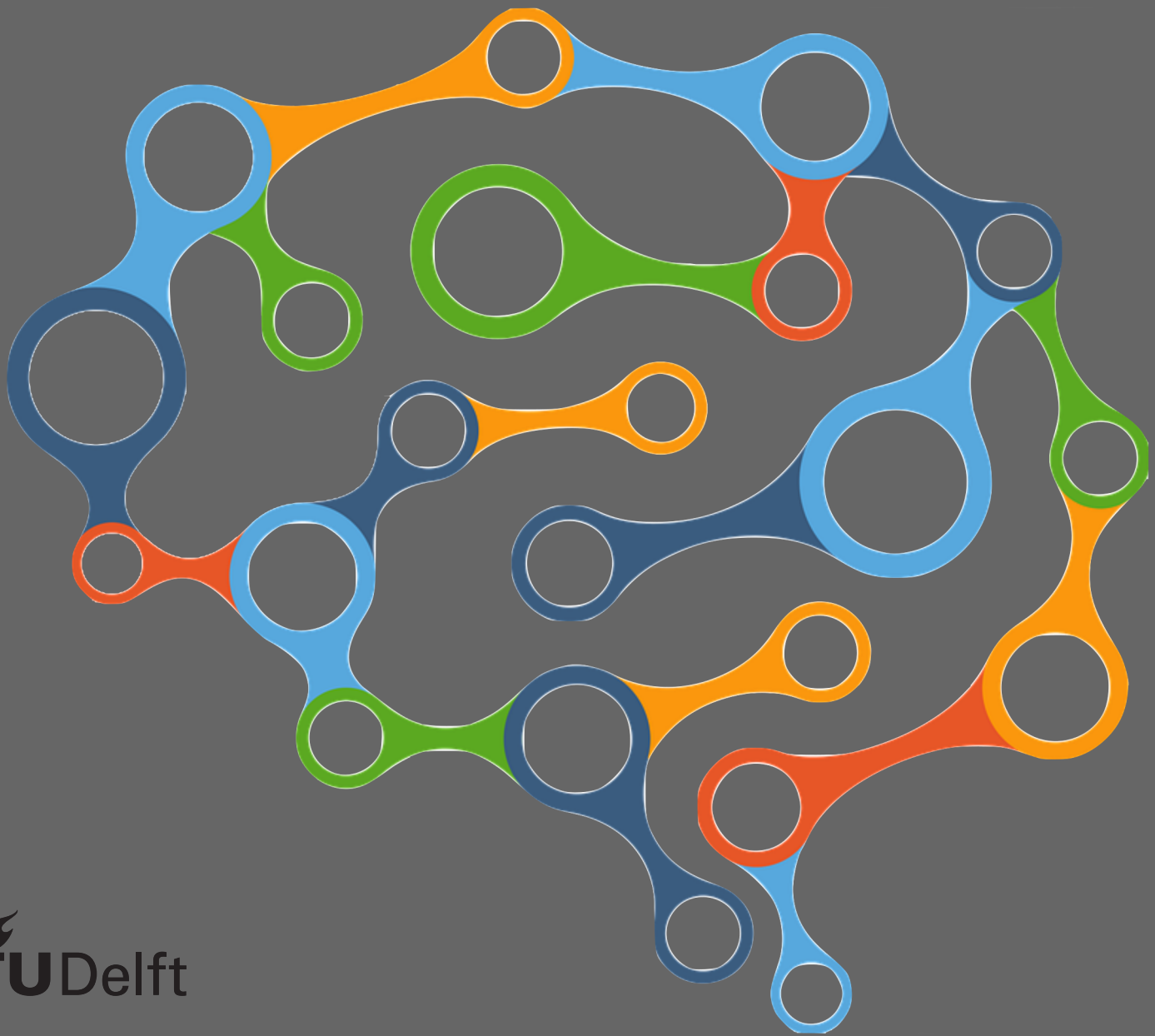
Generalised Motions in Active Inference by finite differences

Active Inference in Robotics

I.L. Hijne

Technische Universiteit Delft



TU Delft

Generalised Motions in Active Inference by finite differences

Active Inference in Robotics

by

I.L. Hijne

to obtain the degree of Master of Science
in Mechanical Engineering (BioMechanical Design),
with the specialization BioRobotics,
at the Delft University of Technology,
to be defended on Thursday 13 August 2020 at 13:00h.

Student number:	4079183	
Project duration:	September 2019 – August 2020	
Thesis committee:	Prof. dr. ir. Martijn Wisse,	Cognitive Robotics, TU Delft, supervisor
	Prof. dr. Robert Babuska,	Cognitive Robotics, TU Delft
	ir. Corrado Pezzato,	Cognitive Robotics, TU Delft

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Preface

For this thesis I have been introduced into the intriguing theory of Active Inference. I could never have imagined working on a neuroscientific theory about perception, action and learning in biological brains that is applicable for the control of robotic systems. The concepts of Active Inference and the Free Energy Principle are truly fascinating, and it perfectly fits the specialization of 'BioRobotics', because, if modelling robot control based on the workings of the biological or human brain is not a BioRobotics topic, then what is? Diving into this theory one quickly learns that the theory is esoteric and vast. It is predominantly (co)written by leading neuroscientist Karl Friston who has published numerous papers on the topic that each are a tough pill to swallow. Many researchers who take on the theory will agree that the work by Karl Friston is very complicated and hard to understand. When starting research on the topic, the literature is intimidating. Yet, it also sparks curiosity for the theory is also so elegant and interesting, as it provides a biologically plausible yet mathematically well-defined unification of action and perception that, in the end, boils down to a fairly simple gradient descent scheme on a quadratic error function. Meetings and conversations with fellow researches, all tackling (slightly) different aspects of this theory, and with my supervisor Martijn Wisse have proven very helpful in tackling this topic and, in hindsight, I'm pleased to have learned so much about it. It has definitely grown on me. As is the case with most research, it is never finished. As my research progressed, my understanding of the topic improved and the current literature a little less hard to read. New ideas for improvement of the work arise and the same literature seems to reveal more and more insights to me each time I browse through. Eventually though, the time to conclude this research has come. Ideas for future research have been included and I hope this work may provide a useful start for a future researcher that wishes to contribute to the application of Active Inference for robotics control. If enough people do so, I believe that, in the future, Active Inference will prove to be a very powerful theory for very natural high performance control of robotics in different environments.

I.L. Hijne
Delft, August 2020

Abstract

This thesis is a contribution to the research on Active Inference for Robotics. Active Inference is an intricate, intriguing theory from neuroscience, a field in which it has already gained a greater following and popularity. This theory, based on the underlying Free Energy Principle, provides a unified account of perception, action and learning in the biological brain. It has great explanatory power of the function of the biological brain and furthermore it is mathematically well-defined. This property makes the theory suitable for a translation to robotics, in which it can also provide a unified account of action and perception. This is not only elegant, but potentially very powerful too. The research for Active Inference in robotics is young, but the current research already shows that Active Inference indeed has great potential for robotics control.

Literature on Active Inference is narrow and complex, and provides a lot of concepts to work with in a translation to robotics control. Once such concept are the generalised coordinates of motion, which are the instantaneous derivatives of a dynamic variable. The incorporation of generalised coordinates, especially in combination with the assumption that the noise encountered in a dynamic environment is coloured, has great potential to be beneficial for both action and perception when it comes to robot control in real environments. Generalised coordinates provide a reference frame for the gradient descent that is applied to provide the action and perception laws, which in a dynamic setting has to 'hit a moving target'. Furthermore, in combination with coloured noise the generalised coordinates are advantageous for dealing with such noise.

In this thesis, detailed research is provided with regards to the application of generalised coordinates in Active Inference for robotics. Current research for robotics in which Active Inference has been applied doesn't exploit the full potential of generalised coordinates. Therefore, this research aims to explore the constructs necessary to apply generalised coordinates of motion in an on-line Active Inference control loop of an LTI State Space system. A detailed derivation of the generalised precision, which relates generalised coordinates and coloured noise, is provided. A method for obtaining generalised output by means of finite differences is proposed, that constructs generalised coordinates from the on-line data in scenarios in which the environment does not provide the required generalised coordinates naturally. The method is implemented in the simulation of a one degree of freedom SISO LTI State Space scenario which highlights the potential but also the difficulties still faced when applying Active Inference for on-line robotics control. Besides the detailed derivations of some aspects of Active Inference for robotics, open problems are identified and suggested for future research that can potentially yield methods to apply Active Inference in robotics at full capacity, providing a true biologically plausible robot control method.

Contents

List of Figures

List of Tables

List of Listings

1 Introduction

This first chapter of this work is an introductory chapter, meant to introduce the topic of Active Inference in robotics and the aim of this research. The research field of control for robotics is rich and robotic solutions exist in many ways, shapes or forms throughout our lives, to improve the quality thereof, for applications ranging from comfort or entertainment-enhancing solutions to the replacement of humans in hazardous environments or for tedious tasks. In contrast to robotics, biological organisms are highly adaptive. To achieve such adaptivity and performance in uncertain environments in robotics, it seems logical to look to nature for inspiration, to mimic the there-found behaviour, intelligence and/or performance. On the rise is the theory of Active Inference, based on the underlying Free Energy Principle, of neuroscientist Karl Friston. It is a unified theory of action, perception and learning in the biological brain and has more recently received some attention in the world of robotics. This chapter explains how a neuroscientific theory is of interest to the world of robotics, what challenges are faced in the research towards robotics control by means of active inference and how those will be addressed in this thesis, by introducing a research question and a set of sub-questions, thereby also presenting an outline of this work.

1.1. Active Inference

Active Inference is a neuroscientific theory, or principle, by Karl Friston. It is a well-known theory within neuroscience and is picking up attention in the research field of robotics. This theory provides a unified account of perception, action and learning under the Free Energy Principle [11, 12]. A unified theory of the workings of the biological brain would be a holy grail in neuroscience, but could also prove very useful for robotics. The Free Energy principle states that all self-organizing biological organisms, or agents, minimize a quantity called free energy, which can be done by means of a gradient descent scheme, to bound the entropy over a life-span in order to survive. By performing this minimization a brain, or agent, acts as a Bayesian Inference machine. The Free Energy Principle and Active Inference are not only biologically plausible, but also mathematically well-defined, which makes research for an application to robotics all the more interesting. The scientific content on Active Inference is esoteric and vast, for which the theory is also criticized. Furthermore, the Free Energy Principle is not falsifiable. It is, as confirmed by Friston, merely a principle [19]. This principle of minimization of Free Energy, or in other words, the reduction of uncertainty, is promising for robotics control for the aforementioned reasons, and also intrinsically accounts for uncertainty. It could prove to be a very complete and elegant method for robotics control if it indeed unifies action and perception (and learning), whilst dealing with uncertainty.

1.1.1. Applications for robotics

Research regarding the application of Active Inference for the control of robotics is still young and narrow, although a few applications exist. It has already been shown that Dynamic Expectation Maximization (DEM, [18]), a theory also based on the Free Energy Principle and also by Karl Friston about the inference of a systems states, parameters and hyperparameters, can outperform a classical Kalman

Filter in the presence of coloured noise [1] (for coloured noise see chapter 3). Applications of Active Inference in robotics exist in the form of simulations and for real robots with on-line control. In many applications of the Free Energy Principle, the focus is on perception. A more action-driven approach is of interest in applications for robotics. Applications based on simulations can be found in the work of [29], which describes simulations of the control of a 7-DoF arm of a PR2 robot with visual input for a reaching task based on Active Inference. In this simulation, the true configuration of the 7-DoF arm is assumed to be known and action (control input) is the result of reflex arcs. A true closed feedback loop is thus missing. Closed loop implementations are provided in [2] for a simulation of a simple wheeled robot that performs phototaxis (it orients itself towards light), and in [23] for simulations of a robot with a 2-DoF arm and a monocular camera for visual sensory input, performing a tracking task. Control of real, on-line robots by means of Active Inference has meanwhile been shown by [26] and [27, 28]. The former shows the control of the 3-DoF arm and head movement of the humanoid iCub robot, which is velocity controlled and has a model defining the relationship between sensory input from the visual field to the joint space of the arm. The latter shows on-line control of a 7-DoF Franka Emika Panda robot arm performing torque-controlled tracking tasks for multiple set points, simulation a pick-and-place task. Furthermore, the performance of this Active Inference control is compared to a state-of-the-art MRAC controller, showing the potential of Active Inference as a high-performance robot control method.

1.1.2. Generalised coordinates of motion

Active Inference is an interesting and promising theory for robot control for various reasons, among which is the aforementioned unification of action and perception, but also the natural ability to deal with uncertainty and the potential to achieve more ‘natural’ behaviour in robotics due to the biological plausible, nature-inspired background of the method. Another key feature of Active Inference is the existence and application of the so-called ‘generalised coordinates of motion’, or ‘generalised motions’ or ‘generalised coordinates’ [10]. These generalised motions are the instantaneous temporal derivatives of a dynamic process and including generalised motions can provide more information on a dynamically correlated process. This is especially interesting when coloured noises are assumed. In contrast to white noise often assumed in classical algorithms, coloured noise is smooth and therefore its temporal derivatives exist. If one assumes all disturbances to a process to be the result of a dynamical process themselves, this assumption is valid. The generalised coordinates can potentially be very beneficial for dealing with uncertainty, albeit unmodelled dynamics or a disturbance from the environment, given the dynamical correlation that exists in the coloured noise.

Generalised motions are applied in some of the existing applications for robotics. One additional dynamical order is present in the work of [29]. This occurs naturally in the equations of motion. For the wheeled robot of [2], generalised motions are not considered at all. In [23] the authors discuss the use of generalised motions up to the third order, but drop the third order for their simulations, considering it to be only noise. The work of [26] mentions generalised motions up to the second order, but applies only those of the first order. Lastly, in the work of [27, 28] the motions up to second order are taken into account in the internal model. All these works in which generalised motions are applied have an assumption in common that the dynamics of different orders of motion are uncorrelated. This also means that it is assumed that there is no correlation of the coloured noise on the different dynamical orders. In the mathematical review of [4] this assumption is also made and the mathematics are well-explained. This has a great impact on the form of the Free Energy and the generalised precision matrix it is parametrized by, as shown in chapters 2 and 3. Furthermore, the generalised motions have only been considered up to an order that is available from the system description or equations of motion. A State Space formulation for Active Inference has been proposed in [20]. Generalised motions are taken into account for this State Space formulation, including correlation between dynamical orders and the resulting form of the Free Energy equation. However, this has only been done for the process dynamics, and also for dynamical orders up to the second only. Generalised motions were not considered for the sensory input, or what is considered the system output in classical control.

1.2. Research objective

The objective of this thesis is to explore the application of Active Inference to robotics, taking into account promising features of Active Inference that currently have not been fully explored. Generalised coordinates of motion are an interesting concept when the noise present in a system is no longer

assumed white but coloured, in which case the generalised coordinates of motion provide additional information about the dynamics of a system and its noise. It is this correlation and potential gain for performance in robotics control that needs to be explored. With it, however, come some difficulties that are still to be solved. In this thesis, the necessities to apply generalised coordinates to their full extent are researched. This means using generalised coordinates up to a desired order without the necessity to be able to measure them or obtain them through system relations. The starting point for this research is a suggestion made by Friston in [10] to create generalised coordinates by means of finite differences (Taylor expansion). The research question for this work is:

What constructs are necessary to apply generalised coordinates to an Active Inference control loop whilst taking correlation between dynamical orders into account?

In order to answer this question, several sub-problems need solving. Current research doesn't show the necessary tools to create an Active Inference control loop that fully supports generalised coordinates and the use of correlation between the dynamical orders.

1.2.1. Research directions

To explore the application of Active Inference to a control loop for robotics, whilst including the promising aspects of Active Inference mentioned, the following will be researched in this thesis. Sub-questions to the above research question are posed, and with it, the structure of the document is explained:

a) *What is the role of generalised coordinates in Active Inference?*
In order to understand the generalised coordinates, it is important to understand the theory of active inference and the underlying free energy principle. This is the content of chapter 2, a chapter of background theory. In this chapter it is shown how a theory from neuroscience, of a brain as a Bayesian Inference machine and of the minimisation of surprisal at all times leads to a mathematical construct that can be applied to robotics control. Assumptions about coloured noise as an influence on dynamical processes and their relationship to generalised coordinates are treated in this chapter and aimed at answering this first sub-question. The position of this research with regards to current literature is also addressed.

b) *What is the relationship between generalised coordinates and coloured noise?*
Generalised coordinates are introduced and described in chapter 2, as well as the role of noise in Active Inference. The coloured noise influences the Active Inference algorithm by means of a generalised precision matrix, which is first introduced in chapter 2. The generalised precision matrix is the major topic of chapter 3, in which a detailed derivation of the matrix is given. It gives insight into the relationship between generalised coordinates and coloured noise, and is required for implementation of generalised coordinates in a robotics control loop.

c) *How can generalised coordinates be generally applied in the Active Inference framework when they are not readily available?*
In Active Inference, generalised coordinates are assumed to exist. In reality, this is true, since each dynamical process has an infinite amount of temporal derivatives. However, in a robotics scenario in which a process is known by the input it is provided and the output that is measured, temporal derivatives (generalised coordinates) are not generally available. A velocity and acceleration measurement of a position state might be available. Even then, only additional orders of motion are available. It can be said that generally, generalised coordinates cannot be measured. In [10] it is suggested that generalised coordinates can be created by means of finite differences (Taylor expansion). This is the topic of chapter 4.

d) *How can Active Inference with generalised motions be applied to an LTI-State Space control loop?*
Active Inference has been applied to control loops in previous research. As mentioned before, it has even been successfully applied to a physical robot [26, 28]. In these scenarios, however, the correlation between dynamical orders that was mentioned earlier is missing. The work in [20] shows a State Space implementation of Active Inference, but with limited generalised coordinates and no generalised output. For the desired practical implementation that this research is focused on, it is important to have a functional State Space formulation of Active Inference with generalised

coordinates and generalised precision. The topic of chapter 5 is the formulation of a simple one-DOF LTI-State Space system that implements generalised coordinates and generalised precision as described in preceding chapters. Simulation results will show the effects of the implemented constructs as well as the challenges it presents.

Finally, chapter 6 is a conclusive chapter. It is aimed at summarising the research and answering the research questions, but also at discussing and explaining directions for future research, because Active Inference in robotics is still young and requires far more research than is presented in this thesis.

1.3. Notation

The math involved in the work of this thesis is written as consistent and clear to a reader from the field of robotics as possible, although a lot of the literature on the topic is from a neuroscientific or other background and notations vary greatly. For clarity, some notation conventions are declared:

- This work assumes dynamical processes. Dynamic variables are therefore time-dependent, but this dependency is generally omitted for brevity and readability. E.g., a process $x(t)$ is denoted simply by x, similarly for others such as $y(t)$, $w(t)$, $z(t)$, $\mu(t)$.
- All vectors are denoted in boldface: $\boldsymbol{x}$, $\boldsymbol{y}$ etc. (also without time-dependency (t)), and matrices are always denoted by a capital letter such as A, B, Σ, Π.
- A semicolon is followed by parameters. So $f(\boldsymbol{x};\mu,\Sigma)$ is a function of $\boldsymbol{x}$, parametrized by the variables μ and Σ.
- Partial derivatives are always written as a fraction: The partial derivative of $f(\boldsymbol{x},\boldsymbol{y})$ is denoted by $\frac{\partial f}{\partial \boldsymbol{x}}$, the second partial derivative by $\frac{\partial^2 f}{\partial \boldsymbol{x}^2}$, etc.
- Temporal derivatives may be represented by a dot (e.g. $\dot{\boldsymbol{x}}$) or by $\frac{\mathrm{d}}{\mathrm{d}t}$, or more generally by a superscript in parenthesis $\boldsymbol{x}^{(d)}$ with $d = 0, 1, 2, ...$
- A tilde signifies generalised coordinates (introduced in section 2.3). Generalised coordinates of motion are instantaneous (beliefs of) temporal derivatives, such as $\tilde{\boldsymbol{\mu}}$, $\tilde{\boldsymbol{y}}$, which are vectors stacked with temporal derivatives. Individually these are represented by dashes, e.g. μ', μ'', but they may also be represented by a superscript in parentheses. Tildes also represent generalised versions of functions or variables related to generalised coordinates, such as $\tilde{A}$ or $\tilde{\Pi}$.
- The Kronecker product of two matrices is represented by $\otimes$ and multiplies each element of the left argument with the entire right argument.

2

Free Energy principle

This chapter is one of background knowledge. Active Inference is a very rich theory, originally from the field of Neuroscience. The neuroscientific theory is esoteric and vast, providing an explanation of life on timescales ranging from evolutionary to functional (the timescale of a dynamical process). The neuroscientific theory is outside the scope of this work. Fortunately, the theory of Active Inference and the underlying Free Energy Principle are mathematically well-defined and therefore suited for a translation to robotics. Because of the vastness of the theory, the underlying mathematics are also very extensive. In this chapter, the mathematics of the Free Energy Principle that are required for understanding the further research in this work are explained in a notation that is as familiar as possible for the field of control theory. The mathematics are accompanied by some explanation of the neuroscientific theory, to put it into context. By the end of this chapter, an answer to the research question 'What is the role of generalised coordinates in Active Inference' can be formulated.

2.1. A theory of biological adaptive systems

The Free Energy principle is the underlying theory of Active Inference. It is an information theoretic quantity, introduced into physical statistics [8] as a construct that converts a difficult or intractable integration of a probability density into an optimization problem. In Neuroscience, it provides a unified account of action, perception and learning [12] for adaptive systems (biological agents like animals or brains). Free Energy is an upper bound on surprisal, related to homeostasis, as explained in section 2.1.1, but also the difference between an agents internal model of environmental states (a recognition density $q(\boldsymbol{x})$) and its model of states and sensory input from the outside world (a generative density $p(\boldsymbol{x}, \boldsymbol{y})$), when a biological brain or agent is considered a Bayesian Inference machine. This is explained in section 2.1.2. In this chapter, the terms 'brain' and 'agent' are used interchangeably, because the origins of the theory lie with biological brains, but when translating the theory to robotics, it makes more sense to talk about an agent. The relation between the two, as both terms are used, becomes clear in this chapter. In the following chapters, the term 'agent' will solely be used. In this chapter, the focus is on the mathematical derivation towards application in robotics, although with connections to the neurological origin of the theory. In [12, 16], among others, more neurological-focused descriptions of the Free Energy Principle and Active Inference can be found.

2.1.1. Homeostasis

As Friston explains [12, 13]: Any biological system that maintains its homeostasis, which is necessary in order to survive, must resist a natural tendency to disorder. From the point of view of a brain, or an agent, both the body (outside the brain) and the surrounding world are part of the environment. Any sensory input to the brain is an output of the environment and in order to maintain homeostasis, the mathematical probability of the sensory states that an adaptive biological system can be in, must have low entropy. The probability of the states of a system in homeostasis must be high, and the probability of remaining states low. Entropy is the long-term average of surprisal, which is the negative log-evidence of probability and a construct from information theory [30]. The relationship between sur-

prise and the probability of a sensory input of the agent is displayed in figure 2.1. By defining surprisal as a logarithmic function, the surprisal of multiple events becomes additive. Figure 2.1 illustrates that a negative log-probability measure is an intuitive representation of surprisal: it is zero for a very typical event that occurs with probability one, and approaches infinity for the low probability of an unlikely (atypical) event.

An intuitive example of surprisal is that of a fish out of water. Out of water, a fish cannot maintain

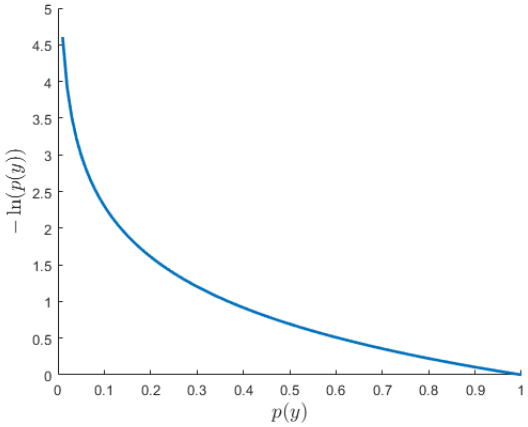


Figure 2.1: Surprisal as a log-probability measure.

homeostasis and in these conditions, the surprisal of that sensory state will be high. Since the fish will spend (nearly) all of its life in water, the probability associated with the state 'out of water' will be low (and thus have high surprisal, if it does occur). Furthermore, entropy is high for a fish that more often finds itself out of water.

Surprisal can, unfortunately, not be minimised directly as it would require an agent to know the distribution of sensory inputs $p(\boldsymbol{y})$. Knowing this distribution means an agent (or brain) knows the distribution of 'surprising' events, which intuitively is highly unlikely, if not impossible. This is where Free Energy comes into play: as it turns out, Free Energy is an upper bound on surprisal. In contrast to surprisal itself, Free Energy can be evaluated because it depends on the agents sensory input ($\boldsymbol{y}$), which is available, and the so-called recognition density: an internal model of (hidden) states in the world $q(\boldsymbol{x})$. By minimising Free Energy, surprisal can be minimised indirectly. This will be explained in section 2.1.2.

2.1.2. Bayesian Inference

Research has shown that biological brains are likely to represent information by means of probability distributions [21]. Both sensory and motor signals contain a degree of uncertainty, and this must be accounted for. It appears that computations in the brain based on perception are 'Bayes optimal'. This means that the brain can be considered a Bayesian Inference machine. What follows is a description of the probability densities that are involved when an agent performs Bayesian Inference based on its perception and internal beliefs, and how this is related to Free Energy. This has been well-explained in [4].

Consider an agent that maintains a model of its environment. This internal model represents the beliefs of the agent about its environment, about the (hidden) states that are the causes of its sensory input. The states are 'hidden' because they cannot be perceived directly, but only by means of the sensory input of the agent (i.e. the 'output' of the environment). For example, it is not possible to perceive the velocity of a passing car directly, but only by means of our visual input. A mapping is required to translate the visual input to (an estimate of) the velocity of the passing car. The model is represented by a probability distribution over all the values that these variables could have. Such a model needs to be maintained: it requires updating when evidence of the environment becomes available through sensory input. By means of a Bayesian Inference process, the agent can update this probability density that is known as the 'Recognition Density' or 'R-density' and effectively model its environment. It is represented in equation (2.1). In order to update the R-density using the sensory input

that is available to the agent, it needs to know how the states translate to their sensory inputs. Therefore, another model is required that is made up of prior beliefs about the (hidden) states of the world. The model is represented by a joint probability density that is also known as the 'generative density' or 'G-density'. This G-density is shown in equation (2.1) and can be factorised into two probability densities:

- The prior density, which encodes the agent's beliefs about the world without any additional knowledge from sensory input.
- The likelihood, a conditional density that encodes the agents belief of relationship between environment states and its sensory input.

$$\underbrace{q(\boldsymbol{x})}_{\text{R-density}} \qquad \underbrace{p(\boldsymbol{x},\boldsymbol{y})}_{\text{G-density}} = \underbrace{p(\boldsymbol{y}|\boldsymbol{x})}_{\text{likelihood}}\,\underbrace{p(\boldsymbol{x})}_{\text{prior}} \tag{2.1}$$

It is assumed all the probability densities are normalised:

$$\int q(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} = \iint p(\boldsymbol{x},\boldsymbol{y})\,\mathrm{d}\boldsymbol{x}\,\mathrm{d}\boldsymbol{y} = \int p(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} = \int p(\boldsymbol{y})\,\mathrm{d}\boldsymbol{y} = 1$$

The density of interest to an agent that aims to perceive the states in its environment is the the posterior density $p(\boldsymbol{x}|\boldsymbol{y})$ which, when evaluated, gives the agent the most information possible about a state $\boldsymbol{x}$ in the environment given its sensory input. Applying Bayes theorem, the posterior belief of $\boldsymbol{x}$ given the sensory input $\boldsymbol{y}$ taking on some value $\boldsymbol{\phi}$ can be computed as in equation (2.2).

$$p(\boldsymbol{x}|\boldsymbol{\phi}) = \frac{p(\boldsymbol{\phi}|\boldsymbol{x})p(\boldsymbol{x})}{p(\boldsymbol{y}=\boldsymbol{\phi})} = \frac{p(\boldsymbol{\phi}|\boldsymbol{x})p(\boldsymbol{x})}{\int p(\boldsymbol{\phi}|\boldsymbol{x})p(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}} \tag{2.2}$$

The denominator of equation (2.2) needs to be evaluated if an agent is to evaluate the posterior belief. However, this integral is often intractable, as it requires the evaluation over all possible sensory states, which requires knowing them all. Intuitively, this seems quite impossible. Since the posterior density cannot be evaluated directly, it requires an approximation which can be done by means of 'variational Bayes' (also called 'ensemble learning' or 'approximate Bayesian inference') [3, 18]. The method of variational Bayes translates the problem of integral evaluation to one of optimisation, a problem which is much easier to solve. It is the Free Energy that can be minimised by means of an optimisation process, and it follows from the Kullback-Leibler Divergence [22] between the R-density and the posterior density (equation (2.2)). The KL-divergence is a measure of difference between two probability densities, and it is defined in equation (2.3). A KL-divergence between the R- and posterior densities that is minimised with respect to the R-density results in an R-density that is a better approximation to the posterior, making it a better model of the environment. The KL-divergence cannot be evaluated, however, since this still requires the evaluation of the posterior density. Rewriting it results in equation (2.4).

$$\begin{aligned} D_{KL}(q(\boldsymbol{x}) \parallel p(\boldsymbol{x}|\boldsymbol{y})) &= \int q(\boldsymbol{x}) \ln\!\left(\frac{q(\boldsymbol{x})}{p(\boldsymbol{x}|\boldsymbol{y})}\right)\mathrm{d}\boldsymbol{x} \qquad (2.3)\\ &= \int q(\boldsymbol{x}) \ln\!\left(\frac{q(\boldsymbol{x})p(\boldsymbol{y})}{p(\boldsymbol{x},\boldsymbol{y})}\right)\mathrm{d}\boldsymbol{x}\\ &= \int q(\boldsymbol{x}) \ln(q(\boldsymbol{x}))\,\mathrm{d}\boldsymbol{x} - \int q(\boldsymbol{x}) \ln(p(\boldsymbol{x},\boldsymbol{y}))\,\mathrm{d}\boldsymbol{x} + \underbrace{\int q(\boldsymbol{x}) \ln(p(\boldsymbol{y}))\,\mathrm{d}\boldsymbol{x}}_{\int q(\boldsymbol{x})\mathrm{d}\boldsymbol{x}=1}\\ &= \underbrace{\int q(\boldsymbol{x}) \ln(q(\boldsymbol{x}))\,\mathrm{d}\boldsymbol{x} - \int q(\boldsymbol{x}) \ln(p(\boldsymbol{x},\boldsymbol{y}))\,\mathrm{d}\boldsymbol{x}}_{\equiv F} + \underbrace{\ln(p(\boldsymbol{y}))}_{neg.surprisal} \qquad (2.4) \end{aligned}$$

From equation (2.4) a definition of the (variational) Free Energy arises. This definition of Free Energy (equation (2.5)) depends on the R- and G-densities and not on the Bayesian posterior. Both the R- and G-densities can, in contrast to the posterior density, be defined and evaluated, as will be shown in sections 2.2.1 and 2.2.2.

$$F \equiv \int q(\boldsymbol{x}) \ln(q(\boldsymbol{x}))\,\mathrm{d}\boldsymbol{x} - \int q(\boldsymbol{x}) \ln(p(\boldsymbol{x},\boldsymbol{y}))\,\mathrm{d}\boldsymbol{x}$$

$$= \int q(\boldsymbol{x}) \ln\left(\frac{q(\boldsymbol{x})}{p(\boldsymbol{x},\boldsymbol{y})}\right) \mathrm{d}\boldsymbol{x} \tag{2.5}$$

Because of Jensen's inequality [5], the KL-divergence is known to be non-negative. With the KL-divergence depending on the Free Energy and on surprisal (equation (2.4) and section 2.1.1), it can be stated that the (variational) Free Energy (equation (2.5) is an upper bound on surprisal. Minimising the Free Energy therefore indirectly minimises surprisal (section 2.1.1) and minimises the KL-divergence, meaning that minimising the Free Energy also makes the R-density a better approximation to the Bayesian posterior:

$$\left.\begin{aligned} D_{KL} &\geq 0 \\ F &= D_{KL} - \ln(p(\boldsymbol{y})) \end{aligned}\right\} \quad F \geq \underbrace{-\ln(p(\boldsymbol{y}))}_{\text{surprisal}}$$

So far, it has been shown that with a theory of the requirement of any self-organizing organism to maintain its homeostasis and with the Bayesian brain hypothesis, variational Bayesian Inference yields a formulation of Free Energy that can be evaluated, in contrast to the original inference problem and, when minimized, can indirectly minimise surprisal and make the R-density a better approximation to the Bayesian posterior, which allows an agent to infer the states of the environment through its sensory input. The Free Energy is currently specified as a function of undefined probability densities. A practical implementation of the Free Energy will be presented in the following section.

