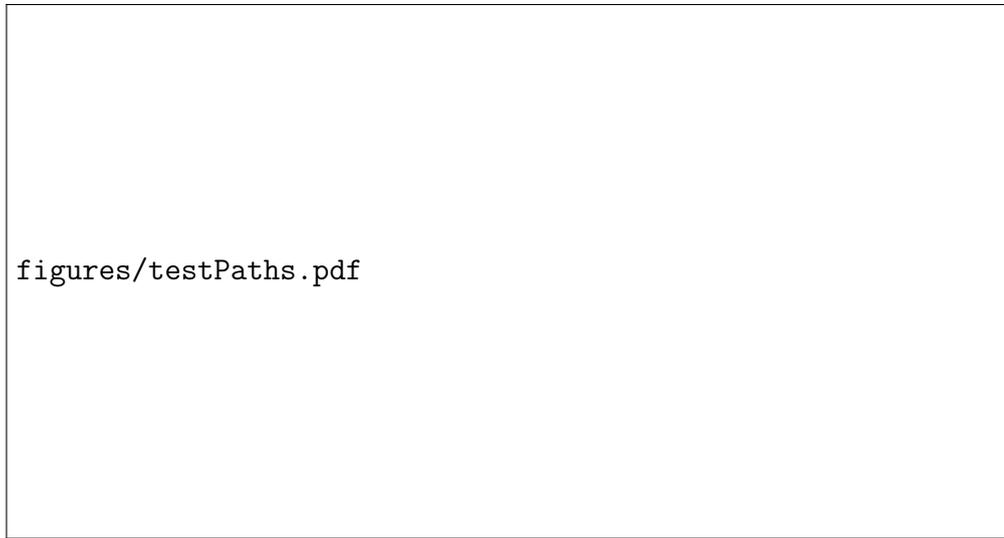


Figure 7.5: Test instances 1 and 2

The formula is then

$$v_{max} = L/100 \text{ cm} \times \lambda \times 60 \text{ seconds} \times 60 \text{ minutes} \times 24 \text{ hours}.$$

The speed measured in grid units is obtained by dividing by the resolution of the atlas. Using $\lambda = 2.0$ and a 100 centimeter long fish, we then get 8.64 grid units for the 20 km atlas and 43.2 for the 4 km gridded atlas. We let $\sigma = v_{max}/3$ for the normal distribution used for generating the daily movements. The depth is set to the true DST depth time series for tag 1664, downsampled using daily averages, to make the depth movements realistic. All test instances use a time-constant atlas. The point of release for all fish is the given point of release for tag 1664, e.g. 67.4°N, 12.6°E, close to Lofoten.

For the temperatures for the test instances for the 20 km gridded atlas, an interpolation scheme of only 13 points was erroneously used, and time has not allowed for creating new temperature time series with more points in the interpolation. Since these are test instances, however, the consequence is considered to be negligible as long as the same interpolation is used for the atlas that the optimization algorithm searches.

Table 7.1: Properties of test instances

Instance	Atlas type	Fish length	Series length
1	20 km	100 cm	100 days
2	20 km	50 cm	100 days
4	4 km	100 cm	30 days

7.3 Solving the problem using standard solvers

Attempts have been made to solve the geolocalization optimization problem for test instances, as a total optimization problem as described in Section 6.7.1, using general solvers and the implementation as in Section 6.7.2. For instance, the `fmincon()` function of MATLAB has been used, with many different varieties of objective function and constraints. The testing has not been sufficiently systematic to be documented properly here. Experience has shown that this approach is not able to provide good matches in the temperature series, and that it has an extremely high CPU running time.

A reasonable explanation is that the interpolated temperature atlas, for one day or time step, is not a convex function. This is due to its nature with possible local maxima and minima at or near interpolation nodes. Then, the total objective function is not convex either. Thus, an optimization algorithm using derivatives, such as active-set and interior point that are used by `fmincon()`, is unable to escape local extrema.

A solver for unconstrained least squares problems (`lsqnonlin()` in MATLAB) and one using a derivative-free search method (`fminsearchcon()`) have been tried, with penalty terms added to the objective to penalize violations of the speed limit. Also, a derivative-free search method for constrained problems (`fminsearchcon()` from MATLAB file central) has been tried, along with experimentation with Particle Swarm Optimization (PSO), a non-deterministic optimization method that can solve non-smooth problems (see e.g. [20]). Methods from the global optimization toolbox of MATLAB were also tried. No significantly better results were obtained for the total optimization problem.

The optimization approach of local optimization of the total function described in Section 6.7.1 is rejected because it is believed to give very slow convergence. The reason for this is that for each point on a candidate solution path, the position is bounded by the speed limit in maximum movement to both the previous and next point in time.

We conclude that general-purpose approaches to optimization are not useful to solve the whole problem, and must instead make a special-purpose algorithm.

7.4 Analysis on one time step

In this section we will study the search for a temperature match in one time step, to aid in the discussion on solution methods for the geolocalization problem. That is, given a release position on day 0, is it possible to find match for the average DST depth and temperature on day 1 within the swimming range of the fish?

7.4.1 A one-step test problem

We consider a test problem where the initial position is $(x_0, y_0) = (218, 250)$ in the 20 km gridded atlas, a point in the open sea between Iceland and Norway. After one day, the synthetic fish has reached position $(x_1, y_1) = (216.9493, 244.2176)$ at a depth of $D_1 = 38.3970$ m, an arbitrarily selected movement within the selected swimming range of 8.64 grid units. The interpolated temperature at this point is $T_1 = 1.9971$ degrees.

We use interpolation both vertically and horizontally to get splines for the temperature at depth D_1 within a box of 24x24 units around the current point. The splines are evaluated on a grid with a spacing of 0.2 units. This gives the plot on the left in Figure 7.6. The swimming range of the fish within a day is indicated by the yellow circle. Then we make a plot on the right in the same figure of the objective (with the depth fixed at D_1)

$$f(x, y; D_1) = |T_1 - \mathcal{T}(x, y, D_1)|.$$



figures/temp_surf_218_250.pdf

Figure 7.6: One step temperature and objective.

In this plot, points on the 0.4 unit grid where the function values are less than 0.1 are indicated by a yellow circle.

7.4.2 Multiplicity of solutions

It can be seen that the target temperature is found within a precision of 0.1 degrees within a long “belt”. Numerical optimization methods can be used to find more exact matches. We run MATLAB’s `fmincon()` with the active-set algorithm, starting at an arbitrarily picked yellow circle within each 1×1 unit box, and use this box as bound constraints. In the case demonstrated, many points with objective function value of less than 10^{-4} were found, which is better than the precision of actual DST measurements. So even though they may have slightly different objective function values, they are equally “correct”.

The conclusion to draw from this is that there are very many positions at the target depth within the daily swimming range of the fish that have the target temperature, including the “true” position of the synthetic fish. A search procedure that starts a new search from all of these positions, at each time step, will quickly have an exponential growth in the number of candidate solution paths. Care must be taken to select only a few solutions to branch on. As a start, only one solution from each unit box is included.

7.5 Suggested algorithm for geolocalization

This section presents a special-purpose optimization algorithm for solving the geolocalization problem, based on the ensemble search alternative described in the previous chapter. An algorithm for this problem must be able to provide results in a reasonable amount of time. It should also be able to estimate uncertainty in solutions, since there may be many solutions having equivalent objective function values. The main idea of the algorithm is to merge solutions that end up at the same point, to reduce the number of branches of solutions paths that the algorithm must maintain. Also, it is hoped that partial solutions that go in the wrong direction will “die out” if they are way off their target, due to inability to find a match for temperature, when staying in the “wrong” area over longer timer intervals.

7.5.1 Data structure for solutions

We introduce a data structure or class which we will call `SolutionPath`. Objects of this class have the following properties, with data type and size indicated:

`<1 × 1 ℕ> length`

`<n × 1 SolutionPath> prevPaths`

`<n × 2 ℝ> endPoints`

`<2 × 1 ℝ> currentEndPoint`

A `SolutionPath` object then contains n endpoints ($n \geq 1$) and a vector of `SolutionPath` objects of one time step shorter length that lead to the current point. A `SolutionPath` object for the initial point is allowed to have no previous paths, and the only endpoint is the initial point. Thus, the object recursively describes the path from a start point to the current position. The current position given by the property `currentEndPoint` is defined by all end-points together; see a later section for details. To keep track of how long a path each object represents, the `length` property is maintained.

7.5.2 Main algorithm

An outline of the suggested procedure is given in Algorithm 4. There are many details left to specify, which will be done in subsequent sections. Also, a more detailed version of the algorithm is given in Appendix B. The result of running the algorithm, if it does not terminate prematurely, is a collection of solutions $\Psi^{(t_{max})}$ containing a number of `SolutionPath` objects of length $t_{max} + 1$. The algorithm uses the depth as a fixed parameter, and it is thus objective function 6.8 that is minimized. Only the speed constraint is used in the algorithm – the ocean constraint has been moved into the objective function in the sense that positions in the ocean with undefined temperature never get temperature matches in each local search.

Algorithm 4 Geolocalization

- 1: **given** a time series $\mathbf{T} = (T_1, T_2, \dots)$, $\mathbf{D} = (D_1, D_2, \dots)$ of temperature and depth and a starting position (x_0, y_0)
 - 2: initialize a set $\Psi^{(0)}$ containing a `SolutionPath` object with the starting point
 - 3: **for** $t \leftarrow 1, \dots, t_{max}$ **do**
 - 4: **initialize** a set $\Psi^{(t)}$ maintaining all solutions at time step t
 - 5: **for each** `SolutionPath` ψ in $\Psi^{(t-1)}$ giving a starting point (x_{t-1}, y_{t-1}) **do**
 - 6: **search** the area within swimming range of (x_{t-1}, y_{t-1}) at depth D_t to find T_t within some tolerance
 - 7: **for each** matching point $(x_t^{(i)}, y_t^{(i)})$ from this search **do**
 - 8: **if** the matching point $(x_t^{(i)}, y_t^{(i)})$ is different enough from all other solutions in $\Psi^{(t)}$ **then**
 - 9: **add** the matching point and the path leading up to it to $\Psi^{(t)}$ as a new `SolutionPath` object
 - 10: **else**
 - 11: **merge** the matching point and path leading up to it with an existing `SolutionPath` object
-

7.5.3 Backtracking solution paths

To obtain the paths that all final solutions represent, each solution must be given to a recursive algorithm that propagates backwards through the solution using the `prevPaths` property. We will call this procedure “backtracking”. This will give increasing number of solutions whenever there is more than one previous path. Some options to handle this are:

- enumerate all paths
- always use e.g. the first previous path
- pick one or a few previous paths randomly

When a day and point of recapture (x_{final}, y_{final}) for the fish is given, we can use this to exclude final `SolutionPath` objects that end up too far away from the recapture point. If all the end points of the final solutions are far away from the recapture point, or if the algorithm has terminated before it has reached the end of the time series, adjustments to the algorithm or parameters must be made to achieve better results.

7.5.4 Details on merging solutions

Line 8 of Algorithm 4 prescribes when to add a matching point as a new solution path, and when to add the end-point and path to an existing solution path. The following details how solutions can be merged.

Recall from Section 7.4.2 that we only considered one solution of each local search within each 1×1 unit box. This can be increased to larger boxes, e.g. 3×3 unit boxes, if necessary to reduce the number of solutions even more. For the first search at time step 1, this is the only way used to limit the number of solutions. Optimization is then used to obtain a “perfect” temperature match within each box, and solutions that do not end up with a close match than some given tolerance can be discarded.

For the searching performed at the second time step and onwards, the method for reducing the number of solutions is usually not sufficient to prevent exponential

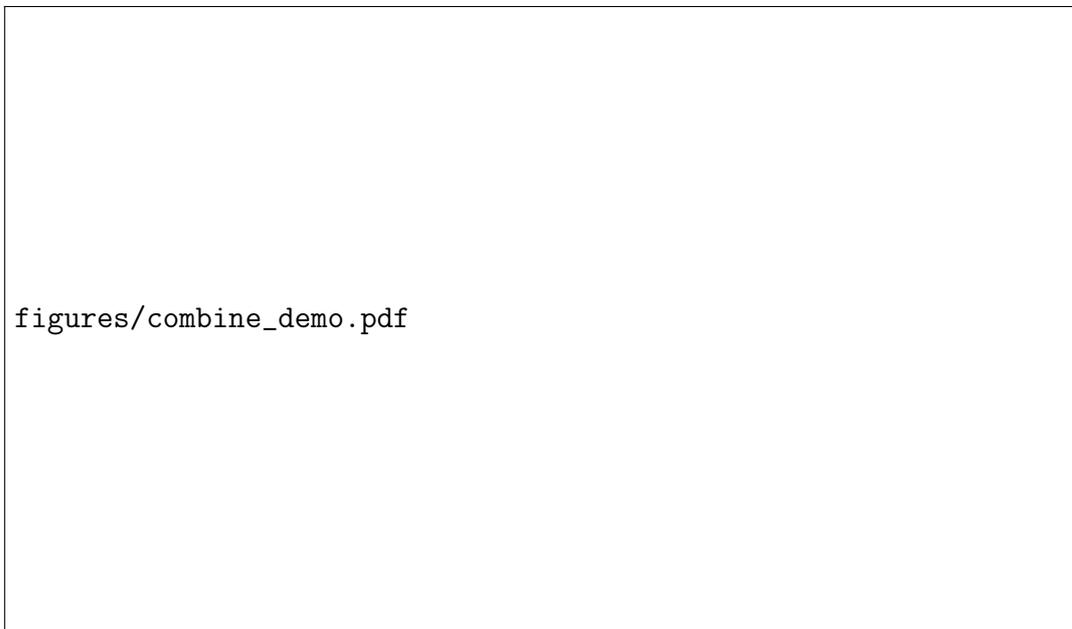


Figure 7.7: Demo of merging solutions

growth in candidate solutions. Then, since there may be different paths “converging” after having moved in different directions at time step 1, there is a point in merging these paths. When a `SolutionPath` is created, it defines a box around the initial end point with some prescribed size. Every candidate end point considered is checked whether it falls within the box of an existing `SolutionPath` object. If so, the endpoint, and the path that leads up to it, are added to the existing solution. After all searching in a time step is finished, the mean of all endpoints of each solution path is used as starting point for an optimization. This optimization searches for a “perfect” match bounded by yet a new box, which is a slightly bigger than the smallest box needed to contain all the endpoints. This optimized end point is stored as the `currentEndPoint` property, and then used as starting point for the search at the next time step. See Figure 7.7 for an example.

It may happen that when starting a search from an endpoint which is made from merging paths, a temperature match is out of range even though it was not out of range from one of the individual endpoints. This may cause premature termination of the algorithm. To prevent this, the swimming range of the fish can be increased

by an amount corresponding to the spread in the endpoints that have been merged. Thus, a solution of the problem at the next time can still be within range.

7.6 Discussion on algorithm

Results from running the algorithm on both the test instances from Section 7.2 and on real DST data are shown in Section 7.7. First, we discuss two important modifications to the algorithm that have proved necessary for efficiency.

7.6.1 Initial coarse search

Experience from testing the algorithm has shown that it is usually not necessary to use interpolation when searching for temperature matches within the one-day swimming distance. It suffices to search the integer points given in the atlas, when allowing for a greater tolerance in the difference to the target temperature. This is also more efficient, since no splines need to be generated at this stage. When all solutions for a time step have been collected, the refining optimization is run within each box defined by all the endpoints, but this time utilizing the interpolation. This is done to check whether there truly is a close temperature match. Solutions that have a resulting objective function value greater than some tolerance can be rejected.

A possible explanation for the fact that only the integer points are needed can be found by appealing to the intermediate-value theorem of calculus. For functions of a single variable, the theorem states the following [1]:

Theorem. *If $f(t)$ is continuous on the interval $[a, b]$ and if s is a number between $f(a)$ and $f(b)$, then there exists a number c in $[a, b]$ such that $f(c) = s$.*

The thin plate splines are continuous functions. A function of one variable $f(t)$ on a closed interval $[a, b]$ is constructed by drawing a line between two neighbouring nodes used for generating the spline, and evaluating the spline at all points on the line. The position of the nodes on the line are a and b with $a < b$. Then, if the target temperature is s , and if $f(a) < s < f(b)$ or $f(b) < s < f(a)$, then by the theorem there will always exist a point c on the line having this target temperature.

Also, since the thin plate splines are “bent” as little as possible away from the values of the interpolation nodes, we can expect the spread in the temperature values at integer points in a local area to be sufficient in determining the spread in interpolated values. The temperature target temperature is thus most likely found within a suitable tolerance if it lies between the temperatures at nearby integer points.

