GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN IN PUBLICA COMMODA
SEIT 1737

Bachelor's Thesis

Classification of Fast Ice under Uncertainty

by

J. David Siantidis

26 July 2023
Department of Mathematics
University of Goettingen

Supervisors:
Prof. Dr. D. Russel Luke
Dr. Lars Kaleschke

Abstract. This thesis covers a stochastic approach on fast ice detection on simulated satellite observations. The main focus lies on the usage of the time dependence of the data points, and how to correct defective and noisy inputs. Monte Carlo simulation is the method of choice, utilizing its versatility. Another approach briefly looks at linear stochastic programming and its similarities to a weighted average in the context of the problem. The numerical comparison shows the predominant performance of the Monte Carlo approach with an average of over 86.5% accuracy in detecting the fast ice, at least on the simulated data.

Acknowledgements

I want to thank my supervisors and reviewers Prof. Dr. D. Russel Luke and Dr. Lars Kaleschke as well as Dr. Stephan Busch and Dr. M. Saee Bostanabad for academically advising me by attending regular meetings and giving constant constructive feedback.

Contents

List of Figures

List of Tables

List of Variables

All non-local variables are outlined in this section. Functions and variables specific for the approach *Stochastic Programming* are found on top of the corresponding section E.

Main Variables and Spaces.

$X_{0:t}$	Not yet made decisions from time 0 to t.
$x_{0:t}$	Made decisions from time 0 to t.
$\mathcal{X}$	Space of the decision variable
$Y_{0:t}$	Unobserved observations from time 0 to t.
$y_{0:t}$	Observed observations from time 0 to t.
$\mathcal{Y}$	Space of the observation variable
T	The maximal time point, it is always $0 \leq t \leq T$.

Mathematical Symbols (Mostly for Section 2).

$\leftarrow$	Notation for a backwards step (Used on kernels, probability distributions, etc.).

State Spaces:

$\mathbb{P}_0$	Initial distribution of X_0.
P_t	Transition kernels for the distribution of X_t given x_{t-1}.
f_t	transition density, likelihood for y_t given x_t.

Feyman-Kac Formalism:

$\mathbb{M}_0$	Initial distribution of X_0.
M_t	Transition kernels for the distribution of X_t given x_{t-1}.
G_0	Potential function at time 0.
G_t	Potential function at time t.
$\mathbb{Q}_t$	Feyman-Kac model, defined through (2.19).
L_t	Normalizing constant at time t, defined through (2.20).
l_t	Normalizing ratios, $l_t \coloneqq L_t/L_{t-1}$.
$H_{t:T}$	Cost-to-go functions, defined in (2.25).
γ_t	Required probability distribution for backwards filtering, eq. (2.50).
$w_t^{(n)}$	Weight of particle number n at time t.
$W_t^{(n)}$	Normalized weight of particle number n at time t.

Model Functions and Parameters (Mostly Section 3).

c_s	Convolution matrix to gather neighbourhood information in a decision x_t.
ζ_t	The neighbourhood information: $\zeta_t \coloneqq (c_2 * x_t)$
$g_{0,1}$	Likelihood function for the fast ice evolution (Introduced in (2.62)).
α_k	Splits g_k in three parts ($k = 0, 1$).
β_k	Determines the point wise evaluation at α_k.
ρ_k	Exponent of g_k.
$\varepsilon^{u,l}$	Minimal chance for fast ice / drift ice in a single evolution.

σ	Constant element of the transition density f_t.
$r_{1,2}$	Linear and quadratic factors for the transition density f_t.
$t^{0,1}$	the two time steps for seasonal changes (to winter / summer).

Notation

We will base our notation on the book *An Introduction to Sequential Monte Carlo* by *N. Chopin and O. Papaspiliopoulos, 2020* [4] as it has a versatile usage, it is later introduced and referenced (see chapter 2). The upper case X stands for random variables, and the corresponding lower case x for their realizations. Further, the notation dx stands for a **set** of the underlying algebra, mostly used in probability Kernels. As an example, $P(x, dy)$ relies on the random variable y in the second argument, an arbitrary set $A \in \mathcal{B}(\mathcal{Y})$ is avoided this way. Subscripts indicate a time dependence and for an array of variables we define the shortcut $x_{0:t} \coloneqq (x_1, x_2, \ldots, x_t)$. The extension of the subscript with $|T$ indicates the dependence on the time T, it is used for probability kernels/distribution, since they may differ throughout time. An example: $P_{t|T}(x_t)$ is the distribution of x_t in step t, factoring in the information available until time T, so this distribution might change for a different choice $S < T$.

Cooperations

This bachelor thesis is written in cooperation with the Alfred-Wegener-Institut (AWI). The AWI is a german institute for climate research, with a focus on sea and polar research. My contact person and supervisor is Dr. Lars Kaleschke from the climate sciences and sea ice physics department.

Introduction

Fast ice is close to motionless sea ice, mostly found in coastal areas attached to land. There, it has a multitude of important properties, ranging from providing habitat to wildlife [26] and humans [8], to regulating temperature exchanges and balancing ice sheet masses [10]. It also appears to be very sensitive to climate change, making it a good indicator of its progression.[17]. In climate sciences sea ice is modeled to infer behavior of nature and to compute possible scenarios. Considering these important properties of fast ice, it is not accurately simulated in many models [14]. For both practical and simulation purposes, fast ice needs to be accurately described in past and (more or less) real time. Today's approaches rely mostly on manually inscribed fast ice on a bi-weekly to weekly basis, sometimes supported by semi-automatic algorithms that flag/mark areas which do not show any movement [24]. As an example, one of the most recent works (mid-2022) is on simulating fast ice in deeper waters (See [16]). Here, as well as in earlier attempts at simulating fast ice the satellite data used for comparison is a two-week average ice concentration with flagged fast ice pixels on a square grid with a resolution of 25 km. Limiting factors in these approaches are the time lags between two data points as well as the resolution. The quality of the input data is crucial, as a simulation can only be as accurate as the used data allows.

More frequent satellite images are available, e.g., with the AWI ice concentration product, providing an image every 12 hours. It is a passive microwave sensor that yields desired results independent of clouds and other weather conditions, making it a useful tool in real time analysis. However, with a grid of approximately 5km it does not offer a huge improvement in terms of resolution. The aim of this work is to formulate and implement a mathematical model to automatically and reliably determine fast ice on a sequence of such satellite images. The ultimate goal would be to establish an autonomous procedure, not needing any human verification. In this thesis we will look at a new method for fast ice detection. How accurate can these new stochastic approaches get?

The structure of the thesis reflects the work process. Chapter 1 starts with the introduction of basic vocabulary and a general concept of fast ice detection. We move on to simulated data points to illustrate many of the practical challenges. The main problem of this thesis is stated and put into a mathematical framework. In the chapter 2 the main mathematical concept is presented: *Bayesian Analysis* in the context of *Monte Carlo Simulations*. In the appendix D a different approach based on *Stochastical Programming* is looked at. Following these two methods, chapter 3 gives a numerical comparison in terms of their performance in detecting the fast ice. Chapter 4 discusses further ideas and different concepts, and finally chapter 5 concludes this thesis with a summary.

1. Fast Ice Detection

Beginning in autumn, fast ice forms along coastlines. Sea ice gets pushed toward land by onshore winds. There, it remains attached to the shore, frozen up and/or pinned down to the seabed until spring, when it slowly unfreezes and breaks up. Other types of fast ice are glacier tongues and the sea ice thats surrounds them, fixing them in place; in rarer cases, icebergs can ground and become fast ice without any connection to land. In Antarctica the phenomenon of grounded icebergs - either on their own or as anchors for landfast ice - can be found in waters up to 500m deep. [18].

The general classification of fast ice is a bit loose, so we will stick to the one made in the paper mentioned in the introduction: "We classify sea ice as landfast ice when the biweekly average sea ice drift velocity is below a critical value of $5 \times 10^{-4}\,\mathrm{m\,s^{-1}}$ [..] This corresponds to a displacement of approximately 600 m in 2 weeks." [16]. More important than the amount of *movement* allowed is the number of days that sea ice must remain stationary to be considered fast ice. Here, we take the suggested 14 days, for landfast and other types of fast ice.

1.1. **General Idea of Fast Ice Detection.** There is not a lot of work on fully automatic fast ice detection, almost everything relies on human verification / support. A recent paper builds upon a drift algorithm established for sea ice drift detection and simply classifies fast ice as non-moving pixels [24]. A different and more complex attempt is made by classifying the fast ice based on its microwave characteristics [28]. Both discuss shortcomings of their methodology, the latter one citing the close to indistinguishable difference of fast ice to thin ice and the complicating effects of atmospheric conditions. In the drift-ice detection approach, the results depend solely on the output of the algorithm on two consecutive images without taking into account its uncertainty and the benefit of combining time-dependent information. This is where we are addressing the problem.

Let us go over the general idea of automatic fast ice detection that is followed in this thesis. The input is satellite data given in a timeline and as a first step a drift detection algorithm is used to determine the movement of the ice. Depending on the attributes of the data there are a lot of different algorithms to choose from. The movement vectorfield produced by the drift ice algorithm is assumed to have a form of likelihood attached to it, along with a land mask. This result can than be used to determine the fast ice, from now on the 'data' refers to this movement vectorfield or their corresponding absolute value-field (see figure 2 for an example). There is no fixed way to tackle the problem of classifying the fast ice given such a vectorfield. The simplest one is to assume that the drift ice is identified (almost) perfectly. The necessary approach would then be to check each pixel in the timeline to see whether it has moved or not and mark it as fast ice in a suitable timeframe. In practice this assumption is clearly not the case (as shortly discussed in 1.3), hence we need some kind of data reconstruction or a model which accounts for incorrect or missing data. Assuming there is a method to reconstruct or correct wrong data points, it would be plausible to add this method into the drift algorithm itself[1]. Thus, there should be no such method, if we assume our (in practice used) drift algorithm to be optimized. This leads to the approach of building a model that handles incorrect or missing data points.

As an exception to this derivation there is a certain uniform and biased error along the coastlines of our data which we want to account for. This error is due to the overlap of (possibly) moving sea ice and a static coast region. We now cover the data we use as reference.

[1]As an example: one could look at the flow of the drift ice and use a flow field correction algorithm on the received vectors. One such approach is described in [7]. Here our initial guess of the flow field is the pre-computed one.

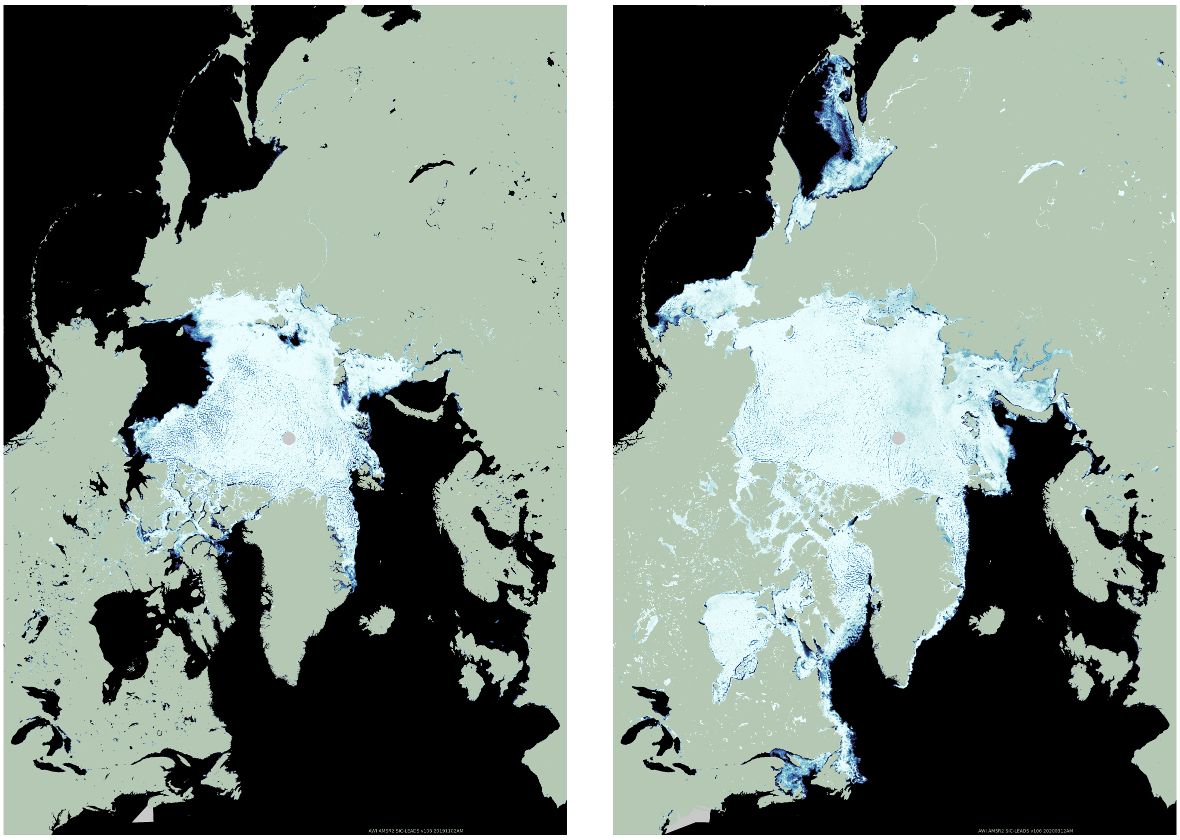


FIGURE 1. Images of original AMSR2 data. Shown is the whole arctic with computed sea ice concentration. the left image corresponds to the data from 03.11.2019, the right is from 12.03.2020.

1.2. **Description of the Original Data Set.** The images originally intended for use are from the AWI AMSR2 data product (see Cooperations). The data is created from the *Advanced Microwave Scanning Radiometer 2* (AMSR2) provided by the *Japan Aerospace Exploration Agency's* (JAXA) and provide an upsampled resolution of $3125 \frac{\text{m}}{\text{pixel}}$, , corresponding to a grid about 5000m in size. In the product the ice concentration is depicted with enhanced ice-leads, these are water or newly formed channels of ice in between older sea icesea ice. It also includes a landmask. [27][22]. This allows for easy land classification and better image correlation as the ice leads offer nice structural points. Two times a day an image is produced. There are rarely any time lags.

Weather changes and melting in the summer season can cause errors and a lot of noise in the product, especially along the coastlines. Example images are shown in figure 1.

1.3. **Simulated Data Points.** To overcome issues like data size, complicated adjustments (e.g., the incorporation of the true fast ice) and a way to reliably control the outcomes, the fast ice and the speed of the ice will be simulated. Notice only the speed is simulated not the direction, a likelihood of data is simulated with a threshold of either given data or a lack of data points. In detecting fast ice the direction of movement is way less important since the main feature is the absence of any movement to distinguish it from any other ice. Especially thinking about boundaries (e.g., for thresholding) one needs to look at the speed. The raw data has no difference from fast to non-fast ice (see figure 2 below). Furthermore, only sea ice will be modeled. Pixels where no ice is *available* (e.g., black to blue areas in original data are neglected (see figure 1).

Reference Set. As reference for this simulation a correlation algorithm is used to detect movement on the AMSR2 product. The yielded vectorfield is then transformed into a velocity field as described above. The basis of the used algorithm is given by *fast normalized cross correlation* (FNCC) [15]. Two ascending images are cut into pre-defined chunks and then each chunk from the first image is fitted into the area around it in the second image, while taking the most likely *path* the chunk could've taken. This is iterated over the whole first image and followed by further refining steps which include filtering vectors and outliers as well as computing all kinds of meta data (e.g., divergence, curl, shearing, etc.). A full description of the procedure is found in this paper: [24]. It is adapted to our products specifics and a simplified, as we do not need many features. Running this adapted algorithm on a reduced region in the Laptev Sea gives a good impression of the velocity fields to simulate and also shows the great error rate in certain areas (see figure 2).

The Simulation. The simulation of the speed relies on the use of Perlin noise, a 'homogeneous' noise developed for mimicking the randomness of nature [20]. It allows (given the right parameters) for smooth transitions in a randomly generated grid. An implementation and in depth description of Perlin noise is found in appendix A.1. to achieve a reasonable product the simulated data should include the following important aspects:

(i) Changing fast ice cover, extending and shrinking with time.
(ii) Changing velocities around the fast ice, simulating the drift ice.
(iii) Noise and missing data with local coherence (e.g., replicate effects of rain).
(iv) Land-mass with strong noise / few information along the coastlines.

To stay consistent the fast ice should be the most concentrated along the coastlines and less frequent the further we go out to sea. A description and implementation of the simulation methods is found in appendix A.2. With recommended parameters from A.2 we achieve results close to what the raw data (1.3) would yield using the mentioned FNCC algorithm (see figure 3). This simulation is representing a local area of ice sheets, meaning all the ice freezes and breaks together. A practical comparison would be to split an actual satellite image into chunks of known fast ice regions, with each region having their own characteristic (melting / freezing time, land connection, etc.). For this reason some of these are a bit randomized for different scenarios (e.g., the freezing time starts at t_0 sampled from a normal distribution).

1.4. **Mathematical Framing of the Problem.**

Given T total observations of velocities, the goal is to deduce the fast ice. Throughout, the *observations* are referred as $Y_{0:T}$ and the final choices of fast ice $X_{0:T}$ are called *decisions* or *hidden process* if seen as the true data. Both lie in the space of real matrices

$$Y_t \in Mat_{m\times n}(\mathbb{R})$$
$$X_t \in Mat_{m\times n}(\mathbb{R}_+) \text{ or } Mat_{m\times n}(\{0,1\})$$

with $n, m \in \mathbb{N}$. The procedure is stepwise, so for every $t \leq T$ a new observation Y_t is made and then decisions can be made (multiple or just one), so in that point in time future observations are unknown. The other unknown is the regular noise and lack of data points in the observations. Everything else about the structure of the problem is adapted in and onto the upcoming models, this includes time dependencies and simplifying properties of distributions.

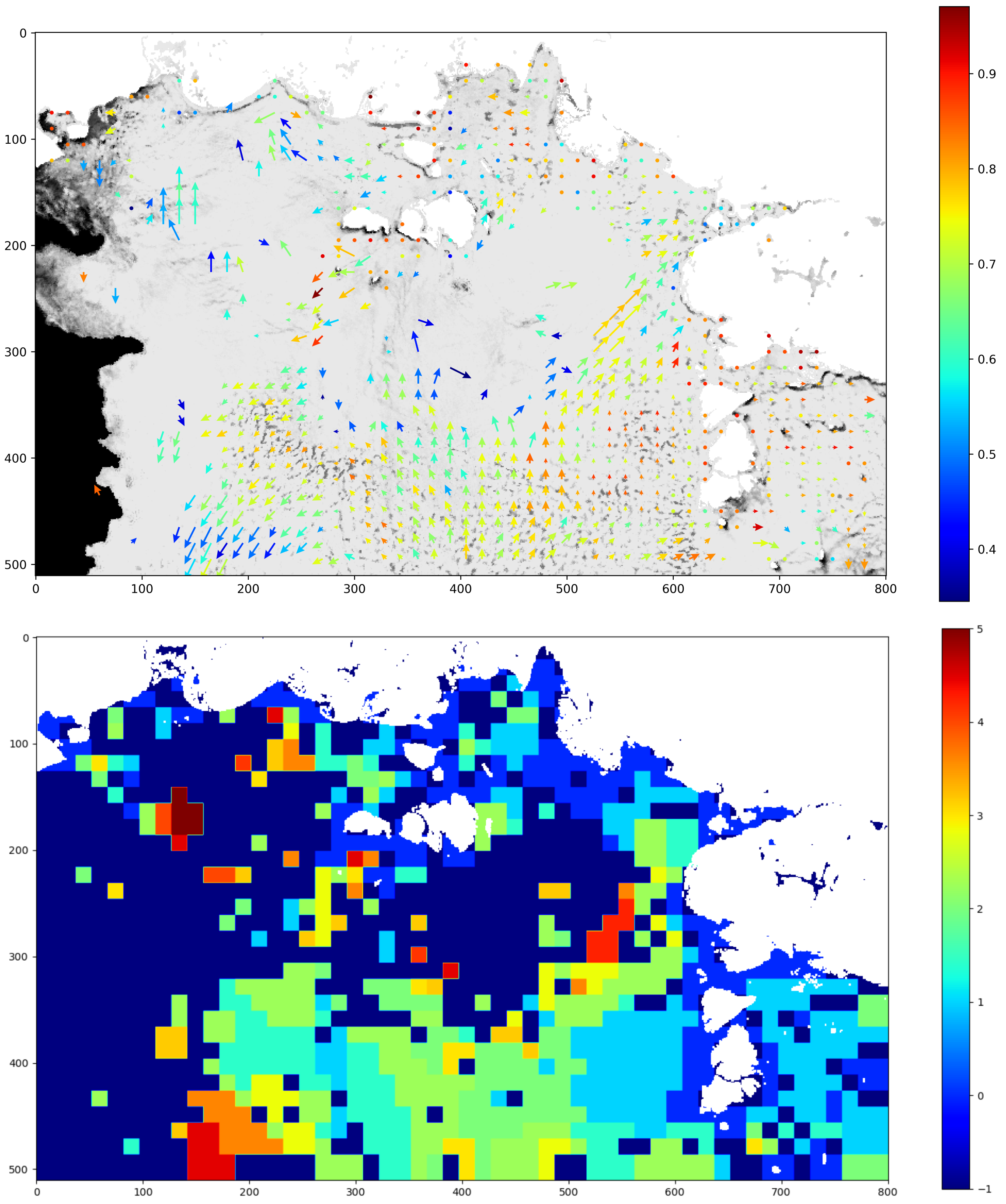


FIGURE 2. This figure shows a result of the drift algorithm, in the upper image with enhanced vector length for better visualization and in the bottom image with a computed velocity field. **BEWARE**, the colors don't match in these images, in the top one the color represents the *likelihood* of the given vector, whereas is the bottom one the color shows the speed in pixels. It is noticeable that in the regions of *clearer* ice (less structures) there is a deficit of data points caused by the almost homogeneous texture, as seen on the left and in the center. These uniform areas are mostly new sea ice, whereas older ice is a lot more spotted due to leads. Also notice the non-moving line along the coast in the lower image. This is a composition of fast ice, the minimum filter mentioned in section 1.2 and a less reliable output of the drift algorithm in coastal regions due to less information. See section 1.3 (*Reference Set*) for a description of the procedure to compute such images.

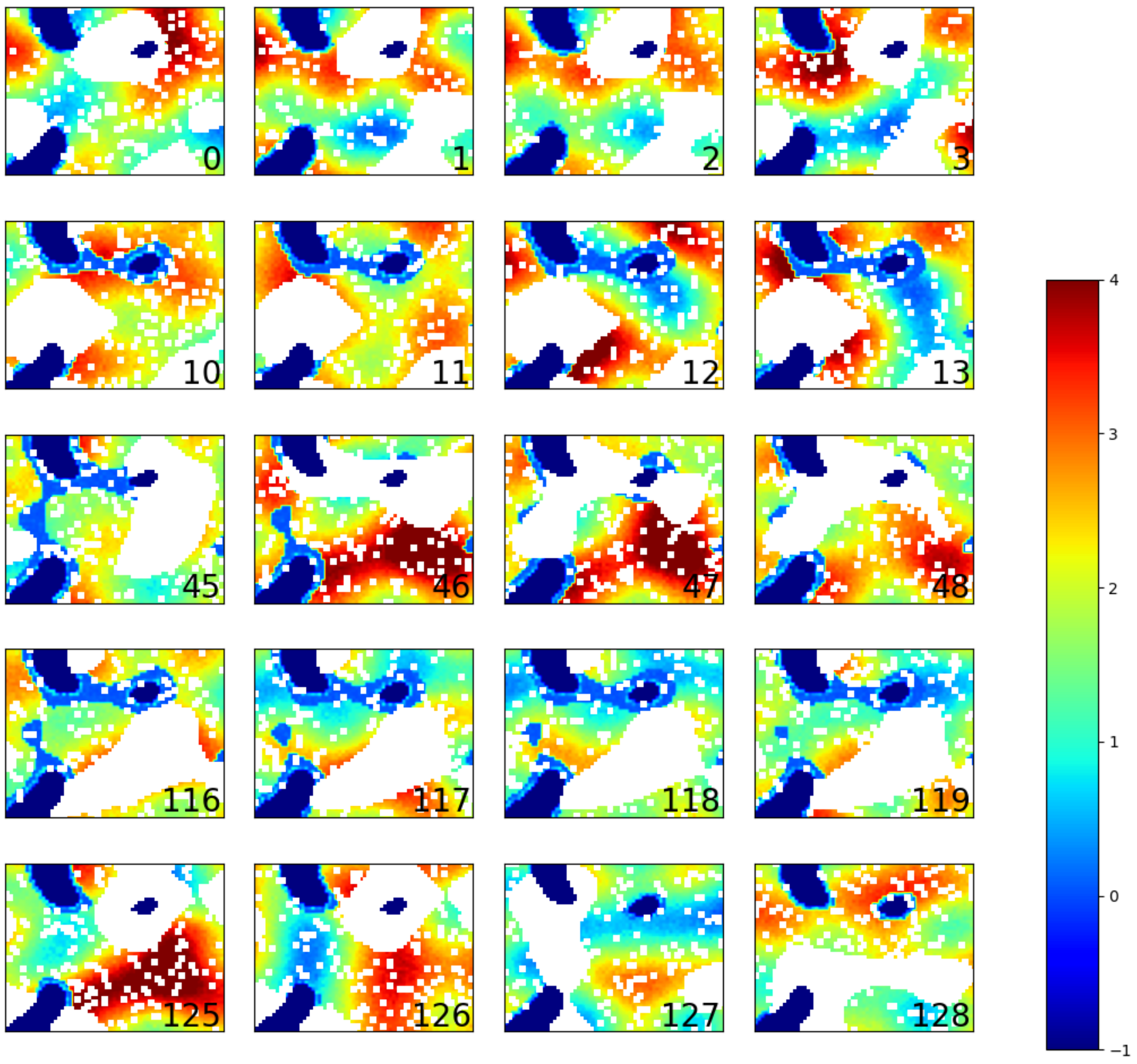


FIGURE 3. Timeline of a generated simulation (Seed=24) where the increase and decrease of fast ice is visible (the cycle length is 150 steps). The more dark and reddish the color is the faster the *measured* velocity, regular blue is the (close to not moving) fast ice. Extra colors are **dark** blue and white displaying land and errors respectively. Notice the stronger noise along the coastlines, making it very challenging to tell whether it is fast ice or not without looking at the neighbours.