2.2. Explicit Free Energy

The Free energy depends on the R- and G-densities ($q(\boldsymbol{x})$, $p(\boldsymbol{x},\boldsymbol{y})$). A practical formulation of the Free Energy requires an explicit definition of these probability density functions. Uncertainty, however, makes an explicit definition impossible. Uncertainties can, for example, be the result of noise or unmodelled dynamics. To arrive at a practical formulation of Free Energy, some assumptions need to be made, which will be described in this section. This process involves:

1. The Laplace approximation to give form to the R-density by means of its sufficient statistics, resulting in the Laplace-encoded Free Energy.
2. Choosing a functional form for the G-density as a product of the conditional and prior densities.

These are the topics of sections 2.2.1 and 2.2.2 respectively.

2.2.1. The Laplace approximation

The R-density $q(\boldsymbol{x})$ is an internal model of hidden states in the environment. It is the probability density that represents an agent's internal belief. Because this density is an internal model, it can be specified 'freely'. It is common to make the Laplace approximation [18, 25], which provides an explicit definition of the R-density by assuming it to be a Gaussian distribution (equation (2.6)) that is defined by its sufficient statistics, the mean and variance. Furthermore, the (co)variance is assumed to be small. This means that the distribution is sharply peaked around its mean value, which is the case if the states weakly covary (independent states), and the better an agent knows its environment, the more accurate this assumption is. This assumption is most valid when the agent is most certain about the R-density (its internal model). This scenario is not unrealistic, but somewhat specific. The agent may have learned about this environment before. For the derivation of the Laplace approximation, a single state x will be considered. An analogous derivation can be made for the multivariate set of states $\boldsymbol{x}$.

$$q(x) \equiv \mathcal{N}\left(x;\mu,\sigma^2\right) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) = z^{-1} \exp\left(-\epsilon(x)\right) \tag{2.6}$$

$$\text{with} \quad \begin{aligned} z &\equiv \sqrt{2\pi\sigma^2} \\ \epsilon(x) &\equiv \tfrac{1}{2\sigma^2}(x-\mu)^2 \end{aligned}$$

Given the definition of the R-density in equation (2.6), together with the assumptions of the Laplace approximation, the integral in the Free Energy in equation (2.5) becomes tractable. This is done in two steps:

- Substituting the definition of the R-density in equation (2.6) into the Free Energy in equation (2.5).

- Apply a Taylor Expansion about the Gaussian that is assumed to be sharply peaked.

For the substitution of equation (2.6) into equation (2.5), the substitute variables z and $\epsilon(x)$ from equation (2.6) are very useful:

$$\begin{aligned} F &= \int q(x) \ln\left(\frac{q(x)}{p(x,y)}\right) \mathrm{d}x \\ &= \int q(x) \ln(q(x)) \, \mathrm{d}x - \int q(x) \underbrace{\ln(p(x,y))}_{-E(x,y)} \mathrm{d}x \\ &= \int q(x)(-\ln(z) - \epsilon(x)) \, \mathrm{d}x + \int q(x) E(x,y) \, \mathrm{d}x \\ &= -\ln(z) - \int q(x)\epsilon(x) \, \mathrm{d}x + \int q(x) E(x,y) \, \mathrm{d}x \end{aligned} \tag{2.7}$$

The Free Energy in equation (2.7) now consists of three terms of which the first two are straightforward to evaluate. They are simply constants:

$$\begin{aligned} -\ln(z) &= -\frac{1}{2}\ln\left(2\pi\sigma^2\right) \\ -\int q(x)\epsilon(x) \, \mathrm{d}x &= -\frac{1}{2\sigma^2} \underbrace{\int q(x)(x-\mu)^2 \, \mathrm{d}x}_{=\sigma^2} = -\frac{1}{2} \end{aligned}$$

The third term is less straightforward. The energy $E(x,y)$, which has been defined as such and is called energy because of its similarities with Helmholtz' Thermodynamic Energy, is still unspecified. It can be simplified because of the assumptions that come with the Laplace approximation (which assumes that the R-density is a sharply peaked Gaussian) by applying a second order Taylor expansion about $x = \mu$. In the following derivation $E(x,y)$ is written without arguments for brevity:

$$\begin{aligned} \int q(x) E(x,y) \, \mathrm{d}x &\approx \int q(x) \left(E\Big|_\mu + \frac{\partial E}{\partial x}\Big|_\mu (x-\mu) + \frac{1}{2} \frac{\partial^2 E}{\partial x^2}\Big|_\mu (x-\mu)^2 \right) \mathrm{d}x \\ &\approx E\Big|_\mu \underbrace{\int q(x) \, \mathrm{d}x}_{=1} + \frac{\partial E}{\partial x}\Big|_\mu \underbrace{\int q(x)(x-\mu) \, \mathrm{d}x}_{=0} + \frac{1}{2} \frac{\partial^2 E}{\partial x^2}\Big|_\mu \underbrace{\int q(x)(x-\mu)^2 \, \mathrm{d}x}_{=\sigma^2} \\ &\approx E(\mu, y) + \frac{1}{2} \frac{\partial^2 E}{\partial x^2}\Big|_\mu \sigma^2 \end{aligned}$$

All three terms of equation (2.7) can now be substituted, which results in a definition of Free Energy as in equation (2.8) that is parametrized by the mean, or expected value μ of the Gaussian R-density. This means that in its internal model, an agent does not track the hidden environment state x, but merely its belief (the expected value under uncertainty) thereof. The Free Energy furthermore depends on the sensory input y and is parametrized by the variance σ^2 of the Gaussian R-density.

$$F\left(\mu, \sigma^2, y\right) = E(\mu, y) + \frac{1}{2}\left(\frac{\partial^2 E(x,y)}{\partial x^2}\Big|_\mu \sigma^2 - \ln\left(2\pi\sigma^2\right) - 1 \right) \tag{2.8}$$

The Energy $E(\mu, y)$ is called the Laplace Encoded Energy, which is the relevant part to arrive at an explicit (practical) definition of the Free Energy. One more simplification is made to remove the dependency of the Free Energy on the variance, which is substituted by its optimal value w.r.t. the Free Energy:

$$\frac{\partial F}{\partial(\sigma^2)} = \frac{1}{2}\left(\frac{\partial^2 E(x, \boldsymbol{y})}{\partial x^2}\Big|_\mu - \frac{1}{\sigma^2} \right) \equiv 0 \qquad \rightarrow \qquad {\sigma^2}^* \equiv \left(\frac{\partial^2 E(x,y)}{\partial x^2}\Big|_\mu \right)^{-1}$$

The result after the Laplace approximation is a Free Energy formulation (equation (2.9)) that depends solely on the agents internal belief μ about the environment state x and on its sensory input y. These

are both quantities that an agent has access to, in contrast to the hidden state x, meaning Free Energy can be evaluated if the Laplace Encoded Energy is defined, which is the topic of section 2.2.2.

$$F(\mu, y) = E(\mu, y) - \frac{1}{2}\left(\ln\left(2\pi\sigma^{2^*}\right) + 1\right) \tag{2.9}$$

In an analogue derivation of the approximation of the Free Energy by the Laplace Encoded Energy for the multivariate set of states $\boldsymbol{x}$ and sensory inputs $\boldsymbol{y}$, the states are represented by the means or expected values of those states, $\boldsymbol{\mu}$.

2.2.2. Gaussian models

With the Laplace approximation in section 2.2.1, the Free Energy has been rewritten to the Laplace Encoded Energy, which is different from the Free Energy by only a constant (equation (2.9)). Minimizing the Laplace Encoded Energy therefore minimizes the Free Energy. From the derivation towards equation (2.7) the definition of the Energy follows. This energy is evaluated for the more general multivariate case:

$$E(\boldsymbol{x}, \boldsymbol{y}) = -\ln\left(p(\boldsymbol{x}, \boldsymbol{y})\right)$$

This Energy depends on the G-density, which in turn depends on the hidden state $\boldsymbol{x}$. In contrast, the Laplace Encoded Energy depends on the internal belief $\boldsymbol{\mu}$ of the external state, which is its sufficient statistic. Therefore the Laplace Encoded Energy in equation (2.10) is defined as a function of the Laplace Encoded G-density, which can be factorised into likelihood and prior densities.

$$E(\boldsymbol{\mu}, \boldsymbol{y}) = -\ln\left(p(\boldsymbol{\mu}, \boldsymbol{y})\right) = -\ln\left(p(\boldsymbol{y}|\boldsymbol{\mu})p(\boldsymbol{\mu})\right) \tag{2.10}$$

Since minimizing this Laplace Encoded Energy minimizes the Free Energy, an explicit definition is required. Remembering that the likelihood density is a generative mapping between the hidden states in the environment, or in this case the belief thereof, and the sensory input it results in, a model can be constructed. The same is true for the prior density regarding the (belief of) the hidden states in the world. This density represents how the agent believes the hidden states in the environment evolve. Both these processes can very well be represented by (stochastic) state space equations, as in equation (2.11).

$$\begin{aligned} \dot{\boldsymbol{\mu}} &= f(\boldsymbol{\mu}) + \boldsymbol{w} \\ \boldsymbol{y} &= g(\boldsymbol{\mu}) + \boldsymbol{z} \end{aligned} \tag{2.11}$$

The first equation in equation (2.11), which describes the state evolution, does not only depend on $\boldsymbol{\mu}$ (and $\boldsymbol{w}$), but also on its derivative $\dot{\boldsymbol{\mu}}$, which has not been discussed as of yet, and is not something that is regarded in the R- or G-densities. It is left as is for now and discussed further in section 2.3. The state space equations are stochastic in nature only due to the stochastic variables $\boldsymbol{w}$ and $\boldsymbol{z}$, which represent noise or unmodelled dynamics. As such, they represent the error on $\boldsymbol{\mu}$ and $\boldsymbol{y}$, and can therefore be considered as $\boldsymbol{\varepsilon}_\mu$ and $\boldsymbol{\varepsilon}_y$ respectively. The definition is presented in equation (2.12).

$$\begin{aligned} \boldsymbol{\varepsilon}_\mu &= \boldsymbol{w} = \dot{\boldsymbol{\mu}} - f(\boldsymbol{\mu}) \\ \boldsymbol{\varepsilon}_y &= \boldsymbol{z} = \boldsymbol{y} - g(\boldsymbol{\mu}) \end{aligned} \tag{2.12}$$

If these noises are assumed to have a Gaussian nature, their probability density function is well defined. Assuming Gaussian noise is further discussed in section 2.3 and chapter 3. Furthermore, the probability density functions of the Laplace Encoded G-density are defined by the probability density functions of these disturbances, since these are fully responsible for the stochastic nature of the state space equations. In equation (2.13) these densities are defined.

$$\begin{aligned} p(\boldsymbol{y}|\boldsymbol{\mu}) = p\left(\boldsymbol{\varepsilon}_y\right) &= \left((2\pi)^q|\Sigma_z|\right)^{-\frac{1}{2}} \exp\left(-\frac{1}{2}\left(\boldsymbol{\varepsilon}_y^T\Sigma_w^{-1}\boldsymbol{\varepsilon}_y\right)\right) \\ p(\boldsymbol{\mu}) = p\left(\boldsymbol{\varepsilon}_\mu\right) &= \left((2\pi)^n|\Sigma_w|\right)^{-\frac{1}{2}} \exp\left(-\frac{1}{2}\left(\boldsymbol{\varepsilon}_\mu^T\Sigma_w^{-1}\boldsymbol{\varepsilon}_\mu\right)\right) \end{aligned} \tag{2.13}$$

Given the conditional and prior densities in equation (2.13), evaluating the Laplace Encoded Energy (equation (2.10)) becomes straightforward:

$$E(\boldsymbol{\mu}, \boldsymbol{y}) = -\ln\left(p(\boldsymbol{y}|\boldsymbol{\mu})\right) - \ln p(\boldsymbol{\mu})$$

$$
\begin{aligned}
&= -\ln\left(\left((2\pi)^q|\Sigma_z|\right)^{-\frac{1}{2}}\exp\left(-\frac{1}{2}\left(\boldsymbol{\varepsilon}_y^T\Sigma_w^{-1}\boldsymbol{\varepsilon}_y\right)\right)\right) - \ln\left(\left((2\pi)^n|\Sigma_w|\right)^{-\frac{1}{2}}\exp\left(-\frac{1}{2}\left(\boldsymbol{\varepsilon}_\mu^T\Sigma_w^{-1}\boldsymbol{\varepsilon}_\mu\right)\right)\right)\\
&= \ln\left(\left((2\pi)^q|\Sigma_z|\right)^{\frac{1}{2}}\right) + \frac{1}{2}\left(\boldsymbol{\varepsilon}_y^T\Sigma_z^{-1}\boldsymbol{\varepsilon}_y\right) + \ln\left(\left((2\pi)^n|\Sigma_w|\right)^{\frac{1}{2}}\right) + \frac{1}{2}\left(\boldsymbol{\varepsilon}_\mu^T\Sigma_w^{-1}\boldsymbol{\varepsilon}_\mu\right)\\
&= \underbrace{\frac{1}{2}\left((n+q)\ln(2\pi) + \ln\left(|\Sigma_w||\Sigma_z|\right)\right)}_{\text{constant}} + \frac{1}{2}\left(\boldsymbol{\varepsilon}_\mu^T\Sigma_w^{-1}\boldsymbol{\varepsilon}_\mu + \boldsymbol{\varepsilon}_y^T\Sigma_z^{-1}\boldsymbol{\varepsilon}_y\right)
\end{aligned}
$$

By means of the Laplace approximation and the assumption of Gaussian disturbances, resulting in Gaussian models, the Free Energy from equation (2.5) can, at the optimal variance of the R-density, be approximated by a quadratic model and sensory prediction error, weighed by their precision. This approximation deviates from the true Free Energy in equation (2.5) by constants, which are smaller whenever the (co)variances are smaller: both of the R-density (the agent has better internal knowledge of its environment) and of the G-density, which is the case when the uncertainties in the environment are smaller (the precision is high). This could be because the dynamical model represents the environment better, or because the disturbances in the environment are small. Precision is simply the inverse of covariance, the weighing factor in the quadratic error formulation of the Laplace Encoded Energy. Therefore, the functional form of the Free Energy that is to be minimized is the quadratic function in equation (2.14), in which precision $\Pi = \Sigma^{-1}$. The influence of the errors on the Energy is thus weighted by the confidence or reliability of the prediction. Even though this is not the Free Energy as in equation (2.5), but an approximation of it, it will be referred to as Free Energy. To mark the difference, it is denoted as $\mathcal{F}$, instead of F.

$$\mathcal{F}(\boldsymbol{\mu}, \boldsymbol{y}) = \frac{1}{2}\left(\boldsymbol{\varepsilon}_\mu^T\Pi_w\boldsymbol{\varepsilon}_\mu + \boldsymbol{\varepsilon}_y^T\Pi_z\boldsymbol{\varepsilon}_y\right) \tag{2.14}$$

With the (approximate) Free Energy defined as a weighted quadratic error function, its minimization has become a very tangible problem. It can be achieved by a simple gradient descent scheme. The minimization addresses both action and perception. Perception is the adjustment of the beliefs about the states in the environment to make the internal beliefs better match the environment states. Action is, in a way, its opposite: by acting on the environment, the agent can change the environment such that it better matches its beliefs about it. How this is done is discussed in chapter 5.

2.3. Generalised motions

The equations in equation (2.11) describe a dynamical system: A system of which the states change over time. It is suggested that a biological agent does not merely model, or keep track of, the (belief of) environment states, but also the temporal derivatives thereof and the sensory input that is the result of these 'generalised motions' [15, 18]. Generalised motions are the instantaneous derivatives of a dynamical process, and they are potentially very useful. In a dynamical environment, Free Energy must be minimized with respect to changing states and changing sensory inputs as a result. As aptly stated in [15]: The gradient descent on Free Energy has to hit a moving target. Generalised motions have effects on the models of the G-density, as shown in section 2.3.1, but are also involved with the uncertainty caused by disturbances (w and z in equation (2.11)), which is discussed in section 2.3.2.

2.3.1. The G-density

The instantaneous derivatives of sensory inputs or internal beliefs of states (state estimates) make a far more precise specification of the environment state. For example, if a state represents a position, all the higher order temporal derivatives (velocity, acceleration, jerk, etc.) give a very good insight into the state trajectory. If an agent can indeed perceive, or keep track of generalised motions, this affects the G-density. The equations in equation (2.11) can be 'generalised' as follows, with p signifying the 'embedding order': The highest dynamical order considered.

$$
\begin{aligned}
\boldsymbol{\mu}' &= f(\boldsymbol{\mu}) + \boldsymbol{w} & \boldsymbol{y} &= g(\boldsymbol{\mu}) + \boldsymbol{z}\\
\boldsymbol{\mu}'' &= \frac{\partial f}{\partial \boldsymbol{\mu}}\boldsymbol{\mu}' + \boldsymbol{w}' & \boldsymbol{y}' &= \frac{\partial g}{\partial \boldsymbol{\mu}}\boldsymbol{\mu}' + \boldsymbol{z}'\\
\boldsymbol{\mu}''' &= \frac{\partial f}{\partial \boldsymbol{\mu}}\boldsymbol{\mu}'' + \boldsymbol{w}'' & \boldsymbol{y}'' &= \frac{\partial g}{\partial \boldsymbol{\mu}}\boldsymbol{\mu}'' + \boldsymbol{z}''
\end{aligned}
$$

$$\vdots \qquad\qquad \vdots$$
$$\boldsymbol{\mu}^{(p+1)} = \frac{\partial f}{\partial \boldsymbol{\mu}}\boldsymbol{\mu}^{(p)} + \boldsymbol{w}^{(p)} \qquad\qquad \boldsymbol{y}^{(p)} = \frac{\partial g}{\partial \boldsymbol{\mu}}\boldsymbol{\mu}^{(p)} + \boldsymbol{z}^{(p)}$$

These generalised equations are simply derivatives of the first equations at increasing higher orders under a local linearity assumption. Since $\boldsymbol{\mu}$ is a time-dependent variable, the chain (and product rule) introduce nonlinear terms at higher-order derivatives of $f(\boldsymbol{\mu})$ and $g(\boldsymbol{\mu})$ with nonlinear combinations of $\boldsymbol{\mu}^{(d)}$, with d indicating a derivative order of 1 or higher. It is common to neglect these terms, assuming local linearity [10, 14, 18]. Generalised motions can be represented in short by a tilde, as demonstrated below for the agent's belief of motion $\tilde{\boldsymbol{\mu}}$. A generalised vector of a signal will be of size $n(p+1)$, if n is the signal dimension and p the embedding order.

$$\tilde{\boldsymbol{\mu}} = \begin{bmatrix}\boldsymbol{\mu}^T & \boldsymbol{\mu}'^T & \boldsymbol{\mu}''^T & \cdots\cdots\cdots & \boldsymbol{\mu}^{(p)T}\end{bmatrix}^T$$

If al generalised signals ($\tilde{\boldsymbol{\mu}}$, $\tilde{\boldsymbol{y}}$, $\tilde{\boldsymbol{w}}$, $\tilde{\boldsymbol{z}}$) are defined similarly, and furthermore the functions $f(\mu)$ and $g(\mu)$ are generalised into $\tilde{f}(\tilde{\boldsymbol{\mu}})$ and $\tilde{g}(\tilde{\boldsymbol{\mu}})$, as shown for $\tilde{f}(\tilde{\boldsymbol{\mu}})$ but completely analogous for $\tilde{g}(\tilde{\boldsymbol{\mu}})$:

$$\tilde{f}(\tilde{\boldsymbol{\mu}}) = \begin{bmatrix}f(\boldsymbol{\mu})^T & \frac{\partial f}{\partial \boldsymbol{\mu}}\boldsymbol{\mu}'^T & \frac{\partial f}{\partial \boldsymbol{\mu}}\boldsymbol{\mu}''^T & \cdots\cdots\cdots & \frac{\partial f}{\partial \boldsymbol{\mu}}\boldsymbol{\mu}^{(p)T}\end{bmatrix}^T$$

The state update equation can now be represented as $\dot{\tilde{\boldsymbol{\mu}}} = \tilde{f} + \tilde{\boldsymbol{w}}$. Since in generalised coordinates, temporal derivatives are represented, the state update equation in generalised form is now merely a shift in dynamical orders. Even though $\dot{\boldsymbol{\mu}} \equiv \boldsymbol{\mu}'$ is not necessarily true, since the dash represents a belief of motion and the dot a true temporal derivative, the agent can represent its G-density model as in equation (2.15), with $\mathcal{D}$ a derivative operator that has a superdiagonal filled with ones.

$$\begin{aligned}\mathcal{D}\tilde{\boldsymbol{\mu}} &= \tilde{f}(\tilde{\boldsymbol{\mu}}) + \tilde{w} \\ \tilde{\boldsymbol{y}} &= \tilde{g}(\tilde{\boldsymbol{\mu}}) + \tilde{z}\end{aligned} \qquad \text{with} \qquad \mathcal{D} = \overset{p+1}{\longleftrightarrow}\ \overset{p+1}{\updownarrow}\begin{bmatrix}0 & 1 & & & \\ & 0 & 1 & & \\ & & \ddots & \ddots & \\ & & & \ddots & 1 \\ & & & & 0\end{bmatrix} \otimes I_n \tag{2.15}$$

The models underlying the G-density now depend on the generalised internal beliefs of environment states $\tilde{\boldsymbol{\mu}}$, the generalised sensory input $\tilde{\boldsymbol{y}}$ and the environment disturbances $\tilde{\boldsymbol{w}}$ and $\tilde{\boldsymbol{z}}$. These disturbances are represented in the Free Energy by means of their precision, or inverse covariance. Generalised coordinates of motion are tightly related to the disturbances or uncertainties, as will be discussed in section 2.3.2 and subsequently chapter 3. In this section it has been shown that the generalised coordinates provide a frame of reference of temporal relations that moves with the expected state ($\tilde{\boldsymbol{\mu}}$) for the gradient descent that is to minimize the Free Energy.

2.3.2. Temporal correlations of disturbances

With the equations underlying the G-density in generalised coordinates (equation (2.15)) come generalised disturbances $\tilde{\boldsymbol{w}}$ and $\tilde{\boldsymbol{z}}$. In section 2.2.2 it has been mentioned that the noises or disturbances are assumed to have underlying Gaussian densities. In contrast to many classical algorithms of perception and control, the Free Energy Principle and Active Inference assume noise to be coloured. Classical white noise, a stochastic process with the Markov property, has no memory. For such processes, there is no correlation between any two samples and temporal derivatives are not defined. It is infinitely rough. Coloured, or non-Markovian noise, in contrast, is smooth. Samples are correlated and most importantly, temporal derivatives are defined, and also smooth. Intuitively, the assumption of coloured noise is a very valid one. The true sources of noise, whether they are unmodelled dynamics or some disturbing force like, for example, wind, are always dynamical processes. Although wind, as an example, can be quite erratic in nature, the process is bound by physics and has underlying dynamics. It cannot change direction infinitely fast, nor increase (or decrease) infinitely fast in speed.

The smooth signals have infinitely many derivatives, which are in turn also smooth. These signals have a finite variance, smooth autocorrelation functions and because of the existing correlations, both

within and among dynamical orders, the derivatives provide information about the trajectories of the noise, just as the derivatives, or generalised motions of $\tilde{\boldsymbol{\mu}}$ provide more information about the trajectory of the states. Not only are the generalised disturbances $\tilde{\boldsymbol{w}}$ and $\tilde{\boldsymbol{z}}$ necessary for equation (2.15) to exist, they provide information about the disturbances in the environment and could therefore be very beneficial in perception and control under the presence of such disturbances.

With generalised equations of motion as in equation (2.15), the Free Energy Formulation becomes 'generalised' too. In order to find the generalised practical formulation of Free Energy, the assumption of correlation between dynamical orders is important. In the works of [4, 23, 26, 27], it is assumed that the noise at different dynamical orders is uncorrelated. So, each dynamical order d is perturbed by a noise $\boldsymbol{w}^{(d)}$ from its own independent source. As a result, the likelihood of the G-density can be written as a product of conditional densities and similarly, the prior density is factorised over consecutive orders (as shown in, among others, [4]), such that:

$$p(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{\mu}}) = \prod_{d=0}^{p} p\big(\boldsymbol{y}^{(d)} | \boldsymbol{\mu}^{(d)}\big) p\big(\boldsymbol{\mu}^{(d+1)} | \boldsymbol{\mu}^{(d)}\big)$$

The resulting practical form of Free Energy, which is the variable part that remains after evaluation the negative logarithm of the G-density, is now a summation over dynamical orders:

$$\mathcal{F}(\tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{y}}) = \sum_{d=0}^{p} \frac{1}{2}\Big(\boldsymbol{\varepsilon}_{\mu}^{(d)T} \Pi_{w}^{(d)} \boldsymbol{\varepsilon}_{\mu}^{(d)} + \boldsymbol{\varepsilon}_{y}^{(d)T} \Pi_{z}^{(d)} \boldsymbol{\varepsilon}_{y}^{(d)}\Big)$$

In some works of Friston [10, 17, 18], however, it is suggested that the noise at different dynamical orders is correlated. This is the case that will be investigated in this work, and under this assumption, the G-density does not factorise into a product over dynamical orders. The resulting practical formulation of Free Energy reads as in equation (2.16) and depends on the generalised precisions $\tilde{\Pi}$.

$$\mathcal{F} = \frac{1}{2}\big(\tilde{\boldsymbol{\varepsilon}}_{\mu}^{T} \tilde{\Pi}_{w} \tilde{\boldsymbol{\varepsilon}}_{\mu} + \tilde{\boldsymbol{\varepsilon}}_{y}^{T} \tilde{\Pi}_{z} \tilde{\boldsymbol{\varepsilon}}_{y}\big) \tag{2.16}$$

The generalised precisions $\tilde{\Pi}_{w}$ and $\tilde{\Pi}_{z}$ are the precision matrices of the generalised noises $\tilde{\boldsymbol{w}}$ and $\tilde{\boldsymbol{z}}$. What such a matrix looks like and how it relates to the generalised coordinates is the topic of chapter 3.

2.3.3. Generalised motions in Active Inference

This last section of this chapter concludes the introduction to Free Energy and Active Inference, having provided an overview of how a neurological inspired theory of the biological brain translates into mathematics suited for robotics perception and control, and aiming to answer the first question from section 1.2.1: 'What is the role of generalised coordinates in Active Inference?'. Generalised coordinates of motion are an important concept within this theory, as they provide information about the trajectory of the (belief of) environmental states, providing a reference frame for the gradient descent that minimizes the Free Energy as well as information about the motion of the disturbances in the environment which potentially allows the agent to better respond to such disturbances. Since the Free Energy Principle unifies perception and action, the role of generalised coordinates is twofold as well. The following chapters are more detailed chapters on the generalised precision matrix that is introduced in equation (2.16) as well as the implications it has in Active Inference (chapter 3), and on the practical side of generalised coordinates: how to work with these in a traditional robotics control setting (chapter 4).

3

Coloured noise

In chapter 2 it has been shown how Free Energy can be formulated and that it is weighted by the precision of the uncertainty of the models that encode the belief the evolution of states in the environment, and how these are mapped onto the sensory input of an agent. When generalised coordinates are applied, the weighting factors in the Free Energy function become generalised precision matrices, which are mentioned by Friston in [10, 17, 18]. This matrix encodes the precision of the noise present in the environment and on the sensory channels, relying on the structure in coloured noise to represent the precision. This chapter is dedicated to the derivation of a generalised precision matrix in great detail, since no such derivation has yet been presented. In-depth knowledge about such a matrix is necessary for the general application of generalised coordinates and for the understanding of its function. After a full derivation of the entries of this matrix, an answer to the research question 'What is the relationship between generalised coordinates and coloured noise?' can be formulated.

3.1. Covariance of coloured noise

For the evaluation of the Free Energy as in equation (2.16) and repeated below, a definition of the matrices $\tilde{\Pi}_w$ and $\tilde{\Pi}_z$ is required.

$$\mathcal{F}(\tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{y}}) = \frac{1}{2}\left(\tilde{\boldsymbol{\varepsilon}}_\mu^T \tilde{\Pi}_w \boldsymbol{\varepsilon}_\mu + \boldsymbol{\varepsilon}_y^T \tilde{\Pi}_z \boldsymbol{\varepsilon}_y\right)$$

The precision matrix $\tilde{\Pi}$ is merely the inverse of the generalised covariance matrix $\tilde{\Sigma}$. This is a covariance matrix of the generalised coloured noise, and it is this covariance matrix that needs derivation. The inversion of it is only a simple, final step in the process of finding the generalised precision matrix. The generalised covariance matrix in question is that of noise or unmodeled dynamics $\tilde{\boldsymbol{w}}$ and $\tilde{\boldsymbol{z}}$ of the prior and conditional densities (equation (2.15)) respectively that make up the G-density. The derivation of the generalised covariance matrix for each of these noises $\tilde{\boldsymbol{w}}$ and $\tilde{\boldsymbol{z}}$ is completely identical and therefore this chapter will focus on the derivation of the generalised precision (covariance) for $\tilde{\boldsymbol{w}}$. The noise 'acts' on the generalised coordinates, which are the belief $\tilde{\boldsymbol{\mu}}$ of the agent in the case of noise $\tilde{\boldsymbol{w}}$. When it is assumed to be coloured, temporal derivatives exist and therefore generalised noise exists. Noise can be assumed to be coloured when it is assumed to be a dynamical process itself. If the noise is the result of a natural process, this assumption is valid. A mathematical definition of the noise follows.