7.6.2 Choice of tolerances

The tolerance setting needed to find intermediate points with the target temperature can be determined by the typical distance in temperature between neighbouring integer points in the atlas. The optimization done at the end of each step can be used to discriminate between

- solutions where the spline surface passes through the level set given by the temperature, indicating an “exact” match, and
- solutions where the closeness in value is due only to the tolerance used in the coarse search itself.

See Figure 7.8 for an example where searching for two different temperatures in an area, using a big tolerance, can lead to both correct and wrong results. A tolerance of 0.4 degrees might be needed to find all integer points where the temperature is close enough to 7 degrees (blue plane), while the same high tolerance setting can also give matches to a search for a temperature of 10.2 degrees (red plane). Clearly, the temperature of 7 degrees is a true match, since the surface passes through the plane, while the red plane for 10.2 degrees does not. That is why the refining optimization at the end of each time step is needed.

One important point to note is that the tolerance should also be considered in light of the uncertainty in DST observations and in the temperature atlas model. Having a smaller tolerance than the uncertainty in data may exclude some valid solutions. However, considering the vast amounts of solution paths that arise from running the algorithm, a small tolerance in the refining optimization may be the only way to discriminate between otherwise equivalent solutions. A heuristic used in the

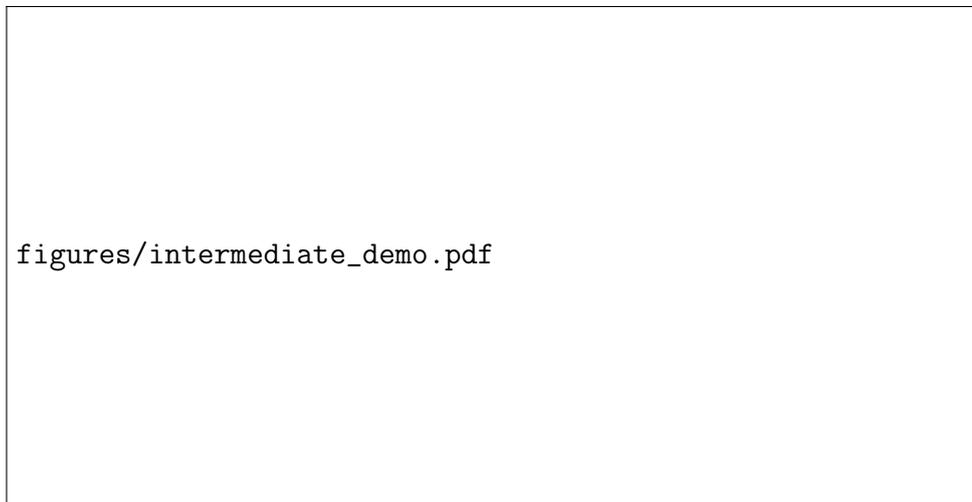


Figure 7.8: Demonstrating the effect of tolerance setting.

algorithm is that the tolerance setting is started at a low level, and increased stepwise only if more points are needed in order to find a sufficient number of matches.

7.6.3 Limiting further the number of solutions

Sometimes, even with the measures described to reduce the number of solutions maintained by the algorithm, the number of solutions may become too high to allow an acceptable running time. It is therefore necessary to introduce an extra measure. When the number of solutions at the end of a time step exceeds a preset threshold, the box around the first end point of each `SolutionPath` object used to merge solutions is increased in size. This new size is used for the following time step. If the number of solutions has fallen below the threshold at the end of the time step, the box size is reduced back to the original size. The two box sizes and the threshold can be chosen by experiments, to find a setting that is a compromise between getting sufficient spread in the solutions and preventing a large growth in the number of solutions.

7.7 Testing the algorithm

The algorithm has been run on the test instances that were presented in Section 7.2. The results shown in this section have been selected to demonstrate a range of situations that may occur, and illustrate how changing settings in the algorithm and the method of backtracking can affect results.

Test instances are used in the algorithm with both 100 days, and with only the first 30 days of 100 day instances. It is assumed that the fish is recaptured on the final day, and that the recapture coordinates are known. Unless otherwise specified, the fish length in the algorithm is set equal to the synthetic fish that has generated the data. In the figures in this section, many solution paths are shown together with the mask matrix, giving a distorted map. Each path has a different colour. The colours are chosen arbitrary within each plot, but the temperature plots have the same colour as the corresponding solution paths.

For most solution paths, the temperature matches are so close to that of test instances that they are not shown, and the function value is very close to 0. Unless otherwise specified, all backtracking has been done randomly, picking only one previous path at each time step. When running the algorithm, it has been necessary to experiment with the different parameters. For full details on which parameters are available, see the complete description of the algorithm in Appendix B.

7.7.1 Too low tolerance can rule out true solutions

When the tolerance for accepting solutions after the “refining optimization” is very small, it may happen that solutions that are close to the original test instance are ruled out. An example of this is shown in Figure 7.9, where the paths of running test instance 2 for 100 days is shown along with the temperatures for each path. Note that the temperatures match so closely that the individual temperature plots can not be seen. Even though the maximum swimming distance per day was set to 86.4 km, and the maximum movement in one day in the instance was 83.0 km, at some point the algorithm “lost track” of the fish. Then it started going in the wrong direction and still did not die out in the course of the 100-day run. The tolerance



Figure 7.9: Solutions from running on test instance 2 with too low tolerance.

was 10^{-4} degrees, which is far too small, especially compared to the precision in the true DST data. This low tolerance settings has, however, been applied successfully on the other two test problems.

7.7.2 Backtracking only on solutions with good endpoints

Test instance 2 was run for 100 days with a much higher tolerance than before, in the order of 0.5 degrees, and all final solution paths are shown in Figure 7.10. Note that some individual temperature plots are slightly different so that they can be seen. The small differences in objective function value could be used to evaluate each solution. However, it turns out that these function values are very small, and that they are not at all related to the closeness to the true path \mathbf{x} . This closeness has been measured for each final solution path $\mathbf{x}^{(i)}$ by computing the Euclidian distance from the true path, e.g.

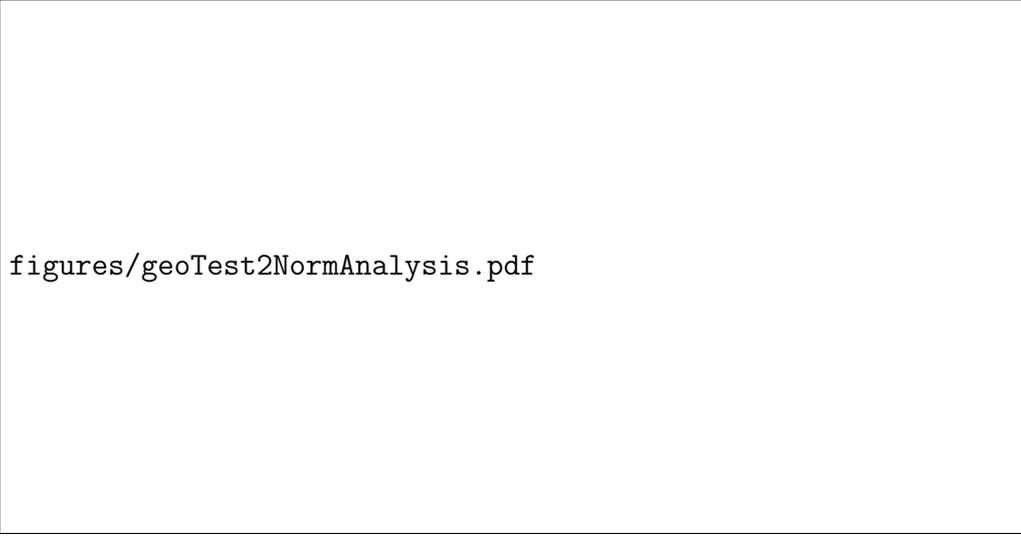
$$\text{dist}(\mathbf{x}, \mathbf{x}^{(i)}) = \left\| \mathbf{x} - \mathbf{x}^{(i)} \right\|_2,$$

where solution paths in two dimensions have been turned into one vector before taking the norm. In Figure 7.11, these distances are shown in blue, in an increasing order. For each solution path, the corresponding objective function value is shown in



figures/geoTest2AllSolutions100.pdf

Figure 7.10: All solutions from running test instance 2 with higher tolerance.



figures/geoTest2NormAnalysis.pdf

Figure 7.11: Analysis of objective function value of solutions to test instance 2.

green. There is no apparent link between function value and closeness to the original paths, and thus all the solutions can be considered equivalent when considering function value alone. It seems that the solutions can be classified into three groups – ending around Iceland, ending near the coast of Møre og Romsdal or ending near Troms. This is advantageous – it means that instead of a seemingly random spread

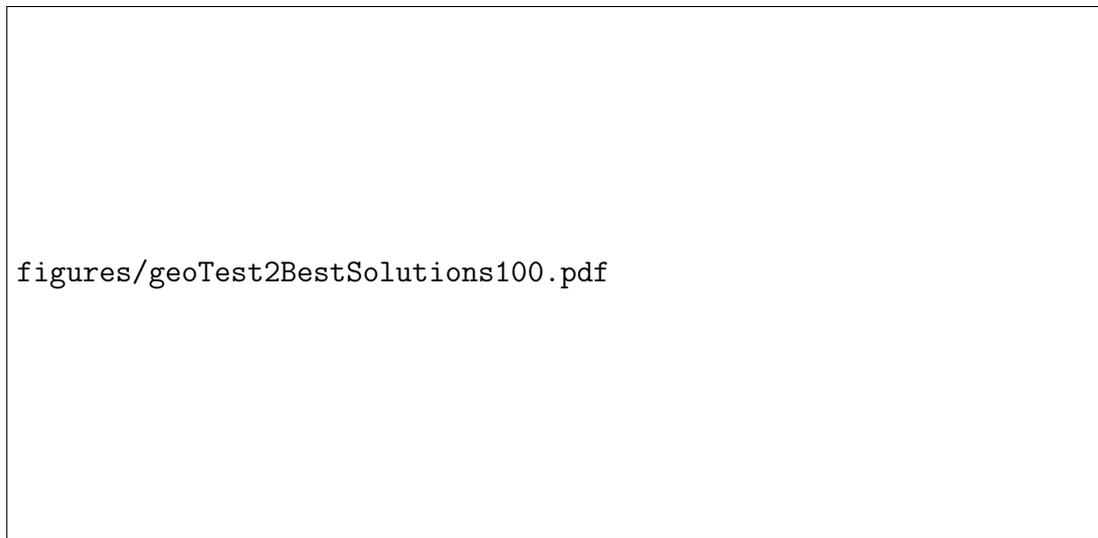


Figure 7.12: Solutions to test instance 2 with best end point matches.

in solutions, there is a clear cut criteria to separate them. A new backtracking where only the solutions with endpoints that are near Troms, corresponding to the recapture position, are shown in Figure 7.12. For the most part, the algorithm has done a very good job in approximately recovering the path.

7.7.3 Wrong solutions die out

While the algorithm propagates forward in time, it is likely to work on solutions that turn out to die out before the final time step. For test instance 1, we show in Figure 7.13 how the algorithm after 30 days has three solutions approaching Spitsbergen, and that these branches of the solution tree are not present after 100 days. Note that they may also be missing as a result of the random branching – this possibility has not been explored.

7.7.4 Large speed limit can give more solutions

Test instance 2, constructed with a 50 cm long synthetic fish, was run through the algorithm with the fish length set to 100 cm and with the low tolerance 10^{-4} degrees as previously. See Figure 7.14 for the results. The spread in final solutions



Figure 7.13: Solutions at 30 and 100 days for test instance 1.

was much larger than in the previous run, with excursions to England, Denmark and Greenland. Interestingly, the algorithm did not terminate prematurely due to the low tolerance as it did in Section 7.7.1. Only one solution remained that was close to the true path in position. We conclude that using a speed limit that is too large can increase the spread in solutions. The extra solutions can be ruled out by e.g. recapture position. However, unnecessary spread in solutions increases the chance that the number of solutions maintained exceeds the threshold for triggering an increase of the size of the box used for merging solutions. In effect, the best solution paths can in this way be affected by other solution paths that are far away.

7.7.5 Developing a good backtracking heuristic

So far we have only used random backtracking. This means that whenever there is more than one path leading up to a point on a final solution path, rerunning the backtracking procedure may result in paths that are closer to the true path. See an example of this for a run of test instance 4 for 30 days in Figure 7.15. In the left plot is shown the result of a random backtracking on all the final paths that have end points closest to the true end point. All paths, including those that end up nearest to the true recapture position, move very far away from the true path. In the right



Figure 7.14: Solutions from running test instance 2 with too high speed limit.

plot, a different path leading up to the endpoint of pale blue path on the left is shown in red. This path is much more accurate, but in a situation with actual DST data this would not be known. This indicates that a backtracking heuristic should be chosen that emphasizes a spread in the final solution paths. Alternatively, the backtracking could enumerate all paths leading up to each end point. This can be used to make a mean path, with uncertainty bounds given by the spread of all the points for each day.

7.7.6 Pruning solutions

Biological and geographical information may be available that can be used to rule out solution paths that existing knowledge indicates must be wrong. As seen in Figure 7.15, the run on test instance 4 gave out many false solutions that were difficult to distinguish. For instance, if we knew that this fish was a Norwegian Coastal cod, we would know that the paths going far into the Barents sea and back are wrong. Then the user running the program could manually prune such solutions every few steps.

Another solution is to modify the mask matrix M to make temperature data available at all points outside the known habitat of the species. An example is shown in Figure 7.16, where the habitat for test instance 4 is assumed to be the area

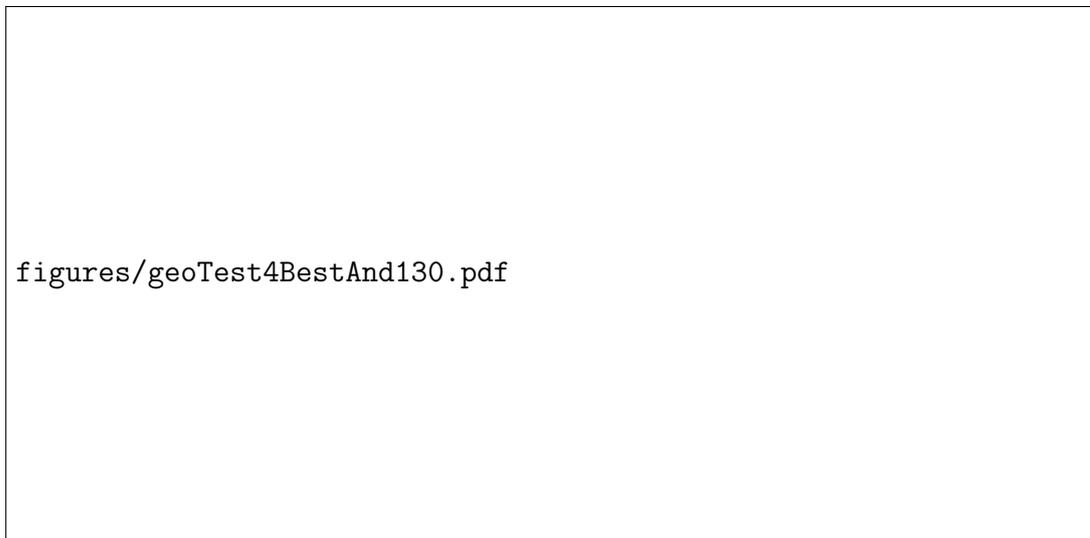


Figure 7.15: Demonstration of how backtracking matters for test instance 4.

shown. On the left are all solutions, and on the right are those with best matching end points. There are still many equivalent solutions that are way off target, but it is possible that running the algorithm for a longer time than the 30 days shown here may cause the solutions near the boundaries of the habitat to die out. This habitat restriction can also be time-varying – for instance using knowledge of what time of year a fish species is known to never inhabit an area. An advantage with this approach is that the number of maintained solutions may be smaller, reducing the chance that the box size for merging is increased (ref. Section 7.6.3).

The approach of habitat restriction should be used carefully, because if too heavy restrictions are made on the habitat, the results may become realistic because of this restriction alone.