1.5. **The Different Approaches.** Having introduced the problem and its formalities the next step is to take a look at some approaches to solve our fast ice problem. The main task is to make a decision taking all available information into account, but most importantly respecting the great amount of uncertainty. In this framework many approaches have been introduced differing in how they model this uncertainty.

The first of the two ideas is called *Stochastic Programming*, a way of modeling uncertainty in the future through minimizing a cost function and an expected value of an upcoming event. Unfortunately, this approach ended up being not too applicable and a linear implementation coincides with a weighted average. All the details can be found in appendix E.

A more promising approach is using *Monte Carlo Simulation*, it attempts to reconstruct a hidden state (the fast ice) through clever sampling possible scenarios. The underlying mathematical idea is Bayesian analysis, which is introduced in the upcoming chapter 2.

2. Approach Two: Bayesian Analysis

Bayesian Analysis works with the principal of conditional probabilities, having two distinct events y and x one can relate them in the following way:

$$p\,(x \,|\, y) = \frac{p\,(y \,|\, x)\; p(x)}{p(y)} \tag{2.1}$$

In many cases the y is seen as a fixed random variable, allowing to write a proportionality instead of an equality in the equation (2.1)

$$p\,(x \,|\, y) \propto p\,(y \,|\, x)\; p(x) \tag{2.2}$$

Exactly this relationship in (2.2) is what we try to exploit. The y models the observation and when observed they stay fixed, whereas the x is a random variable representing the *true* fast ice distribution. The probability of the fast ice given the observations is the quantity needed, or to be more exact the expected valued of the fast ice given the observations, is of interest to us. To come up with an explicit solution or an approximation for this expected value is out of reach, instead the Bayesian approach rephrases the problem to where the order of events makes sense. It is more natural to formulate a relationship from *true* data into a noisy and error afflicted observation than the other way around. Still, this is not obvious and needs some kind of mathematical model to not only describe this relationship between the x and y, but also to compute the values of $p\,(y \,|\, x)$ and $p(x)$, which is not trivial.

A well established method is to use Monte Carlos Simulations within the framework of State-Space models. Both are described in the book *An Introduction to Sequential Monte Carlo* by *N. Chopin and O. Papaspiliopoulos, 2020* [4] which serves as reference for the next chapters presenting these concepts.

2.1. State Space Models and Markov Processes.

We start by introducing State-Space models, the underlying concept for this approach. The usage of State-Space models is wide spread through different fields of science and the formalities differ slightly from one to another. Universally, a State-Space model is made up of two random sequences $(X_t)_t$, $(Y_t)_t$ in their respective (Euclidean) spaces $\mathcal{X}$, $\mathcal{Y}$ with a relation shown in figure 4. The X_t represent an underlying *Markov* process where in every step t only the direct ancestor X_{t-1} influences the upcoming variable X_t. The Y_t are a kind of *noisy* abstraction of X_t often observed in practice (true fast ice and their satellite images, or in our case, the velocity values).

Definition 1 (Markov Property)**.** A sequence $(X_t)_t$ of random variables (r.v.) has the *Markov* property when the dependence only lies between two consecutive variables.

$$p_t\,(X_t \,|\, X_{1:t-1}) = p_t\,(X_t \,|\, X_{t-1})$$

p_t is the probability for X_t in step t (Recall the notation for $X_{1:t-1} = (X_1, \ldots, X_{t-1})$).

2.1.1. *Probability Kernels.*

To adequately describe the mathematical relationships in State Space models a more rigorous background is in need. Having two, potentially equal measure spaces $(\mathcal{X}, \mathcal{B}(\mathcal{X}))$, $(\mathcal{Y}, \mathcal{B}(\mathcal{Y}))$ ($\mathcal{B}(\cdot)$ being the σ-algebra) we define the *probability* and the *backwards kernel.*

Definition 2 (Probability Kernel)**.** A function $P(x, dy) : (\mathcal{X}, \mathcal{B}(\mathcal{Y})) \to [0, 1]$ is called a *probability kernel* from $(\mathcal{X}, \mathcal{B}(\mathcal{X}))$ to $(\mathcal{Y}, \mathcal{B}(\mathcal{Y}))$, if

(i) $\forall x \in \mathcal{X}$, $P(x, \cdot)$ is a probability measure on $(\mathcal{Y}, \mathcal{B}(\mathcal{Y}))$,
(ii) $\forall A \in \mathcal{B}(\mathcal{X})$, $P(\cdot, A)$ is a measurable function in $\mathcal{X}$.

The kernels allow for a nice way of writing conditional distributions. Assume $\mathbb{P}_0(dx)$ is a probability measure on $(\mathcal{X}, \mathcal{B}(\mathcal{X}))$ and $P_1(x_0, dx_1)$ a probability kernel from $(\mathcal{X}, \mathcal{B}(\mathcal{X}))$ to $(\mathcal{X}, \mathcal{B}(\mathcal{X}))$, then the defined

$$\mathbb{P}_1(dx_{0:1}) \coloneqq \mathbb{P}_0(dx_0) P_1(x_0, dx_1) \tag{2.3}$$

is again a probability measure on the product space $(\mathcal{X}^2, \mathcal{B}(\mathcal{X})^2)$, (Recall, dx is the notation for a subset of $\mathcal{X}$). More important now, if we have a r.v. $X_{0:1}$ that is distributed according to $\mathbb{P}_1$, then the conditional distribution of X_1 for $X_0 = x_0$ is exactly given by $P_1(x_0, \cdot)$, proof is given in C.1[2].

In the same way we split the distribution of $X_{0:1}$ into the marginal of X_0 and the conditional part we could also decompose it in the other *backwards* order. Starting with the distribution of $\mathbb{P}_1(dx_1)$ we define the backwards kernel $\overleftarrow{P}_0$ via

$$\mathbb{P}_0(dx_0) P_1(x_0, dx_1) = \mathbb{P}_1(dx_1) \overleftarrow{P}_0(x_1, dx_0) \tag{2.4}$$

If now $\mathbb{P}_1(dx_1)$ dominates $P_1(x_0, dx_1)$ the backwards kernel can be written as a change of measure

$$\overleftarrow{P}_0(x_1, dx_0) = \frac{P_1(x_0, dx_1)}{\mathbb{P}_1(dx_1)} \mathbb{P}_0(dx_0). \tag{2.5}$$

Change of measure implies that $\overleftarrow{P}_0$ is defined through

$$\int_A \mathrm{d}\overleftarrow{P}_0(x_1, dx_0) = \int_A \frac{P_1(x_0, dx_1)}{\mathbb{P}_1(dx_1)} \mathrm{d}\mathbb{P}_0(dx_0), \quad A \subset \mathcal{X} \tag{2.6}$$

2.1.2. *From Markov Processes to State Space Models.* After introducing the probability kernels a Markov process can be defined through the chaining of such.

Definition 3 (Definition and Lemma, Markov Process)**.** Let $X_{0:T}$ be a sequence of r.v. distributed after

$$\mathbb{P}_T(X_{0:T} \in dx_{0:T}) = \mathbb{P}_0(dx_0) \prod_{s=1}^{T} P_s(x_{s-1}, dx_s), \tag{2.7}$$

then $X_{0:T}$ is called a *Markov process* and fulfills the Markov property stated in definition 1 with the kernels being the conditional probabilities

$$\mathbb{P}_T(X_t \in dx_t | X_{0:t-1} \in dx_{0:t-1}) = \mathbb{P}_T(X_t \in dx_t | X_{t-1} \in dx_{t-1}) = P_t(x_{t-1}, dx_t). \tag{2.8}$$

Proof. Works in the fashion of proof C.1 (see above or appendix), but instead of $T = 1$, one splits recursively the distribution into

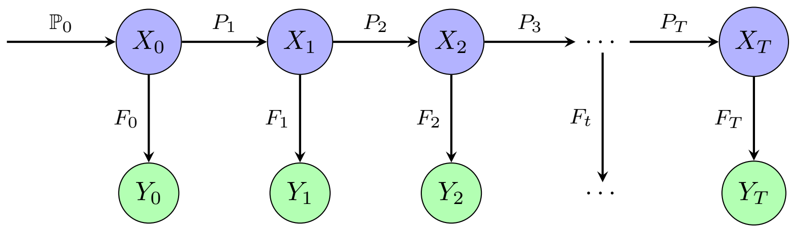
$$\mathbb{P}_T(X_{0:T} \in dx_{0:T}) = \mathbb{P}_{T-1}(X_{0:T-1} \in dx_{0:T-1}) P_T(x_{T-1}, dx_T)$$
$$\mathbb{P}_t(X_{0:t} \in dx_{0:t}) = \mathbb{P}_{t-1}(X_{0:t-1} \in dx_{0:t-1}) P_t(x_{t-1}, dx_t), \quad t = 1, \dots, T-1$$

□

From here, some useful properties can be deduced, by e.g., integrating (2.7) over $x_{t+1:T}$ we get

$$\mathbb{P}_T(dx_{0:t}) = \mathbb{P}_t(dx_{0:t}). \tag{2.9}$$

[2] A little side note: if a prove is given here it is either (i) not given in the book or (ii) unclear in the way it is presented (needs extra explanation or steps). Else there is always a notation to where exactly find a prove of a given statement.



$$\begin{tikzcd} {} \arrow[r, "\mathbb{P}_0"] & X_0 \arrow[r, "P_1"] \arrow[d, "F_0"'] & X_1 \arrow[r, "P_2"] \arrow[d, "F_1"'] & X_2 \arrow[r, "P_3"] \arrow[d, "F_2"'] & \cdots \arrow[d, "F_t"'] \arrow[r, "P_T"] & X_T \arrow[d, "F_T"'] \\ & Y_0 & Y_1 & Y_2 & \cdots & Y_T \end{tikzcd}$$

FIGURE 4. Visual representation of the process model.

Using this and the backwards kernel presents us with the other direction

$$\mathbb{P}_T(X_{0:T} \in dx_{0:T}) = \mathbb{P}_T(dx_T) \prod_{s=1}^{T} \overleftarrow{P}_{T-s}(x_{T-s+1}, dx_{T-s}) \tag{2.10}$$

where $\overleftarrow{P}_t$ is defined as (2.4), but with distribution $\mathbb{P}_T$ and time step t.

Stopping here would allow us to describe the evolution of the *true* fast ice $X_{0:T}$, but we are interested in adding the observations to our model. To do so, we look at the combined stochastic process of $\{(X_t, Y_t)_t\}$ with starting distribution $\mathbb{P}_0(dx_0)$ and transitional kernels $P_t(x_{t-1}, dx_t)$ and $F_t(x_t, dy_t)$ for the conditional distributions X_t given $X_{t-1} = x_{t-1}$ and Y_t given $X = x_t$ respectively. The joint distribution is then computed as

$$\begin{aligned} \mathbb{P}_T(X_{0:T} \in dx_{0:T}, Y_{0:T} \in dy_{0:T}) &= \mathbb{P}_0(dx_0) \prod_{t=1}^{T} P_t(x_{t-1}, dx_t) \prod_{t=0}^{T} F_t(x_t, dy_t) \\ &= \mathbb{P}_T(dx_{0:T}) \prod_{t=0}^{T} F_t(x_t, dy_t), \end{aligned} \tag{2.11}$$

here the term $\mathbb{P}_T(dx_{0:T})$ can be seen as a marginal Markov process, a visualization is seen in figure 4.

In many, but certainly not all practical cases, the transition kernels provide densities (see examples of chapter 2 in [4]). It is assumed (and in our case given) that for the kernel F a so-called *transition density* exists, relative to some measure $\nu(dy)$:

$$F_t(x_t, dy_t) = f_t(y_t|x_t)\nu(dy_t), \quad \forall t. \tag{2.12}$$

Inserting this into (2.11) yields

$$\mathbb{P}_T(X_{0:T} \in dx_{0:T}, Y_{0:T} \in dy_{0:T}) = \mathbb{P}_T(dx_{0:T}) \prod_{t=0}^{T} f_t(y_t|x_t) \prod_{t=0}^{T} \nu(dy_t). \tag{2.13}$$

This expression (2.13) encompasses all the parts used in the upcoming chapters.

2.1.3. *First Equations.*

From above one can conclude further relations such as marginal distributions. Some are now derived and presented. A direct way to compute the distribution of $Y_{0:t}$ is to integrate

(2.13) (until step t instead of T, since this holds for all $t \leq T$) over $X_{0:t}$

$$\begin{aligned}
\mathbb{P}_t(Y_{0:t} \in dy_{0:t}) &= \int_{\mathcal{X}^t} \mathbb{P}_T(x_{0:t}, Y_{0:t} \in dy_{0:t})\,\mathrm{d}x_{0:t} \\
\text{use (2.13)} \rightarrow \quad &= \left[\prod_{s=0}^{t} \nu(dy_s)\right] \int_{\mathcal{X}^t} \left[\prod_{s=0}^{t} f_t(y_s|x_s)\right] \mathbb{P}_t(x_{0:t})\,\mathrm{d}x_{0:t} \\
\text{Definition of expected value} \rightarrow \quad &= \mathbb{E}_{\mathbb{P}_t}\left[\prod_{s=0}^{t} f_s(y_s|X_s)\right] \prod_{s=0}^{t} \nu(dy_s)
\end{aligned} \tag{2.14}$$

This distribution (2.14) has a density (with respect to ν) called the *likelihood*, denoted as

$$p_t(y_{0:t}) \coloneqq \mathbb{E}_{\mathbb{P}_t}\left[\prod_{s=0}^{t} f_s(y_s|X_s)\right]. \tag{2.15}$$

This leads to the *likelihood factors.*

Definition 4 (Definition and Lemma, Likelihood Factors)**.** The conditional densities

$$p_t\left(y_t \mid y_{0:t-1}\right) = \frac{p_t(y_{0:t})}{p_{t-1}(y_{0:t-1})} \tag{2.16}$$

are called the *likelihood factors.*

Proof (of the equality). C.2 □

Thinking back to our starting point of the Bayesian approach, we can now formulate a way of computing the value of interest, X given Y.

$$\begin{aligned}
\mathbb{P}_t(X_{0:t} \in dx_{0:t} \mid Y_{0:t} \in y_{0:t}) &= \frac{\mathbb{P}_t(X_{0:t} \in dx_{0:t},\, Y_{0:t} \in y_{0:t})}{p_t(y_{0:t}) \prod_{s=0}^{t} \nu(dy_s)} \\
&= \frac{1}{p_t(y_{0:t})}\left[\prod_{s=0}^{t} f_s(y_s|x_s)\right] \mathbb{P}_t(dx_{0:t})
\end{aligned} \tag{2.17}$$

2.1.4. *A Small Recap.* Before going over into *Feyman-Kac models*, a more convenient and comprehensive mathematical framework, here a short recap of the previous calculations and an outlook on the goals. Recall, the idea is to compute the fast ice distribution given some observations we make. To do so, a reformulation of the problem is needed as a direct calculation of this quantity is more or less impossible. Here the Bayes' theorem is utilized, yielding the relation (2.17). It is described through an initial distribution (guess) $\mathbb{P}_0$ of the fast ice distribution and then probability kernels P_t to describe the evolution of it. In every time step there is a transition density f_t that specifies the relationship between the fast ice and the velocity observations we make. Two values are of most interest; *filtering* and *smoothing*, the prior describes the distribution of the **newest** fast ice given **all** the past data and the latter is the distribution of **some to all** fast ice given **all** the observations.

- **filtering**: $\mathbb{P}_t(X_t \in dx_t \mid Y_{0:t} \in dy_{0:t}), \quad t \leq T$
- **smoothing**: $\mathbb{P}_T(X_{s:t} \in dx_{s:t} \mid Y_{0:T} \in dy_{0:T}), \quad 0 \leq s \leq t \leq T$
- **likelihood**: $p_t(y_{0:t})$

There are more quantities of concern that can be derived using state space models (e.g., state prediction) but they are of no importance to our problem.

2.2. **Feyman-Kac Models.** Before entering the abstract world of *Feyman-Kac* models we will shortly talk about the idea of how to actually compute any of the named categories (filtering, smoothing, etc.). This should also give some insights into why we chose this model in the beginning.

The most important distribution is $\mathbb{P}_t(X_{0:t} \in dx_{0:t})$, which can be computed recursively with (2.7). This equation allows for particle trajectories, they start by sampling initial particles (in our case fast ice distributions) from $\mathbb{P}_0$ and then evolve them corresponding to the kernels $P_{1:t}$. If now an expected value is required (as in the likelihood densities or the filtering solutions; we are more interested in the expected value of X_t), N such trajectories are generated to approximate these via *Monte Carlo*

$$\mathbb{E}_{\mathbb{P}_t}\left[\phi(X_{0:T})\right] \approx \frac{1}{N}\sum_{i=1}^{N} \phi\left(X_{0:T}^{(i)}\right).$$

How to sample and optimize this method in terms of variance and convergence is a big discussion in and of itself, hence we will only cover the main ideas.

Coming back to *Feyman-Kac* models, they provide a theoretical basis to simulate random paths for stochastic processes and can be viewed as the underlying structure. So, after translating a process model into this mathematical frame different algorithms can be easily obtained and applied. Initially it looks *identically* to our already defined Markov process (2.7) for $X_{0:T}$, including a starting distribution $\mathbb{M}_0$ and transition kernels M_t

$$\mathbb{M}_T(dx_{0:T}) = \mathbb{M}_0(dx_0)\prod_{t=1}^{T} M_t(x_{t-1}, dx_t). \tag{2.18}$$

Now, a resemblance of the transition densities f_t is added. Define the potential functions

$$G_0 : \mathcal{X} \to \mathbb{R}^+$$
$$G_t : \mathcal{X}^2 \to \mathbb{R}^+.$$

Definition 5 (Feyman-Kac)**.** A *Feyman-Kac* model is defined through the functions G_0, G_s, the distribution $\mathbb{M}_0$ and the kernels M_s, $s = 1, \ldots, t$ and is represented as a change of measure from $\mathbb{M}_t$

$$\mathbb{Q}_t(dx_{0:t}) = \frac{1}{L_t} G_0(x_0)\left\{\prod_{s=1}^{t} G_s(x_{s-1}, x_x)\right\} \mathbb{M}_t(dx_{0:t}). \tag{2.19}$$

L_t is the normalizing constant for Q_t and can be obtained by integrating over the potential functions

$$L_t = \mathbb{E}_{\mathbb{M}_t}\left[G_0(X_0)\prod_{s=1}^{t} G_s(x_{s-1}, x_x)\right]. \tag{2.20}$$

The ratios of normalizing constants are denoted as

$$l_t \coloneqq L_t / L_{t-1}. \tag{2.21}$$

Two important classes of Feyman-Kac models, in terms of application to state space models, will be presented now.

2.2.1. *Bootstrap and Guided Model.* In the last chapter we defined a Feyman-Kac model through $\mathbb{M}_0$, G_0, M_t and G_t. Hence, different Feyman-Kac formalisms depend on the way these parameters are chosen. In practice one could image they describe *how a particle is chosen in its evolutionary trajectory* (in reference to the starting explanation of this chapter 2.2). One of such formalisms is the *Bootstrap.*

Definition 6 (Boostrap)**.** Given a state space model as in (2.11), so initial distribution $\mathbb{P}_0(dx_0)$, transition kernels $P_t(x_{t-1}, dx_t)$ and transition densities $f_t(y_t \,|\, x_t)$, the *bootstrap* Feyman-Kac formalism is defined through

$$\begin{aligned} \mathbb{M}_0(dx_0) &= \mathbb{P}_0(dx_0), & G_0(x_0) &= f_0(y_0 \,|\, x_0), \\ M_t(x_{t-1}, dx_t) &= P_t(x_{t-1}, dx_t), & G_t(x_{t-1}, x_t) &= f_t(y_t \,|\, x_t). \end{aligned}$$

Plugging in the corresponding formulas from the last chapter yields:

$$\begin{aligned} \mathbb{Q}_t(dx_{0:t}) &= \mathbb{P}_t(X_{0:t} \in dx_{0:t} \,|\, Y_{0:t} = y_{0:t}) \\ L_t &= p_t(y_{0:t}) \\ l_t &= p_t(y_t \,|\, y_{0:t-1}) \end{aligned}$$

Notice the potential functions G_t, $t \leq T$ only depend on x_t and implicit on the observation y_t, which is seen as fixed (after observation). Also, the quantity $\mathbb{Q}_t$ is exactly what we want to compute. Another bigger family is the *guided* Feyman-Kac formalism, it includes the Bootstrap.

Definition 7 (Guided)**.** Given a state space model as in (2.11), so initial distribution $\mathbb{P}_0(dx_0)$, transition kernels $P_t(x_{t-1}, dx_t)$ and transition densities $f_t(y_t \,|\, x_t)$, the *guided* Feyman-Kac formalism is defined through

$$\begin{aligned} G_0(x_0)\, \mathbb{M}_0(dx_0) &= f_0(y_0 \,|\, x_0)\, \mathbb{P}_0(dx_0), \\ G_t(x_{t-1}, x_t)\, M_t(x_{t-1}, dx_t) &= f_t(y_t \,|\, x_t)\, P_t(x_{t-1}, dx_t). \end{aligned}$$

Again, we get

$$\begin{aligned} \mathbb{Q}_t(dx_{0:t}) &= \mathbb{P}_t(X_{0:t} \in dx_{0:t} \,|\, Y_{0:t} = y_{0:t}) \\ L_t &= p_t(y_{0:t}) \\ l_t &= p_t(y_t \,|\, y_{0:t-1}) \end{aligned}$$

Clearly, the bootstrap is the trivial case of the guided formalism. The differences and the advantages of the wider family of guided Feyman-Kac models will be covered in the section of their application 2.4 (mainly in the filtering section 2.4.1). To get to the applications, relations between different steps should be constructed (e.g., $t \to t+1$), this is covered in the next section.

2.2.2. *Recursions.* Before we can state generic algorithms to compute the quantities of interest, the theoretical relations are required. Most of them rely on some kind of recursion. A particularly important one for *filtering* is the relation of $\mathbb{Q}_{t-1}(dx_{t-1})$ and $\mathbb{Q}_t(dx_t)$. Here, the starting point is the change of measure

$$\mathbb{Q}_t(dx_{0:t}) = \frac{1}{l_t} G_t(x_{t-1}, x_t) M_t(x_{t-1}, dx_t)\, \mathbb{Q}_{t-1}(dx_{0:t-1}), \tag{2.22}$$

which directly follows from the definition 5 (equation (2.19)). To end up with the marginal distributions we split apart the last equation (2.22), realise

$$\begin{aligned} \mathbb{Q}_{t-1}(dx_{t-1:t}) &= \int_{\mathcal{X}^{t-2}} M_t(x_{t-1}, dx_t)\, \mathbb{Q}_{t-1}(dx_{0:t-1}), \quad \text{integrating over } x_{0:t-2} \\ &= M_t(x_{t-1}, dx_t)\, \mathbb{Q}_{t-1}(dx_{t-1}). \end{aligned} \tag{2.23}$$

With the same idea one gets

$$\mathbb{Q}_t(dx_{t-1:t}) = \frac{1}{l_t} G_t(x_{t-1}, x_t)\, \mathbb{Q}_{t-1}(dx_{t-1:t}). \tag{2.24}$$

This separation leads to an algorithm enabling us to compute the marginal filtering distributions of $\mathbb{Q}_t$ called *forward recursion* 1.