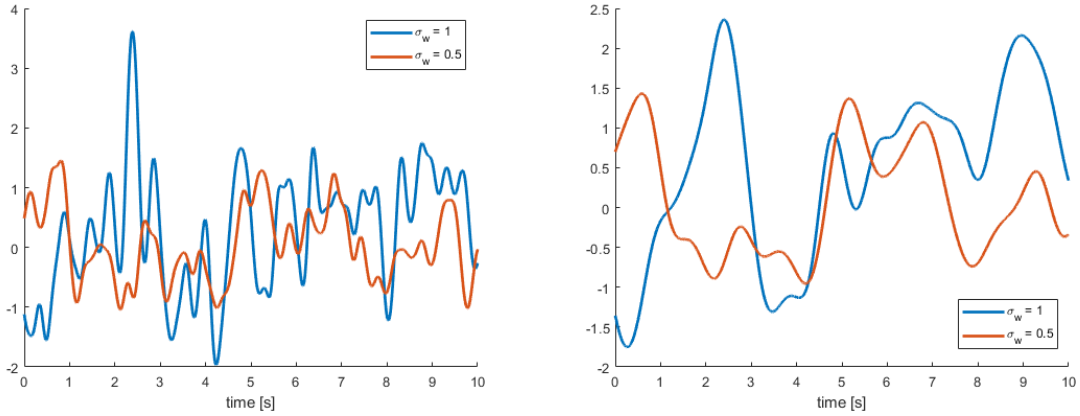
3.1.1. Definition of the noise

Coloured noise is naturally present in physical environments and can take on many forms. For this application of Active Inference, it will be assumed to be of Gaussian nature, as proposed by Friston [18]. Knowledge about the noise signal is required, and therefore a definition of a Gaussian noise is given. Coloured noise can be 'made' by the convolution of a white noise and a Gaussian filter. Given a zero mean white noise $\boldsymbol{\omega}_w$ with variance Σ_w, a Gaussian filter can be constructed that, upon convolution of white noise and filter, yields a coloured noise signal [31]. The filter is shown for a single noise process

ω_w with variance σ_w^2 in equation (3.1).

$$h_w(t) = \sqrt{\frac{\Delta t}{s_w\sqrt{\pi}}}\exp\left(-\frac{t^2}{2s_w^2}\right) \tag{3.1}$$

This filter depends on the sampling time Δt of the white noise signal, and has a kernel width s_w which acts as a smoothing factor. The scaling factor $\sqrt{\frac{\Delta t}{s_w\sqrt{\pi}}}$ ensures that the filtered coloured noise has the same variance σ_w^2 as the white noise it was created from. Throughout this chapter, the time-argument of dynamical variables is often included in contrast to the rest of the report for its relevance in the derivations in this chapter. An added advantage of describing coloured noise as the convolution of white noise and a Gaussian filter is that this method can be applied to fabricate coloured noise for simulations. The resulting coloured noise is smooth, correlated and infinitely differentiable, since each noise is created from a different, uncorrelated white noise signal. As such, all the separate noise signals are uncorrelated. Matlab code to create such coloured noise is available in listing B.1 in appendix B.1, and examples of such smooth noise signals using different filters are shown in figure 3.1. The effects of the variance (or standard deviation) and kernel width are quite clear from these graphs.



(a) Smoothing filter $s_w = 0.1$

(b) Smoothing filter $s_w = 0.3$

Figure 3.1: Two different coloured noise signals ($\Delta t = 0.001s$) by different filters.

3.1.2. The generalised covariance matrix

Noise disturbs all dynamical orders of a generalised signal. Therefore, for an n-dim signal there must be n-dim noise, of which the noise on each dimension can be independent. Generalised noise $\tilde{\boldsymbol{w}}$ of embedding order p is then defined as below, and has dimension $n(p+1)\times 1$:

$$\boldsymbol{w} = \begin{bmatrix} w_1 & w_2 \ldots\ldots\ldots\ldots w_n \end{bmatrix}^T \qquad \tilde{\boldsymbol{w}} = \begin{bmatrix} \boldsymbol{w}^T & \boldsymbol{w}'^T & \boldsymbol{w}''^T \ldots\ldots\ldots \boldsymbol{w}^{(p)T} \end{bmatrix}^T$$

The generalised covariance matrix is then the matrix with the (cross-)covariances (C) of the generalised noise. For an n-dim signal with embedding order p, a generalised covariance matrix is shown below.

$$\tilde{\Sigma}_w = \begin{bmatrix} C[w_1,w_1] & \cdots & C[w_1,w_n] & & C\left[w_1,w_1^{(p)}\right] & \cdots & C\left[w_1,w_n^{(p)}\right] \\ \vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots \\ C[w_n,w_1] & \cdots & C[w_n,w_n] & & C\left[w_n,w_1^{(p)}\right] & \cdots & C\left[w_n,w_n^{(p)}\right] \\ & & & & & & \\ C\left[w_1^{(p)},w_1\right] & \cdots & C\left[w_1^{(p)},w_n\right] & & C\left[w_1^{(p)},w_1^{(p)}\right] & \cdots & C\left[w_1^{(p)},w_n^{(p)}\right] \\ \vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots \\ C\left[w_n^{(p)},w_1\right] & \cdots & C\left[w_n^{(p)},w_n\right] & & C\left[w_n^{(p)},w_1^{(p)}\right] & \cdots & C\left[w_n^{(p)},w_n^{(p)}\right] \end{bmatrix}$$

Because the different noise signals are uncorrelated, as stated in section 3.1.1, so are the derivatives of these signals. Hence, every cross-covariance term is 0. This greatly simplifies the generalised covariance matrix, which is displayed in equation (3.2).

$$\tilde{\Sigma}_w = \begin{bmatrix} C[w_1, w_1] & \cdots & 0 & & C\left[w_1, w_1^{(p)}\right] & \cdots & 0 \\ \vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots \\ 0 & \cdots & C[w_n, w_n] & & 0 & \cdots & C\left[w_n, w_n^{(p)}\right] \\ & \vdots & & \ddots & & \vdots & \\ C\left[w_1^{(p)}, w_1\right] & \cdots & 0 & & C\left[w_1^{(p)}, w_1^{(p)}\right] & \cdots & 0 \\ \vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots \\ 0 & \cdots & C\left[w_n^{(p)}, w_n\right] & & 0 & \cdots & C\left[w_n^{(p)}, w_n^{(p)}\right] \end{bmatrix} \tag{3.2}$$

Equation (3.2) shows the need to evaluate the covariance of a noise signal with its own derivatives. What follows now is a derivation of such a covariance term [6]. Since each w_i is a stationary process, it has the following properties (expectation E, variance V, covariance C) for a single w_i, simply called $w(t)$ from here on:

$$E[w(t)] = 0, \quad V[w(t)] = \sigma_w^2, \quad C[w(t+\tau), w(t)] = \gamma_w(\tau)$$

As such, the autocorrelation function ρ is as in equation (3.3).

$$\rho_w(\tau) = \frac{\gamma_w(\tau)}{\gamma_w(0)} = \frac{\gamma_w(\tau)}{\sigma_w^2} \tag{3.3}$$

Because $w(t)$ was created by convolution with a Gaussian filter (section 3.1.1), it is smooth and differentiable. Hence, the following exist:

$$\frac{\mathrm{d}w(t)}{\mathrm{d}t} = \dot{w}(t), \quad \int_a^b w(t)\,\mathrm{d}t$$

Since $w(t)$ is smooth, it must have continuity at any particular time. This requires convergence in mean square [24], as in equation (3.4).

$$\lim_{\Delta t \to 0} E[w(t+\Delta t) - w(t)]^2 = 0 \tag{3.4}$$

Some small calculations show that this is equivalent to a requirement in terms of autocorrelation:

$$\begin{aligned} \gamma_w(\Delta t) &= E\Big[\big(w(t+\Delta t) - E[w(t+\Delta t]\big)\big(w(t) - E[w(t)]\big)\Big] \\ \rho_w(\Delta t) &= \frac{E[w(t+\Delta t)w(t)]}{\sigma_w^2} \\ E[w(t+\Delta t) - w(t)]^2 &= E[w(t+\Delta t)]^2 - 2E[w(t+\Delta t)w(t)] + E[w(t)]^2 \\ &= \sigma_w^2 - 2E[w(t+\Delta t)w(t)] + \sigma_w^2 \\ &= 2\sigma_w^2(1 - \rho_w(\Delta t)) \end{aligned}$$

And so it is required that the following limit holds:

$$\lim_{\Delta t \to 0} 2\sigma_w^2(1 - \rho_w(\Delta t)) = 0$$

Which means $\rho_w(\tau)$ is continuous at $\tau = 0$. In very similar fashion, the condition in equation (3.5) must hold for the differentiated signal:

$$\lim_{\Delta t \to 0} E\left[\dot{w}(t) - \frac{w(t+\Delta t) - w(t)}{\Delta t}\right]^2 = 0 \tag{3.5}$$

An alternative description of $\dot{w}(t)$, which is well-defined for all $\Delta t \neq 0$, provides the following properties of $\dot{w}(t, \Delta t)$ for expectation and (co)variance:

$$\dot{w}(t, \Delta t) = \frac{w(t + \Delta t) - w(t)}{\Delta t}$$

$$E[\dot{w}(t, \Delta t)] = 0, \quad V[\dot{w}(t, \Delta t)] = \frac{2\sigma_w^2(1 - \rho_w(\Delta t))}{(\Delta t)^2} = \sigma_w^2 \frac{2\rho_w(0) - \rho_w(\Delta t) - \rho_w(-\Delta t)}{(\Delta t)^2},$$

$$C[w(t + \tau), \dot{w}(t, \Delta t)] = \sigma_w^2 \frac{\rho_w(\tau - \Delta t) - \rho_w(\tau)}{\Delta t},$$

$$C[\dot{w}(t + \tau, \Delta u), \dot{w}(t, \Delta t)] = \sigma_w^2 \frac{\rho_w(\tau + \Delta u - \Delta t) - \rho_w(\tau + \Delta u) - \rho_w(\tau - \Delta t) + \rho_w(\tau)}{\Delta t \Delta u}$$

Since $\dot{w}(t, \Delta t)$ is well defined in the limit $\Delta t \to 0$, it is reasonable to expect that all the above tend to finite limits as well. The limits to 0 of the expressions above are gathered in equation (3.6).

$$\begin{gathered} E[\dot{w}(t)] = 0, \quad V[\dot{w}(t)] = \sigma_w^2 \ddot{\rho}_w(0), \\ C[w(t + \tau), \dot{w}(t)] = \sigma_w^2 \dot{\rho}_w(\tau), \quad C[\dot{w}(t + \tau), \dot{w}(t)] = -\sigma_w^2 \ddot{\rho}_w(\tau) \end{gathered} \tag{3.6}$$

For $V[\dot{w}(t)]$ to exist, it is required that $\rho_w(\tau)$ is differentiable twice. This will show in sections 3.2.1 and 3.2.2. The above result could be extended to higher order derivatives, which will not be further elaborated on. However, from a non-mathematical, though practical perspective, the result can be generalized. The covariance of the noise signal with itself or one of its derivatives is the variance of the original noise signal $w(t)$, multiplied by a derivative of the autocorrelation function evaluated at 0 (because no lag is considered). The order of the derivative of the autocorrelation function equals the sum of the order of derivatives of the desired covariance. Furthermore, if this sum is the result of odd derivatives, a minus-sign is present. With this information and knowledge of the derivatives of $\rho_w(\tau)$, the generalised covariance matrix and ultimately the generalised precision matrix, can be constructed. Therefore, the autocorrelation function must be evaluated.

3.2. Autocorrelation of Gaussians

In section 3.1.2 it has become clear that the entries of the generalised covariance matrix are a product of the variance of the signal, which is known to be the variance of the white noise signal underlying the coloured noise, and evaluations of the derivative of the autocorrelation function of the noise (equation (3.6)). This section is a detailed derivation of the derivatives of the autocorrelation function.

3.2.1. The autocorrelation function

The autocorrelation function $\rho_w(\tau)$ of the coloured noise $w(t)$ is equal to the autocorrelation function $\rho_h(\tau)$ of the filter $h_w(t)$ since there is no correlation in the white noise $\omega(t)$ that $w(t)$ was created from. It's definition as the scaled autocovariance $\gamma_h(\tau)$ is shown in equation (3.7) and after computing the autocovariance function $\gamma_h(\tau)$ from the definition in equation (3.8) to the function in equation (3.9) it can be written as in equation (3.10).

$$\rho_h(\tau) = \frac{\gamma_h(\tau)}{\gamma_h(0)} \tag{3.7}$$

$$\gamma_h(\tau) = \int_{-\infty}^{\infty} h(t + \tau) h(t) \, \mathrm{d}t \tag{3.8}$$

$$= \int_{-\infty}^{\infty} \sqrt{\frac{\Delta t}{s_w \sqrt{\pi}}} \exp\left(-\frac{(t + \tau)^2}{2 s_w^2}\right) \sqrt{\frac{\Delta t}{s_w \sqrt{\pi}}} \exp\left(-\frac{t^2}{2 s_w^2}\right) \mathrm{d}t$$

$$= \frac{\Delta t}{s_w \sqrt{\pi}} \int_{-\infty}^{\infty} \exp\left(\frac{1}{2 s_w^2}\left(-(t + \tau)^2 - t^2\right)\right) \mathrm{d}t$$

$$= K_1 \int_{-\infty}^{\infty} \exp\left(\frac{1}{2 s_w^2}\left(-2t^2 - 2t\tau - \tau^2\right)\right) \mathrm{d}t \qquad K_1 = \frac{\Delta t}{s_w \sqrt{\pi}}$$

$$= K_1 \int_{-\infty}^{\infty} \exp\left(-\frac{1}{2 s_w^2}\left(\left(\sqrt{2} t + \frac{1}{\sqrt{2}} \tau\right)^2 + \frac{1}{2} \tau^2\right)\right) \mathrm{d}t$$

$$= K_1 \exp\left(-\frac{\tau^2}{4s_w^2}\right)\int_{-\infty}^{\infty}\exp\left(-\frac{1}{2s_w^2}\left(\sqrt{2}t+\frac{1}{\sqrt{2}}\tau\right)^2\right)\mathrm{d}t$$

$$= K_1K_2\int_{-\infty}^{\infty}\exp\left(-\frac{1}{s_w^2}\left(t+\frac{1}{2}\tau\right)^2\right)\mathrm{d}t \qquad K_2 = \exp\left(-\frac{\tau^2}{4s_w^2}\right)$$

$$= K_1K_2\sqrt{s_w^2\pi} \qquad \left(\int_{-\infty}^{\infty}\exp\left(-a(x+b)^2\right)\mathrm{d}x = \sqrt{\frac{\pi}{a}}\right)$$

$$= \frac{\Delta t}{s_w\sqrt{\pi}}\exp\left(-\frac{\tau^2}{4s_w^2}\right)\sqrt{s_w^2\pi}$$

$$= \Delta t\ \exp\left(-\frac{\tau^2}{4s_w^2}\right) \tag{3.9}$$

$$\rho_h(\tau) = \frac{\Delta t}{\Delta t}\exp\left(-\frac{\tau^2}{4s_w^2}\right) = \exp\left(-\frac{\tau^2}{4s_w^2}\right) \tag{3.10}$$

The resulting expression for autocorrelation in equation (3.10) is a well-defined function of the lag τ and is parametrized by the kernel width of the filter, also the smoothness of the coloured noise.

3.2.2. Derivatives of the autocorrelation function

Given the definition of the autocorrelation function in equation (3.10), its derivatives can be computed. Because of the exponential function, this is a straightforward process of applying the chain rule and product rule. The full computation of the derivatives up to and including the 10th derivative is shown in appendix A. These derivatives of the autocorrelation function must be evaluated at $\tau = 0$. This yields 0 for all odd derivatives and the following expressions for the even derivatives (from equations (A.2) to (A.7)):

$$\rho_h(0) = 1 \qquad \rho_h^{(6)}(0) = -\frac{15}{(2s_w^2)^3}$$

$$\ddot{\rho_h}(0) = -\frac{1}{(2s_w^2)} \qquad \rho_h^{(8)}(0) = \frac{105}{(2s_w^2)^4}$$

$$\rho_h^{(4)}(0) = \frac{3}{(2s_w^2)^2} \qquad \rho_h^{(10)}(0) = -\frac{945}{(2s_w^2)^5}$$

These zero-lag evaluations of the autocorrelation follow a pattern. This is a direct result of the structure of the autocorrelation function and the application of chain- and product rules in differentiation. It is relevant to know the expressions above for any desired even derivative. An analysis of the result above is presented in appendix A and the result can be generalised into equation (3.11). This equation presents the means to quickly compute the entries of the generalised covariance matrix whilst knowing only the kernel width (smoothness) of the filter that defines the coloured noise based on white noise and the variance of the noise signal.

$$\rho_h^{(k)}(0) = \frac{1}{\left(\sqrt{2}\,s_w\right)^k}\prod_{j=0}^{k}(1-j) \qquad k,j \in 2\mathbb{N} \tag{3.11}$$

3.3. The generalised precision matrix

3.3.1. Matrix formulation

All that remains to compute the required generalised precision matrix is to construct the generalised covariance matrix from the knowledge of section 3.1.2. All the required entries are a multiplication of the variance of the original noise signal and a derivative of the autocorrelation function evaluated at 0. All process noise signals $w_i(t)$ have been created using the same filter $h_w(t)$. Only the variances of the signals themselves are different. It is therefore more concise and easier to write the resulting generalised covariance matrix as a Kronecker product of the so-called temporal covariance matrix $S(s_w^2)$

(equation (3.12)) and the covariance matrix of $\boldsymbol{w}(t)$, Σ_w, as in equation (3.13), and the generalised precision matrix as in equation (3.14), finally resulting in equation (3.15).

$$S\left(s_w^2\right) = \begin{bmatrix} \rho_h & 0 & \rho_h^{(2)} & 0 & \rho_h^{(4)} & 0 & \cdots & \rho_h^{(p)} \\ 0 & -\rho_h^{(2)} & 0 & -\rho_h^{(4)} & 0 & -\rho_h^{(6)} & & \vdots \\ \rho_h^{(2)} & 0 & \rho_h^{(4)} & 0 & \rho_h^{(6)} & 0 & & \vdots \\ 0 & -\rho_h^{(4)} & 0 & -\rho_h^{(6)} & 0 & -\rho_h^{(8)} & & \vdots \\ \rho_h^{(4)} & 0 & \rho_h^{(6)} & 0 & \rho_h^{(8)} & 0 & & \vdots \\ 0 & -\rho_h^{(6)} & 0 & -\rho_h^{(8)} & 0 & -\rho_h^{(10)} & \ddots & \vdots \\ \vdots & & & & & & & \\ \rho_h^{(p)} & \cdots & & & & & (-1)^p & \rho_h^{(2p)} \end{bmatrix} \tag{3.12}$$

In equation (3.12), each ρ_h is evaluated at 0. This has been omitted for a more concise and clear notation.

$$\tilde{\Sigma}_w = S(s_w^2) \otimes \Sigma_w \tag{3.13}$$

$$\tilde{\Pi}_w = S(s_w^2)^{-1} \otimes \Sigma_w^{-1} \tag{3.14}$$

Finally:

$$\tilde{\Pi}_w = \begin{bmatrix} 1 & 0 & -\frac{1}{2s_w^2} & 0 & \frac{3}{(2s_w^2)^2} & 0 & \cdots & \rho_h^{(p)} \\ 0 & \frac{1}{2s_w^2} & 0 & -\frac{3}{(2s_w^2)^2} & 0 & \frac{15}{(2s_w^2)^3} & & \vdots \\ -\frac{1}{2s_w^2} & 0 & \frac{3}{(2s_w^2)^2} & 0 & -\frac{15}{(2s_w^2)^3} & 0 & & \vdots \\ 0 & -\frac{3}{(2s_w^2)^2} & 0 & \frac{15}{(2s_w^2)^3} & 0 & -\frac{105}{(2s_w^2)^4} & & \vdots \\ \frac{3}{(2s_w^2)^2} & 0 & -\frac{15}{(2s_w^2)^3} & 0 & \frac{105}{(2s_w^2)^4} & 0 & & \vdots \\ 0 & \frac{15}{(2s_w^2)^3} & 0 & -\frac{105}{(2s_w^2)^4} & 0 & \frac{945}{(2s_w^2)^5} & \ddots & \vdots \\ \vdots & & & & & & & \\ \rho_h^{(p)} & \cdots & & & & & (-1)^p & \rho_h^{(2p)} \end{bmatrix}^{-1} \otimes \begin{bmatrix} \sigma_{w_1}^2 & & 0 \\ & \ddots & \\ 0 & & \sigma_{w_n}^2 \end{bmatrix}^{-1} \tag{3.15}$$

Lastly, equation (3.11) can be rewritten to equation (3.16) which provides the value of the i^{th} diagonal element of $S\left(s_w^2\right)$.

$$S\left(s_w^2\right)_{i,i} = -\frac{1}{\left(2s_w^2\right)^{i-1}} \prod_{j=1}^{i} (2j-3) \qquad i,j \in \mathbb{N}\backslash\{0\} \tag{3.16}$$

Keep in mind that equation (3.16) only provides the diagonal terms of $S\left(s_w^2\right)$. These terms are all positive but the same terms on the anti-diagonals are alternatingly positive and negative. Depending on the application, either equation (3.11) or equation (3.16) might be preferred. Given the derivation leading up to equation (3.15), obtaining the matrix for any desired combination of noise parameters and embedding order has become very simple. In listing 3.1 a Matlab function is displayed that creates such a matrix. The full script with more explanatory comments about inputs and outputs of the function is provided in listing B.2 in appendix B.1.

Listing 3.1: A Matlab function `f_precision` that creates a generalised precision matrix as in equation (3.15). The full script with an explanation of inputs and outputs of the function is provided in listing B.2 in appendix B.1. This code is very similar to that of SPM_DEM_R.m from [9].

```
function Pi_ = f_precision(s,sigma,p)

pp = p+1; % embedding order raised by one, for convenience

k = 0:2:2*p; % order of the required autocorrelation derivatives
```

```
rho(1+k) = cumprod(1-k)./((sqrt(2).*s).^k); % rho^(k)(0)

S = zeros(pp,pp); % preallocation of the temporal variance matrix
for r = 1:pp % One row for every embedding order
    S(r,:) = rho(r:r+p); % assembly of the temporal variance matrix
    rho = -rho; % minus signs change every row
end

Pi_ = kron(inv(S),inv(diag(sigma.^2))); % generalised covariance matrix
```

3.3.2. Generalised precision influence

Given a coloured noise and the generalised noise which contains it derivatives, its generalised covariance matrix can be constructed, of which the generalised precision matrix is the inverse. Using the properties of the noise signal and the properties of the filter it was created with, the autocorrelation function can be used to very easily compute the entries of the required matrix. These results have been generalised such that only very simple, straightforward computations are required that depend only on the variance of the noise and the kernel width of the filter, without requiring any integration or differentiation. Inspection of equation (3.15) reveals the influence of the generalised precision matrix. It depends on two variables: s_w^2 and Σ_w in the case of the process noise $\boldsymbol{w}$. Completely analogous, the observation noise will influence the generalised precision matrix $\tilde{\Pi}_z$ with smoothness s_z^2 and Σ_z. Remember (equation (2.16)) that the generalised precision matrix weighs the squared errors of the agent's belief of the state $\tilde{\boldsymbol{\mu}}$ and the sensory input $\tilde{\boldsymbol{y}}$. The elements of the generalised precision matrix are a ratio of the smoothness of the noise, represented by the kernel width of the Gaussian filter, and by the variance of the noise. It is assumed that the smoothness $s_w < 1$ [18]. Since it occurs in the numerators, precision increases with smoothness, however converges as the dynamic order increases. As a result, there is no added value in an increase of the embedding order beyond $p = 5$ [18]. The effect of variance is opposite. It occurs in the denominators, therefore precision decreases with increased variance, given variance $\sigma_{w_i}^2 < 1$ too. The assumptions that variance and smoothness are smaller than one are not further reviewed in this work. However it is noteworthy that, if they are larger than one, their effects reverse and that is peculiar.

With the knowledge presented in this chapter, an answer to the research question 'What is the relationship between generalised coordinates and coloured noise?' can be formulated. It is clear from the generalised precision matrix that the quadratic errors, which are caused by the noises, are weighted by the precisions thereof, which is greater when noise is smaller and/or smoother. Hence, the more confidence an agent can have over its perception and the knowledge of the environment, the more it can act on perceived error. Intuitively, in the opposite case: If there is a lot of noise present in the environment, one may not want to act to quickly when error is perceived, or change ones belief of the environment based on noise. Furthermore, when smoothness becomes very small, the precision for higher dynamical orders vanishes, with only the zero-order remaining in the extreme case (the upper left 1 in equation (3.15)). So, in the case of a very noisy environment, an agent will rely mostly on lower order dynamics but when precision is high, higher order motions can be taken into account. Generalised precision is only possible when noise is coloured, and because generalised coordinates are employed, the agent has more knowledge of the structure of the noise that affect its models and knows how much confidence to have on the different dynamical orders.

4

Generalised motions

Generalised coordinates of motion are an important element of the Free Energy principle and the Active Inference framework. They were introduced in chapter 2 and their relationship with coloured noise was explored in chapter 3. As stated in chapter 1, generalised coordinates of a dynamic process always exist, but including them in robotics control is something different. Although an agent can keep track of generalised (beliefs of) motions by means of its internal model of the environment, its sensory input is limited to the physical sensors that are applied for robotics control, in contrast to biological systems that work with analogue data [18]. It is generally not possible (or feasible) to measure a decent amount of temporal derivatives of a dynamical process, since no such sensors are generally available. To incorporate generalised motions to their full extent in an Active Inference control loop, they must be derived from the available sensory input. This chapter explores the application of finite differences to achieve this, answering the research question 'How can generalised coordinates be generally applied in the Active Inference framework when they are not readily available?'

4.1. Finite differences

A robotics control loop, assuming it is not executed on an analog system, is always sampled, even if the math is described in continuous time. The sensory input to an agent on a digital system is therefore always a sequence of samples. At a certain measurement point in time, surrounding measurement values (past or future) provide finite differences that can be used to numerically estimate the temporal derivatives of a measurement. In this section, a detailed derivation of the expressions required for these estimates is given.

4.1.1. Taylor expansion

The Taylor expansion for a discrete signal y with interval h for a surrounding value y_{k+j} of y_k can be computed given the signal derivatives at y_k, as in equation (4.1).

$$y_{k+j} = \sum_{i=0}^{i_{max}} \frac{(jh)^i}{i!} y_k^{(i)} \tag{4.1}$$

In a Taylor expansion, $i_{max} = \infty$. However, in practice, the sum is finite (Taylor series) and an error remains. Below expansions up to and including the 4th order term ($i_{max} = 4$), with the remaining error, are shown as an example for values of the signal one and two ($j = -1, -2$) intervals back:

$$y_{k-1} = y_k - hy_k^{(1)} + \frac{h^2}{2} y_k^{(2)} - \frac{h^3}{3!} y_k^{(3)} + \frac{h^4}{4!} y_k^{(4)} + \mathcal{O}(h^5)$$

$$y_{k-2} = y_k - 2hy_k^{(1)} + \frac{4h^2}{2} y_k^{(2)} - \frac{8h^3}{3!} y_k^{(3)} + \frac{16h^4}{4!} y_k^{(4)} + \mathcal{O}(h^5)$$

The largest error is that of the omitted terms with the lowest derivative, which is the 5th in the example above ($i = 5$). With fewer derivative terms the expressions become shorter and the error higher in

magnitude, assuming h is sufficiently small ($\ll 1$). There is a trade-off: more derivative information makes for a better estimate, but a more complicated expression.