7.7.7 Too small speed limit terminates the algorithm

When too small speed limits are set, compared to the test instance, the algorithm may terminate early. Test instance 1, made with a 100 cm long synthetic fish, was run through the algorithm with the fish length set to 50 centimetres (results are not shown). All solutions died out by 47 time steps. The algorithm did, however, manage to track the approximate true solutions for most of the time. The fact that

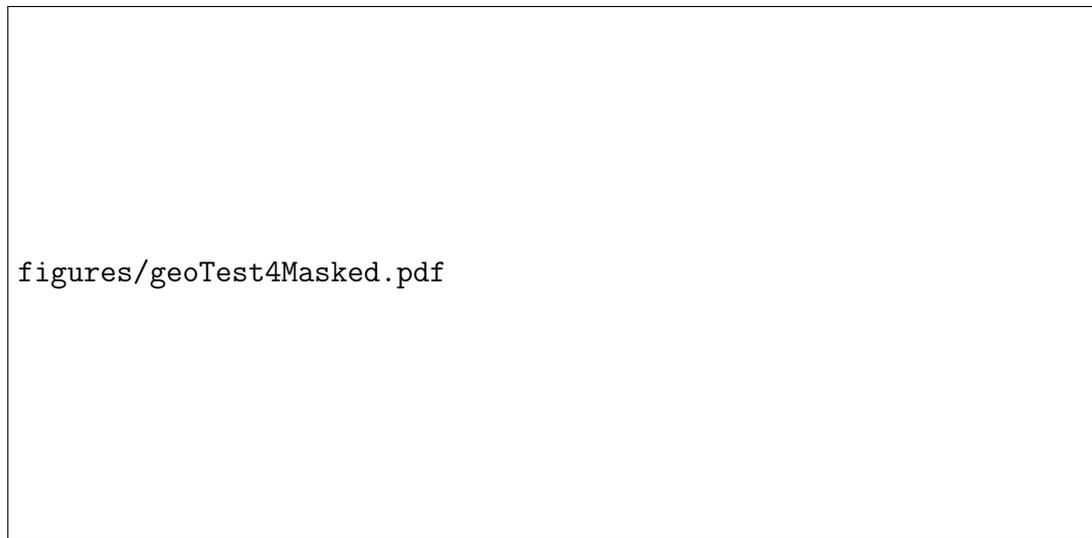


Figure 7.16: Using habitat restriction to reduce spread in solutions for test instance 4.

the algorithm did terminate prematurely is a confirmation that the algorithm works as intended – it never gives solutions that have poor temperature match. Also, it confirms that there is sufficient variation in the temperature in the sea that some of the paths that start out in the wrong direction do die out.

7.8 Results of geolocalization on real data

The geolocalization algorithm has been run on real data with a time-varying atlas for the period in question. Results from this will be presented in the following. No temperature plots are displayed, because all solutions are practically equivalent when it comes to temperature matching. All runs of the algorithm produced so many good solutions, that it was found sufficient to keep a low maximum speed limit. An arbitrary setting of $\lambda = 2/3$ was used, i.e. for the 89 cm long fish of tag 1664 a maximum horizontal displacement of 51.3 km was allowed. Errors arising in approximate conversion to grid units were not corrected. For all runs, the tolerance setting for temperature matching after refining optimization was set to 0.1 degrees.

For the first run (Run 1), there was a lot of spread in the solutions. In Figure 7.17, the resulting paths after 30 days is shown on the left. The land shown is part of

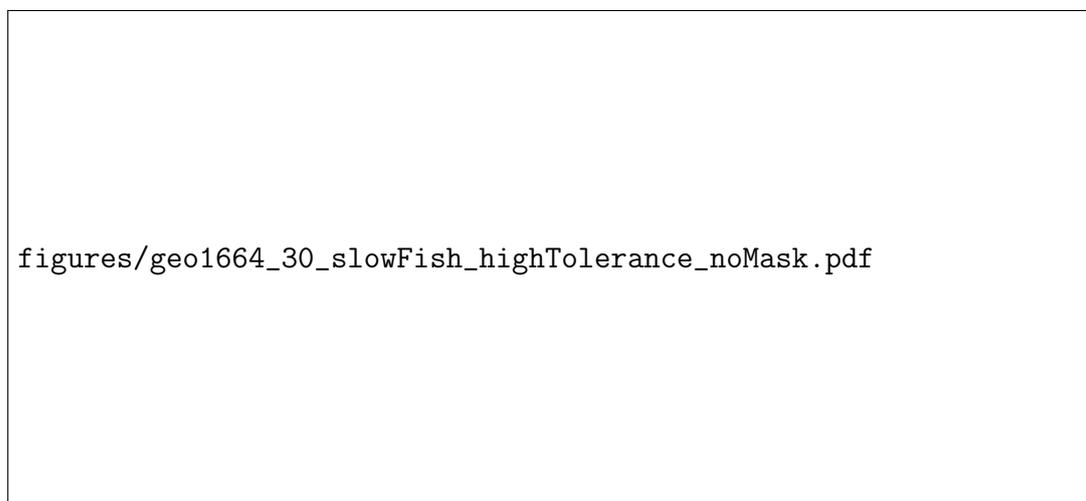


Figure 7.17: Run 1 on DST data, 30 days and a selection of solutions

Northern Norway, though distorted. A selection of those paths that did not venture far out into the open sea are shown on the right. A possibility could be to only allow these paths remain in the pool of solutions. The algorithm was allowed to run further with all solutions retained, for up to 120 days, and the results are shown in Figure 7.18. As we see, for the last 60 days there are many paths that hug the north Norwegian coast tightly, but few that do so the whole time. Also, there are many that travel all the way to Spitsbergen, which is very improbable for this stock.

To both reduce processing time and to let the algorithm focus on solutions in areas that are close to the known habitat, a new run (Run 2) was started where the mask matrix was adjusted to not allow movement outside the square area shown in Figures 7.19 and 7.20. The square was selected very subjectively, and a proper implementation should use more exact masking that is not a square. The figures show the solutions after 30 days, 60 days, 200 days and 400 days. Notice how the paths almost cover the visible part of the Norwegian sea, showing how very many and different paths there are with close temperature matches. Note also how some paths occasionally cross land. If the path to go around e.g. a peninsula is within the daily swimming range this is okay, but otherwise this may be a problem that should be tackled in a future implementation of the model.

The time series for tag 1664 ends seven months before the recapture date. Thus,

figures/geo1664_60_120_slowFish_highTolerance_noMask.pdf

Figure 7.18: Run 1 on DST data, 60 and 120 days

figures/geo1664_30_60_slowFish_highTolerance_masked.pdf

Figure 7.19: Run 2 on DST data with habitat masking, 30 and 60 days

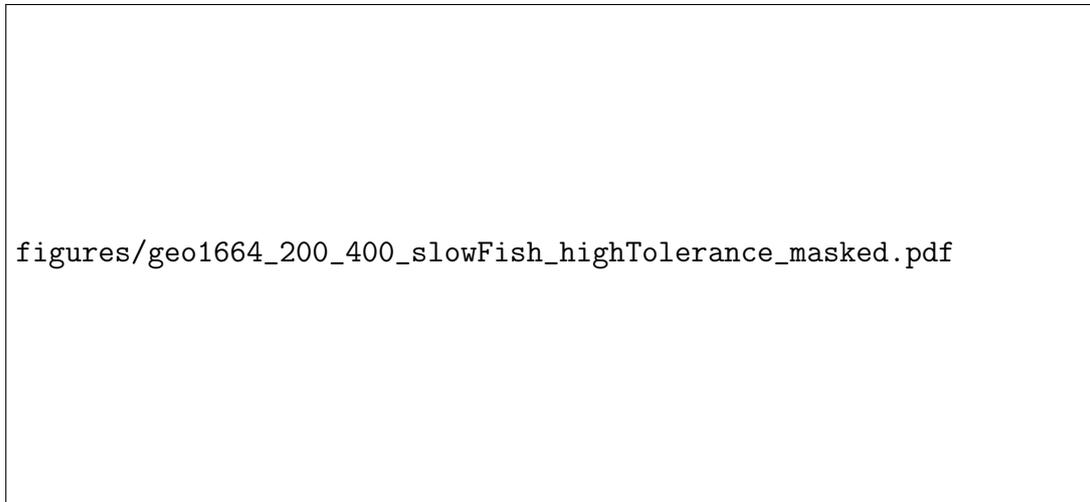


Figure 7.20: Run 2 on DST data with habitat masking, 400 and 200 days

there is no potential for using the recapture position for selecting the best solutions. Also, since the time series is very long (≈ 776 days), the fish could swim very large distances in this time. Thus the utilization of the recapture position would not affect much of the path anyway. Use of recapture position could be more useful for shorter time series.

To conclude, we show in Figure 7.21, in a real map with projection, the shortest and longest paths, respectively, from run 2 after 200 days. Note how similar the two paths are for the first 100 days, but not for the next 100 days. The shortest path is approximately 6648 km long, i.e. an average travelling per day of ≈ 33 km per day, and the longest path is approximately 7700 kilometers long, meaning an average of ≈ 38.5 km travelling per day. The average speed can be used as a criteria for determining the realism of the paths.

7.9 Discussion

Several aspects of the algorithm have been discussed already, and ideas for improving the algorithm are presented in the chapter on future work. Some drawbacks of the algorithm are now discussed.

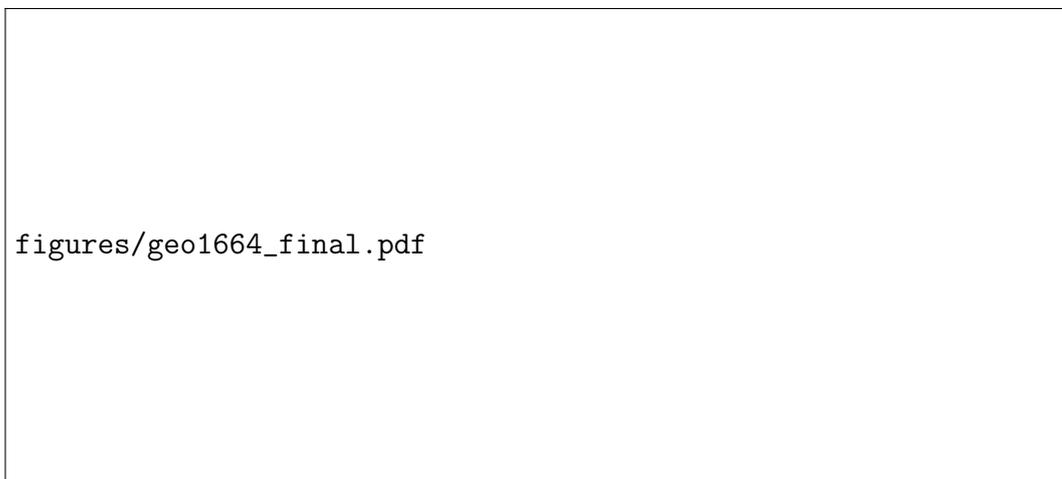


Figure 7.21: Shortest and longest paths from run 2 at 200 days, shown in map

7.9.1 Drawbacks of the algorithm

No search in depth direction

One major drawback of the algorithm is that it does not search in the depth direction. The (downsampled) depth value from DST observations is used directly for obtaining the temperatures for each node for the spline at each time point. This may lead to temperatures not being found, since the depth values have a degree of uncertainty (see Section 4.1). It is hoped, however, that the lack of flexibility in vertical direction is somewhat compensated by flexibility in the horizontal direction, since it is possible that the temperature can be found at other points nearby, but at the fixed depth.

Infeasible solutions

When merging solution paths, the travel distance between points on a path can easily become too large. Also, measured in true geographical distances as analyzed in Section 6.3.1, the violation of the speed constraint may be even larger. See the chapter on future work for a suggestion on solving this issue.

Sensitivity to solution order

The method of merging solutions relies on the position of the first path that ends in an area. Thus, the placement of the box that captures nearby solutions depends on “who gets there first”. Ideally, there should not be such a dependence on the order of paths. However, this thesis does not address this issue.

7.9.2 Concluding remarks

The focus while developing the algorithm has been to make a method that works – not one that can be easily analysed with respect to correctness, predictability or convergence rate. Its main aim is to prioritize spread in solutions while preventing exponential growth.

A lesson learned in the work on the geolocalization problem has been that there is considerable ambiguity in solutions. That is, many radically different paths may give a near-perfect match in temperature values, even over longer time periods. Solution paths very close to a known, optimal test path can have worse objective function value than other paths far away.

It is uplifting that an optimization problem that seemed unsolvable using general-purpose solvers could be solved to approximate, known optimality (perfect temperature match to within precision) by the algorithm presented here. For the geolocalization of true fish, it is neither possible nor desirable to determine the *exact* path, but many approaches have been suggested here that can reduce the number of possible solutions. Also, one can look at in a different way, namely that the points on the solution paths (with careful selection) can be used to express where the fish might experience the depths and temperatures it prefers. This means that even though the one particular fish did not inhabit an area, another fish with similar preferences could have.

The methods suggested here can be combined with geolocalization using other methods and environmental factors, for instance on fish that in the North sea that have already been geolocalized. Each approach may give large but hopefully different spread in solutions. Those solutions that overlap between different methods are more

likely to be closer to the true geolocation of the real fish.

For fish in the Lofoten-Barents sea ecosystem, where tidal and light based methods are less relevant, the only comparable method known to the author is the one given in [58] using biased random walks. It would be interesting to try both methods on the exact same data and atlases. This requires data from a tag with a known recapture position at the end of the time series. It is unknown, however, how the biased random walk method will work when the release and recapture positions are close to each other. It is possible that the fish has travelled far away and back again during the time period in question, while the method seems to rely on the attraction towards the recapture position. A consequence of this may be that the random walk approach shows less spread in solutions than the optimization approach, since solution paths that venture far away from the release and recapture coordinates are less likely to occur.

Part III

Suggestions for future work

Chapter 8

Future work

This chapter continues some of the discussions in previous chapters, and presents suggestions for future work. This comprises possible improvements on the methodologies used and ideas for applications of the fish migration model.

8.1 More on the stochastic state model

In Chapter 5.2.2, the use of a Markov chain for modelling the behavioural state for a fish was rejected for the results in this thesis. We now present some further considerations of the stochastic state model.

8.1.1 Suitability of stochastic state model for parts of data

Many parts of the depth data are characterized by long periods of movements around the same concentration points, and thus long-lasting visits to behavioural states. Other parts of data, however, show a rapid and seemingly random switching between states. See e.g. the October 2004 results in Table 5.2, and compare to the very long visit to a states in the May 2004 data in Table 5.1. It is possible that the data can be characterized as exhibiting either a mostly deterministic behaviour or a mostly random behaviour, and that the model can be specified in the same way. The inference and simulation process, with all the improvements specified in Chapter 5, has been tried on October 2004 depth data using the stochastic state model, and

the model fit turned out to be very good (the key was the improved estimation of b).

8.1.2 Continuous-time Markov chains

The cited references on mixed OU models use continuous-time Markov chains for controlling the state instead of discrete-time Markov chains. It is possible that the dynamics are captured better with continuous-time chains. For determining the Q-matrix of the chain, the method in [42] can be used. Another alternative is to use e.g. optimization to determine the Q-matrices from the transition matrices estimated in this thesis.

8.2 Extending the model to several fish

It is desirable to use the 2-D fish migration model, using DST data from many fish, to obtain knowledge on fish on stock levels. This is called *upscaling*. Since the actual point in time in which different fish exhibit the same types of behaviour are unlikely to be the same, we require that the stochastic state model is used.

8.2.1 Model for many fish

To perform upscaling, one can make numerous simulations of fish behaviour and analyse how the stock migrates on the whole. Three suggestions for how to do this are presented here.

Many simulations using the original model

One way of simulating many fish is to determine sets of concentration points and states for each fish that DST data is available for, and make many simulations for each fish. However, this limits us to the time of data capture for each fish. Also, if further improvements to the stochastic state version of the model are not made, one must use the deterministic model. Then, all the simulated fish based on the same tag data will be very similar.

Inference on all tags at once

It is preferable to be able to express the whole range of variation between fish when upscaling. A suggestion on how to achieve this is to run the inference process on many fish from the same area and time period together. Concentration points can be determined from a kernel density estimate of aggregated data. Care must be taken to choose time periods where all or most of the fish experience similar ranges of behaviour. Fish that show very different behaviour from the rest can be excluded, or modelled separately. Markov chain transition matrices can be estimated through counting transitions in each series, and aggregating.

Stochastic concentration points

Another suggestion for determining concentration points from many fish at once is to start by determining them individually. Then, those from different fish that are close in position to each other can be grouped, such that all the points in a group are associated with the same state or states. The concentration point used as attraction point in each separate OU diffusion can then be a realization of a stochastic variable with a distribution based on the spread in the concentration points belonging to the state.

8.2.2 Systematization of inference process

For doing inference for the migration model on many fish over many time periods, some of the manual input must be replaced by automated procedures.

Smoothing

The smoothing of noise should be done without requiring user input. An alternative to the ADPF smoothing could be adaptive-window polynomial filters [9]. This could also lead to better smoothing. Or, the correlated residuals can be accepted, and these can be modelled using e.g. autoregressive moving average processes.