Algorithm 1 Forward Recursion (Feyman-Kac Formalism)

```
1: Input
2:     Feyman-Kac model: $\mathbb{M}_0$, $G_0$, $M_t$ and $G_t$
3: Initialize
4:     $\mathbb{Q}_{-1}(dx_0) = \mathbb{M}_0(dx_0)$
5: for t = 0, . . . , T do
6:     $\mathbb{Q}_{t-1}(dx_{t-1:t}) = M_t(x_{t-1}, dx_t)\, \mathbb{Q}_{t-1}(dx_{t-1})$        ▷ Extension
7:     $\mathbb{Q}_t(dx_{t-1:t}) = \frac{1}{l_t} G_t(x_{t-1}, x_t)\, \mathbb{Q}_{t-1}(dx_{t-1:t})$        ▷ Change of measure
8:     $\mathbb{Q}_t(dx_t) = \int_{x_{t-1}\in\mathcal{X}} \mathbb{Q}_t(dx_{t-1:t})$        ▷ Marginalization
9: end for
10:
11: $l_0 = L_0 = \int_{\mathcal{X}} G_0(x_0)\, \mathbb{M}_0(dx_0)$
12: $l_t = \frac{L_t}{L_{t-1}} = \int_{\mathcal{X}^2} G_t(x_{t-1}, x_t)\, \mathbb{Q}_{t-1}(dx_{t-1:t})$        ▷ Likelihood values for line 7
```

From here, we are also interested in evaluating the smoothing distribution $\mathbb{Q}_T(dx_{0:T}) = \mathbb{P}(X_{0:T} \in dx_{0:T} \,|\, Y_{0:T} = x_{0:T})$ or marginal smoothing distributions $\mathbb{Q}_T(dx_{s:t}) = \mathbb{P}(X_{s:t} \in dx_{s:t} \,|\, Y_{0:T} = x_{0:T})$, $s \le t$. There are different ideas to do so, the two covered here will first identify the backwards kernels of $\mathbb{Q}_t$, after showing it to be a Markov measure. Then, one strategy is to do a forward recursion to get to $\mathbb{Q}_T(dx_T)$ and use the backwards kernel to generate paths going in reversed direction through the sampled particles. This method is called *Forward Filtering, Backward Smoothing (FFBS)*, figure 5 visualizes this concept. The other approach relies on a *backward* filtering step, essentially the same as a forward filtering but with reversed order of observations. Therefor, its name is *two-filter smoothing.*

To show that $\mathbb{Q}_T$ is a Markov process it is beneficial to impose one more helping function $H_{t:T}(x_t)$ named *cost-to-go* function [3].

$$H_{T:T}(x_T) \coloneqq 1, \tag{2.25}$$

$$H_{t:T}(x_t) \coloneqq \int_{\mathcal{X}^{T-t}} \prod_{s=t+1}^{T} G_s(x_{s-1}, x_s)\, M_s(x_{s-1}, dx_s), \quad t < T.$$

Applying *Tonelli* (G_s and M_s are both positive and so is $H_{s:T}$ for all s) on (2.25) with inserting $H_{t+1:T}$ into $H_{t:T}$ allows us to change the order of integration to yield the recursion

$$H_{t:T}(x_t) = \int_{\mathcal{X}} G_{t+1}(x_t, x_{t+1})\, H_{t+1:T}(x_{t+1})\, M_{t+1}(x_t, dx_{t+1}), \quad t < T. \tag{2.26}$$

In the Bootstrap formalism this quantity is

$$H_{t:T}(x_t) = p_T(y_{t+1:T} \,|\, x_t), \tag{2.27}$$

the likelihood of future observations conditional on the current decision (see appendix C.3 for details).

This lemma shows the transition kernels.

[3]The name *cost-to-go* comes from the field of dynamic programming and is intuitively a bit off in this context, since $H_{t:T}$ represents more a *chance-to-go* value.

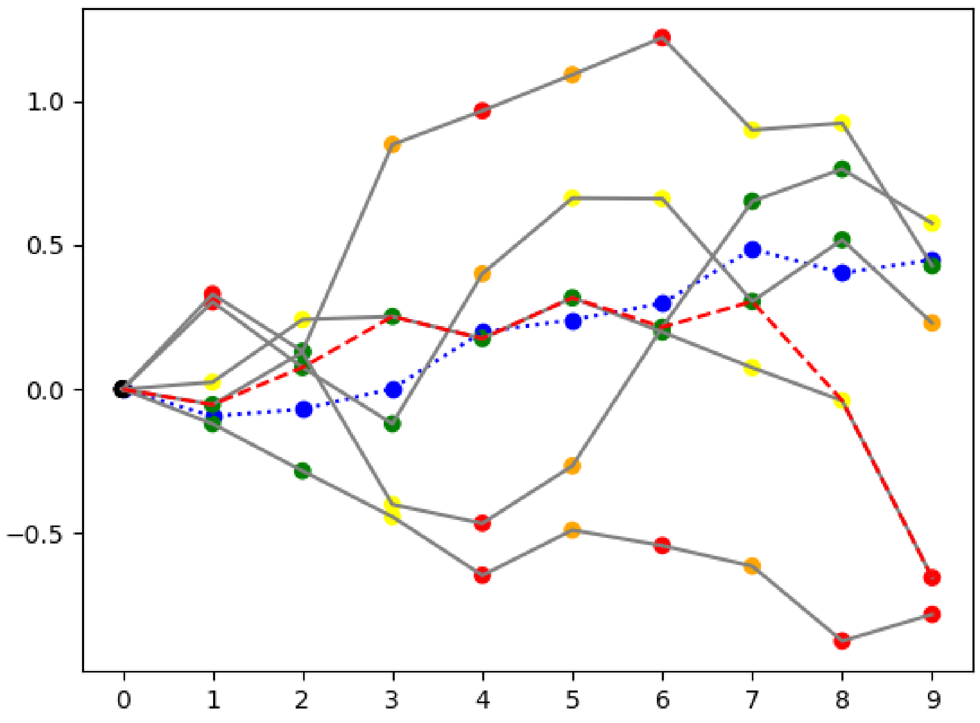


FIGURE 5. Visualizes the concept of *Forward Filtering Backward Smoothing*. Each grey line represents a sampled trajectory. The color scheme of the points represent their weights (corresponding to the distance to the reference in blue; See figure 6 for more details). The red line is a generated path corresponding to a backwards kernel and starts on the right side ($t = 9$). It *chooses* between likelihood of the ancestor and the weight of each particle.

Lemma 8. *$\mathbb{Q}_T$ is a Markov process with initial distribution*

$$\mathbb{Q}_{0|T}(dx_0) = \frac{1}{L_T} G_0(x_0)\, H_{0:T}(x_0)\, \mathbb{M}_0(dx_0) \tag{2.28}$$

and forward/ backward transition kernel

$$Q_{t|T}(x_{t-1}, dx_t) = \frac{H_{t:T}(x_t)}{H_{t-1:T}(x_{t-1})} G_t(x_{t-1}, x_t)\, M_t(x_{t-1}, dx_t), \tag{2.29}$$

$$\mathbb{Q}_T(dx_t)\, \overleftarrow{Q}_{t-1|T}(x_t, dx_{t-1}) = \mathbb{Q}_T(dx_{t-1})\, Q_{t|T}(x_{t-1}, dx_t). \tag{2.30}$$

Proof. Proof is found in [4], page 59. ☐

Remark. Recall, $Q_{t|T}$ indicates the dependence on T.

With lemma 8 and equation (2.9) one can deduce the partial distributions necessary for smoothing.

Lemma 9. *For all $t < T$*

$$\mathbb{Q}_T(dx_{0:t}) = \frac{L_t}{L_T} H_{t:T}(x_t)\, \mathbb{Q}_t(dx_{0:t}), \tag{2.31}$$

integrating over $x_{0:t-1}$ leads to the marginal distributions

$$\mathbb{Q}_T(dx_t) = \frac{L_t}{L_T} H_{t:T}(x_t)\, \mathbb{Q}_t(dx_t). \tag{2.32}$$

Proof. C.4. ☐

At last, to complete the smoothing idea we rewrite the backwards kernel of $\mathbb{Q}$ defined in Lemma 8 utilizing this Lemma 9 to get (see appendix C.5 for computation)

$$\mathbb{Q}_t(dx_t)\, \overleftarrow{Q}_{t-1|t}(x_t, dx_{t-1}) = \frac{1}{l_t} G_t(x_{t-1}, x_t)\, \mathbb{Q}_{t-1}(dx_{t-1})\, M_t(x_{t-1}, dx_t), \tag{2.33}$$

(under the assumption that M_t admits a probability density m_t) that leaves us with

$$\overleftarrow{Q}_{t-1|t}(x_t, dx_{t-1}) \propto G_t(x_{t-1}, x_t)\, m_t(x_t \,|\, x_{t-1})\, \mathbb{Q}_{t-1}(dx_{t-1}) \tag{2.34}$$

(Proof is found in [4], page 62).

This reconstruction gives space to the following idea, start by a forward filtering and save the $\mathbb{Q}_t(dx_t)$, then sample from $\mathbb{Q}_T(dx_T)$ and compute backward trajectories with the backward kernels represented in (2.34) for the smoothing quantity $\mathbb{Q}_T(dx_{0:T})$, it is the earlier described *FFBS* algorithm.

For the *two-filter smoothing* we write out equality (2.32) from Lemma 9 and insert the Bootstrap formalism to end up with

$$\mathbb{P}_T(X_t \in dx_t \,|\, Y_{0:T} = y_{0:t}) = \frac{p_T(y_{t+1:T} \,|\, x_t)}{p_T(y_{t+1:T} \,|\, y_{0:t})} \mathbb{P}(X_t \in dx_t \,|\, Y_{0:t} = y_{0:t}). \tag{2.35}$$

Every part is simply replaced by their Bootstrap relative (using Def. 6 and equation (2.27)). Clearly this represents a change of measure from the filtering distribution with a proportional relation to $p_T(y_{t+1:T} \,|\, x_t)$ (The $p_T(y_{t+1:T} \,|\, y_{0:t})$ in the denominator is not depended on the decisions x_t and can be handled as a normalization constant. Remember the observations are seen as fixed). Now, to obtain this proportionality a backward filtering step is performed. In our case the idea is to follow the freezing and melting process backwards, meaning we start when the ice *un-melts* in summer and end when it *un-freezes* towards the beginning of winter.

2.3. **Sampling and Resampling.** Every filtering or smoothing distribution has to be approximated in some way, since any distribution not given by default can in practice not be computed analytically (e.g., given the starting distribution and the transition kernels, in many practical cases the filtering distribution is intractable). Here, different sampling and resampling methods are used, to obtain a sufficient convergence in terms of exactness and speed. As mentioned in the introduction of section 2.2, the main idea is an approximation with Monte Carlo in what is called *Importance Sampling/Resampling*. Importance sampling is a way of weighting sampled particles according to their fit in a model, so instead of

$$\mathbb{E}_{\mathbb{P}_t}\left[\phi(X)\right] \approx \frac{1}{N}\sum_{i=1}^{N}\phi\left(X^{(i)}\right). \tag{2.36}$$

for $X^{(i)} \sim \mathbb{Q}$ the following expression is evaluated

$$\frac{\sum_{i=1}^{N} w\left(X^{(i)}\right)\phi\left(X^{(i)}\right)}{\sum_{i=1}^{N} w\left(X^{(i)}\right)}, \quad w\left(X^{(i)}\right) \propto \frac{\mathbb{Q}\left(X^{(i)} \in dx\right)}{\mathbb{M}\left(X^{(i)} \in dx\right)}, \quad X^{(i)} \sim \mathbb{M}. \tag{2.37}$$

Here, it is assumed that the Radon-Nikodym derivative of $\mathbb{Q}/\mathbb{M}$ exists. The w is the weighting term and the sum of w in the denominator is for normalization, to shorten the expression we define

$$W^{(i)} = \frac{w\left(X^{(i)}\right)}{\sum_{m=1}^{N} w\left(X^{(m)}\right)} \tag{2.38}$$

The reason for the reformulation (2.37) is a simple identity, when the respective densities m and q exist.

$$\int_{\mathcal{X}} \phi(x)q(x) = \int_{\mathcal{X}} \phi(x)\frac{q(x)}{m(x)}m(x) \tag{2.39}$$

The idea behind it is to sample from $\mathbb{M}$ instead of $\mathbb{Q}$ which can be way easier in most cases. In relation to our problem, it is more convenient to define and sample from the initial distribution $\mathbb{P}_0$ and the transitions kernels P_t defining our $\mathbb{M}_t$ (so to sample some way of fast ice evolution) compared to the conditional distribution $\mathbb{Q}_t$ (the fast ice evolution, conditional on the observations Y_t). As we will see in the section 2.4 describing the algorithms used, the weighting values w are exactly the functions G_t from our Feyman-Kac model.

2.3.1. *Importance Resampling.* Assume you have a decent model for the fast ice evolution and you use importance sampling (or any other sampling method) to generate (independent) trajectories based on this model. The variance of any particle grows with every step, since every transition kernel P_t introduces more. Even with the approach of weighting every particle to *filter out*[4] the ones, which deviate too much it can take a lot of particles to achieve a reasonable convergence.

To accommodate for this, a method called *importance resampling* is utilized. In every new step, instead of sampling from the past states of a particle, the particles are put in a pool along with their weight. Now, N new particles are sampled from this pool of ancestors, according to the weight of the particles. This procedure can reduce the variance of the full sample, but for the cost of introducing dependence along the originally independent trajectories. Figure 6 illustrates the difference.

On the actual sampling methods (what kind of resampling scheme to use and e.g., mc, smc, qsmc, so the choice of how to actually pick a new sample) more in depth information can be found in [4] (Chapters 8, 9 and 13) and will not be covered here.

[4]There is no filtering happening, but the weight gets so small, that the corresponding particles are neglectable

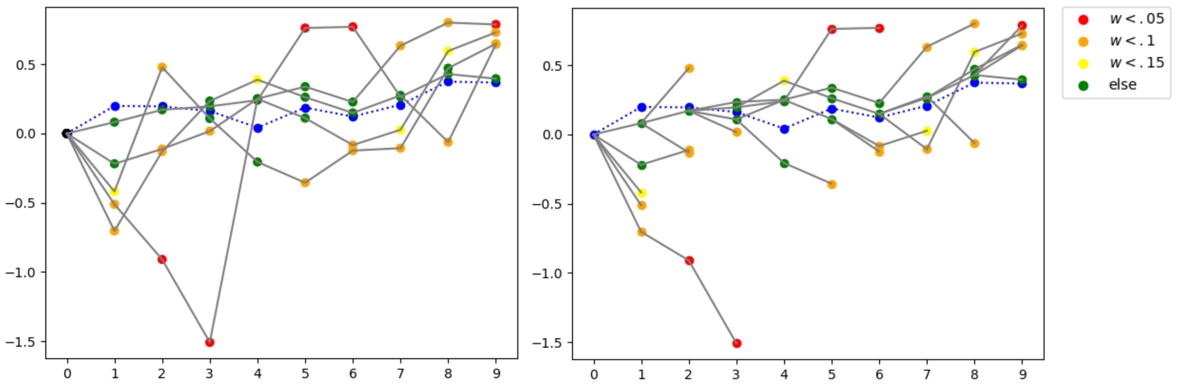


FIGURE 6. Sampling (left) vs. Resampling (right) visualization. The blue line and points are *a reference*. Grey lines connect to corresponding particles sampled, while their color display their given weight $w \propto \frac{1}{\text{distance to reference}}$ with the color coding presented in the top right legend. In every step 5 particles are sampled corresponding to $\mathcal{N}(X_p, \sigma)$, where X_p is the value of the chosen ancestor. Notice, when particles have low weights they are not used to sample the next generation of particles in the resampling scheme.

2.3.2. *When to Resample and the EES.* Resampling is more costly than sampling, since one also has to choose from which particles to sample. Further, the introduction of dependence and the resampling of ancestors can yield to increased variance in past steps. The trade-off is always good and only using importance sampling is not recommended, but a mix of sampling and resampling steps seems to be working well in practice for a more efficient computation time.

The choice of when to resample is not that well studied. An idea is to look at the *effective sample size*, a measure of the efficiency of importance sampling

$$\mathrm{ESS}(W^{(1:N)}) := \frac{1}{\sum_{n=1}^{N} \left(W^{(n)}\right)^2}. \tag{2.40}$$

Clearly, we have the following properties

- $\mathrm{ESS}(W^{(1:N)}) \in [1, N]$
- $\mathrm{ESS}(W^{(1:N)}) \approx N$, for $W^{(i)} \approx W^{(j)}, \quad i, j \leq N$.
- $\mathrm{ESS}(W^{(1:N)}) \to 1$, when there is an $i \leq N$ with $W^{(i)} \to 1$

In general, one can interpret that if $\mathrm{ESS}(W^{(1:N)}) \approx k$, then there are about k particles which are *relevant* (whose weights are not neglectable). Subsequently, when the ESS gets too low (e.g., ESS $< N/2$) one should perform a resampling step to ensure not too many particles degenerate. Take a look at the importance sampling from figure 6 again, time step 2 would be a reasonable point for the first resampling scheme (Even though step 3 is quite good again in this example).

2.4. **Filtering and Smoothing.** Putting everything together, it is now possible to describe the actual filtering and smoothing algorithms. We will use the Feyman-Kac formalisms from chapter 2.2 and their theoretical framework, introduced in section 2.2.2, as well as the sampling/resampling ideas from the last chapter 2.3 for the required approximations.

2.4.1. *Filtering Algorithm.* For the filtering algorithm the goal is to practically implement the already defined algorithm 1 from section 2.2.2. Recall, in any Feyman-Kac model we are given an initial distribution and potential function $\mathbb{M}_0, G_0$ as well as the transition kernels and potential functions $M_{1:T}, G_{1:T}$. The analytically defined functions $\mathbb{Q}_{0:T}$ will be approximated through the procedure of algorithm 1 using Monte Carlo. N particles are sampled from $\mathbb{M}_0$ and weighted according to G_0. Subsequently N new particles are sampled (importance sampling or resampling) using M_t with a weight of G_t. In every step the weight is normalized (this normalization constant corresponds to the likelihood factors L_t (2.20)). Algorithm 2 illustrates the concept with the addition of an adaptive resampling regulated by the *Effective Sample Size* (ESS, introduced in (2.40)).

Algorithm 2 Particle Filtering (with adaptive resampling)

```
 1: Input
 2:     Feyman-Kac model: $\mathbb{M}_0$, $G_0$, $M_t$ and $G_t$,
 3:     Amount of particles: $N$
 4:
 5: $X_0^{1:N} \sim \mathbb{M}_0(dx_0)$                         ▷ N initial samples of our fast ice
 6: $w_0^{1:N} = G_0(X_0^{1:N})$
 7: $W_0^{1:N} = w_0^{1:N} / \sum_m^N w_0^m$                    ▷ Normalizing initial weights
 8:
 9: for t=1, ..., T do                                         ▷ Generating trajectories through time
10:     if ESS($W_{t-1}^{1:N}$) < ESS$_{\min}$ then             ▷ Checking the ESS
11:         $A_t^{1:N} \leftarrow \text{resample}(W_{t-1}^{1:N})$    ▷ Resample step, pick ancestors
12:         $\hat{w}_{t-1}^{1:N} = 1$                            ▷ No recursive weighting
13:     else
14:         $A_t^{1:N} = 1:N$                                    ▷ Importance Sampling
15:         $\hat{w}_{t-1}^{1:N} = w_{t-1}^{1:N}$                ▷ Keep weights for recursive weighting
16:     end if
17:     $X_t^n \sim M_t(X_{t-1}^{A_t^n}, dx_t)$                  ▷ Sample successor
18:     $w_t^{1:N} = \hat{w}_{t-1}^{1:N}\, G_t(X_{t-1}^{A_t^{1:N}}, X_t^{1:N})$   ▷ Compute weights
19:     $W_t^{1:N} = w_t^{1:N} / \sum_m^N w_t^m$                 ▷ Normalizing weights
20: end for
```

The outputs of algorithm 2 are:

$$\mathbb{E}\left[\mathbb{Q}_t(X_t)\right] = \sum_{n=1}^{N} W_t^n \phi(X_t^n), \quad \textbf{Filtering} \tag{2.41}$$

$$l_t^N = \begin{cases} \frac{1}{N}\sum_{n=1}^N w_t^n, & \text{if resampled in step t} \\ \frac{\sum_{n=1}^N w_t^n}{\sum_{n=1}^N w_{t-1}^n}, & \text{else} \end{cases} , \quad \textbf{Likelihood f.} \tag{2.42}$$

Applying this algorithm to a state space model with distributions $\mathbb{P}_{0:T}$ and transition kernels $p_{1:T}$ and $f_{0:T}$ is fairly straight forward in the Bootstrap formalism, simply substitute the corresponding functions. More interesting is the Guided Feyman-Kac model. Here, we first have to define our $\mathbb{M}_0, M_{1:T}$ ($G_{0:T}$ is then defined with the equations from definition 7). But why would it be preferable to first define and compute a new starting distribution, transition kernels and weighting functions?

In every step the new particles are sampled from M_t, so the performance of the algorithm strongly depends on the quality of these transition kernels. In the context of the Bootstrap formalism these kernels only describe the evolution of the observations themselves. In the Guided formalism we can directly influence the progression , e.g. by making the next particles also depended on the given observation and by doing so nudging them in a more likely scenario. Of course the potentional functions G_t have to be addressed too, to not unintentionally modify proportionality. Essentially, the part of taking a next guess is altered, so that they lie in a better proximity to the *real* value. If chosen well, the performance of the filtering increases immensely.

There is no *best* transition kernel M_t for every scenario, but it is possible to formulate an *optimal* transition kernel, at least in some sense. The next lemma shows, that the compromise between choosing a next X_t from X_{t-1} and accounting for the observation y_t minimizes certain variances.

Lemma 10. *Let $\mathbb{P}_0, P_t, f_t$ be given (the defining functions for a State Space model). For every $t \leq T$ the set $\Gamma_t := \{(M_t, G_t) \mid M_t,\, G_t$ fulfill the Guided formalism$\}$ contains an* optimal *kernel*

$$M_t^{opt}(x_{t-1}, dx_t) := \frac{f_t(y_t \mid x_t)}{\int_{\mathcal{X}} f_t(y_t \mid x')\, P_t(x_{t-1}, dx')} P_t(x_{t-1}, dx_t), \tag{2.43}$$

optimal in the sense, that it minimizes the variance of the weights w_t^n, the incremental weights $G_t(X_{t-1}^{A_t^n}, X_t^n)$ and the likelihood factors l_t.

Proof. Proof is found in [4], page 142. □

We can see, M_t^{opt} is exactly X_t given y_t **and** x_{t-1}. In a general application it may not be possible to directly determine this *optimal* kernel, or it could be quite hard to sample from. In any case it is recommended to find a transition kernel which is easy to use and close to the here formulated *optimal*.

2.4.2. *Smoothing Algorithms.* For the smoothing we will compare two different approaches, both mentioned in the section 2.2.2 in Feyman-Kac Models. We start by looking into the Forward-Filtering Backward-Smoothing algorithm.

Forward Filtering Backward Smoothing (FFBS). The idea of FFBS is an initial filtering step into a backwards transition kernel generating trajectories through the sampled particles. For the FFBS method we want to translate the results of the theoretical section 2.3.2 from the Feyman-Kac formalism back into the realm of State Spaces. First, from (2.33) we get

$$\begin{aligned}&\mathbb{P}_t(X_{0:t} \in dx_{0:t})\, \overleftarrow{P}_{t-1|t}(x_t, dx_{t-1}) = \\ &\frac{1}{p_t(y_t \mid y_{0:t-1})} f_t(y_t \mid x_t)\, P_t(x_{t-1}, dx_t)\, \mathbb{P}_{t-1}(X_{0:t-1} \in dx_{0:t-1} \mid Y_{0:t-1} \in dy_{0:t-1}),\end{aligned}$$

which gives with (2.34)

$$\overleftarrow{P}_{t-1|t}(x_t, dx_{t-1}) \propto P_t(x_{t-1}, dx_t)\, \mathbb{P}_{t-1}(X_{0:t-1} \in dx_{0:t-1} \mid Y_{0:t-1} \in dy_{0:t-1}) \tag{2.44}$$

where the normalizing constant is the integral over the density[5] of P_t with measure $\mathbb{P}_{t-1}$. In these two equations the index $t-1|t$ shows the dependence of this backwards kernel on the t first observations, whereas the *normal* kernel (e.g., with index $t-1$) does not depend on any observed data. If we want to approximate this backwards kernel it is necessary to run a filtering step first to obtain

$$\mathbb{P}_t(X_t \in dx_t \mid Y_{0:t} = y_{0:t}) \approx \sum_{n=1}^{N} W_t^n \delta_{X_t^n}(dx_t),$$

[5]We have to assume from this step on, that this density exists. In our application it does, luckily.

and then plugging it into (2.44) (along with the normalizing constant)

$$\overleftarrow{P}^N_{t|t}(x_{t+1}, dx_t) = \frac{\sum_{n=1}^N W_t^n\, p_{t+1}(x_{t+1}\,|\,X_t^n)\delta_{X_t^n}(dx_t)}{\sum_{m=1}^N W_t^m\, p_{t+1}(x_{t+1}\,|\,X_t^m)}. \tag{2.45}$$

Now, computing the *full* smoothing distribution is simply the task of substituting the approximations into

$$\mathbb{P}_T(dx_{0:T}\,|\,Y_{0:T} = y_{0:T}) = \mathbb{P}_T(dx_T\,|\,Y_{0:T} = y_{0:T}) \prod_{t=0}^{T-1} \overleftarrow{P}_{t|t}(x_{t+1}, dx_t), \tag{2.46}$$

$$\mathbb{P}_T^N(dx_{0:T}\,|\,Y_{0:T} = y_{0:T}) = \left[\sum_{n=1}^N W_T^n \delta_{X_T^n}(dx_T)\right] \prod_{t=0}^{T-1} \overleftarrow{P}^N_{t|t}(x_{t+1}, dx_t). \tag{2.47}$$

The first equation is computed through recursion, a proof is found in appendix C.7. This approach has different variations, i.e. it can also compute just marginal distributions. Its advantage is that it is quite easy to implement, but it is afflicted with two levels of approximations. Firstly, every distribution and kernel is approximated (this is the case for every algorithm here), but secondly and more severely, the different trajectories itself approximate $\mathbb{P}_T^N$. We now cover a more complicated, but in practice outperforming algorithm for marginal distributions.