4.1.2. Approximating derivatives

The Taylor series mentioned above allow for the computation of the value of a signal at a previous instant given the derivatives of the signal, but it is the opposite that must be achieved. To do so for a first-order derivative $y_k^{(1)}$, a Taylor series of y_{k-1} that includes this term is required, which can be rewritten to compute the signal derivative (equation (4.2)), and is as follows:

$$\begin{aligned} y_{k-1} &= y_k - h y_k^{(1)} + \mathcal{O}(h^2) \\ y_k^{(1)} &= \frac{1}{h}(y_k - y_{k-1}) + \mathcal{O}(h) \approx \frac{1}{h}(y_k - y_{k-1}) \end{aligned} \tag{4.2}$$

For a small enough value of h, this is consistent with the definition of a derivative by a backwards difference, as in equation (4.3).

$$y_k^{(1)} = \lim_{h \to 0} \frac{1}{h}(y_k - y_{k-1}) \tag{4.3}$$

Similarly, with Taylor series for y_{k-1} and y_{k-2} with ample terms to include the second derivative, an approximation for the second derivative can be obtained. The required equations are shown below. By combining these equations linearly, all terms but the term including the second derivative can be eliminated:

$$\left.\begin{aligned} \tfrac{1}{2} \times \left[y_k &= y_k \right] \\ -1 \times \left[y_{k-1} &= y_k - h y_k^{(1)} + \tfrac{1}{2} h^2 y_k^{(2)} + \mathcal{O}(h^3) \right] \\ \tfrac{1}{2} \times \left[y_{k-2} &= y_k - 2h y_k^{(1)} + 2h^2 y_k^{(2)} + \mathcal{O}(h^3) \right] \end{aligned}\right\} \frac{1}{2} y_{k-2} - y_{k-1} + \frac{1}{2} y_k = \frac{1}{2} h^2 y_k^{(2)} + \mathcal{O}(h^3)$$

So, an approximation for the second derivative of the signal is given in equation (4.4):

$$y_k^{(2)} = \frac{1}{h^2}(y_{k-2} - 2y_{k-1} + y_k) + \mathcal{O}(h) \tag{4.4}$$

Increased accuracy The approximation for the second-order derivative in equation (4.4) has an error of the order of the sampling time. Whenever terms from (k) up to and including $(k-d)$ are utilised to construct the d^{th} derivative, the error will be of $\mathcal{O}(h)$, since the remaining error or the Taylor series is divided by h^d to obtain $y_k^{(d)}$. The accuracy of the estimate of a derivative can be increased simply by taking more past samples into consideration, and enough equations to solve for the desired derivative, at the cost of a higher lag. Below is an example of the necessary equations for a third order derivative with an error of $\mathcal{O}(h^2)$:

$$\left.\begin{aligned} \tfrac{5}{12} \times \left[y_k &= y_k \right] \\ -\tfrac{9}{6} \times \left[y_{k-1} &= y_k - h y_k^{(1)} + \tfrac{1}{2} h^2 y_k^{(2)} - \tfrac{1}{6} h^3 y_k^{(3)} + \tfrac{1}{24} h^4 y_k^{(4)} + \mathcal{O}(h^5) \right] \\ 2 \times \left[y_{k-2} &= y_k - 2h y_k^{(1)} + 2h^2 y_k^{(2)} - \tfrac{4}{3} h^3 y^{(3)} + \tfrac{2}{3} h^4 y_k^{(4)} + \mathcal{O}(h^5) \right] \\ -\tfrac{7}{6} \times \left[y_{k-3} &= y_k - 3h y_k^{(1)} + \tfrac{9}{2} h^2 y_k^{(2)} - \tfrac{9}{2} h^3 y_k^{(3)} + \tfrac{27}{8} h^4 y_k^{(4)} + \mathcal{O}(h^5) \right] \\ \tfrac{1}{4} \times \left[y_{k-4} &= y_k - 4h y_k^{(1)} + 8h^2 y_k^{(2)} - \tfrac{32}{3} h^3 y_k^{(3)} + \tfrac{32}{3} h^4 y_k^{(4)} + \mathcal{O}(h^5) \right] \end{aligned}\right\} \begin{aligned} &\tfrac{1}{4} y_{k-4} - \tfrac{7}{6} y_{k-3} + 2y_{k-2} \\ &\quad - \tfrac{9}{6} y_{k-1} + \tfrac{5}{12} y_k \\ &\quad = \tfrac{1}{6} h^3 y_k^{(3)} + \mathcal{O}(h^5) \end{aligned}$$

And the resulting approximation for the third derivative with an error in $\mathcal{O}(h^2)$ is in equation (4.5).

$$y_k^{(3)} = \frac{1}{h^3}\left(\frac{3}{2} y_{k-4} - 7y_{k-3} + 12y_{k-2} - 9y_{k-1} + \frac{5}{2} y_k\right) + \mathcal{O}(h^2) \tag{4.5}$$

Higher order derivatives The examples above illustrate how a linear combination of Taylor series can result in an equation that can be solved for a derivative $y_k^{(d)}$. For higher-order derivatives and smaller errors, the process becomes tedious. Fortunately, due to the structure in a Taylor expansion, the process can be generalised. From the above example it is clear that all that needs to be solved for are the coefficients c_j with which the y_{k+j} should be multiplied. The resulting linear combination of y_{k+j} only needs to be divided by h^d to obtain $y_k^{(d)}$. The procedure for computing this derivative with an error $\mathcal{O}(h^o)$ in the case of *backward differences* is as follows [7]:

1. Find the linear combination of the Taylor expansion for y_k to $y_{k-(d+o-1)}$ that include terms with y_k to $y_k^{(d+o-1)}$ that eliminates all $y_k^{(i)}$ but $y_k^{(d)}$

To eliminate the $y_k^{(i)}$ for $i \neq d$ and find the coefficients c_j for the y_{k+j}, the system of equations in equation (4.6) is to be solved for the coefficients c_j:

$$\underbrace{\sum_{i=0}^{d+o-1}}_{equations} \underbrace{\sum_{j=-(d+o-1)}^{0}}_{coefficients} j^i c_j = \begin{cases} 0, & i \neq d \\ 1, & i = d \end{cases} \tag{4.6}$$

2. Divide by $\frac{1}{d!}h^d$ to isolate $y_k^{(d)}$

Given the c_j, equation (4.7) can be assembled, from which equation (4.8) can be obtained. This is the final expression for the approximation of $y_k^{(d)}$. Obviously, the equations hold for $d > 0$ since there is no need to approximate $y_k^{(0)}$.

$$\frac{1}{d!}h^d y_k^{(d)} = \sum_{j=-(d+o-1)}^{0} c_j\, y_{k+j} \quad + \mathcal{O}\left(h^{d+o}\right), \quad d > 0 \tag{4.7}$$

$$y_k^{(d)} = \frac{1}{h^d} d! \sum_{j=-(d+o-1)}^{0} c_j\, y_{k+j} \quad + \mathcal{O}(h^o), \quad d > 0 \tag{4.8}$$

Forward and central differences The above derivations and equations are mostly written in a very general fashion in terms of the sample-shift number j, but all examples and equations are written regarding backward differences: The approximations of derivatives $y_k^{(d)}$ consist of past values of the signal: samples occurring before y_k. However, with a derivation completely analogous to the above, one can construct approximations for the derivatives by means of forward or central differences: using future values of the signal, or an equal number of past and future values. Rewriting equations (4.6) and (4.8) to a more general form yields equations (4.9) and (4.10).

$$\sum_{i=0}^{d+o-1} \sum_{j=j_{min}}^{j_{max}} j^i c_j = \begin{cases} 0, & i \neq d \\ 1, & i = d \end{cases} \tag{4.9}$$

$$y_k^{(d)} = \frac{1}{h^d} d! \sum_{j=j_{min}}^{j_{max}} c_j\, y_{k+j} \quad + \mathcal{O}(h^o), \quad d > 0 \tag{4.10}$$

The required values of j_{min} and j_{max} are summarised in table 4.1 for backward, central and forward differences. The forward and central difference method are valid methods for derivative approximation in post-processing scenarios, since they require future values. Therefore, only the backward difference method is causal and applicable in real-time simulations and is of most interest in this research.

4.1.3. Matrix equations

With the means to compute an approximation for the derivative of a signal given past values of said signal (equation (4.10)), it becomes straightforward to create a matrix equation that yields the generalised

Table 4.1: Summation ranges in equations (4.9) and (4.10) to find approximations for derivatives $y_k^{(d)}$ by way of finite differences. *for central differences, o must be chosen such that $d + o$ is odd.

	backward	central*	forward
j_{min}	$-(d+o-1)$	$-\frac{1}{2}(d+o-1)$	0
j_{max}	0	$\frac{1}{2}(d+o-1)$	$(d+o-1)$

coordinates given a so-called time-series (array of past (and/or future) values) in the active inference framework. The objective is to obtain the generalised coordinates of motion of the output (sensory input) $\tilde{\boldsymbol{y}}$. For a one-dimensional signal, that is:

$$\tilde{\boldsymbol{y}} = \left[y_k^{(0)} \quad y_k^{(1)} \ldots\ldots y_k^{(p)}\right]^T$$

An approximation of this generalised array is found by applying equation (4.11). The range of required samples k_{min} to k_{max} is as j_{min} and j_{max} in table 4.1, with d replaced by p, since the embedding order p represents the highest order derivative in the generalised coordinates, and it is this derivative that requires the most samples to approximate.

$$\tilde{\boldsymbol{y}}_k = E\,\check{\boldsymbol{y}}_k, \qquad \check{\boldsymbol{y}}_k = \left[y_{k_{min}} \ldots\ldots y_{k_{max}}\right]^T \tag{4.11}$$

An example of equation (4.11) for an embedding order $p = 5$, an approximation error of $\mathcal{O}(h)$ and for a $q = 1$ dimensional signal y by backward differences is shown below. In case of a q-dim signal $\boldsymbol{y}$, E is simply replaced by the Kronecker product $E \otimes I_q$ and the arrays for $\tilde{\boldsymbol{y}}$ and $\check{\boldsymbol{y}}$ will be extended, since each entry will become a $q \times 1$-array by itself.

$$\begin{bmatrix} y_k^{(0)} \\ y_k^{(1)} \\ y_k^{(2)} \\ y_k^{(3)} \\ y_k^{(4)} \\ y_k^{(5)} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -\frac{1}{h} & \frac{1}{h} \\ 0 & 0 & 0 & \frac{1}{h^2} & -\frac{2}{h^2} & \frac{1}{h^2} \\ 0 & 0 & -\frac{1}{h^3} & \frac{3}{h^3} & -\frac{3}{h^3} & \frac{1}{h^3} \\ 0 & \frac{1}{h^4} & -\frac{4}{h^4} & \frac{6}{h^4} & -\frac{4}{h^4} & \frac{1}{h^4} \\ -\frac{1}{h^5} & \frac{5}{h^5} & -\frac{10}{h^5} & \frac{10}{h^5} & -\frac{5}{h^5} & \frac{1}{h^5} \end{bmatrix} \begin{bmatrix} y_{k-5} \\ y_{k-4} \\ y_{k-3} \\ y_{k-2} \\ y_{k-1} \\ y_k \end{bmatrix}$$

Such an E-matrix can be created in Matlab, for which code that defines the function `f_finitediffmat` is displayed in listing 4.1 and the complete script with explanatory comments regarding the in- and outputs is presented in listing B.3 in appendix B.2.

Listing 4.1: A Matlab function `f_finitediffmat` that creates the matrix for finite differences E as in equation (4.11). The full script with an explanation of inputs and outputs of the function is provided in listing B.3 in appendix B.2.

```
1  function E = f_finitediffmat(dt,p,o,n,method)
2
3  pp = p+1; % number of rows in the matrix E for a 1-dim signal
4  switch method
5  % s: # samples required for the approx. of all desired derivatives
6  % E1: preallocation of matrix E for a 1-dim signal
7  % E1(,)=1: prepare first row of E1 to pass y onto itself in y_
8      case 'f'
9          s = p+o;
10         if p==0; E1 = 1;
11         else; E1 = zeros(pp,s); E1(1,1) = 1;
12         end
13     case 'c'
```

```
        if mod(p+o,2) == 0 % when p+o is even, o is increased by 1
            s = p+o+1;
        else
            s = p+o;
        end
        if p==0; E1 = 1;
        else; E1 = zeros(pp,s); E1(1,ceil(s/2)) = 1;
        end
    case 'b'
        s = p+o;
        if p==0; E1 = 1;
        else; E1 = zeros(pp,s); E1(1,end) = 1;
        end
end
C = zeros(1,s); % preallocation of array w/ coef for finite differences

for d = 1:p % we visit all p-values so we have all the derivatives
    switch method
    % sd: # samples required for the current derivative
    % jmin, jmax: required finite differences around the point of interest
    % imax: total number of samples (-1) required for the approximation
    % sumij: preallocation of the matrix for computation of C
    % sumijC: array with outcomes of the sums in sumij
        case 'f'
            sd = d+o;
            jmin = 0; jmax = sd-1;
            imax = sd-1;
            sumij = zeros(sd,sd);
            sumijC = zeros(size(sumij,2),1);
        case 'c'
            if mod(d+o,2) == 0 % when d+o is even, o is increased by 1
                sd = d+o+1;
            else
                sd = d+o;
            end
            jmax = (sd-1)/2; jmin = -jmax;
            imax = sd-1;
            sumij = zeros(sd,sd);
            sumijC = zeros(size(sumij,2),1);
        case 'b'
            sd = d+o;
            jmin = -(sd-1); jmax = 0;
            imax = sd-1;
            sumij = zeros(sd,sd);
            sumijC = zeros(size(sumij,2),1);
    end
    sumijC(d+1) = 1; % the sum must be 1 for j = d, 0 otherwise
    jrange = jmin:jmax; % range of all the finite difference elements
    for i = 0:imax % filling the matrix w/ elements to sum
        sumij(i+1,:) = jrange.^i;
    end
    switch method % computing C (solving the linear system)
        case 'f'
            C(1:sd) = (sumij\sumijC)';
        case 'c'
            C((s-sd)/2+1:s-(s-sd)/2) = (sumij\sumijC)';
```

```
        case 'b'
            C(s-sd+1:end) = (sumij\sumijC)';
    end
    E1(d+1,:) = (factorial(d)/dt^d).*C; % adding the elements to E
end

E = kron(E1,eye(n)); % E for an n-dim signal
```

In a simulation scenario, the generalised coordinate $\tilde{\boldsymbol{y}}$ must be computed for every k in the timespan of the simulation, which means the time-series array $\check{\boldsymbol{y}}_k$ shifts one sample-set every time, and the information has great overlap. Only the information in $\boldsymbol{y}_k$ is new, so it is therefore possible to use the already computed generalised coordinates $\tilde{\boldsymbol{y}}_{k-1}$ augmented with $\boldsymbol{y}_k$ but missing the highest derivative in $\tilde{\boldsymbol{y}}_{k-1}$, instead of $\check{\boldsymbol{y}}_k$. The matrix E from equation (4.11) has to be rewritten into the matrix Q as in equation (4.12) with an example shown below, in which the embedding order, approximation order and signal dimension are the same as in the previous example ($p = 5$, $o = 1$, $q = 1$).

$$\tilde{\boldsymbol{y}}_k = Q\,\check{\tilde{\boldsymbol{y}}}_k, \qquad \check{\tilde{\boldsymbol{y}}}_k = \begin{bmatrix} \boldsymbol{y}_k & \tilde{\boldsymbol{y}}_{k-1}^T \end{bmatrix}^T \tag{4.12}$$

$$\begin{bmatrix} y_k^{(0)} \\ y_k^{(1)} \\ y_k^{(2)} \\ y_k^{(3)} \\ y_k^{(4)} \\ y_k^{(5)} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{h} & -\frac{1}{h} & 0 & 0 & 0 & 0 \\ \frac{1}{h^2} & -\frac{1}{h^2} & -\frac{1}{h} & 0 & 0 & 0 \\ \frac{1}{h^3} & -\frac{1}{h^3} & -\frac{1}{h^2} & -\frac{1}{h} & 0 & 0 \\ \frac{1}{h^4} & -\frac{1}{h^4} & -\frac{1}{h^3} & -\frac{1}{h^2} & -\frac{1}{h} & 0 \\ \frac{1}{h^5} & -\frac{1}{h^5} & -\frac{1}{h^4} & -\frac{1}{h^3} & -\frac{1}{h^2} & -\frac{1}{h} \end{bmatrix} \begin{bmatrix} y_k \\ y_{k-1}^{(0)} \\ y_{k-1}^{(1)} \\ y_{k-1}^{(2)} \\ y_{k-1}^{(3)} \\ y_{k-1}^{(4)} \end{bmatrix}$$

For this form, the generalised output $\tilde{\boldsymbol{y}}$ needs to be initialized. This form will not be applied in this research.

4.2. Analytic evaluation

To gain insight into the effect of computing derivatives by means of finite differences, this section focuses on some tests to reveal the performance and show what happens in the presence of noise. For proper testing, a ground truth must be established and therefore the analytic function in equation (4.13) is chosen of which the derivatives are known. This function is then sampled to obtain 'data'.

$$y(t) = \sin t + 3\cos\frac{1}{10}t + \frac{1}{100}t^3 \tag{4.13}$$

All the test results reported in this section can be reproduced with the Matlab script of listing B.5 in appendix B.2.

4.2.1. Derivative accuracy

Approximated derivatives by means of backward, central and forward differences are compared with the ground truth in figure 4.1. The sampling time h (or Δt) is 10^{-2}s and derivatives have been computed with an accuracy of $\mathcal{O}(h)$ ($o = 1$). With this sampling time and accuracy, some error is still visible in the plots. From the close up plots (figures 4.1b to 4.1d) it can be seen that the approximations are extremely close to the ground truth. Approximations by backward differences will always underestimate an increasing derivative and overestimate a decreasing derivative. The opposite is true for approximations by forward differences. The higher the order of the derivative, the more samples are required. The close-up plots show how approximations by backwards differences can only be computed after enough samples are available, and that approximations by forward differences cannot be estimated for the final samples. Both of these 'problems' are present for central differences. Table 4.2 shows the mean squared error (MSE) of the approximated derivatives. The samples for which no derivative could be computed are excluded from this MSE. For these samples, the approximated derivative is set equal

Table 4.2: MSE of the approximated derivatives in figure 4.1 for backward, forward and central differences. Sampling time $h = 10^{-2}$s, time $T = 10$s and accuracy is of $\mathcal{O}(h)$.

derivative	backward $\times 10^{-3}$	forward $\times 10^{-3}$	central $\times 10^{-8}$
1	0.0124	0.0125	0.0147
2	0.0530	0.0530	0.0033
3	0.1082	0.1083	0.0324
4	0.2077	0.2075	0.0133
5	0.3000	0.3008	0.2396
6	0.5596	0.5444	8975.8

to 0. The error contribution of these samples depends on the actual size of the derivatives themselves and their contribution to the MSE varies with the total sampling time. Including them therefore makes for an unfair comparison. The data in table 4.2 shows that the MSE of the derivative estimation is fairly small and most noticeably, the MSE for backward and forward differences is extremely similar. For central differences, the error is clearly significantly smaller. With increasing order of derivative the accuracy decreases, which is all the more clear for central differences. For central differences, however, the MSE does not increase with every increasing derivative order. This has to do with the symmetry of central differences, which requires some derivatives to be approximated with more samples since the total number of samples must always be odd. For derivatives up to order 6, the MSE remains small for all approximation methods.

Table 4.3: MSE of the approximated derivatives for backward, forward and central differences. Sampling time $h = 10^{-2}$s, time $T = 10$s and accuracy is of $\mathcal{O}(h^2)$.

derivative	backward $\times 10^{-6}$	forward $\times 10^{-6}$	central $\times 10^{-8}$
1	0.0006	0.0006	0.0147
2	0.0040	0.0041	0.0000
3	0.0159	0.0159	0.0324
4	0.0389	0.0387	0.0000
5	0.3138	0.1618	0.2396
6	9122.1	4181.3	1.8908×10^6

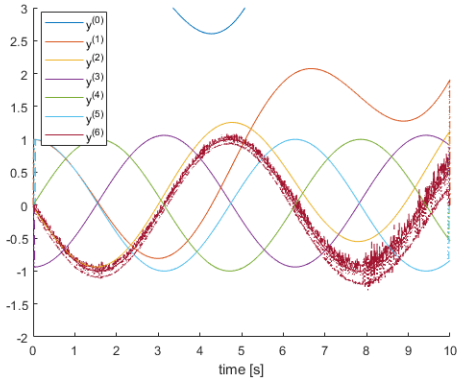
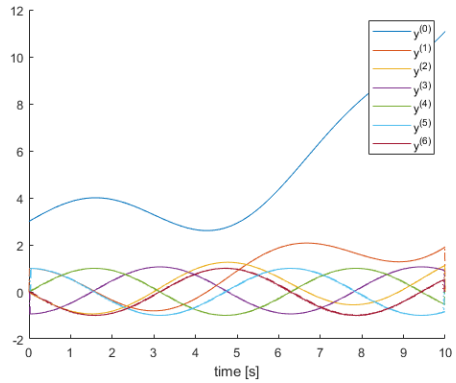
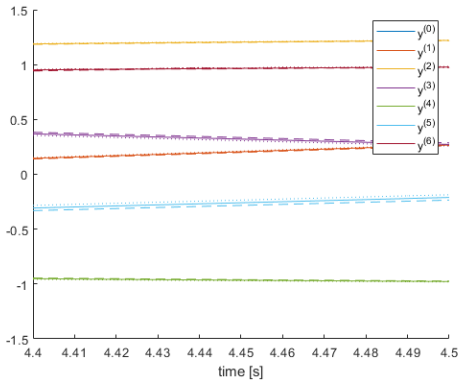


Figure 4.2: Derivative estimates up to 6th order of equation (4.13), sampled with sampling time $h = 10^{-2}$s and accuracy of $\mathcal{O}(h^2)$ ($o = 2$). Solid lines represent true analytic derivatives. Approximations by means of backward differences are shown dashed, forward differences dotted, and central differences dashdotted.

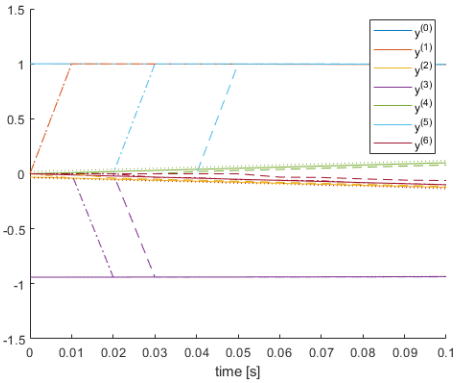
To analyse the effect of the number of samples on accuracy, table 4.3 shows the MSE for an error in $\mathcal{O}(h^2)$, for which the plots are shown in figure 4.2. These results are very interesting, showing that for the first few derivatives, the accuracy has increased. However, as the order of derivative increases, the MSE grows much faster, as is also very clear from figure 4.1b. Using more samples than necessary is therefore inadvisable. Lastly, the effect of the sampling time (h) shall be studied. With an error of $\mathcal{O}(h)$ and a sampling time 10 times smaller than before ($h = 10^{-3}$s), the MSE for signals sampled for 10s is displayed in table 4.4. Comparing this table to table 4.2 it is clear that for lower order derivatives, the accuracy is higher. However, as the order of derivative increases, the error increases very fast, growing very large for 5$^{\text{th}}$ and 6$^{\text{th}}$ derivatives. These derivative approximations are extremely 'noisy'. Unfortunately, this is a culprit of derivative estimation. In the next



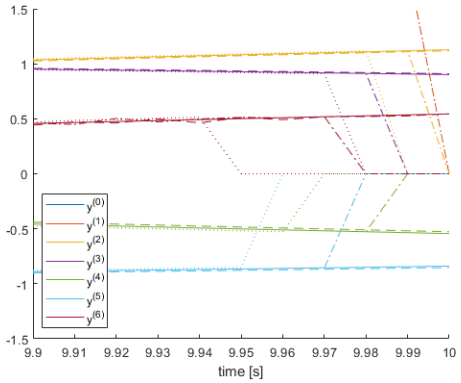
(a) The function and its derivatives



(b) Close-up showing accuracy



(c) Close up of first samples



(d) Close up of the final samples

Figure 4.1: Derivative estimates up to 6th order of equation (4.13), with sampling time $h = 10^{-2}$s and accuracy of $\mathcal{O}(h)$ ($o = 1$). Solid lines represent true analytic derivatives. Approximations by means of backward differences are shown dashed, forward differences dotted, and central differences dashdotted.

section, the effects of noise present in the original signal will be evaluated.

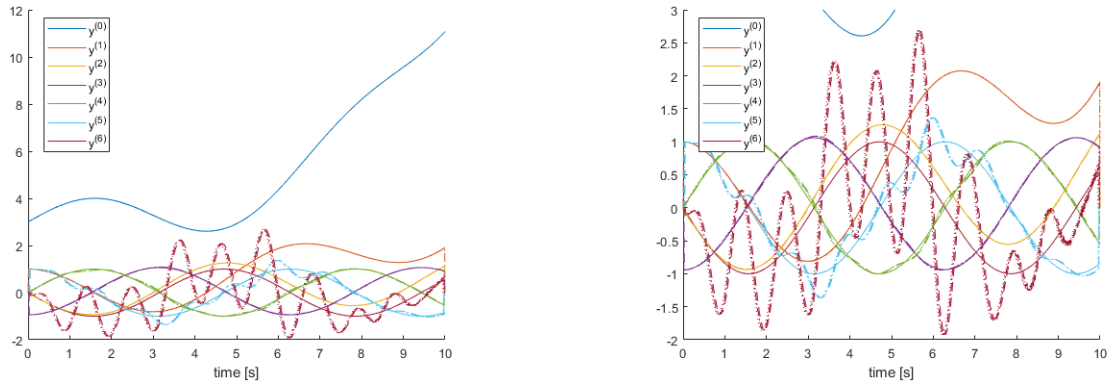
4.2.2. Derivatives and noise

An important part of this research is the presence of coloured noise. For white noise it is well known that numerically approximating derivatives amplifies noise to the extremities. Coloured noise has faster dynamics than the dynamic processes it influences, but it is smooth, in contrast to white noise. The influence of coloured noise on the approximation of derivatives by means of finite differences is explored in this section.

Figure 4.3 shows approximated derivatives for equation (4.13) with added noise. The noise has a standard deviation $\sigma_w = 1 \times 10^{-3}$ and a smoothness $s_w = 0.5$. For the noisy $y(t)$, no ground truth can be established. Therefore, the reference signals are the analytic derivatives of the noise-less signal. This also better shows the effect of noise on the derivative approximation. No MSE is computed for these approximations, since for a fair evaluation, these must be the MSE of the noisy ground truth and the noisy approximations. The former, however, is not available. It is very clear from figure 4.3 that noise adds discrepancies. From the 4$^{\text{th}}$ derivative, this really becomes visible. It is important to keep in mind that the true derivative of the noise signal also does not equal the reference. The noise does increase vastly with every increasing derivative order. Lastly, figures 4.4 and 4.5 show the effects of decreased smoothness or increased variance. The plots in figure 4.4 show fewer derivatives because the increase due to noise starts early (with lower order derivatives) and increases with every order. Showing the higher order derivatives only makes for a more cluttered plot. It is very clear that the performance of derivative approximation by means of finite differences is sensitive to the characteristics of the coloured noise that is present. The sensitivity to the smoothness of the noise appears to be far greater than the sensitivity to the variance. This is well accounted for in the Free Energy Principle, for

Table 4.4: MSE of the approximated derivatives for backward, forward and central differences. Sampling time $h = 10^{-3}$s, time $T = 10$s and accuracy is of $\mathcal{O}(h)$.

derivative	backward $\times 10^{-4}$	forward $\times 10^{-4}$	central $\times 10^{-8}$
1	0.0012	0.0012	0.0000
2	0.0053	0.0053	0.0000
3	0.0107	0.0107	0.0000
4	0.3794	0.1151	0.2483
5	50.1636	50.2149	110 670
6	8.95×10^{11}	1.84×10^{12}	2.4864×10^{12}



(a) Full-scale plot

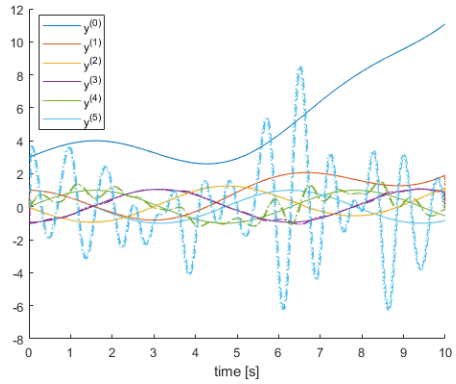
(b) Scaled plot for better derivative visibility

Figure 4.3: Derivative estimates up to 6th order of equation (4.13) with added noise, sampled with sampling time $h = 10^{-2}$s and accuracy of $\mathcal{O}(h)$ ($o = 1$). Noise characteristics (section 3.1.1) $\sigma_w = 1 \times 10^{-3}$, $s_w = 0.5$. Solid lines represent true analytic derivatives (without noise). Approximations by means of backward differences are shown dashed, forward differences dotted, and central differences dashdotted.

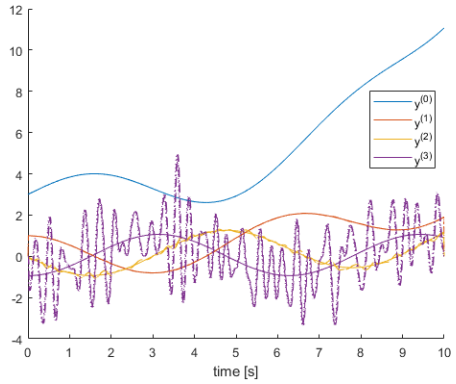
the precision matrices that weigh the errors based on the coloured noise properties have a increase steeper with the smoothness than with the variance, as can be seen in equation (3.15), chapter 3. Although the large fluctuations of approximations to higher order derivatives under the influence of noise seem troublesome, most important is to learn how this affects the Active Inference algorithm. This will become more clear in chapter 5.

4.3. Generalised coordinates from finite differences

Sections 4.1 and 4.2 have shown that by means of finite differences, samples from the sensory input can be used to obtain approximations of derivatives of the perceived signal. As such, this chapter has provided an answer to the research question ‘How can generalised coordinates be generally applied in the Active Inference framework when they are not readily available?’. The performance of the method of finite differences under the presence of noise largely depends on the required embedding order (number of derivatives) and on the characteristics of the noise. How this influences the Active Inference algorithm is discussed in chapter 5. Filtering methods might be a good addition to the method of finite differences to refine the approximated derivatives, but that is outside of the scope of this work.

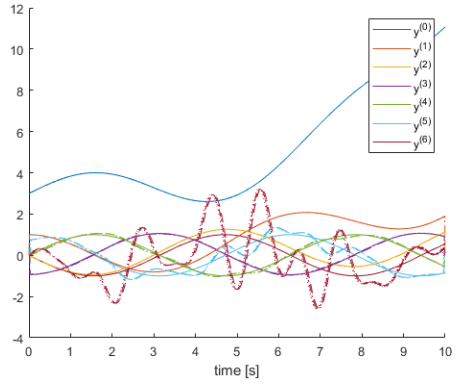


(a) $s_w = 0.3$

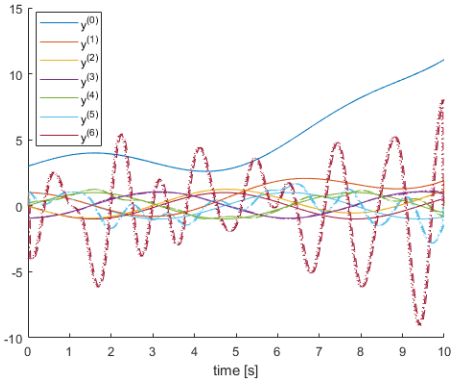


(b) $s_w = 0.1$

Figure 4.4: Derivative estimates of equation (4.13) with added noise, sampled with sampling time $h = 10^{-2}$s and accuracy of $\mathcal{O}(h)$ ($o = 1$). Noise characteristics (section 3.1.1) $\sigma_w = 1 \times 10^{-3}$, and varying smoothness. Solid lines represent true analytic derivatives (without noise). Approximations by means of backward differences are shown dashed, forward differences dotted, and central differences dashdotted.



(a) $\sigma_w = 2 \times 10^{-3}$



(b) $\sigma_w = 5 \times 10^{-3}$

Figure 4.5: Derivative estimates of equation (4.13) with added noise, sampled with sampling time $h = 10^{-2}$s and accuracy of $\mathcal{O}(h)$ ($o = 1$). Noise characteristics (section 3.1.1) are varying standard deviation, and $s_w = 0.5$. Solid lines represent true analytic derivatives (without noise). Approximations by means of backward differences are shown dashed, forward differences dotted, and central differences dashdotted.

5

Active Inference and State Space formulation

In the preceding chapters, the topic of the Free Energy Principle and Active Inference has been explored, as well as some necessary constructs to apply this neuroscientific theory to robotics. The aim of this research is to apply Active Inference to an LTI-State Space control loop whilst taking the aforementioned constructs into account. A State Space description of Active Inference has already been proposed [20], but it lacks generalised output coordinates. To combine generalised coordinates with the precision matrices from chapter 3 that preserve the correlation between dynamical orders, the State Space formulation needs to be researched. This is the topic of sections 5.1 and 5.2. This chapter also provides closed-loop simulations for a simple system with one degree-of-freedom, described in section 5.3, to evaluate the performance in the presence of the applied constructs. For this evaluation, multiple simulations are performed in sections 5.4 and 5.5, altogether providing an answer to the question 'How can Active Inference with generalised motions be applied to an LTI-State Space control loop?'

5.1. Closed loop State Space formulation

Minimization of the Free Energy formulation provided in chapter 2 is what drives an Active Inference agent to update its internal beliefs based on its perception and impose action on the environment to make the environment conform with its internal beliefs. Consider a process in the environment that conforms to an LTI-State Space model, as in equation (5.1).

$$\begin{aligned} \dot{\boldsymbol{x}} &= A\boldsymbol{x} + B\boldsymbol{u} \quad + \boldsymbol{w} \\ \boldsymbol{y} &= C\boldsymbol{x} \qquad\quad + \boldsymbol{z} \end{aligned} \tag{5.1}$$

In equation (5.1), $\boldsymbol{x}$ are the environment states, $\boldsymbol{u}$ are the inputs to the environment (actions) and $\boldsymbol{w}$ is a disturbance in the environment. The observed output $\boldsymbol{y}$ is disturbed by the measurement noise $\boldsymbol{z}$. In this section and section 5.2, a general State Space formulation will be adopted. In section 5.3, the State Space model in equation (5.1) will represent a one-DOF SISO system which is used in simulations.