2-D persistence

For the selection of kernel density estimation bandwidth and for choosing the size and shape of neighbourhoods, an extension of the Persist algorithm to bivariate time series would be useful. Such an extension could identify boxes in the plane, instead of intervals, that correspond to persisting states.

Kernel density estimation bandwidth

The bandwidth for kernel density estimation should be adjusted automatically. We do not address how this can be done for 2-D data. A suggestion for the 1-D model is the following. Use a standard bandwidth setting and obtain Persist bin boundaries. While there are Persist bins with more than one peak, increase the bandwidth by a small number.

Heuristic for discarding KDE peaks

The peaks in the KDE are discarded if they are less than 10 % in height compared to the highest peak. This choice has been arbitrary. A possible heuristic that also uses the time aspect in the data is to discard any peak that corresponds to a persisting state that lasts a very short portion of the whole time series. The Persist algorithm already has a lower threshold that specifies that 5 % of time must be spent in a state for it to be persisting, but this could be set higher.

8.2.3 Systematization of validation

Each inference using a slice of fish tag data should be validated, before used as basis for simulations when upscaling. The validation of the model must thus be more systematic. For these amounts of data, it is undesirable to visually assess e.g. how KDE curves match. It is also highly subjective.

Kernel density estimate validation

The difference in area, for the 1-D model, or in volume, for the 2-D model, between kernel density estimates of simulations and of data can be used to quantify model

fit.

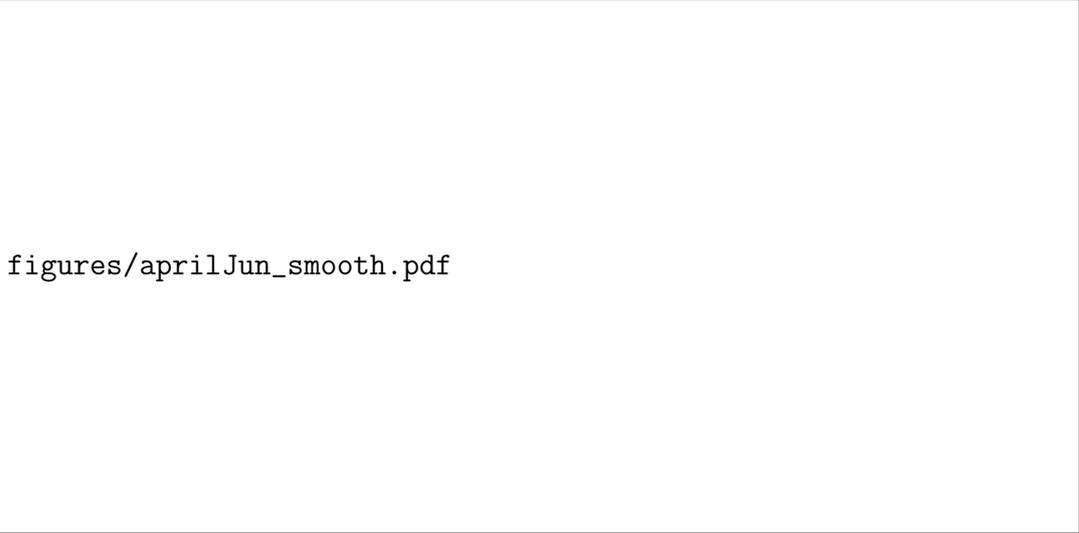
Check Persist bins of simulations

The time aspect of model fit can be measured using the Persist algorithm. If model fit is good, the Persist algorithm should return similar Persist bins for data and for simulations. The difference in position and in number of bins can be used to make a number that measures the model fit.

8.2.4 Pre-smoothing

In Section 5.3.1, short-lasting visits to states were simply removed. Another alternative, which is useful when using a stochastic state model, is smoothing the data before inference is done. The unsmoothed data can be retained for a new smoothing within each state, since it allows for individual window settings at that stage. We suggest a moving average filter.

With moving averages applied to the April-July depth data, an interesting effect of this smoothing appears. We get the modified inference results as shown in Figure 8.1. Now the state set is different, because the state $\{3 \rightarrow 1\}$ is no longer observed. This presents two advantages. First, the shortest lasting visits have disappeared (only two, longer-lasting, visits to state $\{2 \rightarrow 1\}$ remain towards the end of the time series). Second, the peaks in the KDE plot are much narrower, making the peaks in the KDE more defined, such that concentration points are easier to find. For instance, a concentration point at 140 meters is now found as a peak in the KDE (consistent with the existence of a Persist bin for this area), whereas before, the data within this area was too spread out to give a peak in the KDE graph. Actually, this shows that for this particular case, more features are brought out of the data when using smoothing at this stage. It should be noted, however, that for months with behaviour such as in October 2004 with high oscillation between states, this pre-smoothing will not be beneficial since it removes the alternating tendency.



figures/aprilJun_smooth.pdf

Figure 8.1: Inference on data from April-July with presmoothing

8.3 Other choices for parameter estimation

Systematic validation of model fit for other fish may reveal that the parameter estimation methods used are not adequate. Some further ideas are given here.

8.3.1 Drift term parameter

We consider an alternative for estimating the drift term parameter b , that allows for different parameters for temperature and depth in the 2-D model. This is obtained by considering a different choice of the λ parameter of Section 5.4.2. Note that for data from an OU process with $\mu = 0$, we have, for the expected value of an observation given a previous,

$$\mathbb{E}(X_{t+h}|X_t = x_0) = x_0 e^{-bh} \quad (8.1)$$

where h is the time that separates two observations. We can find observations for the point in time that the neighbourhood of μ_i is left (the time t_k of state change to state $\pi_k = \{i \rightarrow j\}$) and the point in time that the neighbourhood of μ_j is entered ($t_k + \hat{h}_{ij}$), which we will denote $x_{leave(i)}$ and $x_{enter(j)}$. Thus we get, when solving (8.1)

for b with these observations inserted,

$$b = \frac{1}{\hat{h}_{ij}} \ln \left(\frac{x_{leave(i)}}{x_{enter(j)}} \right)$$

so we can set

$$\lambda = \ln \left(\frac{x_{leave(i)}}{x_{enter(j)}} \right). \quad (8.2)$$

Estimating λ in this way from several visits to a state can be done by taking a mean of those estimated from each visit to the state. If the two values have different sign for a visit (which is possible for temperature), the logarithm is undefined and the method cannot be used for this visit. The extension to non-zero μ is straightforward by substitution.

8.3.2 Diffusion term parameter

The diffusion term parameter c in Chapter 5.4 relies on estimating the long-term variance of what is assumed to be a realization of an OU diffusion process. The method used was to consider all data after the first entering of the destination neighbourhood as “equilibrium data”. Since the neighbourhood size is chosen rather ad hoc, a different alternative is considered. For a sufficiently long visit to a state π_i , the variance $\tilde{\sigma}_j$ of all data starting from one observation x_j during the visit to the end of the visit can be computed. This is then the variance for each of

$$\{x_k, \dots, x_{leave(\pi_i)}\}, k = t_{enter(\pi_i)}, \dots, t_{leave(\pi_i)} - 1.$$

Plotting this variance, a goal is to find a point in the curve from where the variance has become “stable”, e.g. that it does not change very much when points are removed or added. This stable variance can be used as the long-term variance for estimating c .

8.4 Movement to the far side of a concentration point

It has been observed that occasionally, the fish travels so fast through the water column that it can pass from one side of the neighbourhood of a concentration

point to the other in less time than the sampling interval of 10 minutes. This may lead to state changes not being captured, or recorded too late, and through this to bias in the transition matrix (or for the deterministic state model, bias in the state sequence). Also, depending on the OU process parameter estimation technique, it can introduce bias into the parameters for the state as well (for instance, an extra amount of variability is attributed to the wrong state). A solution is to let any movement *past* the two most extreme concentration points (smallest or largest in value, in depth or temperature), count as a state change. For 2-D data, the analogue involves modifying the neighbourhoods of any concentration points near the edges of the region of the data by stretching them in the extreme direction.

8.5 Improvements to geolocalization algorithm

This section presents some suggestions for improving the algorithm, that should be considered for future implementations.

- When nearing the end of the time series, i.e. the date of recapture, solutions paths that are unlikely to be able to reach the destination area in time, due to restrictions on maximum swimming speed, can be excluded from further search.
- The algorithm can be run backwards in time, from the point of recapture, and then attempts can be made to merge solutions with those from running the algorithm forwards, at the time halfway between release and recapture day. This is similar to an approach used in [58].
- Each local search for temperature matches can be started with a low maximum travelling distance compared to the absolute maximum swimming speed of the fish. Then, the search radius can be extended only if necessary to find the target temperature, stepwise up to the established maximum limit. Also, if the algorithm runs very low on solutions, or even terminates prematurely, the algorithm can go back a few time steps and increase the search radius used, such that new solution paths that “survive” longer can be found.

- The algorithm can be made to adapt itself strategically, by continuously tuning parameters (e.g. distance for merging solutions) to maintain an approximate same number of solutions at all times. Also, all merging can be deactivated if there are very few solutions.
- Interpolating three-dimensional data directly, using some method for 3-D interpolation instead of the two-stage interpolation used in this thesis, might make it easier for algorithms to search for temperature matches also in the depth plane.

8.6 Incorporating prior knowledge

In Section 7.7.6, some ideas were presented for using existing knowledge on fish behaviour to rule out solutions. Some more possibilities for incorporating prior knowledge on oceanography and fish behaviour are:

- identifying tidal signals in the DST data (as in e.g. [50] and [37]), and comparing them to known information on tides at points in the solution paths. If the tidal signal observed is impossible to observe in an area, we can rule out such a solution path.
- when a DST depth recording shows only an identified tidal signal for a longer period of time (e.g. several hours), this is indicative of the fish staying at the seabed [37]. Then, solutions paths where the ocean is too deep compared to the observed depth can be ruled out.
- if a solution path shows that the fish swims very far north, at a time in the winter when there is little daylight in the area, and a diurnal rhythm is still observed in the DST data (after correcting for any tidal effects), we can assume that the solution path is unlikely.

8.7 Correcting infeasible solution

A solution for correcting the problem with infeasible solutions returned from the geolocalization algorithm is now discussed. A final solution that violates the speed limit can be turned over to a standard optimization routine such as MATLABs `fmincon()`, to modify, if possible, certain parts of the path violating the speed constraint, while still maintaining a good objective function value. This has been done on test problems with success. See Appendix C for more on how this was implemented efficiently. The speed constraint can be implemented to check distances using the haversine formula, made possible by converting grid unit coordinates to geographical coordinates e.g. by interpolating the M_λ and M_ϕ matrices.

8.8 Particle swarm optimization for geolocalization

Experimentation with particle swarm optimization has revealed that this is a very powerful optimization method, even though attempts did not provide satisfactory results for the geolocalization problem. The method may well be capable of finding global optima of complicated optimization problems, particularly when it is implemented to use social sub-swarms. The main challenge in applying this method to the geolocalization method is the implementation of constraints – particles that leave the feasible set must be re-initialized at a feasible point. This is not straight-forward to implement.

One advantage with this method is that it should be simple to modify the algorithm such that all processing for each time step is done before proceeding to the next. This would make it compatible with the way the temperature atlas is organized. Another beneficial possibility of this approach is that the different social groups may converge on different solution paths, which has the potential of expressing the spread in solution paths.

8.9 Considering uncertainties

Before using the algorithm for geolocalization to do real analyses, uncertainties and possible errors in the approach should be considered further. Some issues are discussed here.

8.9.1 Daily averaging

The averaging of the DST temperature is done over a whole day, which may include both vertical and horizontal movement. The temperature atlas, however, provides values that are averaged over a day, but for a constant position. This can make it harder to find temperature matches, or cause a bias in the estimated horizontal direction that compensates for the two types of averaging. The algorithm's sensitivity to this drawback can be investigated by finding confidence bounds for the daily averages. This bound must both consider the errors in individual measurements and the errors in estimating the daily mean from discretely observed data. Whenever a search fails to find a target temperature, the depth and temperatures for each extreme bound (i.e. four combinations) can be used instead of the daily average itself. Another source of errors in the daily average is that we have not investigated what time zones have been used in the atlas and in the DST.

Another consequence of the averaging is that the maximum observed depth during the day is not used. A possibility is to keep an extra times series of maximum depth per day, and exclude solution paths where the sea is not deep enough in one day's travelling range to visit this maximum depth.

8.9.2 Uncertainty in atlas

In this thesis, the uncertainty in the hydrodynamic model generating the temperature atlas has not been considered. This should be done, in order to decide what tolerance levels should be used for temperature matches.

8.9.3 Expressing uncertainty in solutions

To express uncertainty in the solutions, many or all of the paths obtained by backtracking from each final solution can be used to compute a mean and standard deviation for each point on the path. It should be noted that we cannot expect such a mean path to have an optimal function value. To do this, a heuristic is needed to identify paths that are radically different and paths that are similar enough to be averaged into a single path.

The algorithm or backtracking can be modified in order to promote as much spread in solutions as possible, in order to attempt to find the entire range of possible paths with good temperature matches.

8.10 Validating the geolocalization algorithm

Before using the geolocalization approach in this paper, some external information should be used to confirm that the method is trustworthy. One way of doing this is to create synthetic data using a real data storage tag. For instance, a DST could be attached to a device at the end of a line pulled by a boat, generating both a known geographical path and a temperature/depth time series. The algorithm can then be tested for its ability to recover this path using a temperature atlas for the time period in question.

Another alternative for validation is to perform geolocalization using several methods. DST data from e.g. North Sea fish that has already been geolocalized can be used, and the results from many methods can be compared. More confidence can be placed in estimated paths that arise from using several different approaches for geolocalization.

8.11 Geolocalization by migration model

An application of the fish migration model of Part I is to perform geolocalization of simulations instead of data directly. Either, one can use a time step of 1 day, or one downsample simulations to one day. This is a useful test of whether the

fish migration model generates depth and temperature data that is adequate for geolocalization. A further possibility is to run geolocalization on many simulations made using an extension of the model to several fish, to geolocalize a whole stock of simulated fish. This can be used for making inferences on stock level.

8.12 Geolocalization by other factors

The algorithm should be straightforward to modify for performing geolocalization using other environmental variables than temperature.

- For instance, a recent technological development is geomagnetic tags, that measure the Earth's magnetic field. Mathematical models of the magnetic field can be used in place of the temperature atlas in the algorithm.
- Some DSTs record salinity, and hydrodynamic models can be used to generate salinity atlases.
- Many factors can be used simultaneously in the geolocalization, by formulating an objective function that promotes matches to the all the different factors.

Appendix A

Article published at NIK

The article attached in this appendix was written as part of the thesis work, and was presented at the conference Norsk informatikkoneferanse 2011, November 21-23 in Tromsø, Norway [31].

Modelling vertical fish migration using mixed Ornstein-Uhlenbeck processes

Erik Natvig*, Sam Subbey†

Abstract

Based on vertical movement data derived from electronic storage tags (DST) attached to fish, we construct a stochastic model that aims at capturing the main characteristics of the observations over one year. We use a mixed Ornstein-Uhlenbeck process to model attraction to a limited number of concentration points on the depth axis. A methodology for determining states and transition probabilities between them, and for setting model parameters for the process, is discussed. We show some examples of simulations using the model, and compare the simulations to the original data.

In general, the model appears to capture the main characteristics of the vertical dynamics, except in cases where the data is characterized by a long-lasting transient state at the start of the time series.

1 Introduction

Electronic data storage tags (DSTs) attached to fish have the potential of providing long term, high resolution observation data on individual fish behaviour. DSTs typically record values of depth and ambient temperature (and occasionally, salinity) sampled at prescribed time frequencies (e.g., every 10 minutes). The release-recapture period for an individual fish may range from a few months to a couple of years in some exceptionally few cases.

Thus, DSTs provide a large volume of data and depth of knowledge about individual fish movement. A major drawback, however, has been the lack of adequate methodology to efficiently handle the large amount of data recorded by DSTs. The vertical movement, for instance, may vary significantly in frequency and magnitude, as well as between months, seasons and years. This characteristic of the data poses a challenge to classical data analysis methodologies. Further, whereas information on an individual fish may be attractive, using such information as a basis for inference on fish behaviour at stock (population) level is more desirable. Upscaling of the dynamics of a single fish, extracted from DST analysis, to stock levels, has not been addressed in the literature.