Two-Filter Smoothing. The starting point is the derived smoothing distribution (2.35) using a change of measure from the filtering distribution.

$$\mathbb{P}_T(X_t \in dx_t\,|\,Y_{0:T} = y_{0:t}) \propto p_T(y_{t+1:T}\,|\,x_t)\, \mathbb{P}(X_t \in dx_t\,|\,Y_{0:t} = y_{0:t}). \tag{2.48}$$

To obtain this proportionality factor we take a look at some of the theoretical steps from that chapter, most importantly from the *cost-to-go* function. Inserting the Bootstrap or Guided formalism into (2.26) and the State Space counter part of $H_{t:T}$ described in (2.27) we obtain

$$p_T(y_{t+1:T}\,|\,x_t) = \int_{\mathcal{X}} f_{t+1}(y_{t+1}\,|\,x_{t+1})\, p_T(y_{t+2:T}\,|\,x_{t+1})\, P_{t+1}(x_t, dx_{t+1}). \tag{2.49}$$

In that section it was already mentioned that this quantity can be computed using backwards filtering, here it becomes clear why. Equation (2.49) is just like the forward recursion used for filtering, but with two differences. Firstly, it works backwards, hence the backward filtering. And secondly, the right hand side can integrate to infinity. To overcome this issue the Feyman-Kac model designated for the backwards filtering is chosen in a way, so that it approximates

$$\gamma_{t|T}(dx_t) \propto \gamma_t(dx_t)\, p_T(y_{t:T}\,|\,x_t) = \gamma_t(dx_t)\, f_t(y_t\,|\,x_t)\, p_T(y_{t+1:T}\,|\,x_t), \tag{2.50}$$

γ_t is a probability distribution and needs to be given. The following lemma gives the recipe for the Feyman-Kac model. To differ the forward and backward model we will use the arrow notation, already used for the backwards kernel notation.

Lemma 11. *When taking*

$$\overleftarrow{G}_T(x_T) = \frac{f_T(y_T\,|\,x_T)\, \gamma_T(dx_T)}{\overleftarrow{\mathbb{M}}_T(dx_T)}, \tag{2.51}$$
$$\overleftarrow{G}_t(x_{t+1}, dx_t) = \frac{f_t(y_t\,|\,x_t)\, \gamma_t(dx_t)\, P_{t+1}(x_t, dx_{t+1})}{\gamma_{t+1}(dx_{t+1})\, \overleftarrow{M}_t(x_{t+1}, dx_t)},$$

the defined Feyman-Kac model fits the required (2.50)*. It is assumed that the Radon-Nikodym derivative exists.*

Proof. C.6. □

The remaining part is to choose appropriate γ_t and $\overleftarrow{M}_t$. Since we have the same problem as in the filtering distribution just with a different formulation, we again have an *optimal* transition kernel:

$$\overleftarrow{M}_t^{\text{opt}}(x_{t+1}, dx_t) = \frac{f_t(y_t \mid x_t)}{\int_{\mathcal{X}} f_t(y_t \mid x') \overleftarrow{P}_{t+1}(x_{t+1}, dx')} \overleftarrow{P}_{t+1}(x_{t+1}, dx_t). \tag{2.52}$$

For γ_t it is recommended[6] to get as close to $\mathbb{M}_t$ as possible, if feasible $\gamma_t(dx_t) = \mathbb{M}_t(dx_t)$.

The actual two filter algorithm is the composition of the two filtering approximations

- **Forward Filtering** $\sum_{n=1}^{N} W_t^n \delta_{X_t^n}(dx_t) \approx \mathbb{P}(X_t \in dx_t \mid y_{0:t})$
- **Backward Filtering** $\sum_{m=1}^{M} \overleftarrow{W}_{t+1}^m \delta_{\overleftarrow{X}_{t+1}^m}(dx_{t+1}) \approx \gamma_{t+1|T}(dx_{t+1})$

plugged into (2.48) with the addition of the transition density $p_{t+1}(\overleftarrow{X}_{t+1} \mid X_t)$, this is required because the simulated particles do not overlap in the forward and backward filtering, hence can not be used for the same computation. Instead the probability for the transition is used to determine the likelihoods. Hereby our final smoothing distribution is

$$\mathbb{E}\left[X_t \mid Y_{0:T} = y_{0:T}\right] \approx \frac{\sum_{n,m=1}^{N} \omega^{n,m} \delta_{X_t^n}(dx_t)}{\sum_{n,m=1}^{N} \omega^{n,m}} \tag{2.53}$$

with

$$\omega^{n,m} := W_t^n \overleftarrow{W}_{t+1}^m \frac{p_{t+1}(\overleftarrow{X}_{t+1}^m \mid X_t^n)}{\gamma_{t+1}(\overleftarrow{X}_{t+1}^m)}. \tag{2.54}$$

2.4.3. *A Small Word on Convergence.* The convergence of particles filters follow the central limit argument, but the requirements and complexity changes depending on the sampling schemes used. We did not introduce these for obvious space and scope reasons, a reference is found at the end of chapter 2.3.1. This way, only the main results are presented from the main book chapter 11.

The first, important result is, that particle filter converge for $N \to \infty$ almost surely, meaning for fixed $t \leq T$

$$\sum_{n=1}^{N} W_t^n \delta_{X_t^n}(dx_t) \overset{\text{a.s.}}{\to} \mathbb{Q}_t(dx_t) \tag{2.55}$$

under the assumption of G_t being upper bounded. The error has a convergence rate of $N^{-\frac{1}{2}}$

$$\sqrt{N}\left(\sum_{n=1}^{N} W_t^n \delta_{X_t^n}(dx_t) - \mathbb{Q}_t(dx_t)\right) \Rightarrow \mathcal{N}(0, \nu_t). \tag{2.56}$$

ν_t is the cumulative error introduced in every time step. Under very strong assumptions it is possible to bound this error uniformly, the idea behind is to somehow *forget* errors of the past. The first assumption is, that G_t is uniformly bounded in time (strictly larger then 0) and the second one is on M_t admitting probability densities m_t with

$$\frac{m_t(x_t \mid x_{t-1})}{m_t(x_t \mid x'_{t-1})} \leq c_M, \tag{2.57}$$

[6]referring to the reference book [4], chapter 12, page 208f.

for some $c_M \geq 1$. These assumptions are more or less never met in practice.

Remember, all the convergences are towards $\mathbb{Q}_t$, meaning our result can not get better than the distribution we try to approximate. Sounds trivial, but this brings an arbitrary barrier into any particle filter, since in practice these distributions are not the given truth.

2.5. **Creating our Fast Ice Model.**

After introducing all the theory we need to compute a distribution depending on some kind of observation, we are now ready to create our fast ice model. Recall, the following distributions and densities must be given by our side:

- $\mathbb{P}_0(dx_0)$: the starting distribution to sample initial fast ice guesses,
- $P_t(x_{t-1}, dx_t)$: the transition kernel to evolve the fast ice through time,
- $f_t(y_t \,|\, x_t)$: the transition densities for a *likelihood* of sampled fast ice.

Also, the result from any filtering/smoothing has its values *continuously*[7] in $[0, 1]$, from which the fast ice (0,1) is still to be concluded. This step is covered at last.

2.5.1. *The Idea.*

We start by defining the decision space $\mathcal{X} := Mat_{m\times n}(\{0,1\})$, every element of the matrix is a small region of the image with the meaning

- 0: Fast ice or land,
- 1: Drift ice.

The double assignment of 0 is advantageous in a later stage and does not confuse anything as the land pixels are fixed and known. Assuming there is a given state $x_t \in \mathcal{X}$ the next state x_{t+1} should be an evolution of the previous one following a cleverly chosen distribution. The strategy is to look in a neighbouring region of any pixel to determine how likely it will become or stay fast ice. Essentially it is a bit like *Conway's Game of Life* [11], where a two dimensional field evolves over time according to set rules, but instead of these strict rules there are given probabilities for each pixel to ensure different outcomes. Also, the choice of likelihoods change over time to assure a fast growth at first and then a sharp decline of fast ice towards summer. From this point of view it is quite clear why a guided Feyman-Kac formalism is superior over a Bootstrap formalism, it is unclear in what direction fast ice should grow without any input of the observations.

For guiding and weighting the particles the transition density is defined through a pixel wise comparison of a scaled particle to the observation. The scaling ν_t is an interpolated and convoluted version of the corresponding observation to achieve a error free and more continuous factor. It is apparent that the scaled particle is simply

$$\underbrace{\nu_t\, x_t}_{\text{Pointwise}} = \begin{cases} 0, & \text{if } x_t = 0 \\ \nu_t, & \text{if } x_t = 1 \end{cases} \quad . \tag{2.58}$$

Lastly, the initial distribution can be based on a -1'st observation along with an evolution step using the transition kernel. The decision x_{-1} is described through a randomly (but small) chosen threshold on y_{-1}. This achieves a random but reasonable first guess. In practice one may need to take more than 1 observation due to a lack of data points, the idea remains the same.

[7]Continuously in the sense of *as continuous as the amount of particles N allow*, every value is a fraction with denominator N.

2.5.2. *Parameters and Functions.* We start by defining the transition kernels $P_t(x_{t-1}, dx_t)$, our goal is to have a pixel by pixel Bernoulli distribution. Having state x_t the first step is to run a convolution over the x_t to obtain neighbourhood information about each pixel. Here are two example operators, one rather simple and one a bit more sophisticated

$$c_1 = \frac{1}{8}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}, \tag{2.59}$$

$$c_2 = \frac{1}{\alpha}\begin{pmatrix} \frac{1}{4} & \frac{1}{\sqrt{5}} & \frac{1}{2} & \frac{1}{\sqrt{5}} & \frac{1}{4} \\ \frac{1}{\sqrt{5}} & \frac{1}{\sqrt{2}} & 1 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{5}} \\ \frac{1}{2} & 1 & 0 & 1 & \frac{1}{2} \\ \frac{1}{\sqrt{5}} & \frac{1}{\sqrt{2}} & 1 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{5}} \\ \frac{1}{4} & \frac{1}{\sqrt{5}} & \frac{1}{2} & \frac{1}{\sqrt{5}} & \frac{1}{4} \end{pmatrix}, \quad alpha = 4\left(\frac{7}{4} + \frac{1}{\sqrt{2}} + \frac{2}{\sqrt{5}}\right) \tag{2.60}$$

In practice there are many different to try out (none, euclidean, exponential decreases and different sizes), empirically it seems to be beneficial to use a bigger and decreasing matrix, so in the numerical part we use c_2.

This yields a new matrix ζ_t with the neighbourhood information

$$\zeta_t \coloneqq (c_2 * x_t), \quad \Rightarrow \zeta_t \in Mat_{m\times n}([0,1]) \tag{2.61}$$

Values closer to 0 imply land or fast ice in the surroundings whereas values closer to 1 show a lot of movement in the close environment. We define two (possibly non-surjective) functions

$$\begin{aligned} g_{0,1} : [0,1] \times 0, 1, \ldots, T &\to [0,1] \\ (\zeta_t^{(i,j)}, t) &\mapsto f_{0,1}(\zeta_t^{(i,j)}, t) \end{aligned} \tag{2.62}$$

which take in the values from ζ_t and the the time t to determine the probabilities $p^{(i,j)}$ of all the Bernoulli distributions ($i \le n$ and $j \le m$).

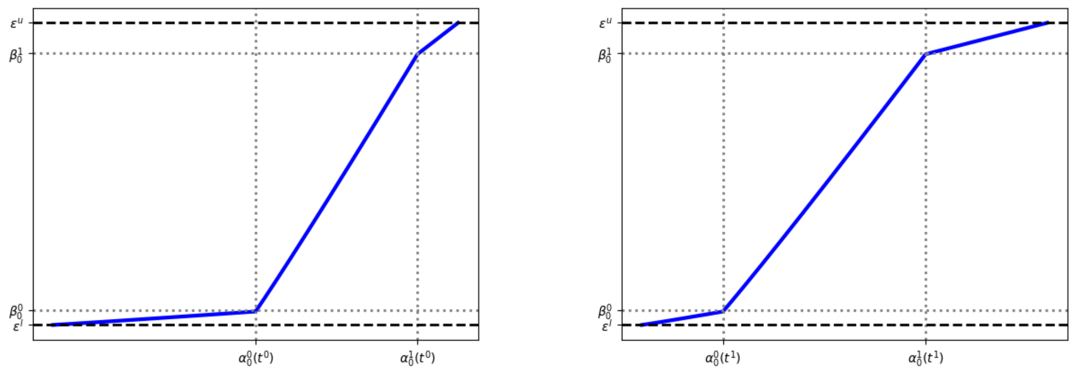
$$P_t(x_{t-1}, dx_t) \sim \left(\text{Bin}\left[1, \tilde{p}_t^{(i,j)}\left(x_{t-1}^{(i,j)}\right)\right]\right)_{i\le n, j\le m} \tag{2.63}$$

with $\tilde{p}_t^{(i,j)}\left(x_{t-1}^{(i,j)}\right) = g_{x_{t-1}^{(i,j)}}(\zeta_t^{(i,j)}, t)$. Essentially, f_0 computes the chance for fast ice to become drift ice and f_1 the chance for drift ice to stay drift ice. Remember, the higher the value $p^{(i,j)}$, the higher the chance for **Drift**-ice. The most important and critical is the definition of g_0 and g_1. Intuitively, it makes sense to split both into three parts depending on the time t and transition from one to another trough time. These three parts separated by $t^0, t^1 \in \{0, 1, \ldots, T\}$ are the same for both functions and are round about

- beginning of winter, when a lot of fast ice develops.
- winter, when the ice is fairly stable.
- spring, when fast ice melts.

See figure 19 from the simulation appendix A for reference.

Another crucial part is the evaluation of the neighbourhood values ζ_t, for what amount of relative *movement* is it reasonable to, in a sense, project a certain outcome by giving a high (drift ice) or low (fast ice) likelihood. For this effort, we introduce two functions depending on t (mostly defined through t^0, t^1) for each function $\alpha_k^0(t), \alpha_k^1(t) \in [0,1]$, $k = 0, 1$ which act as thresholds. To the left of α_k^0 the values stay quite low to increase the likelihood of fast ice and on the right of α_k^1 the values are close to 1 to achieve a good chance of drift ice. In between α_k^0 and α^1 the two values are interpolated.



(A) $t \leq t^0$ (B) $t \geq t^1$

FIGURE 7. Example g_0 function for values $t \leq t^0$ (left) and $t \geq t^1$ (right). The parameters in this case are:

- $\beta_0^{0:1} = 0.05, 0.90$
- $\rho_0 = 1.04$
- $\varepsilon^u = 0.0001$
- $\varepsilon^l = 0.0005$

The functions $\alpha_0^{0:1}(t)$ are later described in the parameter optimization section 2.5.4, equation (2.76). The there defined parameters lie between 0.2 and 0.9.

And finally, most significantly are the values of g_0, g_1 themselves. Here we use five parameters for each function, one $\beta_k^l \in [0,1]$ for each α_k^l, $l = 0, 1$ (the values of g_k at the threshold values), an exponent $\rho_k \in (0, \infty)$ which controls the general behavior of the function and $\varepsilon^u, \varepsilon^l \geq 0$ to have a minimal chance for every point to change to drift ice and fast ice respectively (l stands for *lower* and u for *upper* bound, also in practice $\varepsilon^l, \varepsilon^u$ value will always be > 0). These two are also the only parameters for both functions simultaneously. ρ_k is mostly used for the optimization of the functions, as it changes their whole appearance.

Definition 12. For a defined set of parameters $t^{0:1}, \beta_k^{0:1}, \rho_k, \varepsilon^u, \varepsilon^l$, a function $\alpha_k^{0:1}(t)$ and a convolution matrix c_s the transition kernels P_t are defined for $k = 0, 1$ through (2.63) with

$$g_k(\zeta_t^{(i,j)}, t) = \left(h_k(\zeta_t^{(i,j)}, t)\right)^{\rho_k} \left(1 - (\varepsilon^u + \varepsilon^l)\right) + \varepsilon^l, \quad \text{with} \tag{2.64}$$

$$h_k(\zeta_t^{(i,j)}, t) = \begin{cases} \frac{\zeta_t^{(i,j)}}{\alpha_k^0(t)} \beta_k^0 & , \quad \zeta_t^{(i,j)} \in [0, \alpha_k^0(t)) \\ \frac{(\alpha_k^1(t) - \zeta_t^{(i,j)})}{\alpha_k^1(t) - \alpha_k^0(t)} \beta_k^0 + \frac{\zeta_t^{(i,j)} - \alpha_k^0(t))}{\alpha_k^1(t) - \alpha_k^0(t)} \beta_k^1 & , \quad \zeta_t^{(i,j)} \in [\alpha_k^0(t), \alpha_k^1(t)) \\ \beta_k^1 + \frac{(\zeta_t^{(i,j)} - \alpha_k^1(t))}{1 - \alpha_k^1(t)} (1 - \beta_k^1) & , \quad \zeta_t^{(i,j)} \in [\alpha_k^1(t), 1] \end{cases}$$

Figure 7 shows an example function for a set of parameters.

The next step for the initial distribution $\mathbb{P}_0(dx_0)$ is rather straight forward after defining P_t.

Definition 13. We define the starting distribution with threshold range $(0, \epsilon]$ as

$$\mathbb{P}_0(dx_0) = P_0(x_{-1}, dx_0) \tag{2.65}$$

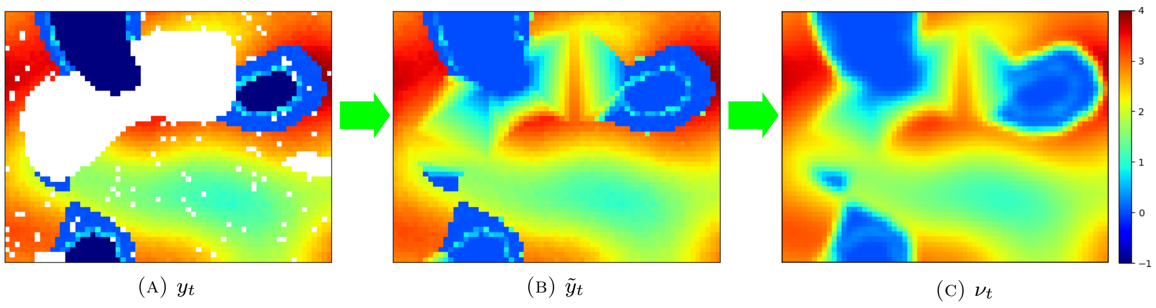


FIGURE 8. The process of transforming an observation into a usable matrix for the observation density function. First is a barycentric interpolation followed by a Gaussian filter.

with

$$x_{-1} := \left(\delta_{(0,\epsilon]}(y_{-1}^{(i,j)})\right)_{i,j}. \tag{2.66}$$

ϵ is chosen in a random environment close to 0 for every new particle as it is not clear which threshold is the best to start with. Also as mentioned before, in practice one might need to use multiple *initial* observations to achieve a satisfying cover of the area. In the case of our simulation we simply imitate this by choosing the first (and last for backwards filtering) observation to have only a little amount of lacking data points.

Lastly, for the observation density $f_t(y_t \,|\, x_t)$ it is necessary to manipulate the t's observation. First, a linear interpolation fills the *holes* in the data point. This linear interpolation is made through triangulation, so finding the three nearest points of a triangle enclosing the points of interest and then performing a linear barycentric interpolation as described in the scipy package [29] under the method *scipy.interpolate.LinearNDInterpolator*[8]. After this, a Gaussian filter reduces sharp edges and allows for a more smooth comparison, this is done by doing a convolution with a two dimensional Gaussian function G with variance σ_G.

$$\nu_t = (G \times \tilde{y}_t), \tag{2.67}$$

where $\tilde{y}_t$ is the interpolated observation y_t

$$\tilde{y}_t = \left(I(y_t^{(i,j)})\right)_{i\leq n, j\leq m}. \tag{2.68}$$

I is the barycentric interpolation of a single point mentioned above. Figure 8 shows a sample process.

At last, a normal distribution with expectation ν_t and variances $\sigma_t^{(i,j)}$ is used to *weight* the particles. The variances adapts depending on the quality of information in a point (e.g., lack of data point and how far is the nearest information). We define this variance as a quadratic approximation

$$\sigma_t^{(i,j)} := \hat{\sigma} + r_1 h(t)^{(i,j)} + r_2 \left(h(t)^{(i,j)}\right)^2 \tag{2.69}$$

where $h(t) \in Mat_{m\times n}\left([0,1]\right)$ reflects the quality of the neighbouring information and is derived from a convolution of $\frac{1}{9}\mathbb{1}_{3\times 3}$ with $\delta_{\text{NaN}}(y_t)$ (a matrix derived from y_t, where pixels are 1 if they are defective, 0 otherwise). $\hat{\sigma}$ is a constant.

[8]The scipy package is an extension for Python and is also used in the numerical part for efficient and exact numerical computations.

Definition 14. Given a Gaussian variance σ_G and local variances $\sigma_t^{(i,j)}$ we define the transition density point wise

$$f_t(y_t^{(i,j)} \,|\, x_t^{(i,j)}) \sim \mathcal{N}(\nu_t x_t,\, \sigma_t^{(i,j)}). \tag{2.70}$$

Another parameter is the chance for a pixel to be defective, a chance between 0 and 1. This however, has no theoretical advantages, as defective pixels are simply ignored. Still, it is found in the code to make sure it is expected to obtain such values [25]. As a last note, for both the Gaussian filter and the convolutions we need to make a choice on how to handle the edge case. We simply choose an extension of the border values. In a one dimensional case it would look like

$$\left(\underbrace{a}_{\text{extension}} \;\middle|\; a \quad b \quad c \quad d \;\middle|\; \underbrace{d}_{\text{extension}} \right),$$

and it introduces artificial errors along the edges making our model less reliable there.

2.5.3. *Fitting it into the Feyman-Kac Model.* For the filtering and smoothing we first need to translate the described functions into the Feyman formalism. The Bootstrap model (def 6) is straight forward, for the Guided model (def 7) we implement the recommended optimal kernel M_t^{opt} from Lemma 10. It is sufficient to compute the probabilities for $x_t = 1$ given x_{t-1}, since M_t is a Bernoulli distribution by construction. We get

$$\begin{aligned} M_t^{\text{opt}}(x_{t-1}, 1) &= \\ &= \frac{f_t(y_t \,|\, 1)}{\int_{\mathcal{X}} f_t(y_t \,|\, x')\, P_t(x_{t-1}, dx')} P_t(x_{t-1}, 1) \\ &= \frac{f_t(y_t \,|\, 1)}{\sum_{x \in \{0,1\}} f_t(y_t \,|\, x)\, P_t(x_{t-1}, x)} P_t(x_{t-1}, 1) \\ &= \frac{\exp\left\{ -\frac{1}{2} \left(\frac{y_t - \nu_t}{\sigma_t} \right)^2 \right\}}{\exp\left\{ -\frac{1}{2} \left(\frac{y_t - \nu_t}{\sigma_t} \right)^2 \right\} \tilde{p}_t(x_{t-1}) + \exp\left\{ -\frac{1}{2} \left(\frac{y_t}{\sigma_t} \right)^2 \right\} (1 - \tilde{p}_t(x_{t-1}))} \tilde{p}_t(x_{t-1}) \end{aligned} \tag{2.71}$$

and

$$G_t^{\text{opt}}(x_{t-1}, x_t) = \frac{P_t(x_{t-1}, dx_t) f_t(y_t \,|\, x_t)}{M_t^{\text{opt}}(x_{t-1}, dx_t)} \tag{2.72}$$

which is well defined, again, by construction. From here it is possible to Filter and use the FFBS algorithm. At last we also want to implement the Two-Filter smoothing, thus requiring a bit more work.