5.1.1. The agent's internal model

The equations in equation (5.1) are a model description for deterministic states $\boldsymbol{x}$. An Active Inference agent, however, considers a Gaussian recognition density $q(\boldsymbol{x})$ and it beliefs the environment state $\boldsymbol{x}$ to be the Gaussian mean $\boldsymbol{\mu} = E[\boldsymbol{x}]$, which is the quantity the agent keeps track of, and as explained in section 2.2.1. Furthermore, the agent considers generalised coordinates meaning it does not only track its belief $\boldsymbol{\mu}$, but the generalised $\tilde{\boldsymbol{\mu}}$ (section 2.3):

$$\tilde{\boldsymbol{\mu}} = \begin{bmatrix} \boldsymbol{\mu}^T & \boldsymbol{\mu}'^T & \boldsymbol{\mu}''^T & \cdots\cdots\cdots & \boldsymbol{\mu}^{(p)T} \end{bmatrix}^T$$

where p is the embedding order. Similarly, it expects to observe $\tilde{\boldsymbol{y}}$, which can be obtained by means of finite differences as explained in chapter 4. Although the environment is modelled as in equation (5.1),

the internal model of the agent is represented in generalised coordinates and as a function of the belief of environment state $\tilde{\boldsymbol{\mu}}$ rather than the actual (hidden) state $\boldsymbol{x}$. Furthermore, the control input term ($B\boldsymbol{u}$, equation (5.1)) is represented by a prior variable $\tilde{\boldsymbol{\xi}}$ [20]. This prior variable allows the embedding of a reference, for example for tracking, in the agent's internal model, and allows the controller designer to influence the prior model $p(\boldsymbol{\mu})$, which is the model of evolution (and behaviour) of states in the environment.

As a result, the generalised internal model of an agent as defined in equation (2.15) in the case of an LTI State Space system is as in equation (5.2). With this model and the Free Energy formulation chapter 2, controller and filter equations can be derived, as shown in section 5.1.2.

$$\begin{matrix} \mathcal{D}\tilde{\boldsymbol{\mu}} = \tilde{A}\tilde{\boldsymbol{\mu}} + \tilde{\boldsymbol{\xi}} & + \tilde{\boldsymbol{\varepsilon}}_\mu \\ \tilde{\boldsymbol{y}} = \tilde{C}\tilde{\boldsymbol{\mu}} & + \tilde{\boldsymbol{\varepsilon}}_y \end{matrix} \quad \text{with} \quad \begin{matrix} \tilde{A} = I_{p+1} \otimes A \\ \tilde{C} = I_{p+1} \otimes C \end{matrix} \quad \text{and} \quad \begin{matrix} \tilde{\boldsymbol{\varepsilon}}_\mu = \left(\mathcal{D} - \tilde{A}\right)\tilde{\boldsymbol{\mu}} - \tilde{\boldsymbol{\xi}} \\ \tilde{\boldsymbol{\varepsilon}}_y = \tilde{\boldsymbol{y}} - \tilde{C}\tilde{\boldsymbol{\mu}} \end{matrix} \tag{5.2}$$

5.1.2. Perception and action

The practical form of Free Energy is a quadratic function that can be minimized by means of a gradient descent (chapter 2 and equation (2.16)). The Free Energy unifies action and perception, depending on both the error of state dynamics and the perception error. To minimize the Free Energy, the agent can change its perception by updating its belief about the perceived states in the environment. This is represented by a filtering equation. Since the Free Energy is twofold, an agent can also act on its environment (providing input), thereby changing the environment and making it comply better with its beliefs, thus also minimizing Free Energy. The Free Energy formulation is repeated from equation (2.16) in equation (5.3).

$$\mathcal{F}(\tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{y}}) = \frac{1}{2}\left(\tilde{\boldsymbol{\varepsilon}}_\mu^T \tilde{\Pi}_w \tilde{\boldsymbol{\varepsilon}}_\mu + \tilde{\boldsymbol{\varepsilon}}_y^T \tilde{\Pi}_z \tilde{\boldsymbol{\varepsilon}}_y\right) \tag{5.3}$$

Perception or filtering is the minimization of the Free Energy w.r.t. the belief of motion $\tilde{\boldsymbol{\mu}}$ by means of a gradient descent. Equation (5.4) shows the perception update rule. It includes α_μ, the learning rate of the gradient descent algorithm, as a tuning parameter. The gradient descent is offset by $\mathcal{D}\tilde{\boldsymbol{\mu}}$, which is required to maintain the belief of motion. When this offset is not introduced, the vanishing of the Free Energy causes the belief of motion to vanish as well. The dependencies in equation (5.4), following a Jacobian notation convention for derivatives, show that perception, or the update of the belief of motion in the environment, is weighed by the precision of both the process and the sensory input (measurement), and that is influenced by prior beliefs, current belief of motion and the perceived sensory input.

$$\begin{aligned} \dot{\tilde{\boldsymbol{\mu}}} &= \mathcal{D}\tilde{\boldsymbol{\mu}} - \alpha_\mu \left(\frac{\partial \mathcal{F}(\tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{y}})}{\partial \tilde{\boldsymbol{\mu}}}\right)^T \\ &= \mathcal{D}\tilde{\boldsymbol{\mu}} - \alpha_\mu \left(\frac{\partial \mathcal{F}}{\partial \tilde{\boldsymbol{\varepsilon}}_\mu}\frac{\partial \tilde{\boldsymbol{\varepsilon}}_\mu}{\partial \tilde{\boldsymbol{\mu}}} + \frac{\partial \mathcal{F}}{\partial \tilde{\boldsymbol{\varepsilon}}_y}\frac{\partial \tilde{\boldsymbol{\varepsilon}}_y}{\partial \tilde{\boldsymbol{\mu}}}\right)^T \\ &= \mathcal{D}\tilde{\boldsymbol{\mu}} - \alpha_\mu \left(\left(\mathcal{D} - \tilde{A}\right)^T \left(\tilde{\boldsymbol{\varepsilon}}_\mu^T \tilde{\Pi}_w\right)^T - \tilde{C}^T \left(\tilde{\boldsymbol{\varepsilon}}_y^T \tilde{\Pi}_z\right)^T\right) \\ &= \mathcal{D}\tilde{\boldsymbol{\mu}} - \alpha_\mu \left(\left(\mathcal{D} - \tilde{A}\right)^T \tilde{\Pi}_w \left(\left(\mathcal{D} - \tilde{A}\right)\tilde{\boldsymbol{\mu}} - \tilde{\boldsymbol{\xi}}\right) - \tilde{C}^T \tilde{\Pi}_z^T \left(\tilde{\boldsymbol{y}} - \tilde{C}\tilde{\boldsymbol{\mu}}\right)\right) \end{aligned} \tag{5.4}$$

Action or the generation of input to the environment also follows from the minimization of the Free Energy, but w.r.t. the action or control input $\tilde{\boldsymbol{u}}$. This control input does not appear directly in the Free Energy formulation. It does, however, indirectly depend on input by way of the sensory input $\tilde{\boldsymbol{y}}$. These are related by means of the forward model: $\tilde{\boldsymbol{y}} = \tilde{G}\tilde{\boldsymbol{u}}$, which is the topic of section 5.2. The action update rule is given in equation (5.5), also adhering to a Jacobian notation convention, and also includes a gradient descent learning rate α_u. Action is weighed only by the precision of sensory input, depends on current belief of motion and sensory input and is not affected by the prior.

$$\begin{aligned} \dot{\tilde{\boldsymbol{u}}} &= -\alpha_u \left(\frac{\partial \mathcal{F}(\tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{y}})}{\partial \tilde{\boldsymbol{u}}}\right)^T \\ &= -\alpha_u \left(\frac{\partial \mathcal{F}}{\partial \tilde{\boldsymbol{\varepsilon}}_y}\frac{\partial \tilde{\boldsymbol{\varepsilon}}_y}{\partial \tilde{\boldsymbol{y}}}\frac{\partial \tilde{\boldsymbol{y}}}{\partial \tilde{\boldsymbol{u}}}\right)^T \end{aligned}$$

$$= -\alpha_u\left(\tilde{G}^T I \tilde{\boldsymbol{\varepsilon}}_y^T \tilde{\Pi}_z\right)$$
$$= -\alpha_u \tilde{G}^T \tilde{\Pi}_z\left(\tilde{\boldsymbol{y}} - \tilde{C}\tilde{\boldsymbol{\mu}}\right) \tag{5.5}$$

With equations for perception and action (equations (5.4) and (5.5), filter and controller), closed-loop

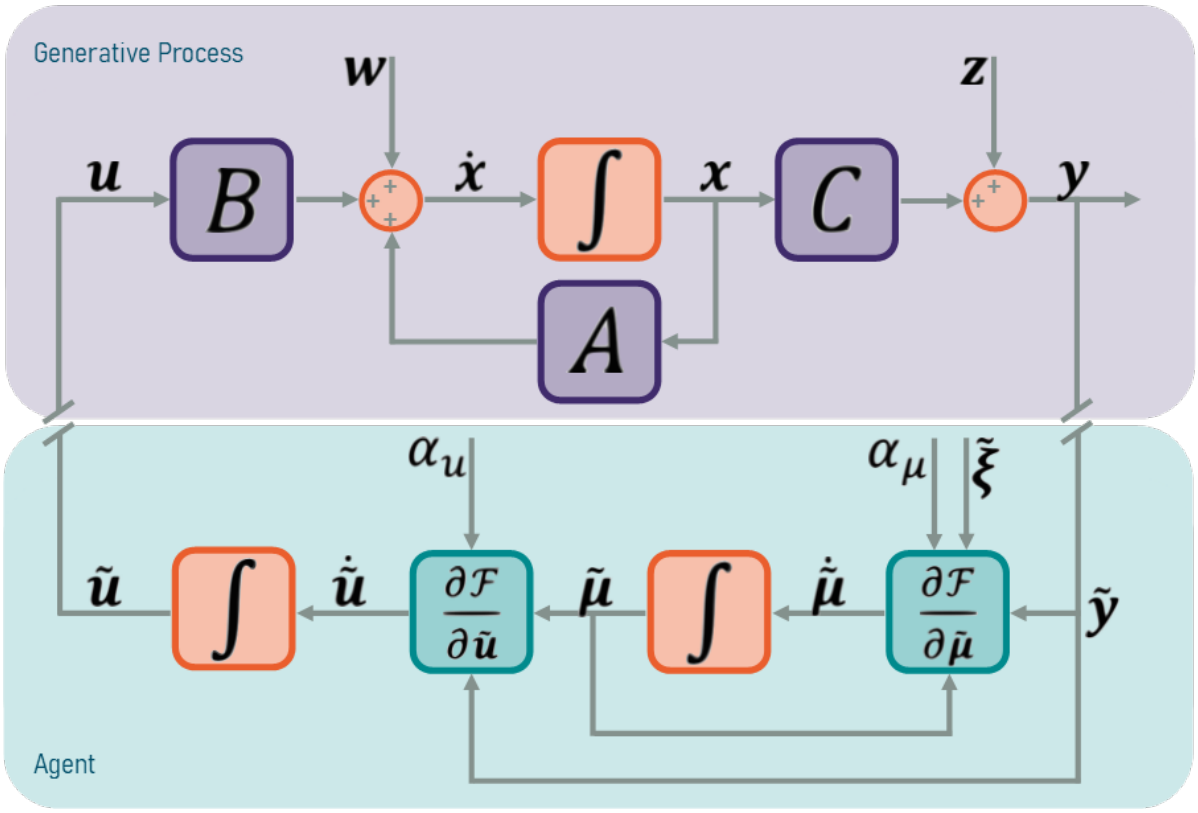


Figure 5.1: Block scheme of the LTI-State Space control loop consisting of the generative process (plant) and agent (controller/observer).

control can be implemented. A block scheme of an Active Inference agent in a closed loop with the generative process (environment or plant) is displayed in figure 5.1. There is a discrepancy between the generative process, or environment, and the agent with regards to generalised coordinates. This is addressed in section 5.4.

5.2. Generalised forward model

In order to write an expression for the partial derivative of the free energy with respect to the input, as in equation (5.5), the expression of $\tilde{\boldsymbol{y}}$ as a function of $\tilde{\boldsymbol{u}}$ is required, since the Free Energy only depends on $\tilde{\boldsymbol{u}}$ via $\tilde{\boldsymbol{y}}$, and not directly. In an LTI-State Space formulation, like equation (5.1) without the noises $\boldsymbol{w}$ and $\boldsymbol{z}$, this relationship is generally known as the forward model G:

$$\boldsymbol{y} = G\boldsymbol{u}, \quad G = -CA^{-1}B$$

This forward model can be obtained by assuming steady-state, or $\dot{\boldsymbol{x}} = 0$. In this scenario, $\boldsymbol{y}$ can be solved for $\boldsymbol{u}$. If a generalised forward model to relate $\tilde{\boldsymbol{y}}$ to $\tilde{\boldsymbol{u}}$ is naively constructed similar to $\tilde{A}$ or $\tilde{C}$ as a block-diagonal matrix with blocks G on the diagonal ($\tilde{G} = I_n \otimes G$), one has set the entire generalised system to steady-state: On every dynamical order, the state-update equation will be in steady state. This causes an unnecessary loss of information. If the generalised forward model is built from the highest dynamical order down, however, the relationship between the dynamical orders can be preserved. Consider the generalised LTI-state space equations in equation (5.6) that are not

perturbed by any noise.

$$\begin{aligned} \dot{\boldsymbol{x}} &= A\boldsymbol{x} + B\boldsymbol{u} & \boldsymbol{y} &= C\boldsymbol{x} \\ \dot{\boldsymbol{x}}' &= A\boldsymbol{x}' + B\boldsymbol{u}' & \boldsymbol{y}' &= C\boldsymbol{x}' \\ &\vdots & &\vdots \\ \dot{\boldsymbol{x}}^{(p)} &= A\boldsymbol{x}^{(p)} + B\boldsymbol{u}^{(p)} & \boldsymbol{y}^{(p)} &= C\boldsymbol{x}^{(p)} \end{aligned}$$

The procedure below then provides a generalised forward model $\tilde{G}$ by iteratively determining the relationship between $\tilde{\boldsymbol{y}}$ and $\tilde{\boldsymbol{u}}$ at some given dynamical order.

$$\begin{aligned} 0 &= A\boldsymbol{x}^{(p)} + B\boldsymbol{u}^{(p)} \\ \boldsymbol{x}^{(p)} &= -A^{-1}B\boldsymbol{u}^{(p)} & \longrightarrow \quad \boldsymbol{y}^{(p)} &= -C\left(A^{-1}B\boldsymbol{u}^{(p)}\right) \\ \\ \dot{\boldsymbol{x}}^{(p-1)} \equiv \boldsymbol{x}^{(p)} &= A\boldsymbol{x}^{(p-1)} + B\boldsymbol{u}^{(p-1)} \\ \boldsymbol{x}^{(p-1)} &= -A^{-2}B\boldsymbol{u}^{(p)} - A^{-1}B\boldsymbol{u}^{(p-1)} & \longrightarrow \quad \boldsymbol{y}^{(p-1)} &= -C\left(A^{-2}B\boldsymbol{u}^{(p)} + A^{-1}B\boldsymbol{u}^{(p-1)}\right) \\ &\vdots & &\vdots \\ \dot{\boldsymbol{x}}^{(1)} \equiv \boldsymbol{x}^{(2)} &= A\boldsymbol{x}^{(1)} + B\boldsymbol{u}^{(1)} \\ \boldsymbol{x}^{(1)} &= -A^{-p}B\boldsymbol{u}^{(p)} - A^{-(p-1)}B\boldsymbol{u}^{(p-1)} \ldots & \longrightarrow \quad \boldsymbol{y}^{(1)} &= -C\left(-A^{-p}B\boldsymbol{u}^{(p)} - A^{-(p-1)}B\boldsymbol{u}^{(p-1)} \ldots\right. \\ &\quad - A^{-1}B\boldsymbol{u}^{(1)} & &\quad \left. -A^{-1}B\boldsymbol{u}^{(1)}\right) \\ \\ \dot{\boldsymbol{x}} \equiv \boldsymbol{x}^{(1)} &= A\boldsymbol{x}^{(0)} + B\boldsymbol{u}^{(0)} \\ \boldsymbol{x}^{(0)} &= -A^{-(p+1)}B\boldsymbol{u}^{(p)} - A^{-p}B\boldsymbol{u}^{(p-1)} \ldots & \longrightarrow \quad \boldsymbol{y}^{(0)} &= -C\left(-A^{-(p+1)}B\boldsymbol{u}^{(p)} - A^{-p}B\boldsymbol{u}^{(p-1)} \ldots\right. \\ &\quad - A^{-1}B\boldsymbol{u}^{(0)} & &\quad \left. -A^{-1}B\boldsymbol{u}^{(0)}\right) \end{aligned} \tag{5.6}$$

Following this derivation, a generalised forward model $\tilde{G}$ can be constructed, as is provided in equation (5.7). A Matlab function to create such a generalised forward model is presented in listing B.7 in appendix B.3.

$$\tilde{\boldsymbol{y}} = \tilde{G}\tilde{\boldsymbol{u}}$$

$$\begin{bmatrix} \boldsymbol{y}^{(0)} \\ \boldsymbol{y}^{(1)} \\ \vdots \\ \boldsymbol{y}^{(p)} \end{bmatrix} = -\begin{bmatrix} CA^{-1}B & CA^{-2}B & \cdots & CA^{-p}B & CA^{-(p+1)}B \\ 0 & CA^{-1}B & \cdots & CA^{-(p-1)}B & CA^{-p}B \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ & & & CA^{-1}B & CA^{-2}B \\ 0 & \cdots & \cdots & 0 & CA^{-1}B \end{bmatrix} \begin{bmatrix} \boldsymbol{u}^{(0)} \\ \boldsymbol{u}^{(1)} \\ \vdots \\ \boldsymbol{u}^{(p)} \end{bmatrix} \tag{5.7}$$

5.3. The simulated system

Given the update rules for perception and action in equations (5.4) and (5.5) and the generalised forward model in equation (5.7), a control loop can be implemented. This research includes simulations of a simple one-dimensional SISO-system, to simplify as much as possible in order not to complicate the problem too much. An analysis of the workings and performance of an Active Inference agent with generalised sensory input by finite differences can be made.

The simulated system is a point mass with a single state: its velocity. It can move in one dimension. One could consider a car driving on an endless straight road, or a boat sailing straight on open sea with no position-reference. A Free Body diagram of such a system is shown in figure 5.2. If the system has a point mass m[kg], velocity v[m/s] which is the state x, input force $u = F_u$[N], and damping force dv[N], with d[N s/m] some damping constant, and there is a disturbance force, proportional to the mass $w = \frac{1}{m}F_w\left[\text{m/s}^2\right]$ acting on the system, the following force-equilibrium must hold:

$$m\dot{x} = u + mw - dx$$

Considering a direct velocity measurement, corrupted by some noise z[m/s], the State Space matrices

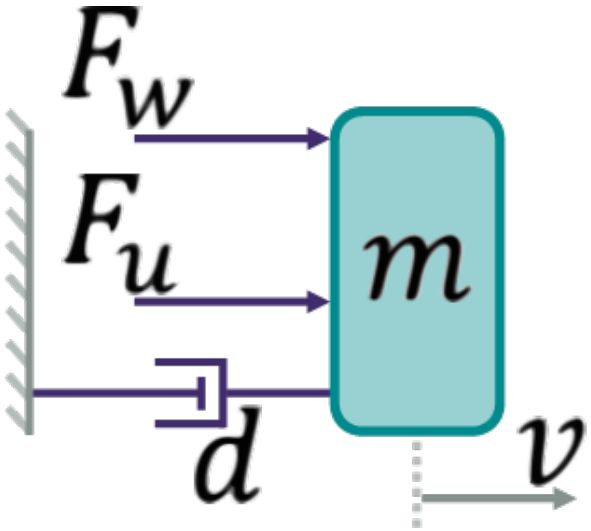


Figure 5.2: Free Body Diagram of a one-DOF SISO system for simulations.

of equation (5.1) are as follows:

$$A = -\frac{d}{m}, \quad B = \frac{1}{m}, \quad C = 1$$

For simplicity, $m = d = 1$. Given this data about the model, all that is left is to artificially create coloured noise to add to a simulation. The noise parameters (variance and smoothness) are then known and can be used to construct the generalised precision matrices. When a choice for embedding order p and accuracy of the finite differences for generalised output o (chapter 3) are set, together with the gradient descent learning parameters α_μ and α_u, and the prior knowledge $\tilde{\xi}$ is chosen, the agent's internal model and perception and action rules can be implemented. It is worth noting that in this simulation scenario, the agent knows the simulated model and thus has nearly perfect knowledge of its environment. Even the actual noise parameters are known to the agent. This may seem unrealistic, however, the Free Energy principle is an underlying principle for many mechanisms, of which Active Inference is just a part. One example is that of Dynamic Expectation Maximization (DEM) [18], which is not a control algorithm but meant for parameter estimation. It leverages the Free Energy principle to estimate system parameters and hyperparameters, including noise characteristics. Therefore, the current simulation scenario can be considered one of a well-known generative process, possibly identified earlier using DEM, that is now merely being controlled.

5.4. On-line simulation with generalised output

The control-scenario of interest is one in which a real system, affected by coloured noise, is being controlled by an Active Inference agent. The agent has access to a model of the system (environment), can provide it with input and can observe the system output by means of some sensory measurement. The closest simulation scenario is one in which the environment is simulated by means of a model, disturbed by artificially created noise. All simulations for this section and section 5.5 are performed using the Matlab code displayed in appendix B, with the simulation scripts specifically in appendices B.4.1 and B.4.2.

5.4.1. Implementation

Because the control is to be done on-line, generalised coordinates can only be obtained by means of backward differences (section 4.1.3). Simulations are implemented in Matlab, for which the scripts are provided in appendix B.4. An interesting challenge in this simulation, in which a plant is simulated by means of Forward-Euler integration, is a lack of compatibility between the agent and plant when it comes to input (action). The plant takes a non-generalised input $\boldsymbol{u}$ and provides a non-generalised output $\boldsymbol{y}$. This non-generalised output is generalised in simulation by means of a backward differences matrix E, multiplied with a time-series array $\check{\boldsymbol{y}}$. The update rule for action in equation (5.5), which is part of the agent, works entirely on generalised coordinates and as such provides a generalised input $\tilde{\boldsymbol{u}}$.

This input cannot be processed by the plant, and thus an alternative solution is required. The gradient descent in equation (5.5) could be replaced by a partial derivative w.r.t. $\boldsymbol{u}$ instead of $\tilde{\boldsymbol{u}}$. This would result in a fourth chain rule term in the second line of equation (5.5), the partial derivative $\frac{\partial \tilde{\boldsymbol{u}}}{\partial \boldsymbol{u}}$. This partial derivative can be represented by a matrix of size $l(p+1) \times l$, of which the first l rows are filled with ones and all other rows with zeros. In a one-DOF SISO case, this boils down to an array $(p+1) \times 1$, of which only the top entry is a 1. This effectively results in the disposal of the generalised action computed by the agent and taking only the zero-derivative of this generalised action to provide to the generative process. This obviously is a loss of information in the closed-loop. Such a simulation is performed with the choice of parameters displayed in table 5.1, the choice of which is explained later in this section.

Table 5.1: Parameters for simulation of a one-DOF SISO system.

Δt[s]	p	o	σ_w[m/s]	σ_z[m/s]	s_w	s_z	α_μ	α_u
1×10^{-3}	3	1	0.05	0.05	0.1	0.1	0.2	0.1

Coloured noise is created for process and observation. The Gaussian filter that is used to make artificial coloured noise is non-causal and the noise is therefore created before the simulation begins. In the case of a real-life system, the noise would just be present in the environment so there is no need to be able to create it on-line in simulation. For repeatability of experiments, the random seeds to create the noise are fixed. The noise signals are shown in figure 5.3.

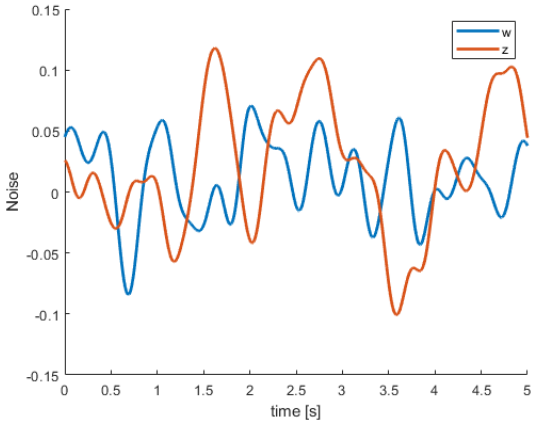


Figure 5.3: Coloured noises with $\sigma_{w,z} = 0.05$ and $s_{w,z} = 0.1$. Random seeds in Matlab are 4 for w and 6 for z.

Simulations with the parameters from table 5.1 are run for a simulation time of 5s. The prior variable $\tilde{\boldsymbol{\xi}}$ steers the controller towards a desired behaviour. The error of the belief (equation (5.3)) is minimized in the free energy. Therefore, choosing $\tilde{\boldsymbol{\xi}}$ as follows below makes the agent steer the belief $\tilde{\boldsymbol{\mu}}$ of the state $\tilde{\boldsymbol{x}}$ towards the desired $\tilde{\boldsymbol{\mu}}_{ref}$.

$$\tilde{\boldsymbol{\xi}} = (\mathcal{D} - \tilde{A})\tilde{\boldsymbol{\mu}}_{ref}$$

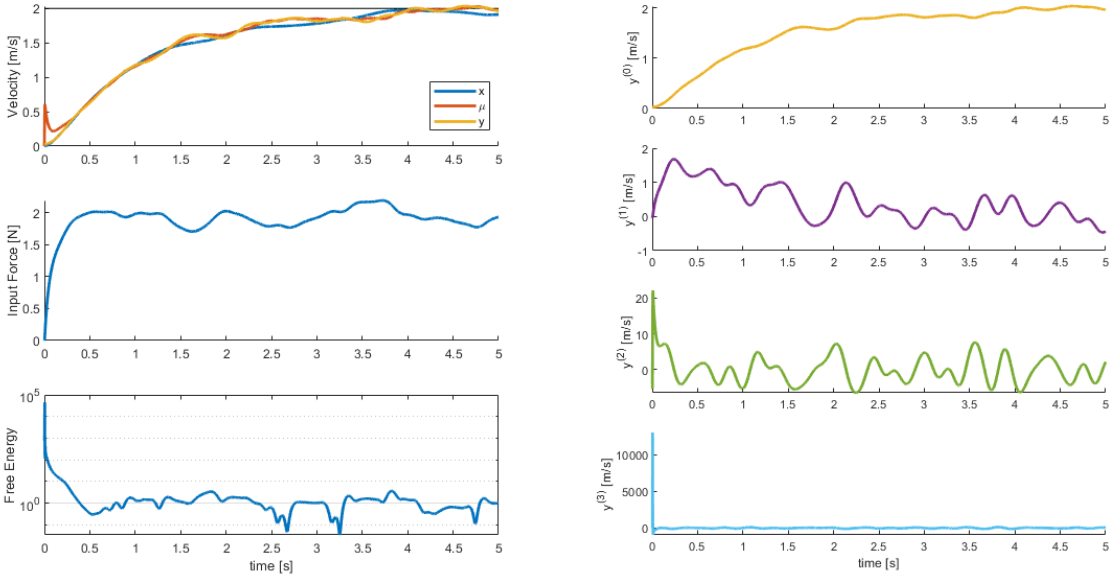
To steer the mass towards a constant velocity, the generalised prior variable is constant for all time instances and equals the desired velocity (an arbitrary 2m/s for the zero-order and zero otherwise. With an embedding order $p = 3$, the prior for this simulation is constant:

$$\tilde{\boldsymbol{\xi}} = \begin{bmatrix} 2 & 0 & 0 & 0 \end{bmatrix}^T$$

Furthermore, the state x and belief of motion $\tilde{\boldsymbol{\mu}}$ are initialized at 0. Simulation results are presented in figure 5.4. This simulation shows some interesting results. It has been tuned using the noise parameters and gradient descent learning rates. Unfortunately, there is not a lot of freedom in the choice of these parameters. As they are, reference tracking under the influence of coloured noise is achieved, but there is little to no freedom to tweak the system and evaluate behaviour and performance. Many choices of parameters render the system unstable.

At first glance, in figure 5.4a shows that the state x, output y that measures the state and the belief of the state μ are all steered towards the reference velocity of 2m/s. The influence of noise is clearly present, but as figure 5.3 shows, the noise has significant amplitude. An interesting peak in the agent's belief μ shows at the beginning of the simulation. This behaviour is inherent to the duality of the Free Energy: both $\tilde{\boldsymbol{\varepsilon}}_\mu$ and $\tilde{\boldsymbol{\varepsilon}}_y$ need to be minimized, and initially those are in conflict due to the influence of the prior $\tilde{\boldsymbol{\xi}}$ on $\tilde{\boldsymbol{\varepsilon}}_\mu$. The Free Energy decreases and remains small from the moment the belief is consistent with the state.

Figure 5.4b shows the generalised output coordinates $\tilde{\boldsymbol{y}}$. These are obtained on-line during simulation by means of equation (4.11). The amplitude of the output with increasing dynamical order clearly



(a) Simulation results: states and outputs, input and Free Energy.

(b) Generalised outputs

Figure 5.4: Simulation results of an Active Inference control loop with generalised output coordinates by means of backward finite differences and parameters as in table 5.1.

increases, which is to be expected for a signal with oscillating behaviour. However, by the 3rd derivative, the output is extremely high for the first few iterations of the simulations, as is clear from the enormous peak. A smaller version of this peak already shows for the 2nd derivative. For this reason, the embedding order for this simulation is limited to $p = 3$. For every embedding order added, there is a larger peak, rendering the simulation unstable.