This article presents a stochastic modelling approach that seeks to mimic the vertical movement of fish as seen in an example data set, assuming that the fish makes transitions between several *behavioural states*. The aim is to capture the main features of water column usage for a single fish such as depth-homing, which might encapsulate several responses. The next step in future work is to extend the model to several other data

*Master student, Department of Informatics, University of Bergen, P.O. Box 7803, 5020 Bergen, Norway. E-mail: erik.natvig@student.uib.no. URL: <http://eriknatvig.net/>

†Institute of Marine Research, P.O. Box 1870, Nordnes, 5817 Bergen, Norway. E-mail: samuels@imr.no. URL: <http://uncertainty.imr.no/>

This paper was presented at the NIK-2011 conference; see <http://www.nik.no/>.

series for fish tagged and released either in the same period or in the same location in the ecosystem. The ultimate aim is to extend inference on individual fish to population levels, based on the variability in model parameters obtained by analysis of large sets of observations.

Section 2 gives an overview of the methodological approach, assumptions, and presents some related work. Section 3 presents the theory needed for the modelling and simulation procedures, while in Section 4 we suggest a method for determining Markov states for the vertical dynamics, and formulation of the associated transition matrix. Section 5 presents the data and inference results, and shows examples of simulations for a given number of months. Section 6 concludes the paper, considers validity and suggests points for improvement of the model. All methods have been implemented in MATLAB.

2 Overview and related work

Within short time intervals, one can assume that the individual fish moves according to a single diffusion (Ornstein-Uhlenbeck) process. However, the depth data encapsulates switches between different behavioural and physiological processes, which include, among others, feeding, swimming, passive tidal transport and spawning. Thus the vertical movement, over a long time period, involves switches between several states representing the varying behavioural/physiological processes, and thus a mixed diffusion process. We will need the following definitions in the rest of the paper:

Definition 1. A *concentration point* is a unique point in the space of the data being considered, with a high density of actual data points around it. Concentration points will be denoted by μ_i . The space of the data will be partitioned into subsets which contain one concentration point each, and the neighbourhood of a concentration point will be defined as a symmetric region around the point.

Definition 2. A (behavioural) *state* is uniquely defined by the combination of the concentration point the fish was attracted to last and the one the fish is currently attracted to, i.e. a state with movement from μ_i to μ_j is denoted by $\pi_{ij} = \{i \rightarrow j\}$.

The idea is to model the transitions between behavioural states as a discrete-time Markov chain S_t , and model the fish trajectories using a *mixed Ornstein-Uhlenbeck* (OU) process with parameters determined by the state of the Markov chain. The OU process is a mean-reverting process, such that it can model random movement but with attraction towards a fixed point μ , and alternating the behavioural states makes it suitable for capturing the data characteristics mentioned above.

Simulations generated using the model we create will be compared to the original data, to see if important characteristics have been captured. This poses four challenges which will be addressed:

- Determining which state a given observation belongs to.
- Forming a transition matrix for a discrete-time Markov chain controlling the behavioural state.
- Simulating the mixed OU process that mimics the behaviour of the fish.
- A metric for comparing simulations to observed data.

Related work

Bivariate Ornstein-Uhlenbeck models are used for modelling wildlife movement in Dunn and Gipson [2], Preisler et al. [7] and Blackwell [1], though only Blackwell actually used a *mixed* OU process and a set of several behavioural states (corresponding to resting, feeding and travelling for wood mice). The procedure we use in this paper is directly inspired by the work on modelling movements of mobile phone users by Rosenblum [8].

3 Modelling and simulation theory

Markov Chains

A *Markov stochastic process* is special in that the outcome/change in the system after the passage of some time is dependent *only* on the state of the system before the passage of time. This means that a Markov process is without memory – wherever the state has been before the previous point does not matter (see [6]). A Markov *chain* is a stochastic process with a countable set of states. Transition probabilities for finite-state discrete-time Markov chain can be given by a *transition matrix* \mathbf{P} whose entries are the probabilities for transitions from each state to all others in one time step. A state of a Markov chain is called *transient* if there is zero probability for the chain of returning to it once the state has been left once, and *recurrent* otherwise.

Ornstein-Uhlenbeck process

A simple model for attraction towards a concentration point μ is the Ornstein-Uhlenbeck (OU) process (originally used to describe Brownian motion in physics, see [10] and [4]). It is defined as the stochastic process $X(t)$ which satisfies the stochastic differential equation (SDE)

$$dX(t) = b(\mu - X(t)) dt + c dW(t) \quad (1)$$

where $dW(t)$ denotes an infinitesimal increment of a Wiener or white-noise process and b and c are parameters. Using the relevant expressions from [4], and substitution for the case when $\mu \neq 0$, we get the long-term expectation and variance of the process, with $X(0) = x_0$ as initial condition:

$$E[X(t)] = \mu + (x_0 - \mu)e^{-bt} \rightarrow \mu \text{ as } t \rightarrow \infty, \quad (2a)$$

$$\text{Var}(X(t)) = \frac{c^2}{2b} \left(1 - e^{-2bt}\right) \rightarrow \frac{c^2}{2b} \text{ as } t \rightarrow \infty. \quad (2b)$$

This clearly shows that the process is attracted towards μ . The *drift term parameter* b determines the intensity of attraction. The second term (diffusion term) provides the randomness in the process. A first-order approximation for use in simulating this SDE is the updating formula from [4]:

$$X(t + \Delta t) \approx X(t) + b(\mu - X(t)) \Delta t + cn\sqrt{\Delta t}, \quad (3)$$

where n is a sample value of a standard normally distributed variable N ($N \sim \mathcal{N}(0, 1)$).

Mixed Ornstein-Uhlenbeck process

To model the attraction of the fish towards concentration points, we follow the idea in [8], and use a *mixed Ornstein-Uhlenbeck* (OU) process. Let X_t be the stochastic process that is a solution of the stochastic differential equation (1) above that governs the movement of the fish in the model, with a time index t running from 1 to T . Next, assume that there are m different concentration points on the depth axis, with labels μ_1, \dots, μ_m , that the fish may be attracted to.

We further assume that a discrete-time Markov chain S_t can model which concentration point the fish is attracted to at any time step. The state-space of S_t is denoted Π , and for each ordered pair of concentration points μ_i and μ_j we define a state of the Markov chain as $\pi_k = \{i \rightarrow j\}$, representing movement from μ_i to μ_j . The reason for including one state for each *pair* of concentration points, instead of just one for each, is that we might want to model different variabilities and intensities of attraction towards the concentration point depending on where the fish came from. For instance, diving down in the sea might be quicker than moving towards the surface or vice versa, so the model should be able to capture that by varying the drift term coefficient b between states (although in practice this possibility will not be explored in this paper).

Now, for each state $\pi_k \in \Pi$, we define a set of parameters for the OU process: $\{\mu^{(k)}, b^{(k)}, c^{(k)}\}$. In the model we let X_t follow a OU process with parameters given by the state of the Markov chain S_t . When the chain transitions to a new state, the new parameters are used instead of the old, so that we get a new OU process starting in the end point of the previous process.

The parameters for the OU process for each state of the Markov chain, along with the selection of concentration points, must be estimated using the data collected from actual fish. In the following sections we suggest a method for this.

4 Determining states

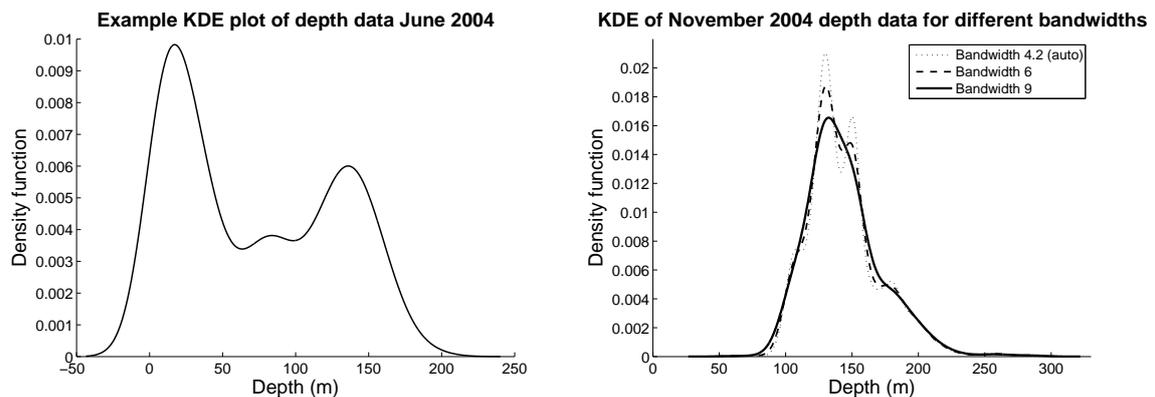
Kernel density estimation

Kernel density estimation (KDE) is a non-parametric method used to estimate an unknown probability density function from n data points assumed to be realizations of n independent and identically distributed random variables. A kernel is a symmetric function that integrates to one (usually taken to be the standard normal density function). By placing a kernel at each data point and summing them up, dividing by the number of data points, we get a new function that integrates to one and has peaks at regions with high density of data points. Formally, the estimated function \hat{f}_h of the data d is

$$\hat{f}_h(d) = \frac{1}{nh} \sum_{i=1}^n K\left(\frac{d-d_i}{h}\right), \quad (4)$$

where h is a smoothing parameter known as the bandwidth, $K()$ is the kernel and d_i are the observed data points. The bandwidth that will be used by default is one optimal for estimating normal densities, but we will see that it might need to be adjusted to reveal the main features of interest.

Figure 1: Kernel density estimate examples



A kernel density estimate for an example time series is shown in Figure 1 on the left. Observe that the estimate has three peaks that can be considered as concentration points. The next step involves defining neighbourhoods around such points.

Persistence analysis

Common methods for discretizing continuous values in a dataset into discrete symbolic values, such as the histogram, do not take the time aspect into account. They may also place cuts in regions of the data with high density. The temporal structure, i.e. the order in which observations occur in a time series, is valuable information that should be used. The Persist Algorithm [5] uses this information to find cuts such that the resulting symbolic values, or states, are persisting. This means that for a point in time there is high probability to observe the same symbol as in the previous point in time, when using the discretization given by the algorithm. The bins given by the algorithm often coincide with peaks in the KDE plot, such that each peak has its own bin.

Combining persistence analysis and kernel density estimation

Choosing concentration points

We wish to determine, for a slice $\tilde{X}_t, t = 1, \dots, T$ of the time series (we will be using whole months), which concentration points to use in the model. The following method is used:

1. Run the Persist algorithm in order to find bin boundaries for the data.
2. Make KDE plot and find local maxima (peaks in the graph).
3. Adjust KDE bandwidth if needed:
 - (a) Some of the local maxima may lie too close to each other (rule of thumb: less than 10 meters apart), or have too low value on the y-axis compared to other local maxima, so we consider smoothing the plot by increasing the bandwidth manually until the KDE has desirable properties. This can be improved in future work by adjusting the bandwidth more systematically. See Figure 1 on the right for an example of bandwidth adjustment.
 - (b) One should also check that the bin boundaries from Persist are such that each concentration point has its own bin – e.g. if there are two peaks in one bin, the bandwidth should probably be increased so that the two peaks join into one peak.
4. Discard peaks that are less than some percentage in height compared to the highest peak. This paper uses a cut-off of 10%. However, this value is arbitrary. Future work will seek to develop a more heuristic approach to determining a reasonable cut-off value.
5. Store the m remaining peaks as concentration points μ_1, \dots, μ_m and define a possible state of the Markov chain for each ordered pair of points to define the set of states Π .

Neighbourhoods and states

Having defined the state $\pi_k = \{i \rightarrow j\}$, we define the *neighbourhood* around μ_j as the interval $(\mu_j - n\sigma_j, \mu_j + n\sigma_j)$, where σ_j is the standard deviation of the data in the same bin as the concentration point, and $n = 1$ (for the moment). Define a number k for each state in Π , then a mapping $\Psi(i, j)$ from the numbering of the concentration points to the numbering of the states, such that if state $\{i \rightarrow j\}$ corresponds to state π_k , then $\Psi(i, j) = k$.

We define two vectors $\boldsymbol{\phi}$ and $\boldsymbol{\tau}$, which store origin and destination concentration points, respectively, for each data point. We use the following algorithm:

```

1:  $\boldsymbol{\phi} \leftarrow \mathbf{0}$  {zero vector of length  $T$ }
2:  $\boldsymbol{\tau} \leftarrow \mathbf{0}$ 
3: for all  $\tilde{X}_t$  do
4:   if  $\mu_i - n\sigma_i \leq \tilde{X}_t \leq \mu_i + n\sigma_i$  for some concentration point  $\mu_i$  then
5:      $\tau_t \leftarrow i$ 
6:   end if
7: end for
8: if  $\tau_T = 0$  then {take special care of the end of time series}
9:   find the last time-point  $t_{last}$  such that  $\tau_{t_{last}} \neq 0$ 
10:   $\tau_t \leftarrow \tau_{t_{last}}$  for  $t = t_{last} + 1, \dots, T$ 
11: end if
12: for all  $1 \leq t \leq T-1$  do
13:   if  $\tau_{T-t} = 0$  then {attraction to next point visited}
14:      $\tau_{T-t} \leftarrow \tau_{T-t+1}$ 
15:   end if
16: end for
17:  $\phi_1 \leftarrow \tau_1$ 
18: for  $2 \leq t \leq T$  do
19:   if  $\tau_t \neq \tau_{t-1}$  then {attraction to new point, i.e. state change}
20:      $\phi_t \leftarrow \tau_{t-1}$ 
21:   else {same attraction as before}
22:      $\phi_t \leftarrow \phi_{t-1}$ 
23:   end if
24: end for
25:  $\tilde{S}_t \leftarrow \Psi(\phi_t, \tau_t)$  for  $t = 1, \dots, T$ 

```

Even though the process assigns to each point \tilde{X}_t a state \tilde{S}_t , there will be unencountered states. We discard unencountered states and renumber those that remain, so that we get a new set of states Π . Note that the first data point, and those that follow until a new attraction is observed, are assigned a special state $\pi_{\text{initial}} = \{i \rightarrow i\}$ with the same destination and origin concentration point μ_i . This state is transient by construction, since once it has been left there will always be a previously visited concentration point, different from the one currently being sought.

Counting transitions and forming the transition matrix

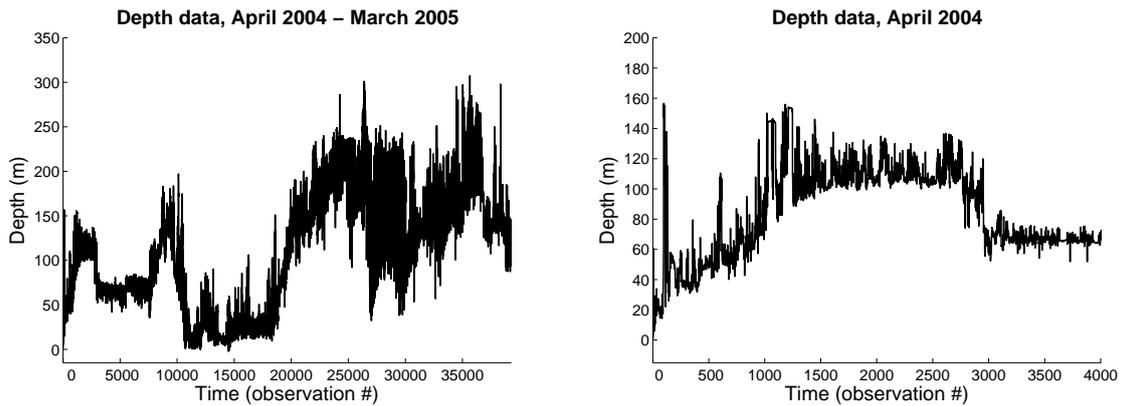
A consistent estimator for the elements of the transition matrix \mathbf{P} for a finite-state discrete-time Markov chain S_t is, according to [6],

$$\hat{p}_{ij} = \left(\frac{\sum_{n=0}^{N-1} \mathbf{1}_{\{S_n=i, S_{n+1}=j\}}}{\sum_{n=0}^{N-1} \mathbf{1}_{\{S_n=i\}}} \right), \quad (5)$$

which is the proportion of all transitions from state i that go to state j . Here, $\mathbf{1}_{\{\cdot\}}$ is the indicator function which is equal to 1 if the subscripted expression in braces is true, and 0 if it is false. As N tends to infinity, \hat{p}_{ij} will tend to p_{ij} with probability 1, by the strong law of large numbers.

We now consider \tilde{S}_t , our time series of states, as a sample path of a discrete-time Markov chain S_t . We wish to estimate the $|\Pi| \times |\Pi|$ transition matrix \mathbf{P} for this chain. In order to do this, we summarize the chain by a matrix \mathbf{M} with entries m_{kl} counting transitions from π_k to π_l . Next, we form a matrix $\hat{\mathbf{P}}$ which has the normalized rows of \mathbf{M} , so that each entry \hat{p}_{kl} of $\hat{\mathbf{P}}$ has the proportion of all transitions from π_k that go to π_l , according to the estimator (5) above.