At first it is necessary to create a backwards filtering model. The theoretical optimal backwards kernel stated in equation (2.52) is not feasible since the backwards transition kernel $\overleftarrow{P}_{t+1}(x_{t+1}, dx_t)$ depend on the marginal distributions of $\mathbb{P}_t(dx_t)$ and $\mathbb{P}_{t+1}(dx_{t+1})$ which are approximated with the forward filtering, but since they are discrete distributions it does not work for newly sampled particles from backwards filtering. Option two is to approximate this optimal kernel, we do so by using the developed theory for forward filtering with adapted parameters. Finally, to obtain $\gamma_t(dx_t) \approx \mathbb{M}_t(dx_t)$ we sample M particles from x_t using the backwards kernel and use the average weighting to decision y_{t-1}

$$\gamma_t(x_t) = \frac{1}{M} \sum_{m=1}^{M} f_{t-1}(y_{t-1} \,|\, X_{t-1}^{(m)}), \quad X_{t-1}^{(m)} \sim \overleftarrow{P}_t(x_t, dx_{t-1}). \tag{2.73}$$

Name	Space	Description
$\alpha_k^{0:1}$	$[0,T] \to [0,1]$	splits g_k into 3 parts: low, transition, high likelihood
$\beta_k^{0:1}$	$[0,1]$	determines the point wise evaluation of g_k at $\alpha_k^{0:1}$
ρ_k	$(0,\infty)$	parameter to change appearance of g_k (close to 1)
$\varepsilon^u, \varepsilon^l$	[0,1]	minimal chance for fast/drift ice
σ_G	$(0,\infty)$	variance of the Gaussian filter
$\hat{\sigma}$	$(0,\infty)$	constant variance for the transition density f_t
r_1, r_2	$\mathbb{R}$	coefficients of the quadratic variance function

Table 1. System Parameters (for $k = 0, 1$), the parameters which emerge from the model choice itself and need to be optimized using a kind of performance test.

Name	Space	Description
$t^{0:1}$	$\{0, 1, \ldots, T\}$	time stamps with important changes: development of stable and melting of fast ice

Table 2. World Parameters, the parameters which have a resemblance in the real world and can be optimized (or even computed) referencing the data.

This of course does not explicitly approximate $\mathbb{M}_t(dx_t)$, but is indirectly assessing the different x_t, and after normalizing the weights over all t it is a significant quantity. A check of the pre-requirement for the potential functions $\overleftarrow{G}_t$ for Lemma 11 is found in appendix C.8, we now go over into the numerical part.

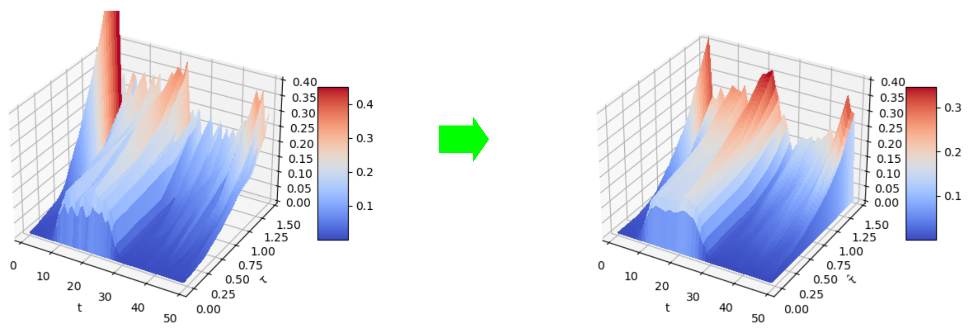
2.5.4. *Optimizing the Parameters.* The first numerical part is to optimize the parameters introduced in the last sections to strive for a well working model. For this, we first sort the parameters into *system parameters* and *world parameters*, tables 1 and 2 explain the difference.

World Parameters. The world parameters t^0 and t^1 can be chosen after an approximated total fast ice on the observations, and can change tremendously from one to the next case, in contrast to the system parameters. The idea is to find a resemblance of the function (A.1) used in the simulation of the fast ice (appendix A), basically we want to identify the two slopes in the graph 19. To this end, a threshold τ is used to determine the relative fast ice in every observation, relative to the non-error pixels. We call this ratio

$$s_t = \frac{\tau(y_t)}{T(y_t)} \tag{2.74}$$

with $\tau(y_t)$ being the amount of pixels $< \tau$, and $T(y_t)$ the total amount of error-free pixels. Figure 9 shows such ratios for an example. It is clearly quite noisy, so a moving average (step size 3) is used to reduce the volatility. Notice the similarity to graph 19.

This exemplary image also shows the empirically chosen value for τ to lie in $[0.1, 0.3]$, if not said otherwise this value is 0.2. To obtain the two slopes we use the *fundamental theorem of calculus* and assume the plotted function in figure 9 to be a rectangle (along the time axis). For that matter we choose a threshold $\hat{\varepsilon} = 0.025$ and get two starting values for the bump, one from the left a and one



(A) True values (B) Running mean

FIGURE 9. This figure shows the ratio values s_t for different thresholds τ (y-axis) though time t (x-axis). Left is the true value, whereas right is a running mean of the values (step size 3).

from the right b. After numerically integrating the area under the graph for a chosen τ we get

$$A = \int_a^b c\, dx = c(b-a) \quad \Rightarrow \quad c = \frac{A}{b-a}, \tag{2.75}$$

and use the computed c to obtain two new values $\hat{a}, \hat{b}$ which define our two time steps $t^0 = \hat{a}$, $t^1 = \hat{b}$. This principle is visualized in figure 10.

System Parameters. The idea for the system parameters is to find a general region for every parameter and only adapt any parameters within that region in different practical scenarios. Also, to optimize these parameters mostly benefits the convergence of any of the filtering/smoothing algorithm and is not necessary for it to work. Nevertheless, any kind of optimization can result in a better performance over all (qualitatively and time effectively).

Any of the real valued system parameters (so all but the functions $\alpha_k^{0:1}(t)$) can be set as a distribution instead of a fixed value. The idea is to then sample such parameters and compute the likelihood values through a filtering step (introduced in equation 2.15), this is repeated for M times, where after every repetition the parameters get changed slightly. In the end a weighted mean consisting of the likelihood values and the sampling density determines the average value for one observation cycle. This algorithm is called *Particle Marginal Metropolis-Hastings* (PMMH) and can be implemented easily after assigning starting distributions. A more in depth description (both methodically and mathematically), as well as other procedures for parameter estimation can be found in source [4] in chapter 16.To obtain a not

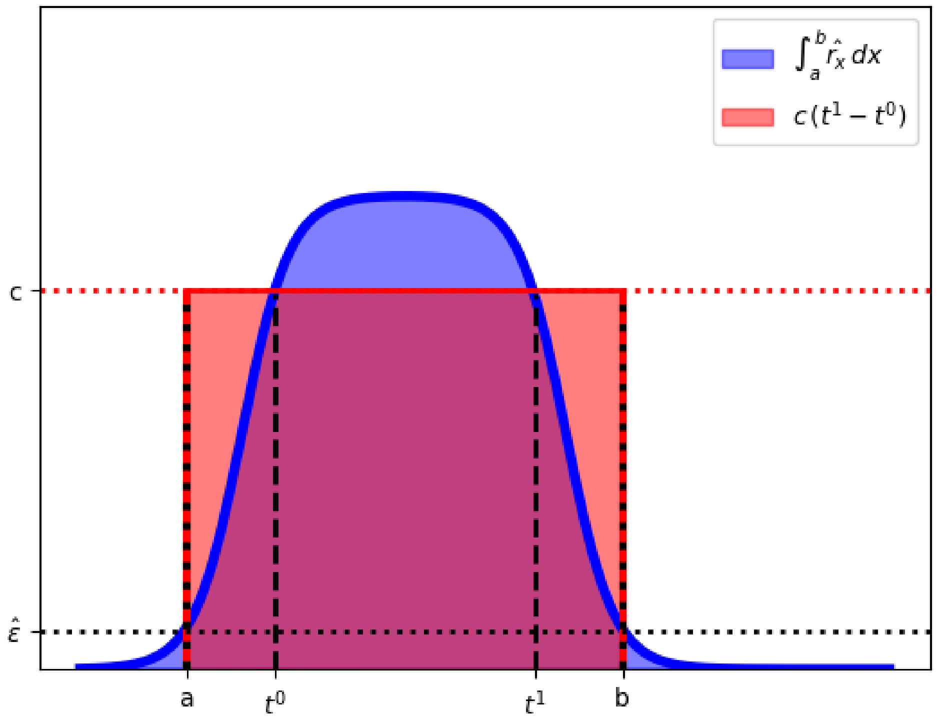


Figure 10. Conceptual visualization of the computation for the parameters t^0, t^1. $\hat{r}$ is the value obtained through the running mean.

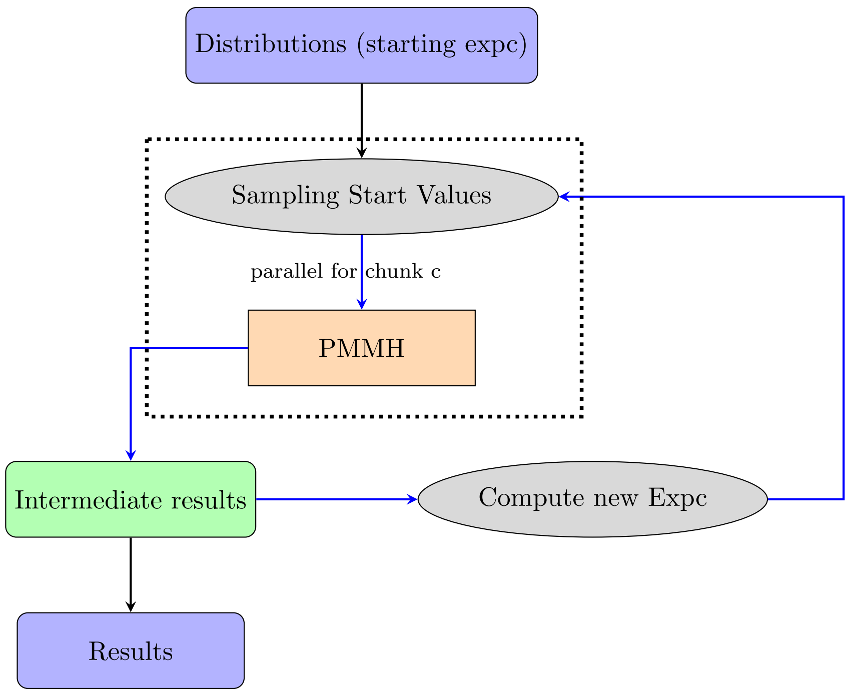


FIGURE 11. This flowchart shows the optimization process using the PMMH algorithm. The blue arrows symbolize the repeated executed action. The black, dotted rectangle signifies the execution of multiple (chunks of size c) optimizations at once.

to complex expression for $\alpha_k^{0:1}(t)$ we introduce (even more) parameters, 2 for each function, denoted as $\hat{\alpha}_{k,0:1}^{0:1}$ [9], with $0 \leq \hat{\alpha}_{k,1}^{0:1} \leq \hat{\alpha}_{k,0}^{0:1} \leq 1$. The order is inverted since for lower t these threshold are higher (towards winter we want more fast ice and the further right the thresholds are the likelier it gets). We then define the functions as a linear interpolation as follows

$$\alpha_k^{0:1}(t) = \begin{cases} \hat{\alpha}_{k,0}^{0:1} & , \quad t \leq t^0 \\ \frac{(t-t^0)\hat{\alpha}_{k,1}^{0:1} + (t^1-t)\hat{\alpha}_{k,0}^{0:1}}{t^1-t^0} & , \quad t \in (t^0, t^1) \\ \hat{\alpha}_{k,1}^{0:1} & , \quad t \geq t^1 \end{cases} \tag{2.76}$$

Now, we want to optimize the 18 parameters (see table 1, in total 8 for the α functions and 4 for the β values), excluding the σ_G for the Gaussian variance which we set to 1. The optimization works as follows: first an initial expectation for every parameter is chosen as well as a fixed variance for a normal distribution (truncated normal to $[0, 1]$ if necessary). Then, a set chunk of parameters get optimized by sampling the starting value from the distributions and running the PMMH algorithm on a sequence of observations. Hereon, the expectation in the distributions get adapted according to the already existing results and the next chunk is optimized. This is repeated. A visualization of this process is shown in figure 11. In our case the parameters for the PMMH algorithm are chosen as follows

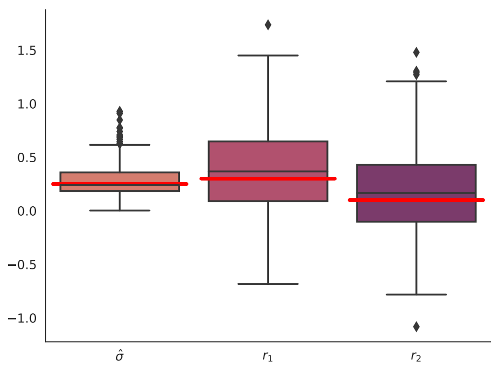
[9] I apologize for the confusing notation! As a reminder: the k stands for the original g function, so k=0,1. It clarifies if the predecessor of a pixel was fast ice (0) or drift ice (1). The index next to k shows that there are 2 different parameters (the two we just introduced). The last $0:1$ expresses the two functions for each of the original g function. So in total, there are 4 functions $\alpha_k^{0:1}(t)$ with each 2 parameters, called $\hat{\alpha}_{k,0:1}^{0:1}$

- $N = 512$, total amount of scenarios,
- $c = 64$, chunk size for parallel computing,
- iter $= 45$, number of iterations in any PMMH cycle,
- part $= 100$, amount of particles per filtering step (per iteration),
- $\text{seed}_0 = 1000$, starting seed for the fast ice generation,
- $T = 50$, the cycle length for the generated fast ice,
- grid $= (65, 50)$, the image size for any observation in pixels.

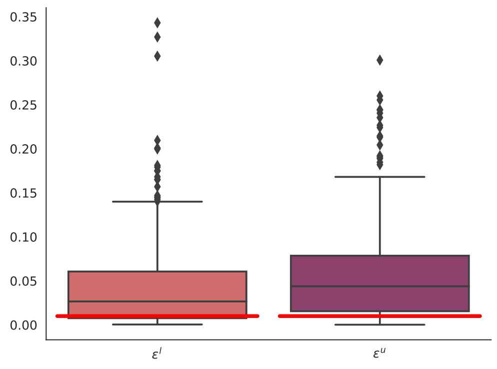
The paramters T being only 50 is of course not the best choice if the original data has over 100 observations (so way more data points), but they suffice regardless. The main reason for reducing the amount of observations is the cap on computational power.[10].The final results are depicted in the figure 12 for an estimated defective percentage of 0.3 and a medium to low noise level on the observations. The red line in the graphs depicts the initial starting guess. In the appendix B.1 two further results are shown for different observation scenarios. In the beginning the exponential parameters p_0 and p_1 decreased rapidly, spoiling all the other parameters; In a second round of optimization they were limited close to 1 and the results look more promising. Interestingly all the optimizations yield similar values for the different parameters independent of the error and noise level. Most of the parameters did not move too far away from the initial guess, probably because of the method itself when using normal distributions. It seems though, that the function thresholds $\hat{\alpha}$ (on the x-axis) moved close to a point on the lower side (controlling the fast ice) and further apart on the higher side (controlling the drift ice). This effect I can not explain. Also, in small numerical tests the differences in performance are vague and not really noticeable compared to the starting parameters.

2.5.5. *Transformation of the Results.* At last, we need to transform the results into a comparable state, values of either 0 or 1. A classification with a hard break is made, all values over 0.8 are drift ice, all others fast ice (except for Bootstrap). This value is chosen using the ROC-curve depicted in the next section Numerical Experiments (figure 14).

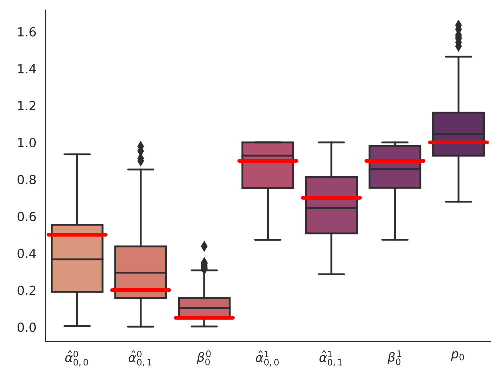
[10]This optimization took over 8 hours on 64 cores on a HCP server. And the observation amount does not factor linearly.



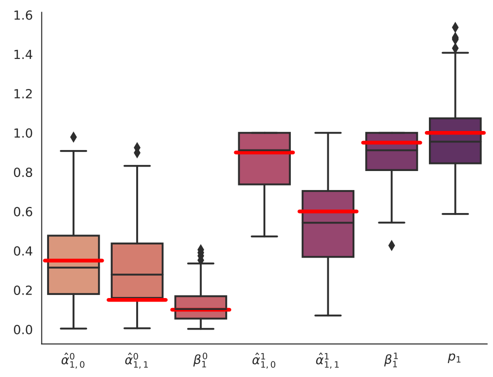
(A) σ_t



(B) g_0 and g_1



(C) g_0



(D) g_1

FIGURE 12. Results from the parameter optimization for 30% errors and low noise level. (A) are the parameters for the variance computation in the transition density. (B) shows the two global function parameters for g_0 and g_1. (C) and (D) are the unique parameters for each of the fast ice likelihood functions. The red line is the initial starting guess.

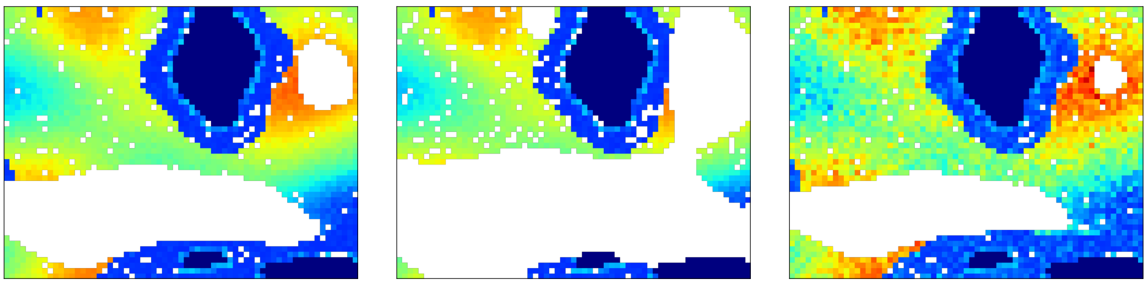


FIGURE 13. The three different scenarious described in the list above. A *normal* (A), a strongly defective (B) and a very noisy (C) setting.

3. Numerical Experiments

In this chapter, we will numerically test the two distinct smoothing algorithms (FFBS and Two-Filter) for the Guided model against a *trivial* weighted mean function[11] as well as a Bootstrap approach for comparison. In total we will look at three different scenarios showcased in the parameter optimization section 2.5.4 and appendix B.1

(A) normal error rate (30%) with little noise,
(B) strong error rate (60%) with little noise,
(C) low error rate (20%) with strong noise.

Figure 13 shows a sample image for an observation on the same seed.

These tests will run on different seeds in relation to the parameter optimization. The parameters are chosen as the mean from the optimization.

A big thanks to Prof. Dr. N. Chopin for his python implementation of most of the different filtering / smoothing algorithms, GitHub: [3].

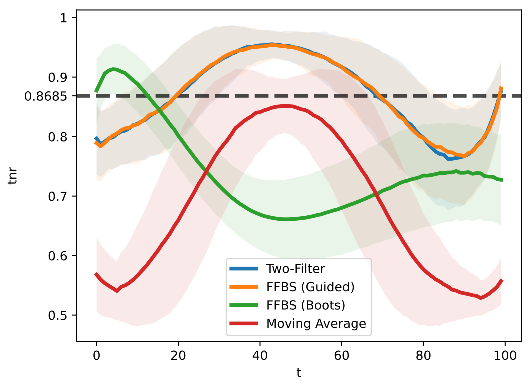
3.1. A *Normal* Scenario (A).

Figure 14 shows the different methods on the standard scenario (A) in its performance depicting their *balanced accuracy* as well as the *ROC-curve*. The *balanced accuracy* is

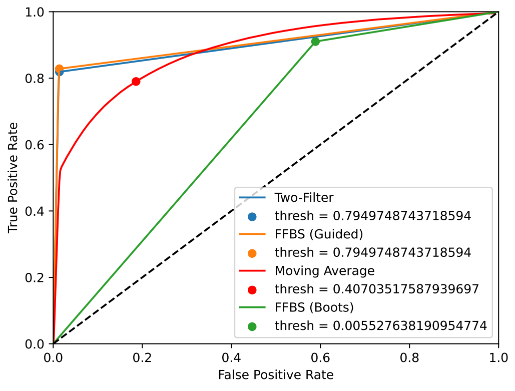
$$\mathrm{BA} = \frac{TPR + TNR}{2}, \tag{3.1}$$

where TPR is the true-positive-rate and TNR the true-negative-rate. This metric shows the proportion of correctly identified pixels and gives a good overview about the altogether performance through time. The *ROC-curve* illustrates the relationship of true positives to false positives, the time dependence is lost in this representation. All methods had a sample size of 512 with a particle amount of $N = 1000$ (the moving average had length of 11 time steps). One can clearly see the superior performance of the Monte Carlo approach in comparison to the moving average. Particularly in the beginning and end they deviate a lot. Interestingly the Bootstrap does achieve very good results in the beginning, even outperforming the Guided approaches, but then dramatically decreases with time. This behaviour was predicted in the mathematical part.

[11]This *trivial* weighted mean can be seen as the result of the linear Stochastical Programming.

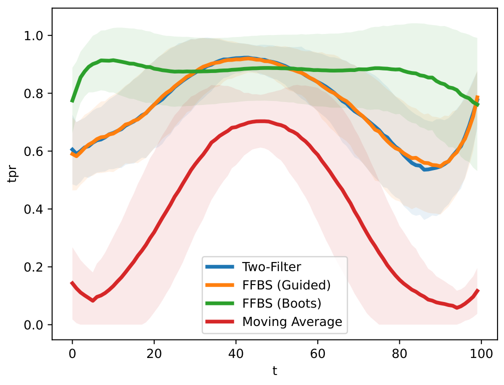


(A) Balanced Accuracy

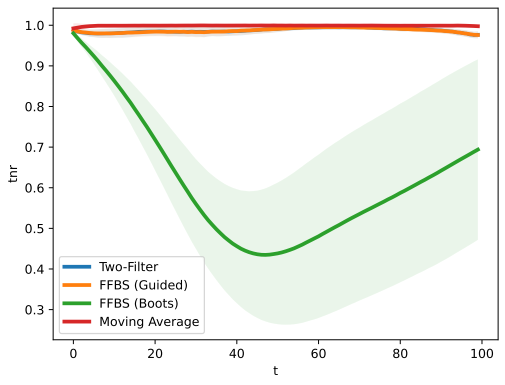


(B) ROC-Curve

FIGURE 14. Shows the Balanced Accuracy (left) and ROC-curve (right) of the different methods used on the *normal* scenario. The black line in the balanced accuracy is the average over time of the Two-Filter / FFBS (Guided). The dots in the ROC-curve shows the *optimal* threshold value (closest to $(0, 1)$).



(A) True-Pos-Rate

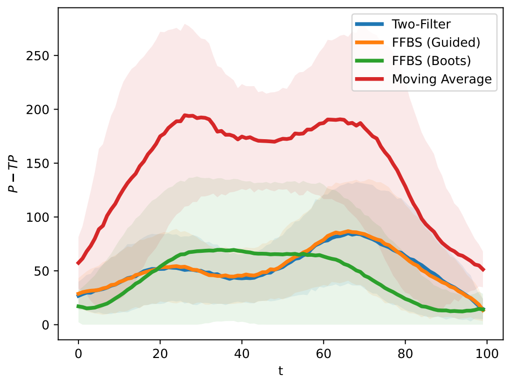


(B) True-Neg-Rate

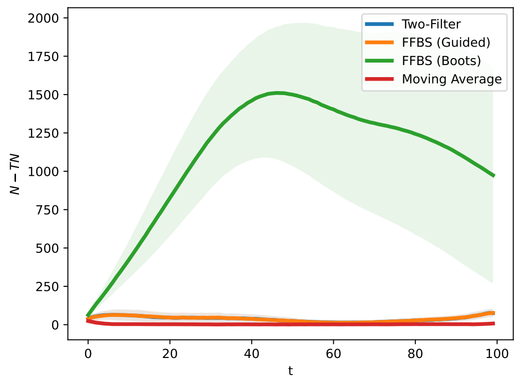
FIGURE 15. This figure shows the two single components of the BA. The True-Positive-Rate left and True-Negative-Rate right, expected value with one standard deviation.

The mediocre outcomes of the Two-Filter and FFBS algorithms at about $t \approx 85$ are a bit unexpected. Also, the relative rate is a bit misleading in this context. Other than that they seem to be performing more or less identically. A bit more insight is given by the single parts (true-positive and true-negative) of the BA, 15.

Indeed, FFBS and Two-Filter smoothing do perform identically. The Bootstrap *overshoots*, catching most of the true fast ice but also classifies way too much of the rest as fast ice. Let us look at the absolute values for comparison, figure 16.