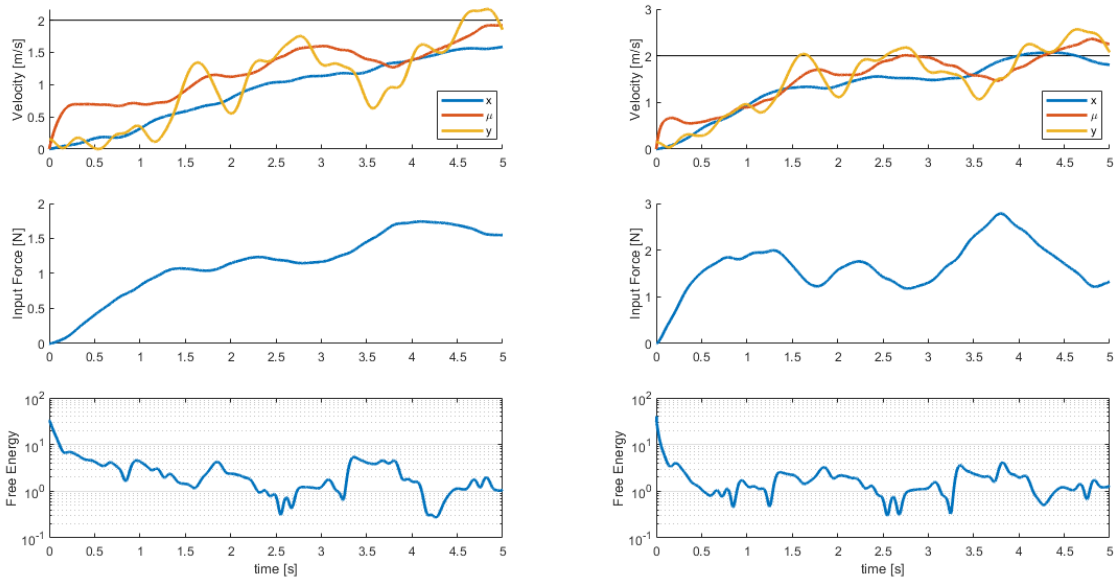
5.4.2. Varying noise characteristics

Since the influence of noise on the simulation is obvious from figure 5.4a, varying the noise to evaluate behaviour and performance is an important step towards understanding the mechanics of an active inference control loop with generalised output coordinates. The embedding order is still bound to $p = 3$, smaller than the desired $p = 5$, but varying of the noise yields some important insights. Most importantly, smaller (smaller variance) or smoother (larger kernel width) noise is more problematic to this simulation than the opposite. Simulations for a varying set of parameters, as displayed in table 5.2 show this with the results in figure 5.5. Larger noise variance does not only increase the magnitude of

Table 5.2: Parameters for simulation of a one-DOF SISO system with varying noise parameters.

	Δt[s]	p	o	σ_w[m/s]	σ_z[m/s]	s_w	s_z	α_μ	α_u
Inc. st. dev. σ	0.001	3	1	0.3	0.3	0.1	0.1	0.2	0.1
- w/ adj. learn. rates	0.001	3	1	0.3	0.3	0.1	0.1	0.5	0.5
Dec. smoothness s	0.001	3	1	0.05	0.05	0.05	0.05	0.5	0.5
- w/ adj. learn. rates	0.001	3	1	0.05	0.05	0.05	0.05	0.5	0.5

the noise, but decreases precision in $\tilde{\Pi}_w$ and $\tilde{\Pi}_z$. This shows clearly in figure 5.5a, with parameters as in the first row of table 5.2, in which the effect of the noise is much more present than in figure 5.4a. The decreased precision also makes the control behaviour of the agent less aggresive: the state progresses towards the desired value at a lower pace. Interestingly enough, the effect of larger noise on the state can barely be seen. The belief of the agent deviates far more from the actual environment state, since it is affected by the disturbed sensory input. The environment state, however, remains relatively smooth in comparison to the present noise, suggesting the Active Inference agent is successful in counteracting



(a) Increased noise variance ($\sigma_{w,z} = 0.3$).

(b) Adjusted learning rates ($\alpha_{\mu,u} = 0.5$).

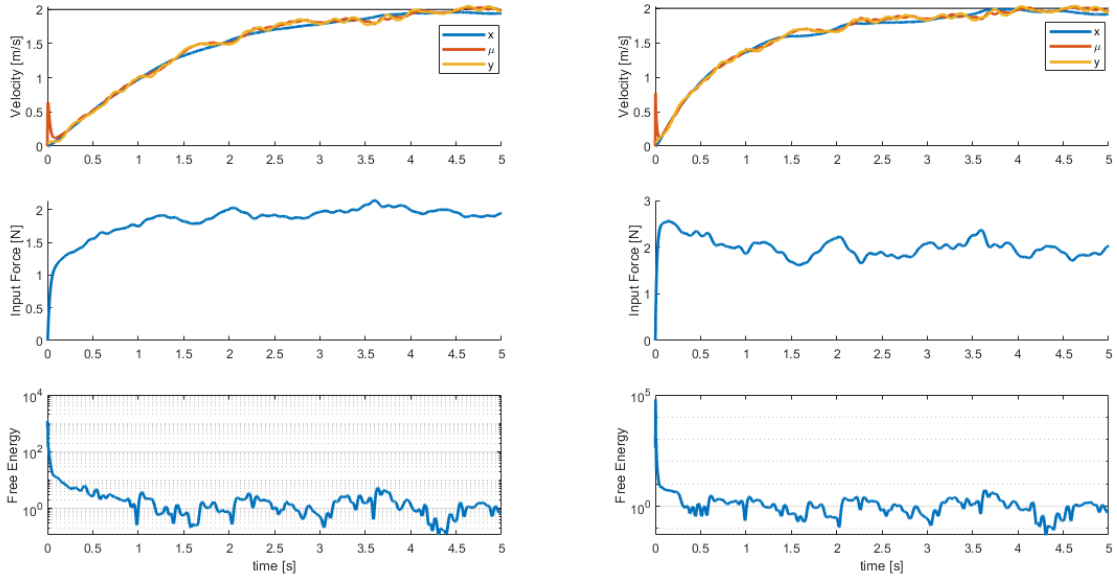
Figure 5.5: Simulation results of an Active Inference control loop with generalised output coordinates, with varying noise parameters (increased variance) as in table 5.2.

coloured noise. An increase of the learning rates as in the second row of table 5.2 yields results displayed in figure 5.5b, showing that the decreased control action and influence of noise on sensory input and belief (the relevant signals for the agent) can be counteracted somewhat by an increase in learning rates. Aside from variance, the characteristics of the noise are also greatly influenced by the Gaussian smoothness or kernel width. A decrease also causes a decrease of precision in $\tilde{\Pi}_w$ and $\tilde{\Pi}_z$. With parameters as in the third and fourth rows of table 5.2, results are shown in figure 5.6. It seems that when the agent has less confidence in its sensory input and knowledge of the environment, the noise indeed has a smaller influence on the belief of motion μ which influences the action of the agent, which in turn influences the sensory input. Increase in learning rates can make the control action more aggressive (figure 5.6b).

Intuitively, one might expect larger noise, or noise that is less smooth to make a system more prone to instability. However it seems from the results in figures 5.5 and 5.6 that the effect of the change of the precision matrices $\tilde{\Pi}_w$ and $\tilde{\Pi}_z$ has a much greater effect on the closed loop than the change in the actual noise signals. Furthermore, comparing figures 5.5 and 5.6 with figure 5.4 shows that the decreased precision results in a lower free energy, especially early in the simulation. In the scenario in this section, the noise parameters that were used to create the noise are the same parameters that make up the precision matrices. Given the influence of the precision matrices of the behaviour in the closed loop, it is important to research the effects of the precision matrices as tuning parameters. When noise is not artificially created, the characteristics of it might be estimated but they are never truly known. It thus seems logical that the precision matrix does not represent the precision due to noise exactly. This research, however, is not part of the research for this thesis.

5.5. Simulation of a generalised plant

In section 5.4 it has been shown that closed-loop control can be achieved for the tracking of a reference, using generalised outputs obtained by backward finite differences in a state space formulation. This scenario comes with a lot of drawbacks, however, among which are the sacrifice of generalised action, sensitivity to parameter changes, especially regarding noise, and limitation of embedding order. The simulations in this section are aimed at providing a better understanding of the closed-loop behaviour in a scenario with generalised coordinates, taking advantage of the artificially generated noise and thus the knowledge thereof. With this knowledge, it is possible to simulate a generalised plant, as



(a) Decreased noise smoothness ($s_{w,z} = 0.05$).

(b) Adjusted learning rates ($\alpha_{\mu,u} = 0.5$).

Figure 5.6: Simulation results of an Active Inference control loop with generalised output coordinates, with varying noise parameters (decreased smoothness) as in table 5.2.

in equation (5.8). In this generalised plant, the generalised noise, obtained by finite differences, is added to the equations, meaning the plant outputs generalised sensory input for the agent and takes its generalised action.

$$\begin{aligned} \dot{\tilde{\boldsymbol{x}}} &= \tilde{A}\tilde{\boldsymbol{x}} + \tilde{B}\tilde{\boldsymbol{u}} + \tilde{\boldsymbol{w}} \\ \tilde{\boldsymbol{y}} &= \tilde{C}\tilde{\boldsymbol{x}} + \tilde{\boldsymbol{z}} \end{aligned} \quad \text{with} \quad \begin{aligned} \tilde{A} &= I_{p+1} \otimes A, \quad \tilde{B} = I_{p+1} \otimes B \\ \tilde{C} &= I_{p+1} \otimes C \end{aligned} \tag{5.8}$$

In order to simulate the plant in equation (5.8), the generalised noises $\tilde{\boldsymbol{w}}$ and $\tilde{\boldsymbol{z}}$ need to be known. With those, different scenarios can be created to identify problems of the simulations of section 5.4 and possibly find solutions. or identify future research towards finding solutions. Sections 5.5.1 and 5.5.2 are about the alternative scenarios and the resulting simulations, respectively.

5.5.1. Generalised noise

Because the artificial noise is generated before simulation, it is possible to use this information to compute generalised noise. In the case of backward differences, this can be done on-line, very similar to computing the generalised output in section 5.4. However, doing so is completely similar to a pre-processing step in which all the generalised noise is computed, which is more efficient to implement. Since there is no practical difference, this also allows for the exploration of central and forward finite differences for derivative approximation (which could not be done on-line). Three different scenarios are created, described below. In listing B.8 they are parametrized by `gp`, an integer taking on one of the values below. The variable occurs all throughout listing B.9 to distinguish between the different scenarios. The scenarios are made to be as close as possible to the original scenario in section 5.4.

1. Generalised coloured noise is created by means of finite differences, possibly backward, central or forward, given a coloured noise signal that is longer (has more samples) than the simulation has time-samples, such that there need to be no zero-samples in the derivatives for the first and/or last few samples of the simulation. A full embedding order is available for all of the simulation. This scenario, together with others should show whether the lack of full embedding order for the first few samples and increase thereof during the simulation in section 5.4 is problematic and possibly the cause of the generalised output peaks described in section 5.4.

2. In this scenario, generalised coloured noise is also created by means of finite differences. How-

ever, in contrast to scenario 1 above, there are no additional samples for the noise. Only backward differences are considered. This means that the higher-order derivatives of the noise are zero for the first few samples.

3. A final scenario is closest to the scenario of section 5.4. Backward differences are applied to obtain coloured noise before the generalised plant is simulated. Due to the lack of approximations for higher-order derivatives for the first few samples, the embedding order in simulation is gradually increased as the higher order derivatives (of the noise) can be approximated.

The next section described simulation results from the scenarios of a generalised plant and what these reveal about the closed-loop control with an Active Inference agent.

5.5.2. Simulations

Simulations are performed for the above scenarios by varying the integer for the variable `gp` inlisting B.8. Other parameters are as those in table 5.1, for a fair comparison, aside from the embedding order, which is increased to $p = 5$, and the prior variable which is extended accordingly, to $\tilde{\boldsymbol{\xi}} = \begin{bmatrix} 2 & 0_{1\times p} \end{bmatrix}^T$. The parameters for the simulations are displayed in table 5.3. Running a simulation for all three scenarios

Table 5.3: Parameters for simulation of a one-DOF SISO system with a generalised plant.

Δt[s]	p	o	σ_w[m/s]	σ_z[m/s]	s_w	s_z	α_μ	α_u
0.001	5	1	0.05	0.05	0.1	0.1	0.2	0.1

described in section 5.5.1, which can be reproduced using the scripts in appendix B, shows that the differences between the scenarios are so minimal, that they are not worth reporting. There are, however, very clear differences with the simulations of section 5.4. For comparison, results of the third scenario described in section 5.5.1 are compared to those in section 5.4. With the parameters as in table 5.3, generalised noise and outputs are shown in figure 5.7, and simulation results are shown in figure 5.8. The latter only shows non-generalised state, belief and input, to keep the plots clean and interpretable.

Analysing the results of this simulation, some interesting remarks can be made. Starting with the noise plots in figure 5.7a, it can be observed right away that the noise derivatives have a mellow behaviour and no disproportional peaks are present, which also holds for the generalised outputs in figure 5.7b. The effect on states and the belief thereof can be seen in figure 5.8 and considering that the parameters for this simulation are the same as for the simulation shown in figure 5.4a (apart from the embedding order), it is noteworthy how different the behaviour is. The control action is much larger, resulting in a far shorter rise time but a more oscillatory response. Noise seems to be counteracted quite well. Understanding the underlying cause of these differences might be very helpful towards understanding the control scenario of section 5.4, which could lead to improving it. This however, is a topic for future research.

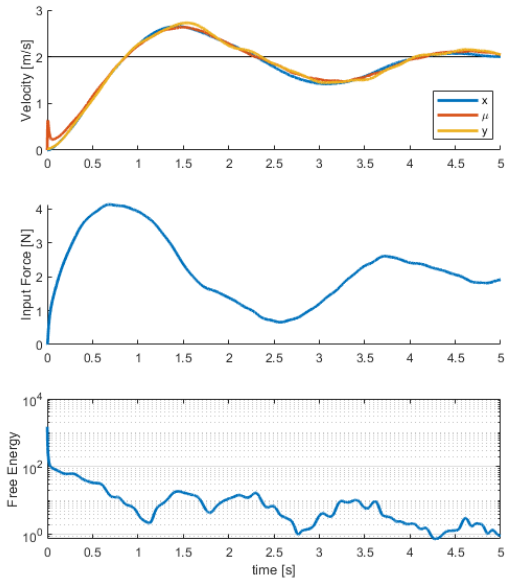
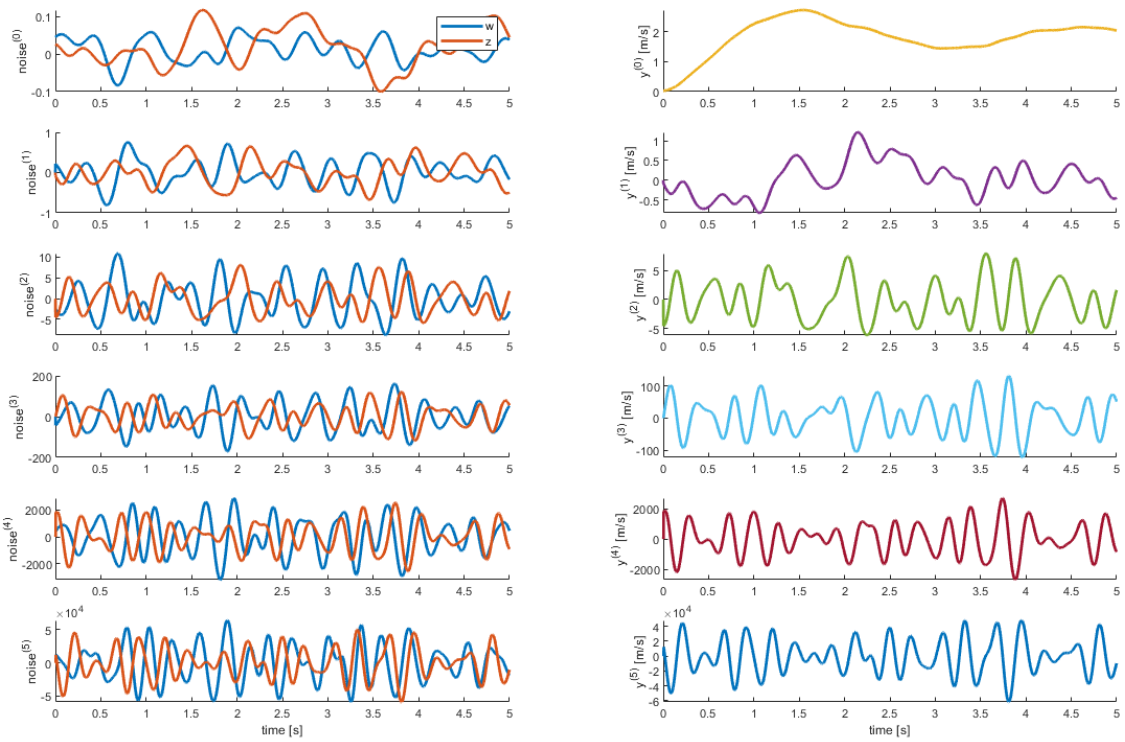


Figure 5.8: Simulation results of an Active Inference control loop with a generalised plant, with parameters as in table 5.3.

The generalised outputs of this simulation (figure 5.7b) require further examination. Just from looking at these plots, it can be seen that the plot for $y^{(1)}$ does not correspond to the temporal derivative of $y^{(0)}$. This is further inspected in figure 5.9, in which the generalised outputs are compared to approximate derivatives by means of



(a) Generalised noise

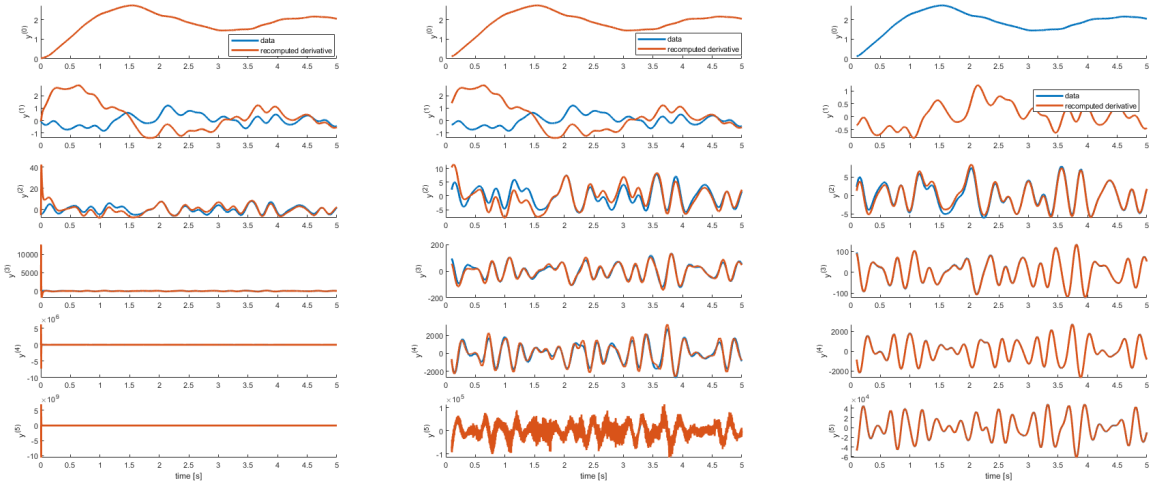
(b) Generalised output

Figure 5.7: Generalised noise and output signals of an Active Inference control loop with a generalised plant, with parameters as in table 5.3.

backward finite differences, based on one of the signals $y^{(d)}$. In listing B.6 is a Matlab function `f_diffcheck` that makes this comparison. The first plots in figure 5.9a show the data from figure 5.7b together with backward difference-approximated derivatives based on $y^{(0)}$. Two things become clear: the first derivative from the simulation does not nearly match the approximated derivative, although it converges as simulation time progresses, and, higher order derivatives have disproportionately large values for the first samples. This is the same phenomenon as observed in section 5.4. Therefore, in figure 5.9b the same comparison is displayed, but with scaled plots. The disproportionately large values are not displayed, such that the rest of the results can be reviewed. It is now clear that the discrepancy between the simulation data and the approximated derivatives based on $y^{(0)}$ vanishes as the dynamical order increases, and as the simulation time progresses. Lastly, in figure 5.9c the approximated derivatives are based on $y^{(1)}$ instead of $y^{(0)}$, with the first samples not displayed like in figure 5.9b. It is clear that the simulation outputs and approximated derivatives match very well. The underlying reasons for these discrepancies are not very evident, but an evaluation of the closed-loop State Space formulation is in place. This is the topic of section 5.6. Lastly, the varying of parameters in listing B.8 shows that the simulation with a generalised plant suffers from the same problem as the simulation in section 5.4: Increased precision renders the closed-loop unstable.

5.6. Evaluation

For the simulations and post-processing results in sections 5.4 and 5.5, some remarks can be made. The similarities and differences of the simulations in these sections provide interesting insight into the closed loop control scenario, divided into four topics. They are discussed below.



(a) Using $y^{(0)}$ as a reference.

(b) Using $y^{(0)}$ as a reference, scaled plot.

(c) Using $y^{(1)}$ as a reference, scaled plot.

Figure 5.9: Generalised outputs $\tilde{y}$ of the simulation in figure 5.8, compared to approximated derivatives by finite differences ($o = 1$).

Disproportionately large peaks at the beginning of a simulation are present in the derivative approximations resulting directly from the application of finite differences. This is the case for the generalised outputs in section 5.4 and for the post-processed approximations of section 5.5 in figure 5.9. This behaviour presents itself when the finite differences approximation is applied to the output y directly. Section 4.2 and figure 5.7a show that this is not a direct result of the application of finite differences. It is, however, known that approximating derivatives numerically is notoriously unreliable, especially in the presence of noise. In this case, the noise itself is not necessarily problematic. However, any relatively large difference between samples will be amplified by the numerical approximation. The problem may therefore lie with the combination of the workings of free energy minimization and the finite differences. Clearly, the peaks cause instability and it is important to research their cause and how to avoid it. It might, for example, be related to the behaviour of the belief $\tilde{\boldsymbol{\mu}}$, as discussed in section 5.4.

High precision in the precision matrices $\tilde{\Pi}_{w,z}$ as a result of large smoothness or small variance of the noise render the closed-loop unstable. It seems the coloured noise itself is not problematic for the Active Inference control loop. It can deal with large noises fairly well. When noise is large enough, the disturbances are very visible, but the system remains stable. The contents of the precision matrices, on the other hand, have a large influence on the behaviour of the closed loop. Despite the noise being small and/or smooth, which is represented by high precision, instability seems inevitable. In the simulations performed for this research, the precision matrices are defined by the characteristics of the actual noise. In the case of a real-world environment, the characteristics of the noise can only be estimated. The estimate may be very good, but the true characteristics are never known, meaning the precision matrices will not exactly match the noise. It is therefore questionable whether they should, and if precision (or the noise characteristics that define it) should be tuning parameters. This may positively impact the closed-loop control characteristics.

Input incompatibility is an issue in the scenario of section 5.4. By approximating generalised output by means of finite differences, providing a solution to the lack of generalised measurements from an environment that are expected by the agent, an opposite problem is created when it comes to the action generated by the agent. This generalised action is incompatible with the input accepted by the environment, as discussed in section 5.4, which results in a loss of information provided by the agent to the environment. In order to fully understand the Active Inference framework for robotics, it is important to research this issue.

Derivative mismatches between the generalised outputs of the simulations of section 5.5, figures 5.7b and 5.9 provide insights into the closed-loop simulation. Two problems can be pinpointed that can contribute to the presence of this mismatch. The first lies with the initial conditions. In the simulation, initial conditions are provided for all the dynamic variables: $\tilde{\boldsymbol{x}}$, $\tilde{\boldsymbol{\mu}}$, $\tilde{\boldsymbol{y}}$. The system is assumed to have zero velocity, but also zero acceleration or any other higher order motion (the mass is not moving) at the start of the simulation, hence all these variables are initialized at zero . Due to the presence of the noise, however, this is not necessarily true. With the wrong initial condition, the generalised coordinates can evolve differently than would be expected. Because of the correlation between dynamical orders, this should be corrected during the simulation, which appears to be the case. This is, however, not as straightforward as it seems, which is the second problem. Consider equations (5.2) and (5.8), which depend on the generalised system matrices $\tilde{A}$ and $\tilde{C}$ (and $\tilde{B}$). These matrices are (block-)diagonal (this follows from the Kronecker product with an Identity matrix), which means there is no correlation between the dynamical orders in the equations they are part of. Even though an equation $\dot{\tilde{\boldsymbol{x}}} = \tilde{A}\tilde{\boldsymbol{x}}$ has the d^{th} derivative on the left-hand side on the d^{th} row and on the right-hand side on the $(d+1)^{\text{th}}$ row (in case of a single state x), which makes it seem like the rows are cross-correlated, this is not actually the case in a simulation in which the state transition is performed by a state update rule $\tilde{\boldsymbol{x}}_{i+1} = \tilde{\boldsymbol{x}}_i + \dot{\tilde{\boldsymbol{x}}}_i \Delta t$. The same is true when the equation includes an input term $\tilde{B}\tilde{\boldsymbol{u}}$ or in case of the agent's internal model in equation (5.2). Yet, the derivatives in figure 5.9 converge to the approximate derivatives. This can only be due to the generalised forward model from section 5.2, equation (5.7), illustrating its importance. It remains, however, questionable whether the current form of State Space model for closed-loop Active Inference-based control (with generalised coordinates) is correct. Intuitively, these State Space equations should represent the dynamic correlation between the generalised coordinates.

Research towards the topics or problems mentioned above could provide a greater understanding of Active Inference and generalised coordinates for robotics. The research in Active Inference for robotics is young and there is far more research to be done before the topic is well-understood. The topics mentioned above are left for future research. This is discussed further in section 6.2. An answer to the research question that this chapter is devoted to can be provided. It reads: 'How can Active Inference with generalised motions be applied to an LTI-State Space control loop?' A short answer is provided by sections 5.1, 5.2 and 5.4, that show the required update equations for perception and action following the gradient descent on Free Energy. The application of generalised precision matrices from chapter 3, finite differences from chapter 4 and the generalised forward model from section 5.2 make closed-loop control with generalised coordinates possible in an environment that does not provide them. In section 5.4 it is shown that closed-loop control for reference tracking can be achieved. However, from sections 5.4 and 5.5 it is also clear that the current State Space formulation for generalised coordinates is far from perfect and more research is required for a true answer to the question. The tools researched and applied in this work (generalised precision, finite differences and a generalised forward model) are keys to the state-space implementation of Active Inference with generalised coordinates, but the problems described in this section, and probably more, are in need of further research.

6

Conclusion

This final chapter is one of conclusion, discussion and recommendations for future work. The research posed in previous chapters will be summarized and the research questions posed in chapter 1 will be revisited. This research does not only answer open questions in the research towards the application of Active Inference in robotics, but also provides new questions, opening up opportunities for future research that can hopefully, eventually lead to a successful implementation of Active Inference in robotics that outperforms existing control methods in certain scenarios.

6.1. Research summary

The aim of this research was to explore aspects of Active Inference mentioned in neuroscientific literature and in some of the pioneering literature of its application to robotics that have not yet been applied in their full potential when it comes to the control of robotics. The field of Active Inference is vast and although the first successful implementations in robotics exist [26–28], the full potential has most definitely not yet been reached. One aspect of Active Inference and the Free Energy Principle that has not been widely explored in robotics so far is the application of generalised coordinates, or generalised motions. These are mentioned often, but their application is either limited in the number of generalised coordinates or lacks some of the key features of these generalised coordinates, such as the dynamical correlation among them and the useful information it provides in a control loop. The dynamical correlation that generalised coordinates provide are tightly related to the assumption of the presence of coloured noise to disturb system states or measurements. This research therefore aims to answer the question (chapter 1):

What constructs are necessary to apply generalised coordinates to an Active Inference control loop whilst taking correlation between dynamical orders into account?

This research question can be answered after the sub-questions are answered. The work of this thesis is summarized below, together with answers to these sub-questions from section 1.2.1.

6.1.1. Generalised coordinates

The first sub-question of this research reads ‘What is the role of generalised coordinates in Active Inference?’. An understanding of generalised coordinates of motion is of utter importance for this research. Chapter 2 is all about the Free Energy Principle, which is the underlying theory for Active Inference. The literature on this topic is narrow and intricate. A review [4] already provides a good insight into the math involved with the Free Energy Principle and Active Inference, but makes some simplifying assumptions about the lack of correlation between dynamical orders in generalised coordinates that are not made in this research. The chapter provides the required background necessary to understand the research in this thesis, as familiar to a reader from the field of robotics or control as possible. Free Energy is a quantity that, due to its form, can be optimized, in contrast to the surprisal or the recognition (probability) density that it bounds. Minimizing Free Energy, which in practice boils down to a

quadratic cost function of state and observer error, weighted by the precision of these quantities, indirectly minimizes the surprisal and the difference between an agents model (recognition density) and the true Bayesian posterior of a state given an observation. Minimization of the Free Energy is achieved by minimizing w.r.t. the (belief of) the environment state (perceptual inference), and w.r.t. the environment input or action on the environment (active inference). In order to do so, explicit representations of the underlying probability densities of internal beliefs about the environment and the generative mapping (generative density) between environment states and sensory inputs are required. Assuming Gaussian densities, the agent's internal beliefs about the environment are encoded by the Gaussian sufficient statistics: mean ($\boldsymbol{\mu}$) and variance. The generative density consists of a prior density (state equation) and conditional density (output equation). Both densities are considered to be made up of a deterministic part (state and output model), influenced by noise, which is responsible for the stochastic nature of the probability densities. Assuming Gaussian (coloured) noise on both process and measurement has multiple advantages. It presents the opportunity to define the generative density as a Gaussian density, of which the statistics are represented by the coloured noise. This yields the practical quadratic form of the Free Energy that is weighted by the Gaussian covariance, which is important in the second sub-question. Furthermore, with coloured noise all dynamical processes remain smooth, instead of having the roughness that comes with white noise. As such, the temporal derivatives of the dynamical processes exist and are correlated. This is what generalised coordinates of motion are: instantaneous temporal derivatives of a dynamical process that are correlated. It appears that biological agents register these generalised motions. Due to the correlation present, these provide information about the (progression of) the coloured noise. This is potentially very advantageous. So, generalised coordinates provide information about the dynamical processes involved in perception and control in the presence of coloured noise that can improve perception and action when influenced by noise. The research for the second sub-question provides more knowledge of the relationship between the coloured noise and generalised coordinates.

6.1.2. Generalised precision

Following up on the first research sub-question is 'What is the relationship between generalised coordinates and coloured noise?'. The research of chapter 2 and from the research and summary above it is already shown that these are tightly related. The Free Energy function that is minimized in the process of Active Inference weighs the quadratic error on state and observation by their believed precision, which fully depends on the characteristics of the noise. Precision is simply the inverse of covariance. Because different dynamical orders (the generalised coordinates) are assumed correlated, the covariance comes in the form of a generalised covariance matrix, which is a matrix that contains the (co-)variances of the noise signal and the derivatives thereof. Chapter 3 is dedicated to a thorough derivation of the contents of such a matrix, in which different noise sources (eg noises on different states) are assumed to be uncorrelated, but the correlation of a noise signal and its own derivatives is non-zero. As it turns out, each covariance term in the matrix is equal to the product of the negative variance of the noise and a derivative of the autocorrelation function of the noise, evaluated at zero lag. For every derivative in the covariance term, the autocorrelation function must be differentiated once. In the case of Gaussian noise, the autocorrelation function can be that of a Gaussian filter, which is well-defined and the derivatives of this analytic function are easy to compute. The result is a generalised covariance matrix of which the elements all depend on the variance and a power of the smoothness or kernel width of the Gaussian filter. In the generalised precision matrix, the inverse of the generalised covariance matrix, smoothness is a factor in the numerator of the elements and variance occurs in the denominators. With the assumption that both are smaller than one, increased variance decreases precision and increased smoothness increases precision. The power of the smoothness increases towards the bottom right end of the matrix, meaning precision decreases with increased embedding order (higher order derivatives). It has been shown that for sixth derivatives and higher, the precision essentially becomes zero, meaning the embedding order of generalised coordinates should generally not be higher than five. Until there, the generalised precision as a result of the application of generalised coordinates can weigh the quadratic error in the Free Energy function, meaning that when precision is high (the noise is smooth and/or small), error must be accounted for but whenever precision is lower (in the cases of rougher or larger noise, meaning larger uncertainty), perception and action should not be aggressively adjusted when perceptions don't match the agent's beliefs about the environment.