Figure 2: Example time series, the whole year April 2004 - March 2005 (left) and April 2004 (right)



5 Data and problem description

In order to test the methodology introduced in the previous section, we use the first year of the depth data for a fish (cod) that was tagged and released in April 2004 and recaptured in May 2006. The data is sampled at 10 minute intervals. This sampling frequency is sufficient, and actually far too high, considering the ambient environment. The dominant tidal frequencies are clustered around 6 and 24 hour periods. It has been established in the literature that fish may use tidal transport for horizontal/vertical migration (see [11] and [3]). Given this, it makes sense to expect that the vertical dynamics would encapsulate signals with periodicity 6 and 12 hours. Further, it is natural to expect diurnal periodic behaviour due to the influence of sunlight (see [9]).

In Figure 2 we show on the left the whole time series, while on the right we show the data for April 2004. A visual inspection shows that there are basically three or four regions on the depth axis with a high concentration of data points. In this section we will determine concentration points and transition matrices for this data, present simulations using these and compare the simulations to the original data.

Example of inference results

Running the procedure for determining states and transition matrices on this data gives us the results shown in Table 1. The left-hand figure shows the (flipped) kernel density estimate with the concentration points as lines, and the neighbourhoods as gray backgrounds. Notice that the used bandwidth is indicated in the title of the figure. The right-hand figure also shows the concentration points and neighbourhoods, but also a line for the data with different line styles indicating which state the points are considered as being in. (The data in the plot, and in all the similar plots following, is slightly smoothed using moving averages for better visual quality.)

Simulation details

The modelling process requires both the simulation of the Markov chain and OU process at the same time. Thus, the Markov chain S_t is considered first. It is simulated using numbers u from a pseudo-random number generator on $[0, 1]$, where the interval is partitioned into a set of disjoint subintervals for each state, as described in [6]. The OU parameters belonging to the state at each time step are used in the updating formula (3) for $X(t)$.

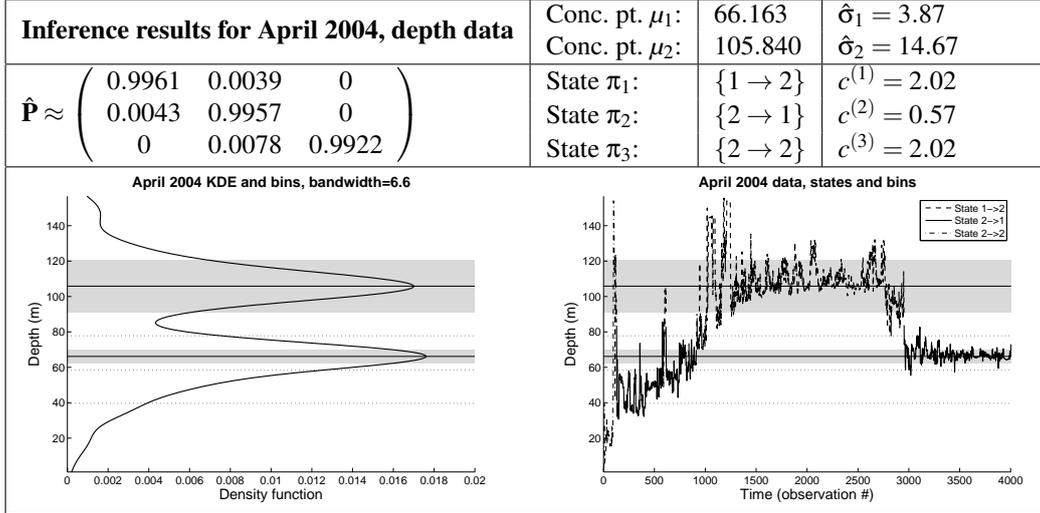


Table 1: Inference results for April 2004, depth data

Parameters for the Ornstein-Uhlenbeck process

For each state of the Markov chain S_t we determine the parameters b (drift term coefficient) and c (diffusion term coefficient) for the Ornstein-Uhlenbeck process. b is in this paper chosen empirically and equal for all states. This is done by visually inspecting the plots of the simulated paths and adjusting b so that they resemble the paths in the data ($b = 0.05$ is used throughout). A systematic way of doing this is suggested in [8], but this has not been implemented in this paper due to time constraints. The analysis shows that the results have low sensitivity to the accurate determination of b , especially in terms of picking the main dynamics of the migratory patterns. From the expression (2b), using σ^2 to denote the long-term variance of the OU process, we get the relation $c = \sigma\sqrt{2b}$. We estimate the long-term variance of the process for a state $\pi_k = \{i \rightarrow j\}$ as the squared standard deviation $\hat{\sigma}_j^2$ of all data points that are within the neighbourhood of concentration point μ_j , so we set $c^{(k)} = \hat{\sigma}_j\sqrt{2b^{(k)}}$. Thus, all states with the same destination concentration points have the same OU-parameters. Nevertheless, keeping a state for each combination of concentration points in the model allows for further experiments in future work by larger variations of b and c between states.

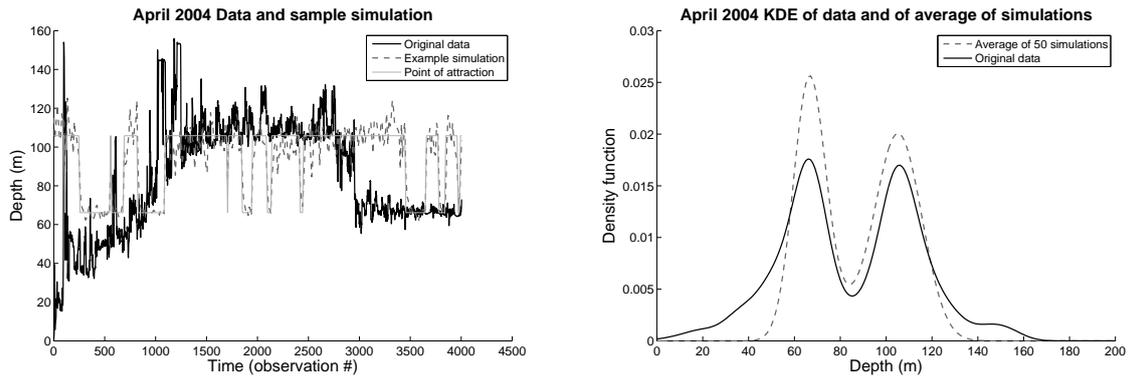
Simulation time step

The time step used in the simulation should not be the same as in the data, because the random effects in the fish behaviour occur over a larger time-scale than just one time step of 10 minutes. For instance, one may observe a steady increase in depth over an hour before the depth decreases again, all while the fish is in the same state. Had we used a time step of only 10 minutes in the simulation, we would get large oscillations in the values for very short time intervals, since the random part of the model is very dominant once the process is close to a concentration point. Thus, we choose for now to simulate using a 2 hour time step instead, corresponding to 12 time steps in the data. Since the transition matrix we have estimated is for time steps of 10 minutes, we need to use the 12-step transition probabilities for S_t , which are the entries of the matrix $\hat{\mathbf{P}}^{12}$. So in practice, the updating formula above is used with $\Delta t = 12$ and with OU parameters that depend on the state of the Markov chain in the new time-point. We define

$$X_{t+1} = X_t + b^{(k)} \left(\mu^{(k)} - X_t \right) \Delta t + c^{(k)} n \sqrt{\Delta t}, \quad (6)$$

with k given by the Markov chain in the sense that $S_{t+1} = \pi_k$.

Figure 3: Example simulation for April 2004 and KDE plot



Markov chain and OU process initial states, and burn-in

We choose as initial state S_1 the special, transient state $\pi_{\text{initial}} = \{i \rightarrow i\}$ for the first concentration point μ_i visited. Also, we let the OU process start in this concentration point by letting $X_1 = \mu_i$. The initial state chosen for a Markov chain will slightly affect the mean time spent in each state over the whole simulation period, but the longer the chain is allowed to evolve, the less this “noise” from the start of the process affects it. To get results that are not affected too much by the initial state, we let the process evolve for 200 time steps before recording the data. This is known as “burn-in”.

Simulation examples and comparing to data

In Figure 3 we show on the left an example of a simulation using the states, the transition matrix and the OU parameters inferred from the April 2004 depth data shown in Table 1. Note the solid gray line indicating which concentration point the process is attracted towards. The number of data points in the simulation are 1/12 of those in the data, but we have plotted the simulation such that the two paths are aligned in time, the units of the x-axis corresponding to the original numbering of the observation data points.

We see that some features of the original data for April 2004 are recreated in the simulation, in the sense that the process spends time around the same concentration points, and with more variability around the concentration point μ_2 at 106 meters than μ_1 at 66 meters. Comparing kernel density estimates for the data and the simulation (making sure to use the same bandwidth in order to keep the plots comparable) confirms this. But since this is a random process, both due to the randomness in the Markov chain and due to the random term in the updating formula, the simulated path will be different every time. To get an idea of the “mean behaviour” of the model, we take the average value of the KDEs for 50 different simulations and compare that with the KDE for the data.

The resulting mean KDE plot for the April 2004 simulations is shown with a dashed line on the right in Figure 3 together with a solid plot of the KDE of the data. We see that the time spent around μ_1 is overestimated by the model. But this is not surprising – there is a lot of area under the solid curve to the left that corresponds to data points whose characteristics we have not made any attempt to capture in the model. The peak around μ_1 at 66 meters could have been a little wider to capture more of the original data spread, indicating that the diffusion term coefficient c used in the state that generated those data points was too small. This indicates that the method for choosing c should be improved.

Table 2 shows complete inference and simulation results for a different month in the example time series: January 2005. Here we also see that the distribution of the data in the simulation matches quite well to the distribution of the original data.

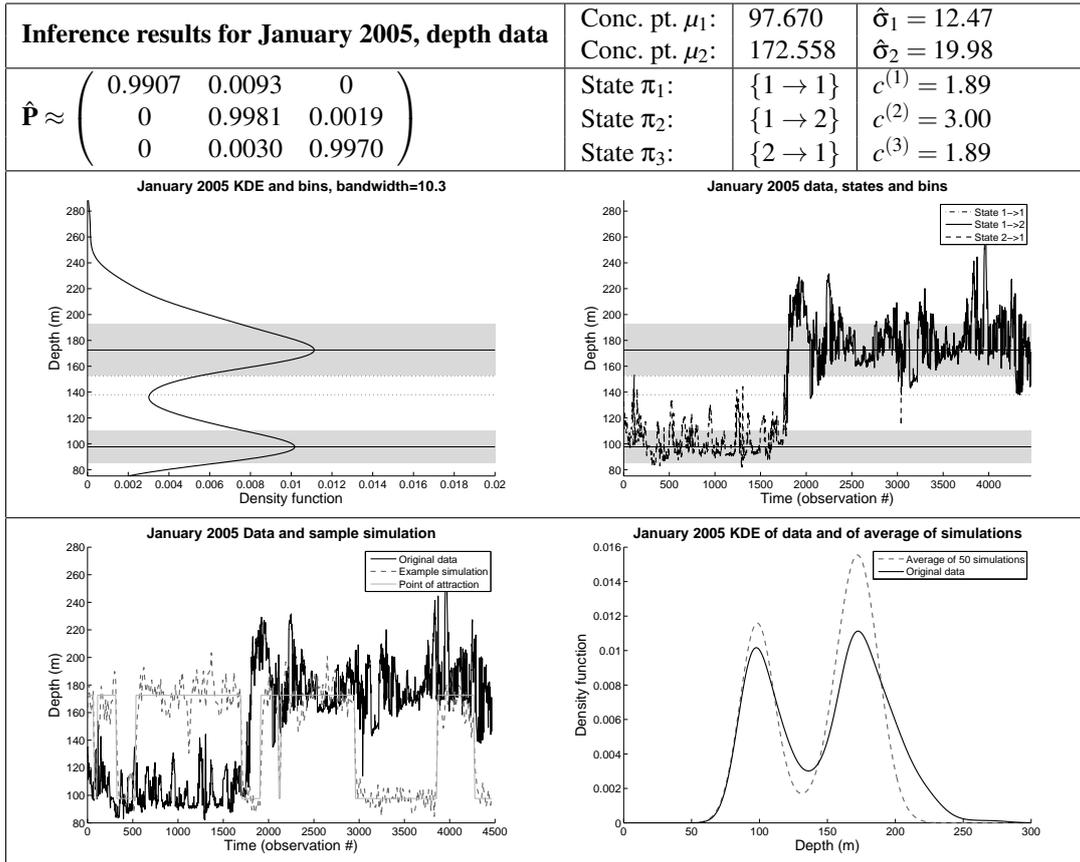


Table 2: Inference results and simulations for January 2005, depth data

Simulations matching poorly

We see that the plot of the average of the KDEs matches pretty poorly with the data KDE in the results for May 2004 in Table 3. The two peaks from the simulations have almost opposite heights, which is very different from the situation in the data. Not all such average KDEs of simulation series using the parameters of this data have shown as poor results as this, but none of them show good model fit. Poor model fit may occur whenever the fish has stayed close to one concentration point μ_i for a long time at the start of the month. The problem in such cases is that the initial state $\pi_{\text{initial}} = \{i \rightarrow i\}$ accounts for a large portion of the time spent around μ_i . Randomness might bring the Markov chain away from this state earlier than observed in the data, (possibly during burn-in) and since it is transient, the chain will never revisit the state. Then, the proportion of time spent around each concentration point for the rest of the simulated path will depend on the corresponding proportion in the data *after* π_{initial} was left, which might not correspond to the total proportion of time spent around each point in the whole month. This problem needs to be solved in order to make the model usable for months where this occurs. A possible solution is to avoid using whole months as cut-points for the data.

6 Conclusion

The model does quite a good job of capturing the main aspects of the vertical dynamics for time series where the initial transient state is left early in the data. This is a good starting point for modelling vertical migration of the fish. However, for months such as May 2004, starting with a long-lasting transient state, the model fails to capture the main characteristics.

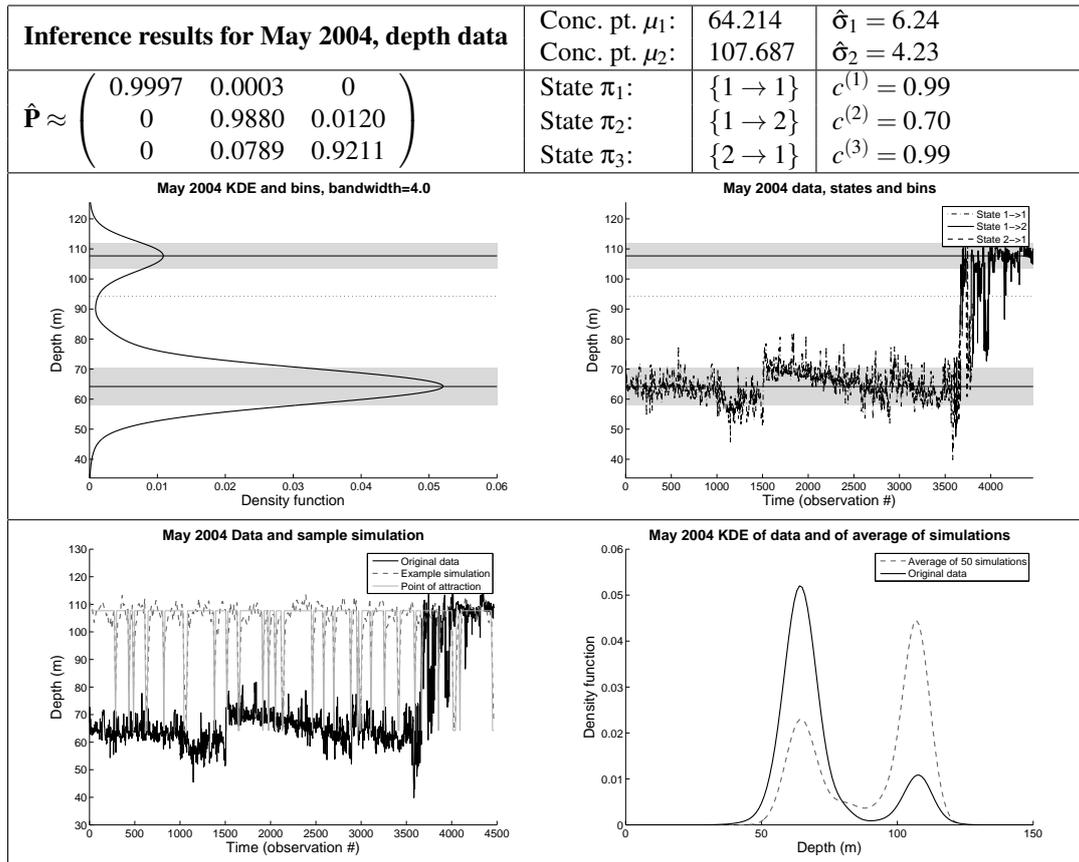


Table 3: Inference results and simulations for May 2004, depth data

Possible improvements to the model

Obtaining better model fit to data requires a method for estimating the OU process parameters and a systematic approach for varying the parameters between states. More systematic adjustment of KDE bandwidth must be considered. One might argue that a transition between states should only be recorded if the state change lasts for some time. A short excursion of 4-5 data points towards a different concentration point should perhaps not be counted as a transition, since it often will affect the transition matrix and thus the model dramatically. It is also necessary to find a way to avoid the problem with long-lasting transient states. Finally, attempts would be made to use second-order or exact updating formulas for X_t in the simulations.