(A) P − TP



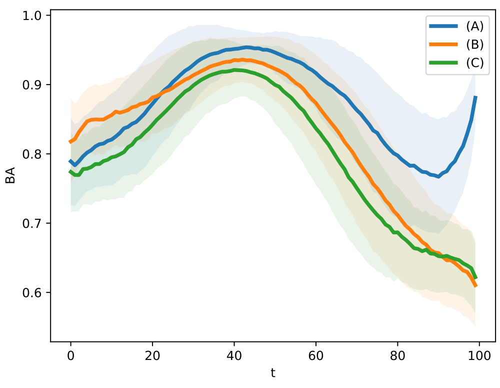
(B) N − TN

FIGURE 16. Absolute error of the True-Positive (left) and True-Negativ (right).

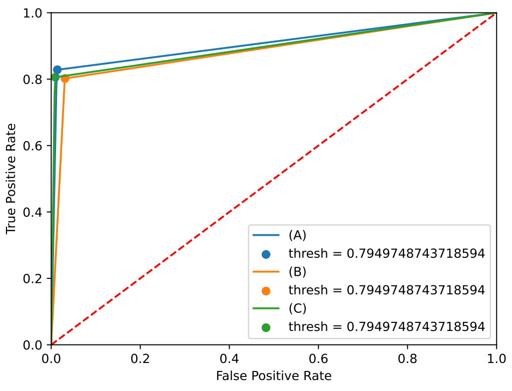
As a first observation, the absolute errors go down on each side, which makes sense since there is just a bit of fast ice in the beginning and end of each cycle; most of it being at the coast and afflicted by a biased error. The little bump in left graph at $t \approx 70$ is due to the melting of fast ice There, the smoothing algorithms show some troubles, it is first gone in the model and then in the actual data. This makes it a bit hard to determine a correct timing of the breaking of fast ice. In comparison to the Bootstrap the other methods *undershoot* a bit, but achieve an averaged accuracy of 87%.

3.2. **Defective and Noisy Scenario (B) (C).** The two other scenarios (B) and (C) are compared using the two-filter method with the same specifications to the first scenario (A) and are shown in figure 17.

The noisy and defective observations perform quite similar, with the defective scenario (B) being a bit more accurate. Both loose their accuracy towards $t \to T$, probably because they can not find the remaining fast ice pixels spread along the coastlines. Interestingly, the defective scenario yields a bit better results at the start, this is a bit mysterious. The ROC-curve also hints to a more common false positive classification in (B). This we can see in figure 18.



(A) Balanced Accuracy



(B) ROC-Curve

Figure 17. Shows the Balanced Accuracy (left) and ROC-curve (right) of the Two-Filter algorithm in the different scenarios. For the BA the thick line is the mean with the shaded area being one standard deviation.

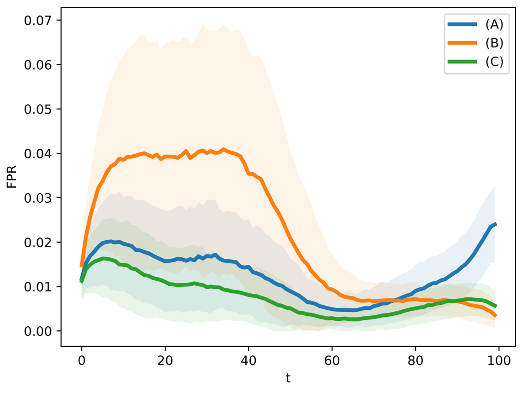


Figure 18. Shows the False-Positive-Rate of the different scenarios.

Building upon the Monte Carlo approach, there are plenty more ideas and adaptations to be tried, some of those bringing a definite improvement. One of those would be the use of Markov Chain Monte Carlo, to speed up the smoothing using Metropolis-kernels [5]. This would allow for more particles[12] or simply more numerical tests. There is also more on parameter optimization than I stated here. Also, there are probably other ways of describing the fast ice evolution than to use pixel by pixel Bernoulli distributions, e.g., using matrix transformations.

Of course the next step would be to use this approach on real data points, all the groundwork is done. The biggest task is the preparation of the data itself, as described in 1.3. Looking into that the following things could be added to the model (and maybe to the simulated training data):

(i) add water depth information,
(ii) spontaneous ice breaks,
(iii) error bias along the coast lines,
(iv) current and weather data,
(v) an extension from a single fast ice region towards multiple with local changes.

Most of these informations can probably be integrated into the transition kernels through g_k.

A completely different approach would to frame the fast ice using a curve and finding the best fit to do so in every step. This idea comes from weather forecasting and utilizes variational data assimilation to track curves through (sometimes missing) noisy data points. Since this approach relies on visual boundaries (e.g., of clouds) the idea of using the velocity of ice suggests itself for the same reason stated in this work. I pondered to go for this idea but ultimately decided together with my supervisors on Monte Carlo Simulations. A good reference for this alternative procedure can be found here [19] or here [2].

Last but not least, one could definitely try to utilize convolutional neural networks, training it on the same simulated data I optimized my method on.

[12] This time effect is **huge**, in some test up to 50 times faster (half an hour instead of 24), but describing everything mathematical goes beyond the scope of this thesis. So it is implemented in the code, but not used or described.

5. Conclusions

The proposed ideas operate in a promising way on detecting fast ice under uncertainty while benefiting from the time dependency. An average balanced accuracy of about 87% is not yet sufficient to work without human verification, but shows the potential of the presented approach. The two smoothing algorithms presented in the Monte Carlo simulation perform identically well and outperform the linear Stochastical Programming (weighted mean) by a lot. The variance of any iteration stays stable over time and only affects outcomes by up to $3-5\%$ (absolute). Especially through the main phase of fast ice the results convince, having an (balanced) accuracy of over 93%. Problematic are the two (time) edges (starting and end-time), notably the dent on the right. The relative findings drop down to a True-Positive-Rate of about 60%. This is caused by fast ice only remaining at the coast, if at all. There, a biased error makes it hard for the algorithms to find the fast ice. In the end, all results except the Bootstrap *undershoot* the actual fast ice, this means that if a pixel is flagged for fast ice it most definitely is fast ice. This outcome can be adapted in next steps, e.g., by decreasing likelihoods of g_t and increasing thresholds for the classification.

Changing the incoming data points in terms of defectiveness does not influence the results too much, surprisingly even after doubling the error rate to 60%. A noticeable effect is only felt after $t > \frac{3}{5}T$, when the noisy and defective scenario diverge with a linear decrease in accuracy. However, only a small decrease in performance is seen in the ROC-curve.

In summary: even though the here particular implementation has room for slight improvement (ideas and examples given in Further Ideas and Follow Ups), the framework of Monte Carlo Simulations has potential in the application of fast ice detection and shows promising results for a new stochastic approach.

In this segment you will find everything about the simulation of the data.

A.1. **Perlin Noise.** Perlin Noise is playing a huge role in the fast ice simulation, which is why we will do a rough coverage. It is named after its creator Ken Perlin and is the first ever implemented gradient noise published in a paper by 1985 [20]. The original idea has since been adapted and optimized to the so-called Simplex Noise [21], this we will not cover here as the standard Perlin Noise is sufficient.

For (2D) gradient noise one fixes a grid size and sets each integer knot to 0. To determine the values in between the grid points, each integer grid point is assigned a (pseudo-)random gradient. These gradients are then dot multiplied with the vectors going from the grid point to the point in question. At last, an interpolation between each corner point yields the final value [6]. In the case of Perlin noise the random gradients are pseudo-random to allow the same computation by repeated assignments. A standard choice are eight unit vectors evenly distributed around the points, e.g.,: $g_i = [\mathrm{Re}\left(e^{i\phi}\right), \mathrm{Im}\left(e^{i\phi}\right)]$ with $\phi = k\pi/4\,,\ k = 0, \ldots, 7$. For interpolation the polynomial function[13] $f(t) = 6t^5 - 15t^4 + 10t^3$ is used as it has zero first and second derivative at both ends $t = 0, t = 1$, being the integer lattice points[21]. Here a scheme for computing Perlin noise in a single point $P = [x, y]$, algorithm 3.

Algorithm 3 Perlin for a single point

```
1: $[x_i, y_i] \ = floor([x, y])$   ▷ Using top left corner as reference for vector
2: $[x_f, y_f] = [x, y] - [x_i, y_i]$   ▷ Vector to point P
3: $g_0, \ldots, g_3 = randomVec(ix, iy)$   ▷ Generating the Pseudo-Random gradients at each vertex around P
4: $d_i = g_i \cdot [x_f, y_f]$   ▷ Computing the dot product with the vector
5: **Return** inter$(d_0, \ldots, d_3)$   ▷ Interpolating the values $d_0, \ldots, d_3$ according to $f$ and distances $x_f, y_f$
```

In practice this computation is vectorized over all points in the grid [25], the python implementation is found in the GitHub.

A.2. **Description of the Velocity Simulation.** The simulated data should be constructed forward in time, so in every new step the velocity field is derived from the last few steps according to the rules and ideas stated below. In general these points should be acquired:

(i) Changing fast ice cover, extending and shrinking with time.
(ii) Changing velocities around the fast ice, simulating the drift ice.
(iii) Noise and missing data with local coherence (e.g., replicate effects of rain).
(iv) Land-mass with strong noise / few information along the coastlines.

Starting with point (i) and (iv) we generate an initial Perlin noise grid. Interpreting the values as a height map we set the sea level at the α-quantile of all pixels, giving a land mass of intended size. For the fast ice we use a threshold function depending on the time step, allowing for a controlled but random extension of the fast ice cover and later a reduction in similar matter, the values of this function lie in between a lower bound and the α-quantile of the land. Here, the lower bound fixes the maximal amount of fast ice reachable as it is *filled in* in between the α-quantile[14] and the yielded value of the function.

[13]Here the original posted interpolation function $f(t) = 3t^2 - 2t^3$ would get by as well, the second order continuities are not needed.

[14]In practice the upper bound is below the land threshold to assure at least a bit of fast ice all year long.

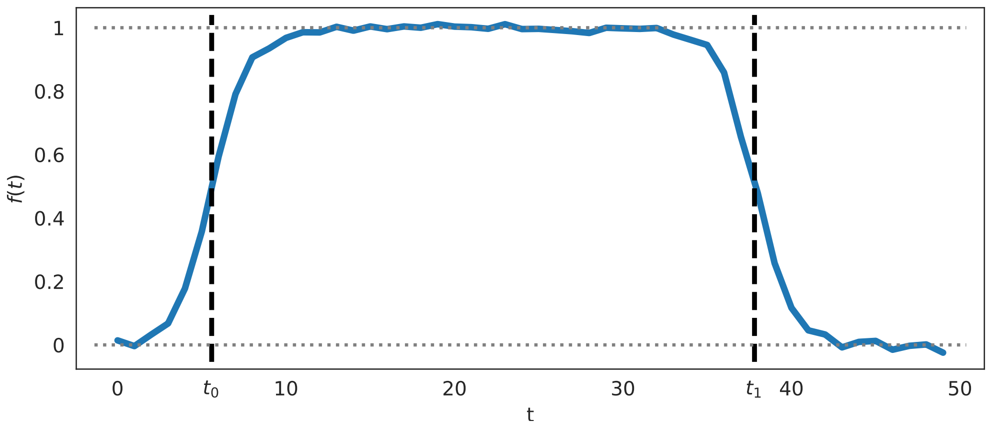


FIGURE 19. Example of the threshold-function f with parameters $b_l = 0$, $b_u = 1$, $\alpha = 23$, $T = 50$ and a variance of $\sigma = 0.05$.

Consequently most of the fast ice will be generated along the coastlines 'reaching' into the sea, the way it is aimed to be. The choice on the threshold function is not that relevant as long as it grows and shrinks in a rapid way towards winter and summer respectively, this is because most of the characteristic of fast ice is already obtained from the original Perlin noise. In our case two hyperbolic tangents are utilized:

$$f(t) = b_l + \frac{\tanh\left(\alpha(\frac{t}{T} - t_0)\right) - \tanh\left(\alpha(\frac{t}{T} - t_1)\right)}{2}(b_l - b_u) + \Phi_t \tag{A.1}$$

with t being the time step, T the cycle length determining how long a winter period is (in observation steps), t_0 and t_1 the beginning and ending of the fast ice (relative to T), b_u and b_l the upper and lower bound, α a factor to control the steepness of the slopes and Φ_t a normal distribution with scaled down variance. All pixels falling into these bounds get a speed of zero, all the land pixels an identifiable constant (in practice -1) and along the coastline of the land the pixels are sampled from a scaled uniform random distribution to obtain a biased error (e.g.,. $\sim U\left([\epsilon_l, \epsilon_u]\right), 0 < \epsilon_l < \epsilon_u < 1$).

Looking at point (ii), simulating the velocities around the fast ice and land pixels, it is fairly easy to see the usage of (again) a Perlin noise, the grouped but continuously (in space) changing speeds are natural features, both in drift ice and Perlin noise. So in every step a new Perlin noise is generated representing the velocities of each pixel, for that matter it is affinely transformed by simply offsetting all values until they are non-negative and then scaling it by the *maximal speed* $\hat{v}$ drift ice should attain. Let $v_{min} < 0$ be the minimum of the values generated [15], then the point wise transformation looks like

$$h(x) = (x - v_{min})\,\hat{v} \tag{A.2}$$

In order to imitate some correlation in the drift speeds (e.g., the movement of ice in a direction) the newly generated Perlin noise is (randomly) linearly interpolated with the last one, this creates mild effects of moving ice as different spots stay loosely consistent in time and also seem to *drift* in some direction, see figure 20.

[15]it is always < 0 in Perlin noise which has more than one grid-square, this is a direct result of the computation of the values inside them, see A.1.

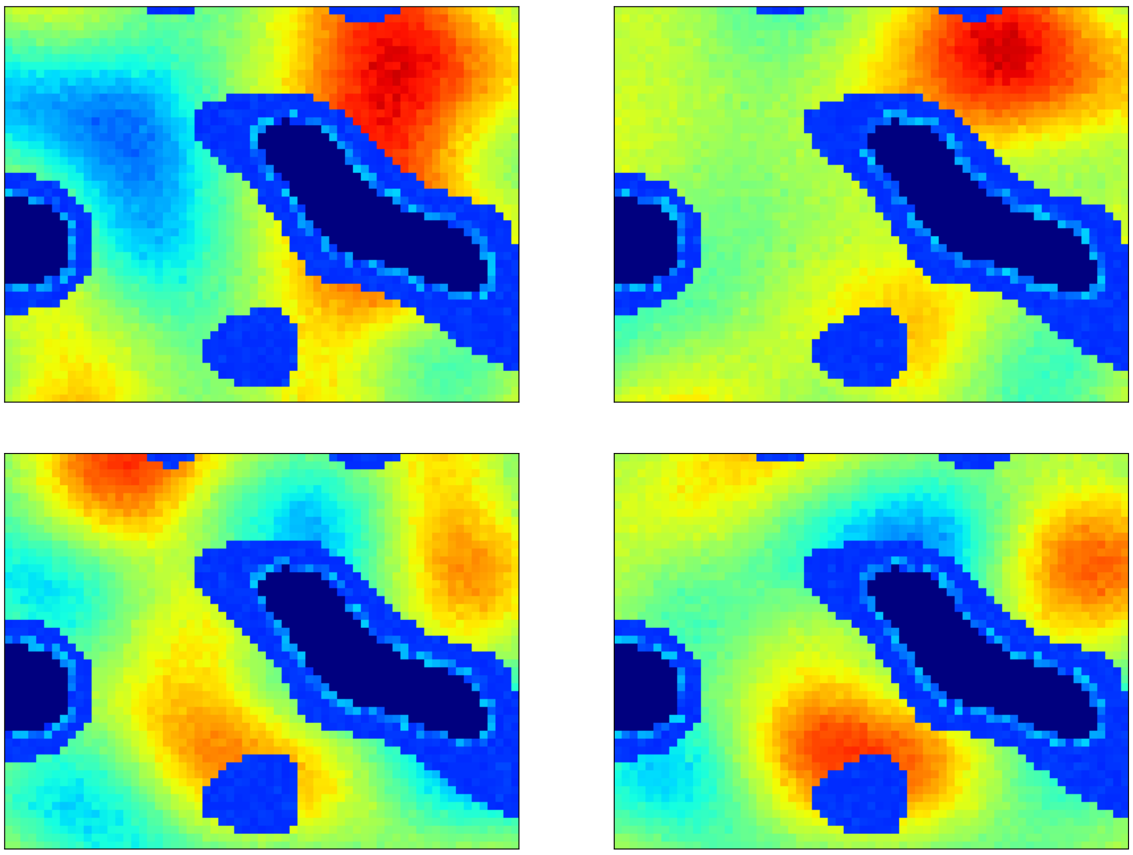


Figure 20. This figure emphasizes the velocities of the drift ice. The land pixels are **dark** blue, the fast ice is in a lighter blue shade vel ≈ 0. The values from the drift ice are in between 0 to 5 from light blue to dark red. Notice how the the different velocity regions stay more or less consistent but also move around and i.e. diffuse a bit. At other times there will also be jumps in speeds due to *weather and current changes.*

Parameters		
Name	Description	Range[16]
shape	Size of grid to simulate in pixels	(30,40) to (100,120)
cycleLength	The number of observations for a winter cycle	100 to 200
amounFI	Percentage of the maximal amount of fast ice	.2 to .4
amountLand	Percentage of land	.05 to .15
fluctuation	Fluctuation of fast ice and error rates per step	.005 to .05
noise	Variance of the normal distribution for noise	.01 to .2
errorRate	Expected percentage of missing data points	0 to .5
arealError	Relative size of grouped errors	.5 to .95
clustered	How clustered the fast ice and land is	1.5 to 3
maxSpeed[17]	Maximal velocity of the drift ice	2 to 10
steepness	The tilt of the slopes in fig 19	> 15
seed	The seed to generate the data	any integer

TABLE 3. Parameters for the simulation

Lastly, the noise and errors are added (iii). The noise is fairly basic and only consists of a pointwise normal distribution with low variance added onto all the speed values, including the fast ice, especially here we need to be aware of any negative values arising.

For the errors two different kinds are modeled, pointwise and grouped errors. The idea being that the first simulates the computational errors of drift ice, e.g., outliers of vectors which get filtered out, the latter covers broader regions and aims to reproduce the effects of larger smooth ice regions with little to no texture where no real drifts can be computed. Pointwise errors are sampled from a binomial distribution $bin(n, p)$ with n the amount of pixels and p indirectly chosen through the parameter *errorRate* and *arealError* describing how many total expected errors there are and the relative amount of grouped errors. Every pixel has an independent error chance, independent not only in space, but also in time. Again the natural look of Perlin noise is used to produce the grouped errors. It also uses a linear interpolation as the drift ice simulation for some consistency in time, but in a more agitated way. In every step a threshold is computed to again choose which parts of the Perlin noise count as errors. The errors are then masked over everything except the land pixels.

Putting all of this together we get an algorithm seen in figure 21.

In total we end up with 12 parameters to customize the simulation, from cover percentages to structural changes. In table 3 the parameters are named and described, further the in the numerical part used ranges of these parameters are denoted. Putting everything together a simulated observation sequence is shown in figure 3 (in the main body of this thesis). The code implementing this simulation is found in the GitHub [25].

[16]These are the values for which the simulated data makes sense (meaning: it fits what we try to achieve).
[17]This parameter mostly changes the visualization as the values are spread further apart or closer together.

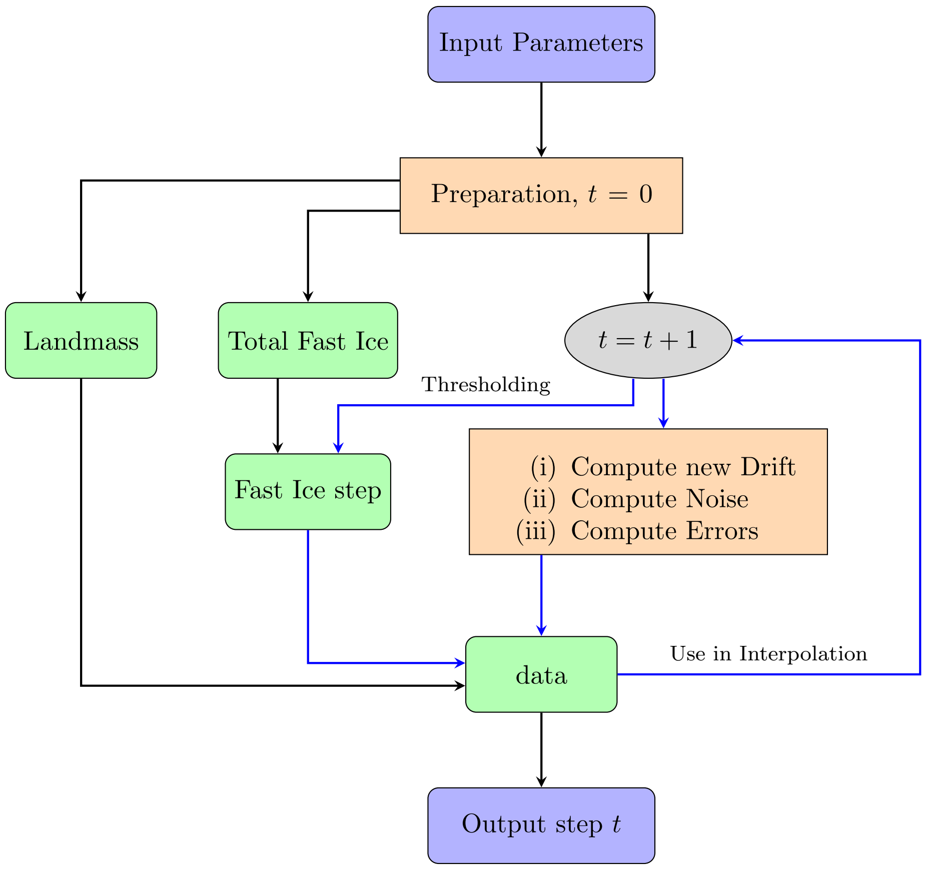


FIGURE 21. Flowchart of the simulation. Blue arrows indicate the repeated action in every time step t. A green box implies some kind of saved data, orange means a heavy computational step whereas the gray box only computes a little bit. Starting with the parameters the landmass and the full fast ice computed, then in every step the fast ice gets reduced by a threshold and every other part is computed. In the end of every step t everything is combined as an output. For the next step the last results get reused as described in the text.

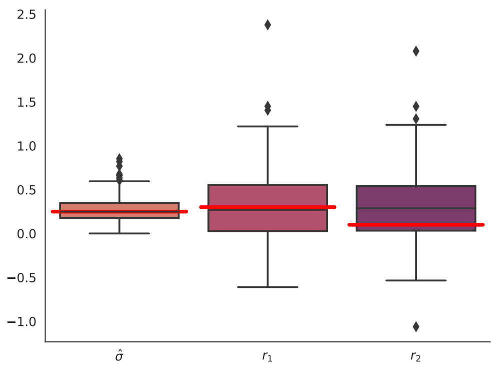
Appendix B. Results and further Graphs

In this appendix further results are shown and discussed and more graphics are presented.

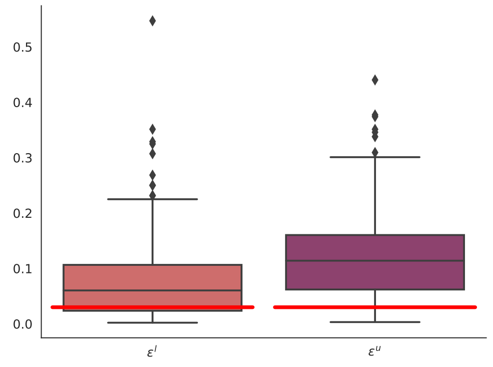
B.1. **Parameter Optimization.** Three scenarios are covered, the most important one in the main text (section 2.5.4), here stated as number (i)

(i) normal error rate (30%) with little noise,
(ii) strong error rate (60%) with little noise,
(iii) low error rate (20%) with strong noise.

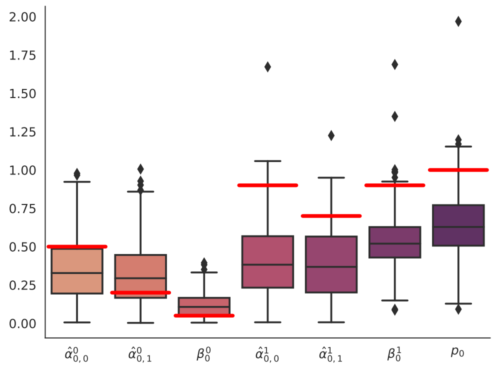
The results for the other two settings yield the following results, depicted in figures 22 and 23.