6.1.3. Perceiving generalised motions

The role and potential of generalised coordinates have become clear from the research in chapters 2 and 3, yet in literature no general application of generalised coordinates can be found. The application is usually limited to the available generalised output, which is little. A practical problem with generalised coordinates is that, although biological agents appear to be able to perceive them, sensors for robotics generally cannot. The third sub-question therefore reads: 'How can generalised coordinates be generally applied in the Active Inference framework when they are not readily available?'. Research in chapter 4 has explored the use of finite differences to obtain generalised output measurements. In the case of backward differences, this process is causal and can be applied on-line. A procedure is derived that eliminates the need to go through the process of deriving the finite difference equations and allows to set up the required matrix without further computations. Tests with an analytic ground truth reveal that for higher order derivatives, numerical approximation is noisy (as is a well-known culprit of numerical derivative approximation), however for an embedding order up to 5 the approximations are accurate enough. The addition of coloured noise shows that the derivative approximations remain smooth, but oscillations are much bigger, starting at lower embedding orders. Part of the oscillations are due to the true derivatives, because the original signal has more oscillations as a result of the coloured noise. Since generalised precision in the Free Energy formulation already decodes that precision decreases with increasing orders of derivatives, this culprit is partially accounted for in Active Inference. Therefore, finite differences can provide generalised coordinates in Active Inference.

6.1.4. State Space control with Active Inference

An Active Inference control loop based on an LTI-State Space model has been implemented in chapter 5 to evaluate the application of Active Inference to robotics in the presence of generalised coordinates and provide an answer to the question 'How can Active Inference with generalised motions be applied to an LTI-State Space control loop?' In other works, the State Space-description lacks generalised outputs or lacks the correlation between dynamical orders of generalised coordinates, both of which change the mechanics of the control loop. To retain the correlation between dynamical orders, proper derivations of the controller and filter equations are presented and a generalised forward model is proposed that relates the in- and output variables of all dynamical orders. Simulations of a simple one-dimensional point-mass system reveal the complexities of applying generalised coordinates in a control loop. When considering a robotic system (plant) as is usual, which runs in non-generalised coordinates, a compatibility issue between plant and agent is present. Providing an agent with generalised output (sensory input) results in generalised input (action), which the plant is incompatible with. A quick solution to run a working simulation is to discard the generalised input provided by the agent when simulating the plant. This causes a loss of information generated by the agent. Simulation of a reference tracking task with such a system is found to be unstable for embedding orders higher than 3, and marginally stable otherwise w.r.t. the tuning parameters. In stable scenarios, the agent is capable of performing the tracking task in the presence of coloured noise. A different simulation was created that runs a plant in generalised coordinates to identify the problems of the former simulation scenario. This scenario is only possible in simulation, when the noise is artificially created and therefore known. This plant yields generalised outputs and accepts generalised inputs, something a real-life robotic system is generally not capable of. It shows that an Active Inference based agent can perform a tracking task under the presence of coloured noise, but it also reveals a lack of connection between dynamical orders in the generalised description that underlies the filter- and input equations of the agent. The outputs of the process reveal the discrepancy between the generalised coordinates and raise the question whether the generalised state space-description is not fundamentally wrong. So, Active Inference with generalised motions can be applied to an LTI-State Space control loop when applying finite differences to obtain generalised output and adopting a generalised forward model, but accepting the discrepancy between the input generated by the agent and accepted by the plant is necessary. More research is required before Active Inference can be applied to robotics systems without the current compromises. This is further discussed in section 6.2.

6.1.5. The research question

Answers to the research sub-questions posed in section 1.2.1 have been provided throughout this research and summarized in sections 6.1.1 to 6.1.4. The research question of this work is now revisited:

What constructs are necessary to apply generalised coordinates to an Active Inference control loop whilst taking correlation between dynamical orders into account?

The application of generalised coordinates in Active Inference in general, without the explicit means to measure generalised outputs first of all requires a method to obtain generalised measurements. It has been shown that this can be done by means of finite differences, with which derivatives can be numerically approximated. Generalised coordinates of motion are tightly related to coloured noise. Due to the dynamical correlation between the dynamical orders of generalised motions, Active Inference can supposedly better deal with coloured noise than other methods. In order to respect the correlation between dynamical orders, a generalised precision matrix is required. This matrix is derived in detail and weighs the errors in the Free Energy based on the variance and smoothness of the noise. Precision decreases for higher order derivatives, of which the computed generalised coordinates are also less accurate. With the means to numerically approximate derivatives, with a generalised precision matrix and a generalised forward model to preserve correlation between dynamical orders, update equations for the agent's belief and for the input or action can be derived. An agent based on such equations, however, is not compatible with a non-generalised plant. This is further discussed in section 6.2.

6.2. Discussion and recommendations

Active Inference is a very interesting neuroscientific theory that has a lot of potential for robotics control for many reasons: The biologically inspired unification of action and perception is not only elegant, but shows great potential for very natural behaviour even under noisy circumstances. The interest for the topic in the field of robotics is growing and progression is being made, and the Free Energy Principle and Active Inference are already very popular theories in the field of neuroscience. The theory seems very suitable for a translation to robotics, but despite that, a lot more research is required before Active Inference can be applied to robotics with all its benefits.

The contribution of this work is the detailed exploration of aspects of a core strength of Active Inference: the generalised coordinates. The necessary knowledge of Free Energy for the understanding of generalised coordinates have been bundled with a very detailed account of generalised precision and generalised coordinates, together with a simulation scheme for State Space models. This work confirms that Active Inference is a very intriguing, intricate topic that requires far more research effort within the research field of robotics. The topics of generalised precision, obtaining generalised coordinates and preserving the relations between dynamical orders have been researched in detail. This provides some insight into these aspects of Active Inference, but also shows directions for future research. There is definitely a need for more research regarding the validity of some of the assumptions made, such as the local linearity assumption for the generalised models of the G-density (section 2.3), or the assumption that smoothness and variance of coloured noise are generally smaller than 1 (section 3.3.2). Furthermore it is clear from chapters 4 and 5 that the current scenario for control as applied in this work is far from ideal. Improvements can likely be made with more research effort on obtaining generalised coordinates from data. Filtering may improve the results obtained using finite differences, although this research must also include the effects filtering has on the properties (smoothness, variance) of the noise, and if this should be accounted for in the generalised precision matrix. Lastly, if Active Inference is to be applied on-line for robotics control, improvements can be made regarding the closed-loop. The current incompatibility between the agent-provided action and control input to the environment is unfortunate and needs solving. Possibly, with a better description of the closed loop control, classical performance and stability measurements can be applied or fitting measurements can be derived.

All in all, there is a lot to learn and a lot to gain in this very intriguing and intricate biologically inspired topic from neuroscience and hopefully, this work inspires to further the research towards true Artificial Intelligence.

A

Autocorrelation derivatives

This appendix contains a full derivation of the autocorrelation function equation (3.10) *of section 3.2.2 to gain an insight in the structure of the derivatives of this autocorrelation function, which need to be evaluated at* $\tau = 0$ *to learn the values of the entries of the generalised covariance matrix. The derivatives up to and including the 10th are presented, which are required for an embedding order* $p = 5$*. This gives enough insight to generalise the result of derivatives evaluated at* $\tau = 0$ *to obtain values of even higher order derivatives evaluated at* $\tau = 0$*.*

A.1. The derivations

The autocorrelation function in question is stated in equation (3.10) and is repeated below, in equation (A.1).

$$\rho_h(\tau) = e^{-\frac{\tau^2}{4s_w^2}} \tag{A.1}$$

The notation is simplified and some definitions are created in appendix A.1.1 and manual derivations of the autocorrelation function are provided in appendix A.1.2.

A.1.1. Notations and definitions

To keep notation clear and concise, and the derivation easier to read, the following notation and knowledge regarding analytic derivatives is adopted given the autocorrelation function in equation (3.10) and repeated in equation (A.1).

- The subscripts h and w are omitted, such that ρ and s remain.
- The exponential function will be written as e^u, because it remains the same throughout the derivation and $u(\tau) = -\frac{\tau^2}{4s^2}$ is only applied in the chain rule.
- For the application of the chain rule we know $\dot{u}(\tau) = -\frac{\tau}{2s^2}$.
- The denominator $2s^2$ of $\dot{u}(\tau)$ is substituted by d, such that $\dot{u}(\tau) = -\frac{\tau}{d}$.

A.1.2. Analytic derivatives

Below follow the analytic derivatives of the autocorrelation function in equation (A.1) with respect to τ.

$$\rho(\tau) = e^u \tag{A.2}$$

$$\begin{aligned}\dot{\rho}(\tau) &= -\frac{\tau}{d}e^u \\ \ddot{\rho}(\tau) &= \left(-\frac{1}{d}\right)e^u - \left(-\frac{\tau}{d}\right)\frac{\tau}{d}e^u \\ &= \left(-\frac{1}{d} + \frac{\tau^2}{d^2}\right)e^u \end{aligned} \tag{A.3}$$

$$
\begin{aligned}
\ddot{\rho}(\tau) &= \left(\frac{2\tau}{d^2}\right)e^u - \left(-\frac{1}{d} + \frac{\tau^2}{d^2}\right)\frac{\tau}{d}e^u \\
&= \left(\frac{2\tau}{d^2} + \frac{\tau}{d^2} - \frac{\tau^3}{d^3}\right)e^u \\
&= \left(\frac{3\tau}{d^2} - \frac{\tau^3}{d^3}\right)e^u \\
\rho^{(4)}(\tau) &= \left(\frac{3}{d^2} - \frac{3\tau^2}{d^3}\right)e^u - \left(\frac{3\tau}{d^2} - \frac{\tau^3}{d^3}\right)\frac{\tau}{d}e^u \\
&= \left(\frac{3}{d^2} - \frac{3\tau^2}{d^3} - \frac{3\tau^2}{d^3} + \frac{\tau^4}{d^4}\right)e^u \\
&= \left(\frac{3}{d^2} - \frac{6\tau^2}{d^3} + \frac{\tau^4}{d^4}\right)e^u
\end{aligned} \tag{A.4}
$$

$$
\begin{aligned}
\rho^{(5)}(\tau) &= \left(-\frac{12\tau}{d^3} + \frac{4\tau^3}{d^4}\right)e^u - \left(\frac{3}{d^2} - \frac{6\tau^2}{d^3} + \frac{\tau^4}{d^4}\right)\frac{\tau}{d}e^u \\
&= \left(-\frac{12\tau}{d^3} + \frac{4\tau^3}{d^4} - \frac{3\tau}{d^3} + \frac{6\tau^3}{d^4} - \frac{\tau^5}{d^5}\right)e^u \\
&= \left(-\frac{15\tau}{d^3} + \frac{10\tau^3}{d^4} - \frac{\tau^5}{d^5}\right)e^u \\
\rho^{(6)}(\tau) &= \left(-\frac{15}{d^3} + \frac{30\tau^2}{d^4} - \frac{5\tau^4}{d^5}\right)e^u - \left(-\frac{15\tau}{d^3} + \frac{10\tau^3}{d^4} - \frac{\tau^5}{d^5}\right)\frac{\tau}{d}e^u \\
&= \left(-\frac{15}{d^3} + \frac{30\tau^2}{d^4} - \frac{5\tau^4}{d^5} + \frac{15\tau^2}{d^4} - \frac{10\tau^4}{d^5} + \frac{\tau^6}{d^6}\right)e^u \\
&= \left(-\frac{15}{d^3} + \frac{45\tau^2}{d^4} - \frac{15\tau^4}{d^5} + \frac{\tau^6}{d^6}\right)e^u
\end{aligned} \tag{A.5}
$$

$$
\begin{aligned}
\rho^{(7)}(\tau) &= \left(\frac{90\tau}{d^4} - \frac{60\tau^3}{d^5} + \frac{6\tau^5}{d^6}\right)e^u - \left(-\frac{15}{d^3} + \frac{45\tau^2}{d^4} - \frac{15\tau^4}{d^5} + \frac{\tau^6}{d^6}\right)\frac{\tau}{d}e^u \\
&= \left(\frac{90\tau}{d^4} - \frac{60\tau^3}{d^5} + \frac{6\tau^5}{d^6} + \frac{15\tau}{d^4} - \frac{45\tau^3}{d^5} + \frac{15\tau^5}{d^6} - \frac{\tau^7}{d^7}\right)e^u \\
&= \left(\frac{105\tau}{d^4} - \frac{105\tau^3}{d^5} + \frac{21\tau^5}{d^6} - \frac{\tau^7}{d^7}\right)e^u \\
\rho^{(8)}(\tau) &= \left(\frac{105}{d^4} - \frac{315\tau^2}{d^5} + \frac{105\tau^4}{d^6} - \frac{7\tau^6}{d^7}\right)e^u - \left(\frac{105\tau}{d^4} - \frac{105\tau^3}{d^5} + \frac{21\tau^5}{d^6} - \frac{\tau^7}{d^7}\right)\frac{\tau}{d}e^u \\
&= \left(\frac{105}{d^4} - \frac{315\tau^2}{d^5} + \frac{105\tau^4}{d^6} - \frac{7\tau^6}{d^7} - \frac{105\tau^2}{d^5} + \frac{105\tau^4}{d^6} - \frac{21\tau^6}{d^7} + \frac{\tau^8}{d^8}\right)e^u \\
&= \left(\frac{105}{d^4} - \frac{420\tau^2}{d^5} + \frac{210\tau^4}{d^6} - \frac{28\tau^6}{d^7} + \frac{\tau^8}{d^8}\right)e^u
\end{aligned} \tag{A.6}
$$

$$
\begin{aligned}
\rho^{(9)}(\tau) &= \left(-\frac{840\tau}{d^5} + \frac{840\tau^3}{d^6} - \frac{168\tau^5}{d^7} + \frac{8\tau^7}{d^8}\right)e^u \\
&\qquad - \left(\frac{105}{d^4} - \frac{420\tau^2}{d^5} + \frac{210\tau^4}{d^6} - \frac{28\tau^6}{d^7} + \frac{\tau^8}{d^8}\right)\frac{\tau}{d}e^u \\
&= \left(-\frac{840\tau}{d^5} + \frac{840\tau^3}{d^6} - \frac{168\tau^5}{d^7} + \frac{8\tau^7}{d^8} - \frac{105\tau}{d^5} + \frac{420\tau^3}{d^6} - \frac{210\tau^5}{d^7} + \frac{28\tau^7}{d^8}\right. \\
&\qquad \left. - \frac{\tau^9}{d^9}\right)e^u
\end{aligned}
$$

$$
\begin{aligned}
&= \left(-\frac{945\tau}{d^5} + \frac{1260\tau^3}{d^6} - \frac{378\tau^5}{d^7} + \frac{36\tau^7}{d^8} - \frac{\tau^9}{d^9}\right)e^u \\
\rho^{(10)}(\tau) &= \left(-\frac{945}{d^5} + \frac{3780\tau^2}{d^6} - \frac{1890\tau^4}{d^7} + \frac{252\tau^6}{d^8} - \frac{9\tau^8}{d^9}\right)e^u \\
&\quad - \left(-\frac{945\tau}{d^5} + \frac{1260\tau^3}{d^6} - \frac{378\tau^5}{d^7} + \frac{36\tau^7}{d^8} - \frac{\tau^9}{d^9}\right)\frac{\tau}{d}e^u \\
&= \left(-\frac{945}{d^5} + \frac{3780\tau^2}{d^6} - \frac{1890\tau^4}{d^7} + \frac{252\tau^6}{d^8} - \frac{9\tau^8}{d^9} + \frac{945\tau^2}{d^6} - \frac{1260\tau^4}{d^7} + \frac{378\tau^6}{d^8}\right. \\
&\quad \left. - \frac{36\tau^8}{d^9} + \frac{\tau^{10}}{d^{10}}\right)e^u \\
&= \left(-\frac{945}{d^5} + \frac{4725\tau^2}{d^6} - \frac{3150\tau^4}{d^7} + \frac{630\tau^6}{d^8} - \frac{45\tau^8}{d^9} + \frac{\tau^{10}}{d^{10}}\right)e^u
\end{aligned}
\tag{A.7}
$$

These derivatives of the autocorrelation function must be evaluated at $\tau = 0$, as stated in section 3.2.2. This yields 0 for all odd derivatives and the following expressions for the even derivatives (from equations (A.2) to (A.7)):

$$
\begin{aligned}
\rho(0) &= 1 & \rho^{(6)}(0) &= -\frac{15}{d^3} \\
\ddot{\rho}(0) &= -\frac{1}{d} & \rho^{(8)}(0) &= \frac{105}{d^4} \\
\rho^{(4)}(0) &= \frac{3}{d^2} & \rho^{(10)}(0) &= -\frac{945}{d^5}
\end{aligned}
$$

A.2. Evaluation

From these evaluated derivatives it is clear that there is a lot of structure to them, which is a direct result of the structure of the autocorrelation function and the resulting application of chain and product rule in differentiation. The derivatives up to and including the 10th are now known, but it is relevant to generalise this result such that one can find the zero-lag evaluation of any derivative of the autocorrelation function. Also, the expressions for the derivative functions themselves are not of interest, but merely the terms that remain when $\tau = 0$. Therefore, an analysis of the results follows:

- Every derivative function comes in the form of $a(\tau)e^u(\tau)$, with $u(\tau)$ as defined earlier in appendix A.1.1. $a(\tau)$ is always a polynomial in τ, with powers of τ in the numerators and powers of s in the denominators.
- Every derivative is a result of a combination of the chain and product rules. Both a and u are functions of τ and as such, the product rule causes a set of terms that is differentiated: the terms go down one power in τ. On the other hand, the chain rule causes all terms to get multiplied with $\dot{u}(\tau)$, which causes them to go up one power in τ. The chain rule also causes terms to switch sign upon every differentiation.
- Because of the chain rule, the denominators are always multiplied with $d = 2s^2$, causing the power of d to be raised by one or equivalently, the constant multiplication of $2s^2$ to be doubled and their power to be raised by 2.
- All the odd derivatives contain only odd powers of τ, hence they are zero when $\tau = 0$. Even derivatives contain even powers of τ, and the constant terms remain when $\tau = 0$.

B

Matlab scripts

This appendix contains all the Matlab *scripts created for this research. Parts of some scripts have been included into the chapters for illustration purposes. The full scripts are included here. All the code has been run in* Matlab *R2018a, which by default uses the 'Mersenne Twister' for random sequences*[1]*. This is relevant for the reproduction of the results in this work. With the scripts in this appendix, all results in the report can be reproduced. All scripts are equipped with explanatory comments and a description of all involved parameters.*

B.1. Coloured noise & generalised precision

This section presents two scripts, both defining a function for coloured noise and generalised precision, which are the topic of chapter 3.

B.1.1. Coloured noise generator

This script in listing B.1 defines the function `f_colourednoise` given parameters regarding white noise and a Gaussian filter, as well as sampling parameters and an optional random seed for reproducibility. It generates coloured noise as described in section 3.1.1.

Listing B.1: The Matlab function `f_colourednoise`.

```
%% Coloured noise generator
%{

Iris Hijne
August 2020

INFO_____________________________________________________________________
This function creates a white noise signal which is then filtered by a
gaussian filter, given the white noise standard deviation, the filter's
standard deviation, the sample time and end time of the signal, and
optionally a random seed.

INPUTS___________________________________________________________________
sigma [1xn]: the sd of the white noise
s     [1x1]: the sd of the filter
dt    [1x1]: the sample time
T     [1x1]: the end time of the time signal
seed  [1x1]: [optional] the desired random seed

```

[1] https://nl.mathworks.com/help/matlab/ref/rng.html

```
OUTPUTS_____________________________________________________________________
w [nxN]: The coloured noise signal

%}

%%
function w = f_colourednoise(sigma,s,dt,T,varargin)

if nargin > 4 && ~isempty(varargin{1})
    rng(varargin{1}) % choose the random seed
end

n = length(sigma);
N = length(0:dt:T);

omega = sigma'.*randn(n,N); % compute the white noise signal

tau = -T:dt:T; % width of the filter
h = sqrt(dt/(s*sqrt(pi))).*exp(-(tau).^2./(2*s^2)); % Gaussian filter

w = zeros(n,max(length(h)-N+1,0)); % preallocate w
for i = 1:n % Convolve
    w(i,:) = conv(h,omega(i,:),'valid');
end
```

B.1.2. Generalised precision matrix

The script presented in listing B.2 defines `f_precision`, which generates a generalised precision matrix $\tilde{\Pi}$ as in equation (3.15) given only the variance and smoothness of the noise, and an embedding order to know the size of the matrix. This matrix is very easy to construct given the derivation in chapter 3, where part of this script is presented (listing 3.1) for illustration purposes.

Listing B.2: The Matlab function `f_precision`.

```
%% Generalised precision matrix
%{

Iris Hijne
August 2020

INFO________________________________________________________________________
This function computes the generalised precision matrix of a coloured
noise that is a gaussian-filtered white noise. This is the generalised
precision matrix as used in the Free Energy function as used in Active
Inference.

INPUTS______________________________________________________________________
s     [1x1]: the sd of the filter
sigma [1xn]: the sd of the white noises, n = number of noise signals
p     [1x1]: the embedding order

OUTPUTS_____________________________________________________________________
Pi_ [npp x npp]: The generalised precision matrix

%}

%%
```

```
function Pi_ = f_precision(s,sigma,p)

pp = p+1; % embedding order raised by one, for convenience

k = 0:2:2*p; % order of the required autocorrelation derivatives
rho(1+k) = cumprod(1-k)./((sqrt(2).*s).^k); % rho^(k)(0)

S = zeros(pp,pp); % preallocation of the temporal variance matrix
for r = 1:pp % One row for every embedding order
    S(r,:) = rho(r:r+p); % assembly of the temporal variance matrix
    rho = -rho; % minus signs change every row
end

Pi_ = kron(inv(S),inv(diag(sigma.^2))); % generalised covariance matrix
```

B.2. Finite differences

In this section are the scripts regarding the finite differences in chapter 4, one function to create the matrix E and one function to compute approximated derivatives using the matrix E. A script for the tests in section 4.2 that relies on these functions is also provided.

B.2.1. Finite difference matrix E

The script in listing B.3 defines the function `f_finitediffmat` which generates the matrix E as in equation (4.11) in chapter 4. In listing 4.1 the script without explanatory comments is displayed.

Listing B.3: The Matlab function `f_finitediffmat`.

```
%% Finite difference matrix
%{

Iris Hijne
August 2020

INFO_____________________________________________________________________
This function generates a finite difference matrix that approximates the
derivatives of a signal when multiplied by an array of surrounding values,
using Taylor Expansion with a forward, central  or backward difference.

INPUTS___________________________________________________________________
dt     [1xn]: the sampling time [s] (>0)
p      [1x1]: the embedding order (int>=1) (number of derivatives)
o      [1x1]: the error order (int>=1)
n      [1x1]: the signal dimension (int>=1)
method [str]: the method: 'f' forward, 'c' central, 'b' backward

OUTPUTS__________________________________________________________________
E [n(p+1) x n(p+o(+1))]: The finite difference matrix

%}

%%
function E = f_finitediffmat(dt,p,o,n,method)

pp = p+1; % number of rows in the matrix E for a 1-dim signal
switch method
% s: # samples required for the approx. of all desired derivatives
% E1: preallocation of matrix E for a 1-dim signal
```

```
% E1(,)=1: prepare first row of E1 to pass y onto itself in y_
    case 'f'
        s = p+o;
        if p==0; E1 = 1;
        else; E1 = zeros(pp,s); E1(1,1) = 1;
        end
    case 'c'
        if mod(p+o,2) == 0 % when p+o is even, o is increased by 1
            s = p+o+1;
        else
            s = p+o;
        end
        if p==0; E1 = 1;
        else; E1 = zeros(pp,s); E1(1,ceil(s/2)) = 1;
        end
    case 'b'
        s = p+o;
        if p==0; E1 = 1;
        else; E1 = zeros(pp,s); E1(1,end) = 1;
        end
end
C = zeros(1,s); % preallocation of array w/ coef for finite differences

for d = 1:p % we visit all p-values so we have all the derivatives
    switch method
    % sd: # samples required for the current derivative
    % jmin, jmax: required finite differences around the point of interest
    % imax: total number of samples (-1) required for the approximation
    % sumij: preallocation of the matrix for computation of C
    % sumijC: array with outcomes of the sums in sumij
        case 'f'
            sd = d+o;
            jmin = 0; jmax = sd-1;
            imax = sd-1;
            sumij = zeros(sd,sd);
            sumijC = zeros(size(sumij,2),1);
        case 'c'
            if mod(d+o,2) == 0 % when d+o is even, o is increased by 1
                sd = d+o+1;
            else
                sd = d+o;
            end
            jmax = (sd-1)/2; jmin = -jmax;
            imax = sd-1;
            sumij = zeros(sd,sd);
            sumijC = zeros(size(sumij,2),1);
        case 'b'
            sd = d+o;
            jmin = -(sd-1); jmax = 0;
            imax = sd-1;
            sumij = zeros(sd,sd);
            sumijC = zeros(size(sumij,2),1);
    end
    sumijC(d+1) = 1; % the sum must be 1 for j = d, 0 otherwise
    jrange = jmin:jmax; % range of all the finite difference elements
    for i = 0:imax % filling the matrix w/ elements to sum
```

```
        sumij(i+1,:) = jrange.^i;
    end
    switch method % computing C (solving the linear system)
        case 'f'
            C(1:sd) = (sumij\sumijC)';
        case 'c'
            C((s-sd)/2+1:s-(s-sd)/2) = (sumij\sumijC)';
        case 'b'
            C(s-sd+1:end) = (sumij\sumijC)';
    end
    E1(d+1,:) = (factorial(d)/dt^d).*C; % adding the elements to E
end

E = kron(E1,eye(n)); % E for an n-dim signal
```

B.2.2. Derivatives by finite differences

In listing B.4 is the script with the function definition for `f_finitediff`, which computes generalised coordinates based on the input data. It uses `f_finitediffmat` from listing B.3 to compute the E-matrix. This function is used for the tests in section 4.2 by means of the script in listing B.5.

Listing B.4: The Matlab function `f_finitediff`.

```
%% Backwards difference derivative computation
%{

Iris Hijne
August 2020

INFO_____________________________________________________________________
This computes the derivatives of a signal by means of a backwards
difference given the original signal, the number of desired derivatives,
the order of error of the derivative calculation and the sample time.

INPUTS___________________________________________________________________
y       [nxN]: the original signal
p       [1x1]: the embedding order (#derivatives)
o       [1x1]: the order of error of the backwards difference (O(dt^o))
dt      [1x1]: the sample time of the original signal
method [str]: the method: 'f' forward, 'c' central, 'b' backward

OUTPUTS__________________________________________________________________
y_ [nppxN]: The generalised signal

%}

%%
function y_ = f_finitediff(y,dt,p,o,method)

% Dimensions
q = size(y,1);
N = size(y,2);
pp = p+1;

% Derivative preallocation
y_ = zeros(q*pp,N);

```

```
% Creating the E-matrix
switch method
    case 'b'; E = f_finitediffmat(dt,p,o,q,'b');
    case 'f'; E = f_finitediffmat(dt,p,o,q,'f');
        pmax = p; ppmax = pmax+1;
        % Forward method only changes pmax at final samples
    case 'c'; E = f_finitediffmat(dt,p,o,q,'c');
end
ns = size(E,2)/q; % number of time-samples in yv

% Computing the derivatives
for i = 1:N
    switch method
        case 'b'
            if i <= ns % The first few samples, when not enough available
                pmax = min(max(0,i-o),p); ppmax = pmax+1; % max at sample
                E = f_finitediffmat(dt,pmax,o,q,'b'); % Get proper size E
            end
            if pmax == 0; yv = y(:,i);
            else; yv = reshape(y(:,i-pmax-o+1:i),[],1); % Stack samples
            end
        case 'f'
            if i > N-ns+1 % The last samples, when not enough available
                pmax = min(max(0,N-i-o+1),p); ppmax = pmax+1; % at sample
                E = f_finitediffmat(dt,pmax,o,q,'f'); % Get proper size E
            end
            if pmax == 0; yv = y(:,i);
            else; yv = reshape(y(:,i:i+pmax+o-1),[],1); % Stack samples
            end
        case 'c'
            if i <= ceil(ns/2) || i > N-ceil(ns/2)+1 % first/last samples
                pmax = min(min(max(2*i-1-o,0),p),...
                    min(max(2*(N-i)+1-o,0),p)); ppmax = pmax+1;
                E = f_finitediffmat(dt,pmax,o,q,'c'); % Get proper size E
            end
            if pmax == 0; yv = y(:,i);
            elseif mod(pmax+o,2) == 0 % when p+o even
                yv = reshape(y(i-(pmax+o)/2:i+(pmax+o)/2),[],1);
            else % when p+o odd
                yv = reshape(y(i-(pmax+o-1)/2:i+(pmax+o-1)/2),[],1);
            end
    end
    y_(1:ppmax*q,i) = E*yv; % Compute the derivatives
end
```

B.2.3. Finite difference testing

Listing B.5 provides a script that has been used to perform the tests in section 4.2. In it the analytic function is defined, it is sampled and the derivatives are computed using the functions in listings B.3 and B.4, after which performance measurements are computed and results are plotted.