Validity and reliability

The procedures in this paper have not been tested with other data than those mentioned, and conditions for them to work well for other data will now be considered. The model depends on data having a mean-reverting tendency. The data should be characterized by more than one concentration point. Otherwise the model will simplify to a single-state OU process starting at the mean, making it a white-noise process. No attempts have been made to enforce biological plausibility in the model, for instance “speed limits” or behavioural restrictions. The time aspect (i.e. the time spent between transitions) of the model versus original data has not been considered.

Extensions

Plans for further work include:

- Improve model fit by predefining a transition matrix and OU parameters, and trying to recover them using the inference methods described here on simulations produced using these settings.
- Testing the model also on temperature data and extending it to two dimensions: depth and temperature, where concentration points are defined in the depth/temperature plane.
- Using the model as a basis for species discrimination (North-East Arctic cod versus Norwegian coastal cod).
- Attempting geolocation of the fish by combining the model with a depth and temperature atlas.
- Using the model as a basis for making inferences on behaviour on the population level.

References

- [1] P.G. Blackwell. Random diffusion models for animal movement. *Ecological Modelling*, 100(1-3):87 – 102, 1997.
- [2] J.E. Dunn and P.S. Gipson. Analysis of Radio Telemetry Data in Studies of Home Range. *Biometrics*, 33(1):85–101, 1977.
- [3] R.N. Gibson. Go with the flow: tidal migration in marine animals. *Hydrobiologia*, 503(1-3):153–161, 2003.
- [4] Daniel T. Gillespie. Exact numerical simulation of the Ornstein-Uhlenbeck process and its integral. *Physical Review E*, 54(2):2084–2091, 1996.
- [5] Fabian Mörchen and Alfred Ultsch. Optimizing time series discretization for knowledge discovery. In *Proceedings The Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 660–665, 2005, Chicago, IL, USA.
- [6] J.R. Norris. *Markov Chains*. Cambridge Series on Statistical and Probabilistic Mathematics. Cambridge University Press, 1997.
- [7] Haiganoush K. Preisler, Alan A. Ager, Bruce K. Johnson, and John G. Kie. Modeling animal movements using stochastic differential equations. *Environmetrics*, (15):643–657, 2004.
- [8] Michael Rosenblum. Mobility modeling with a mixed Ornstein-Uhlenbeck process. Found on website <http://people.csail.mit.edu/mrosenblum/work/>.
- [9] Sam Subbey, Kathrine Michalsen, and Geir Nilsen. A tool for analyzing information from data storage tags: the continuous wavelet transform (CWT). *Reviews in Fish Biology and Fisheries*, 18:301–312, 2008. 10.1007/s11160-007-9078-2.
- [10] G.E. Uhlenbeck and L.S. Ornstein. On the theory of the brownian motion. *Physical Review*, 36(5):0823–0841, Sep 1930.
- [11] D. Weihs. Tidal stream transport as an efficient method for migration. *Journal du Conseil International pour l'Exploration de la Mer*, (38):92–99, 1978.

Appendix B

Algorithms

For reference and reproducibility, this appendix gives some algorithms/functions omitted in the main presentation.

B.1 Estimating a transition matrix

Algorithm 5 function estimateTransitionMatrix()

Input: \mathbf{S}, Π

Output: $\hat{\mathbf{P}}$

- 1: $m \leftarrow |\Pi|$
 - 2: $\hat{\mathbf{P}}, \mathbf{E} \leftarrow \mathbf{0}_{m,m}$ {Zero matrix of dimension $m \times m$ }
 - 3: **for** $t \leftarrow 2, \dots, N$ **do**
 - 4: $E_{S_{t-1}, S_t} \leftarrow E_{S_{t-1}, S_t} + 1$
 - 5: **for** $i \leftarrow 1, \dots, m$ **do**
 - 6: **for** $j \leftarrow 1, \dots, m$ **do** {Take $\hat{\mathbf{P}}$ as normalized rows of \mathbf{E} }
 - 7: $\hat{P}_{i,j} \leftarrow E_{i,j} / \sum_{k=1}^m E_{i,k}$
-

B.2 Simulating a Markov chain

The simulation of a Markov chain in practice in MATLAB is done by exploiting the function `histc()`. This function can find out which class in a histogram an observation should lie in, using as classes the cumulative sum vector padded by minus and plus infinity. The `find()` function finds the location of the non-zero element of the vector from `histc()`.

```
% given transition matrix P for n states,
% the chain is in state i
find(histc(rand,[-inf,cumsum(P(i,1:n-1)),inf]))
```

By re-running this expression and letting the `i` variable contain the outcome of the previous iteration, we effectively simulate a Markov chain.

B.3 Local maxima of a matrix

The function `localMaxima()` takes a matrix, and returns the indices of any position in the matrix that is considered as a local maximum, along with the value at that position. A local maximum of a matrix is in this context defined to be a position where the value is smaller at any neighbouring position on the row, column or diagonal (i.e., the position has greater value than all the values in the 8 positions around it).

B.4 Detailed description of geolocalization algorithm

We now give the algorithm for geolocalization with much more detail than the outline in Algorithm 4 in Chapter 7. It is presented as one main algorithm and several subalgorithms in the form of functions (in the computer programming sense of the word). Some of the enhancements suggested in the original presentation are not included here. Global functions always available are the atlas functions $\mathcal{D}(x, y)$ and $\mathcal{T}(x, y, d, t)$. In addition, the following parameteres are global:

- Δ – the closeness for merging solutions
- v_{max} – swimming range in one day
- κ_0 – initial tolerance for accepting solution
- κ_ρ – factor with which to increase tolerance if needed
- κ_{max} – maximum tolerance allowed
- κ_f – tolerance used after running `fmincon()` for refining optimization.

Algorithm Main algorithm for the geolocalization problem

Input: $\mathbf{T}, \mathbf{D}, P_{init}$

- 1: $\Psi^{(0)} \leftarrow \{\text{new SolutionPath}(P_{init})\}$
- 2: **for** $t \leftarrow 1, \dots, t_{max}$ **do**
- 3: $\Psi^{(t)} \leftarrow \emptyset$
- 4: **for each** $\psi \in \Psi^{(t-1)}$ **do**
- 5: $P_0 \leftarrow \psi.\text{currentEndPoint}$
- 6: $\Phi \leftarrow \text{geoLocalSearch}(P_0, D_t, T_t, t)$
- 7: **for each** $\phi \in \Phi$ **do**
- 8: $\Psi^{(t)} \leftarrow \Psi^{(t)} \cup \text{newPath}(\Psi^{(t)}, \phi, \psi)$
- 9: $\Psi^{(t)} \leftarrow \text{optimizeEndpoints}(\Psi^{(t)})$

function $\Phi = \text{geoLocalSearch}(P_0, D, T, t)$

- 1: $(x_0, y_0) \leftarrow P_0$
- 2: $\kappa \leftarrow \kappa_0$
- 3: $\Phi \leftarrow \emptyset$
- 4: **while** $|\Phi| < 2$ **do**
- 5: **for each** $\left\{ (i, j) : \left\| (x_0 - i, y_0 - j) \right\|_2 \leq v_{max} \right\}$ **do**
- 6: **if** $|\mathcal{T}(i, j, D, t) - T| < \kappa$ **then**

```

7:    $\Phi \leftarrow \Phi \cup (i, j)$ 
8:    $\kappa \leftarrow \kappa \times \kappa_\rho$  {Increase tolerance}
9:   if  $\kappa > \kappa_{max}$  then
10:    return  $\Phi$ 
11:  $\Phi \leftarrow \text{trimSolutions}(\Phi)$ 
12: return  $\Phi$ 

```

```

function path = newPath( $\Psi^{(t)}, \phi, \tilde{\psi}$ )

```

```

1: Note: Now  $\tilde{\psi}$  is a path of shorter length than those in  $\Psi^{(t)}$ .
2: if  $\Psi^{(t)} = \emptyset$  or  $t = 1$  then
3:   return new SolutionPath( $\phi, \tilde{\psi}$ )
4: for each  $\psi \in \Psi^{(t)}$  do
5:   if hasEquivalentEndPoint( $\psi, \phi$ ) and pathAlreadyHasPath( $\psi, \tilde{\psi}$ ) then
6:     return  $\emptyset$ 
7:   else if hasEquivalentEndPoint( $\psi, \phi$ ) then
8:     addPathAndEndpoint( $\psi, \phi, \tilde{\psi}$ )
9:     return  $\emptyset$ 
10: return new SolutionPath( $\phi, \psi$ )

```

```

function addPathAndEndpoint( $\psi, \phi, \tilde{\psi}$ )

```

```

1:  $\psi.\text{endPoints} \leftarrow \psi.\text{endPoints} \cup \phi$ 
2:  $\psi.\text{prevPaths} \leftarrow \psi.\text{prevPaths} \cup \tilde{\psi}$ 

```

```

function  $\tilde{\Psi} = \text{optimizeEndPoints}(\Psi, D, T, t)$ 

```

```

1:  $\tilde{\Psi} \leftarrow \emptyset$ 
2:  $f(x, y) \leftarrow |\mathcal{T}(x, y, D, t) - T|$ 
3: for each  $\psi \in \Psi$  do
4:    $(x_0, y_0) \leftarrow \text{mean}(\psi.\text{endPoints})$ 

```

```

5:  if  $(x_0, y_0)$  is an infeasible point then
6:     $(x_0, y_0) \leftarrow \psi.\text{endPoints}(1)$  {Use the first endpoint instead}
7:     $\text{LB}, \text{UB} \leftarrow$  optimization bounds such that all endpoints are within them
8:     $(x, y, f_{xy}) \leftarrow \text{fmincon}(f, (x_0, y_0), \text{LB}, \text{UB})$ 
9:    if  $f_{xy} < \kappa_f$  then
10:      $\psi.\text{currentEndPoint} \leftarrow (x, y)$ 
11:      $\tilde{\Psi} \leftarrow \tilde{\Psi} \cup \psi$ 
12: return  $\tilde{\Psi}$ 

```

```
function hasEquivalentEndPoint()
```

This algorithm detects the closeness of a new path, returns true if the endpoint of the new path falls within a box of extent Δ from the original path end point.

```
function pathAlreadyHasPath()
```

This algorithm is used to ensure that paths that are virtually identical are added to the solution space. In particular, if the two paths are equal up to and excluding the end point, and the endpoint is close enough for merging, this function returns true.

Appendix C

Implementation issues

This appendix contains info on implementation, details that are not necessary for the mathematical discussion but only for the efficiency in implementing methods on a computer. This is hoped to be of aid for any reproducing of the results in this thesis.

C.1 2-D kernel density estimation

The 2-D kernel density estimation uses software from the MATLAB Central File Exchange. An issue with applying this method on long time series, such as the four-month data used in the thesis, is that the running time and memory consumption is extremely high if a high resolution in the estimate is desired. Hence, the 2-D KDEs shown in the thesis are made using rather low resolution, and are interpolated by MATLAB in the plots. This leads to the placement of concentration points to be a bit inaccurate compared to the true peak of the function.

C.2 Adaptive-degree polynomial filter

The software for ADPF smoothing was downloaded from the MATLAB Central File Exchange ¹. This script is very general, because it also provides values for the smoothed time series at arbitrary points, i.e. with higher resolution than the sampling of original data. It also has good graphics. The script was modified to make it into a pure filter, with same number of points both before and after smoothing.

ADPF smoothing is time-consuming, because polynomial regression must be performed at each time step. This makes it difficult (e.g. slow) to make automated experiments with the width of the filter window. This is another reason for considering adaptive-window filters instead.

C.3 Implementation of spline interpolation

The spline interpolation for temperature is made using the function `tpaps` in MATLAB Spline Toolbox. The interpolation would not be efficient if the spline had to be constructed every time an atlas value is requested by the algorithm. To solve this, the spline used for each grid cell at each time step was stored in a MATLAB cell array, with linear indexing. The mapping from position in the atlas to position in the cell array was stored in a sparse matrix for each time step, which is effective for keeping the memory consumption low.

A possible improvement for keeping memory consumption even lower is to make splines for larger regions at a time, but the mapping from position to cell array would be more complicated. Another issue is that the algorithm in MATLAB changes nature when the number of interpolation nodes exceeds 728, and the running time quickly becomes unacceptable. The region for each spline can thus not be too big. It is therefore not possible to make e.g. a single spline that can be evaluated at all the points in the search radius of a previous position in the algorithm.

Situations have occurred when the storage of a huge number of splines (more than 10000) has made the algorithm very slow. Storing the spline took more time than

¹<http://www.mathworks.com/matlabcentral/fileexchange/34892-adaptive-degree-smoothing-and-differentiation>

making it. This calls for deleting some of the splines if the memory consumption becomes too high.

C.4 Running time of geolocalization algorithm

The main time-consumators in the geolocalization algorithm are loading the atlas for each time step, doing the linear interpolation of depth and enumerating the temperature at all integer points in a reasonable distance from the previous position. To solve this, computed integer temperature were stored in a sparse array for each time step. The splines must also use these stored interpolated values for its nodes. Also, with the storage of splines suggested in the previous section, evaluating temperatures in areas that already have been explored by the algorithm is much quicker than just getting the linear interpolations for depth for an area.

Recall the solution to the issue with infeasible solutions described in the chapter on future work. It turns out that the optimization to correct this can be very efficient, even with a time-varying atlas, if the perturbations of the path needed are so small that no new areas are explored, and only existing splines are evaluated. The chance of this is increased if upper and lower bounds are set on each value of the series such that the search does not venture too far.