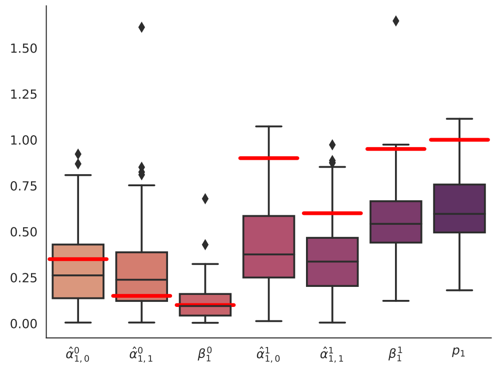
(A) σ_t



(B) g_0 and g_1

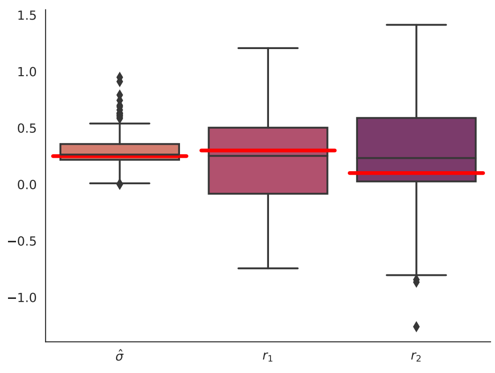


(C) g_0

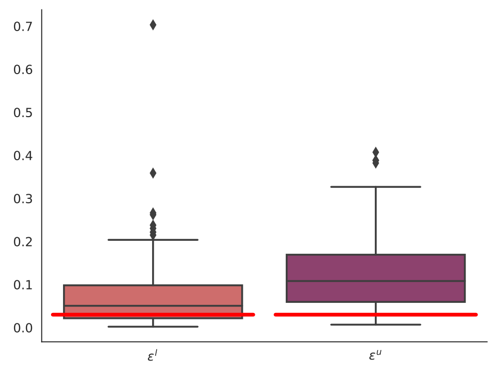


(D) g_1

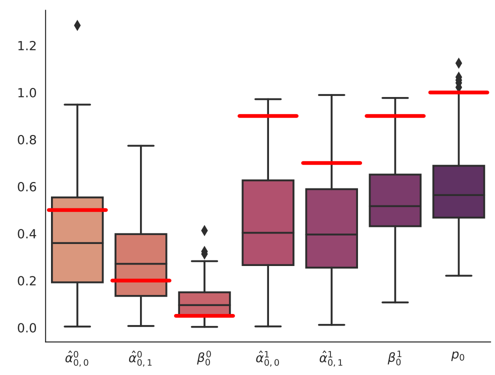
FIGURE 22. Results from the parameter optimization for (ii), 60% errors and low noise level. (A) are the parameters for the variance computation in the transition density. (B) shows the two global function parameters for g_0 and g_1. (C) and (D) are the unique parameters for each of the fast ice likelihood functions. The red line is the initial starting guess.



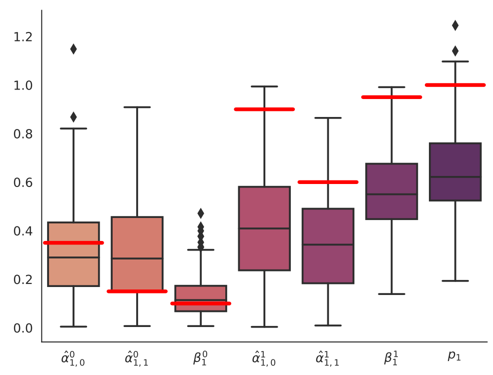
(A) σ_t



(B) g_0 and g_1



(C) g_0



(D) g_1

FIGURE 23. Results from the parameter optimization for (iii), 20% errors and a strong noise level. (A) are the parameters for the variance computation in the transition density. (B) shows the two global function parameters for g_0 and g_1. (C) and (D) are the unique parameters for each of the fast ice likelihood functions. The red line is the initial starting guess.

In this segment the proofs will be stated and further computations are made.

C.1. **Lemma, Transition Kernels as Conditional Distributions.** Found in section 2.1

Lemma. *Let $X_{0:1}$ be distributed according to $\mathbb{P}_1(dx_{0:1}) = \mathbb{P}_0(dx_0)P_1(x_0, dx_1)$, then the conditional distribution of X_1 given $X_0 = x_0$ is exactly given by $P_1(x_0, \cdot)$.*

To proof this Lemma we will use a helping Theorem from *Foundations of modern probability by O. Kallenberg 1997* [12] (Theorem 5.3, page 84).

Theorem. *For two Borel spaces $(\mathcal{X}, \mathcal{B}(\mathcal{X}))$, $(\mathcal{Y}, \mathcal{B}(\mathcal{Y}))$ and $\hat{x} \in \mathcal{X}$, $dy \in \mathcal{B}(\mathcal{Y})$ there exists a probability kernel P from $(\mathcal{X}, \mathcal{B}(\mathcal{X}))$ to $(\mathcal{Y}, \mathcal{B}(\mathcal{Y}))$ s.t. $\mathbb{P}(y \in dy \,|\, \hat{x}) = P(\hat{x}, dy)$ a.s., P is unique.*

Proof. Proof is found on page 84 of [12]. □

Proof of Lemma. The conditional probability of X_1 given $X_0 = x_0$ can be written in the decomposition of $\mathbb{P}_1$

$$\mathbb{P}_1(dx_{0:1}) = \mathbb{P}_0(dx_0)\mathbb{P}(dx_1 \,|\, x_0).$$

Given by the theorem above, there exists a kernel P s.t. $\mathbb{P}(dx_1 \,|\, x_0) = P(x_0, dx_1)$. From the distribution of $X_{0:1}$ $(\mathbb{P}_0(dx_0)\mathbb{P}(dx_1 \,|\, x_0) = \mathbb{P}_0(dx_0)P_1(x_0, dx_1))$ and the uniqueness of P follows $P = P_1$. □

C.2. **Lemma 4, Likelihood Factors.**

Lemma. *The conditional densities are given as the ratios of two consecutive likelihood densities:*

$$p_t(y_t | y_{0:t-1}) = \frac{p_t(y_{0:t})}{p_{t-1}(y_{0:t-1})}$$

Proof. We start by expanding the likelihood $p_t(y_{0:t})$ in terms of conditional densities using Bayes' Theorem.

$$\begin{aligned}
p_t(y_{0:t}) &= p_t\left(y_{1:t} \,|\, y_0\right) \, p_t(y_0) \\
&= p_t\left(y_{2:t} \,|\, y_{0:1}\right) \, p_t\left(y_1 \,|\, y_0\right) \, p_t(y_0) \\
&= p_t(y_0) \prod_{s=0}^{t} p_t\left(y_s \,|\, y_{0:s-1}\right) \\
&= p_t(y_0) \prod_{s=0}^{t} p_s\left(y_s \,|\, y_{0:s-1}\right)
\end{aligned}$$

The last equality follows from $p_t(y_{0:s}) = p_s(y_{0:s})$, $\forall s \leq t$, which is a direct application of (2.9) in (2.15). This last line directly shows the claim in the lemma. □

C.3. **cost-to-go Function.** Computation of the equality (2.27)

$$H_{t:T}(x_t) = p_T(y_{t+1:T} \,|\, x_t)$$

from section 2.2.2 for the Bootstrap formalism (def 6). We start by showing (2.25) for $t = T-1$ and then prove the induction step $t \to t-1$.

$$\begin{aligned}
H_{T-1:T}(x_{T-1}) &= \int_{\mathcal{X}} G_T(x_{T-1}, x_T)\, M_T(x_{T-1}, dx_T) \\
\text{Boostrap formalism} \to \quad &= \int_{\mathcal{X}} f_T(y_T \,|\, x_T)\, P_T(x_{T-1}, dx_T) \\
\text{def. of kernel} \to \quad &= \int_{\mathcal{X}} f_T(y_T \,|\, x_T, x_{T-1})\, p_T(x_T \,|\, x_{T-1})\, \mathrm{d}x_T \\
y_T \,|\, x_T \text{ independent of } x_{T-1} \to \quad &= \int_{\mathcal{X}} p_T(y_T, x_T \,|\, x_{T-1})\, \mathrm{d}x_T \\
&= p_T(y_T \,|\, x_{T-1}).
\end{aligned}$$

Assume, the statement holds for a $t < T$, then:

$$\begin{aligned}
(2.26) \to H_{t:T}(x_t) &= \int_{\mathcal{X}} G_{t+1}(x_t, x_{t+1})\, H_{t+1:T}(x_{t+1})\, M_{t+1}(x_t, dx_{t+1}) \\
\text{induction step} \to \quad &= \int_{\mathcal{X}} G_{t+1}(x_t, x_{t+1})\, p_T(y_{t+2:T} \,|\, x_{t+1})\, M_{t+1}(x_t, dx_{t+1}) \\
\text{Bootstrap formalism} \to \quad &= \int_{\mathcal{X}} f_{t+1}(y_{t+1} \,|\, x_{t+1})\, p_T(y_{t+2:T} \,|\, x_{t+1})\, p_{t+1}(x_{t+1} \,|\, x_t)\, \mathrm{d}x_{t+1} \\
y_{t+1} \,|\, x_{t+1} \text{ indep. of } y_{t+2:T} \,|\, x_{t+1} \to \quad &= \int_{\mathcal{X}} p_T(y_{t+1:T} \,|\, x_{t+1})\, p_{t+1}(x_{t+1} \,|\, x_t)\, \mathrm{d}x_{t+1} \\
y_{t+1:T} \,|\, x_{t+1} \text{ independent of } x_t \to \quad &= \int_{\mathcal{X}} p_T(y_{t+1:T}, x_{t+1} \,|\, x_t)\, \mathrm{d}x_{t+1} \\
&= p_T(y_{t+1:T} \,|\, x_t).
\end{aligned}$$

C.4. **Lemma 9, Marginal Distributions of $\mathbb{Q}_T$.**

Lemma. *For all $t < T$*

$$\mathbb{Q}_T(dx_{0:t}) = \frac{L_t}{L_T} H_{t:T}(x_t)\, \mathbb{Q}_t(dx_{0:t}),$$

integrating over $x_{0:t-1}$ leads to the marginal distributions

$$\mathbb{Q}_T(dx_t) = \frac{L_t}{L_T} H_{t:T}(x_t)\, \mathbb{Q}_t(dx_t).$$

Proof. We use Lemma 8 and equation (2.9).

$$\begin{aligned}
\text{Markov process} \to \quad \mathbb{Q}_T(dx_{0:t}) &= \mathbb{Q}_{0|T}(dx_0) \prod_{s=1}^{t} Q_{s|T}(x_{s-1}, dx_s) \\
\text{Lemma 8 \& telescope } H_{s:T} \to \quad &= \frac{1}{L_T} G_0(x_0)\, H_{t:T}(x_t)\, \mathbb{M}_0(dx_0) \prod_{s=1}^{t} G_s(x_{s-1}, x_s)\, M_s(x_{s-1}, dx_s) \\
\text{def. of } \mathbb{Q}_t \to \quad &= \frac{L_t}{L_T} H_{t:T}(x_t)\, \mathbb{Q}_t(dx_{0:t})
\end{aligned}$$

The integration to reach the marginal distribution is straight forward. □

C.5. **Backward Kernels for Smoothing.** Computation of the equality (2.33)

$$\mathbb{Q}_t(dx_t)\,\overleftarrow{Q}_{t-1|T}(x_t, dx_{t-1}) = \frac{1}{l_t} G_t(x_{t-1}, x_t)\,\mathbb{Q}_{t-1}(dx_{t-1})\,M_t(x_{t-1}, dx_t)$$

from section 2.2.2. Starting point is the defining equality (2.30) for the backward kernel

$$\begin{aligned}
\mathbb{Q}_T(dx_t)\,\overleftarrow{Q}_{t-1|T}(x_t, dx_{t-1}) &\overset{(2.30)}{=} \mathbb{Q}_T(dx_{t-1})\,Q_{t|T}(x_{t-1}, dx_t)\\
\frac{L_t}{L_T} H_{t:T}(x_t)\,\mathbb{Q}_t(dx_t)\,\overleftarrow{Q}_{t-1|T}(x_t, dx_{t-1}) &\overset{(2.32)}{=} \frac{L_{t-1}}{L_T} H_{t-1:T}(x_{t-1})\,\mathbb{Q}_{t-1}(dx_{t-1})\,Q_{t|T}(x_{t-1}, dx_t)\\
\mathbb{Q}_t(dx_t)\,\overleftarrow{Q}_{t-1|T}(x_t, dx_{t-1}) &\overset{(2.21)}{=} \frac{1}{l_t}\frac{H_{t-1:T}(x_{t-1})}{H_{t:T}(x_t)}\,\mathbb{Q}_{t-1}(dx_{t-1})\,Q_{t|T}(x_{t-1}, dx_t)\\
\mathbb{Q}_t(dx_t)\,\overleftarrow{Q}_{t-1|T}(x_t, dx_{t-1}) &\overset{(2.29)}{=} \frac{1}{l_t} G_t(x_{t-1}, x_t)\,\mathbb{Q}_{t-1}(dx_{t-1})\,M_t(x_{t-1}, dx_t).
\end{aligned}$$

C.6. **Lemma 11, Backwards Feyman-Kac Model.**

Lemma. *When taking*

$$\begin{aligned}
\overleftarrow{G}_T(x_T) &= \frac{f_T(y_T \mid x_T)\,\gamma_T(dx_T)}{\overleftarrow{\mathbb{M}}_T(dx_T)},\\
\overleftarrow{G}_t(x_{t+1}, dx_t) &= \frac{f_t(y_t \mid x_t)\,\gamma_t(dx_t)\,P_{t+1}(x_t, dx_{t+1})}{\gamma_{t+1}(dx_{t+1})\,\overleftarrow{M}_t(x_{t+1}, dx_t)},
\end{aligned}$$

the defined Feyman-Kac model fits the required (2.50)*. It is assumed that the Radon-Nikodym derivative exists.*

Proof. We have (up to a normalization constant)

$$Q_t(dx_t) \propto \int_{\mathcal{X}^{T-t-1}} \overleftarrow{\mathbb{M}}_T(dx_T)\,\overleftarrow{G}_T(x_T) \prod_{s=t}^{T-1} \overleftarrow{M}_s(x_{s+1}, dx_s)\,\overleftarrow{G}_s(x_{s+1}, x_s)$$

when integrating over $x_{t+1:T}$. Inserting the defined G_t from the lemma yields

$$\begin{aligned}
Q_t(dx_t) \propto &\int_{\mathcal{X}^{T-t-1}} f_T(y_T \mid x_T)\,\gamma_T(dx_T) \prod_{s=t}^{T-1} \frac{f_s(y_s, x_s)\,\gamma_s(dx_s)\,P_{s+1}(x_s, dx_{s+1})}{\gamma_{s+1}(dx_{s+1})}\\
&= \int_{\mathcal{X}^{T-t-1}} \gamma_t(dx_t)\,f_t(y_t \mid x_t) \prod_{s=t+1}^{T} f_s(y_s, x_s)\,P_s(x_s, dx_{s+1})\\
&= \gamma_t(dx_t)\,f_t(y_t \mid x_t) \int_{\mathcal{X}^{T-t-1}} \prod_{s=t+1}^{T} f_s(y_s, x_s)\,P_s(x_s, dx_{s+1})\\
&\overset{(2.25)}{=} \gamma_t(dx_t)\,f_t(y_t \mid x_t)\,H_{t:T}(x_t)\\
&\overset{(2.27)}{=} \gamma_t(dx_t)\,f_t(y_t \mid x_t)\,p_T(y_{t+1:T} \mid x_t)
\end{aligned}$$

□

C.7. **Full Smoothing Distribution.** Computation of the equality (2.46)

$$\mathbb{P}_T(dx_{0:T}\,|\,Y_{0:T}=y_{0:T}) = \mathbb{P}_T(dx_T\,|\,Y_{0:T}=y_{0:T})\prod_{t=0}^{T-1}\overleftarrow{P}_{t|t}(x_{t+1},dx_t)$$

using basic independence rules:

$$\begin{aligned}
\text{RHS} &= \mathbb{P}_T(dx_T\,|\,Y_{0:T}=y_{0:T})\prod_{t=0}^{T-1} p_t(x_t\,|\,x_{t+1},y_{0:t})\\
\text{Ind. of } y_{t+1:T} \rightarrow \quad &= \mathbb{P}_T(dx_T\,|\,Y_{0:T}=y_{0:T})\prod_{t=0}^{T-1} p_t(x_t\,|\,x_{t+1},y_{0:T})\\
\text{Def. trans. kernel} \rightarrow \quad &= \mathbb{P}_T(dx_T,dx_{T-1}\,|\,Y_{0:T}=y_{0:T})\prod_{t=0}^{T-2} p_t(x_t\,|\,x_{t+1},y_{0:T})\\
&= \dots\\
&= \mathbb{P}_T(dx_{0:T}\,|\,Y_{0:T}=y_{0:T})
\end{aligned}$$

C.8. **Pre-Requirements.** Checking the pre-requirements for Lemma 11.

$$\overleftarrow{G}_T(x_T) = \frac{f_T(y_T\,|\,x_T)\,\gamma_T(dx_T)}{\overleftarrow{\mathbb{M}}_T(dx_T)}$$

is well defined by definition, since $\overleftarrow{\mathbb{M}}_T(dx_T)$ is defined through the forward filtering and all particles either have weight > 0, or when a weight is $= 0$ for a particle it will not be chosen.

Taking a look at the next steps

$$\overleftarrow{G}_t(x_{t+1},dx_t) = \frac{f_t(y_t\,|\,x_t)\,\gamma_t(dx_t)\,P_{t+1}(x_t,dx_{t+1})}{\gamma_{t+1}(dx_{t+1})\,\overleftarrow{M}_t(x_{t+1},dx_t)}$$

it is the case that $\gamma_{t+1}(dx_{t+1})$ can theoretically not be $= 0$ (in practice it can through underflow), as the transition density f_t is by construction > 0 for all x_t, y_t. Since we did not chose $\overleftarrow{M}_t$ as the optimal kernel we miss the direct connection to the forward kernel P_{t+1}. This would have given us the domination needed, instead we have to make sure that our $\varepsilon^u, \varepsilon^l$ (see def 12) are > 0 to assure us, that $\overleftarrow{M}_t > 0$ as well (again, only theoretical, in practice underflows are a real possibility, which we need to handle properly).

Variables and Parameters, Stochastic Programming

All variables specific for Stochastic Programming.

Functions and Variables.

f	Objective function we try to minimize.
J	Minimization of f and its expected value.
$\hat{\mathcal{X}}$	Set of feasible decisions.
g_k	Constraint functions.
ϕ	infimum of the objective function.
Φ	argument infimum of the objective function.
F_t	empirical Cumulative Distribution Function with t given observations.

Appendix D. Approach One: Stochastic Programming

Stochastic Programming aims to minimize a given cost function to make the next decision depending on the past decisions and observations as well as constraints on the decision vector. Important in each decision step is the modeling of the future decision, also impacting the minimization process. Here a probability distribution of our observations is needed [23]. In the following the basic model is described, as reference the book *Stochastic Programming Models* by *A. Ruszczyński et al., 2003* is used [23]. We then cover a special case of stochastic programming, linear stochastic programming, including a numerical example. In the end we go into the flaws of the named approach related to our specific conceptual formulation.

D.1. **Method Description.**

D.1.1. *Basic Framework.* First, a space for decisions and observations is needed, denoted by $\mathcal{X}$ and $\mathcal{Y}$ respectively. Both spaces may change from one step to the next, but are (here) always considered to be real and finite dimensional, e.g., $\mathbb{R}^n$. The observations $Y_t \in \mathcal{Y}^{(t)}$ are thought to be distributed according to a probability measure $P^{(t)}$ (the (t) indicates a possible alternation in space/distribution) and do not need to be independent of one another. In the same way the decisions $X_t \in \hat{\mathcal{X}}^{(t)} \subset \mathcal{X}^{(t)}$ are allowed to differ in space, where $\hat{\mathcal{X}}^{(t)}$ is the set of feasible decisions in iteration t. At last we need an objective function $f(x_{1:t}, y_{1:t})$ which we assume to be *convex* in $x_{1:t}$, this function also defines our standard set of feasible decisions if no constraints are given $\hat{\mathcal{X}}_0^{(t)}(y_{1:t}) \coloneqq \{x_t \in \mathcal{X}^{(t)} \mid f(x_{1:t}, y_{1:t}) < +\infty\}$. This objective function rises costs on the decisions and should be apportionable into parts which cover single decisions (e.g., $f(x_0, x_1, y) = f_0(x_0) + f_{1,(x_0,y)}(x_1)$).

The goal of minimizing $f(x_{1:t}, y_{1:t})$ leads directly to the idea of Stochastic Optimization, because when $Y_{1:t}$ is only partly or not at all observed yet there is no real minimum of f. In order to achieve a meaningful solution the expectation of f is minimized, so we get:

$$\min_{x_{1:t} \in \hat{\mathcal{X}}^{(1:t)}} J(x_{1:t}),\ J(x_{1:t}) \coloneqq \mathbb{E}_{y_{k:t}}\left[f(x_{1:t}, y_{k:t})\right] \tag{D.1}$$

which is an integral of the form[18]

$$J(x_{1:t}) = \int_{\mathcal{Y}^{(k:t)}} f(x_{1:t}, y_{k:t})\, \mathrm{d}P(y_{k:t}). \tag{D.2}$$

The general feasibility criteria from above is not adequate to guarantee finiteness of J in the occurrence of a continuous random variable, but in our case only discrete random variables are employed dictated by an empirical Cumulative Distribution Function (eCDF). One property we can directly deduce is convexity.

Lemma 15. *The expected cost function $J(x)$ is convex.*

Proof. D.4 □

We now want to add constraints.

D.1.2. *Modeling Constraints.* To introduce constraints is to *shrink* our set of feasible decisions. In a standard way the feasible set is written as

$$\hat{\mathcal{X}}^{(t)} := \{x_t \in \hat{\mathcal{X}}_0^{(t)} \mid g_k(x_t) \leq 0,\ k = 1, \ldots, m\} \tag{D.3}$$

with $\hat{\mathcal{X}}_0$ being the standard feasible set and $g_k(x)$ real valued functions.

And how is set of feasible decisions in an uncertain scenario defined?

X_t depends on the observations $Y_{1:t}$ and can therefore meet or fail constraints under certain observations. A simple way of dealing with this is to neglect that fact and wait for the realization of $y_{1:t}$ if possible, thus the problem to solve is not fully known until all observations are made. A different idea is to consider the expected value of the constraints, this is often used when some objectives of the program are put into the constraints. [19]

D.1.3. *The Two Stage Program.* The two stage program covers the fundamental structure and can then be easily extended into a multistage problem. We have two decisions X_0, X_1 to make, x_0 being an initial decision and X_1 the decision depending on the *only* observation Y. The problem reads as in equation D.1 but just with X_0, X_1:

$$\min_{\substack{x_0 \in \hat{\mathcal{X}}^{(0)} \\ x_1 \in \hat{\mathcal{X}}^{(1)}}} \mathbb{E}_y \left[f(x_0, x_1, y) \right]. \tag{D.4}$$

The set $\hat{\mathcal{X}}^{(1)}$ depends on the Y as well. Splitting this problem (D.4) into the parts of *before* and *after* the observation yields as a second step (in which y is already observed and x_0 already set) the parametric optimization problem

$$\min_{x_1 \in \hat{\mathcal{X}}^{(1)}} f_1(x_0, x_1, y), \tag{D.5}$$

[18]Here the $P(y_{k:t})$ is either a marginal distribution (e.g., the next observation in the future with $k = t$) or the joint distribution of multiple/all ($k < t$).

[19]In my problem I can easily wait for the observation, in others, e.g., flood prevention as in example 2 *Reservoir Capacity* of [23] this is not possible, a decision must be made well before the observation of the flood.

with solution and solution arguments (if existing)

$$\varphi(x_0, y) := \inf_{x_1 \in \hat{\mathcal{X}}^{(1)}} \{f_1(x_0, x_1, y)\} \tag{D.6a}$$

$$\Phi(x_0, y) := \arg\min_{x_1 \in \hat{\mathcal{X}}^{(1)}} \{f_1(x_0, x_1, y)\} \tag{D.6b}$$

The first part has Y still in random state and hereby introduces the known expected value. It is a minimization of the expectation of the next step summated with the cost of the initial decision

$$\min_{x_0 \in \hat{\mathcal{X}}^{(0)}} f_0(x_0) + \mathbb{E}_y \left[\varphi(x_0, y)\right]. \tag{D.7}$$

The optimal solution of this deconstruction indeed accords with the original two stage problem (D.4):

Lemma 16. *The optimal values of equation* (D.4) *and equation* (D.7) *coincide. If in addition this value is finite then the decision* $\bar{x}(dy) = (\bar{x}_0, \bar{x}_1(dy))$ *is optimal for* (D.4) *iff*

- x_0 *is an optimal solution to* (D.7)*, and*
- $\bar{x}_1(y) \in \Phi(\bar{x}_0, y)$ *almost surely.*

Proof. D.5 □

D.2. **The Stochastic Programming Model (Linear).** Coming up with a suitable stochastic programming model is a challenge in and of itself, knowing how to solve it a completely different and (most often) way more complicated one. Luckily, under certain preconditions general solving mechanisms exist, hence it is logical to try to formulate the problem under these conditions first to not only guarantee a working program, but also check if stochastic programming may be a viable strategy for modeling (methodically). Two basic models are the linear and the quadratic one, here also the *existence* of optimizing solutions are given. In the linear programming model the objective function and the constraints need to be affine. The only difference to the quadratic program is the usage of a quadratic objective function in the latter. In the following construction it gets evident why in our case the linear suffices over the quadratic program.