Listing B.5: A Matlab script to perform tests with finite differences.

```
%% Finite differences derivative estimation test
%{

Iris Hijne
```

```
August 2020

INFO_____________________________________________________________________
This script tests the functionality of the E-matrix to compute derivatives
by finite differences. An analytical function is chosen such that a ground
truth of derivatives is available and the samples of the analytical
function serve as the signal to compute the derivatives of. The results
are plotted and errors are computed.
%}

clearvars
close all

%% Symbolic function and derivatives

% The symbolic function
syms x
yref = sin(x) + 3*cos(0.1*x) + 0.01*x^3;

% Desired embedding order and order of estimation error
p = 6; pp = p+1;
o = 1;

% Analytic derivatives of the symbolic function
yref_ = sym(zeros(pp,1));
yref_(1) = yref;
for i = 1:p
    yref_(i+1) = diff(yref_(i));
end

%% The sampled signal

% Time signal and sampled function
dt = 0.01; T = 10;
t = 0:dt:T; N = length(t);
y_ = zeros(pp,N);
for i = 1:pp
    y_(i,:) = double(subs(yref_(i),x,t));
end

% Estimated derivatives by means of E
yb_ = f_finitediff(y_(1,:),dt,p,o,'b');
yf_ = f_finitediff(y_(1,:),dt,p,o,'f');
yc_ = f_finitediff(y_(1,:),dt,p,o,'c');

%% Plots

figure('Name','Derivative test','NumberTitle','off');
hold on
for i = 1:pp
    plot(t,y_(i,:))
end
ax = gca; ax.ColorOrderIndex = 1;
for i = 1:pp
    plot(t,yb_(i,:),'--');
end
```

```
ax = gca; ax.ColorOrderIndex = 1;
for i = 1:pp
    plot(t,yf_(i,:),':');
end
ax = gca; ax.ColorOrderIndex = 1;
for i = 1:pp
    plot(t,yc_(i,:),'-.');
end

legendstrings = [];
for i = 1:pp
    legendstrings = [legendstrings; strcat('y^{(',num2str(i-1),')}')];
end
legend(legendstrings)
xlabel('time [s]')

%% Performance measurements

% Samples to skip
skip = p+o;

% Mean squared error
MSEb = mean((yb_(:,skip:N-skip+1)-y_(:,skip:N-skip+1)).^2,2);
MSEf = mean((yf_(:,skip:N-skip+1)-y_(:,skip:N-skip+1)).^2,2);
MSEc = mean((yc_(:,skip:N-skip+1)-y_(:,skip:N-skip+1)).^2,2);

disp('MSE of b-, f-, c-difference');
disp([MSEb(2:end) MSEf(2:end) MSEc(2:end)])

%% Noise

sigma = 0.001;
s = 0.5;
z = f_colourednoise(sigma,s,dt,T);

y_noise = y_(1,:)+z;

% Estimated derivatives by means of E
yb_noise = f_finitediff(y_noise,dt,p,o,'b');
yf_noise = f_finitediff(y_noise,dt,p,o,'f');
yc_noise = f_finitediff(y_noise,dt,p,o,'c');

%% Plots

figure('Name','Derivative test with noise','NumberTitle','off');
hold on
for i = 1:pp
    plot(t,y_(i,:))
end
ax = gca; ax.ColorOrderIndex = 1;
for i = 1:pp
    plot(t,yb_noise(i,:),'--');
end
ax = gca; ax.ColorOrderIndex = 1;
for i = 1:pp
    plot(t,yf_noise(i,:),':');
```

```
end
ax = gca; ax.ColorOrderIndex = 1;
for i = 1:pp
    plot(t,yc_noise(i,:),'-.');
end

legendstrings = [];
for i = 1:pp
    legendstrings = [legendstrings; strcat('y^{(',num2str(i-1),')}')];
end
legend(legendstrings)
xlabel('time [s]')
```

B.2.4. Finite difference evaluation

The function in listing B.6 takes data and computes generalised coordinates based on this data, then plots both the data and the recomputed derivatives by finite differences to compare them against generalised data from a simulation loop.

Listing B.6: The Matlab function `f_diffcheck`.

```
%% Derivative check
%{

Iris Hijne
August 2020

INFO_____________________________________________________________________
This function takes a generalised signal from a simulation in generalised
coordinates and then uses the finite difference method to recompute the
derivatives from the reference signal (one of the generalised orders, so
0<= ref <=p). The data together with the recomputed derivatives are
plotted for comparison.

NB ref means: From which derivative in data (eg y) should the rest of the
derivatives be computed.
if ref=0, then y is used to compute all the derivatives using E.
if ref=1, then y' is used to compute the derivatives y'' and onward using
E, and so on.

NB start is a lazy way to get rid of the peaks at the beginning that I'm
aware of. By getting rid of it, the y-axis scaling of the plots are such
that I can see the rest of the signal better.

INPUTS___________________________________________________________________
data [nppxN]: the generalised data from a simulation
p    [1x1]: the embedding order
o    [1x1]: the accuracy order of the derivatives (dt^o)
ref  [1x1]: the order of derivative of the reference signal (0<=ref<=p)
dt   [1x1]: the sample time
T    [1x1]: the simulation time
start[1x1]: the number of the sample to start plotting from

OUTPUTS__________________________________________________________________
none

%}
```

```
37
38  %%
39  function f_diffcheck(data,p,o,ref,dt,T,start)
40
41  % Derivative computation
42  y_ = f_backdiff(data(ref+1,:),p-ref,o,dt);
43
44  % Setup for plots
45  pp = p+1;
46  time = 0:dt:T;
47
48  % Comparison plot
49  figure('Name','Derivative checks','NumberTitle','off');
50  for i = 1:pp
51      subplot(pp,1,i)
52      hold on; ax = gca; ax.ColorOrderIndex = 1;
53      plot(time(start:end),data(i,start:end),'LineWidth',2)
54      if i > ref
55          plot(time(start:end),y_(i-ref,start:end),'LineWidth',2)
56      end
57      if i==ref+1
58          legend('data','recomputed derivative')
59      end
60      str = strcat('y^{(',num2str(i-1),')}');
61      ylabel(str)
62  end
63  xlabel('time [s]')
```

B.3. Generalised forward model

The generalised forward model in equation (5.7) from section 5.2 has a very clear structure to it that can easily be assembled using the Matlab function in listing B.7.

Listing B.7: The Matlab function `f_genforwardmodel`.

```
1   %% Generalised forward model
2   %{
3
4   Iris Hijne
5   August 2020
6
7   INFO_____________________________________________________________________
8   This function generates generalised forward model (G_ or G_tilde)
9
10  INPUTS___________________________________________________________________
11  A [nxn]: the state transition matrix
12  B [nxl]: the input matrix
13  C [1x1]: the output matrix
14  D [1x1]: the feedthrough matrix
15  p [str]: the embedding order (int>=0) (number of derivatives)
16
17  OUTPUTS__________________________________________________________________
18  G_ [q(p+1) x l(p+1)]: The generalised forward model
19
20  %}
21
22  %%
```

```
function G_ = f_genforwardmodel(A,B,C,D,p)

l = size(B,2);
q = size(C,1);

pp = p+1;

G_ = zeros(q*pp,l*pp);
Garray = zeros(q,l*pp);

for i = 1:pp
    Garray(:,(i-1)*l+1:i*l) = -C*A^(-i)*B + D;
end

for i = 1:pp
    G_((i-1)*q+1:i*q,(i-1)*l+1:end) = Garray(:,1:end-(i-1)*l);
end
```

B.4. Simulation

The simulations in sections 5.4 and 5.5 have been performed using the Matlab scripts in this section, which are the script that sets up the simulation and a script with a function that is the simulation. The simulation in appendix B.4.2, relies on all the functions defined before (listings B.1 to B.4 and B.7) plus additional functions defined in listings B.10 to B.12 from appendices B.4.3 and B.4.4.

B.4.1. Parameter script

In listing B.8 is the parameter script that defines all the simulation parameters. It is not necessary, since the function `f_sim` (listing B.9) can be called directly, but it is convenient and shows how all the parameters are set. Parameters can be changed as desired and running the script runs the whole simulation.

Listing B.8: A Matlab script to setup and run simulations.

```
%% SISO-system with 1 DOF in generalised coordinates

%{
Robotics control by means of Active Inference and the Free Energy
Principle using time-series for generalised coordinates

Iris Hijne
August 2020
%}

%{
A 1-DOF SISO system (a point mass with a velocity and an input force) is
used as a simple test case. (Although the script should work with a MIMO
system, but this is not tested). This script is used to set parameters and
then run a simulation by forward Euler method of an LTI State Space system
using the Active Inference framework.
%}

close all
clearvars

%% Simulation method

% 0: The plant is not generalised, the generalised output is computed by
```

```
%   means of Taylor series on the plant output signal. The method of
%   differentiation is always by backwards differences. The embedding
%   order is increased in the first few iterations as more samples become
%   available.
% 1: The plant is generalised, which is done by computing generalised
%   noise before simulation and simulating a generalised plant, which
%   yields a generalised output. The generalised noise is made from a
%   noise signal that has more samples than required, such that enough
%   data is available to compute all the required derivatives.
%   Differentiation can be done by backward, forward or central
%   differences.
% 2: The plant is generalised as in 1, but the generalised noise is made
%   without the availability of additional samples, just as would be the
%   case in 0. The method of differentiation is always by backwards
%   differences.
% 3: As 2, but the embedding order is increased in the first few
%   iterations as more non-zero information from the noise is available,
%   just like 0.

gp = 3;

%% The generative process

% Parameters
m = 1; %kg
d = 1; %N/(m/s)

% System matrices
A = -d/m;
B = 1/m;
C = 1;

% State reference
xref = 2; %m/s

% Initial values
x0 = 0; %m/s
u0 = 0; %N

% Noise parameters
sigma_w = 0.05; % standard deviations of each process noise
s_w = 0.1; % standard deviation of the filter
sigma_z = 0.05; % standard deviation of each observation noise
s_z = 0.1; % standard deviation of the filter

seed_w = 4; % random seed for process noise. may also be empty []
seed_z = 6; % random seed for observation noise. may also be empty []

% Derivative calculation (only relevant if gp = 1. Otherwise ignored)
diffmethod = 'b'; % b, c or f
o = 1; % Derivative accuracy order (dt^o)

%% The agent

% The embedding order
p = 10;
```

```
81
82  % Learning rates
83  alpha_mu = 0.2;
84  alpha_u = 0.1;
85
86  %% Time signal
87
88  dt = 0.001; %s
89  T = 5; %s
90
91  %% Run the simulation
92
93  par.genproc = {A,B,C,gp};
94  par.init = {xref,x0,u0};
95  par.noise = {sigma_w,s_w,seed_w;sigma_z,s_z,seed_z};
96  par.taylor = {diffmethod,o};
97  par.agent = {p,alpha_mu,alpha_u};
98  par.time = {dt,T};
99
100 [states,beliefs,inputs,outputs] = f_sim(par);
```

B.4.2. Simulation program

The script in listing B.9 defines the simulation function `f_sim`, which runs a full simulation, relying on all other functions listed in this appendix B. It takes the parameters from the parameter script in listing B.8, then prepares the simulation and runs a simulation loop, afterwards plotting the results.

Listing B.9: The Matlab function `f_sim`.

```
1   %% AI Simulation
2   %{
3
4   Iris Hijne
5   August 2020
6
7   INFO_____________________________________________________________________
8   This function performs a simulation of an Active Inference controlled
9   system given all the required parameters
10
11  INPUTS___________________________________________________________________
12  struct par: struct with all parameters
13
14  par.genproc: the generative process (state space)
15      A  : the state transition matrix
16      B  : the input matrix
17      C  : the output matrix
18      gp : 0,1,2,3 stating if the generative process is to be generalised:
19               0: no generalised plant. Output is generalised in simulation
20               1: generalised plant. Noise is generalised with ample samples
21               2: generalised plant. Noise is generalised with limited
22                   samples, like output is generalised in case 0.
23               3: generalised plant. Like case 2, but embedding order starts
24                   at 0 instead of p.
25
26  par.init: the reference signal and initial values
27      xref : the state reference value(s)
28      x0   : the initial state
```

```
    u0   : the initial input

par.noise: the parameters for generating the noise
    ROW 1: PROCESS NOISE
    sigma_w : array with the standard deviations of the noise signal (one
        for each state)
    s_w     : the standard deviation of the Gaussian filter for smoothing
    seed_w  : the seed for the random generator

    ROW 2: OBSERVATION NOISE
    sigma_z : array with the standard deviations of the noise signal (one
        for each output)
    s_z     : the standard deviation of the Gaussian filter for smoothing
    seed_z  : the seed for the random generator

par.taylor: the parameters for choosing how to compute the derivatives of
        signals.
    diffmethod : 'b', 'c' or 'f'
    o          : the order of error (dt^o) of derivative computation

par.agent: the tuning parameters of the agent
    p        : the embedding order (number of generalised coordinates)
    alpha_mu : learning rate of state estimation
    alpha_u  : learning rate of control action

par.time: the parameters that define the time signal
    dt : the sample time [s]
    T  : the total simulation time [s]

OUTPUTS_____________________________________________________________________
states: x (gp=0) or x_ (gp=1,2,3)
beliefs: mu
inputs: u_
outputs: y_

%}

function [states,beliefs,inputs,outputs] = f_sim(par)

% Generative process
A = par.genproc{1};
B = par.genproc{2};
C = par.genproc{3};
gp = par.genproc{4};

% Initial values and reference
xref = par.init{1};
x0 = par.init{2};
u0 = par.init{3};

% Noise parameters
sigma_w = par.noise{1,1};
s_w = par.noise{1,2};
seed_w = par.noise{1,3};

sigma_z = par.noise{2,1};
```

```
s_z = par.noise{2,2};
seed_z = par.noise{2,3};

% Signal differentiation
diffmethod = par.taylor{1};
o = par.taylor{2};

% Agent parameters
p = par.agent{1};
alpha_mu = par.agent{2};
alpha_u = par.agent{3};

% Time signal
dt = par.time{1};
T = par.time{2};

%% Setting up the system

% Time signal
time = 0:dt:T; %s

% Dimensions
N = length(time);
n = size(A,1);
l = size(B,2);
q = size(C,1);
pp = p+1; % for convenience

% Initial generalised state
mu0 = zeros(n*pp,1);
mu0(1:n) = x0;

% Reference signal and prior
muref = zeros(n*pp,1); muref(1:n) = xref;
D = kron(triu(ones(pp),1)-triu(ones(pp),2),eye(n));
A_ = kron(eye(pp),A);
xi = D*muref-A_*muref; % xi = muref;

%% Noise signals

% COLOURED NOISE
switch gp
    case {0,2,3}
        w = f_colourednoise(sigma_w,s_w,dt,T,seed_w); % Process noise
        z = f_colourednoise(sigma_z,s_z,dt,T,seed_z); % Observation noise
    case 1
        % Process noise
        Ew = f_finitediffmat(dt,p,o,n,diffmethod); % The E-matrix
        nsw = size(Ew,2)/n; % Number of samples required in time-series
        w = f_colourednoise(sigma_w,s_w,dt,T+(nsw-1)*dt,seed_w);

        % Observation noise
        Ez = f_finitediffmat(dt,p,o,q,diffmethod); % The E-matrix
        nsz = size(Ez,2)/q; % Number of samples required in time-series
        z = f_colourednoise(sigma_z,s_z,dt,T+(nsz-1)*dt,seed_z);
end
```

```
141
142 % GENERALISED NOISE
143 switch gp
144     case 1
145         w_ = zeros(n*pp,N); % prealloc of generalised process noise
146         z_ = zeros(n*pp,N); % prealloc of generalised observation noise
147         wv = [zeros(n,1); reshape(w(:,1:nsw-1),[n*(nsw-1),1])]; ...
148             % Creating the initial time-series array for w_
149         zv = [zeros(q,1); reshape(z(:,1:nsz-1),[q*(nsz-1),1])]; ...
150             % Creating the initial time-series array for z_
151         for i = 1:N % Computing the derivatives
152             wv = [wv(n+1:end); w(:,i+nsw-1)]; % Shift time-series array
153             w_(:,i) = Ew*wv; % Compute the derivatives
154             zv = [zv(q+1:end); z(:,i+nsz-1)]; % Shift time-series array
155             z_(:,i) = Ez*zv; % Compute the derivatives
156         end
157
158     case {2,3}
159         w_ = f_backdiff(w,p,o,dt); % Process noise
160         z_ = f_backdiff(z,p,o,dt); % Observation noise
161 end
162
163 % NOISE PLOTS
164 figure('Name','Noise signals','NumberTitle','off');
165
166 switch gp
167     case 0
168         hold on;
169         plot(time,w(1,:),'LineWidth',2)
170         plot(time,z(1,:),'LineWidth',2)
171         ylabel('Noise')
172         legend('w','z')
173
174     case {1,2,3}
175         for i = 1:pp
176             subplot(pp,1,i)
177             hold on; ax = gca; ax.ColorOrderIndex = 1;
178             plot(time,w_(i,:),'LineWidth',2)
179             plot(time,z_(i,:),'LineWidth',2)
180             str = strcat('noise^{(',num2str(i-1),')}');
181             ylabel(str)
182             if i==1
183                 legend('w','z')
184             end
185         end
186 end
187 xlabel('time [s]')
188
189 %% Simulation
190
191 % PREALLOCATION & INITIALIZATION
192 switch gp
193     case 0
194         x = zeros(n,N); x(:,1) = x0;
195         yv = [];
196         Ey = f_finitediffmat(dt,p,o,q,'b');
```

```
        ns = size(Ey,2);
    case {1,2,3}
        x_ = zeros(n*pp,N); x_(1:length(x0),1) = x0;
end
u_ = zeros(l*pp,N); u_(1:length(u0),1) = u0;
mu = zeros(n*pp,N); mu(1:length(mu0),1) = mu0;
y_ = zeros(q*pp,N);

% SIMULATION LOOP
switch gp
    case 0
        maxsystemsize = 0; % To prevent unnecessary calculations later
        for i = 1:N
            [xdot,y_(1:q,i)] = ...
                f_plantupdate(x(:,i),u_(1:l,i),A,B,C,w(:,i),z(:,i));
            if i ~= N; x(:,i+1) = x(:,i)+xdot*dt; end % Skip x(N+1)
            pmax = min(max(0,i-o),p); ppmax = pmax+1;
            if length(yv) < ns
                yv = [yv; y_(1:q,i)];
                Ey = f_finitediffmat(dt,pmax,o,q,'b');
            else
                yv = [yv(q+1:end); y_(1:q,i)];
            end
            y_(1:ppmax*q,i) = Ey*yv;
            if length(yv) <= ns && ~maxsystemsize
                A_ = kron(eye(ppmax),A);
                C_ = kron(eye(ppmax),C);
                G_ = f_genforwardmodel(A,B,C,zeros(q,l),pmax);
                D = kron(triu(ones(ppmax),1)-triu(ones(ppmax),2),eye(n));
                xi = D*muref(1:n*ppmax)-A_*muref(1:n*ppmax);
                Pi_w_ = f_precision(s_w,sigma_w,pmax);
                Pi_z_ = f_precision(s_z,sigma_z,pmax);
                if length(yv) == ns; maxsystemsize = 1; end
            end
            [mudot,udot] = f_agentupdate(mu(1:ppmax*n,i),...
                y_(1:ppmax*q,i),A_,C_,D,G_,xi,Pi_w_,Pi_z_,...
                alpha_mu,alpha_u);
            if i ~= N
                mu(1:ppmax*n,i+1) = mu(1:ppmax*n,i)+mudot*dt;
                u_(1:ppmax*l,i+1) = u_(1:ppmax*l,i)+udot*dt;
            end
        end

    case {1,2}
        % Generalised system matrices
        A_ = kron(eye(pp),A);
        B_ = kron(eye(pp),B);
        C_ = kron(eye(pp),C);
        % The forward model
        G_ = f_genforwardmodel(A,B,C,zeros(q,l),p);

        % The differentiator matrix (shift mu)
        D = kron(triu(ones(pp),1)-triu(ones(pp),2),eye(n));
        Pi_w_ = f_precision(s_w,sigma_w,p); % Gen precision mat w
        Pi_z_ = f_precision(s_z,sigma_z,p); % Gen precision mat z
        for i = 1:N
```

```
            [x_dot,y_(:,i)] = f_plantupdate(x_(:,i),u_(:,i),...
            A_,B_,C_,w_(:,i),z_(:,i));
            if i ~= N; x_(:,i+1) = x_(:,i)+x_dot*dt; end % Skip x(N+1)
            [mudot,udot] = f_agentupdate(mu(:,i),y_(:,i),...
                A_,C_,D,G_,xi,Pi_w_,Pi_z_,alpha_mu,alpha_u);
            if i ~= N
                mu(:,i+1) = mu(:,i)+mudot*dt;
                u_(:,i+1) = u_(:,i)+udot*dt;
            end
        end

    case 3
        for i = 1:N
            pmax = min(max(0,i-o),p); ppmax = pmax+1;
            if sum(w_(:,i) ~= 0) <= size(f_finitediffmat(dt,p,o,n,'b'),2)
                A_ = kron(eye(ppmax),A);
                B_ = kron(eye(ppmax),B);
                C_ = kron(eye(ppmax),C);
                G_ = f_genforwardmodel(A,B,C,zeros(q,l),pmax);
                D = kron(triu(ones(ppmax),1)-triu(ones(ppmax),2),eye(n));
                xi = D*muref(1:n*ppmax)-A_*muref(1:n*ppmax);
                Pi_w_ = f_precision(s_w,sigma_w,pmax);
                Pi_z_ = f_precision(s_z,sigma_z,pmax);
            end
            [xdot,y_(1:ppmax*q,i)] = f_plantupdate(x_(1:ppmax*n,i),...
                u_(1:ppmax*l,i),A_,B_,C_,w_(1:ppmax*n,i),z_(1:ppmax*q,i));
            [mudot,udot] = f_agentupdate(mu(1:ppmax*n,i),...
                y_(1:ppmax*q,i),A_,C_,D,G_,xi,Pi_w_,Pi_z_,...
                alpha_mu,alpha_u);
            if i ~= N
                x_(1:ppmax*n,i+1) = x_(1:ppmax*n,i)+xdot*dt;
                mu(1:ppmax*n,i+1) = mu(1:ppmax*n,i)+mudot*dt;
                u_(1:ppmax*l,i+1) = u_(1:ppmax*l,i)+udot*dt;
            end
        end
end

%% Simulation results

% MSE of \mu and x
switch gp
    case 0; MSE_mu_x = sum(mu(1:n,:)-x,2).^2;
    case {1,2,3}; MSE_mu_x = sum(mu-x_,2).^2;
end
disp('MSE \mu and x: ');
disp(MSE_mu_x(1:n));
F = f_freeenergy(mu,y_,A_,C_,D,xi,Pi_w_,Pi_z_);

% Plots
figure('Name','Simulation','NumberTitle','off');

subplot(3,1,1)
hold on; ax = gca; ax.ColorOrderIndex = 1;
plot([time(1) time(end)],xref.*[1 1],'k')
switch gp
    case 0; p1 = plot(time,x,'LineWidth',2);
```

```
    case {1,2,3}; p1 = plot(time,x_(1:n,:),'LineWidth',2);
end
p2 = plot(time,mu(1:n,:),'LineWidth',2);
p3 = plot(time,y_(1:q,:),'LineWidth',2);
ylabel('Velocity [m/s]')
legend([p1 p2 p3],'x','\mu','y')

subplot(3,1,2)
hold on; ax = gca; ax.ColorOrderIndex = 1;
plot(time,u_(1:l,:),'LineWidth',2)
ylabel('Input Force [N]')

subplot(3,1,3)
semilogy(time,F,'LineWidth',2)
set(gca,'YGrid','on')
ylabel('Free Energy')

xlabel('time [s]')

% Generalised output plots
figure('Name','Generalised output','NumberTitle','off');

for i = 1:pp
    subplot(pp,1,i)
    hold on; ax = gca; ax.ColorOrderIndex = i+2;
    plot(time,y_(i,:),'LineWidth',2)
    str = strcat('y^{(',num2str(i-1),')} [m/s]');
    ylabel(str)
    xlim([0 T])
end
xlabel('time [s]')

%% Function outputs

switch gp
    case 0; states = x;
    case {1,2,3}; states = x_;
end
beliefs = mu;
inputs = u_;
outputs = y_;
```

B.4.3. Dynamic update rules

In the simulation loop of `f_sim` (listing B.9), both the plant and agent require an update on every iteration, which can be achieved with the simple functions `f_plantupdate` and `f_agentupdate` in listings B.10 and B.11 respectively.

Listing B.10: The Matlab function `f_plantupdate`.

```
%% Plant update: Generative process
%{

Iris Hijne
August 2020

INFO_____________________________________________________________________
```

```
This function computes the state derivative and output by means of the
state space equations of the generative process.

INPUTS_____________________________________________________________________
x  [nx1]: current state
u  [lx1]: current input
A  [nxn]: Generative process state-matrix
B  [nxl]: Generative process input-matrix
C  [qxn]: Generative process output-matrix
w  [nx1]: The current process noise
z  [qx1]: The current sensor noise

OUTPUTS____________________________________________________________________
x_dot [nx1]: The state derivative
y_new [qx1]: The new output

%}

%%
function [xdot,y] = f_plantupdate(x,u,A,B,C,w,z)

xdot = A*x + B*u + w;
y = C*x + z;
```

Listing B.11: The Matlab function `f_agentupdate`.

```
%% Agent update: generalised coordinates and control
%{

Iris Hijne
August 2020

INFO_______________________________________________________________________
This function computes udot and then the new control input by adding the
increase of one time-step to the previous control input.

INPUTS_____________________________________________________________________
mu        [npp*1]: current generalised coordinates
y_        [qpp*1]: current generalised system output
A_        [npp*npp]: Generalised state transition matrix
C_        [qpp*npp]: Generalised output matrix
D         [npp*npp]: Differentiator (shift) matrix
G_        [qpp*lpp]: Generalised forward model
xi        [npp*1]: Prior
Pi_w_     [npp*npp]: State precision matrix
Pi_z_     [qpp*qpp]: Output precision matrix
alpha_mu  [1x1]: The generalised state belief learning rate
alpha_u   [1x1]: The control update learning rate

OUTPUTS____________________________________________________________________
mudot [npp*1]: The generalised state belief derivative
udot  [lpp*1]: The control action derivative

%}

%%
function [mudot,udot] = f_agentupdate(mu,y_,A_,C_,D,G_,xi,Pi_w_,Pi_z_,...
```

```
    alpha_mu,alpha_u)

mudot = D*mu-...
    alpha_mu*((D-A_)'*Pi_w_*(D*mu-A_*mu-xi)-C_'*Pi_z_*(y_-C_*mu));

udot = -alpha_u*G_'*Pi_z_*(y_-C_*mu);
```

B.4.4. Free Energy computation

Evaluation of the Free Energy is not necessary for simulation, but can be insightful. Listing B.12 defines the simple function `f_freeenergy` that can evaluate the Free Energy for one or many time-iterations. It can be used on-line, which is unnecessary, or during postprocessing.

Listing B.12: The Matlab function `f_freeenergy`.

```
%% Free Energy evaluation
%{

Iris Hijne
August 2020

INFO_____________________________________________________________________
This function computes the free energy

INPUTS___________________________________________________________________
mu_       [npp*N]: generalised coordinates
y_        [qpp*N]: generalised system output
A_        [npp*npp]: Generalised state transition matrix
C_        [qpp*npp]: Generalised output matrix
D         [npp*npp]: Differentiator (shift) matrix
xi        [npp*1]: Prior
Pi_w_     [npp*npp]: State precision matrix
Pi_z_     [qpp*qpp]: Ouput precision matrix

OUTPUTS__________________________________________________________________
F [1xN]: The Free Energy

%}

%%
function F = f_freeenergy(mu_,y_,A_,C_,D,xi_,Pi_w_,Pi_z_)

F = zeros(1,size(mu_,2));
for i = 1:size(mu_,2)
    eps_mu_ = (D-A_)*mu_(:,i)-xi_;
    eps_y_ = y_(:,i) - C_*mu_(:,i);
    F(i) = 1/2*(eps_mu_'*Pi_w_*eps_mu_ + eps_y_'*Pi_z_*eps_y_);
end
```

Bibliography

[1] Ajith Anil Meera and Martijn Wisse. Free energy principle based state and input observer design for linear systems with colored noise. *American Control Conferene 2020*, 2020.

[2] Manuel Baltieri and Christopher L Buckley. An active inference implementation of phototaxis. In *Artificial Life Conference Proceedings 14*, pages 36–43. MIT Press, 2017.

[3] Matthew James Beal et al. *Variational algorithms for approximate Bayesian inference*. PhD thesis, university of London London, 2003.

[4] Christopher L Buckley, Chang Sub Kim, Simon McGregor, and Anil K Seth. The free energy principle for action and perception: A mathematical review. *Journal of Mathematical Psychology*, 81:55–79, 2017.

[5] M Cover Thomas and A Thomas Joy. Elements of information theory. *New York: Wiley*, 3:37–38, 1991.

[6] D.R. Cox and H.D. Miller. *The theory of stochastic processes*, chapter 7.4, pages 293–295. Methuen & Co, 1965.

[7] David Eberly. Derivative approximation by finite differences, 2020.

[8] Richard Phillips Feynman. *Statistical mechanics: a set of lectures*. WA Benjamin, 1972.

[9] The Wellcome Centre for Human Neuroimaging, UCL Queen Square Institute of Neurology, London, UK. SPM software - statistical parametric mapping. `https://www.fil.ion.ucl.ac.uk/spm/software/`, January 2020.

[10] Karl Friston. Hierarchical models in the brain. *PLoS computational biology*, 4(11), 2008.

[11] Karl Friston. The free-energy principle: a rough guide to the brain? *Trends in cognitive sciences*, 13(7):293–301, 2009.

[12] Karl Friston. The free-energy principle: a unified brain theory? *Nature reviews neuroscience*, 11 (2):127–138, 2010.

[13] Karl Friston. Life as we know it. *Journal of the Royal Society Interface*, 10(86):20130475, 2013.

[14] Karl Friston, Jérémie Mattout, Nelson Trujillo-Barreto, John Ashburner, and Will Penny. Variational free energy and the laplace approximation. *Neuroimage*, 34(1):220–234, 2007.

[15] Karl Friston, Klaas Stephan, Baojuan Li, and Jean Daunizeau. Generalised filtering. *Mathematical Problems in Engineering*, 2010, 2010.

[16] Karl Friston, Thomas FitzGerald, Francesco Rigoli, Philipp Schwartenbeck, and Giovanni Pezzulo. Active inference: a process theory. *Neural computation*, 29(1):1–49, 2017.

[17] Karl J Friston. Variational filtering. *NeuroImage*, 41(3):747–766, 2008.

[18] Karl J Friston, N Trujillo-Barreto, and Jean Daunizeau. DEM: a variational treatment of dynamic systems. *Neuroimage*, 41(3):849–885, 2008.

[19] KJ Friston, M Fortier, and DA Friedman. Of woodlice and men: A bayesian account of cognition, life and consciousness. an interview with karl friston. *ALIUS Bulletin*, 2:17–43, 2018.

[20] Sherin Grimbergen. The state space formulation of active inference. Master's thesis, TU Delft, 2019.

[21] David C Knill and Alexandre Pouget. The bayesian brain: the role of uncertainty in neural coding and computation. *TRENDS in Neurosciences*, 27(12):712–719, 2004.

[22] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[23] Pablo Lanillos and Gordon Cheng. Active inference with function learning for robot body perception. In *Proc. Int. Workshop Continual Unsupervised Sensorimotor Learn.*, pages 1–5, 2018.

[24] Georg Lindgren. Lectures on stationary stochastic processes. *PhD course of Lund's University*, 2006.

[25] David J.C. MacKay. *Information theory, inference and learning algorithms*, chapter 27, pages 341–342. Cambridge university press, 2003.

[26] Guillermo Oliver, Pablo Lanillos, and Gordon Cheng. Active inference body perception and action for humanoid robots. *arXiv preprint arXiv:1906.03022*, 2019.

[27] Corrado Pezzato. Active inference for adaptive and fault tolerant control. Master's thesis, TU Delft, 2019.

[28] Corrado Pezzato, Riccardo Ferrari, and Carlos Hernández Corbato. A novel adaptive controller for robot manipulators based on active inference. *IEEE Robotics and Automation Letters*, 5(2): 2973–2980, 2020.

[29] Léo Pio-Lopez, Ange Nizard, Karl Friston, and Giovanni Pezzulo. Active inference and robot control: a case study. *Journal of The Royal Society Interface*, 13(122):20160616, 2016.

[30] Claude E Shannon. A mathematical theory of communication. *Bell system technical journal*, 27 (3):379–423, 1948.

[31] M. Wisse. Derivation of generalised covariance matrix. July 2019.