C.5 Atlas matrices

The position, depth and surface temperature matrices for the area around the Lofoten Archipelago are given here.

$$M_\phi = \begin{pmatrix} 68.7 & 68.6 & 68.5 & 68.4 & 68.2 & 68.1 & 68.0 & 67.9 \\ 68.6 & 68.5 & 68.4 & 68.2 & 68.1 & 68.0 & 67.9 & 67.8 \\ 68.5 & 68.4 & 68.2 & 68.1 & 68.0 & 67.9 & 67.8 & 67.6 \\ 68.4 & 68.2 & 68.1 & 68.0 & 67.9 & 67.8 & 67.6 & 67.5 \\ 68.2 & 68.1 & 68.0 & 67.9 & 67.8 & 67.6 & 67.5 & 67.4 \\ 68.1 & 68.0 & 67.9 & 67.8 & 67.6 & 67.5 & 67.4 & 67.3 \\ 68.0 & 67.9 & 67.7 & 67.6 & 67.5 & 67.4 & 67.3 & 67.2 \\ 67.8 & 67.7 & 67.6 & 67.5 & 67.4 & 67.3 & 67.1 & 67.0 \\ 67.7 & 67.6 & 67.5 & 67.4 & 67.3 & 67.1 & 67.0 & 66.9 \\ 67.6 & 67.5 & 67.4 & 67.2 & 67.1 & 67.0 & 66.9 & 66.8 \end{pmatrix}$$

$$M_\lambda = \begin{pmatrix} 14.1 & 14.4 & 14.8 & 15.1 & 15.4 & 15.7 & 16.1 & 16.4 \\ 13.8 & 14.1 & 14.4 & 14.8 & 15.1 & 15.4 & 15.7 & 16.0 \\ 13.4 & 13.8 & 14.1 & 14.4 & 14.8 & 15.1 & 15.4 & 15.7 \\ 13.1 & 13.4 & 13.8 & 14.1 & 14.4 & 14.8 & 15.1 & 15.4 \\ 12.8 & 13.1 & 13.5 & 13.8 & 14.1 & 14.5 & 14.8 & 15.1 \\ 12.5 & 12.8 & 13.1 & 13.5 & 13.8 & 14.1 & 14.5 & 14.8 \\ 12.2 & 12.5 & 12.8 & 13.2 & 13.5 & 13.8 & 14.2 & 14.5 \\ 11.8 & 12.2 & 12.5 & 12.9 & 13.2 & 13.5 & 13.9 & 14.2 \\ 11.5 & 11.9 & 12.2 & 12.6 & 12.9 & 13.2 & 13.5 & 13.9 \\ 11.2 & 11.6 & 11.9 & 12.3 & 12.6 & 12.9 & 13.3 & 13.6 \end{pmatrix}$$

$$M_{\mathcal{D}} = \begin{pmatrix} 435.5 & 223.1 & 112.8 & 65.6 & 50 & 50 & 50 & 50 \\ 388.7 & 203.5 & 111.3 & 73.0 & 57.8 & 50 & 50 & 50 \\ 344.3 & 186.8 & 112.5 & 83.4 & 71.1 & 61.1 & 50.4 & 50 \\ 304.1 & 172.4 & 114.6 & 94.9 & 86.3 & 75.5 & 61.1 & 50 \\ 269.9 & 160.8 & 117.4 & 106.3 & 101.9 & 91.0 & 73.1 & 54.7 \\ 244.6 & 154.6 & 122.8 & 118.5 & 117.4 & 106.2 & 85.1 & 62.3 \\ 230.7 & 156.0 & 132.7 & 132.5 & 132.4 & 119.6 & 95.2 & 68.6 \\ 228.8 & 165.1 & 147.2 & 147.9 & 145.9 & 129.9 & 101.9 & 72.4 \\ 237.8 & 180.6 & 164.8 & 163.9 & 157.6 & 136.7 & 105.1 & 73.4 \\ 256.2 & 201.7 & 185.6 & 181.0 & 168.8 & 142.1 & 106.4 & 73.0 \end{pmatrix}$$

$$M_{\mathcal{T}} = \begin{pmatrix} 7.447 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 7.462 & 7.624 & 0 & 0 & 0 & 0 & 0 & 0 \\ 7.508 & 7.611 & 0 & 6.206 & 5.529 & 0 & 0 & 0 \\ 7.550 & 0 & 0 & 6.372 & 6.465 & 0 & 0 & 0 \\ 7.486 & 0 & 6.789 & 7.227 & 7.310 & 6.877 & 0 & 0 \\ 7.477 & 0 & 7.100 & 7.814 & 7.387 & 6.868 & 0 & 0 \\ 7.629 & 7.052 & 7.293 & 7.480 & 7.551 & 7.624 & 7.637 & 0 \\ 8.196 & 7.706 & 7.457 & 7.396 & 7.448 & 7.554 & 7.652 & 0 \\ 7.906 & 7.807 & 7.638 & 7.438 & 7.286 & 7.254 & 7.428 & 0 \\ 8.001 & 7.645 & 7.513 & 7.555 & 7.403 & 7.402 & 7.621 & 0 \end{pmatrix}$$

List of Figures

2.1	Example trajectories of OU processes	23
2.2	Example of ADPF smoothed time series	34
2.3	Examples of kernel density estimates for 1-D (with varying bandwidth) and for 2-D data (blue is low density, red is high).	37
2.4	Depth map of Norwegian and Barents seas, and of the area around the Lofoten Archipelago. Source: The ROMS atlas, see Section 6.3.	41
3.1	Mobile phone modelling: dividing an area into neighbourhoods	47
4.1	All the data from tag 1664	55
4.2	Two example 10-day trajectories shown as scatterplot and as time series	55
4.3	Box plots for each month of data	56
4.4	Correlation between depth and temperature	57
5.1	Depth data showing slow variation	71
5.2	Example simulation for January 2005 and KDE comparison plot	73
5.3	Inference on data for four periods	79
5.4	Example of mean of 50 simulations	80
5.5	Example of weighted mean of 50 simulations	82
5.6	Simulation results for the depth data for May 2004 using a deterministic sequence of states, instead of states generated by a Markov chain.	83
5.7	All visits to state $\{2 \rightarrow 3\}$ for April-July depth data.	86

5.8	ACF and PACF for 1000 depth data points in April 2004, with confidence bounds.	88
5.9	Sample autocorrelation and Q-Q plot of residuals after smoothing, for one short and one long visit to a state of the April-July depth data. Smoothing window used: 27.	90
5.10	Smoothing that fits very closely to precision noise	91
5.11	Comparison of 2-D KDEs for April-July 2004 data and simulations	101
6.1	20 km and 4 km atlases for different times	109
6.2	The coordinates in the matrix and matrix row/column indices in map	110
6.3	Error made in distance calculation in the 20 km gridded atlas when travelling in i direction, j direction and on the diagonal from a point north of Spitsbergen.	111
6.4	Map showing directions travelled when calculating distance errors	112
6.5	Depth values and example surface temperature around Lofoten	113
6.6	Illustration of use of layers for three adjacent points for matrix positions $i = 203$ and $j = 259, 260, 261$	114
7.1	Illustration of interpolation of temperature for depth	129
7.2	Mask and vertically interpolated temperatures at 120 meters.	130
7.3	Different interpolation schemes and informal error analysis.	131
7.4	Horizontal interpolation of surface temperature and temperature at 120 meters for Lofoten area.	133
7.5	Test instances 1 and 2	134
7.6	One step temperature and objective.	137
7.7	Demo of merging solutions	141
7.8	Demonstrating the effect of tolerance setting.	144
7.9	Solutions from running on test instance 2 with too low tolerance.	146
7.10	All solutions from running test instance 2 with higher tolerance.	147
7.11	Analysis of objective function value of solutions to test instance 2.	147
7.12	Solutions to test instance 2 with best end point matches.	148
7.13	Solutions at 30 and 100 days for test instance 1.	149

7.14 Solutions from running test instance 2 with too high speed limit. . . . 150

7.15 Demonstration of how backtracking matters for test instance 4. . . . 151

7.16 Using habitat restriction to reduce spread in solutions for test instance
4. 152

7.17 Run 1 on DST data, 30 days and a selection of solutions 153

7.18 Run 1 on DST data, 60 and 120 days 154

7.19 Run 2 on DST data with habitat masking, 30 and 60 days 154

7.20 Run 2 on DST data with habitat masking, 400 and 200 days 155

7.21 Shortest and longest paths from run 2 at 200 days, shown in map . . 156

8.1 Inference on data from April-July with presmoothing 166

Bibliography

- [1] Robert A. Adams. *Calculus: a complete course*. Addison Wesley Longman, 5th edition, 2003.
- [2] Richard Bainbridge. The speed of swimming of fish as related to size and to the frequency and amplitude of the tail beat. *Journal of Experimental Biology*, 35:109–133, 1958.
- [3] Phillip Barak. Smoothing and differentiation by an adaptive-degree polynomial filter. *Analytical Chemistry*, 67(17):2758–2762, 1995.
- [4] Simeon M. Berman. A bivariate markov process with diffusion and discrete components. *Communications in Statistics. Stochastic Models*, 10(2):271–308, 1994.
- [5] P.G. Blackwell. Inference for Ornstein-Uhlenbeck processes in random environments. Research Report, Sheffield Univ. (United Kingdom). Dept. of Probability and Statistics, 1996.
- [6] P.G. Blackwell. Random diffusion models for animal movement. *Ecological Modelling*, 100(1-3):87 – 102, 1997.
- [7] F. L. Bookstein. Principal warps: Thin-plate splines and the decomposition of deformations. *IEEE Trans. Pattern Anal. Mach. Intell.*, 11(6):567–585, June 1989.
- [8] Peter J. Brockwell and Richard A. Davis. *Introduction to Time Series and Forecasting*. Springer, 2nd edition, March 2002.

- [9] M. Browne, N. Mayer, and T. R. H. Cutmore. A multiscale polynomial filter for adaptive smoothing. *Digit. Signal Process.*, 17(1):69–75, January 2007.
- [10] William Henry Burt. Territoriality and home range concepts as applied to mammals. *Journal of Mammalogy*, 24(3):pp. 346–352, 1943.
- [11] Peter J. Diggle. *Time series: a biostatistical introduction*. Clarendon, 1990.
- [12] J.E. Dunn and P.S. Gipson. Analysis of Radio Telemetry Data in Studies of Home Range. *Biometrics*, 33(1):85–101, 1977.
- [13] Anders Fernö, Terje Jørgensen, Svein Løkkeborg, and Paul D. Winger. Variable swimming speeds in individual atlantic cod (*gadus morhua* l.) determined by high-resolution acoustic tracking. *Marine Biology Research*, 7:310–313, 2011.
- [14] International Council for the Exploration of the Sea. Advice for 2011 on norwegian coastal cod. <http://www.ices.dk/committe/acom/comwork/report/2010/2010/cod-coas.pdf>. View date April 24th 2012.
- [15] José Carlos García Franco. Maximum likelihood estimation of mean reverting processes. Published on web: http://www.investmentscience.com/Content/howtoArticles/MLE_for_OR_mean_reverting.pdf. View date January 25th 2012.
- [16] Daniel T. Gillespie. Exact numerical simulation of the Ornstein-Uhlenbeck process and its integral. *Physical Review E*, 54(2):2084–2091, 1996.
- [17] Olav Rune Godø and Kathrine Michalsen. Migratory behaviour of north-east arctic cod, studied by use of data storage tags. *Fisheries Research*, 48:127–140, 2000.
- [18] Desmond J. Higham. An algorithmic introduction to numerical simulation of stochastic differential equations. *SIAM Review*, 43:525–546, 2001.
- [19] Roger A. Horn and Charles R. Johnson. *Topics in matrix analysis*. Cambridge University Press, 1994.

- [20] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942–1948 vol.4. IEEE, November 1995.
- [21] David Kincaid and Ward Cheney. *Numerical Analysis: Mathematics of Scientific Computing*. Brooks/Cole, 3rd edition edition, 2002.
- [22] Jack P. C. Kleijnen. Verification and validation of simulation models. *European Journal of Operational Research*, 82(1):145–162, April 1995.
- [23] Y. K. Kwok. *Mathematical models of financial derivatives*. Springer, 1998.
- [24] Vidar S. Lien. Institute of Marine Research. Personal communication, 2012.
- [25] Vidar S. Lien, Paul Budgell, Bjørn Ådlandsvik, and Einar Svendsen. Validating Results from the model ROMS (Regional Ocean Modelling System) with respect to volume transport and heat fluxes in the Nordic Seas. *Fisken og havet* 2/2006, Institute of Marine Research, 2006.
- [26] Movable Type Ltd. Calculate distance, bearing and more between latitude/longitude points. Web site: <http://www.movable-type.co.uk/scripts/latlong.html>. View date April 10 2012.
- [27] Reinhard Mahnke, Jevgenijs Kaupužs, and Ihor Lubashevsky. *Physics of Stochastic Processes*. WILEY-VCH Verlag GmbH & Co., 2009.
- [28] W. L. Martinez and A. R. Martinez. *Computational statistics handbook with MATLAB*. Chapman & Hall/CRC, 2002.
- [29] Cleve Moler and Charles Van Loan. Nineteen Dubious Ways to Compute the Exponential of a Matrix, Twenty-Five Years Later. *SIAM Review*, 45(1):3–49, 2003.
- [30] Fabian Mörchen and Alfred Ultsch. Optimizing time series discretization for knowledge discovery. In *Proceedings The Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 660–665, 2005, Chicago, IL, USA.

- [31] Erik Natvig and Sam Subbey. Modelling vertical fish migration using Mixed Ornstein-Uhlenbeck processes. In *Proceedings Norsk Informatikkonferanse NIK 2011*, pages 73–84, 2011, Tromsø, Norway.
- [32] Jennifer L. Nielsen, Nuno Fragoso, Molly Lutcavage, Haritz Arrizabalaga, Alistair Hobday, and John Sibert (editors). *Tagging and Tracking of Marine Animals with Electronic Devices*, volume 9 of *Reviews: Methods and Technologies in Fish Biology and Fisheries*. Springer, 2009.
- [33] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer Series in Operation Research and Financial Engineering. Springer, second edition edition, 2006.
- [34] J.R. Norris. *Markov Chains*. Cambridge Series on Statistical and Probabilistic Mathematics. Cambridge University Press, 1997.
- [35] Norwegian Ministry of Fisheries and Coastal Affairs. Press release, march 17th 2010. <http://www.regjeringen.no/nn/dep/fkd/pressesenter/Pressemeldingar/2010/Gjenoppyggingsplan-for-kysttorsk-nord-for-62N.html?id=597705>. View date April 24th 2012.
- [36] Institute of Marine Research. Web page on fish species. <http://www.imr.no/temasider/fisk/en>. View data April 7th 2012.
- [37] Martin Væver Pedersen. Hidden markov models for geolocation of fish. Master’s thesis, Technical University of Denmark, Kongens Lyngby, Denmark.
- [38] Haiganoush K. Preisler, Alan A. Ager, Bruce K. Johnson, and John G. Kie. Modeling animal movements using stochastic differential equations. *Environmetrics*, (15):643–657, 2004.
- [39] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 2007.

- [40] Gilks W. R., S. Richardson, and D. J. Spiegelhalter. *Markov Chain Monte Carlo in Practice: Interdisciplinary Statistics (Chapman & Hall/CRC Interdisciplinary Statistics)*. Chapman and Hall/CRC, 1 edition, 1995.
- [41] G. A. Rose, B. deYoung, and E. B. Colbourne. Cod (*gadus morhua* l.) migration speeds and transport relative to currents on the north-east newfoundland shelf. *ICES J. Mar. Sci.*, 52:903–913, 1995.
- [42] Michael Rosenblum. Mobility modeling with a mixed Ornstein-Uhlenbeck process. Found on website <http://people.csail.mit.edu/mrosenblum/work/>.
- [43] Sheldon M. Ross. *Introduction to Probability Models*. Academic Press, 10th edition edition, 2010.
- [44] Egil Sakshaug, Geir Johnsen, and Kit Kovacs (editors). *Ecosystem Barents Sea*. Tapir Forlag, 2009.
- [45] A. Savitzky and M. J. E. Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical Chemistry*, 36(8):1627–&, 1964.
- [46] Robert R. Shumway and David S. Stoffer. *Time Series Analysis and Its Applications*. Springer Texts in Statistics. Springer, 2006.
- [47] R. W. Sinnott. Virtues of the Haversine. *Sky and Telescope*, 68(2):159+, 1984.
- [48] William Smith. On the simulation and estimation of the mean-reverting ornstein-uhlenbeck process. Web page: <http://commoditymodels.com/2010/02/24/parameter-estimation-mean-reverting-process/>. View date January 25th 2012.
- [49] J. Steinier, Y. Termonia, and J. Deltour. Comments on smoothing and differentiation of data by simplified least square procedure. *Analytical Chemistry*, 44(11):1906–&, 1972.
- [50] S. Subbey, K. Michalsen, H. Otneim, and G. Dingsør. Does Cod (*Gadus morhua*) Ride the Tides? Under review – Canadian Journal of Fisheries Science (2012).

- [51] Sam Subbey, Kathrine Michalsen, and Geir Nilsen. A tool for analyzing information from data storage tags: the continuous wavelet transform (CWT). *Reviews in Fish Biology and Fisheries*, 18:301–312, 2008. 10.1007/s11160-007-9078-2.
- [52] S. Subbey et al. Manuscript under preparation (2012).
- [53] Lloyd N. Trefethen and David Bau. *Numerical Linear Algebra*. SIAM: Society for Industrial and Applied Mathematics, June 1997.
- [54] G.E. Uhlenbeck and L.S. Ornstein. On the theory of the Brownian motion. *Physical Review*, 36(5):0823–0841, Sep 1930.
- [55] Thijs van den Berg. Calibrating the Ornstein-Uhlenbeck (Vasicek) model. Web page <http://www.sitmo.com/article/calibrating-the-ornstein-uhlenbeck-model/>. View date October 24th 2011.
- [56] J. J. Videler and C. S. Wardle. Fish swimming stride by stride: speed limits and endurance. *Reviews in Fish Biology and Fisheries*, 1:23–40, 1991.
- [57] Frode B. Vikebø, Åse Husebø, Aril Slotte, Erling K. Stenevik, and Vidar S. Lien. Effect of hatching date, vertical distribution, and interannual variation in physical forcing on northward displacement and temperature conditions of norwegian spring-spawning herring larvae. *ICES Journal of Marine Science: Journal du Conseil*, 67(9):1948–1956, December 2010.
- [58] Bjørn Ådlandsvik, Geir Huse, and Kathrine Michalsen. Introducing a method for extracting horizontal migration patterns from data storage tags. *Hydrobiologia*, 582:187–197, 2007.
- [59] Bernt Øksendal. *Stochastic differential equations : an introduction with applications*. Springer, 2003.