D.2.1. *The construction.* We start by listing the inherent factors of the upcoming model and then go over to cover the key properties which are required to be included in the model. Thereafter different ideas of how to translate these properties into the model are discussed, and finally the program is formulated and implemented.

The observations y_t in every time step are real valued (non-negative), two dimensional matrices representing the velocity of the ice on the water. Every value is positive except the land value which can be seen more as a mask and the error values being $-\infty$. The size and amount of the matrices depend solely on the selected values in the simulation part and should be chosen representatively. Since we are interested in the fast ice (zero velocity over a period of time) the decisions are also matrices of the same size, containing a speed value as well, in a sense the *averaged expected* value. This is easier to translate but more importantly leaves a broader range of options on how to use the data, e.g., thresholding into a $(0, 1)$ choice or a more continuous allocation of likelihood. So here we get the explicit spaces:

$$y_t \in Mat_{m \times n}(\mathbb{R}_+), \ t = 1, \dots, T$$
$$x_t \in Mat_{m \times n}(\mathbb{R}_+), \ t = 0, \dots, T$$

In the danger of repeating, these are the main features of fast ice. Fast ice is only seen as fast ice if it does stay in the same spot over a longer period, thus every new decision made should greatly depend (in the way of being close to it) on past decisions and observations. Also, the spatial relation is important, but it gets really awkward when directly included in the linear case. Instead, a certain behavior from the observation distribution indirectly introduces this important attribute.

So how does this distribution look like?

That is a really tough question to answer, so in short: No idea! The real distribution is based on a natural occurrence with varying characteristics and topological changes over time. These dynamics consist, even if we only focus on the static ice and neglect (in thought) the values in the other parts of the matrix[20]. Crucial is also the resemblance in spatial coherence of the fast ice, satisfying this point institutes the indirect areal relation we want. All this leads to what the simulation (A.2) is trying to mimic, but to simulate data does not infer the existence of a corresponding probability distribution to it. A rather simple idea to still have a distribution to work with is to use an eCDF based on the observations themselves. Of course this has major flaws, they will be discussed in section D.3.

$$F_n(r) = \frac{1}{t}\sum_{i}^{t} \underbrace{\mathbb{1}_{\{y_i \leq r\}}}_{\text{Pointwise}}, \; t = 1, \dots, T \tag{D.8}$$

Using this distribution captures almost all of the wanted behaviors of fast ice and does not only make sense in a point wise manner but even asks for a point wise sampling since the neighbourhood relations are already contained in the observations and this way in the drawn samples as well. Same goes for the computational aspects, in any sampled or measured pixel of the matrix most necessary regional information is accorded for. One weakness is the arising of missing or defective measurements, but I argue that any deviations in velocity will be filtered out over time (an advantage of stochastic approaches) and since missing data points mostly cover wider regions (As derived in A.2) there is no advantage in checking neighbouring pixels for information. Moreover by using neighbouring pixels there will be no distinct edges on the deduced fast ice, it is basically an interpolation. In this sense, the eCDF is a pretty good choice.

And how is the temporal context introduced?

To achieve in time consistent decisions we have to utilize the affine cost and constraint functions in a way that reflects the behavior of fast ice. If a piece of ice (i.e. one pixel) does not move in the latest observation but did move ($\gg 0$) in the last observation and/or decision then it is not really qualified for being fast ice. Only when the velocity stays below the defined margin of error for the chosen two weeks time it qualifies. This means, when making a decision either the constraints or the objective function (or both) need to be accommodated for not just the last, but n last decisions and observations. Looking at the objective function in any step $t \leq T$ we try to minimize a linear combination of the values in the possible decisions $X_t(dy_{1:t})$, if translated into a vector of length mn, f can be represented[21] as a standard scalar product $f_t(x_t(y_{1:t}), y_{1:t}) = c_t(y_{1:t}) \cdot x_t(y_{1:t})$ where $c_t(y_{1:t}) \in \mathbb{R}^{nm}$ also depends on the past decisions $x_{0:t-1}$. There are two main ideas now, either to minimize every element in $x_t(y_{1:t})$ by setting $c_t(y_{1:t}) \equiv 1$ and letting the constraints (from below) do the work, or to set $c_t(y_{1:t})$ element-wise $\geq$ or < 0 depending on the observations in the attempt to nudge the values in the right direction (< 0 favors bigger and ≥ 0 smaller speeds). The mathematical embodiment of the first idea is described in

[20]Meaning, it does not really matter what velocity values non-fast ice pixels get as long as they are distinguishable from the actual fast ice.

[21]This is an direct application of functional analysis, *Riesz representation theorem*[1].

appendix E and is (after a bit of work) equivalent to a weighted average, though not really exciting it still delivers quite a reasonable result to use as a basis for comparison, see chapter 3. It also gives a new perspective into a standard weighted average. Viewing these two concepts, it is fairly obvious that a quadratic objective function does not add any noteworthy value, if it were possible to reach a notion of distance there would be plenty more options, this however, is sadly not possible here. Before we go any further, let us first take a look at the downsides of stochastic programming.

D.3. **Non-Linear Cases and Problems with the Approach.**

In the derivation of a linear problem it became quite clear that to achieve a thorough description of our scenario we need more a complex mathematical scope, exceeding linear or quadratic formulation. Especially the part of constructing an objective function to minimize is challenging and requires e.g., some notion of distance between two time-consecutive or space-related points to allow for a noteworthy objective function. There are methods to solve particular non-linear programming problems by iterating over linear ones (see e.g., [13]) and also more general methods for specific classes of stochastic programming (see [9], here there are no constraints intended but the objective function can be non-convex and non-linear as long as its gradient exists in a noisy, point wise matter and also fulfills a Lipschitz inequality). Coming up with a function which has a general solving method has huge limits and supposing we find a good one, three essential problems remain:

Viewing our frame conditions again one, can notice having uncertainty in the future observations is not the key problem to solve here, it is rather to optimize for the uncertainty in the already occurred observations. Instead of *guessing* part of the upcoming observation the goal should be to *guess* the correct velocity measurement, assuming they exists in some *all known universe*. Exactly this is the major flaw in the stochastic programming approach, independent of the individual program itself (let it be linear, quadratic or a more complicated approach). Secondly, there is no natural way of altering the already made decision when new information is available, e.g., changing decision x_i when observation w_{i+1} is made. This is actually quite important but would need further mathematical extensions, doing for example some kind of back propagation. Lastly, finding a suitable distribution for the fast ice or velocity distribution is not in reach (as discussed above D.2.1), but using an eCDF has some drawbacks on its own. First, it is rather imprecise with a few observations and only gradually gets better (if at all with merely 150 observations). In addition our situation is a bit unique and imposes a changing distribution depending on the time, an eCDF will always *lack behind* the actual distribution. As an example, towards summer the fast ice starts melting and breaking while the eCDF only just *learned* that fast ice grows steadily and/or remains consistent.

Thus in the upcoming chapter we take a look at a different idea, but try to stick to the same notation, to not deviate too far from already introduced vocabulary.

D.4. **Lemma 15, Convex Cost Function $J(x)$.**

Lemma. *The expected cost function $J(x)$ is convex.*

Proof. Let $x, \hat{x} \in \hat{\mathcal{X}}^{(i)}$ be two feasible decisions for $y \in \mathcal{Y}$. Then for any $t \in [0,1]$ the decision $z(t) = tx + (1-t)\hat{x}$ fulfills:

$$\begin{aligned} J(z(t)) &= \mathbb{E}_y\left[f(z(t), y)\right] \\ &= \mathbb{E}_y\left[f(tx + (1-t)\hat{x}, y)\right] \\ &\le \mathbb{E}_y\left[tf(x,y) + (1-t)f(\hat{x}, y)\right] \\ &= t\mathbb{E}_y\left[f(x,y)\right] + (1-t)\mathbb{E}_y\left[f(\hat{x}, y)\right] \\ &= tJ(x) + (1-t)J(\hat{x}) \end{aligned}$$

where we used the convexity of f and monotonicity of the expectation in the third line and the linearity of the expectation for the rest. □

D.5. **Lemma 16, Optimal Solution.** The two main equations for this Lemma:

$$\text{(D.4)} \quad \min_{\substack{x_0 \in \hat{\mathcal{X}}^{(0)} \\ x_1 \in \hat{\mathcal{X}}^{(1)}}} \mathbb{E}_y\left[f(x_0, x_1, y)\right]$$

$$\text{(D.7)} \quad \min_{x_0 \in \hat{\mathcal{X}}^{(0)}} f_0(x_0) + \mathbb{E}_y\left[\varphi(x_0, y)\right].$$

Lemma. *The optimal values of equation* (D.4) *and equation* (D.7) *coincide. If in addition this value is finite then the decision* $\bar{x}(dy) = (\bar{x}_0, \bar{x}_1(dy))$ *is optimal for* (D.4) *iff*

- x_0 *is an optimal solution to* (D.7), *and*
- $\bar{x}_1(y) \in \Phi(\bar{x}_0, y)$ *almost surely.*

Proof. Note with p, p_0 the optimal values of (D.4) and (D.7), assume they are finite. For any *pointwise* feasible decision $x_{0:1} = (x_0, x_1(y)) \in \hat{\mathcal{X}}^{(0:1)}$ we always get

$$\text{(D.9)} \qquad f_1(x_0, x_1(Y), Y) \ge \inf_{x_1 \in \hat{\mathcal{X}}^{(1)}} \{f_1(x_0, x_1, y) = \varphi(x_0, y).$$

When $x_1(y) \in \Phi(x_0, y)$ this inequality is in fact an equality by definition. Integrating now over all y gives the expected value and by adding $f_0(x_0)$ we get

$$\begin{aligned} J(x(dy)) &= f_0(x_0) + \mathbb{E}_y\left[f_1(x_0, x_1(y), y)\right] \\ &\ge f_0(x_0) + \mathbb{E}_y\left[\varphi(x_0, y)\right] \\ &\ge p_0 \end{aligned}$$

If now $\bar{x}_1(y) \in \Phi(x_0, y)$ a.s. then the first inequality must be an equality as before. Moreover, if the first inequality is in fact an equality then $\bar{x}_1(y) \in \Phi(x_0, y)$, else there would be a non empty set over which (D.9) is a strict inequality contravening the equality after integration. Denote with $\underline{x}(dy) := (x_0, \bar{x}_1(dy))$ the partly optimal decision, then

$$\begin{aligned} J(\underline{x}(dy)) &= f_0(x_0) + \mathbb{E}_y\left[f_1(x_0, \bar{x}_1(y), y)\right] \\ &= f_0(x_0) + \mathbb{E}_y\left[\varphi(x_0, y)\right] \\ &\ge p_0 \end{aligned}$$

where the inequality becomes an equality when $\bar{x}_0$ is an optimal solution to (D.7) (exists since we p_0 is finite). So the infimum p is, if finite, equal to p_0. Further, we get the equivalence of the optimal decision as stated in the lemma. □

Remark. This proof only shows the statement for observations which are discretely distributed (as it is in our case), for the continuous case more assumptions on f_1 are needed.

Appendix E. Linear Stochastical Programm - Weighted Average

The first of the two linear stochastic programs relies only on the constraints with the intention to minimize the velocity of every pixel in a decision and reach thereby the most amount of fast ice possible. The corresponding objective function looks like

$$f_t(x_t(y_{1:t}), y_{1:t}) = \sum_{j,k} \underbrace{x_t^{(j,k)}(y_{1:t})}_{\geq 0 \ \forall t,j,k}, \ t = 0, \dots, T \tag{E.1}$$

with the possible minimum of $f(0, y_{1:t}) = 0$ which is always attained with no constraints. We don't mind if any fast ice pixels get set to 0, but every other pixel is in need of a lower bound which fixes a *least possible* velocity. So the constraint functions in step $t \leq T$ are of the form

$$g_t(x_t(y_{1:t})) = \alpha_t + \beta_t(y_{1:t}) - x_t(y_{1:t}), \tag{E.2}$$

creating the feasible sets

$$\hat{\mathcal{X}}^{(t)}(y_{1:t}) = \{x_t(y_{1:t}) \in Mat_{m\times n}(\mathbb{R}_+) \mid \underbrace{x_t(y_{1:t}) \geq \alpha_t + \beta_t(y_{1:t})}_{\text{Pointwise}}\}. \tag{E.3}$$

α_t and $\beta_t(y_{1:t})$ are both real valued matrices generated by a linear combination of past decisions and the observations respectively. This could very well be a non-linear combination, i.e. taking different moments of the values, but even when this enhances the difference between moving and stationary pixels, it also creates huge jumps and errors when a mis-measured movement is evaluated. Still, this leaves some room for experimentation.

Choosing the weights is conditioned on the results we try to achieve and definitely needs to be argued, especially the choice of using older decisions in the bound. A decision in any step should in most events be strongly influenced by the newest observation, and for temporal consistency also older ones. I aver, that the decision to make should stay consistent with the last decisions, at least to some part. In theory there is no noise and no missing data points in them which adds a reliable *base* to which to build upon. That we can not just count on the latest observation but some older ones should be evident.

All together the weights sum up to 1, so we fix the primary *split* into the observations $r_y^{(t)}$ and decision $r_x^{(t)}$ part to then define the weights for each individual observation/decision.

$$r_y^{(t)} \in [0, 1], \tag{E.4a}$$

$$r_x^{(t)} = 1 - r_y^{(t)} \tag{E.4b}$$

In the time $t \leq T$ the observation weights $v_y(j)$ and the decision weights $v_x(j)$ fulfill the inequalities

$$r_y^{(t)} = \sum_{j\leq t} v_y^{(t)}(j), \qquad v_y^{(t)}(t) \geq v_y^{(t)}(t-1) \geq \cdots \geq v_y^{(t)}(0) \geq 0 \tag{E.5a}$$

$$r_x^{(t)} = \sum_{j<t} v_x^{(t)}(j), \qquad v_x^{(t)}(t-1) \geq v_x^{(t)}(t-2) \geq \cdots \geq v_x^{(t)}(0) \geq 0 \tag{E.5b}$$

Notice, there is a 0'th observation. Before any observation there is no base for a decision to make, besides there is no observation distribution at all. The strategy is to use the first observation directly as the first decision and start of the eCDF, afterwards it could be ignored for the sake of compliance to the formulated program, but why waste useful information. we can now write down our constraint matrices

$$\alpha_t = \sum_{j \le t} v_y(j)\, y_j \tag{E.6a}$$

$$\beta_t(y_{1:t}) = \sum_{j<t} v_x(j)\, x_j \tag{E.6b}$$

With this, all the needed information is formulated and the next step is to solve the problem. In every step t a decision is made via the formulated two step approach (D.7), where instead of an initial decision the first part resembles a parameterized minimization problem (here a simple point wise minimization)

$$\min_{x_t \in \hat{\mathcal{X}}^{(t)}} f_t(x_t) + \mathbb{E}_{y_{t+1}}\left[\varphi_t(x_t, y_{t+1})\right], \quad \varphi(x_t, y_k) = \underset{x_{t+1} \in \hat{\mathcal{X}}^{(t+1)}(y_{1:k})}{\textit{infimum}} \{f_{t+1}(x_t, x_{t+1}, y_{1:k})\} \tag{E.7}$$

This is our finalized *linear stochastic programming problem.* In practice there is one other challenge to consider, the lack of data points: How are they handled? For traceability reasons we first rewrite the program into a weighted average and then cover the approach for the missing data points[22].

All our f_j's are point wise minimizations only reliant on the positive sum of the (non-negative) past decisions and observations, therefore the expression (E.7) can be simplified to

$$\min_{x_t \in \hat{\mathcal{X}}^{(t)}} f_t(x_t) \tag{E.8}$$

as the point wise minimization of $f_t(x_t)$ is also the minimum of $\mathbb{E}_{y_{t+1}}\left[\varphi_t(x_t, y_{t+1})\right]$. Now introducing the constraints in (E.8) we get the decision in step t

$$x_t(y_{1:t}) = \alpha_t + \beta_t(y_{1:t}) \tag{E.9}$$

Following this with the definitions from (E.6) yields

$$\begin{aligned} x_t(y_{1:t}) &= \sum_{j=0}^{t} v_y^{(t)}(j)\, y_j + \sum_{j=0}^{t-1} v_x^{(t)}(j)\, x_j \\ &= \sum_{j=0}^{t} v_y^{(t)}(j)\, y_j + \sum_{j=0}^{t-1} v_x^{(t)}(j) \left[\sum_{k=0}^{j} v_y^{(j)}(k)\, y_k + \sum_{k=0}^{j-1} v_x^{(j)}(k)\, x_k \right] \\ &= \sum_{j=0}^{t} v_y^{(i)}(j)\, y_j + \sum_{j=0}^{t-1} v_x^{(t)}(j) \left[\sum_{k=0}^{j} v_y^{(j)}(k)\, y_k + \sum_{k=0}^{j-1} v_x^{(j)}(k)\, [\dots] \right] \\ &= \sum_{j=0}^{t} l_j^{(t)}\, y_j \end{aligned} \tag{E.10}$$

[22]Depending on the formulation, the process of working with lacking data points may be altered.

with (after expanding and rearranging)

$$
\begin{aligned}
l_j^{(t)} &= v_y^{(t)}(j) + v_y^{(t-1)}(j) \left(v_x^{(t)}(t-1) \right) \\
&+ v_y^{(t-2)}(j) \left(v_x^{(t)}(t-1)\, v_x^{(t-1)}(t-2) + v_x^{(t)}(t-2) \right) \\
&+ \dots \\
&+ v_y^{(t-k)}(j) \left[\sum_{h=1}^{k} \prod_{s=0}^{k-h} v_x^{(t-s)}(t-h-s)) \right] \\
&+ \dots \\
&+ v_y^{(j)}(j) \left[\sum_{h=1}^{t-j} \prod_{s=0}^{t-j-h} v_x^{(t-s)}(t-h-s) \right]
\end{aligned} \tag{E.11}
$$

which can be written in a more compact form

$$
l_j^{(t)} = v_y^{(t)}(j) + \sum_{k=1}^{t-j} v_y^{(t-k)}(j) \left[\sum_{h=1}^{k} \prod_{s=0}^{k-h} v_x^{(t-s)}(t-h-s) \right] \tag{E.12}
$$

ending us with a closed form of an weighted average over the observations

$$
x_t(y) = \sum_{j=0}^{t} \left(v_y^{(t)}(j) + \sum_{k=1}^{t-j} v_y^{(t-k)}(j) \left[\sum_{h=1}^{k} \prod_{s=0}^{k-h} v_x^{(t-s)}(t-h-s) \right] \right) y_j \tag{E.13}
$$

Assigning weights to the average has now a different interpretation, giving more mass to the past decisions emphasizes an intended smooth transitions between them, more mass for the direct observations supports quick adjustments. Also the way the weights are distributed within one of these categories stresses different intentions, i.e. the more distributed the more coherent the decisions seem. This all gives the weighted average a different viewpoint which is not accessible without this derivation and computation.

Before the implementation can be done, one last key question must be answered: How to handle the missing data points in this context?

Since all the computations are point wise the whole algorithm can be seen as point wise and therefore allows to ignore missing values all together. If in a given step a pixel is missing it simply gets skipped in the process and ignored in the future steps. This approach is sound following the arguments in chapter D.2.1.

References

[1] Ben Adler. Hilbert spaces and the Riesz representation theorem. *The University of Chicago Mathematics REU*, 2021.

[2] Christophe Avenel, Etienne Mémin, and Patrick Pérez. Stochastic level set dynamics to track closed curves through image data. *Journal of mathematical imaging and vision*, 49:296–316, 2014.

[3] N. Chopin. particles. https://github.com/nchopin/particles, 2022.
Commit: 20c2730ed6ab8f68fcc700390dbf8db739f34d81.

[4] Nicolas Chopin, Omiros Papaspiliopoulos, et al. *An introduction to sequential Monte Carlo*, volume 4. Springer, 2020.

[5] Hai-Dang Dau and Nicolas Chopin. On backward smoothing algorithms, 2023.

[6] D.S. Ebert. *Texturing and Modeling: A Procedural Approach*. The Morgan Kaufmann Series in Computer Graphics. Elsevier Science, 2014.

[7] Wilfried Enkelmann. Investigations of multigrid algorithms for the estimation of optical flow fields in image sequences. *Computer Vision, Graphics, and Image Processing*, 43(2):150–177, 1988.

[8] John C. "Craig" George, Henry P. Huntington, Karen Brewster, Hajo Eicken, David W. Norton, and Richard Glenn. Observations on shorefast ice dynamics in Arctic Alaska and the responses of the Iñupiat hunting community. *Arctic*, 57(4):363–374, 2004.

[9] Saeed Ghadimi and Guanghui Lan. Stochastic first- and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 23(4):2341–2368, 2013.

[10] C. A. Greene, D. A. Young, D. E. Gwyther, B. K. Galton-Fenzi, and D. D. Blankenship. Seasonal dynamics of Totten Ice Shelf controlled by sea ice buttressing. *The Cryosphere*, 12(9):2869–2882, 2018.

[11] Nathaniel Johnston and Dave Greene. *Conway's Game of Life, Mathematics and Construction*. Self-published, 2022.

[12] Olav Kallenberg. *Foundations of modern probability*, volume 2. Springer, 1997.

[13] Shinji Kataoka. A stochastic programming model. *Econometrica*, 31(1/2):181–196, 1963.

[14] Jean-François Lemieux, Ji Lei, Frédéric Dupont, François Roy, Martin Losch, Camille Lique, and Frédéric Laliberté. The impact of tides on simulated landfast ice in a pan-Arctic ice-ocean model. *Journal of Geophysical Research: Oceans*, 123(11):7747–7762, 2018.

[15] J.P. Lewis. Fast normalized cross-correlation. *Ind. Light Magic*, 10, 10 2001.

[16] Yuqing Liu, Martin Losch, Nils Hutter, and Longjiang Mu. A new parameterization of coastal drag to simulate landfast ice in deep marginal seas in the Arctic. *Journal of Geophysical Research: Oceans*, 127(6):e2022JC018413, 2022. e2022JC018413 2022JC018413.

[17] Andy Mahoney, Hajo Eicken, Allison Graves Gaylord, and Lewis Shapiro. Alaska landfast sea ice: Links with bathymetry and atmospheric circulation. *Journal of Geophysical Research: Oceans*, 112(C2), 2007.

[18] R. A. Massom, K. L. Hill, V. I. Lytle, A. P. Worby, M. J. Paget, and I. Allison. Effects of regional fast-ice and iceberg distributions on the behaviour of the Mertz Glacier polynya, East Antarctica. *Ann. Glaciol.*, 33:391–398, 2001.

[19] Nicolas Papadakis and Etienne Mémin. A variational technique for time consistent tracking of curves and motion. *Journal of Mathematical Imaging and Vision*, 31:81–103, 2008.

[20] Ken Perlin. An image synthesizer. *SIGGRAPH Comput. Graph.*, 19(3), jul 1985.

[21] Ken Perlin. Improving noise. *ACM Trans. Graph.*, 21(3):681–682, jul 2002.

[22] J Röhrs and L Kaleschke. An algorithm to detect sea ice leads by using AMSR-E passive microwave imagery. *The Cryosphere*, 6(2):343–352, 2012.

[23] Andrzej Ruszczyński and Alexander Shapiro. Stochastic programming models. In *Stochastic Programming*, volume 10 of *Handbooks in Operations Research and Management Science*, pages 1–64. Elsevier, 2003.

[24] Valeria Selyuzhenok and Denis Demchev. An application of sea ice tracking algorithm for fast ice and stamukhas detection in the Arctic. *Remote Sensing*, 13(18), 2021.

[25] D. Siantidis. Bachelor_2023. https://github.com/P3ngwings/Bachelor_2023, 2023.
Commit: 50a819a6acc86bb7e2c15e4234069e0480839dc3.

[26] Thomas G. Smith and Michael O. Hammill. Ecology of the ringed seal, phoca hispida, in its fast ice breeding habitat. *Canadian Journal of Zoology*, 59(6):966–981, 1981.

[27] G. Spreen, L. Kaleschke, and G. Heygster. Sea ice remote sensing using AMSR-E 89-GHz channels. *Journal of Geophysical Research: Oceans*, 113(C2), 2008.

[28] Takeshi Tamura, Kay I. Ohshima, Thorsten Markus, Donald J. Cavalieri, Sohey Nihashi, and Naohiko Hirasawa. Estimation of thin ice thickness and detection of fast ice from SSM/I data in the Antarctic Ocean. *Journal of Atmospheric and Oceanic Technology*, 24(10):1757 – 1772, 2007.

[29] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua

Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental algorithms for scientific computing in python. *Nature Methods*, 17:261–272, 2020.

Statement of Authorship

I hereby declare that I am the sole author of this bachelor's thesis and that I have not used any sources other than those listed in the bibliography and identified as references. I further declare that I have not submitted this thesis at any other institution in order to obtain a degree.

[illegible]

Bremen, 26 July 2023

John David Siantidis